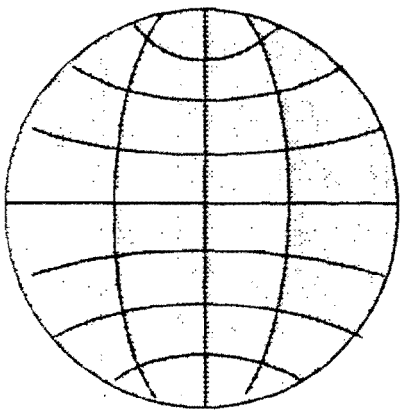
OS-9 User Notes

Volume One



By: *Peter C. Dibble*

Copyright © Peter C. Dibble and The Computer Publishing Center

Peter Dibble

OS-9 User Notes Volume I

April 1985

CONTENTS

Part 1: Columns 1

Introductions 3
Column One 5
Opening Remarks 5
GIMIX-III OS-9 6
A Null Device 6
Documentation for Null Device Descriptor 7
Null Program 7
Column Two 9
OS-9 Level Two Version 1.1 9
Generating a New Bootstrap 9
Building a New System Disk 9
Using Multiple Processes 10
Column Three 11
The FORK Superviser Service Request 11
Communications Via the Parameter Area 11
Assembly Language Procedures for FORKing Processes 11
StrtTask-One 13
Driver One 14
Column Four 15
Basic/BasicO9 15
Interprocess Communication 15
Communication via the Parameter Area 15
Data Modules 16
Locking Data Modules 16
Locker Program 18
Calc Program 21
Driver Program 22
Column Five 25
More About Locking 25
Getting a Good "Mix" 25
An Assembly Language Program Which Sets Printer Options 27
The OS-9 User's Group 29
The Future of this Column 29
POpt Program 30
Column Six 35
New Release of Microware Pascal 36
OS-9 Directories 36
Standard Terminal Support for OS-9 39
Column Seven 41
A Letter 42
Letter from Bengt-Allan Bergvall 42
Parammod 44
Help_B 45
Column Eight 47
The OS-9 User Seminar 47
Shell Commands 48
A Logical Device Driver 48
VCIA Device Driver 51
Column Nine 55
Protection 55
The "Suspend State" 57
Column Ten 59
More About Computers at School 59
Pipes 59
A More Advanced Approach to Pipes 60
Installation 61
Operation and Modification 62
Welcome COCO 62
The Users Group 62
BWord 63
CharCt 64
Grapher 66
StrtTask 67
Rast 70
Column Eleven -- The OS-9 I/O System 73
The Unified Input/Output System 73
Changing OS-9's Device Support 74
Column Twelve -- The CoCo 77
Notes on Compuserve 77
Thank You GIMIX 78
A Handy Shortcut 78
Column Thirteen 81
Big System Hardware 81
Big System Software 81
The Compuserve OS-9 SIG 81

OS-9 on the Color Computer . . . 81
Installation of Beep/Beeper . . . 82
Applications for /Beep . . . 82
The Users Group . . . 83
Sound . . . 83
Beeper . . . 84
TestBeep . . . 86
Column Fourteen . . . 87
More About the CoCo Disk Driver . . . 87
Where Next? . . . 87
More Noise from the CoCo . . . 89
This Month's Driver . . . 89
The Users Group . . . 89
Beeper2 . . . 90
TBeep2 . . . 92
Column Fifteen . . . 95
The OS-9 Seminar . . . 95
OFlex . . . 95
New Manuals . . . 96
C Functions . . . 96
The Butterfly . . . 96
Dynaspell . . . 96
A Nice Experience . . . 97
Tricks for Level Two . . . 97
TstSSig . . . 99
FRexp . . . 100
Modf . . . 100
Column Sixteen . . . 101
Standards . . . 101
Standards that are the User's Responsibility . . . 102
The Users Group . . . 102
Column Seventeen -- The First Step Into OS-9 . . . 103
Format . . . 103
Backup . . . 104
Dir . . . 104
Chx and Chd . . . 105
Dops . . . 105
Column Eighteen . . . 107
My Life . . . 107
Non-standard Hardware . . . 107
Directories as Files . . . 107
DList Program . . . 109
DList2 Program . . . 110
ld Program . . . 111
DFormat Program . . . 112
Column Nineteen . . . 113
More Games with Directories . . . 113
Dr Program . . . 115
DirSqz Program . . . 118

Part 2: Reviews . . . 119

A Review of D-F . . . 121
General System Description . . . 121
Limitations . . . 121
Operation . . . 121
Evaluation . . . 121
Summary . . . 122
Review of OS-9 CIS COBOL . . . 123
Overview . . . 123
Enhancements . . . 123
Limitations . . . 124
Benchmarks . . . 125
Summary . . . 125
COBOL Test Program . . . 126
COBOL Sieve . . . 128
COBOL Benchmark program . . . 129
Review of Software by Clearbrook Software Group . . . 131
DEdit . . . 131
Overview . . . 131
Details . . . 131
Limitations . . . 131
Summary . . . 131
BT9 . . . 131
Overview . . . 131
Details . . . 131
Limitations . . . 132
Summary . . . 132
O-Series Utilities -- DDIR, DDEL, DCOPY and DATTR . . . 132
Overview . . . 132

Details 132
Problems and Limitations 132
Summary 133
Reconsideration 133
A Review of DynaCalc for OS-9 135
Overview 135
Some Details 135
Limitations and Problems 136
Summary 136
Review of Dynamite 137
Overview 137
Some Details 137
Operation 138
Limitations 138
Summary 138
A Review of RMS 141
Overview 141
Some Details 141
Flaws 142
Summary 142
Review of RMA and RLINK 145
Overview 145
Some Details 145
The Separate Assembly Facility 146
Some Internals 146
Limitations 147
Summary 147

Index **149**

FIGURES

1. Execution Sequence for Lockout 25
2. Output of DIR Command 55
3. Sample Startup file 56
4. Password File Entry 57
5. Sample Input for rast program 62
6. Hex dump of a directory 108
7. Dynacalc Terminal Support 135
8. Sample RMS Definition 141
9. RMA Macro 145

TABLES

1. Definitions files routinely included in assemblies 27
2. Dynamite Label Classes 137
3. Dynamite Addressing Modes 137

INTRODUCTIONS

This book is an anthology of all I have written for 68 Micro Journal since I started writing the OS-9 User Notes column in February 1983. Some errors in spelling and grammar have been removed (I imagine others have crept in to take their place.), but I forced myself to leave in mistakes that I made. The most glaring have footnotes pointing out the truth.

This document was prepared using Waterloo Script and a Xerox 9700 Laser Printer. This let me add footnotes, figures, boxes, and an index to the columns. All the footnotes are additions I made while preparing this anthology. The figures and boxes are features I wished for when I was preparing the original column. The contents are from the column, but the presentation is different.

The people at the University of Rochester Computing Center deserve special thanks for their help with this document. This book pushed some of the limits of the system. They were always friendly and helpful.

The index is an attempt to make the helter-skelter arrangement of the columns bearable. Each month I write about what takes my fancy. Sometimes I write about several things. Columns written like that don't combine into a cohesive book very well. I hope the index will guide you to the information you need wherever it hides.

COLUMN ONE

OPENING REMARKS

This is the first of what I hope will be a long series of columns about OS-9 Level Two. I plan on discussing some interesting aspect of programming in each column. I also intend to use this as a soap box for my radical ideas about computing in hopes of stirring up some controversy. My computer was financed largely by teaching computer science. Please bear with me when I have a fit of teaching.

First, by way of introduction, I work as a systems programmer on a variety of machines. I teach computer science courses at a local technical college, and take computer science courses at the local university. I worked my way up to my job as a systems programmer through years of work on payroll, student systems, and other business programming type things - you might say I paid my dues. I got started on microcomputers by building SWTPC's 6809 computer kit. I now own two mongrel computers, one small, running only FLEX, and seldom used, the other large and frequently used. My large computer has a GIMIX DMA disk controller, and a GIMIX 6809 CPU board, two eight inch disk drives, 344K of useful memory, and assorted I/O boards. It can run OS-9 Level Two or FLEX.

I have a collection of strong opinions about computing in general, and microcomputing in particular. The most relevant opinion is that I think the staggering sum I spent to buy OS-9 Level Two together with languages and utilities was money that could not have been better spent though I do wish the prices were lower. I think everyone should get to watch their computer seem to come alive, not just those people who are willing to work two jobs and live on pasta to save enough money. I belong to the school of radicals who believe that Basic is bad for your brain. I like Pascal, but find it a little dull. Assembly language is lots of fun, but slow going. I am looking forward to getting C; it sounds promising.

I think it is practically immoral to force even two people to attempt to use a 6809 at the same time. The fact that it sometimes does a passable job with several users is not a sign that there is plenty of power for several users. The 6809 only runs just so fast. No operating system can make it run faster. Digital Equipment Corp. seems to think its small VAX is probably a good single user machine. I have noticed that the Xerox Star is very slow when it's editing. Both of those computers can run circles around any 6809 machine (and cost far more). Both of them run software written by top quality programmers. The difference is that those computers are expected to make things as easy as possible for their users at any reasonable expense. People who use and program microcomputers don't expect that much out of their machines. Our machines are microcomputers. We expect them to do the same kinds of things other microcomputers do. Our machines are small, but they are part of a new generation. They can do the work of several of last generation's micros. We can use that power to give several users the same poor service, but I would rather see one user well pleased by a computer than several somewhat dissatisfied users.

There is some truly excellent software available for the 6809. I would rate Microware's Pascal as one of the best Pascals I have used on any machine. A lot of features are missing from OS-9 Level Two, but what is there is up to the highest standards and it should be easy to add most of what's missing. From my reading of the manual, BasicO9 seems to be an excellent language (as Basic goes). I own a great deal of software for FLEX and OS-9, but I can't think of any other programs in that league. I am open to suggestions. The challenge is to be at least as good as any similar program on ANY MACHINE. For example, I would love to find an editor that qualifies. My life would be much easier if I could run an editor comparable to EMACS, XEDIT, SPF, or SED on my micro. DYNACALC seems, from its advertisements, to be as good as any of the Visiclones, maybe the best of them. I am holding a grudge against that program because it only supports 3000 cells (under OS-9 Level Two) That's as good as most Visiclones, but I have enough memory for much more than that. The chance to be the first spread sheet program to support almost a megabyte of storage (maybe 30000 cells) in memory on a micro was only a few hundred instructions away, and they didn't do it. I would like to propose some programming challenges to the 6809 community.

I have used spelling checkers that can be asked for a list of suggestions for the spelling of a questionable word. The good ones will provide synonyms on demand too. Doing this at a decent clip, and fitting the dictionary on a floppy disk should be an interesting challenge.

I don't know of any high level language for the 6809 that can use more than 64K even with restrictions. No, I take that back. Microware's Pascal can use a sort of virtual storage scheme to deal with more than 64K of code, but there is no easy way to use more than 64K of data. What I had in mind was a language that could make use of extended addressing. There are lots of useful tricks, playing with the DAT, using software interrupts cleverly, or simply running all procedures (subroutines if you like that term better) as FORKed tasks. Minicomputers used to be limited to a 64K address space. Some of the tricks used to fit big programs into them can probably be adapted to our problems.

A state-of-the-art editor would go a long way toward promoting the 6809. As basic requirements, such an editor should be a screen editor capable of using any available memory. It should include the ability to edit multiple files of arbitrary size without resorting to the "new" or "more" kludge. It should include the best of Wylbur, EMACS, and the other common editors.

If I seem a little shrill about software, it is because I see my beloved 6809 machine being squeezed out by the flood of high quality microcomputers on the market. From my point of view, the best feature of the 6809 is its elegant architecture. It is

so easy to program that it should be pulling ahead of the field with a flood of superb software. I see only a trickle.

GIMIX-III OS-9

Ok, now I'll get off the soap box and down to business. I am thinking of selling both my computers. I positively lust after the new GIMIX CPU board and "level Three" operating system. If there is another microcomputer en the market that does what it does, I haven't heard of it. Large computers such as IBM 370 architecture, and large DECs can cause attempts to write into "protected storage" or execute invalid instructions to fail. Special code is executed whenever a program attempts either of these activities. Usually the program that did it is stopped. Microcomputers don't do that kind of thing. The computer will do something (maybe something ridiculous such as "halt and catch fire") with any data its program counter is pointed at. This can cause a faulty program to go out of control in unpredictable ways. There is no way for the microprocessor to know that it shouldn't write into some part of memory. If you want you can write your name all over the BasicO9 interpreter. The results of that kind of thing are disastrous, particularly if you are sharing the interpreter with someone. You just have to make very sure programs you write never try to execute, or write into anything they shouldn't. Of course that is just good programming.

The new board from GIMIX was designed to work with OS-9. It is alleged to support protected storage and to prevent invalid operations from being presented to the microprocessor. This should prevent any program from interfering with any other program, even, in many cases, itself. For those of you who try to support several users, if you use the new GIMIX hard/software no user should be able to cause the system, or another user's program to fail. Even people like me who don't share time with anyone can gain a lot from this kind of safety net. Sometimes when I am debugging a program everything just comes to a stop and I have to re-boot in order to continue. It is even worse when there is a long pause then the disk starts seeking. I haven't had any data destroyed that way yet, but I worry. This new hardware should give everyone who can afford it a lot of peace of mind. GIMIX has also been able to remove every trace of the operating system from each task's address space. Programs can be run with up to 64K. The board and accompanying software have lots of other features, but the other one that excites me a lot is the memory-to-memory DMA. A lot of time is spent moving data from one address space to another in OS-9 Level Two. This involves several operations for each byte and slows I/O operations and other inter-task communications down quite a lot. The special hardware on this new CPU board can move blocks of data at 2 cycles per byte. At two megahertz that comes to one million bytes per second. I understand that, all things taken together, the new system runs OS-9 substantially faster than what I have now. I want to find out for myself. If you see an advertisement from me in the classified section you will know I broke down and got a new, faster, better 6809 computer.

A NULL DEVICE

One of the nicest features of OS-9 (both levels) is the relative ease with which it can be adapted to new hardware. For example, there is a module included with the operating system called ACIA which is responsible for interfacing the rest of the system with ACIAs (Asynchronous Communications Interface Controllers, or serial ports). There is another module called PIA which does a similar job for parallel ports, and another module which deals with whatever type of disk controller you have - more modules if you have more than one type of disk controller. If you feel the need you can add more Device Drivers (the name of this type of module) any time you like. If you want to write your own driver, it is good to have an example to work from. The source for ACIA and PIA (available from Microware) are both good starting places though I found ACIA more useful.

There is a rather odd sort of device which is available with most operating systems, but not OS-9. I have seen it called DUMMY and NULL. This device makes anything written to it disappear, and returns an endfile if it is read from. It is surprising how often it is nice to have any easy way to throw data away.

The Null Device Driver that I am going to present here is a SCF (Sequential Character File) type device. The requirements for this kind of driver are given in the OS-9 System Programmer's Manual, but in general there are six entry points: Initialize device, read, write, get device status, set device status, and terminate the device. This driver is so simple that of those six, five just clear the carry bit and return. Read is the only operation requiring more than two lines of code. Read is supposed to return with the character read in accumulator A. If an error takes place, the carry bit should be turned on, and the error code placed in accumulator B. We want to return end-of-file, which is an error, and I have found that is a good idea to return null (Chr(0)) as the character read even if it is end-of-file. I return the end-of-file from the driver though it is usually generated by the SCF file manager. If you want to modify the program such that the file manager is the module that generates the end-of-file, load accumulator A with the end-file character which can be found in the path descriptor (pointed to by Y) and return with carry clear.

A Device Driver may be used for several devices provided that they use the same hardware. Each individual device is described by a "Device Descriptor" which includes everything unique to a particular device such as the address of the device. The NL device descriptor is at the bottom of the program. It will be loaded into memory at the same time as the Driver although it will show up as a separate module in the module directory.

DOCUMENTATION FOR NULL DEVICE DESCRIPTOR

If the file Null is loaded and the module NL is linked a new device called /NL will become available for input and output.

```
OS9 Load Null
OS9 Link NL
```

The device NL will accept input in any quantity and simply make it disappear. If a read is directed at it, it will reply <end of file>. Other than eating data without a sign it acts like a perfectly normal SCF type device... a very fast and efficient one!

Example:
```
OS9: asm MyProg o #48k >/nl &
```

Would assemble MyProg in background and make all its (non-error path) output disappear.

Note: Be careful when using /NL for input. Some programs (such as debug) don't respond to <End of File> - these programs will act very oddly if /NL is used as the input device for them.

NULL PROGRAM

```
Microware OS-9 Assembler 2.1   08/05/84 22:40:30                    Page 001
Dummy I/O driver - Definitions

                                  NA    Dummy I/O driver
                                  TTL   Definitions
          *-------------------------------------------------------*
          *  Dummy                                1July82 Peter Dibble *
          *  return end of file to any read                        *
          *  Put any output down the bit bucket.                   *
          *  No error returns                                      *
          *  Public Domain software as of 19Feb83.                 *
          *-------------------------------------------------------*
                                  IFP1              use /D0/DEFS/Defslist
                                  ENDC
     00E1              Type       set   DRIVR+OBJCT
     0082              Revs       set   REENT+2
     0000 87CD002E                MOD   Dummy1,DumNam,Type,Revs,Entry,Memsize
   D 001D                         ORG   V.SCF       leave space for SCFman overhea
   D 001D              Memsize    equ   .
     000D 07                      fcb   READ.+WRITE.+EXEC. driver mode
                                  TTL   Dummy I/O Driver
     000E 446DF9       DumNam     fcs   /Dmy/
     0011 01                      fcb   1           Edition number
     0012              Entry
   W 0012 16000F                  lbra  Init
   W 0015 16000E                  lbra  Read
   W 0018 160009                  lbra  Write
   W 001B 160006                  lbra  GetStat
   W 001E 160003                  lbra  PutStat
   W 0021 160000                  lbra  Term

     0024              Init
     0024              Write
     0024              GetStat
     0024              PutStat
     0024              Term
     0024 5F                      clrb              zero return code
     0025 39                      rts               Do nothing

     0026              Read
     0026 4F                      clra
     0027 53                      comb              set carry flag
     0028 C6D3                    ldb   #E$EOF      return end of file
     002A 39                      rts               return
     002B 848D35                  emod
     002E              Dummy1     equ   *
                                  TTL   Device Descriptor
          *-------------------------------------------------------*
          *  NL device descriptor                                 *
          *-------------------------------------------------------*
     00F1              T pe       set   DEVIC+OBJCT
     0000 87CD001E                mod   DDend,DDNam,Type,Revs,FMNam,DRVNam
     000D 07                      fcb   READ.+WRITE.+EXEC. modes
     000E FF0000                  fcb   $FF,0,0     PORT ADDRESS OF 0
     0011 0100                    fcb   1,DT.SCF    Options
     0013 4ECC         DDNam      fcs   /NL/        device name
     0015 5343C6       FMNam      fcs   /SCF/       File Manager Name
     0018 446DF9       DRVNam     fcs   /Dmy/
     001B BD5979                  emod
     001E              DDend      equ   *
```

OS-9 LEVEL TWO VERSION 1.1

I just installed OS-9 Level Two Version 1.1. Finally it's not "preliminary" any more. Since OS-9 never was very unreliable it is hard to tell whether it is more reliable, but it is very easy to appreciate the new utilities. I spent months writing a PWD program. It prints the name of the current data or execution directory. I hoped someday maybe I could sell that program. Well, Microware beat me to it. The new versions of OS-9 include PWD and PXD, Print Working Directory and Print eXecution Directory. They also added a DELDIR command which deletes a directory with all the files in it, a command called IDENT which displays information about modules in files, a file comparison utility called CMP, and two commands called BINEX and EXBIN which convert a file to and from Motorola standard S-Record format. DCHECK, the program which checks disk structure, now seems to work correctly, and DSAVE, the command which constructs a procedure file to copy groups of files, has been substantially enhanced, but Level Two users will have to continue to live with numeric error messages. A command called PRINTERR, which is supposed to instruct the operating system to use text error messages, wasn't on my distribution disk.

An important new feature in OS-9 is support for XON/XOFF. The ASCII character set includes 32 special codes such as backspace ($08) and escape ($1B) which don't generally represent printable characters, but still have defined meanings. XON and XOFF are among the more useful of these special codes. If, for instance, you have a terminal which usually runs at 19.2KB, but can only accept input at about 200 characters per second when it is in insert mode, it would be nice to be able to constantly adjust the speed at which the computer is transmitting to match the speed at which the terminal can receive. In general you can't do that, but often it is sufficient to be able to tell the computer to "hold it," and "go ahead." If the computer can deal with XOn/XOff protocol, it will "hold it" whenever it receives an XOff, and "go ahead" whenever it receives an XOn. There are quite a few terminals and printers around which run much better when they are attached to a computer which supports XOn/XOff. It is interesting to note that XOff (often called DC3) is entered as <CTRL>S, and XOn (DC1) is <CTRL>Q. In order to use this protocol you've got to find some character other than <CTRL>Q to use as the "quit" character. I wonder whether Frank Hogg is going to be able to adjust DynaStar so it can live without <CTRL>Q and <CTRL>S.

GENERATING A NEW BOOTSTRAP

One of the first things I do with a new version of OS-9 is put together a new bootstrap. There is nothing really wrong with the bootstrap that comes with the system, but I have my own Device Descriptors and Drivers, and even if I didn't need to, I probably would want to re-generate the bootstrap just on the principle of the thing. The modules in the bootstrap are automatically loaded when the system is booted, packed efficiently into memory, and made permanent. It sounds as though, if you have enough memory, it would be a good idea to include in the bootstrap file all the modules you would like permanently in memory. Don't do it! Modules in the boot file are not only permanently in storage, they are also permanently attached to the other programs in the boot. Say you put a P-Code interpreter in the bootstrap - when you link to that module in order to use it, you drag everything else in the bootstrap along with it. If you have a 48K bootstrap you would only be able to run programs which use up to about 12K total. Modules you expect to link to should not be included in the bootstrap. If you include a utility command such as COPY, you may find that you can only use a relatively small amount of memory with COPY. The best way to handle commonly used commands is to merge just less than some small multiple of 4K of them into a utilities file and load it using a LOAD command in the startup file. Since my system allocates memory in blocks of 4K, small programs like COPY and PWD only waste memory if they are loaded by themselves. By collecting groups of programs together you use memory more efficiently, essentially keeping two or more programs in the space normally allocated to one. If your version of OS-9 allocates memory in different sized hunks, the size of the group of programs should be changed to reflect the new constraints. Users of Level One systems don't have to worry about any of this stuff.

The first time I generated a new bootstrap was a little bit intimidating. It is important to realize that, provided you are marginally careful (don't spill chocolate milk on an important disk, etc.), the worst you can do is waste your time. If you don't have a lot of memory the chance to remove unused device descriptors from the bootstrap may be worth the trouble involved in running OS9GEN. If you want to change any modules which are in the bootstrap (addresses in Device Descriptors for instance), the cleanest way to do it is to modify them them with DEBUG, save the modified modules, fix their CRC with VERIFY, and build a new bootstrap with the modified modules. A module must be saved on disk in order to be included in the bootstrap. You should use the SAVE command to create files containing each module you might want in the new bootstrap. Build a file with the names of those files you want to combine into the new bootstrap, and use that list of files as input to OS9GEN. Finally use OCOPY to copy all the other files on your system disk over to the new one.

BUILDING A NEW SYSTEM DISK

I have many files on my system disk that are not part of the OS-9 operating system. An important part of installing a new version of OS-9 which is not mentioned in the manuals is copying all the non-OS-9 files you need onto your new system disk. I have discovered an easy way to do this. I imagine most of you OS-9 users already know this trick, but I wish someone had told me about it a year ago. By running DSAVE on

your old system disk you can create a file containing a copy command for each of the files on your old system disk. If you add a "-x" as one of the first few lines in that file it won't quit if one of the commands fails. The copy commands for files that are already on the new disk will fail, but the procedure will precede to the next command instead of quitting. The result is a disk with all the files you want on it.

USING MULTIPLE PROCESSES

Most of the programming I do is on machines with far more than 64K available to each program. It is easy to get used to having effectively unlimited memory. The 6809 can only use 64K, but with the help of OS-9 Level Two (not Level One) it is possible to use more memory than most people can afford. Over the next few months I expect to spend some time discussing various ways of doing this.

One of the basic facilities in OS-9 (and most other sophisticated operating systems) is called FORK. The effect of FORK is to set a program up and start it running without interfering with the program which FORKed it. Each FORKed program is called a Process or a Task. A process can run for all practical purposes at the same time as the program that FORKed it. Part of setting a process up is finding enough memory for it to run. In OS-9 Level Two each process runs in its own "address space"... that is, no user process shares any memory with any other process except by special arrangement. If you have enough memory, each process can occupy all of its 64K address space except a shred reserved for OS-9.

I have been spending a lot of time writing a program which I call a "smart terminal" program. It started out as a program to allow me to communicate with a variety of computers without having to unhook my terminal from my computer, and fuss with half/full duplex. It just keeps growing. One thing I decided to do was include a way of printing a screen full of data. You can't just stop everything and print the screen; it would take so long to print that the input buffer from the modem would overflow, and at best data would be lost. A solution is to use a FORKed process to print the screen. Once I realized that I could start a process to print the screen, I carried it a step farther and fixed things so I can ask to have lots of screens printed, start a process for each screen, and let them queue up for a chance at the printer while the process doing the smart terminal bit runs cheerfully along. At about 4K per process[1] (the minimum allocation on my Level Two system) I can queue up about 20 screens in the 200K I usually have available. Using the more efficient allocation of storage available under Level One I could probably have queued up about 10 screens in a 56K system. I admit this is a trivial example of the use of extended storage, but the point is that this is a simple example of the kind of thing you can do with extended storage. It is easiest to use multiple processes to get at lots of storage when you can spin off a task that can run in isolation. Communicating between processes is a harder problem than running them in isolation. Several method for communication will be developed in later columns.

[1] The module is reentrant, so only the variable storage needs to be allocated for each process beyond the first.

THE FORK SUPERVISER SERVICE REQUEST

A large number of the exciting things that can be done with OS-9 involve processes. Every program running under OS-9 is a process. Each process runs as if it had the machine to itself (except for speed). When a new process is started, OS-9 loads the Program module for the process if it isn't already in core, creates a Process Descriptor for it, allocates the necessary amount of memory, gives it standard input and output files, and lets the new process go. One of the ongoing tasks of the operating system is to divide processor time between all processes so that the system's resources are used as efficiently as possible, and all the processes run without too many noticeable jerks. You can tell OS-9 to favor a process by giving it a high priority (with the SETPR command), or you can give a process a low priority if you don't much care how quickly it runs.

A new process is created with the OS-9 service request F$Fork. Before issuing this service request you must set up the registers as follows:

X	Address of the name of the module you want to FORK or the file that contains the module.
Y	The size of the parameter area.
U	The beginning address of the parameter area.
A	The Language/Type code. That is, the type of module you want to fork. BasicO9 has to be treated differently from object code.
B	The amount of optional storage to give the new process.

COMMUNICATIONS VIA THE PARAMETER AREA

When the FORK Service request is used to start a new process OS-9 is able to send a block of data to the new process using the parameter area. The new process will be started with X pointing to the start of a copy of the parameter area and D containing the length of the parameter area. In languages other than assembler, the parameter area can be found by noting that the parameter area is the place where the shell places the command line parameters for a program. The shell usually starts programs by FORKing them, so in any language, if you can get to the command line parameters, you can get at parameters passed through Fork in the same way.

By using the parameter area you can pass a lot of information to a new process, but you can't get anything back through the parameter area. Remember that the parameter area gets copied into the new process's address space. It is like a Pascal pass-by-value parameter -- changes don't get back to the invoking process. Still, for many jobs, the one time, one way communication afforded by the parameter area is sufficient.

ASSEMBLY LANGUAGE PROCEDURES FOR FORKING PROCESSES

Neither BasicO9 nor Pascal has all the necessary functions for dealing with forked processes, but they can be reached through assembly language subroutines. I have included two short assembly language subroutines which should help. StrtTask, and WaitTask are meant to be called from BasicO9, though modified versions could be called from Pascal or any other normal language. StrtTask starts execution of a process, and WaitTask waits until a child of the calling process completes before returning to the caller. These aren't examples of elegant coding, but they are good enough to play around with from BasicO9. The BasicO9 programs Driver, and BTest are respectively a driver for the assembly language modules and a stub for testing them.

StrtTask is an interface between a BasicO9 program and the OS-9 Fork service request. Normally, a fork is done with the SHELL statement in BasicO9. By using StrtTask instead of SHELL to start "child" processes, a program can gain better control of the parameters. StrtTask allows full control of the F$fork system service request.

The first parameter which StrtTask expects is the name of the module to be started. It should be passed as a character string with a terminator, such as a space or carriage return, after the last character of the module name. If the module might not be in memory, the name of the file which should be loaded to get the module should be the first parameter instead of just the module's name. The F$Fork system service request description in the OS-9 System Programmer's Manual has more details about this, and all the other parameters for StrtTask.

The second parameter is the process number of the new task. It is a byte field which need not be initialized. StrtTask will place the process number of the newly started process in this byte. This is the only parameter which is returned from StrtTask. The process number is useful if you want to communicate with the new process, or to wait for a particular process to complete.

The third parameter is the language/type byte which describes the module you want to run as a child process. The easiest way to discover the proper value for this byte is by checking the module you want to fork. You can see the language/type byte for a module by loading it and doing a MDIR E command, or by doing a IDENT command on the file the module is in. Remember that this byte is displayed in hex. Object code programs (generated from assembly language) generally have a language/type byte of $11, or decimal 17.

The fourth and fifth parameters are the length of the parameter area to be passed to the forked process, and the parameter area itself. The parameter area can be any type of data you want to pass to the new process. The length of the parameter area is passed as an integer. If you invoke a module which is usually started from the shell, the parameters should be a character string terminated with a carriage return. If you want to invoke a module which runs under Basic09, it is particularly important to include the carriage return at the end of the parameter area (which contains the name of the Basic09 I-code module to run and any parameters for it). Strange things happen if you don't.

The last parameter is the amount of optional storage space you want to give the new process. This is the number usually placed after the "#" on a shell command line. The number can range from zero to 255 (it is a byte field), and may only be in units of pages, not Kbytes.

If the fork service request itself gets a bad return code, it will be returned to the calling program as an error. In general the new process will still be running when StrtTask returns to the calling program, so there is no way to know what the completion code of the new process is (going to be).

Sometimes you may want to start a process going and continue without waiting for the new process to complete, but you may need to wait for it to complete at some point. This is where WaitTask comes in. WaitTask will wait (just sit there) until one of its children (a child of the program that called WaitTask) completes. If there are several children, the first one to complete will let WaitTask return to its caller. If there are no children, WaitTask will return with an error. If a child process terminates before it is waited for, its process descriptor will linger around in memory until a wait is done by the parent process.

WaitTask has two parameters, both of which are set by WaitTask. The first parameter is a byte containing the process number of the process whose completion let WaitTask return. The second parameter is the completion code of that process. If there are several children that might terminate, the process number parameter can be used to cause the calling program to keep calling WaitTask until the necessary process completes.

To use this package of modules (StrtTask, WaitTask, Driver, and BTest):

Assemble a file containing StrtTask and WaitTask

```
asm StrtTask o #24k
```

Save the packed form of BTest

```
BASIC09
in BTest
save
pack
```

load StrtTask and WaitTask

load StrtTask

or if you are still in Basic09

```
$load StrtTask
```

Type in Driver (the basic driver program)

```
run Driver
```

There are a lot of interesting things that can be done with these modules. You can fork any program you want, not just packed Basic09 modules, but the special features of the shell, such as I/O redirection, aren't provided by StrtTask.[2] You don't need to wait for the new process to complete, but if the new process does I/O to standard paths, it can be very hard to tell what is going on on the screen. If you haven't made a mistake that causes several processes to use the terminal for I/O at the same time yet, you should. It is educational.

The thing about new processes that particularly excites me is that under Level Two each new process gets a new address space with up to 64K. The main problem with the modules included with this column is that there is only one-way communication with forked processes. The parameter area goes from the parent to the child, but the child only sends a completion code back to the parent. There are easier ways to communicate. We'll get to them later.

---

[2] A version of StrtTask with support for pipes appears in Column Ten.

STRTTASK-ONE

```
 ttl Start a subtask (called from Basic09)
 nam StrtTask
*-------------------------------------------------------*
*  StrtTask is a subroutine for Basic09.                *
*  Start a named module as a subtask.                   *
*    Let the new task run as nchronously.               *
*    return the new tasks process number, and the       *
*    condition code from the Fork.                      *
*   Calling sequence:                                   *
*   run StrtTask (Name, Process_Num, Lang_Type,         *
*                 Param_L, Param, Opt_size)             *
*   Name is any length, but has a valid terminator      *
*    (high bit set on last byte, or delimiter after it)*
*                                                       *
*   Process_Num byte field, process number of new task.*
*   Lang_Type byte field, language/type byte for        *
*    forked module.                                     *
*   Param_L, integer field, length of parameter area.   *
*   Param field of any type, parameter area to be       *
*    passed to forked process.                          *
*   Opt_Size byte field, optional data area size in     *
*    pages.                                             *
*   Process_Num, and Return_Code are altered by         *
*    StrtTask, no other parameters are.                 *
*-------------------------------------------------------*
 IFP1
 use /H0/DEFS/defslist
 ENDC
Type set SBRTN+OBJCT
Revs set REENT+1
 mod TLen,StrtTask,Type,Revs,SEntry,0
StrtTask fcs /StrtTask/
 fcb 1 version
SEntry
 ldd 2,S get param count
 cmpd #6 are there 6 params?
 bne BadExit no; leave now.
 ldx 4,S address of module name
 ldy [16,S] length of parameters
 lda [12,S] type of module to invoke
 ldb [24,S] optional data area size
 ldu 20,S  pointer to parameters
 OS9 F$Fork start the new process
 bcs BadExit2
 sta [8,S] save new process number
 clrb clear carry
 rts return
BadExit
 coma set carry
BadExit2
 rts return
 EMOD
TLen equ *
 ttl Wait for a (child) process to complete
 nam WaitTask
```

```
*-------------------------------------------------*
*   WaitTask is a subroutine for Basic09          *
*   Wait for the a child process to complete.     *
*   Return the process ID of the process that completed*
*   in parameter one.                             *
*   Return the competion code of the process      *
*   in parameter two.                             *
*   This subroutine will wait using no CPU time until *
*   a child process completes.                    *
*   If a child completed just before WaitTask was *
*   called, it will return almost immediatly.     *
*   If there are no children, an error will be returned*
*   with a process number of 0.                   *
*   Calling sequence:                             *
*    RUN WaitTask (Process_No, Comp_Code)         *
*   both process_no and Comp_Code are BYTE variables. *
*-------------------------------------------------*
Type set SBRTN+OBJCT
Revs set REENT+1
 mod WLen,WaitTask,Type,Revs,WEntry,0
WaitTask fcs /WaitTask/
 fcb 1 edition
WEntry
 clr [4,S] zero the process ID
 ldd 2,S param count
 cmpd #2 if not exactly 2 params then
 bne WBExit2 the caller is making a bad mistake
 OS9 F$Wait wait for a child
 bcs WBExit
 sta [4,S] return the process ID
 stb [8,S] return the completion code
 rts return
WBExit2
 coma set carry
WBExit
 rts return
 EMOD
WLen equ *
 end
```

DRIVER ONE

```
PROCEDURE Driver
DIM process_No,Comp_Code,Opt_Size,Lang_Type:BYTE
DIM Parm_L:INTEGER
DIM name:STRING
DIM Parms:STRING[20]
(* ------------------------------------------------*)
(* Set up to call StrtTask which will fork the named *)
(* module, passing it the parameter string in Parms. *)
(* ------------------------------------------------*)
name="Basic09 "
process_No=0
Opt_Size=0
Lang_Type=$11 \(* attributes of forked module (object code, program)
Parms="BTest"+CHR$(13) \(* The parms must end with <CR> for Basic09
Parm_L=LEN(Parms) \(* The length of the parameters must be correct
(*
(* Call assembler subroutines to Fork and wait for the started
(* process
*
RUN StrtTask(name,process_No,Lang_Type,Parm_L,Parms,Opt_Size)
RUN WaitTask(process_No,Comp_Code)
(*
(* Acknowledge that everything is done
(*
PRINT "Forked task complete"
PRINT "Completion code for process "; process_No; " was "; Comp_Code
```

COLUMN FOUR

BASIC/BASIC09

A month ago I installed BasicO9 on my machine. I have been proud of not having a Basic on my computer, but OF (An OS-9/Flex copy program) requires BasicO9, so I swallowed my pride and installed Basic. I have spent too many hours breaking students of the bad habits they learned in elementary computing courses taught using Basic to have any affection at all for that language, but I think I could learn to love BasicO9. It is able to masquerade as Basic, but it feels just like a modern structured programming language to me. I am sure that there were valid marketing reasons for including "basic" in the name of BasicO9, but I wish they had named it Advanced Programming Language or something; I would feel much more comfortable learning to love the language if it had a different name.

INTERPROCESS COMMUNICATION

Last Column I promised to continue wrestling with the problem of communication between processes ... Writing about processes without using technical terms is getting to be too much for me. I am going to give loose definitions of some of the important terms here.

Process or Task A module (Program, subroutine, or whatever) which the operating system views as an independent piece of work. A program is usually a process though sometimes a program is divided into several processes.

Concurrent processes Strictly speaking concurrent processes must actually run at the same time. This requires a separate processor for each process. The term is sometimes loosely applied to processes (like OS-9's) that are actually using one processor in turns, but seem to be running at the same time.

Dispatch Give a process access to the processor. The operating system will dispatch each active process in turn. Only one process can be running at any time, so the operating system must have a way of interrupting a process as well as dispatching it.

Schedule Closely related to dispatch. If the operating system shows any intelligence at all about which process to dispatch next, it can be said to schedule them.

Spawn Create a new process. This is a more general term than FORK because not all operating systems call the operation which spawns a new process FORK.

Parent/Child The process that spawns a new process is called the Parent (used to be father) of the new process. The new process is said to be the child (used to be son) of the process which spawned it. The family tree analogy can be taken as far as you like; processes can have siblings, ancestors, descendants...

Asynchronous Not depending on the same clock.

Don't take these definitions as gospel. They are superficial -- barely enough to be useful in the context of this column.

COMMUNICATION VIA THE PARAMETER AREA

Passing a parameter area to a FORKED process is simple, but of limited usefulness. The limitations associated with communication with processes via the parameter area are that the communication is generally one way, and that, since a copy of the parameter area is made for the new process, large parameter areas will use a lot of memory, and increase the length of time the FORK operation takes. Under OS-9 Level One, all processes share one 64K address space along with all the assorted system overhead (OS-9 itself, memory mapped I/O, etc). Spawning a new process with a 20K parameter area will cost 40K just for the parameter area (20K for the original and 20K for the new process's copy). That kind of thing can chew up a lot of memory in short order. With Level Two, the memory problem isn't so important, but, unless you have the Gimix III version of OS-9, it is time consuming to copy a large parameter area into a new address space.

Some of the characteristics of the parameter area make it possible for new families of bugs to creep into programs that use them for inter-process communications. Under OS-9 Level Two, each new process gets its own address space. There is no sign of any other process in that address space except a copy of the parameter area passed from the parent process. If the parameter area includes any addresses, they will be pointing to places that were significant in the parent's address space. In the new process's address space those addresses may be empty or contain something unexpected. The tricky thing about this is that, under Level One, addresses in the parameter area are meaningful. Since there is only one address space, the addresses just reach out into the parent's memory and grab, or change, the data the parent pointed them at. Being able to read and change data in the parent process's memory is a mixed blessing.

Let's say you want to print the contents of an array without stopping to wait for the printer. A very good way to do this is to spawn a task to do it. If you pass the array to the new task as a parame-

ter, everything will be fine except that, if the array is large, you may run out of memory. If you conserve memory by passing only the address of the array, everything will still be fine (under Level One) provided that neither process changes the array while the child is running. If the child changes the array, it is very likely to be a surprise for the parent. If the parent changes the array (e.g., by starting to work on new data) the child will see the changes, and print an array that is part the old one and part the new one.

It would nOt be too hard to track down the reason for that kind of garbled printing, but there is an especially virulent form of that bug which not only is hard to find once you set out to look for it, but also sometimes doesn't show up under most forms of testing and looks suspiciously like a hardware glitch. The operating system lets each process run for a fraction of a second, then interrupts it and dispatches another process. If you read some of another process's data, then change it and put it back (something like A = A + 1, which reads A, adds 1 to it, and stores the result in A), you can't be sure that the other process hasn't changed the data between the time you read it and the time you wrote it unless you have masked interrupts for the duration of the operation. If some process changed the value of A in the middle of the add, the new value of A will be wiped out when the result of the addition is put into A. Every process looks entirely innocent when viewed alone, but, taken together, they are chaos. If you change a program with this kind of error, even to add diagnostics, the problem may seem to disappear. The timing has to be very precise for this kind of error to show up, and (Murphy's Law being what it is) the timing is never what you want it to be. Finding and fixing this kind of bug is the kind of thing that makes a programmer want to join a commune and raise corn.

OS-9 Level Two prevents this kind of trouble with the parameter area by making addresses in the parameter area unusable. Some programmers working in OS-9 Level One without a crystal ball to predict the nature of Level Two passed address to other processes. Their programs (I believe DYNASTAR/DYNAFORM is an example) have restrictions when they are used under OS-9 Level Two because under Level Two those addresses are not meaningful.

If addresses are included in the parameter area, and you are using Level One, a process can send data to its perent by changing the parent's variables. If you prudently don't use that questionable trick, this type of communication is like heredity: strictly from parent to child.

DATA MODULES

The parameter area is certainly the simplest path for inter-process communication, but there are are several other methods. The most powerful tool for interprocess communication is the "data module." The data module is a rather mysterious module type intended to be used to store collections of constant data. The usefulness of data modules stems from the way OS-9's LINK system service request works.

The LINK request returns the address of the module you link to. Level One simply returns the address, but Level Two must put the module in question into the address space of the process that does the LINK in order to be able to provide a meaningful address. If the module is marked "reentrant," the system memory map will be adjusted so the memory containing the module being linked to will appear in the address space of each process which is LINKed to it. This is a way to make a block of memory accessible to several processes. By making a module reentrant you assure the operating system that several processes can use the module without interfering with one another. Usually that means nobody changes the module. In the case of a shared data module it is sometimes a good idea to lie to OS-9. If you let a single process change a reentrant data module while other processes only read what's there, there is not much chance of getting into trouble. Data modules can be written into by many processes, but this requires careful management. The problems which can plague Level One users playing with two way communications through the parameter area all apply to shared data modules which are written into by more than one process.

A rather annoying problem with data modules is that they must be loaded from disk like any other module. It is possible to build a module in memory, but the system service request which forces OS-9 to include the module in its directory of modules in memory is a supervisor state request. It is possible to circumvent that restriction, but the method is too involved to tackle this month.

LOCKING DATA MODULES

It is practical to have a data module with two or more "writers" because there are ways to "lock" a data module. A lock is a system for checking that a resource is free, then, if it is free, marking it "in use." Every program that uses a shared resource must check and respect the lock in order for it to be effective, but there is no way to enforce the locking in such a way that no program can get at the shared module without going through the locking protocol (GIMIX III might provide a way to do this). The easiest way to lock a module (or anything else) is to write a pair of operating system services to lock and unlock any specified resource. These services are usually called ENQ/DEQ after the sensible English words enqueue and dequeue, or P/V after two Dutch words. Dijkstra is responsible for the P/V terminology; IBM may have thought up ENQ/DEQ. Perhaps I'll write the OS-9 function handlers for P and V someday, but until those services are available, modules can be locked quite effectively in any assembly language program.

There are several instructions in the 6809 instruction set which can read and write memory all in one instruction. Altering a byte by reading and writing it

in one instruction prevents any other process from accessing the byte in the middle of the alteration. The machine instructions that read and write in one instruction are: shift instructions, rotate instructions, increment, decrement, complement, and negate. The instructions which are usually used for "locking" a module are increment and decrement. The basic idea is that you set aside a locking byte in the data module with an initial value of -1. To lock the module, increment the byte, and, if increment returns with the zero flag set, continue; the module is locked. If the zero flag is not set some other process has the module locked, so decrement the locking byte, and sleep for a while... then try again. See the assembly language modules Lock, and UnLock, for examples of this procedure.

The LINK service request is only able to find modules that are already in memory. If the module is not in memory it must be loaded from disk using the LOAD service request. This problem could be dealt with by writing two assembly language subroutines, one to do LINKs, the other to do LOADs. This offers the most flexibility, but requires the calling program to know more about OS-9 than I like. The assembly language program that accompanies this column attempts to load a module from the execution directory if it can't be found in memory. The problem with this approach is that the file which contains the data module must have the same name as the module.

The data module itself is created by the assembler. The main difference between a data module and a program module is that a data module has no permanent storage size in the module header, and no executable code . I use the execution offset field in the module header to point to the beginning of the shareable data. By convention, I use the first byte in the shareable data as a locking byte. For OS-9 Level One users, it is good to keep the module to a multiple of 256 bytes. Under Level Two, a module loaded by itself will use a multiple of the page size (usually 4096 or 2048 bytes), but a module loaded from a file containing several modules will share a page with other modules from that file if it can.

Together, the assembly language modules SLink, SUnlink, Lock, and Unlock, provide the tools necessary for a BasicO9 program to use shareable data modules. Before a data module can be used, it must be linked to; SLink returns the address at which OS-9 placed the data module. This address will be usable until the module is UnLinked. Before any data in the module is used or changed, the module should be locked by calling Lock. Lock will not return control to the calling program until it has control of the data module. It would be possible to rewrite lock so it would return with an error code if some other process had control of the data module, allowing the calling program to choose to do something other than wait if the module is not available. As soon as possible after locking the data module, it should be unlocked to release other processes waiting for the data module. Before stopping, a program that links a module should unlink it. OS-9 maintains a counter of how many times a module has been linked to, and deletes the module from memory when its link count goes to zero.

I have included two trivial BasicO9 programs to demonstrate module locking. Calc only calculates the sum of the squares of a list of numbers, but it could be the mainstay of a mail system, a matrix manipulation routine, or a print spooler (to name a few possibilities). Driver2 is a program who's greatest virtue is that it calls Calc. There are two forms of locking going on in the Driver-DataMod-Calc system: the first byte of data in DataMod is used by Lock. The second byte of data in DataMod is used for communication between Driver2 and Calc. Each process waits for this byte to take on a value set by the other process before it accesses the rest of DataMod. This is a very simple protocol which can only be used in trivial cases such as signaling between two modules. In this case, the main lock is used to prevent several modules from trying to change the communications byte at the same time. Once a process gets the lock, no other process can get it until the process holding the lock releases it. The process which has the lock can use the communications byte, and the rest of the data module, to call for the services of Calc in an organized fashion.

I use a module from last month's column called StrtTask in this set of programs. If you are especially interested in memory efficiency, merge the file containing the StrtTask module with the file containing this month's assembly language modules. Calc must be packed in order to work (at any rate, I can't puzzle out any reasonable way to use it in source form). To make the contraption go, load the file containing SLink, SUnLink, Lock, and UnLock. If StrtTask is in a separate file you might want to load that too; then start up BasicO9 and run Driver2. Driver2 will pause for a while, starting up Calc, then ask for a number five times. Give it small numbers -- they have to fit into byte variables. When all five numbers are entered, Calc will calculate the sum of their squares which will be displayed by Driver2. If you want to try it again, reply Y to the next prompt. The last thing Driver2 will do before ending is ask whether you want to shut down Calc. You do. In a system with several processes using Calc you would want to leave it running, but, with only one process using Calc, it will just be a nuisance if it is not cleaned up when its one user terminates.

```
 NAM SLink
*------------------------------------------------------*
*   SLink                                              *
*   Attempt to link to a module.                       *
*   If it isn't found attempt to load it.              *
*   Return the address of the module header, and the   *
*   entry address.                                     *
*   Errors:                                            *
*         1   Wrong number of arguments in parameter   *
*             list.                                    *
*     other   Return code from F$Link, or F$Load.      *
*                                                      *
*   Calling sequence (from Basic09) is:                *
*      RUN Link (Module_Name, Module_Type,             *
*                Header_Addr, Entry_Addr)              *
*    Module_Name is a character string containing the  *
*          name of the module which should be linked   *
*          to. It should be terminated with a <CR>.    *
*    Module_Type is a byte containing the language/    *
*          type of the module. A data module would be  *
*          $40.                                        *
*    Header_Addr is the address of the module header of*
*          the linked module. It is returned from      *
*          Link. Integer field.                        *
*    Entry_Addr is an integer field which is used to   *
*          return the address of the entry point of    *
*          linked module.                              *
*------------------------------------------------------*
 ifp1
 use /h0/defs/defslist
 endc
 TTL Subroutine callable from Basic09 to do Link SSR
 MOD LinkEnd,LinkNam,SBRTN+OBJCT,REENT+1,LinkEnt,LnkMemS
LinkNam fcs /SLink/
 fcb 1 version
LnkMemS equ .
LinkEnt
 ldd 2,S get parameter count
 cmpd #4 must be four
 bne LinkErr1
 ldd 14,S get length of entry address field
 cmpd #2
 bne LinkErr2
 ldd 18,S get length of header address field
 cmpd #2
 bne LinkErr2
 ldx 4,S Module name's address
 lda [8,S] Type/Language
 pshs U
 OS9 F$Link
 bcc LinkRtn Carry clear; clean return
 puls U
 cmpb #E$NEMod Non-existent module?
 bra LinkErr2 no; bad error was bne
 ldx 4,S Module name's address
 lda [8,S] Type/Language
 pshs U
 OS9 F$Load
 bcc LinkRtn
 puls U
LinkErr2
 coma
 rts return with error code in B and carry set
LinkErr1
 ldb #$FE error code of 1
 comb set carry
 rts
LinkRtn
 stu [14,S] Header address
 sty [18,S] data address
 puls U
 clrb clear carry
 rts return
 EMOD
```

```
LinkEnd equ *
 NAM SUnLink
*-------------------------------------------------------*
*   SUnlink          Unlink a Linked Module.            *
*   Calling sequence (from Basic09).                    *
*     RUN Unlink (Header_Addr)                          *
*   Errors:                                             *
*         1   Wrong number of arguments in parameter    *
*             list.                                     *
*     other   Error code from F$Unlink.                 *
*                                                       *
*   Header_Addr is the integer address of the header    *
*         returned from the link request for the        *
*         module you want to unlink.                    *
*-------------------------------------------------------*
 TTL Subroutine callable from Basic09 to do Unlink SSR
 MOD UnlkEnd,UnlkNam,SBRTN+OBJCT,REENT+1,UnlkEnt,ULkMemS
UnlkNam fcs /SUnLink/
 fcb 1 version
ULkMemS equ .
UnlkEnt
 ldd 2,S get parameter count
 cmpd #1 must be one
 bne ULnkErr1 not one; error
 pshs U
 ldu [6,S] get module header's address
 OS9 F$Unlink unlink the module
 puls U recover U
*  return code and carry set by F$Unlink
 rts return
ULnkErr1
 ldb #$FE
 comb
 rts
 EMOD
UnlkEnd equ *


 NAM Lock
*-------------------------------------------------------*
*  lock           Lock protocol                         *
*  Wait for a "lock" byte to indicate unlocked, then    *
*  lock the byte.                                       *
*  Calling sequence:                                    *
*    RUN Unlock (Lock_Addr)                             *
*  Errors:                                              *
*    1   Wrong nu ber of arguments in para eter list.   *
*                                                       *
*  Lock_Addr is the integer address of the byte used    *
*    for the locking protocol.                          *
*-------------------------------------------------------*
 TTL Subroutine callable from Basic09 to perform "lock" protocol
 MOD LockEnd,LockNam,SBRTN+OBJCT,REENT+1,LockEnt,LokMemS
LockNam fcs /Lock/
 fcb 1 version
LokMemS equ .
LockEnt
 ldd 2,S get parameter count
 cmpd #1 must be one
 bne LockErr1 not one; error exit
LockLoop
 ldx [4,S] get address of lock byte
 inc ,X test and set it
 beq Locked
 dec ,X can't get it
 ldx #2 interval for brief sleep (tunable)
 OS9 F$Sleep
 bra LockLoop
Locked
 clrb turn off carry
 rts return
LockErr1
 ldb #$FE
 comb
 rts return
 EMOD
LockEnd equ *
 NAM UnLock
*-------------------------------------------------*
*    UnLock         Perform the Unlock protocol   *
*    Restore the "lock" byte to the unlocked      *
*    state.                                       *
```

```
*                                                    *
*      RUN UnLock (Lock_Addr)                        *
*    Lock_Addr is the integer address of the byte    *
*    used for the locking protocol.                  *
*----------------------------------------------------*
 TTL Subroutine callable from Basic09 to perform UnLock
 MOD ULokEnd,ULokNam,SBRTN+OBJCT,REENT+1,ULokEnt,ULokMemS
ULokNam fcs /UnLock/
 fcb 1 version
ULokMemS equ .
ULokEnt
 ldd 2,S get parameter count
 cmpd #1 must be one
 bne ULokErr1 not one; error exit
 ldx [4,S] get address of lock byte
 dec ,X release the lock
 clrb set carry bit off
 rts return
ULokErr1
 ldb #$FE error code of 1
 comb set carry
 rts
 EMOD
ULokEnd equ *



 NAM DataMod
 TTL A Lockable data module
*----------------------------------------------------*
*    This a generic data module.                     *
*    It contains a locking byte and up to 232        *
*    bytes of unspecified data.                      *
*----------------------------------------------------*
 MOD ModEnd,ModNam,DATA,REENT+1,LockByte,0
ModNam fcs /DataMod/
 fcb 1 edition
LockByte fcb -1
UnSpec fcc /12345678901234567890123456789012345678901234567890/ 40
 fcc /12345678901234567890123456789012345678901234567890/ 80
 fcc /12345678901234567890123456789012345678901234567890/ 120
 fcc /12345678901234567890123456789012345678901234567890/ 160
 fcc /12345678901234567890123456789012345678901234567890/ 200
 fcc /12345678901234567890123456789012/ 232
 EMOD
ModEnd equ *
```

CALC PROGRAM

```
PROCEDURE Calc
(*
(* Calculate the sum of the squares of the numbers
(* stored in DataMod.
(* A process signals that it wants service by storing a
(* hex 01 in the byte one off the start of data in DataMod.
(* When Calc sees a 1 in that byte, it calculates the sum of
(* the squares and puts it at 7 and 8 off the start of data in
(* DataMod, then sets the status byte (1 off the start) to hex 00
(* indicating that calculation is done.
(*
DIM Module_Name:STRING
DIM Module_Type:BYTE
DIM Header_Addr,Data_Addr:INTEGER
DIM Status_Addr,Array_Addr,Return_Addr:INTEGER
DIM sum,i:INTEGER
(*
(* Setup
(*
Module_Type=$40
Module_Name="DataMod"+CHR$(13)
RUN SLink(Module_Name,Module_Type,Header_Addr,Data_Addr)
Status_Addr=Data_Addr+1
Array_Addr=Data_Addr+2
Return_Addr=Data_Addr+7
POKE Status_Addr,0 \(* set idle (ready for work)
(*
(* Wait for the status byte in DataMod to
(* indicate that an operation is waiting to be done.
(*
WHILE PEEK(Status_Addr)<>1 DO
SHELL "SLEEP 2"
ENDWHILE
WHILE PEEK(Status_Addr)=1 DO
sum=0
FOR i=0 TO 4
sum=sum+PEEK(Array_Addr+i)*PEEK(Array_Addr+i)
NEXT i
(* The calculation is done. Save the result
POKE Return_Addr,sum/256
POKE Return_Addr+1,MOD(sum,256)
(* and indicate that the results are ready
POKE Status_Addr,0
WHILE PEEK(Status_Addr)=0 DO
SHELL "SLEEP 2"
ENDWHILE
ENDWHILE
POKE Status_Addr,0 \(* we're dead
RUN SUnLink(Header_Addr)
BYE
```

DRIVER PROGRAM

```
PROCEDURE Driver2
(*
(* Driver for "Locker"
(* Demonstates simple Module lock/unlock
(*
(* Operation:
(* Link to DataMod
(* Fork Calc (a simple process for demonstration purposes)
(* Wait for the second Data byte in datamod to become $00
(* indicating that Calc is running.
(* Start of Loop
(* Lock DataMod
(* Store data into bytes 2, 3, 4, 5, and 6 off the start of
(* data in DataMod
(* Change the byte at 1 off the start of data in DataMod to $01
(* indicating that there is data in the module to be operated on
(* Wait for the second data byte to change $00
(* Get the result of the calulation (an integer) at 7 and 8
(* off the start of data in DataMod.
(* UnLock DataMod
(* Loop until end is called for
(*
(* Lock DataMod
(* Change the the first data byte to $02 (which tells calc to stop)
(* Wait for the first data byte in DataMod to change to a $00
(* UnLock DataMod
(* UnLink DataMod
(* All done


DIM Header_Addr,i:INTEGER
DIM Process_Num,Module_Type:BYTE
DIM Param_Len,Data_Addr:INTEGER
DIM Opt_Size,op:BYTE
DIM Num:INTEGER
DIM Params:STRING
DIM Module_Name:STRING
DIM NY:STRING


Module_Type=$40
Header_Addr=0
Data_Addr=0
Module_Name="DataMo"+CHR$($80+ASC("d"))
RUN SLink(Module_Name,Module_Type,Header_Addr,Data_Addr)
(* Set up for FORK operation
Module_Type=$21 \(* Subroutine/Object code
Params="Calc"+CHR$(13)
Param_Len=LEN(Params)
Opt_Size=10
Module_Name="Basic09"+CHR$(13)
RUN StrtTask(Module_Name,Process_Num,Module_Type,Param_Len,Params,Opt_Size)
(* Calc is starting now
(*
(* Wait for the first data byte in DataMod to become zero
(* the first data byte is located at the address in Data_Address
(*
GOSUB 100 \(* wait for calc to send ready


(*
(* Calc is running. Send it data
(*
REPEAT
RUN Lock(Data_Addr)
(* load DataMod with data
(*
FOR i=2 TO 6
INPUT "Enter a number (1..255), or 0 to stop):",Num
POKE Data_Addr+i,Num
NEXT i
POKE Data_Addr+1,1 \(* mark the module "ready for operation"
GOSUB 100 \(* wait for calc to indicate ready
PRINT "Sum of squares is "; PEEK(Data_Addr+7)*256+PEEK(Data_Addr+8)
RUN UnLock(Data_Addr)
INPUT "More calculations? (Y,N):",NY
UNTIL NY="N" OR NY="n"
INPUT "Shut Down Calc Module? (Y,N):",NY
IF NY="Y" OR NY="y" THEN
RUN Lock(Data_Addr)
```

```
POKE Data_Addr+1,2 \(* command for stop
WHILE PEEK(Data_Addr+1)<>0 DO
SHELL "SLEEP 2"-
ENDWHILE
RUN UnLock(Data_Addr)
RUN SUnLink(Header_Addr)
ENDIF
END


100 (* Wait for the status byte in DataMod to
(* indicate ready
WHILE PEEK(Data_Addr+1)<>0 DO
SHELL "SLEEP 2"-
ENDWHILE
RETURN
```

MORE ABOUT LOCKING

Last month I discussed shared data modules, and demonstrated a locking method which could be used to permit only one process at a time to access a data module, or, for that matter, any shareable resource.

The locking protocol I demonstrated last month has two serious problems. One is only a problem for those who, like most of us, can only run more than one process by sharing a processor between several processes. The other problem limits the usefulness of concurrent processes. Both problems have solutions.

The locking algorithm I demonstrated last month used a technique called "busy waiting." This is usually the easiest way to make a process wait until something happens, but it wastes processor cycles. I tried to reduce the amount of time wasted in the locking module as much as possible by putting a "sleep" in its wait loop, but the solution I gave was, nevertheless, inefficient. If waiting for the lock uses processor time, even very slowly, all you have to do is line up enough processes waiting for the lock and you can slow the computer down to a crawl. You might think that you could always make the waiting processes as cheap to run as necessary by putting a longer sleep into the wait loop, but if the sleep is made very long, there may be a significant time during which the lock is off and the all the processes which want it are sleeping. If the goal is performance, (using performance in the same sense as "high performance car") it is not good to leave a scarce resource like the lock unused for any length of time. The goal is to design an algorithm which allows waiting processes to be completely idle until the lock is available, then awakens one process and gives it the lock.

If each process is running on its own processor, the processor running a waiting process has nothing better to do than zip around the wait loop. Some people think busy waiting is bad even then. I tend toward the opposite extreme. The problems with busy waiting are obvious, the alternatives have trickier problems. The issues involved in choosing a busy waiting algorithm over a more sophisticated one are much like those involved in choosing a bubble sort over one of the flashier sorting algorithms, that is, for a small problem the simple algorithm will do fine.

The other problem with the locking algorithm I gave is that it permits "lockout," ie. a process can wait forever without ever getting the lock even when no other process holds the lock forever. If there will seldom be a process waiting for the lock, lockout isn't a big problem, but for locks that usually have a process or more waiting for them lockout is an important consideration.

It is tricky to detect lockout in an algorithm, but here are the basic rules for finding it: Imagine that you are controlling the computer's dispatcher (deciding which process runs and for how long). It is your job to prevent a certain process from ever getting the lock. You may run any mix of programs you like any way you like except that the process you are trying to prevent from getting the lock must be allowed to run every now and then. If it is possible to prevent that process from ever getting the lock, there is lockout. The sequence of events that demonstrates that lockout is possible for the algorithm I gave last month is: Two processes are running, A and B. Both processes are simple programs which just get the lock then release it again and again. Either process could be locked out, but the "execution sequence" in figure Figure 1 only demonstrates that process B can be locked out.

```
Start
A: lock
B: try to lock
A: unlock
A: lock
B: try to lock
A: lock
etc.
```

Figure 1: Execution Sequence for Lockout

You see that by allowing process A to run long enough so that can get the lock again each time it releases it I can shut process B out completely. This may seem unfair, but it shows that the algorithm permits lockout. Murphy's law certainly dictates that if it is possible to prevent a process from ever getting the lock (and you want it to get the lock), the improbable execution sequence which leads to lockout will happen at the worst possible moment. This is one of the kinds of problem that cause strange behavior in complicated systems.

There are many ways to do locking that don't use busy waiting or have deadlock. I am not going to discuss these tricks this month, but I will leave you with two hints. The OS-9 SEND service request offers an alternative to busy waiting. Locking can be done without deadlock by using any of several algorithms including one called the Doorman Algorithm.

GETTING A GOOD "MIX"

The standard use for multiple processes is to make maximum use of a processor when the work to be done involves a lot of waiting for outside events, such as terminal input. A process could spend most of its time waiting for input from a terminal, and delegate any major work to child processes. This way the program would almost always be ready to accept input from the terminal, even when some previous piece of work was still in progress. Using a special process to print a screen is a particularly apt use of this principle. There is really no reason why someone should have to wait for a print request to complete before continuing,and there is usually no need for the

process that is doing the printing to communicate with its parent process. The process that is driving the printer spends most of its time waiting for the printer, and the process that is responsible for the screen is, very likely, spending most of its time waiting for input from the terminal. These tasks can be in progress at the same time with almost no effect on one another. When one process is waiting for something, the other process can run without interference. With tasks like printing and screen handling, the computer will spend most of its time with both processes waiting.

Some programs run well together, other programs interfere badly with each other. Finding good sets of programs to run at the same time, and adjusting their priorities so they all will run as fast as possible is called finding a good "mix." Tuning hardware and software so a single program can run as fast as possible is a complicated job, but choosing groups of programs which will run well together, and tuning the system so the groups will run as fast as possible is more of a black art. I like to keep my personal computer rather lightly loaded (no more than two or three processes active at a time), but it is good question just how much time a computer should spend waiting. If you give the machine so much work to do that it never has to wait, each process will run slowly. A computer that has no resources in reserve is said to be saturated.

Consider the case of a program which is reading from the terminal. Usually, in a saturated computer, there are several processes waiting for processor time at any moment. The process waiting for the input character will have to wait for at least one process, maybe several, to have their turn before it will get a chance to run. If each process gets a turn one tenth of second long, and there are an average of two processes waiting to run, then a process will take about two tenths of a second to respond to a simple keystroke. That comes to 5 characters per second, or 300 characters per minute. For perspective, my terminal repeats at 10 characters per second.

Fortunately, programs running under OS-9 don't actually do any I/O. OS-9 is arranged so that input and output are done by OS-9 rather than by user programs. The device drivers are responsible for all I/O. OS-9 always gives very fast service to device drivers. Almost anything will be interrupted to allow a device driver to deal with input or output. Some device drivers have a reservoir for 100 (or so) characters which they can save up and give to a process in a burst next time the process is started.

For the best performance a computer should be kept idle most of the time, that way it will immediately jump on any work you give it. It would be nice to have enough money to buy overpowered computers so there would be lots of idle time and excellent performance, but if money is a concern you have to strike a compromise between getting fast response, and getting the maximum amount of work out of your machine.

It is possible to speed up important processes by changing their priority. The heavier the load on a computer, the more important it is to fuss with priorities. An edit session, a listing to the printer, and an assembly can share the machine very nicely if the priorities are properly set. The edit session is interacting with an impatient human, so it should have a high priority assigned to it. Since editing usually involves a lot of dead time while the human doing the editing stares at the screen, the editor will actually use very little processor time. The processor that is printing is very much the same story. It isn't interacting with a human, but even a 200 character per second printer is slow by computer standards. The process that is driving the printer should be given an intermediate priority so it will be able to run the printer at a good clip without interfering to any great extent with the edit process. The assembly should be given a very low priority. Assemblies are the type of thing that will use a lot of processor time if they are allowed. Even if it is given a low priority, the assembly will get time that the other processes don't want, so since both will usually be waiting for something, the assembly will get plenty of time.

Most business programs, as well as compilers, assemblers, and disk utility programs, spend a lot of time waiting for the disk to do something. The sound of a disk clucking and buzzing is a pleasant busy sound, but it actually signifies wasted time. While the disk is doing mechanical things like starting, seeking, loading the head, and even turning, some program is likely to be waiting. OS-9 makes some effort to speed disk access, but with several processes wanting to access the same disk the problem is more than a small operating system can handle. There are standard tricks for reducing the amount of time a program spends waiting for the disk drive. The easiest of these for a regular user to get at is the use of large buffers. Most programs that access the disk will run faster if they are given enough storage so they can read and write large blocks of data. If you want to hear some very busy noises from your drives, start a COPY with only a little bit of memory, then do a DIR for a large directory on the same disk you are COPYing on. The disk drive will chuckle madly as it shuttles back and forth from directory to file in an attempt to serve both the copy program and the DIR command. Switching from file to file on a disk (even a Winchester) is slow. The best way to deal with this is to avoid the problem by not running more than one program accessing a particular drive at a time. It will be obvious if there is a problem. If programs are run in the wrong combinations, they will run very slowly, and the disk will sound very active. If you have to make the best of a bad mix, give processes as much memory as you can. Well designed programs can use extra storage to cut down disk usage, or to transfer (read or write) more data for each turn they get.

AN ASSEMBLY LANGUAGE PROGRAM WHICH SETS PRINTER OPTIONS

I just installed a new printer on my system, an Okidata Microline 92 (nice printer). I used to set the options on my MX80 with a group of procedure files. An example would be the file called Comprint which contained the command "display 0f >/p". It would have been possible to set the printer to compressed printing mode by typing the display command instead of invoking the procedure file by typing /d0/comprint, but I can never remember the Epson control codes. Installing a new printer seemed like a good excuse to find a better way of setting the printer options. The program POpt is the first complete assembly language program I have published here. I hope you find it as useful as I do.

POpt doesn't do anything technically exciting, but it is a fairly simple assembly language program which includes most of the elements found in assembler programs. I am going to go through the interesting points of the program moving generally from the beginning to the end.

The NAM and TTL statements in the first two lines of the program are purely cosmetic. They provide information which the assembler puts in the page headings. The block following those two lines is the introductory comment for the program. All comments might be considered cosmetic, but although they don't generate any code, I think of them as essential parts of assembly language programs. Any line with an asterisk in column one is a comment; the box I draw around the comment is just to make it look nice.

If you are looking at the output of the assembler, the two lines after the introductory comment are IFP1 and ENDC. In the original source there is a line between these, "use /d0/defs/defslist", which calls in a list of USE commands which make files containing all the system definitions part of the program. These definitions help make the rest of the program more readable. The words Prgrm+Objct would have to be replaced with the much less understandable $11 if the system definitions, or some other similar set of definitions, weren't included in the program. Throughout the rest of the program I used the symbols defined in the definitions files (and a few additional SET commands in the program itself) whenever I could. The IFP1/ENDC which is wrapped around the use statement prevents the extra files from being read on the second pass the assembler takes through the file. No statement in the system definitions defines any memory so there is no reason for the assembler to read it on both the first and second passes; not reading it during the second pass saves a good deal of time, and prevents the lines in the definitions from being included in the program's line numbering. The lines used from the definitions files don't print both because they aren't read on the second pass (when output is generated), and because the first line in the definitions file is the assembly directive OPT -l which directs the assembler not to print anything until it encounters the OPT l directive. The definitions files I routinely include in assemblies are listed in table Table 1.

Table 1: Definitions files routinely included in assemblies

```
OS9Defs
OS9SysDefs
OS9IODefs
OS9RBFDefs
OS9SCFDefs
```

There are a lot of symbols in all those files; a program with the full set of definition files generally needs to be assembled in a region of at least 24K to accommodate the large symbol table. If you want to use your memory more economically, create a stripped down definitions file with only the definitions you expect to use, and use it instead of the standard files; but be prepared to scrap your file and build a new one if you get a new version of the operating system. Level One and Level Two definitely have different definitions, and if you dig around deep enough in the operating system, it is unwise to count on things staying fixed even from version to version.

A few lines down from the ENDC is the MOD statement. This statement generates the OS-9 module header, a block of data which OS-9 needs. The fields in the module header are:

- *PROGRAM LENGTH*

  Trying to fill in a number here would be foolish. The assembler can figure out the length of the module for you. The symbol PgmLen is defined in the last line of this program.

- *SYMBOL USED FOR PROGRAM NAME LOCATION*

  This isn't the program name itself, but the name you choose to assign to the location containing the name. I like the name "Name" for that location. The program's name is usually placed close to the module header, but it can be placed elsewhere in the module if it is convenient for you to do it that way.

- *MODULE TYPE*

  I like to define the module type as a symbol before the MOD statement and just put the symbol here. The module type tells OS-9 what kind of thing to expect this module to do. This module is a program (not a subroutine or data), and it consists of object code (not data or some sort of intermediate code).

- *REVISION*

  This field contains two types of information. It indicates whether the module is reentrant, usable by several different users at the same time, and gives the revision number of the program. Most well written programs are reentrant, so, since OS-9 uses reentrant modules more efficiently than non-reentrant modules, most programs

should be labeled reentrant. The revision number is used when a module is loaded from disk to determine whether the module should replace a module by the same name already in memory. A module with a higher revision number will replace a module with a lower number.This is particularly useful if you want to override a module which has been placed in ROM. Unless you want to supersede a module in memory the revision should be 1.

- *ENTRY POINT*

  The name assigned to the first instruction in the program. I usually insert a line before the first instruction in the program with this name on it. This saves a little bit of typing if I want to add instructions before the first instruction in a program.

- *MINIMUM AMOUNT OF PERMANENT STORAGE REQUIRED*

  The amount of storage the program will need in addition to the storage used for the module itself. This number, like the program length, can be calculated by the assembler -- note the MemSize equ . a few

The line after the MOD statement tells the assembler to reserve space for one byte of storage. The next two lines reserve 255 more bytes. The total memory requirements of this program are one byte for the printer's path number, and 255 bytes for the stack. The stack probably doesn't need to be that large for this simple program, but OS-9 is going to allocate memory in 256 byte units even on a level I system, so I played it safe and squandered the memory on the stack. The results of allocating too little space for the stack are very unpleasant.

The equate after the rmb for the stack uses the "." special symbol which means the current offset in the data definitions. This is an easy way to get the assembler to tell us how much storage the program will need for use in the MOD statement.

The next two lines are the module's name (pointed to by the module name field in the module header), and the version number. The module name must be defined with a FCS statement. This type of data definition closes the string it is defining by setting the high order bit on in the last byte -- "t" is $F4 instead of $74. This lets OS-9 know where the end of the name is. The byte after the name is by convention the version number of the program. Some utility programs display this number, but it is optional. Nothing awful will happen if you start right in with data or program after the program name.

The version number is the last overhead until the very end of the program. The fcc's and fcb's for the next 40 or so lines define constants needed in the program. About the only interesting thing about them is that each of the strings defined with a fcc is followed by fcb C$CR, a carriage return. At first it looks like I could have saved space by using one <CR> for all the strings, but it turns out that the extra code needed for that approach uses more memory than the extra carriage returns.

The program scans the parameter string, and if certain characters are found, sends character strings to the printer. There are three phases: first the input length, in D, is checked. If it is one (or lower) there is no parameter string; in this case display a menu of options. Second, scan the parameter string for the character "/" which denotes a device name. If there is a device name in the parameter, open that device as the printer, otherwise open the device /P. Third, scan the parameter string again ignoring any characters in a device name. Translate each character to upper case and compare the translated character to each significant character. Each time a significant character is found, transmit the appropriate character string to the printer, and send a line to the standard output path describing what has been done.

There are a couple of simple tricks which are useful while scanning the parameter string. The shell always terminates the parameter string with a carriage return. This lets me terminate the scan when I encounter a carriage return instead of having to count bytes. Data bytes may have the parity bit on or off. I remove the parity bit with "anda #$7F." If the parity bit is left on, twice as many comparisons need to be done. For example, "a" could be $61 or $E1. In this case, I thought it would be best to treat both upper and lower case characters as the same. The easiest way to do this is to translate all lower case letters to upper case (or vice versa if you like). Once you determine that a character is an upper case letter it can be translated to a lower case letter by subtracting $20 from it, or anding %11011111 with it.

There are two sections of this program responsible for output. Common1 writes strings two bytes long to the printer. It uses the I$Write service request which writes a specified number of characters without any editing. There is nothing special about two bytes; it is just the length of the longest control string I wanted to be able to send to the printer. I padded the shorter control strings to two bytes by adding a $00, a null, to them. Common2 writes up to 80 characters to the standard output path. Common2 uses the I$WritLn service request which treats the carriage return as a special case. When it encounters a carriage return it does whatever the path descriptor is set up to do on end of line (normally send <CR><LF>) and returns. This means that by terminating each string to be written by Common2 with a <CR> I make it unnecessary to know the length of any of the strings.

This program ends in either of two places. If there are no errors, after the second scan the program branches to Exit which clears the carry bit in the condition code and performs the F$Exit service request returning control to OS-9. If there is an error, control goes to ErrXit which sets the carry bit and returns control to OS-9. You might expect that the best way to set or clear the carry bit in

the condition code register is with the andcc and orcc instructions. Those instructions certainly are able to turn the carry bit on and off, but the COM instruction turns the carry bit on faster (and the CLR instruction turns it off faster) than the obvious instruction. Whenever the A or B accumulator is free, it is fastest to set or clear the carry flag by playing with the accumulator.

At the very end of POpt there are two final lines of overhead. The EMOD directive causes the assembler to generate a checksum for the module which is used when this program is run to make certain that the module is valid and undamaged. The line with "PgmLen equ *" calculates the length of the module for use in the MOD statement at the very beginning of the module.

THE OS-9 USER'S GROUP

An OS-9 User's Group was formed last summer. I couldn't say it's thriving, but it is coming along. The club has a telephone bulletin board, and lots of dreams. It isn't going to go anywhere unless plenty of OS-9 users join it. Membership is $25 for individuals (payable to OS-9 Users Group c/o Terry Straehley 1005 Roble Lane, Santa Barbara CA 93103). I strongly suggest that all OS-9 users join the group. Even with the relatively small membership the group now has, a lot of interesting information passes through the bulletin board. If we all join, this group could become a great resource.

THE FUTURE OF THIS COLUMN

There is enough material for another six months or more of columns about concurrent processes, but I am going to move on to some other subjects for a while. It seems there are a great many new OS-9 users out there, some of whom have written to me asking for help with the fundamentals of the system. The program this month is a first attempt to help these people. I'll try to devote at least part of this column to OS-9 basics for the next few months.

POPT PROGRAM

```
Microware OS-9 Assembler 2.1   08/05/84 23:06:32                    Page 001
POpt - Change Printer Setup Options for ML 93

00001                           NAM   POpt
00002                           TTL   Change Printer Setup Options for ML 93
00003          *------------------------------------------------*
00004          *   Printer Setup Options                        *
00005          *     c     Correspondence Quality               *
00006          *                                                *
00007          *     0     Ten CPI                              *
00008          *     2     12 CPI                               *
00009          *     7     17 CPI                               *
00010          *           Double Width Characters              *
00011          *     5     Five CPI                             *
00012          *     6     Six CPI                              *
00013          *     8     Eight CPI                            *
00014          *                                                *
00015          *     r     reset to initial conditions          *
00016          *                                                *
00017          *     /     Lead in for alternate path name. Default  *
00018          *           is /P. The path name must either be the   *
00019          *           last parameter, or separated from the next *
00020          *           parameter by a delimiter.            *
00021          *                                                *
00022          *    The options are specified as parameters when  *
00023          *    POpt is run. If no options are specified, a menu  *
00024          *    is presented.                               *
00025          *                                                *
00026          *    Examples:                                   *
00027          *         popt rc2                               *
00028          *    ->     Printer Reset                        *
00029          *    ->     Correspondence Quality Printing      *
00030          *    ->     Print Density twelve characters per inch  *
00031          *                                                *
00032          *         popt r6 /p1                            *
00033          *    ->     Printer Reset                        *
00034          *    ->     Print Density six characters per inch  *
00035          *    ->    {output was directed to the printer at /p1 }*
00036          *    You can put the print options on either or both  *
00037          *    sides of the device name...                 *
00038          *    popt rc /p1    is the same as   popt r /p1 c    *
00039          *------------------------------------------------*
00040                          IFP1
00042                          ENDC
00043   0011             Type     set   Prgrm+Objct
00044   0081             Revs     set   ReEnt+1
00045   0000 87CD04BA            MOD   PgmLen,Name,Type,Revs,Entry,MemSize
00046 D 0000             PrtPthN  rmb   1
00047   00FF             StackSz  set   255
00048 D 0001                      rmb   StackSz    space for stack
00049 D 0100             Memsize  equ   .
00050   000D 504F70F4    Name     fcs   /POpt/
00051   0011 01          Edition  fcb   1
00052   0001             StdOut   set   1          Number of Standard Output Path
00053   0012 2F50        DPrtNam  fcc   "/P"       Default Printer Name
00054   0014 0D                   fcb   C$CR
00055          *******
00056          *  Responses for each printer option set
00057          *
00058   0015 5072696E    Msg5CPI  fcc   /Print density five characters per inch (
00059   004A 0D                   fcb   C$CR
00060   004B 5072696E    Msg6CPI  fcc   /Print density six characters per inch (d
00061   007F 0D                   fcb   C$CR
00062   0080 5072696E    Msg8CPI  fcc   /Print density 8.5 characters per inch (d
00063   00B4 0D                   fcb   C$CR
00064   00B5 5072696E    Msg10CPI fcc   /Print density 10 characters per inch (no
00065   00E2 0D                   fcb   C$CR
00066   00E3 5072696E    Msg12CPI fcc   /Print density twelve characters per inch
00067   010B 0D                   fcb   C$CR
00068   010C 5072696E    Msg17CPI fcc   /Print density 17 characters per inch/
00069   0130 0D                   fcb   C$CR
00070   0131 436F7272    MsgCQ    fcc   /Correspondence Quality Printing/
00071   0150 0D                   fcb   C$CR
00072   0151 5072696E    MsgRst   fcc   /Printer Reset/
00073   015E 0D                   fcb   C$CR
```

```
00142          *-----------------------------------------*
00143          *  No alternate printer path found        *
00144          *-----------------------------------------*
00145  0343              Loop1D
00146  0343 308DFFCB             leax  DPrtNam,PCR
00147  0347 20ED                 bra   Loop11      Open the default printer path
00148  0349              Loop1E
00149  0349 3536                 puls  D,X,Y       restore
00150          *-----------------------------------------*
00151          *  Loop2 scans the parameter string for   *
00152          *  printer control options. If an option is *
00153          *  found the corresponding subroutine is  *
00154          *  called.                                *
00155          *-----------------------------------------*
00156  034B              Loop2
00157  034B A680                 lda   ,X+
00158  034D 847F                 anda  #$7F        clear parity bit
00159  034F 810D                 cmpa  #$0D        <CR>?
00160  0351 2738                 beq   Exit
00161  0353 8120                 cmpa  #$20        control character?
00162  0355 23F4                 bls   Loop2       yes; loop
00163  0357 812F                 cmpa  #'/         start of a path name?
00164  0359 102700E0             lbeq  SkipPN      Yes; Skip over the path name
00165          *-----------------------------------------*
00166          *      Translate lower to upper case if   *
00167          *      necessary.                         *
00168          *-----------------------------------------*
00169  035D 8161                 cmpa  #'a
00170  035F 2506                 blo   Loop21
00171  0361 817A                 cmpa  #'z
00172  0363 2202                 bhi   Loop21
00173  0365 8020                 suba  #$20        lower to upper case
00174  0367              Loop21
00175          *-----------------------------------------*
00176          *         Analyse the parameter           *
00177          *-----------------------------------------*
00178  0367 8152                 cmpa  #'R         reset?
00179  0369 2724                 beq   Reset
00180  036B 8143                 cmpa  #'C         Correspondence quality?
00181  036D 2734                 beq   CQ
00182  036F 8130                 cmpa  #'0         Ten CPI
00183  0371 2743                 beq   TenCPI
00184  0373 8132                 cmpa  #'2         Twelve CPI
00185  0375 2751                 beq   TwlvCPI
00186  0377 8135                 cmpa  #'5         Five CPI?
00187  0379 2760                 beq   FiveCPI
00188  037B 8136                 cmpa  #'6         6 CPI
00189  037D 276F                 beq   SixCPI
00190  037F 8137                 cmpa  #'7         Seventeen CPI
00191  0381 277E                 beq   SvntnCPI
00192  0383 8138                 cmpa  #'8         Eight and a half CPI?
00193  0385 1027008B             lbeq  EightCPI
00194  0389 20C0                 bra   Loop2


00195  038B              Exit
00196  038B 5F                   clrb              set B (return code) to 0 and t
00197  038C 103F06               OS9   F$Exit      return to OS-9
00198  038F              Reset
00199  038F 3410                 pshs  X
00200  0391 308DFDD8             leax  CCRst,PCR   point at Reset control string
00201  0395 17008F               lbsr  Common1     write it
00202  0398 308DFDB5             leax  MsgRst,PCR  point at remark
00203  039C 170094               lbsr  Common2     write it
00204  039F 3510                 puls  X
00205  03A1 20A8                 bra   Loop2       go search for next option
00206  03A3              CQ
00207  03A3 3410                 pshs  X
00208  03A5 308DFDB6             leax  CCCQ,PCR
00209  03A9 8D7C                 bsr   Common1
00210  03AB 308DFD82             leax  MsgCQ,PCR
00211  03AF 170081               lbsr  Common2
00212  03B2 3510                 puls  X
00213  03B4 2095                 bra   Loop2
```

```
00074                    ******
00075                    *  Printer Control Strings
00076                    *
00077    015F 1B31       CCCQ     fcb   $1b,'1      Set correspondence quality
00078    0161 1E1F       CC5CPI   fcb   $1E,$1F     Five CPI
00079    0163 1C1F       CC6CPI   fcb   $1C,$1F     Six CPI
00080    0165 1D1F       CC8CPI   fcb   $1D,$1F     Eight CPI
00081    0167 1E00       CC0CPI   fcb   $1E,0       Ten CPI
00082    0169 1C00       CC2CPI   fcb   $1C,0       Twelve CPI
00083    016B 1D00       CC7CPI   fcb   $1D,0       Seventeen CPI
00084    016D 1800       CCRst    fcb   $18,0       reset printer
00085                    **********
00086                    *  The Menu
00087                    *
00088    016F 4E6F206D   ErrMsg1  fcc   /No more than 127 bytes of parameters are
00089    01A0 0D                  fcb   C$CR
00090    01A1 492F4F20   ErrMsg2  fcc   .I/O error on printer path.
00091    01BA 0D                  fcb   C$CR
00092    01BB 504F7074   Menu1    fcc   /POpt accepts the following parameters:/
00093    01E1 0D                  fcb   C$CR
00094    01E2 2052202D   Menu2    fcc   / R - Reset the printer/
00095    01F8 0D                  fcb   C$CR
00096    01F9 2043202D   Menu3    fcc   / C - Correspondence quality print/
00097    021A 0D                  fcb   C$CR
00098    021B 2035202D   Menu4    fcc   / 5 - Print at five characters per inch/
00099    0241 0D                  fcb   C$CR
00100    0242 2036202D   Menu5    fcc   / 6 - Print at six characters per inch/
00101    0267 0D                  fcb   C$CR
00102    0268 2038202D   Menu6    fcc   / 8 - Print at eight and a half character
00103    029A 0D                  fcb   C$CR
00104    029B 2030202D   Menu7    fcc   / 0 - Print at ten characters per inch/
00105    02C0 0D                  fcb   C$CR
00106    02C1 2032202D   Menu8    fcc   / 2 - Print at twelve characters per inch
00107    02E9 0D                  fcb   C$CR
00108    02EA 2037202D   Menu9    fcc   / 7 - Print at seventeen characters per i
00109    0315 0D                  fcb   C$CR
00110    0316            Entry
00111                    ***************
00112                    * X points to the start of the parameter area.
00113                    * Y points to the end of the parameter area.
00114                    * The last character in the parameter area is a <CR>.
00115                    * D contains the length of the parameter area.
00116                    *
00117    0316 10830001            cmpd  #1          Check length of parameter area
00118    031A 10230137            lbls  Menu        if there is nothing there; Dis
00119    031E 10830080            cmpd  #128        It's hard to deal with paramet
00120    0322 1024017E            lbhs  Error1      high; parameter area too long

00121                    *-----------------------------------------*
00122                    * Search parameters for output device overide*
00123                    *-----------------------------------------*
00124    0326 3436                pshs  D,X,Y       save everything
00125    0328            Loop1
00126    0328 A680                lda   ,X+
00127    032A 847F                anda  #$7F        clear parity bit
00128    032C 810D                cmpa  #$0D        <CR>?
00129    032E 2713                beq   Loop1D
00130    0330 812F                cmpa  #'/         start of path name?
00131    0332 26F4                bne   Loop1
00132    0334 301F                leax  -1,X        back up one to /
00133    0336            Loop11
00134                    *-----------------------------------------*
00135                    *      Open alternate printer path        *
00136                    *-----------------------------------------*
00137    0336 8602                lda   #Write.
00138    0338 103F84              OS9   I$Open
00139    033B 1025014F            lbcs  Error2
00140    033F 9700                sta   PrtPthN     save the path number
00141    0341 2006                bra   Loop1E
```

```
00214   03B6             TenCPI
00215   03B6 3410                pshs  X
00216   03B8 308DFDAB            leax  CC0CPI,PCR
00217   03BC 8D69                bsr   Common1
00218   03BE 308DFCF3            leax  Msg10CPI,PCR
00219   03C2 8D6F                bsr   Common2
00220   03C4 3510                puls  X
00221   03C6 2083                bra   Loop2
00222   03C8             TwlvCPI
00223   03C8 3410                pshs  X
00224   03CA 308DFD9B            leax  CC2CPI,PCR
00225   03CE 8D57                bsr   Common1
00226   03D0 308DFD0F            leax  Msg12CPI,PCR
00227   03D4 8D5D                bsr   Common2
00228   03D6 3510                puls  X
00229   03D8 16FF70              lbra  Loop2
00230   03DB             FiveCPI
00231   03DB 3410                pshs  X
00232   03DD 308DFD80            leax  CC5CPI,PCR
00233   03E1 8D44                bsr   Common1
00234   03E3 308DFC2E            leax  Msg5CPI,PCR
00235   03E7 8D4A                bsr   Common2
00236   03E9 3510                puls  X
00237   03EB 16FF5D              lbra  Loop2
00238   03EE             SixCPI
00239   03EE 3410                pshs  X
00240   03F0 308DFD6F            leax  CC6CPI,PCR
00241   03F4 8D31                bsr   Common1
00242   03F6 308DFC51            leax  Msg6CPI,PCR
00243   03FA 8D37                bsr   Common2
00244   03FC 3510                puls  X
00245   03FE 16FF4A              lbra  Loop2
00246   0401             SvntnCPI
00247   0401 3410                pshs  X
00248   0403 308DFD64            leax  CC7CPI,PCR
00249   0407 8D1E                bsr   Common1
00250   0409 308DFCFF            leax  Msg17CPI,PCR
00251   040D 8D24                bsr   Common2
00252   040F 3510                puls  X
00253   0411 16FF37              lbra  Loop2
00254   0414             EightCPI
00255   0414 3410                pshs  X
00256   0416 308DFD4B            leax  CC8CPI,PCR
00257   041A 8D0B                bsr   Common1
00258   041C 308DFC60            leax  Msg8CPI,PCR
00259   0420 8D11                bsr   Common2
00260   0422 3510                puls  X
00261   0424 16FF24              lbra  Loop2

00262   0427             Common1
00263   0427 9600                lda   PrtPthN    Printer Path Number
00264   0429 108E0002            ldy   #2         length
00265   042D 103F8A              OS9   I$Write
00266   0430 255C                bcs   Error2     I/O error on printer path
00267   0432 39                  rts
00268   0433             Common2
00269   0433 8601                lda   #StdOut    output path for remarks
00270   0435 108E0050            ldy   #80        max length of strings
00271   0439 103F8C              OS9   I$WritLn
00272   043C 39                  rts
00273                    *-----------------------------------------*
00274                    *    Skip over alternate printer path name   *
00275                    *-----------------------------------------*
00276   043D             SkipPN
00277   043D A680                lda   ,X+
00278   043F 847F                anda  #$7F
00279   0441 810D                cmpa  #C$CR      <CR>?
00280   0443 1027FF44            lbeq  Exit       yes; done
00281   0447 8120                cmpa  #C$SPAC    <space>?
00282   0449 1027FEFE            lbeq  Loop2      end of path name
00283   044D 812C                cmpa  #',
00284   044F 1027FEF8            lbeq  Loop2      end of path name
00285   0453 20E8                bra   SkipPN
```

```
00286   0455             Menu
00287   0455 308DFD62            leax  Menu1,PCR
00288   0459 8DD8                bsr   Common2
00289   045B 308DFD83            leax  Menu2,PCR
00290   045F 8DD2                bsr   Common2
00291   0461 308DFD94            leax  Menu3,PCR
00292   0465 8DCC                bsr   Common2
00293   0467 308DFDB0            leax  Menu4,PCR
00294   046B 8DC6                bsr   Common2
00295   046D 308DFDD1            leax  Menu5,PCR
00296   0471 8DC0                bsr   Common2
00297   0473 308DFDF1            leax  Menu6,PCR
00298   0477 8DBA                bsr   Common2
00299   0479 308DFE1E            leax  Menu7,PCR
00300   047D 8DB4                bsr   Common2
00301   047F 308DFE3E            leax  Menu8,PCR
00302   0483 8DAE                bsr   Common2
00303   0485 308DFE61            leax  Menu9,PCR
00304   0489 8DA8                bsr   Common2
00305   048B 16FEFD              lbra  Exit
00306        *-----------------------------------*
00307        *          End with an error        *
00308        *-----------------------------------*
00309   048E             Error2  equ   *          Error in printer path
00310   048E 3404                pshs  B          save error code
00311   0490 108E0050            ldy   #80
00312   0494 308DFD09            leax  ErrMsg2,PCR
00313   0498 8602                lda   #2
00314   049A 103F8C              OS9   I$WritLn
00315   049D 3504                puls  B          recover error code
00316   049F 103F0F              OS9   F$PErr     print error message
00317   04A2 200F                bra   ErrXit

00318   04A4             Error1  equ   *          Parameter string too long
00319   04A4 108E0050            ldy   #80
00320   04A8 308DFCC3            leax  ErrMsg1,PCR
00321   04AC 8602                lda   #2         Error output
00322   04AE 103F8C              OS9   I$WritLn
00323   04B1 C601                ldb   #1         error code
00324   04B3             ErrXit
00325   04B3 43                  coma             set carry
00326   04B4 103F06              OS9   F$Exit
00327   04B7 285030              EMOD
00328   04BA             PgmLen  equ   *

00000 error(s)
00000 warning(s)
$04BA 01210 program bytes generated
$0100 00256 data bytes allocated
$24F1 09457 bytes used for symbols
```

COLUMN SIX

OS-9 by itself does very little useful work. You won't find an editor, assembler, compiler, spelling checker, or payroll system anywhere on the standard distribution disk. That isn't to say that you can't get these programs for OS-9, or even that some of them aren't sometimes packaged with the operating system (Gimix packages Microware's editor, assembler, debugger, BasicO9, and RunB with every OS-9 system), but OS-9 can be purchased with no frills, and in that form it is essentially useless.

For an experienced microcomputer user with lots of friends using OS-9 and a nearby store with a large stock of OS-9 software the task of choosing the right array of software could be fun, but for me it was frightening. The least expensive software I could find cost about fifty dollars a crack, and it went up fast from there. I didn't know anyone running OS-9, and, though there were many computer stores in Rochester, the only one which dealt in 6809 based machines believed strongly (nearly exclusively) in TSC software. I gritted my teeth and bought what looked good to me. I was surprised to find that everything I bought was at least OK. In retrospect I can see that it wasn't so very surprising that I was lucky in my software purchases; most of the software for OS-9 is good.

With OS-9 I got the Microware Editor, Assembler, Debugger, and Pascal. I have no special love for the Microware Debugger, but I still use it because it is the only game in town. It usually is packaged with OS-9, and it is hard to get along without, especially if you do assembly language programming, but I hope Microware feels a touch of humiliation each time they send out a copy of that program -- it is not up to the standard set by their other programs. The assembler is unexciting, but it does the job. There are other assemblers around, but the Microware assembler is the standard.

The Microware Editor is hard to classify. It is the only non-screen-oriented editor for OS-9 that I know of. It works fine as a simple editor, but it might be more accurate to call it a simple string processing language. The editor features multiple work spaces, and a high powered macro language which can be used to write fairly sophisticated programs. The bad side of all this sophistication is that it is a little bit hard to use the editor for simple things. I have never been able to figure out how to copy a range of lines without using a disk file as a temporary holding place. I don't use the Microware Editor very frequently since I got a screen-oriented editor, but I got a lot of work done on it when it was the only editor I had, and I still use it occasionally. I should add that some people think editors like the Microware editor are better for programming than the more word processing oriented editors.

It is hard for me to be moderate in my praise for Microware's Pascal. I wish it included a debugger, and the procedure for linking to external procedures is a bit clumsy, but I love it. I use it to develop programs for classes where the students use DEC Pascal and IBM Pascal and have no compatibility problems. There are enough enhancements to make this Pascal useful for real applications (such as a PROMPT built-in procedure which forces out the contents of an output buffer without a carriage return). The compiler generates intermediate code which can be executed by either of two interpreters (one normal, and the other supporting large programs by a paging arrangement), or translated into efficient native code.

Recently I got BasicO9. You may have guessed from my comments that I am getting to like it even though it is called Basic.

I have DynaStar, DynaForm, and DynaSpell from Frank Hogg Labs. None of these programs are exceptional, but I use them all regularly. DynaStar is a screen-oriented editor with which I have typed and revised many hundreds of pages. It is best at editing documents, but usable for programs. I expect the reason the program is called DynaStar is that it borrows heavily from Wordstar. My mother uses Wordstar, and I find that I can help her untangle some problems with Wordstar by assuming that it is keystroke for keystroke identical with DynaStar. I have some small complaints about DynaStar, but the bottom line is that I like it well enough to have spent hundreds of hours using it.

DynaForm is a text formatting/mail merge program. It is full of fancy Mail-Merge features that I never use. I use it to print files with optional page headers and trailers, underlining, and bold printing. A few times I have used its ability to generate indexes and a table of contents. DynaForm doesn't do well when compared to the high powered text formatting packages used on large computers, but I don't think it is intended to compete with that kind of thing. The thing about DynaForm that annoys me most frequently is that it can't be customized to use the special features of my printer. It prints bold text by simply printing the bold characters three times. DynaStar can be used to imbed printer control characters in text, but DynaForm only knows one way to print bold or underlined text. I also wish it would use the standard input and output paths instead of allocating special paths.

DynaStar and DynaForm were written by Allan Jost. They show signs of being written by a programmer with a very professional attitude. They are not loaded with features but they are so reliable that I just take them for granted.

DynaSpell is a spelling checker. I need a spelling checker very badly. Some people buy computers to run a spreadsheet program. I might have bought one to run a spelling checker. DynaSpell essentially looks up each word in a document in a set of dictionaries. Any words that it doesn't find are treated as questionable words. These words can be fixed, accepted as is, or accepted and added to a dictionary. DynaSpell isn't as carefully written as the programs by Allan Jost; there is nothing major wrong, but the meticulous care isn't there. When DynaSpell runs out of space to store words in, it spews out pages of

"overflow" messages. There is no way to check the contents of the directory when DynaSpell is asking for the name of the file to check. When you abort the program (with a control C) in order to check the directory again, DynaSpell leaves the terminal's device descriptor in a strange state. DynaSpell has most of the features commonly found in spelling checkers for microcomputers, but it doesn't compare with similar programs on larger machines. Maybe a spelling checker is one of those tasks which needs fast machines with large memories. I want a spelling checker which helps me correct misspelled words by giving me a list of suggested spellings, and a built in thesaurus would be another nice touch. Still, I use DynaSpell when it is inconvenient to ship my files off the the IBM to be checked. It isn't a great program, but it does its job.

I reviewed DynaCalc a few months ago. I still like the program, and it is still heavily used. I wouldn't have chosen DynaCalc as part of my core group of software (I mostly program and write with my computer) but I can imagine people who might not need any other program.

NEW RELEASE OF MICROWARE PASCAL

I just got release 2.0 of Microware's Pascal. It is a major revision, including a new intermediate code language, a single general purpose I-code-to-native-code translator, and new run time support modules. I didn't do any careful comparisons of the two versions, but I get the strong feeling that the new release compiles faster, and runs faster. The new manual is significantly better organized and more complete than the old one, but still makes no attempt to teach Pascal. Two new standard functions have been added: GETCHAR, which returns a single character from input, and IDREADY, which returns true if there is input ready. These new functions should be useful for interactive applications like editors.

OS-9 DIRECTORIES

A directory is a special type of file containing information about files. It could be seen as something like a library's card file. It contains the names of files along with information about them, especially where they can be found. Unlike anything a proper library would contain, the entries in a directory aren't kept in any particularly useful order. You can get a formatted listing of the contents of a directory with the DIR command.

OS-9, like UNIX and many other multi-user operating systems, supports hierarchies of directories on disk. Directories can be used for a number of things, or, if you like, largely ignored. A directory can contain any number of other directories in adoition to normal files.

Every disk has a root directory on it which is created when the disk is formatted, and cannot be done away with. Unless you fuss around with INIT and SYSGO the disk you boot off of must have a directory called CMDS in its root directory. There may also be a SYS, and a DEFS directory in the root directory on the boot disk when you install OS-9.

You (the user) can create new directories with the MAKDIR command. To use the command type MAKDIR followed by the name of the new directory you want to create:

```
MAKDIR /D1/SOURCE.DIRECTORY
```

It has become a convention to use capital letters for directories' names. OS-9 doesn't have any trouble with lower case directory names, but it is an easy way of reminding oneself which files are directories.

It is sometimes tricky to keep track of a library of several hundred (maybe thousand) files. Multiple directories are a major help in organizing files in such a way as to maximize the chance of finding them again. Long ago I found that I couldn't fit all my files on one disk (that was a 100K floppy back then). I put each major project on a separate disk. When I got disk drives with greater data capacity, I found that it wasn't an unadulterated good thing. Each disk contained so many files that it was a major job to locate a file even knowing which disk it was on. I worked out naming conventions that made the job easier, but they used up the first two characters of each file name -- the resulting file names were pretty cryptic. I still keep hundreds of files on each disk, but my largest directory has about forty files in it.

The root directory on a disk I have labeled "pascal1" contains nothing but seven directories: DIST.SRC, UTIL.SRC, SUBR.SRC, BUGS, DEFS, DOC, and PCODE. Each of those directory names describes what I expect to find in them pretty well (to me anyhow). Each directory with programs in it contains a directory called DOC which contains related documentation. If it seems like I have large numbers of directories called DOC, it's true. Pretty near everything needs documentation. Sometimes I find that a directory begins to get out of control. Projects that I expect to need about ten files have a way of expanding to forty or fifty files. A project like that really belongs in a directory of its own, so I create a new directory in the directory that contains the files for the project, and move all the files that are part of that project into the new directory.

Any file can be accessed by giving its full name, e.g., /D1/UTIL.SRC/DFIX/Compacter would denote the file Compacter in the directory DFIX which is in the directory UTIL.SRC in the root directory on disk D1, but that's more typing than I would choose to do except as an act of desperation. The most commonly used shortcuts are the CHD, and CHX commands. The CHD command changes the directory which is treated as the root directory for data. CHX does the same thing for the execution directory.

When OS-9 is booted the data directory is set to the root directory of the boot disk, and the execution directory is set to

CMDS in the root directory on the boot disk. If you want to use files in the root directory on the boot disk, all you need to do is give the file name, if you want to use files in a directory which is in that directory you give the name of the directory with the file name, e.g., to get at the file OS9Defs in DEFS in the data directory use DEFS/OS9Defs. If the default data directory isn't convenient for you, a new directory can be selected with the CHD command, for example, to change the data directory to the root directory on /D1 use CHD /D1. The CHX command works the same way CHD does, but it effects the execution directory.

There are two special entries in every directory. The "." entry points to the directory itself, and the ".." entry points to the directory the current directory is in, the parent directory. A typical use of the ".." entry is to refer to sibling directories. When a project gets large, I break it up into a set of directories, all in a directory which I set aside for the project. If a program needs access to the file HexDefs in the directory DEFS which is a sibling of the directory SRC (where the program is), I can use the shorthand name "../DEFS/HexDefs" for the file. I have found this a good convention to stay with. As long as I continue to keep related families of files in directories that are siblings, the notation "../DEFS" will always get me to the appropriate DEFS directory, and "../DOC" will always refer to the related Documentation directory.

To experiment with directories, start with a disk with some empty space on it, and use CHD to set the data directory to the root directory. Build some directories:

```
MAKDIR TESTD1
MAKDIR TESTD2
MAKDIR TESTD3
```

Make things a little more complicated:

```
CHD TESTD2
MAKDIR TESTD21
MAKDIR TESTD22
MAKDIR TESTD23
CHD TESTD21
MAKDIR TESTD211
MAKDIR TESTD212
MAKDIR TESTD213
CHD ../TESTD22
MAKDIR TESTD221
MAKDIR TESTD222
CHD ../TESTD23
MAKDIR TESTD231
MAKDIR TESTD232
CHD ..
CHD ..
```

Now we're back at the root directory. The DIR command should show the files that were in the directory before you started this experiment plus the directories TESTD1, TESTD2, and TESTD3. DIR TESTD1 will show an empty directory. DIR TESTD2 will show the directories TESTD21, TESTD22, and TESTD23. The following commands will all show the contents of the directory TESTD23:

```
DIR TESTD2/TESTD23
DIR ./TESTD2/TESTD23
CHD TESTD2 ; DIR TESTD23
CHD TESTD2/TESTD23; DIR
```

The first two command lines leave the data directory at the root directory. The third command line moves the data directory to TESTD2, and the fourth command line moves the data directory all the way out to TESTD23.

It is easy to create new directories, but a little involved to delete a directory. Perhaps it is a good thing that it requires more than one quick operation to remove a directory. If a directory with files in it is erased, all the files in the removed directory will remain on the disk, but OS-9 won't be able to locate them. Older versions of OS-9 don't have any command which will delete a directory. To do away with a directory with these older versions: delete all the files (and directories) in the directory, use the ATTR command to change the directory into a normal file (ATTR <dirname> -d), and delete the file that used to be the directory. Be particularly careful not to use ATTR to change the directory into a regular file until the directory is empty. There is no easy way to change the file back into a directory so you can delete the files in it. With the new release of OS-9, the command DELDIR can be used to delete directories. DELDIR simply automates the steps I just went through.

Directories are an important feature of UNIX-like operating systems. They allow files to be grouped in manageable clusters, and make it easier to handle many concurrent users.

I am preparing to eat some of the words I set down in my first column. I am looking forward to this with a good deal of pleasure -- they were critical words. Some people have gone to a fair amount of effort to convince me that I was wrong. If things go well I'll hold the word eating ceremony next month.

STANDARD TERMINAL SUPPORT FOR OS-9

One of the first programs I wrote for a micro played "Life." The game starts with a given pattern, and, by repeatedly applying a set of rules, generates and displays new patterns. If the patterns are displayed properly on a CRT, the changing figures on the screen can be fascinating. (Note: Life was invented by John Horton Conway, and has been extensively discussed in Scientific American and BYTE.) I wanted my program to be usable with most terminals; so after investing a few days in the program, I spent another few weeks trying to make it "device independent." I never really finished. It was an uncommonly fast game, but, since I couldn't generalize the terminal control, no-one without a Hi9 will ever be able to enjoy it.

Many micros avoid this problem by not using a terminal (e.g., the Color Computer), but people, like me, who program computers without a built-in screen must either use only those control codes common to all terminals (like carriage return, and line feed), or expend a lot of effort writing special code to handle different terminals.

Full screen editors are the prime example of a type of program that must have control of some of the features of the terminal, but many other high quality programs support some of the features that most terminals share. Every program with generalized terminal support must be configured for the terminal (or terminals) it is supposed to work with as part of the installation of the program.

Some programs use a special module which contains terminal-specific code for a few crucial functions. It is simple to install a program that uses this kind of terminal control provided that the necessary module is provided. If a suitable module for your terminal is not available, a new one must be written in assembly language.

Another approach to generalized terminal control is to use a configuration program to ask questions about the terminal being used and store the information in tables which enable a single terminal control module to drive any reasonable type of terminal.

It is sad to see so much effort used solving the same problem over and over. It is so hard to write a program so it can be adjusted for use with any terminal that even some commercial programs don't do it. For small programs it can take more work to implement terminal support than to write the rest of the program. Frank Hogg Labs seems to have developed a standard for terminal control, the GOTOXY module. Once the module is installed for one program, it need not be done again except for a new type of terminal. If every software distributor would standardize on GOTOXY, it would make life a lot easier for programmers and purchasers of software. Frank Hogg tells me the GOTOXY modules are not proprietary, so this is an alternative -- UCSD Pascal makes do with no more. Unfortunately, GOTOXY is hard to call from some languages, and supports a terribly limited set of operations.

I would like to propose a standard interface for CRT terminals. It would be much easier for Microware to build the standard control system than for me to do it, but it looks like the job is mine. I will kick the problems I find around for a month or so. Please help me with this. If I have to devise a standard in a vacuum, it may not please enough people to be widely used.

Any standard is a compromise. The most important goal is to make it easy for any programmer to use the interface. This rules out all the language-specific interfaces. The other two important goals are that all the currently existing programs with (or without) terminal control modules must continue to operate without modification, and that the interface should provide the most sophisticated terminal control possible.

Since many languages can't use GETSTAT/SETSTAT, or other exotic ways of doing I/O, I believe the standard terminal control module should either be a callable module like GOTOXY, or some form of filter. The callable module would be more efficient; but different languages call subroutines differently, and it would be sad to forsake the built-in I/O facilities of a language in order to route all terminal I/O through a single module. There are several places a module could be placed in the terminal I/O path where it could act as a filter isolating terminal specific control strings on the terminal side of the filter, and standard strings on the program side. I don't believe that the difference in efficiency between the filter and the subroutine method of terminal control is all that great. The filter method seems to be the best approach to the terminal-independent program problem.

The filter method requires that all programs act as if they are being used with some standard terminal. That terminal could be imaginary, but with so many different terminals available why invent another. Two terminals seem like attractive choices; the VT52 and the VT100. The VT100 is especially attractive because it implements the ANSII standard. It would be nice to go with the accepted standard, and I think I will finally decide to use a subset of the ANSII standard -- a subset because I don't relish the idea of trying to emulate all those flashy features on a dumb CRT. The worst disadvantage of the ANSII standard protocol is that its cursor control sequences will be hard to generate in assembly language programs. The row and column have to be in ASCII characters. It hurts me to think of a programmer being forced to include binary-to-ASCII conversion code in his program just so the terminal control module can convert the numbers back to binary. The VT52 is representative of most moderately intelligent terminals. It certainly includes every function I would want to include in the subset of the ANSII standard I plan to implement. In the short run the VT52 is a better choice than the VT100; it could be emulated more efficiently, and would be just as useful as any practical subset of the ANSII standard.

Still, I believe that in the long run adhering to the most widely accepted standard is the best policy. I am looking for a good excuse to use the VT52 as the standard, but haven't found a good enough one yet.

The choice of the subset is another tricky decision. The minimum useful subset is either the direct cursor positioning command, or the set of cursor up, down, left, and right commands. Actually, home cursor is adequate for most purposes, but it takes a substantial amount of work to program for a terminal that is that dumb. There are many powerful commands that make it easier to program for a terminal, and, more important, cut down the number of characters that need to be sent to the terminal to accomplish some operation. If fewer characters need to be sent to (say) clear the screen, then the screen will clear faster and the number of interrupts the computer will need to service will be decreased. However, the more fancy terminal control commands are included in the standard, the larger the terminal control module will get.

There is no reason the filter trick can't be applied to terminal input as well as output. For some of the less powerful terminals it will be necessary to pass all input through the filter it order to know where the cursor is; however, all terminals will benefit from filtered input. An input filter will permit standard program function keys, arrow keys, the clear screen key, and perhaps some other special keys to be defined.

The following is a list of terminal control strings in descending order of likelihood to be in the subset:

- Direct cursor positioning
- Clear to end of line
- PFkeys/Clear Key/Arrow Keys
- Alternate cursor (block/underscore)/normal cursor
- Highlight on/off (either reverse video or intensify)
- 25th (or other special) line support

The following are significantly harder:

- Save cursor position/return to saved position
- Insert/delete line
- Delete character
- Enter/leave insert character mode

I will consult everyone I can think of about this, and hope the people I don't think of will write or call me with their thoughts. After a month or two's thought, I will try to write the code to support the standard for at least one terminal. I would appreciate any help or advice I can get.

COLUMN SEVEN

Last month I promised that I would eat some words this month. In the first column I wrote for 68 Micro Journal, I said that I was sorry no one was using more than 64K for a single program under OS-9, and I made the point rather strongly that 6809-based computers should not be shared.

Several months ago David Brown asked me to look at his version of MUMPS for the 6809. Strictly speaking, since MUMPS doesn't run under OS-9, it is out of my area, but it is intriguing. The version of MUMPS David Brown sent me uses a fairly sophisticated virtual memory scheme, and is not effected by 64K boundaries. Since it doesn't run under OS-9, I still challenge someone to be the first with a program that uses more than 64K at once under OS-9, but since Dave Brown's work is impressive, I gave it a mixed but generally nice review.

My mother is the secretary of the school board back in the town where I grew up. She has given me a very interesting pipeline into the workings of a municipal school system. Recently there has been a lot of fuss about computers at school. Pre-college schools have to make a number of difficult decisions in the process of integrating computers in the educational process. Even the choice of the best computer is complicated for them by the scarcity of good software for their purposes (and their uncertainty concerning what software they need), and by the worst kind of financial problems. When I heard that my home town was going to commit itself to a gaggle of microcomputers running Basic, I felt motivated to research the subject with an eye toward talking them out of Basic. The OS-9 users' group's bulletin board is often a good source of information, and in this case it was surprisingly useful; it turns out that many OS-9 users are involved in education. Once started on the idea that OS-9 might be a good solution for some of a schools system's problems, I rubbed some figures together and came to some conclusions that shouldn't have surprised me.

It is clear that financial considerations are crucially important to all the school systems I know of. One micro can be inexpensive enough to fit into a budget, but one Apple is not very useful for teaching a class of thirty. I figure that a high school computer lab should be set up to teach Pascal, word processing, the use of a spread sheet, and the use of computers in the sciences. I know from experience that students can be lab partners and work as a team of two without too much trouble, but three or more students working together will have problems. Figuring thirty students in a class, the lab will need fifteen stations. The minimum configuration I can put together is fifteen micros, each including:

- A spread sheet -- $100.00
- Pascal -- $200.00
- Wordstar type editor -- $300.00
- Operating system -- $100.00
- One 5.25" floppy drive and controller -- $600.00
- A printer -- $250.00
- A monitor -- $200.00
- The micro -- $500.00

All those prices are rough, but reflect the cheap alternative, not the quality that students deserve. Each micro will come to $2250.00 (though I doubt that they could actually be put together for that little). Fifteen of them cost $33750.00. That's serious money, and it only buys a minimal system for each lab team.

If a large OS-9 system could handle fifteen students, it would be possible to purchase a top of the line CPU with a hard disk, a floppy disk, fifteen serial ports (intelligent), a half meg of memory, and top of the line software, for about $14,000.00. Fifteen very nice terminals would cost $9000.00 bringing the cost of the system to $23,000.00. Two thousand dollars will buy a very nice printer, bringing the total cost to about $25,000.

I have talked to several people who run many users on a Gimix-III system. If half of what the Gimix-III users say is true, it would be reasonable to have eight or ten students sharing a machine. If all that they say is true, it might be possible to hook thirty students to one CPU and expect them to run at a reasonable speed. I now have a second terminal on my level two system. I can say from my own experience that my system can handle two users with very few signs of being loaded down.

Based on what I know about my system, and what I have been able to find out about Gimix-III, I think a Gimix-III system with at least 256K of memory would be able to handle four to six users with a level of service that I would find acceptable. Given a choice of a toy computer with bargain basement software, and the bare minimum of peripherals, or a fifteenth of a fully configured Gimix-III system; I would pick the piece of a large system like a flash.

I confess to being an ivory tower idealist. I want people to like computers, so I flinch at the idea of giving out slices of computer so small that there is not enough power to allow software to be friendly. That means that I think a individual deserves at least a level two system with lots of memory. Realistically, most hobbyists can't afford to commit that much money to their computer; businesses need a much stronger argument than friendly relationships between staff and computers; and schools simply have to choose the least expensive way to do things most of the time. I maintain that I am philosophically opposed to sharing micros, but if I am forced to consider the alternatives, I am strongly in favor of sharing a computer -- provided it is the right computer.

A LETTER

Don Williams sent me a letter from Bengt-Allan Bergvall who sent along an interesting program that amounts to a special sort of shell for BasicO9 programs. It gives me encouragement in my plan to write a enhanced shell, but is useful as it stands. His letter follows this column.

The assembler program which was included with the letter, and which I will include here, is an interesting extension on the program called "StrtTask" which I gave a few months ago. If we were using real UNIX we would solve the problem of passing parameters to BASIC09 programs by modifying the shell; ParamMod is a sort of special purpose mini-shell which runs BASIC09 programs.

LETTER FROM BENGT-ALLAN BERGVALL

Microware's BASIC09 is an excellent interpreter, easy to use for producing your own utilities. Unfortunately, it is lacking a straightforward method of passing parameters. For example, if you are going to write a "Help" utility, you want to type

```
OS9:help dir
```

to learn about the dir command. This is impossible if Help is a BASIC09 program. If Help is a packed BASIC09 program, interpreted by RunB, you can type

```
OS9:help
```

only, and let the program ask you what help you want. If you don't have RunB, you have to type

```
OS9:basic09 #5k help
```

However, even if Microware doesn't tell you, you can also pass parameters in RunB or BASIC09 by using the syntax

```
OS9:help ("dir")     or OS9:basic09 #5k help("dir")
```

and using the PARAM statement in the Help program. This is OK if you will use the program rarely, but if the program will be used often, and perhaps not by yourself, this is a very clumsy syntax.

The desired syntax can of course be accomplished by writing Help in another language that permits the desired parameter syntax, i.e. in assembler. This is probably the wrong way for a user utility program. To solve the problem, I have written a short "universal" program in assembler, called ParamMod, with the following characteristics:

- ParamMod allows the desired parameter syntax.
- ParamMod transforms the parameter list from the desired syntax to the syntax required by BASIC09 or RunB. The resulting parameters are all of the type STRING. To be used as numeric types, the strings have to be transformed using the VAL function.
- ParamMod forks to either BASIC09 or RunB, and the main program is written in BASIC09.
- ParamMod has to duplicated and customized on three text strings and needed BASIC09 memory for each utility:
  - innam

    The wanted utility name. In the given case, Help. Other utilities could be names Compare or Analyze.
  - Outname

    The name of the file that contains the BASIC09 procedure and performs the desired action. It could be named Help_B or /D0/COM/Compare_B or AnalyzeBody.interprt.
  - interprt

    The name of the BASIC09 interpreter to be forked to. Either BASIC09 if outname is a saved procedure or RunB if it is a packed procedure.
  - Memory

    The total number of bytes needed for the procedures and their data areas.

In the following, we are assuming you are writing a Help utility. For other utilities, change the names accordingly.

First customize ParamMod's three text strings and BASIC09 estimated memory size with your text editor. Then assemble it with Microware's assembler, using the command:

```
OS9:asm ParamMod o=Help #10k
```

and the resulting code for Help will be in your execution directory.

Then write your BASIC09 program, naming the outermost procedure Help_B. You must save or pack Help_B to run it through Help. RUN it from within BASIC09 with the command (including parameter):

```
B:$help dir
```

You may also during the development phase run the program without Help. In that case you must use BASIC09 parameter syntax:

```
B:run help_b("dir")
```

Included is the assembly listing for ParamMod, customized for a Help utility and a dummy Help_B program.

Bengt Allan Bergvall
Blavingev. 1
S-561 49 Huskvarna
Sweden

PARAMMOD

```
* Program written by Bengt-Allan Bergvall, Blavingev. 1,
* s-561 49 Huskvarna, sweden.
*
* Program to reformat a parameter list from an easily
* typed form to the clumsier form required when running a
* BASIC09 program.
* Given the command
* OS9:help param1 param2   param4  (note the extra space)
* This program will fork to the RunB or BASIC09 program
* Help_B as if given the equivalent command:
* OS9: BASIC09 #5k Help_B("param1","param2","","param4")
*
*
* This program is general and can reformat the resulting
* parameter list up to 256 characters, but the name
* strings inname and outname has to be changed for each
* implementation.
*
* if interprt is runB, then outname has to be a packed
* BASIC09 program in the execution directory.
*
* If interprt is BASIC09 then outname has to be a saved
* BASIC09 program either in the present data directory or
* in another file with outname giving the full path name,
* e.g., /D0/COM/Help_B
*
* The memory needed by BASIC09 or RunB must also be
* given.
 nam parameter list modifier
 ttl for BASIC09 or RunB
 ifp1
 use /D0/DEFS/defslist
 endc (use os9defs)
 mod pgend,inname,prgrm+objct,reent+1
 fdb pgstart,stack
* data variables
parend5 rmb 2 output parameter limit -5
outpar rmb 256
varend equ .
stck rmb 200 stack area
stack equ . stack pointer
*****************************************
** Customization area

inname fcs .Help. Name of utility
outname fcs .Help_B. Name of BASIC09 procedure
 fcb 0
interprt fcs .BASIC09. Either BASIC09 or runB
* Total memory needed in bytes by BASIC09 or RunB
* process: (equivalent to the needed BASIC09 MEM value)
memory equ 5000

** End customization
*****************************************
pgstart

* Modify parameter list from free form into BASIC09
* string form. Example of free form: param1 param2  param4
* Resulting BASIC09 string form:
*  Help_B("param1","param2","","param4")

* prepare limit check for parameter list, allow for
* ending last parenthesis.
 leay varend-5,U
 sty parend5

* copy outname into output parameter list
 pshs X
 leay outpar,U
 leax outname,PCR
namechar lda ,X+
 beq nameend
 sta ,Y+
 bra namechar
nameend puls X input parameter list

* append modified input parameter list to output
* parameter list
 lda #'(
 sta ,Y+
```

```
 lda #'"
 sta ,Y+
parchar lda ,X+
* check the resulting parameter list not too long
 cmpy parend5
 blo parOK
 comb set carry
 ldb #56 BASIC09 parameter error
 OS9 F$Exit

parOK cmpa #$20 space?
 beq nextpar
 cmpa #$0D carriage return ends parameter list
 beq lastpar
 sta ,Y+
 bra parchar
* reformat next parameter
nextpar lda #'"
 sta ,Y+
 lda #',
 sta ,Y+
 lda #'"
 sta ,Y+
 bra parchar
* list end
lastpar lda #'"
 sta ,Y+
 lda #')
 sta ,Y+
 lda #$0D carriage return
 sta ,Y+

* fork to interprt (RUnB or BASIC09)
 leax interprt,PCR
 ldy #$100 allow one page parameters
 leau outpar,U
 lda #prgrm+objct
 ldb #(memory+255)/256 data area
 OS9 F$Fork
 bcs ut
 OS9 F$Wait
 bcs ut
 clrb no error
ut OS9 F$Exit
 emod
pgend equ *
 end
```

HELP_B

```
PROCEDURE Help_B
REM Dummy Help utility
REM prints the parameter
PARAM text:STRING
PRINT text
BYE  \REM bye needed to give automatic return to OS-9 when run by Basic09
```

THE OS-9 USER SEMINAR

On August 12 the OS-9 User Seminar opened rather slowly as I and a few other people stood in line in front of the exhibit hall on the third floor of the Des Moines Marriott. We watched as various Microware staff struggled to get a Radio Shack computer (running OS-9 of course) interfaced with a television. When I got into the hall, I was surprised at the number of exhibitors. I have always thought of the OS-9 community as about the size of a large family -- there were 24 booths listed in the exhibitor guide. I had a ball wandering through the hall, meeting people I have only known through phone conversations, and seeing some exciting hardware.

Several of the exhibitors were showing machines that used OS-9 as a process control environment. One booth sported a rack of equipment that would have been more at home next to an assembly line.

Smoke Signal broadcasting had a video tape rig showing a movie of a military-looking man. I remember a bugle and a lot of strutting up and down, but I just can't remember what he was talking about; I think he was promoting the TMP package. Smoke Signal had a compact SS50 based machine that I have never seen before.

There were a couple of Japanese engineers demonstrating Fujitsu FM-7 and FM-11 computers. Very well done. I wish they were available in this country. A particularly nice feature of the software on the Fujitsu machines was split-screen support. I saw them editing on one part of the screen while two other sections displayed moving graphics ... all running at the same time.

Tano was showing a Dragon computer, imported (I believe) from England. The Dragon is a small, inexpensive computer with color graphics and OS-9 Level One. I only saw it playing games, but it does that pretty well.

Privac was showing the graphics board that I have been coveting for months now. It looks even better in reality than it appears in an advertisement. There was a program running almost continuously that demonstrated the board. Figures and characters would appear, disappear, rotate, and float across the screen. I had always wondered how well the Privac board was supported under OS-9; it turns out that OS-9 is the operating system they use. The demo program was written in BasicO9.

Wires from the Gimix booth seemed to spread all over the hall. The OS-9 User Group, JBM Group, and Frank Hogg Labs all were borrowing computer services from Gimix. Perhaps to demonstrate the tireless ability of GMX-III to spew characters out on many terminals, unused terminals were kept busy listing strange programs. Whenever I walked by, one of the terminals at the FHL booth was listing a COBOL program. At the Gimix booth I met the engineer responsible for my hardware (who is also the president and the service manager of Gimix). He thinks Gimix hardware should move in about the same direction I want it to go. If things go well, there should be some terrific new hardware coming out sometime in the indefinite future.

The JBM Group is hard for me to characterize. They had some utilities that sounded good except that they were written at least partly in BasicO9. The thing that upset and fascinated me was that they have a sort which they claim runs very fast. They claim to have compared their sort to a standard disk-based merge sort, and come out significantly better. Either the algorithm used by the program they compared theirs to was not the best available in the literature, or their claim may have to be placed in the same class as perpetual motion machines. The man who invented their sorting algorithm wasn't there for me to ask about the details of his method, and I had no way to check their figures, so I will continue to view their sort with skepticism; however, even if it is only an average sort, its manual documents a fine general sorting program of a type which is much needed by serious OS-9 users. They had several other packages including a set of BasicO9 subroutines for ISAM file handling that sounded much less exotic, but interesting.

There were, of course, many exhibitors I haven't mentioned (for example Microware's own booth), but I don't intend to make this column into a walking tour of the exhibit hall.

Friday there was plenty of time to look around. Saturday and Sunday were so busy there was barely time to eat. Microware filled most of the weekend with classes, presentations, and "roundtables" ranging from OS-9 and BasicO9 Features, which covered things like the BasicO9 editor, to the OS-9 Roundtable, which gave us a chance to interrogate the parents of OS-9 about its workings. In the evenings a few of the exhibitors ran "hospitality suites" which gave some of us an excuse to stay up late and talk about our computers.

Saturday night there was a meeting of the OS-9 User Group. The User Group is having some troubles which seem to stem mostly from having only a few members spread over a wide geographical area. We elected officers for the next year: Dale Puckett (President), myself (Vice President), Goerge Dorner (Treasurer), and Tom Murphy (Secretary). We are respectively responsible for the Software Exchange Committee, the Membership Committee, the Communications Committee, and the By Laws Committee.

Monday those of us who were still left around went off to Microware's offices. I had a chance to discuss some of the difficulties I am having with OS-9 and C with the appropriate people, and discovered that those programmers are seriously crowded. They desperately need to make the move to a larger facility that they have been planning.

SHELL COMMANDS

The shell is a program that interprets command lines and does what is called for. The full UNIX shell is a programming language in itself. The OS-9 shell is only a subset of the UNIX shell, but it has enough flexibility to be useful. The first thing to learn about the shell is how to use the built-in shell commands. The chd, chx, ex, kill, w, and setpr commands are built into the shell. The shell commands are used to control the environment of the programs that are run by the shell.

I use the chd command, which is the command which changes the working data directory, more than any other shell command. The working data directory is the directory which will be used for most files you read or write without specifying a directory in the file name. It is usually much better to change the working directory than to explicitly include directories in file names so I frequently change directories as I change from one task to another. It is a rare day when I use the chx command, the command which changes the working execution directory, even once. I imagine that someone with a smaller system disk than mine would use the chx command much more frequently than I do because OS-9 remembers where the working directories are on disk, and needs to be reset with chd and chx commands when a disk is changed. If you forget to change directories when you change disks, OS-9 will give you a nasty message next time you try to use the directory. I have never gotten into trouble by forgetting, but it is not wise to trust an operating system too far.

The ex command should be classed as an advanced command. It replaces the shell with another program. Replacing the shell is certainly a good thing to be able to do, especially for users with smaller systems, but it can have disconcerting results -- mainly that when the program ends, the shell won't be there.

The rest of the shell commands are primarily useful for those who run programs concurrently. You can instruct the shell to start a program running, then give you another shell prompt by putting an & after the command on the command line:

```
OS9: dir >/p&
```

would list the files in the data directory on the printer while you run other programs.

If you run programs concurrently, the kill, setpr, and w commands will be useful. The kill command should be used about the way you use the quit control key (usually <CNTL>Q or <CNTL>E). The quit key only works on the last program to do I/O to the terminal, the kill command works on any program. The setpr command is used to control the way the computer's resources are divided up. The higher the priority of a program, the larger a share of the computer it will get, and the faster it will run. A program's (or, more properly, process's) priority can be anywhere in the range 1 to 255. The w command causes the shell to wait for a child process to finish. That means that the shell won't prompt for another command until a program that was started by it terminates. The main use of this command is to recover from the mistake of running a program that does I/O to the terminal in background. The usefulness of the w command can be appreciated by trying the following experiment:

```
OS9: dir x&
```

Now try to get some useful work done ... when you are disgusted with the screwy behavior of your terminal, type w at the OS9 prompt:

```
OS9: w
```

There is one particularly nice feature of the shell which is, so far as I know, undocumented. If you run a program like the assembler with its output directed somewhere other than the terminal, then decide that you would like to run another program at the same time, you can cut the assembly loose from the shell with the interrupt control key (usually <CNTL>C). The interrupt control key will usually terminate the program which most recently did I/O to the terminal, but, if the program in control of the terminal (the assembler in this case) doesn't do any I/O to the terminal at all, it won't kill it. Instead, the shell sees the interrupt, and converts the program in control of the terminal to a concurrent program.

A LOGICAL DEVICE DRIVER

This column is an experiment with a new format. There is a demand for information for new OS-9 users, but I have also heard requests for more advanced discussions. In this column I am trying to include something for everyone. What follows may be of general interest, but, for an inexperienced computer user, it may be heavy going.

Several months back I started a project whose objective was to find a way to give OS-9 a terminal-independent way to control CRTs. I have a special device driver which does just what is called for, but it is built around Microware's ACIA source. I may be able to get permission from Microware to publish the modified driver, but I would rather not have the terminal mapping tied that closely to the computer's I/O port. Not every computer uses an ACIA chip for its serial interface, and my special driver only works with ACIA serial interface chips. What is needed is a virtual, or logical, device that can insulate the terminal mapping code from the physical interface.

The idea of a logical device driver has many applications beyond a terminal-independent interface. At the User Seminar I spent some time talking to the engineers from the Fujitsu booth. They wanted my opinion of a proposal to make logical devices a part of OS-9 in order to allow the system drive to have a consistent name regardless of the type of hardware being used. This would make it easier to write programs that referenced files on the system drive. A logical device can certainly do this. It would be possible to set up a logical disk with some obvious name like SYS, and have it know the name of the phys-

ical drive being used as the system drive: D0, H0, or whatever.

The idea can be extended even further. There is no compelling reason why the logical device should refer to exactly one physical drive. The logical device could refer to several physical devices or just a part of one.

Some possible uses of the concept are:

- A disk drive with associated cache storage.
- A neat, and fairly easy way to support split screen terminal displays with each section of the screen treated as a separate terminal.
- A gateway to a network.
- A way to associate a printer with a terminal for screen dumps.
- A terminal-independent interface.

The device driver I have included with the column is a logical SCF device driver for a Level Two system. A RBF driver, or a driver for Level One would be somewhat different, but only the details would need to be changed. The driver, which I named VCIA, doesn't do anything at all except waste time. It is a skeleton for something interesting to be built on.

Starting from the top, let's go through the interesting parts of the program. A logical driver must look just like a real driver to the system, so it must have the type Drivr+Objct, and it must have a byte after the normal module header reflecting the modes in which the driver can be used. This one says it is good for update, it might be a good idea to add execution. The storage required by a device driver is called the device static storage, it is allocated, and partly initialized by the file manager, in this case SCF. The file manager uses the first section of the device static storage, the storage reserved for it with the "org V.SCF."

Performance is crucial at this level. Every character read from or written to the terminal will pass through this module so even a small improvement in efficiency is good. Normal good coding practice is still important, but the priorities shift somewhat. SCF will branch to entry, entry+3, entry+6 ... depending on what service it wants. The normal convention is to put a list of lbra instructions here, but a bra instruction is a little faster, so I used them and padded them to three bytes each with nops (which are never executed in this context).

The INIT call must find the physical device driver and set up the proper environment for it. The physical driver, which I call P.D., is found and mapped into the address space with a F$LINK call. This is a bit tricky. In Level Two, the module is linked into the address space belonging to the process doing the link. The device driver uses the process number of the process that opened it, but it runs in the system address space. I had to fool the operating system into thinking I was running under the system process number by playing with the pointers in the system direct page. If this were Level One, I think I could have simply used a F$SLink call without all the fussing around.

The real device driver is going to need its own device static storage, so the logical device driver plays SCF for a moment and gets the amount of storage P.D. needs. I use the Level Two system memory request, Level One users can probably use the Level One analog. The address of the memory must be saved for future calls, and I save the size for convenience.

One never knows when SCF will change its part of the static storage, so before each call information from VCIA's static storage must be copied to P.D.'s static storage, and after each call information must be copied back from P.D.'s static storage, to VCIA's.

The INIT call, and each other call, basically changes from VCIA's static storage to P.D.'s and calls the appropriate entry in P.D.

The TERM entry is responsible for cleaning up as the device is closed. After calling P.D. to allow it to close down the physical device, it frees the device static memory that INIT allocated for P.D., and unlinks P.D. It is worth noting that TERM is the mirror image of INIT.

The device descriptor I use for VCIA is called VTERM. Since SCF thinks VTERM is a real device, its device descriptor is important. Even the address of the port that VTERM uses is important. If you keep things as simple as I did, the logical device driver will map everything including the information from the device descriptor directly to the physical device driver, but, if you want to support something like split screen, you will have to give each logical terminal a different port address, or SCF will know they all are referring to the same device, and get in the way.

Fortunately this virtual driver works with GIMIX I/O processors. This is pure luck because GIMIX doesn't publish enough information about their driver for me to design an interface for it. Let's consider this a gentle push for Richard Don at Gimix to release more information about his proprietary software.

It should be possible to move a good deal less data back and forth between VCIA's and P.D.'s static storage than I do, but I played it safe in spite of the large cost. It would be good to try to find some fields that don't need moving so time could be saved by not moving them.

This module demonstrates that, although OS-9 Level One is compatible with Level Two for user programs, it is not compatible for system modules. This shouldn't be a surprise, but it is something to be cautious about. In many cases all that needs to be changed is an entry in a definitions file, but if you try to run VCIA as it stands in a Level One system, the best you can hope for is that it will give an error code and quit.

Debugging code in system state is not something I will do if I have a choice.[3] The debugger won't work on modules that need to run in system state, and debugging code that writes out helpful messages as a program runs doesn't make sense in a device driver module. If the driver doesn't work what do you write to? I debugged this module by using its return code. If you have VCIA set carry before it returns to SCF, the program that is trying to use it will get the value that was in the B accumulator when VCIA returned to SCF. This a slow way to learn things, but it works.

One final point: it is expensive, but otherwise impeccable technique to pile logical device on logical device. VCIA has no way of knowing whether the device it believes controls the physical device is real or logical.

[3] Microware is said to have an "Over the top" debugger that can be used to debug system software. The only person I know that has looked into it says it doesn't work with version 1.2 of OS-9. It seems Microware now uses debugging hardware.

```
 nam vcia
 ttl Virtual (logical) device driver
*------------------------------------------------------*
*    This module should be used as a SCF (Sequential  *
*    Character File) device driver.  It doesn't       *
*    drive any specific device, but, rather, calls    *
*    a physical device driver, such as ACIA, to       *
*    deal with the physical device.                   *
*         Possible uses:                              *
*         *  Mapping various terminals to a standard  *
*         *  Implementing windows.                    *
*         *  Linking a PIA and an ACIA to provide     *
*            switchable printing of terminal output   *
*                                                     *
*    As it stands this module is a dummy. It passes   *
*    all calls through with minimum interference.     *
*                                                     *
*    The INIT call must set up the environment for    *
*    the device driver before passing the call on.    *
*                                                     *
*    Read, write, getstat, and setstat can probably   *
*    get away with less than they do, but all         *
*    variables are mapped between control blocks      *
*    to ensure that this module is transparent.       *
*                                                     *
*    The TERM call must release memory allocated for  *
*    the physical driver before returning.            *
*------------------------------------------------------*
  IFP1 use /D0/DEFS/defslist
  use /H0/DEFS/defslist
  ENDC
Type set Drivr+Objct
Revs set ReEnt+1
  MOD VciaLen,Name,Type,Revs,Entry,MemSize
  fcb Updat. Driver can be used for updated (read + write)
Name fcs /VCIA/
  fcb 1 edition number
PDNam fcs /ACIA/
**********
*   Device static storage for this virtual driver
*
  org V.SCF room for SCF variables
PDModH rmb 2 Pointer to Physical Driver's Header
PDModE rmb 2 Pointer to Physical Driver's Entry
PDModM rmb 2 Pointer to Physical Driver's Static Memory
PDModMS rmb 2 amount of memory allocated for PD static mem
MemSize equ .
  spc 2
*************
*  Block of entry points
*
Entry
  bra Init
  nop pad out each entry to three bytes
  bra Read
  nop
  bra Write
  nop
  bra GetStat
  nop
  bra PutStat
  nop
  lbra Term
  spc 2
**************
*   Init finds the driver it will use a physical device driver,
*   and allocates and initiallizes its static storage
*   Passed:
*         U   Points to static storage
*         Y   Points to device descriptor
*
Init
  pshs Y,U
*********
* adjust process number to system process
* so the link will be into the system address space.
*
  ldd D.Proc
  pshs D save A
```

```
 ldd D.SysPrc
 std D.Proc

 lda #Type driver module type
 leax PDNam,PCR point X at the name of the P Driver
 OS9 F$Link
 puls D restore old process number
 std D.Proc
 lbcs Error1


 ldx 2,S copy address of device static store to x
 stu PDModH,X save P.D.'s Module Header address
 sty PDModE,X save P.D.'s Entry address

 ldd M$Mem,U memory requirement of P Driver
 OS9 F$SRqMem request system Memory
 lbcs Error1
 std PDModMS,X save amount of memory allocated
 stu PDModM,X save pointer to memory

************
*  At this point X points at vcia's static storage
*  U points to P.D.'s static storage
*
 ldb #V.SCF length to move
************
*  Move the entire set part of the device static storage
*  into P.D.'s static store.
*
LMoveE
 decb
 bmi XMove
 lda B,X
 sta B,U
 bra LMoveE go around loop again
XMove
 puls Y but leave U in the stack
**************
*  U points to P.D.'s static storage
*  Y points to the device descriptor
*
 ldx PDModE,X
 jsr D$Init,X do P D init

 tfr U,X
 puls U
 pshs B
********
* now X points at PD static store
* U points at vcia static store
*
   bsr MapIn

 puls B,PC return to SCF
 spc 2
*-------------------------------------------------------*
*    SCF needs to see any changes the physical device   *
*    driver makes to V.Paus, (or V.Err).                *
*    X points at P.D. static storage.                   *
*    U points at vcia static storage.                   *
*-------------------------------------------------------*
MapIn
 ldb V.Paus,X
 stb V.Paus,U
 ldb V.Err,X
 stb V.Err,U
 rts
 spc 2
Read
 ldb #D$Read
 bra Common
 spc 2
GetStat
 ldb #D$GSta
 bra Common
 spc 2
PutStat
 ldb #D$PSta
 bra Common
 spc 2
Write
```

```
 ldb #D$Writ
 spc 2
*-------------------------------------------------------*
*    Code used by all entries except INIT               *
*    Passed B -- offset from P.D's entry point for this op.*
*-------------------------------------------------------*
Common
 pshs D,U
 ldx PDModM,U
*-------------------------------------------------------*
*    The physical device driver needs to have           *
*    V.LPRC, and V.BUSY copied to it each time it is    *
*    called.                                            *
*    At this point U points at vcia static storage.     *
*    X points at P.D static storage.                    *
*-------------------------------------------------------*
 ldd V.LPRC,U
 std V.LPRC,X
 ldd V.LPRC+2,U
 std V.LPRC+2,X
 ldd V.LPRC+4,U Line
 std V.LPRC+4,X
 ldd V.LPRC+6,U Dev2
 std V.LPRC+6,X
 ldd V.LPRC+8,U Intr
 std V.LPRC+8,X
 ldd V.LPRC+10,U PChr
 std V.LPRC+10,X
 ldd V.LPRC+12,U XOn
 std V.LPRC+12,X
 spc 2
*-------------------------------------------------------*
*    The P.D. requires Y, and A to be as they were      *
*    when vcia was called.  A  and Y haven't been       *
*    been disturbed.                                    *
*    U must point to P.D.'s static storage.             *
*    When Common was called,                            *
*    B pointed to the offset from P.D.'s entry point    *
*    that we should jump to.                            *
*-------------------------------------------------------*
 ldx PDModE,U
 ldu PDModM,U
 puls D recover offset in P.D. and A from stack
 jsr B,X
 spc 2
 tfr U,X point X at PD static storage
 puls U recover pointer to vcia static
 pshs B save return code
 bsr MapIn (X points to PD static, U points to vcia static)
 puls B,PC return to SCF
 spc 2
 spc 2
Term
 ldb #D$Term
 bsr Common call PD
 pshs CC,B,U CC and error # from P.D.

 tfr U,X
 ldu PDModM,X address of PD static storage
 ldd PDModMS,X size of memory
 OS9 F$SRtMem return memory
 bcs Error2

 ldu PDModH,X address of module header
*********
* adjust process descriptor to system process
*
 ldd D.Proc
 pshs D save D
 ldd D.SysPrc
 std D.Proc
 spc 1
 OS9 F$UnLink unlink PD
 puls D recover old process descriptor
 std D.Proc and restore it in place
 bcs Error2
 spc 1
 puls CC,B,U,PC return to SCF
 spc 2
Error1
 puls Y,U
 rts
```

```
Error2
 leas 2,S clear CC and B off stack
 puls U,PC return to SCF
 spc 2
 EMOD
VciaLen equ *
 use vterm
 END
```

PROTECTION

As I was working away, distracted by the problem of choosing a topic for this month's column, I deleted a bunch of files by mistake. Worse, I didn't notice that I had done myself in until minutes later -- too late to get the files back. This event made the choice of a subject for this month substantially easier. The first topic for this month is file security.

Users on OS-9 are known by a number. If you use OS-9 as it came off the distribution disk you will be the only user and have the user number 0. User 0 is special: UNIX users would call him the superuser. The superuser has special privileges that enable him to circumvent the protection of files. All other users, and, to some extent the superuser, are separated from disk files by OS-9's file protection scheme.

If you use the DIR command with the "E" option:

```
OS9: DIR E
```

you will get a list of the files in your current working directory with a lot of information about each file as illustrated in Figure 2.

```
   directory of .  19:50:32

Owner Last modified attributes sector bytecount name
----- ------------- ---------- ------ --------- --------
    1  83/05/10 2234  ------wr     C0      4141 Column4
    1  83/09/11 2351  -s--r-wr     AF      45BD Column8
    1  83/06/11 1630  --------     B4      6081 Column5
    0  83/09/14 2036  d-ewrewr     BD       3E0 PROGRAMS
    1  83/08/28 1614  ----r-wr    245       8E9 Dictionry
```

Figure 2: Output of DIR Command

The information in this display that relates to a file's protection are its owner and attributes. All the files in this directory, except the file (a directory file) called PROGRAMS, belong to user number 1. The type of protection given to a file depends on the contents of the attributes field.

The first position in the attributes field is for the directory attribute. Directory files have several special characteristics, the one relating to protection is that they can't be deleted with the DEL command.

The second position in the attribute field is the shareable attribute. If there is an "s" in this position, the file can only be accessed by one process at a time.

The next six positions in the attribute field are two groups of three attributes each: public execute, write and read, and private execute write and read. If a public attribute is on (indicated by a letter instead of dash in that position) then any user can do that class of operations. If a private attribute is on, the owner of the file can do that class of operation.

The file called Column4 has typical protection. User 1, who is the owner of the file, can read or write it, and nobody else can do anything to it except observe that it is there.

Column8 is protected such that any user can read the file, but only user 1 can write to it. It also has the non-shareable attribute which protects it against being accessed by more than one user at a time. The non-shareable attribute prevents things from getting confusing when user 1 is updating the file and some other user is reading it, by preventing that situation from coming about. Whoever gets to the file first has exclusive access to it until he closes it. If several users want to read a file at the same time there is usually no reason not to let them do so, problems start to appear when a user wants to write to the file while other access it, and things get really sticky if several users want to update the file at the same time. The non-shareable attribute is most important when several users want write to a file concurrently.

The protection of column5 demonstrates one of the more useful applications of file protection in a single user system. It is impossible for anyone, even the owner of the file, to write to it without first changing its attributes. Since the class of operations controlled by write protection includes writing, renaming, and deleting, a file which is write protected can't be deleted by mistake. If I had write protected my files I wouldn't have been able to delete them without thinking about it.

It would appear to be impossible to ever delete a write protected file, but the owner of the file can use the attr command to change the attributes. The procedure for deleting a write protected file is: use the attr command to remove the write protection:

```
OS9: attr column5 w
```

Then delete the file with the normal del command.

None of the data files in this directory have the execute attribute. They are all text files and manifestly not executable. OS-9 will only load a file for execution if it has the executable attribute. The separation of the execute attribute from the read attribute makes it possible

to create an execute-only file. It would be difficult for someone to copy, dump, or disassemble an execute-only file. The execute-only attribute is a useful trick for protecting proprietary software.

```
Echo Press carriage return to initiate logon >/TE M
Echo Press carriage return to initiate logon >/IT1
Echo Press carriage return to initiate logon >/IT2
TSMON /IT1&
TSMON /IT2&
TSMON /TERM
```

Figure 3: Sample Startup file

A particularly sneaky problem is related to the execute attribute. Merging executable files together to form a file with all the modules used by some program, or to allow a set of popular utilities to be loaded compactly under Level Two, will create a file which doesn't have the execute attribute. OS-9 won't let you execute or load the resulting file. It gives an error 214, "NO PERMISSION." The fix for this problem is to use the ATTR command to give the file the executable attribute.

If you don't intend to have more than one user on your computer there is no reason for you to worry about user numbers. If you want to share your computer with other people -- either taking turns using the computer or using OS-9 as a multi-user operating system -- it is a good idea to have a separate user number for each person who uses the computer. The best way to set your user number is to start the TSMON process in the startup file. The last line in the startup file should be something like:

```
TSMON /TERM
```

TSMON will just sit there until you type a carriage return. This may give you the impression that something is wrong with the computer unless you are ready for this stolid lack of activity. To comfort myself I include the line:

```
ECHO Press carriage return to
initiate logon>/TERM
```

before the TSMON command in the startup file. It leaves directions on the screen after I boot the system. If you are lucky enough to own a system large enough to support three terminals, the sequence of commands in Figure 3 should be included in the startup file to get everything going. It is important to start the last TSMON as a foreground task (no &).

The main business of TSMON is done by the LOGIN command. The LOGIN command uses files called password and motd which must be in the SYS directory on the same disk the default data directory is on (normally /D0). The password file includes the user-name, user-number, and, optionally, a password for each user authorized to use the computer. It also includes a lot of information used to set up the environment for each user. The full contents of each line in the password file are:

- User Name
- Password
- User Number
- Initial priority
- Initial execution directory (usually .)
- Initial data directory (usually .)
- Initial program to execute (usually "shell")

The login command prompts for a user-name, and, if that user has a password in the password file, for a password. If the user-name isn't in the password file, or the password isn't correct, LOGIN announces the mistake and prompts for user name again.

The login command protects each user number from unauthorized use by insisting on getting a good user-name/password match before letting someone use a user-number. Many different users can share a user- number, allowing them to share files in a group, but each user-name can only be associated with one user number.

If you find a need to change your user number in the middle of a session with your computer you may be able do it with the LOGIN command. The LOGIN command can only be used if your default data directory is on the same disk the password file is on. The LOGIN command needs to read the password file. If you protect the password file against public read to keep everyone from browsing through the passwords, nobody but the superuser can use the LOGIN command.

The motd file contains the "message of the day." If there is any text in motd it will be displayed on the screen each time anyone logs on. It can be used to display a general greeting, or to give system status information of general interest; e.g., "We are running a new release of Pascal today."

Some tricks can be done with the "initial program" in the password file. It is possible to specify not only the initial program, but also a parameter string for it. This opens up extensive possibilities. Most operating systems allow a user to have the commands in a file (sometimes called a user profile or a login command file) executed every time he logs on. If you are willing to accept some limitations, the initial command can be used to do much more than start a shell for you when you log on.

The simplest possible entry in the password file might go something like:

```
myname,,3,100,.,.,shell
```

which would set up a user called myname. Myname would have a usernumber of 3, and would be started with a priority of 100. His data and execution directories would be standard -- for most systems /D0 and /D0/CMDS. Whenever myname logs in a shell will be stared for him.

A somewhat more demanding user can make the password file do much more for him. If the line in Figure 4 is inserted in the password file, it sets up a user with a password of xyzzy, gives him non-standard data and execution directories, and runs FREE and MFREE for him before leaving him running a shell.

```
hisname,xyzzy,2,150,/D0/HISDIR,/D0/BASICX,shell free;mfree;ex shell
```

Figure 4: Password File Entry

The important thing is that the sequence of commands the user wants executed must start with the name of the program that will interpret the rest of the line. If that program is the shell, the last command in that shell's parameter string must be an ex for whatever command you want to start the user with.

If you want to start a user with a particularly long script of commands, perhaps enough commands to hold him for an entire session, use a shell command file. The trick is to have the initial command be "shell" with a file name as the parameter. If the file isn't in the default data directory its full path-name must be specified. A sample password file entry might go like:

```
hername,wltrs,5,130,.,.,
   shell her.cmd.file ; ex shell
```

In this case the file "her.cmd.file" must be in the system default data directory. The command file invoked at login is just like any other shell command file. The important restriction to remember is that the shell command file is run by a different shell from the one that the user will be using when the command file is finished. If you change the directories in the command file, those changes will effect only the shell running the commands, not the shell that will be running after the command file is done.

THE "SUSPEND STATE"

Microware has added a nifty performance enhancement to the latest version of OS-9. They discovered that device drivers were spending a significant amount of time using the F$SEND service request (SR) to communicate between the interrupt service routine for the port and the rest of the device driver. In order to understand why the send was done you need some background in the way the OS-9 SCF device drivers work. The simplest way to write a device driver is to read and write to the port directly from the read and write entries of the driver, but this requires that the driver go into a wait loop while the interface chip is performing the operation. A wait loop isn't a bad thing if the processor has nothing to do until the I/O is complete, but, in an environment like OS-9, there are likely to be several tasks waiting to get done. The "right" way to write a device driver under OS-9 is to have the actual I/O done by an interrupt service routine, and have the read and write entries of the device driver share queues with the interrupt routine.

A character to be written goes to the write entry of the device driver which puts the character into the write queue if there is space for it, or goes to sleep if there isn't. The interface chip should be set to generate an interrupt whenever it is ready to write another character. The interrupt service routine will be started every time an interrupt is received from the port it is responsible for. If the interrupt was an output interrupt, the interrupt service routine will take a character out of the output queue and send it to the port. If the device driver is sleeping, waiting for an empty slot in the queue to appear, the interrupt service routine should send it a wakeup signal.

The procedure for reading a character is roughly the reverse of that for writing. The queue for input goes from the interrupt routine to the read entry and the device driver sleeps if a read is done when the queue is empty.

All this sending from the interrupt service routine to the driver is expensive. A new system state called the "Suspend State" was invented to keep device drivers from having to use F$SEND requests to start and stop its read and write operations. The "suspend state" is a lot like a light nap. The process is in the grey area between sleep and activity. Suspended processes remain in the active process queue where they quickly age to the top of the queue, but while the suspend bit is on in their process descriptor they can't be scheduled. To wake a suspended process up just turn the suspend bit off in its process descriptor. The following code would wake a suspended process from the interrupt routine of a driver:

```
ldx (Address of process
     descriptor for the process
     you want to awaken)
lda #255-Suspend
anda P$State,X
sta P$State,X
```

This sequence of instructions can be done a great deal faster than a F$SEND.

A process can suspend itself by turning the suspend bit in its process descriptor on, then sleeping for a tick. The sleep is just a way of giving up the rest of the time slice. Even without the F$Sleep, next time the dispatcher sees the suspended process descriptor it will treat it as suspended, and won't start it again until the suspend bit is turned off.

There are a few important limitations to the suspend state. The first is that a process can't get out of suspend state on its own. The second limitation is that the suspend bit is in the process descriptor which is the in the system address space. A non-system process has no easy way of directly modifying the process descriptor. The last limitation is implicit in the advantage of suspend state, suspended processes stay in the active process queue. They will slow the dispatcher down slightly because it will have to pass over them each time it looks for the next process to run.

COLUMN TEN

MORE ABOUT COMPUTERS AT SCHOOL

I had my first chance to look through a microscope when I was very young. My sister was deeply ingrossed in the microscopic world so I, being a typical younger brother, hung around and made a pest of myself until she showed me what she was working on. I couldn't see anything but a blur which sometimes faded out altogether. I didn't see much point in looking at a blur. As years went by I was given my own microscope, but chemistry sets, and my own experiments, were much more interesting. I still had trouble getting interested in blurs.

In ninth grade I encountered a real microscope for the first time. It was a fine old instrument. The teacher treated it with great respect, and insisted that we do the same. When I first used it I got a surprise that stays with me to this day. It was nothing like the microscopes I had used before, focusing it with the fine adjustment knob was no problem, and when something was in focus, even a single cell or a bacterium, it was very clear. I could have happily spent weeks peering through the eyepiece at everything I could fit on the stage. Eventually the class moved on to other things, but I had a new appreciation for the world of the very small.

It is unfair to blame my parents for not getting me a high quality microscope when I was eight, but it bothers me to think of what I missed. I was fascinated by what the microscope revealed when I was a teenager. The effect would have been even stronger if I had been younger.

My experience with the microscope is what makes me keep complaining about the tendency of schools to use the lowest quality hardware and software they can find. The younger the students, the lower the quality. The argument is that sophisticated hardware and software isn't needed for any but the most advanced students. This is a serious error. With computers "foolproof" means either trivial, or very sophisticated. It requires good hardware, and excellent software to deal satisfactorily with the worst a child can do. The kids at most schools are getting the same kind of experience with computers I got with my early microscope ... only a blurry image of what it should be.

The section of a column I wrote a few months ago about computers for schools has drawn more comment than any other column I have written, maybe more than all of them put together. Some people wrote to agree, others disagreed. I was glad to hear from those who agreed with me, but I was most interested in the letters from people who took issue with one or more of my points. Two of my points drew particularly heavy criticism. I calculated the price of an imaginary (but realistic) single-user computer. Several people thought an adequate computer could be purchased for less than I suggested. I also spent some time wishing schools would stop using Basic. It didn't surprise me that several readers felt Basic was a fine language.

The little story about the microscope was intended to address the question: "why bother to provide decent computers at school?" Students should be given a chance to use a computer that they don't have to struggle with, and a language that encourages clear thinking. Kids don't know enough to complain about Basic on the cheapest computer that can be found. I do, so I am complaining for them.

....

There were about five more paragraphs here about Basic, and the evils of skimping on computers for children, but while rereading the column I decided that I sounded a bit shrill. Please forgive the abrupt transition, but the smooth conclusion of this argument has been pruned with a quick block-delete.

PIPES

One of the most useful features of OS-9 (and UNIX) is the pipe. Pipes by themselves aren't good for much, but if you build a good set of "software tools," pipes make many tasks surprisingly easy.

A pipe is a special device which forms a connection between two programs such that the output from one is directed into the input of the other. The shell is a major user of pipes. You can ask the shell to connect the standard output of one program to the standard input of another by putting an exclamation point "!" between the commands. The "!" separates commands like the ";" and "&" do, but it also redirects the output of the command before it into the input of the command after it. You could get the same effect by using intermediate files (Have the first command save its output into a disk file. When the first command ends, run the second command with its input coming from the file the first command wrote.), but intermediate files are neither as fast nor as easy to use as pipes.

When you first start using OS-9, pipes won't be of much use to you. For one thing they are a bit confusing, but, more important, the standard OS-9 utilities don't include many filters.

A filter is a program which reads from the standard input file and writes to the standard output file until end of file on standard input. They can be used without pipes, but, in combination with pipes, a good toolbox of filters can be among the most useful facilities available under OS-9.

The most elementary filter would simply copy bytes from standard input to standard output. More advanced filters change data on its way through. Some common filters sort the data, break it into words, remove duplicate lines, count bytes, words, and lines, and translate upper case letters to lower case.

It is relatively easy to write special filters to solve problems one at a time. The trick is to write filters which, in combination with others, can do lots of useful things. I have a filter which I

call "words" (available from the OS-9 Users Group, but too long for this column) which breaks the input up into one word per line. I wrote another program which counts the number of <CR>s in the input and writes that number out when the end of the input is reached. I can hook those two programs together with a pipe to form a command line that counts the words in a file:

```
OS9: words <column10 ! linect
```

That command line feeds column10 into words which slices it up, one word per line. The output of words is fed into the standard input of linect which responds by giving me the number of lines in its input -- the number of words in column10. I can use linect by itself to find the number of lines in a file.

I have written other filters called sort and uniq. Sort sorts the standard input into the output. Uniq removes duplicate lines; for example:

```
Line One
Junk Line
Junk Line
Junk Line
Another line
```

would come out of Uniq

```
Line One
Junk Line
Another line
```

The command line:

```
OS9: words <col10 ! sort ! uniq
```

would break column10 into words, sort the list, remove duplicate lines, and give me a sorted list of the words I used in that column.

Since I have written a number of programs in assembler and Basic09 for this column, I thought I might include a few filters written in Pascal this month. Unfortunately old releases of OS-9 had a flaw in PIPEMAN which prevented it from working with Pascal programs. Pascal rewinds its standard input file when it starts. PIPEMAN wouldn't put up with a rewind with the upshot that filters written in Pascal couldn't even get started. The easiest language I know for writing filters is C, but since C isn't as widely used as assembler and Basic09, I'll include two filters, BWord in Basic09, and CharCt in assembler.

Both BWord and LineCt are crude programs. They are nowhere near as efficient as they can be. In particular, reading one character at a time is intensely bad practice under OS-9. Both of these programs could be generalized by using command parameters more extensively.

CharCt counts the number of occurrences of the first character in the command line parameter area in the standard input file. It could be generalized to look for character strings, or regular expressions. It might also be improved by using more than three bytes for the counter.

The shell always places at least a carriage return in the parameter area passed to a program it starts (FORKs). CharCt relies on this to give it an easy way to default to counting carriage returns in its input. If you want to count some other character use it as a parameter on the command line:

```
OS9: charct . <testfile
```

would count periods in testfile.

```
OS9: charct <testfile
```

would count carriage returns in testfile.

BWord splits the input file into lines, one word per line. A word is defined as a string of characters between spaces, tabs, or carriage returns. It would be more generally useful if it would define a word as a string of characters delimited by any given set of characters. One use of this that comes to mind is to divide a file into sentences by breaking it at each period.

BWords should be entered with Basic09, and packed. If you have RUNB you can run words with a command line like:

```
OS9: words <testfile
```

which will divide the text in testfile into words. If you don't have RUNB you might need to use a somewhat longer command line:

```
OS9: basic09 words <testfile
```

It is easy to spend a great deal of effort writing filters you will never use. What is needed is a set of general purpose tools. There are several sources for good ideas for filters. Books about UNIX often give descriptions of filters which are commonly used under UNIX. In general, if a concept is useful for UNIX it will also be for OS-9. The standard programming book, Software Tools, by Kernighan and Plauger, is an especially good source for ideas and algorithms.

A MORE ADVANCED APPROACH TO PIPES

The Shell uses pipes to connect strings of its children together. Any program that has access to OS-9 system calls can use the same trick the shell uses to make the standard output of one of its children feed directly into the standard input of another, but it is simpler to use pipes as a connection between a process and its parent. If you need a formatted list of processes (the information given by the procs command) you can either mess with the process descriptors yourself, or use a pipe to intercept the output from procs.

If your algorithm can be divided into several sections that communicate in only one direction (Say, one section collects information, the second sorts it, and the third formats a report.), the job can easily be done by three separate processes dispatched from the command line with the shell managing the pipes. If the steps aren't fixed (Perhaps you either report or update a file depending on the date.), it might be easier to deal with the pipes

yourself. This type of thing requires pipes to be defined for each new process's standard input path.

Using a pipe as the standard output path from a child process is useful for more than intercepting the output from system utilities. The first experiments to try with this mechanism are with system utilities, but the most interesting applications are with processes designed especially for this use. An example might be a program which uses a process attached via a pipe to get data from a remote computer. The process at the end of the pipe would dial the remote computer up, go through the logon formalities, and deal with any communication protocols. The main process would just read distilled information from the pipe.

All three standard paths can be used for pipes. I haven't thought of a use for all three paths, but a combination of input and output paths is useful. The child process is given work to do through its standard input path and returns the results of its work through its standard, or error, output path. The parent process gives the child work through one pipe and at an appropriate time (maybe much later) gets the results by reading from a different pipe.

A FORKed process inherits the three standard paths of its parent. If it were OK to give up after setting up pipes, the way to set them up would be to close the standard files, and create three pipes, one each for path one, two and three. The instructions to open a pipe in the standard input path would be:

```
Pipe fcs "/PIPE"
  lda #0 std in
  OS9 I$CLOSE
  leax Pipe,PCR
  lda #UPDATE.
  OS9 I$OPEN
```

New paths always take the lowest available path number so the pipe would fall into path zero. A process forked from this process would inherit its standard paths including the pipe in path zero. The new process would treat its path 0 as a normal standard input path. Characters written into the pipe by the parent would be read by the child.

If a pipe is opened with no process FORKed to use it, the pipe will act like a queue. A process can write a limited number of bytes into the pipe and read them out again in the same order they were written. If there isn't room in the queue for the data from a write to be stored the process doing the write will be put to sleep until there is space to complete the write. If the process that reads from the pipe is the same one that is sleeping until the queue empties a little there is a deadlock. A deadlock can only be avoided, or broken by some outside agency ... the human at the terminal for instance. Because of this deadlock problem, and the small size of the queue in the pipe, the idea of using a pipe as a queue is only a novelty.

The example of communications via pipes that I have invented is a BasicO9 program that prompts for pairs of coordinates, and passes the pairs to a C program which "rasterizes" the lines between the points defined by the coordinates. The BasicO9 program passes as many pairs as it likes to the C program, then closes the path it has been writing the data to. When the parent closes his end of the pipe the child will get an end-of-file. The C program sends the rasterized data back through its standard output path. This data consists of a string of zeros and ones indicating where dots should be placed on each horizontal line in order to draw the vectors received as input.

Rasterizing vector graphics information is a particularly good application for a separate process. In a Level Two system each process can use an entire address space of almost 64K. The size and resolution of the graph that is produced depends on the amount of memory available for the bit map of the graph. I have a version of rast that uses 46K for its bit map and can generate an 8"X8" graph on my Okidata at 72 dots per inch. I am not very experienced with graphics; there is probably a much better way to rasterize data than what I used. My program seems too complicated for such a simple task, but it works.

It is particularly important to keep track of interactions between two processes communicating via pipes. If the processes ever get into a situation where both are waiting for input from a pipe leading to the other process, they will be stuck until you free them by killing one of the processes.

The important part of this system of programs is an assembly language subroutine for the BasicO9 program. The subroutine is descended from the StrtTask subroutine I published months ago, but has been enhanced to open pipes to the new process. The I$DUP call is used to preserve the standard input and output files of the BasicO9 program while paths zero and one are turned into paths then back into whatever they were before.

Installation

This system of programs is written in three separate languages. If you don't have C it should be fairly easy to translate rast into BasicO9, but if you rewrite rast in BasicO9 be certain that you don't try to fork it directly. BasicO9 should be the program you fork; rast should be the parameter. If you want to keep the old StrtTask around, rename either it or the new one. Grapher should be typed into BasicO9 and saved. Particularly if you are using Level One, you should pack Grapher and use RunB to save memory. In Summary:

- Enter StrtTask and rast.c using an editor
- Assemble StrtTask
- Compile rast.c
- Enter Grapher using BasicO9
- Save the source

- If you intend to run Grapher from the command line add the line: BYE to the end of Grapher and Pack Grapher

- Run Grapher which will load StrtTask and rast from the execution directory

Operation and Modification

Grapher will prompt for pairs of coordinates. After each pair is entered it will ask you to verify that you want to plot that line. Be careful with this. There is no validation in any part of this system. There is no reason it shouldn't be there either. Please add enough error checking to make you comfortable if you intend to do more than play with this program a little. If you try to draw a line way off into the wild blue yonder your computer will give it a good try, mashing everything in its way. After you enter the last pair of coordinates respond to the (y,n,d) prompt with D. The D response sends the last pair to rast and charts the response from rast on the screen. I like to draw conservative patterns like the one given by the input in figure Figure 5.

```
0 0 79 23
0 23 79 0
0 0 0 23
0 0 79 0
0 11 79 11
39 0 39 23
```

Figure 5: Sample Input for rast program

Rast is set up to rasterize a 80 by 24 graph. That is the size of a standard terminal, but if you want to deal with larger or smaller graphs, change VDIMENSION to the number of vertical dots in the graph, and HDIMENSION to the number of horizontal dots.

Pipes are a powerful tool for interprocess communications. They can be used with good effect to solve almost any interprocess communication problem if the connection can be made. The worst problem with pipes is that they can only be used between processes that are very closely related (between siblings, or parent/child). There is also a performance problem under Level Two; not only is there the cost of a system request per transfer, but OS-9 has to move the characters from one address space to another -- taking a surprising length of time. If you feel ambitious you will find that it is possible to make a major performance improvement to rast by using a compression algorithm on its output.

WELCOME COCO

I have been reading messages in the COCO special interest group on Compuserve. It sounds like Microware put a real version of OS-9 on that little machine. I am seriously impressed with the reality of a very inexpensive computer with a UNIX-like, multitasking, even -- if I may stretch a point -- multi-user operating system. There may be a number of interesting ways to integrate COCOs with each other and with larger OS-9 systems to get a bargain version of advanced distributed computing. It may not be too much to hope for that Tandy will find a way to put OS-9 Level Two on some descendant of the COCO. There is some chance that I will be able to take the viewpoint of a COCO user in this column in the future. I haven't made up my mind yet, but I need a Level One system, and the Color Computer may be the way to get one. I would appreciate advice.

THE USERS GROUP

The executive committee of the OS-9 Users Group has met twice since the annual meeting (I am writing this in November). We have struggled with various issues and defined assorted policies, mostly rather dull. Very likely by the time this column is printed the members will have received a newsletter, and everyone will have seen information in this and other magazines. Right now our software library is ready to go. I know it has good stuff in it; several programs of mine are part of the collection. Our plan is to give a standard selection of software from the library to the existing membership and to each new member. The other programs in the library will be available for small amounts of money, or software contributions. The address of the Users Group is:

OS-9 Users Group
PO Box 8027
Des Moines, Iowa
50301

BWORD

```
PROCEDURE bword
 0000       (* ----------------------------------------------*)
 0036       (* Filter to divide input into words. One word per  *)
 006C       (* line.                                           *)
 00A2       (* ----------------------------------------------*)
 00D8       DIM chr:BYTE
 00DF       DIM inword:BOOLEAN
 00E6       DIM StdOut,StdIn,StdErr:INTEGER
 00F5       ON ERROR GOTO 100
 00FB       StdIn=0
 0102       StdOut=1
 0109       StdErr=2
 0110       inword=FALSE
 0116       LOOP
 0118         GET #StdIn,chr
 0122         IF inword THEN
 012B           IF chr=ASC(" ") OR chr=9 OR chr=13 THEN
 0147             inword=FALSE
 014D             WRITE #StdOut
 0153           ELSE
 0157             PRINT #StdOut,CHR$(chr);
 0163           ENDIF
 0165         ELSE
 0169           IF chr=ASC(" ") OR chr=9 OR chr=13 THEN
 0185           ELSE
 0189             inword=TRUE
 018F             PRINT #StdOut,CHR$(chr);
 019B           ENDIF
 019D         ENDIF
 019F       ENDLOOP
 01A3       BYE
 01A5 100   REM end of file handler
 01BE       DIM errnum:INTEGER
 01C5       errnum=ERR
 01CB       IF errnum=211 THEN
 01D7         BYE
 01D9       ELSE
 01DD         ON ERROR
 01E0         PRINT #StdErr,"Error Number: "; errnum
 01FB         BYE
 01FD       ENDIF
```

```
Microware OS-9 Assembler 2.1   11/08/83 22:55:20                    Page 001
CharCt - Count a occurances of a specified character

00001                           NAM   CharCt
00002                           TTL   Count a occurances of a specified character
00003          *------------------------------------------------*
00004          *    CharCt              Written  1 November 83    *
00005          *                   Last Modified  5 November 83    *
00006          *    A filter to count occurances of any specified*
00007          *    character in the standard input.  If no      *
00008          *    character is specified, default to counting  *
00009          *    carriage returns.                            *
00010          *------------------------------------------------*
00011                           IFP1
00013                           ENDC
00014    0011             Type     set   Prgrm+Objct
00015    0081             Revs     set   ReEnt+1
00016    0000 87CD009A             MOD   pgmlen,CharCt,Type,Revs,Entry,Memsize
00017 D  0000             Count    rmb   3           stored in BCD
00018 D  0003             InChr    rmb   1
00019 D  0004             TstChr   rmb   1
00020 D  0005             OutStr   rmb   6
00021 D  000B             CR       rmb   1           for a CR
00022 D  000C                      rmb   200         Stack
00023 D  00D4             Memsize  equ   .
00024    000D 43686172    CharCt   fcs   /CharCt/
00025    0013 01                   fcb   1           version
00026          *------------------------------------------------*
00027          *  At entry:                                     *
00028          *    U and DP point at local storage.            *
00029          *    X points at the parameter area.             *
00030          *------------------------------------------------*
00031    0014             Entry
00032    0014 0F00                 clr   Count
00033    0016 0F01                 clr   Count+1
00034    0018 0F02                 clr   Count+2
00035    001A A684                 lda   ,X
00036    001C 9704                 sta   TstChr
00037    001E 3043                 leax  InChr,U
00038    0020 108E0001             ldy   #1          characters to read
00039    0024             Loop
00040    0024 8600                 lda   #0          std in
00041    0026 103F89               OS9   I$READ
00042    0029 251E                 bcs   Quit
00043    002B D603                 ldb   InChr
00044    002D D104                 cmpb  TstChr
00045    002F 26F3                 bne   Loop
00046          *------------------------------------------------*
00047          *              Increment Count                   *
00048          *------------------------------------------------*
00049    0031 8601                 lda   #1
00050    0033 9B02                 adda  Count+2
00051    0035 19                   daa
00052    0036 9702                 sta   Count+2
00053    0038 8600                 lda   #0
00054    003A 9901                 adca  Count+1
00055    003C 19                   daa
00056    003D 9701                 sta   Count+1
```

```
00057   003F 8600              lda   #0
00058   0041 9900              adca  Count
00059   0043 19                daa
00060   0044 9700              sta   Count
00061   0046 4F                clra                std in
00062   0047 20DB              bra   Loop
00063                *-------------------------------------------------*
00064                *    If we reached EOF print the total count and  *
00065                *    exit.                                        *
00066                *    If some other caused us to stop. Return      *
00067                *    with an error code.                          *
00068                *-------------------------------------------------*
00069   0049         Quit
00070   0049 C1D3              cmpb  #E$EOF
00071   004B 2636              bne   Exit
00072   004D 3045              leax  OutStr,U
00073   004F 9600              lda   Count
00074   0051 8D33              bsr   Cnvt
00075   0053 9601              lda   Count+1
00076   0055 8D2F              bsr   Cnvt
00077   0057 9602              lda   Count+2
00078   0059 8D2B              bsr   Cnvt
00079   005B 3045              leax  OutStr,U
00080   005D 960A              lda   OutStr+5      mark last position in OutStr
00081   005F 8A80              ora   #$80          set carry bit
00082   0061 970A              sta   OutStr+5
00083   0063 108E0007          ldy   #7            length
00084   0067 8630              lda   #'0
00085   0069         FndLen
00086   0069 A184              cmpa  ,X
00087   006B 2606              bne   OutPut
00088   006D 313F              leay  -1,Y          decrease length
00089   006F 3001              leax  1,X
00090   0071 20F6              bra   FndLen
00091   0073         OutPut
00092   0073 860D              lda   #$0D          <CR>
00093   0075 970B              sta   CR
00094   0077 960A              lda   OutStr+5
00095   0079 847F              anda  #$7F          clear the carry bit out
00096   007B 970A              sta   OutStr+5
00097
00098   007D 8601              lda   #1            std out
00099   007F 103F8C            OS9   I$WRITLN
00100   0082 5F                clrb                clean up for exit
00101   0083         Exit
00102   0083 103F06            OS9   F$Exit
00103   0086         Cnvt
00104   0086 1F89              tfr   A,B
00105   0088 44                lsra                shift the high order nible into low
00106   0089 44                lsra
00107   008A 44                lsra
00108   008B 44                lsra
00109   008C 8B30              adda  #'0           convert to ASCII digit
00110   008E A780              sta   ,X+
00111   0090 C40F              andb  #$0F          remove high order nyble
00112   0092 CB30              addb  #'0           convert to ASCII digit
00113   0094 E780              stb   ,X+
00114   0096 39                rts
00115   0097 A1D953            EMOD
00116   009A         Pgmlen    equ   *
```

```
PROCEDURE Grapher
 0000       DIM process_No,Comp_Code,Opt_Size,Lang_Type:BYTE
 0013       DIM Parm_L:INTEGER
 001A       DIM InPipe,OutPipe:BYTE
 0025       DIM ch:STRING[1]
 0031       DIM YN:STRING
 0038       DIM x1,y1,x2,y2:INTEGER
 004B       DIM name:STRING
 0052       DIM Parms:STRING[20]
 005E       (* ------------------------------------------------------*)
 009C       (* Set up to call StrtTask which will fork the named     *)
 00DA       (* module, passing it the parameter string in Parms.     *)
 0118       (* ------------------------------------------------------*)
 0156       name="rast"
 0161       process_No=0
 0168       Opt_Size=0
 016F       Lang_Type=$11 \(* attributes of forked module (object code, program)
 01AC       Parms=""+CHR$(13)
 01B7       Parm_L=LEN(Parms) \(* The length of the parameters must be correct
 01EF       (* ------------------------------------------------------*)
 0230       (* Call assembler subroutines to Fork and wait for the started *)
 0271       (* process                                              *)
 02B2       (* ------------------------------------------------------*)
 02F3       RUN StrtTask(name,process_No,Lang_Type,Parm_L,Parms,Opt_Size
                                                          ,InPipe,OutPipe)
 0320       (* ------------------------------------------------------*)
 0361       (* Write data for "rast" into path #InPipe which         *)
 03A2       (* corresponds to the standard input path for rast       *)
 03E3       (* ------------------------------------------------------*)
 0424       PRINT "Enter the endpoints of lines you want drawn. X must be in"
 0461       PRINT "the range 0..79. Y must be in the range 0..23."
 0493       LOOP
 0495         INPUT "Enter X Y coordinates for the ends of a line: ",x1
                ,y1,x2,y2
 04D7         PRINT "The line will be drawn between ("; x1; ","; y1; ") and ("
                                                        ; x2; ","; y2; ")"
 0521         INPUT "OK ?  (Yes,No,Done): ",YN
 053E         YN=LEFT$(YN,1)
 0549         IF YN="y" OR YN="Y" THEN
 055E           PRINT #InPipe,"1",x1,y1,x2,y2
 0578         ENDIF
 057A       EXITIF YN="d" OR YN="D" THEN
 058F         PRINT #InPipe,"1",x1,y1,x2,y2
 05A9       ENDEXIT
 05AD       ENDLOOP
 05B1       ON ERROR GOTO 100
 05B7       CLOSE #InPipe
 05BD       (* ------------------------------------------------------*)
 05FE       (* When #InPipe is closed rast will get an end-of-file  *)
 063F       (* on its standard input path.                          *)
 0680       (* ------------------------------------------------------*)
 06C1       LOOP
 06C3         (* ----------------------------------------------------*)
 0704         (* Read from #OutPipe (which corresponds to rast's standard  *)
 0745         (* output until end-of-file on that path.  The end-of-file   *)
 0786         (* indicates that the other end of the pipe has been closed  *)
 07C7         (* (in this case rast has ended).                       *)
 0808         (* ----------------------------------------------------*)
 0849         GET #OutPipe,ch
 0853         IF ch="0" THEN
 0860           PRINT " ";
 0866         ELSE
 086A           PUT #1,ch
 0873         ENDIF
 0875       ENDLOOP
 0879 100
 087D       ON ERROR
 0880       CLOSE #OutPipe
 0886       RUN WaitTask(process_No,Comp_Code)
 0895       IF Comp_Code<>0 THEN
 08A1         PRINT "Completion code for  "; name; " # "; process_No; " was "
                                                          ; Comp_Code
 08D4       ENDIF
```

```
00001                           ttl   Start a subtask (called from Basic09)
00002                           nam   StrtTask
00003          *-------------------------------------------------*
00004          *  StrtTask is a subroutine for Basic09.          *
00005          *  Start a named module as a subtask.             *
00006          *    Let the new task run asynchronously.         *
00007          *    Open pipes to the modules standard in and standard*
00008          *    out paths.                                   *
00009          *    Return the new tasks process number, the path*
00010          *    numbers for the pipes, and the condition code*
00011          *    from the Fork.                               *
00012          *   Calling sequence:                             *
00013          *   run StrtTask (Name, Process_Num, Lang_Type,   *
00014          *                 Param_L, Param, Opt_size        *
00015          *                 InPipeN, OutPipeN)             *
00016          *   Name is any length, but has a valid terminator*
00017          *    (high bit set on last byte, or delimiter after it)*
00018          *                                                 *
00019          *   Process_Num byte field, process number of new task.*
00020          *   Lang_Type byte field, language/type byte for  *
00021          *    forked module.                               *
00022          *   Param_L, integer field, length of parameter area.*
00023          *   Param field of any type, parameter area to be *
00024          *    passed to forked process.                    *
00025          *   Opt_Size byte field, optional data area size in*
00026          *    pages.                                       *
00027          *   InPipeN, integer field, path number           *
00028          *   OutPipeN, integer field, path number          *
00029          *   Process_Num, InPipeN, OutPipeN, and Return_Code*
00030          *   are altered by StrtTask, no other parameters are.*
00031          *-------------------------------------------------*
00032                           IFP1
00034                           ENDC
00035          ******
00036          *  Offsets to arguments
00037          *
00038  0002              ACount   equ   2
00039  0004              ModuleN  equ   4
00040  0008              ProcNum  equ   8
00041  000C              ModType  equ   12
00042  0010              ParmLen  equ   16
00043  0014              Parms    equ   20
00044  0018              MDatSize equ   24
00045  001C              InPipeN  equ   28
00046  0020              OutPipeN equ   32
00048  0021              Type     set   SBRTN+OBJCT
00049  0081              Revs     set   REENT+1
00050  0000              StdIn    equ   0
00051  0001              StdOut   equ   1
00052  0000 87CD00B1             mod   TLen,StrtTask,Type,Revs,SEntry,0
00053  000D 53747274     StrtTask fcs   /StrtTask/
00054  0015 2F504950     Pipe     fcs   "/PIPE"
00055  001A 01                    fcb   1           version
00056  001B              SEntry
00057  001B EC62                  ldd   ACount,S    get param count
00058  001D 10830008              cmpd  #8          are there 8 params?
00059  0021 10260083              lbne  BadExit     no; leave now.
00060          *-------------------------------------------------*
00061          *           Set up Pipes for StdIn and StdOut.    *
00062          *    The procedure is:                            *
00063          *         Dup the stdin and stdout paths to save them.*
00064          *         Close stdin and stdout.                 *
00065          *         Open /PIPE twice. One will be path 0 the next*
00066          *              path 1.                            *
00067          *         Fork the new process.                   *
00068          *-------------------------------------------------*
00069          *  Offsets from S for local storage
00070  0000              DStdIn   equ   0
00071  0001              DStdOut  equ   1
00072  0002              LocalSiz equ   2
00073          *
```

```
00074   0025 327E               leas  -LocalSiz,S make space for temp storage
00075   0027 8600               lda   #StdIn
00076   0029 103F82             OS9   I$Dup       Dup Stdin
00077   002C 257D               bcs   BadExit2
00078   002E A7E4               sta   DStdIn,S
00079   0030 8601               lda   #StdOut
00080   0032 103F82             OS9   I$Dup       Dup StdOut
00081   0035 2574               bcs   BadExit2
00082   0037 A761               sta   DStdOut,S
00083
00084   0039 8600               lda   #StdIn
00085   003B 103F8F             OS9   I$Close     Close StdIn
00086   003E 256B               bcs   BadExit2
00087   0040 8601               lda   #StdOut
00088   0042 103F8F             OS9   I$Close     Close StdOut
00089   0045 2564               bcs   BadExit2
00090
00091   0047 308DFFCA           leax  Pipe,PCR
00092   004B 8603               lda   #UPDAT.
00093   004D 103F84             OS9   I$Open      Open a pipe in path 0
00094   0050 2559               bcs   BadExit2
00095                *  This will be path 0
00096
00097   0052 308DFFBF           leax  Pipe,PCR
00098   0056 8603               lda   #UPDAT.
00099   0058 103F84             OS9   I$Open      Open a pipe in path 1
00100   005B 254E               bcs   BadExit2
00101                *  This will be path 1
00102   005D 103F82             OS9   I$Dup       Dup it
00103   0060 2549               bcs   BadExit2
00104   0062 A7F822             sta   [LocalSiz+OutPipeN,S]
00105
00106   0065 8600               lda   #StdIn
00107   0067 103F82             OS9   I$Dup       Dup it
00108   006A 253F               bcs   BadExit2
00109   006C A7F81E             sta   [LocalSiz+InPipeN,S]
00110
00111   006F AE66               ldx   LocalSiz+ModuleN,S address of module name
00112   0071 10AEF812           ldy   [LocalSiz+ParmLen,S] length of parameters
00113   0075 A6F80E             lda   [LocalSiz+ModType,S] type of module to invoke
00114   0078 E6F81A             ldb   [LocalSiz+MDatSize,S] optional data area size
00115   007B EEE816             ldu   LocalSiz+Parms,S  pointer to parameters
00116   007E 103F03             OS9   F$Fork      start the new process
00117   0081 2528               bcs   BadExit2
00118   0083 A7F80A             sta   [LocalSiz+ProcNum,S] save new process number
00119                *-------------------------------------------------------*
00120                *    Restore the original stdin and stdout files to     *
00121                *    paths 0 and 1.                                     *
00122                *-------------------------------------------------------*
00123   0086 8600               lda   #StdIn      Close StdIn and StdOut
00124   0088 103F8F             OS9   I$Close
00125   008B 8601               lda   #StdOut
00126   008D 103F8F             OS9   I$Close
00127   0090 A6E4               lda   DStdIn,S    path number of duped stdin
00128   0092 103F82             OS9   I$Dup       dup it into path 0
00129   0095 A6E4               lda   DStdIn,S
00130   0097 103F8F             OS9   I$Close     and close it
00131   009A A661               lda   DStdOut,S   path number of duped stdout
00132   009C 103F82             OS9   I$Dup       dup it into path 1
00133   009F A661               lda   DStdOut,S
00134   00A1 103F8F             OS9   I$Close     and close it
00135   00A4 3262               leas  LocalSiz,S clear stack
00136   00A6 5F                 clrb              clear carry
00137   00A7 39                 rts               return
```

```
00138   00A8             BadExit
00139   00A8 43                  coma              set carry
00140   00A9 327E                leas  -LocalSiz,S dummy push
00141   00AB             BadExit2
00142   00AB 3262                leas  LocalSiz,S clear stack
00143   00AD 39                  rts               return
00144   00AE 239951              EMOD
00145   00B1             TLen    equ   *
00146                            ttl   Wait for a (child) process to complete
00147                            nam   WaitTask
00148        *-------------------------------------------------------*
00149        *   WaitTask is a subroutine for Basic09                *
00150        *   Wait for the a child process to complete.           *
00151        *   Return the process ID of the process that completed*
00152        *   in parameter one.                                   *
00153        *   Return the competion code of the process            *
00154        *   in parameter two.                                   *
00155        *   This subroutine will wait using no CPU time until   *
00156        *   a child process completes.                          *
00157        *   If a child completed just before WaitTask was       *
00158        *   called, it will return almost immediatly.           *
00159        *   If there are no children, an error will be returned*
00160        *   with a process number of 0.                         *
00161        *   Calling sequence:                                   *
00162        *    RUN WaitTask (Process_No, Comp_Code)               *
00163        *   both process_no and Comp_Code are BYTE variables.   *
00164        *-------------------------------------------------------*
00165   0021             Type    set   SBRTN+OBJCT
00166   0081             Revs    set   REENT+1
00167   0000 87CD0032            mod   WLen,WaitTask,Type,Revs,WEntry,0
00168   000D 57616974    WaitTask fcs   /WaitTask/
00169   0015 01                  fcb   1           edition
00170   0016             WEntry
00171   0016 6FF804              clr   [4,S]       zero the process ID
00172   0019 EC62                ldd   2,S         param count
00173   001B 10830002            cmpd  #2          if not exactly 2 params then
00174   001F 260C                bne   WBExit2     the caller is making a bad mistake
00175   0021 103F04              OS9   F$Wait      wait for a child
00176   0024 2508                bcs   WBExit
00177   0026 A7F804              sta   [4,S]       return the process ID
00178   0029 E7F808              stb   [8,S]       return the completion code
00179   002C 39                  rts               return
00180   002D             WBExit2
00181   002D 43                  coma              set carry
00182   002E             WBExit
00183   002E 39                  rts               return
00184   002F 4C34C4              EMOD
00185   0032             WLen    equ   *
00186                            end

00000 error(s)
00000 warning(s)
$00E3 00227 program bytes generated
$0000 00000 data bytes allocated
$218B 08587 bytes used for symbols
```

```
#include <stdio.h>
#define VDIMENSION 24
#define HDIMENSION 80
#define BYTES_HDIMENSION/8
#define TRUE 1
#define FALSE 0
/*-------------------------------------------------------*
 *                  Data Structure                       *
 *  The rasterized data is kept in an array of bits.     *
 *  The Setbit and BitSet routines are responsible for   *
 *  determining which bit correspondes to each           *
 *  position.  They also are the only procedures with    *
 *  access to the "bit" array.                           *
 *-------------------------------------------------------*/
main()
{
int x1, y1, x2, y2;
int i;
char op;  /* takes values of  L  Line  (n,n,n,n)
                               C  Circle (n,n,0,0)
                               S  Spline (open) (n,n,n,n,n,n)
                               E  Spline (closed) (n,n,n,n,n,n)
          */
register int j;

while (scanf("%c %d %d %d %d",&op, &x1,&y1, &x2,&y2) != EOF)
/*  Ignore "op" for now */
   if (x1 < x2)
      draw(x1,x2,y1,y2);
   else
      draw(x2,x1,y2,y1);

for (i=VDIMENSION-1;i>=0;i--)
   {
      for (j=0;j<HDIMENSION;j++)
         putchar(bitset(j,i) ? '1' : '0');
      printf("\n");
   }
return;
} /* end of main */

draw(x1,x2,y1,y2)
   int x1, x2, y1, y2;
   {
      int deltay, deltax, x, y, dy, dx;
      float e, slope;
      register int i;

      deltay = y2-y1;
      deltax = x2-x1;
      x = x1;
      y = y1;
      if ((deltax == deltay) &&  (deltay == 0))
         { /* special case -- draw a point */
            plot(x,y);
            return;
         }

      if (deltax > deltay)
         {
            if (deltax == 0)
               { /* prevent division by zero */
                  y = (y1 <= y2) ? y1 : y2;
                  for (i=0;i<=((deltay >= 0) ? deltay : -deltay);i++)
                     plot(x,y++);
                  return;
               }
            slope = (float)deltay/(float)deltax;
            if (slope >= 0)
               {
                  e = slope-0.5;
                  dy = 1;
               }
            else
               {
                  e = slope+0.5;
                  dy = -1;
               }
```

```
79              for (i=0; i<=deltax; i++)
80                  { /* actually draw the line */
81                     plot(x,y);
82                     if (((slope >  0.0) &&  (e>0.0)) ||
83                         ((slope < 0.0 ) &&  (e<0.0)))
84                        {
85                           y += dy;
86                           e -= dy;
87                        }
88                     x++;
89                     e += slope;
90                  } /* actually draw the line */
91           }
92        else
93           {
94              slope = (float)deltax/(float)deltay;
95              if (slope > 0)
96                 {
97                    e = slope-0.5;
98                    dx = 1;
99                 }
100             else
101                {
102                   e = slope+0.5;
103                   dx = -1;
104                }
105             for (i=0; i<=deltay; i++)
106                { /*----------------------------------------*
107                   * draw a line with slope greater than one *
108                   * for this type of line y needs to be     *
109                   * incremented more frequently than x.     *
110                   *----------------------------------------*/
111                   plot(x,y);
112                   if (((slope >  0) &&  (e>0)) || ((slope < 0 ) && (e<0)))
113                      {
114                         x += dx;
115                         e -= dx;
116                      }
117                   y++;
118                   e += slope;
119                }
120          }
121       return;
122    } /* end of draw */
123
124 plot(x,y)
125    int x,y;
126    {
127       setbit(x,y);
128       return;
129    }
131 static char bit[VDIMENSION][BYTES];
132
133 setbit(x,y)
134    int x,y;
135    {
136       int temp=1;
137       register int tx;
138
139       temp = temp << (x%8);
140       tx = x/8;
141       bit[y][tx] = bit[y][tx] | temp;
142       return;
143    }
144
145 bitset(x,y)
146    int x,y;
147    {
148       int temp=1;
149
150       temp = temp << (x%8);
151       return(bit[y][x/8] & temp);
152    }
153
```

COLUMN ELEVEN -- THE OS-9 I/O SYSTEM

OS-9 uses a modular I/O system designed for simplicity and flexibility. Because of this modularity an exceptionally ambitious user could write a new I/O subsystem and graft it into OS-9 without making any changes to the rest of the operating system. But there are other aspects of the I/O system which don't require any programming to exploit, and so useful that new OS-9 users should play with them as soon as possible.

THE UNIFIED INPUT/OUTPUT SYSTEM

Each OS-9 process has three standard paths (files) open when it starts. Path 0 is called standard input, Path 1 is standard output, and path 2 is standard error. It is possible for a program to close these paths and re-open them for its own purposes, but most programs leave them open and use them as one might think they should be used.

The standard input path usually reads from the keyboard (terminal), and is used as the primary source of input from the user. Programs can and often do open other input files, sometimes the majority of the input is from some path other than standard input, but standard input is by convention the path used for communication with the user.

The standard output path typically writes to the screen (terminal), and is used for routine output to the user. Every character that appears on your screen probably came from a standard output path.

The standard error path is seldom used. By convention it is used for error messages. Normally the standard error path is directed to the screen together with the standard output path. The rationale for having separate paths for routine output and error messages rises from a special characteristic of the standard paths. Each of the paths can be directed wherever the user wishes before a program is started. This can prove useful when it is convenient to have different things done with error messages than with the rest of the output of a program.

The standard paths are open when a program starts because they are inherited from the process that started it, in most cases the shell. The shell takes advantage of this ability to pass its standard paths on to the programs it starts to change the paths from the standard (all to the terminal) to any other disposition a user might specify.

Options on a shell command line indicate to the shell what needs to be done to the standard paths. The options are ">xxxxx" for "redirect standard output to xxxxx," "<xxxxx" for "redirect standard input to xxxxx," and ">>xxxxx" for "redirect standard error to xxxxx." If any standard path is not redirected it is simply inherited from the shell; it usually goes to the terminal.

The ability to redirect the standard paths is called device independent I/O because paths can be directed to any device, not just another device of the same type as the default device for the path. The power of this feature is easiest to see with a few examples:

```
OS9: list filename
```

Is a command with no redirection. It lists the contents of the file called "filename" on the screen through the standard output path.

```
OS9: list filename >/P
```

lists the contents of filename on the device called /P, usually the printer. The single ">" at the end of the command tells the shell to redirect the standard output to the file whose name follows the >. I can't think of any reason for someone to want to put the output of the list command into a disk file, but:

```
OS9: list filename >lstfile
```

does just that. It puts the output of the list command into a file named lstfile. If you are using a multi-user system you can send the output of a command to another user with a command like:

```
OS9: asm test.a 1 >/T2
```

which would send the listing from the assembly of test.a to the device called /T2, which is usually a terminal.

I redirect Standard Output more than the other paths, but there are reasons to redirect the other paths as well. The Standard Input path is the one which programs usually read from. A program can be fed a canned script of commands by redirecting its Standard Input to a disk file with the commands in it. I sometimes insert this command in my startup file:

```
debug <startup.debug >/NL
```

This runs the Microware debugger with its input coming from startup.debug, and its output going to a special SCF device which I made public in the first column I wrote (/NL is a null device -- it makes anything you send to it disappear). By putting debug in my startup file like this I can easily apply patches to resident modules every time I boot my system.

The Standard Error path is used so infrequently that it is easy to forget that it exists. It is the path which programs usually use for serious error messages. Usually, it is a good idea to leave the Standard Error path directed to the screen, but sometimes it should be redirected. Some compilers send syntax errors, or at least summary statistics out the Standard Error path. If you want to run a program that uses the Standard Error path in background while you edit in foreground, it is wise to redirect the both the Standard Output and the Standard Error paths of the compiler to disk files or the printer, otherwise you may find messages from the compiler cropping up in the middle of your screen at awkward times.

Redirection almost always works fine, but there are some problems lurking around. It shouldn't be the responsibility of a user to watch out for these problems, but OS-9 is designed with the assumption that programs will follow some conventions applying to their use of the standard paths. Some programs rely on dealing with particular devices. These programs should open special paths to those devices, but some use the standard paths for device dependent I/O. These programs should be avoided if possible.

The typical OS-9 system comes with three types of files, Sequential Character Files, Random Block Files, and Pipes. Sequential Character Files (usually called SCF files) are written or read from beginning to end. The most common SCF files are Terminal input and output, printer output, and modem input and output. The bytes in a RBF file (files handled by the RBFMAN file manager) can be read in any order. Disk files and other files like them, such as files in bubble memory or main memory, are usually RBF files. There is only one type of Pipe file, that is a temporary file kept in main memory which is used a buffer between one program's output and another program's input.

Unless a program concerns itself with timing issues or uses the more exotic GETSTAT/SETSTAT system service requests, there is no way for it to tell the difference between one device and another provided the devices are of the same type (RBF, SCF, or Pipe). Some programs can't have their standard I/O redirected to a RBF file or a Pipe, but the great majority can. If a program uses SCF-specific GETSTAT/SETSTAT codes it will only be possible to use it with the proper type of files, but all but one of the programs that I know of from Microware and other major vendors can have their I/O redirected without restriction. The one exception is Microware's Pascal with old versions of OS-9. All programs written in that language, including the compiler itself, try to rewind their standard output file when it starts. The SCF file manager deals with this strange request correctly by ignoring it, but the Pipe manager returns an error if anyone tries to rewind it. If you try to redirect the output of a program written in Pascal to a Pipe, the program will die as soon as it's started. Microware has a fix for this problem if you run into it.

CHANGING OS-9'S DEVICE SUPPORT

The modular design of OS-9's I/O system allows new devices to be added and the support of old devices to be enhanced with the only restrictions being the wishes and budget of the person responsible, and the memory constraints of the computer. Support for I/O starts at the IOMAN module which fields each I/O system service request and sometimes does a little work before passing it off to the appropriate module. File managers including SCF and RBF are the next level down from IOMAN; they do most of the file handling work that isn't specific to a particular piece of hardware. The device drivers, such as ACIA and PIA, handle the interface with the I/O hardware. The device descriptor modules contain the directions which all these modules follow. There is a descriptor for each device in an OS-9 system containing no executable instructions, but lots of data which controls the other I/O modules.

Hardware that requires complicated new modules for the I/O system should come with the necessary modules. The hardware vendor has to have the modules written (or write them), but a customer need only load the modules -- normally by including them in his boot -- in order to add software support for the device to his system. This sets OS-9 apart from many operating systems in which a major part of the operating system has to be changed for any new device.

Hardware vendors often need to write I/O modules in order to sell their products to the OS-9 community, but anyone can write I/O modules if the need or the mood takes them. Writing an entire new I/O subsystem would require a lot of work, but most problems can be solved with much less effort. Many devices can be accommodated by OS-9 without any serious programming at all by creating new device descriptors. Device descriptor modules specify how each device is to be treated. The device descriptor contains fields which indicate (to IOMAN) which file manager and device driver should be used for the device, an absolute physical address for the device, and any other data specific to the particular device

The first 18 bytes of all device descriptors have the same format. The first nine bytes are common to all module headers (Sync Bytes, Module size, Offset to Module Name, Type/Language ($F1), Attributes/Revision, and Header Parity check). Of these, the module attributes are most interesting in the context of the device descriptor. If the device descriptor module is marked reentrant, the device can be used by more than one process at a time; otherwise, it can only be linked to or opened by one process at a time. Device descriptors which are not reentrant are not only restricted to use by only one process at a time, they can't be linked to by debug at all if they are in the boot. Some devices, such as the printer, shouldn't be reentrant unless you feel very ready to be responsible. OS-9 will happily mix output from several programs line by line on the printer if you tell it to.

The format of the next nine bytes is common to all device descriptors. The fields are: the offset to the File Manager name (e.g., RBF) for two bytes, the offset to the Device Driver name (e.g., ACIA) for two bytes , the mode (what the device can do, e.g. Read/Write/execute) for one byte, the device controller's real address for three bytes, and the length of the initialization table.

After the first 18 bytes, different types of devices have different fields. The initialization table which follows the byte with its length contains most of the fields that are interesting to play with. After the initialization table there is nothing but module names and the CRC.

There are eleven fields in the initialization table for RBF-type devices (disk drives). The first field is one byte long

and contains a 1 indicating that this is a RBF device. The other fields are:

- drive number
- step rate
- device type
- media density (0=single,1=double)
- number of cylinders (two bytes long)
- number of surfaces, verify (0=verify writes)
- default sectors per track for two bytes
- default sectors per track on track zero for two bytes
- sector interleave factor
- segment allocation size

The step rate can take on values of 0..3 with the higher numbers reflecting higher stepping rates.

In the device type byte three bits are significant. Bit zero indicates a 8" floppy if it is one. Bit six indicates a non-standard format is being used if it is one. Bit seven being one indicates that the device is a hard disk.

In the media density byte two bits are significant. Bit zero = 1 indicates that the device can handle double density. Bit one = 1 indicates that the disk is capable of double track density (96 tpi).

The fields in the device descriptor are interpreted by the device driver and the file manager. Changing a value in the device descriptor can't force the other modules to do something they weren't written to do. For example, it probably isn't possible to use the device driver which is designed for floppy disks to control a hard disk -- changing the device type byte won't change the capabilities of the device driver. It is the option of the person writing the device driver to ignore anything in the device descriptor he wants. This means that there is no guarantee that the options in the device descriptor will work. I have heard that the floppy disk driver on the color computer ignores many of the options. I'll confirm this when I get one.

A different set of fields are in the initialization table for SCF devices. Most of these fields control the line-editing function of the SCF manager. These are the values that are temporarily set by TMODE. They can be set permanently by changing them in the device descriptor.

The initialization table in the device descriptor is copied into the path descriptor when a path is opened. There it can be changed and read by GETSTAT/SETSTAT calls, but the change applies only to that particular path. Changes to the device descriptor become the default for all paths opened to that device.

The easiest way to change the device descriptors is with debug. If, for example, you want to add a new terminal to your system which you don't have a device descriptor for, you can modify a similar descriptor with debug to fit your requirement (probably changing only the controller address and module name), save the result with the save command, and verify it with the update option to fix its CRC. The resulting module can be loaded and used.

A device descriptor can be modified even while the device it specifies is in use because the descriptor iqiself is seldom referenced. In fact, as far as I know, the device descriptor is only used when a path is opened to the device.

The device descriptor is the controlling part of the OS-9 I/O structure. There are several things that can be done with them that I haven't covered yet, but that will be material for other columns.

COLUMN TWELVE -- THE COCO

I now have a Radio Shack Color Computer with OS-9. I had hoped that this column would be about my first experiences as a new CoCo/OS-9 Level One user, but I have only had a few hours to play with the new machine and this column is due.

Even just a few hours with the CoCo version of OS-9 is enough to form some first impressions. First, that really is OS-9 in there. All the standard commands and utility programs are included. Even XMODE, which didn't come with my Level Two system, was on the CoCo OS-9 disk. I am impressed with the performance of the CoCo. I am used to a two megahertz GIMIX system, and the CoCo is distinctly slower than that; but, I bet Basic09 on a CoCo would give an IBM-PC running its version of Basic a good race. I hope I have a chance to do some benchmarks soon.

For a user moving from Color Basic to OS-9 the change must be wonderful, but confusing. OS-9 brings out much of the power hidden in that little off-white box. It also demonstrates the limitations of the Color Computer. After this column I intend to concentrate on positive aspects of the CoCo, but right up front I have to say that my new CoCo is a sit-down lawnmower with the soul of a Grand Pre racer. I want to get my complaining out of the way early, so this column is elected.

On the hardware side, I guess my complaints can be summarized as: this computer seems to have been designed to sell for under a thousand dollars. It is really unfair for me to think that this computer should have DMA (Direct Memory Access) for its disk I/O and a chip to do its serial I/O. By doing those tasks in software Radio Shack hurt OS-9's performance, but they also kept the cost of the computer down.

Certainly, my main reaction to the Radio Shack version of OS-9 was pleasure, but that didn't keep me from finding a few things to complain about. In my last column I hinted that the disk driver included with CoCo OS-9 doesn't adhere to OS-9 standards. I didn't make a strong statement because I didn't know from personal experience. I can tentatively confirm the information now -- the CCDisk disk driver doesn't seem to refer to the parameters set in the disk device descriptors.

The documentation that came with OS-9 was also a disappointment. I expected entirely new books explaining the trickier aspects of OS-9 so any fool could understand it. The manuals I got are just prettied-up versions of the Microware manuals with some parts missing. The documentation seems to have been very quickly done. I checked out the section on device descriptors first thing; the manual includes a full description of the device descriptor with no indication that some parameters don't work on the CoCo. Most of the information from Microware's manuals about adapting OS-9 to a new system are missing from Radio Shack's OS-9 documentation.

My complaints may sound significant, but they are not. The hardware limitations of the Color Computer are no worse than one would expect in a low-cost computer. The limited disk driver is only waiting to be replaced by a more general one. If no one else writes one, I may do it myself. The documentation problem is an invitation to people like me. If OS-9 on the CoCo continues to be as big a success as it has been, books will appear about it in fairly short order.

NOTES ON COMPUSERVE

I spent over two hours reading through the messages in the new OS-9 SIG on Compuserve. That bulletin board is really picking up! People are beginning to buy Basic09 for the CoCo and are having trouble installing it. Some messages went something like: I installed Basic09 on my system and it doesn't work -- HELP. I can't imagine how anyone is able to figure out what went wrong from that kind of complaint; I certainly couldn't. Several other people gave more detailed descriptions of their troubles. It sounded to me like they were having troubles with directories.

When you start OS-9 running it will find a directory called /D0/CMDS on your system disk. This is the directory OS-9 will always execute programs out of unless you explicitly direct it to another directory. Specifically, if you give the command

```
BASIC09
```

OS-9 will look for an executable file called BASIC09 in the /D0/CMDS directory. If it finds the program, everything is fine; otherwise, OS-9 will search the default data directory (initially /D0) for a file called BASIC09. If BASIC09 in found in the data directory it will be taken as a shell command file, and a shell will be started up to execute the commands. If that file turns out to be full of the machine code for Basic09, the shell will be understandably confused. If you copy Basic09 from its distribution disk to the root directory for your system disk (which is what the command:

```
copy /D1/basic09 Basic09
```

will do) your shell will get wrapped around the axle in about the way I just described. The way to avoid that problem is to put Basic09 in your execution directory with a command like:

```
copy /D1/basic09
/D0/CMDS/Basic09
```

The system disk on my CoCo is very full. If I had any number of my own programs on that disk it would overflow. When that happens it is time to divide the files on that disk between two disks. One way to split things up is to put Basic09 and a few other programs that are frequently used with Basic09 on a disk by themselves, and replace the system disk with the special Basic09 disk when it is time to use Basic. There is nothing wrong with the idea, but there is a nice pitfall waiting here too.

Directories are files, and, to save time, OS-9 remembers where the files you are using are on disk. When you boot OS-9 it determines where the directory /D0/CMDS is and will look right there next time it needs to find a program. If you pull out the system disk and put in your special Basic09 disk, OS-9 will read the location on the Basic09 disk where the /D0/CMDS directory was on the system disk. In the best case you will get a meaningful error, but you may not. The way to get around this problem is to remember to change your execution (and perhaps your data) directory when you change the disk it is on. That is:

```
Take the system disk out
Put the Basic09 disk in
type  CHX /D0/CMDS
```

which will cause OS-9 to find the /D0/CMDS directory again. Of course, if you decide to call the execution directory on your Basic disk something other than CMDS, that's fine; just change the execution directory appropriately. For example:

```
OS9: CHX /D0/BASIC.CMDS
```

If you put Basic09 on a disk separate from many of your other programs you may find yourself unable to get at some important program while you are using Basic09. There are at least three ways to solve this problem.

OS-9 lets you load programs into memory and keep them there. You don't want to load too many because main memory is a very limited resource, but sometimes it can prove very useful to have a program or two in memory. If you insert your Basic disk, oad /D0/CMDS/basic09 (note that I specified the full directory name instead of changing the execution directory -- either will work, but this way I won't need to change the directory back), then remove the Basic disk and put the system disk back in. Now Basic09 is in main memory. You can see Basic09 in the output of the MDIR command, and the MFREE command will show that there is much less free memory in the system than there was before you loaded Basic09. Now, if you type

```
OS9: basic09
```

you will find yourself in basic much faster than when it had to be loaded from disk. To get rid of the copy of Basic09 in main memory use the UNLINK command:

```
OS9: UNLINK basic09
```

If there is some small number of small programs you want to use from within Basic09 you can load them into memory while the system disk is mounted. For example:

```
OS9: LOAD copy
OS9: LOAD list

remove the system disk
insert the basic disk

OS9: CHX /D0/CMDS

and perhaps change the data
directory

OS9: CHD /D0/BASIC.PROGS

then start basic09

OS9: BASIC09
```

If, for one reason or another, neither of these tricks will serve, you can change the execution directory from within Basic09. For example, starting from a time when Basic09 is running with the basic disk on drive /D0:

Replace the basic disk with the disk with the programs you need

```
B:  chx /D0/CMDS      or whatever
```

do what needs to be done, then, before exiting from basic, replace the basic disk in the drive.

The Basic09 CHX command only changes the execution directory within Basic09 and any programs that are run from it. When you exit from Basic09 the directories that were active before you started Basic09 will be active again.

THANK YOU GIMIX

Ever since the CoCo version of OS-9 was announced with a different disk format from all other versions of OS-9 the users of large OS-9 systems have been grumbling about the incompatibility of our disk formats and the CoCo format. GIMIX has released a new floppy disk driver for their systems that supports reading and (if you have a 40 track drive) writing disks in CoCo OS-9 format. I am very grateful, and I am sure I represent many other OS-9 users when I thank GIMIX for their efforts.

A HANDY SHORTCUT

I always use 32K when I run Dynastar, and I almost always use 24K for the Microware Assembler. I am seldom content to use the minimum memory requirement given in the module header for any program. I have modified the module headers of several programs so they will automatically request the amount of memory I usually request for them. Debug can be used to do this. The commands which will modify Dynastar (DS) to default to its maximum memory size (32K) instead of the minimum (8K) are:

```
load ds
debug
1 ds
. .+b       To point at the permanent storage size in
              the module header.
              The value of this byte is $20
=7F
=FF
Q           The change is made so quit debug

Test ds to make certain the new default is working.
I first made certain I could edit a large file, then
invoked procs from within ds and noted that ds was
using 128 pages.
```

If you want to make the change permanent use the following sequence:

```
OS9: save /D0/x ds
OS9: verify U </D0/x
>/D0/CMDS/ds2
```

Check its attributes

```
OS9: attr /D0/CMDS/ds2
```

You will find that the execute and public execute attributes are missing, so turn them on

```
OS9: attr /D0/CMDS/ds2 e pw
```

Save the old version

```
OS9: rename /D0/CMDS/ds old.ds
```

Install the new one

```
OS9: rename /D0/CMDS/ds2 ds
```

COLUMN THIRTEEN

BIG SYSTEM HARDWARE

Gimix has offered CoCo owners an attractive deal. Gimix its value. Even with this roughly thousand dollar break in the price of a Gimix the upgrade is expensive, but, speaking as a person who has used a Gimix for many many hours, if you can find the money, take this opportunity. What makes it worth thousands of dollars to move from a CoCo to a SS50 system? The most important difference is that everything works right on the larger systems. Another is that the more expensive systems are faster. A two megahertz 6809 runs more than twice as fast as a CoCo in its normal mode. The DMA disk controller and other powerful I/O devices also make a noticable difference.

The upgrade from a CoCo to a SS50 system isn't the end of the line. All the major SS50 systems that support OS-9 support both OS-9 Level One and Level Two. The move to Level Two involves a new version of the OS-9 operating system, but no change in applications programs. All the modern SS50 systems I know of can be upgraded with little or no change to the hardware (the main requirement is memory management hardware). I imagine that OS-9 Level Two might run with the 56K of memory that Level One uses, but just barely. Level Two begins to come into its own at 128K. At 344K, I have never run out of memory.

BIG SYSTEM SOFTWARE

There is a bit of controversy arising in the OS-9 world. Smoke Signal Broadcasting has been responsible for a lot[4] of 6809 software over the years. There is even an operating system which they are responsible for. Now they are contributing to OS-9 software. My understanding is that Smoke commissioned someone to work on the version of OS-9 licensed to them. Their consultant made OS-9 less modular in order to improve its performance. The Smoke users I know confirm that the revisions make the Smoke version of OS-9 run faster than it used to. Running faster would seem to be an advantage, but the changes Smoke has made turn out to be a mixed blessing. There appear to be subtle incompatibilities between OS-9 as it comes from Microware and OS-9 from Smoke Signal Broadcasting. I have spoken to Microware and they say that they can't support Smoke's version of OS-9 (that may have changed by the time you read this). I have had trouble exchanging software with Smoke users.

The Smoke users are amazingly tolerant. I have read exchanges on the Compuserve OS-9 SIG in which Smoke users exchange tips on ways to prevent the DIR command from intermittently producing junk.

I certainly approve of improving OS-9's performance, but it is very important that an operating system be as standard as possible. If I were buying a system from Smoke Signal Broadcasting, I would want strong assurances that their version of OS-9 was compatible with Microware's on every level. A good test would be that all applications programs and system modules that run under standard OS-9 should run under the modified one, and vice versa.

THE COMPUSERVE OS-9 SIG

The OS-9 Special Interest Group on Compuserve is booming. Messages flow through the bulletin board so fast I am beginning to question my ability to read them all. Many experienced OS-9 users regularly check in, but it is a particularly good resource for newcomers. I strongly suggest that you join Compuserve if there is an access point close to you. It is worth it even if you only use it to access the OS-9 SIG.

OS-9 ON THE COLOR COMPUTER

I have been saying nasty things about Tandy which aren't true. I blamed the sloppy programming in the CCDISK device driver on Tandy when it seems the blame should fall on Microware and Microsoft. The bootstrap for the CoCo is in ROM. There is only one bootstrap ROM, designed by Microsoft for use with Color Disk Basic (I guess). Microware had to design the CoCo implementation of OS-9 so it could be loaded with that Bootstrap. The CoCo boot ROM reads 15 sectors off track 35 into a fixed location in memory. The OS9Boot file had to fit into those 15 sectors. This memory constraint forced Microware to pay even more attention to writing compact code than they usually do. Since 6809 instructions that do direct memory references take less memory than indexed instructions, Microware used them whenever they could. Since versatile device drivers take more memory than limited drivers, they wrote limited drivers. Tandy, I apologize for the nasty thoughts I sent your way.

I decided to write this month's project for the CoCo. I noticed that Color Basic has a number of commands which make assorted honks and beeps emerge from my TV. Basic09 has no way to make those noises. I checked the "Color Computer Technical Reference Manual" for information about the sound generator, and found that the Color Computer generates sound with a Digital to Analog converter. The output from the D/A converter is routed through an analog multiplexer to the modulator, and hence to the TV. It looked like OS-9 could learn to make noise.

I expect that the reason Microware didn't include sound generation in their OS-9 for the Color Computer is that sound generation with an D/A converter is a very time dependent operation. A note is played by gradually (in computer terms) raising and lowering the voltage generated by the D/A converter. This has to be done with a timing loop in a program. The timing loop must have exclusive use of the computer, or

---

[4] This problem was resolved to everyone's satisfaction when Smoke agreed to offer their users a choice of modified or unmodified OS-9.

the rate at which the voltage rises and falls will vary causing the note being generated to rise and fall. Some people might find the resulting yodel surprising. A program can give itself exclusive use of the computer by masking out interrupts, but locking out interrupts for more than a few millionths of a second is antisocial behavior for any program -- even a part of the operating system.

Still, the ability to at least be able to generate a beep seems important to me. I started by writing a program called Sound to investigate sound production. The program generates a saw-tooth wave that sounds rather like a saber saw cutting thin plywood, but it works. The most important discoveries I made while writing Sound were how to initialize the multiplexer so the D/A converter's output would be routed to the TV. The control registers at $FF03 and $FF23 both need to be modified. The fact that they could be modified was another interesting discovery. I am used to control registers being either readable or writeable. These registers are to some extent read/write. CoCo programmers may take this for granted, but I was pleasantly surprised.

Once the control registers are set, sound can be generated by simply writing different values into the most significant 6 bits of the byte at $FF20. The faster the value is changed the higher the pitch. I wrote the program to send 1000 waves, then stop.

There is lots of room for improvement in Sound. The quality of the note created by the program could be improved, and the program might even be made to play a song. I decided to drop Sound and work on building a Device Driver for the D/A converter.

The Device descriptor I wrote for the D/A converter, Beep, is almost as small as a Device Descriptor can be. The D/A converter is not a random access device so I decide to use the SCF file manager to drive it. There are no options except the one byte which indicates that it is a SCF device. There are three addresses in the descriptor. Normally a descriptor only needs one port address, but in this case, since the three addresses used in making the D/A converter make sound aren't related, I included all the addresses explicitly.

The Device Driver, called Beeper, is not interrupt driven. Most OS-9 device drivers use interrupts to give them a way to avoid wait loops, but I couldn't find a way to get the D/A converter to generate interrupts. In this case interrupts weren't necessary; the device responds as fast as data can be pumped into it.

The initialization entry puts some values that will be needed in the termination routine into device static storage, and sets the two PIA registers that need to be adjusted to permit sound to be made. The termination entry sets the two control registers back the way they were before Beeper started, and the GetStat and PutStat entries don't do anything at all. The read and write entries deal with the fact that the D/A converter only uses the high-order six bits of the register it is accessed through.

INSTALLATION OF BEEP/BEEPER

Beep and Beeper have to be typed in and assembled. As usual, the USE statements between the IFP1 and ENDC don't come out in the assembly listing. You will have to include use statements for both OS9DEFS and SCFDEFS for these programs. When you assemble the Beeper file it will generate a file in the execution directory called Beeper with both Beep and Beeper in it.

To use beeper first load it with the OS-9 command line:

```
OS9: load beeper
```

then link beeper with the command line:

```
OS9: link beeper
```

Since beeper is the second module in the file it will have a tendency to disappear if you don't link it.

As a first try you can get a low growl out of your computer by listing a file to /Beep. I used

```
OS9: list beeper >/beep
```

To get a more interesting sound out of the device you will need to feed it meaningful data. The BasicO9 program called TestBeep generates a thousand bytes of sine wave. TestBeep is intended to be packed and run out of the execution directory. If it is run from source the BYE should be removed. It takes a long time to initialize the array, so be patient. The wave can be sent one byte at a time with a loop like:

```
for I=1 to 1000
      put #sound,note(I)
next I
```

But OS-9 doesn't do very well at outputting a single character at a time. This program segment demonstrates that by generating a low, raspy note. To get a higher, smoother note I sent the entire thousand-byte array with one write. The quality of the tone still leaves a lot to be desired, but it's the best I could do quickly.

APPLICATIONS FOR /BEEP

I imagine that the timbre of the tone generated by TestBeep could be improved by spending more time with the wave form: the rough sin wave I use is pretty crude. Certainly the pitch can be varied by changing the frequency of the wave. I discovered that TestBeep just as it stands is a useful demonstration of OS-9's multitasking behavior. I started TestBeep with the command line:

```
OS9: BASIC09 TestBeep&
```

if you have RUNB

```
OS9: TestBeep&
```

will work fine. This runs the program as a background task. When the noise started, I ran a variety of different programs and noticed the effect on the sound.

If you want to generate a higher pitch than you can get out of Beeper, I suggest doing more work in the device driver. The approach I have in mind is to add a buffer in the device static storage for Beeper. When Beeper receives a request to write a zero value it will load the next 256 bytes written into the buffer. When the buffer isn't being loaded, each value written to Beep will indicate a number of times to send the buffer out the D/A. I believe that this approach will prove to be really useful, especially if there is a default wave loaded into the buffer by the INIT code.

THE USERS GROUP

I hope all the members of the OS-9 Users group will have their disks by the time you read this. I am afraid that some of you will have received the wrong type of disk. I am responsible for this. We don't have any record of the type of disk (size and format) any of our early members use. Some of the people who have joined recently have included information about their disk, but in most cases I have had to guess. If you get a disk you can't deal with, write to the Users Group address, and we will try to get you a disk you can read.

SOUND

```
Microware OS-9 Assembler 2.1   02/15/84 03:00:48                    Page 001
Sound - Sound generator for CoCo

00001                                   nam   Sound
00002                                   ttl   Sound generator for CoCo
00003                                   IFP1
00005                                   ENDC
00006    0011                 TYPE      SET   PRGRM+OBJCT
00007    0000 87CD0065                  MOD   ENDSND,NAM,TYPE,REENT+1,ENTRY,DSIZE
00008 D  0000                 CNTL      RMB   2          Address of D/A control registe
00009 D  0002                 CNTL2     RMB   2          Address of another D/A control
00010 D  0004                 PORT      RMB   2          Address of D/A input
00011 D  0006                 CNTR      RMB   2          Number of waves to send
00012 D  0008                 CNTLV     RMB   1          Initial value of first Control
00013 D  0009                 CNTL2V    RMB   1          Initial value of other control
00014 D  000A                           RMB   200        STACK
00015 D  00D2                 DSIZE     EQU   .
00016    000D 534F554E        NAM       FCS   /SOUND/
00017    0012                 ENTRY     EQU   *
00018                         ************
00019                         *  Initialize addresses in local storage
00020                         *
00021    0012 CCFF23                    LDD   #$FF23
00022    0015 DD00                      STD   CNTL
00023    0017 CCFF20                    LDD   #$FF20
00024    001A DD04                      STD   PORT
00025    001C CCFF03                    LDD   #$FF03
00026    001F DD02                      STD   CNTL2
00027                         ************
00028                         *  Save initial values of control registers
00029                         *  and set them to route D/A output to sound
00030                         *
00031    0021 A6D4                      LDA   [CNTL,U]
00032    0023 9708                      STA   CNTLV
00033    0025 8A08                      ORA   #$08
00034    0027 A7D4                      STA   [CNTL,U]
00035    0029 A6D802                    LDA   [CNTL2,U]
00036    002C 9709                      STA   CNTL2V
00037    002E 84F7                      ANDA  #$FF-$08
00038    0030 A7D802                    STA   [CNTL2,U]
00039                         ************
00040                         *  Initialize the counter
00041                         *
00042    0033 CC03E8                    LDD   #1000
00043    0036 DD06                      STD   CNTR
00044                         ***********
00045                         *  Send waves
00046                         *
00047    0038                 LOOP2
00048    0038 8600                      LDA   #0
```

```
00049          **********
00050          *  Send each wave
00051          *
00052    003A             LOOP1
00053    003A A7D804              STA   [PORT,U]
00054    003D 8B04                ADDA  #4
00055    003F 12                  NOP
00056    0040 12                  NOP
00057    0041 12                  NOP
00058    0042 12                  NOP
00059    0043 12                  NOP
00060    0044 12                  NOP
00061    0045 12                  NOP
00062    0046 12                  NOP
00063    0047 12                  NOP
00064    0048 8100                CMPA  #0
00065    004A 26EE                BNE   LOOP1
00066          *********
00067          *  End of sending one wave.
00068          *  See if we still need to send more
00069          *
00070    004C DC06                LDD   CNTR
00071    004E 830001              SUBD  #1
00072    0051 DD06                STD   CNTR
00073    0053 26E3                BNE   LOOP2
00074          **********
00075          *  Restore initial values to control registers
00076          *
00077    0055 9608                LDA   CNTLV
00078    0057 A7D4                STA   [CNTL,U]
00079    0059 9609                LDA   CNTL2V
00080    005B A7D802              STA   [CNTL2,U]
00081    005E 5F                  CLRB                clear carry
00082    005F 103F06              OS9   F$EXIT        return to OS-9
00083    0062 528D69              EMOD
00084    0065             ENDSND  EQU   *

00000 error(s)
00000 warning(s)
 0065 00101 program bytes generated
  D2 00210 data bytes allocated
  F8 03832 bytes used for symbols
```

BEEPER


```
00001                            NAM   BEEPER
00002                            IFP1
00006                            ENDC
00007                            USE   BEEP        Device Descriptor
00008                            TTL   DEVICE DESCRIPTOR
00009                            NAM   BEEP
00010    00F1            TYPE    SET   DEVIC+OBJCT
00011    0000 87CD0027            MOD   BPEND,BPNAM,TYPE,REENT+1,FMNAME,DRVNAM
00012    000D 03                  FCB   READ.+WRITE. MODES
00013    000E FFFF20              FCB   $FF,$FF,$20 PORT ADDRESS
00014
00015    0011 01                  FCB   OPTL        Length of options section
00016    0012            OPTIONS EQU   *
00017    0012 00                  FCB   DT.SCF
00018    0001            OPTL    EQU   *-OPTIONS
00019
00020    0013 FF23       CNTL1   FDB   $FF23       address of control byte 1
00021    0015 FF03       CNTL2   FDB   $FF03       address of control byte 2
00022    0017 424545D0   BPNAM   FCS   /BEEP/      name of this module
00023    001B 5343C6     FMNAME  FCS   /SCF/       File Manager name
00024    001E 42454550   DRVNAM  FCS   /BEEPER/    Device driver name
00025    0024 58AEA3              EMOD
00026    0027            BPEND   EQU   *
00027                            TTL   DEVICE DRIVER FOR D/A
```

```
00028   00E1             TYPE     SET   DRIVR+OBJCT
00029   0081             REVS     SET   REENT+1
00030   0000 87CD0076            MOD   BPREND,BPRNAM,TYPE,REVS,ENTER,MEMSIZE
00031   000D 03                  FCB   READ.+WRITE. DRIVER MODE
00032   000E 42454550    BPRNAM  FCS   /BEEPER/
00033   0014 01                  FCB   1           EDITION
00034                    **************
00035                    *  Device Static storage
00036                    *
00037 D 001D                     ORG   V.SCF       System part of Static Storage
00038                    ************
00039                    *  Local part of static storage
00040                    *
00041 D 001D             PORTA    RMB   2           PORT ADDRESS
00042 D 001F             CTL1V    RMB   1           HOLD CNTL1 VAL
00043 D 0020             CTL2V    RMB   1           HOLD CNTL2 VALUE
00044 D 0021             CTL1A    RMB   2           HOLD CNTL1 ADDR
00045 D 0023             CTL2A    RMB   2           HOLD CNTL2 ADDR
00046 D 0025             MEMSIZE  EQU   .
00047
00048                    ************
00049                    *  Entry vectors
00050                    *
00051   0015             ENTER
00052 W 0015 16000F              LBRA  INIT
00053 W 0018 16002C              LBRA  READ
00054 W 001B 160031              LBRA  WRITE
00055 W 001E 16003E              LBRA  GETSTAT
00056 W 0021 16003B              LBRA  PUTSTAT
00057 W 0024 16003A              LBRA  TERM

00058   0027             INIT
00059                    *******
00060                    *  U ADDRESS OF DEVICE STATIC STORAGE
00061                    *  Y ADDRESS OF DEVICE DESCRIPTOR MODULE
00062                    *
00063   0027 AEA813              LDX   CNTL1,Y     Get control address 1 out of D
00064   002A AFC821              STX   CTL1A,U     Save the address
00065   002D A684                LDA   ,X           Get the present value of cntl
00066   002F A7C81F              STA   CTL1V,U     save it for later restore
00067   0032 8A08                ORA   #$08        set it for sound
00068   0034 A784                STA   ,X
00069
00070   0036 AEA815              LDX   CNTL2,Y     do the same stuff for cntl2
00071   0039 AFC823              STX   CTL2A,U
00072   003C A684                LDA   ,X
00073   003E A7C820              STA   CTL2V,U
00074   0041 84F7                ANDA  #$FF-$08
00075   0043 A784                STA   ,X
00076
00077   0045 5F                  CLRB              CLEAR CARRY
00078   0046 39                  RTS               RETURN
```

```
00079   0047                READ
00080                       *********
00081                       * U ADDRESS OF DEVICE STATIC STORAGE
00082                       * Y ADDRESS OF PATH DESCRIPTOR
00083                       * RETURN CHARACTER READ IN A
00084                       *
00085   0047 AE41                   LDX   V.PORT,U   port address from device descr
00086   0049 A684                   LDA   ,X         D/A value
00087   004B 44                     LSRA
00088   004C 44                     LSRA             Shift out low order bits
00089   004D 5F                     CLRB             Clear carry
00090   004E 39                     RTS

00091   004F                WRITE
00092                       ********
00093                       *  U DEVICE STATIC STORAGE
00094                       *  Y PATH DESCRIPTOR
00095                       *  A VALUE TO WRITE
00096                       *
00097   004F AE41                   LDX   V.PORT,U
00098   0051 48                     LSLA             Shift out high order bits
00099   0052 48                     LSLA
00100   0053 3402                   PSHS  A          save value to write
00101   0055 A684                   LDA   ,X         Get current value at Port
00102   0057 8403                   ANDA  #%00000011 clear D/A value
00103   0059 AAE0                   ORA   ,S+        put value to write in
00104   005B A784                   STA   ,X         send it
00105   005D 5F                     CLRB
00106   005E 39                     RTS              RETURN

00107   005F                GETSTAT
00108   005F                PUTSTAT
00109   005F 5F                     CLRB
00110   0060 39                     RTS
00111   0061                TERM
00112                       ********
00113                       *  U  DEVICE STATIC STORAGE
00114                       *
00115   0061 AEC821                 LDX   CTL1A,U    restore original Cnt11 value
00116   0064 A6C81F                 LDA   CTL1V,U
00117   0067 A784                   STA   ,X
00118   0069 AEC823                 LDX   CTL2A,U    restore original Cnt12 value
00119   006C A6C820                 LDA   CTL2V,U
00120   006F A784                   STA   ,X
00121   0071 5F                     CLRB             clear carry
00122   0072 39                     RTS
00123   0073 A182B1                 EMOD
00124   0076                BPREND  EQU   *

00000 error(s)
00006 warning(s)
$009D 00157 program bytes generated
$0008 00008 data bytes allocated
$164B 05707 bytes used for symbols
```

TESTBEEP

```
PROCEDURE TESTBEEP
DIM NOTE(1000):BYTE
DIM I:INTEGER
DIM SOUND:INTEGER
OPEN #SOUND,"/BEEP":WRITE
FOR I=1 TO 1000
NOTE(I)=32*(1+SIN(I))
NEXT I
FOR I=1 TO 100
PUT #SOUND,NOTE
NEXT I
BYE
```

COLUMN FOURTEEN

MORE ABOUT THE COCO DISK DRIVER

After in sending last month's column I had second thoughts about what I said about the OS-9 disk driver for the CoCo. I didn't believe what I had written. The gist of what I said was that Microware and Microsoft together were to blame for the non-standard disk driver included with the CoCo OS-9. The boot ROM in the CoCo loads just 15 sectors from track 34 on the boot disk into set locations in memory and jumps to them. This is Microsoft's idea of a nice way to boot a computer. What I said last month was that Microware managed to squeeze all of OS-9 into those 15 sectors by extensive compression of the code. This sounded pretty extreme to me, but I thought that was what I had heard from Ken Kaplan out at Microware.

Later, I became certain that I misunderstood Ken. There is no way all the core resident parts of OS-9 could be squeezed into that amount of disk, and, if all of OS-9 was loaded by the ROM boot, why does the CoCo have a two stage boot?

I called Microware to check my facts. I was wrong. In the first stage of the boot the CoCo ROM does load data from 15 sectors on track 34 into memory and jump to it, but only a few important parts of OS-9 are loaded: the kernal, the Init module, and the OS-9 bootstrap. These are the modules that are found in ROM on other OS-9 systems. The next stage of the boot uses the OS-9 bootstrap which was loaded in the first pass to do a normal OS-9 boot. The parts of OS-9 loaded in the first phase of the boot had to be squeezed hard, but much of the disk driver is loaded in the second phase of the boot.

There were a number of ways for Microware to get a full-featured disk driver into the CoCo, but they didn't. The restrictions on the first phase of the boot forced them to deviate from OS-9 standards in the boot module part of the disk driver. I believe they couldn't find a way to interest Tandy in the extra work (and memory) required to discard the boot after its work was done and load a driver that worked independently. That is certainly reasonable. Why should Tandy be interested in making it easy for people to use non-Tandy peripherals?

In any case, the problem seems to be solved. O. P. Johnson is advertising software that lets CoCo OS-9 deal with every disk format my Gimix can handle. I haven't tried his software, but I have heard from satisfied customers. I also own a 256K memory board made by Dan Johnson. I purchased one of the first boards he sold and had the kind of difficulties one might expect. I came to respect Dan Johnson while we struggled together to fix the problems which I discovered. He is good with hardware and software and VERY conscientious. I can't recommend the software because I haven't tried it (yet). I do recommend the man who sells it.

WHERE NEXT?

I have two very different OS-9 systems, a very large Gimix Level Two system and a CoCo. They fall at almost opposite extremes of the spectrum of microcomputers. The CoCo is so light and small that I think nothing of tucking it under my arm and walking a mile down to campus. The Gimix is so heavy that I am daunted by the thought of moving that stack of hardware even a few feet. The CoCo can't really handle more than one concurrent user. I routinely have two users on my Gimix and know people whose Gimix machines typically serve four or more concurrent users. The CoCo includes full graphics and a "terminal" protocol which is consistent across all CoCos. This is a big issue for other OS-9 users, particularly software developers who have to write programs which can be configured for any terminal.

Noting the similarities and differences between these computers has given me a lot of ideas about the kind of hardware I would like to see OS-9 running on. I imagine all computer users spend some time dreaming about the system they would have if only...

My dream computer is a personal computer, or, to use the popular phrase, a personal work station. I have grown used to the idea of OS-9 Level Two as a multi-user operating system, but I still prefer to think of it as a very powerful single-user system. Sharing computers is a way to save money. When I imagine the computer I would like, I don't consider money first.

Naturally, my dream computer runs OS-9 Level Two. It includes a bit-mapped screen (color optional), several dedicated processors, support for some graphics input device (I haven't chosen between a bit pad, a mouse and a light pen), and more than plenty of memory.

Many people seem to think that 128K is the right amount to run OS-9 Level Two in. Now you CAN run Level Two in even less, but you don't really appreciate it until you get to at least 192K. My dream machine would have at least 192K upgradable to 256K, better still, 512K. There are so many uses for memory! Solid state disk drives or caches give better access times than hard disks but use a lot of memory. Complex programs can take lots of memory, but, when they are well written, they are powerful and easy to use. Sometimes lots of memory is needed for simple storage of data. I know a woman who keeps running out of space for her spread sheet on an IBM PC. She has about 600K![5] So lets put lots of memory in the dream machine.

Graphics hardware is never good enough. At any rate that's the way I react to it. If the resolution and the number of colors is sufficient, the screen takes too long to update. If data is displayed by fussing with parameter lists and registers, the system is too limited. If the screen is bit-mapped, it takes too much attention from the CPU to control the screen. The best solution seems to be to have a sepa-

[5] Of course, with that much RAM you need extra high-capacity disks to save what you're working on.

rate processor that deals with a bit-mapped display. If the graphics processor has a very high speed connection to the rest of the system, and can be dynamically programmed to do more than just update the screen, the result should be speed and flexibility in graphics.

There is use for more than one special processor in my dream computer. If graphics support is included in the package, it would be foolish to require a terminal to be attached to the computer; an attached keyboard would be sufficient. A dedicated processor to scan the keyboard would take another load off the main processor. The other I/O devices could also use their own processors. My Gimix uses a 6809 on one of its serial cards to take some of the interrupt load off the main processor. It speeds my machine up a little, but doesn't have any other use. If the software for the I/O processor was loaded (and reloaded) by the main processor it would let the serial board be programmed to handle high-speed networks and other applications where timing is important. Even disk controllers could use their own special processors. I don't know of any programmable disk controllers they could do for disk I/O what smart seria cards has done for terminal I/O. The Gimix intelligent serial card contains a good part of SCFMAN. By unloading this work onto a special processor more cycles are left for user programs. RBFMAN is more complicated than SCFMAN and uses more CPU time. If most of that work could be done by a separate processor still more of the resources of the main processor would be available for the user.

In fact, why talk about the main processor? In many cases OS-9 processes don't share memory with one another. If the dream computer had a bus where additional processor boards with some memory and perhaps I/O could be inserted, OS-9 could run independent processes on their own procedures. Most personal work station users don't need to run more than three or four processes at a time, so including many of what amounts to separate computers in the package would be wasteful. But, if the power is available the applications will arrive.

Mice are making a big splash these days. The Xerox Star, the Apple Macintosh and Lisa, and lots of more expensive work stations are using them. I would definitely pick a mouse over a joy stick. I have more trouble deciding that a light pen or graphics pad isn't a better tool than a mouse. The graphics pad is very precise and the stylus can be used about like a mouse. The arguments against graphics pads are that they are expensive, require desk space, and, for some applications, force the person using them to mentally map from the bit pad to the screen. The cost problem I will ignore -- after all this is a dream computer. The other two problems apply to mice as well. A light pen doesn't require desk space or a mental mapping, out I don't find them very precise and my hand obscures the screen when I am pointing. I can't make up my mind.

A fancy computer like this deserves fancy software. The peanut-butter and jelly programs now available for OS-9 just don't live up to the hardware.

My pet peeve with OS-9 software has always been its lack of excellent editors. I like Dynastar fine, and I have heard nice things about Screditor and Stylograph, but these programs are at least five years behind the state-of-the-art. My dream machine deserves something special. Do you suppose EMACS could be ported to OS-9?

A real database program would be nice. Something more than a filing cabinet or stack of index cards metaphor.

I bet Knuth's TeX would run on something like this. Some good graphics programs, especially a graphics editor would make the graphics support a lot more useful. A real statistical program like SAS, or SPSS would make some people happy. Others need really good communications software.

Languages aren't as important as the software written in them, but OS-9 is still painfully short of languages. I bet APL would run well under OS-9. Fortran is old fashion, but we really should have it. Those are the fundamental languages, but there is an endless list, including: Pilot, PL/1, Logo, Smalltalk, and others.

Networking is another sexy topic these days. Expensive computers (which my dream machine is turning out to be) are generally used by people for whom communication is terribly important. Electronic mail, electronic calendars, and sharing of files and other resources are important to them. OS-9 doesn't include networking software, but I think it will be at least as easy to run over a network as any other operating system.

Enough of the dreaming. Truly, my dream machine is not so very far away. I/O processors exist, and I am sure more are coming. I have heard talk about slave processors. There are graphics boards available for the SS-50 bus that are a lot like what I have in my dream machine. The CoCo comes with bit mapped graphics standard.

For my Gimix I can hope for I/O and slave processors and a better (and less expensive) graphics board. For my CoCo I can aim low and hope for a disk controller with an onboard buffer, or aim high and look for a real Level Two system with as much done in hardware as possible (I/O, sound, and graphics). From my viewpoint as a Level Two user I think Tandy would be crazy not to offer a CoCo with Level Two. For a software person like me, it is fun to think up lots of things that hardware people should do for us, but the most important part of any computer is its software.

Some of my software wish list will have to wait for better hardware, in particular for more memory. Much of it can be done now. I have done some primitive networking myself. A really special database program or editor would push a 6809 hard, but might be possible. I have heard from people who are working on lots of nice things for OS-9. Pretty near every piece of software for my dream machine is a project someone is working on now.

MORE NOISE FROM THE COCO

Last month I included a driver for the Digital-to-Analog converter in the CoCo. That driver was useful for low-speed D/A applications, but it didn't do very well at sound generation. The highest pitch my driver could manage was something of a gurgle. The speed problem wasn't in the driver. It takes a long time for a character to get through SCFMAN. Even when a block of characters goes through together there is enough delay in the transmission of each character to make smooth, high- frequency waves impossible. Fortunately, generating music isn't the only purpose for an D/A converter. Controlling lab instruments, motors, and such are all fine applications which only require a voltage to be changed infrequently -- 10 times per second at most.

I ended last month's column with a few suggestions for ways to make the D/A driver better at generating sound. This month I went ahead and took my suggestions. This month's A/D driver does a pretty good job of making music. It even makes nice chords. I made the improvement I suggested last month. If the driver receives a zero, it places the next 360 bytes sent to it in a special buffer. Characters that don't go into the buffer cause the contents of the buffer to be transmitted through the D/A a number of times corresponding to the magnitude of the character written. Since it takes a fixed amount of time to transmit the buffer, each character from $01 to $FF will take a fixed amount of time to send. This way each character sends a note of a set duration whatever the pitch.

At first I used a buffer 128 bytes long. That was easy to handle in BEEPER, but it was hard to build a wave in. It is important that a whole number of cycles fit into the buffer. It was difficult to generate a wave that fit precisely into 128 values. Numbers like 90 and 360 work better when angles are measured in degrees (if they are measured in radians it is hard to make any integers come out evenly.) I tried a 90 byte buffer, but I found it hard to store smooth, high-pitched tones in it. After the buffer got over 128 bytes long, I used the D register to offset the index into it so length didn't make much difference. I chose 360 bytes as the length of the buffer because it is an easy number to work with when generating the wave.

Interrupts are a problem to time-dependent things like music. I tried BEEPER with interrupts masked and unmasked. When interrupts are unmasked the sound is definitely not pure; however, when the interrupts are masked lots of bad things happen. With interrupts masked nothing happens except operation of the D/A. Time doesn't get updated, the keyboard doesn't get scanned, and, if you are using the RS-232 port, it comes to a halt. In the version of BEEPER included with this column I commented out the ORCC and ANDCC. Try it both ways and choose for yourself.

THIS MONTH'S DRIVER

I got a little carried away with the test driver for BEEPER. The program calls for a magnitude and frequency for a wave (the numbers are only relative). The sine wave generated with these numbers is added to whatever wave has already been generated. The resulting wave is displayed. If a Y is entered, the wave is loaded into BEEPER's buffer and a few beeps are sent, if an A is entered another sine wave is prompted for and added to the existing wave, and if anything else is entered the wave is erased and the program starts over with a clean slate.

I am afraid that this test driver is another program that needs work. BEEPER truncates numbers greater than 63 to 63. If the sum of the sine waves loaded into BEEPER's buffer is greater than 63 at any point the wave will be clipped (as hi-fi people say). It would be good if TBEEP2 would check for this. I also get pretty frustrated when I don't like the last sine wave I added to a wave I am building and have to wipe out the entire waveform to get rid of it. On the other hand I am rather partial to the graphic display of the waveform.

THE USERS GROUP

Things aren't moving as quickly for the OS-9 Users Group as we hoped they would. We published our first news letter (called MOTD) months ago. By the way, if you are a member and didn't receive a copy of MOTD, send a note to the Users Group. Our system for keeping track of members seems pretty reliable, but it may have cracks in it. We are working on the second issue. The most important thing to most members seems to be the software exchange. There have been a number of problems getting the software exchange disks out. The most interesting problem has been a disk incompatibility between 40 track disks written by 80 track drives on two different manufacturer's systems. Watch for this problem!

There also seems to be some trouble getting disks. Three dollars per disk delivered to a member is a very low price. It's hard to be too impatient. In any case, barring another serious hold-up, the disks should be in the mail by late March. Let me say again that we don't know the disk format many users need. If I guess wrong, send the Users Group a letter and we'll try to find a way to straighten things out.

```
Microware OS-9 Assembler 2.1   03/13/84 20:31:36                    Page 001
BEEPER - OS-9 System Symbol Definitions

00001                                   NAM   BEEPER
00002                                   IFP1                Use OS9DEFS, SCFDEFS and IODEF
00004                                   ENDC                and ENDC
00005                                   USE   BEEP
00006                                   TTL   DEVICE DESCRIPTOR
00007                                   NAM   BEEP
00008   00F1                TYPE        SET   DEVIC+OBJCT
00009   0000 87CD0027                   MOD   BPEND,BPNAM,TYPE,REENT+1,FMNAME,DRVNAM
00010   000D 03                         FCB   READ.+WRITE. MODES
00011   000E FFFF20                     FCB   $FF,$FF,$20 PORT ADDRESS
00012
00013   0011 01                         FCB   OPTL          Length of options section
00014   0012                OPTIONS     EQU   *
00015   0012 00                         FCB   DT.SCF
00016   0001                OPTL        EQU   *-OPTIONS
00017
00018   0013 FF23           CNTL1       FDB   $FF23         address of control byte 1
00019   0015 FF03           CNTL2       FDB   $FF03         address of control byte 2
00020   0017 424545D0       BPNAM       FCS   /BEEP/        name of this module
00021   001B 5343C6         FMNAME      FCS   /SCF/         File Manager name
00022   001E 42454550       DRVNAM      FCS   /BEEPER/      Device driver name
00023   0024 58AEA3                     EMOD
00024   0027                BPEND       EQU   *
00025                                   TTL   DEVICE DRIVER FOR D/A


Microware OS-9 Assembler 2.1   03/13/84 20:31:41                    Page 002
BEEP - DEVICE DRIVER FOR D/A

00026   00E1                TYPE        SET   DRIVR+OBJCT
00027   0081                REVS        SET   REENT+1
00028   0000 87CD00D8                   MOD   BPREND,BPRNAM,TYPE,REVS,ENTER,MEMSIZE
00029 D 001D                            ORG   V.SCF
00030 D 001D                PORTA       RMB   2             PORT ADDRESS
00031 D 001F                CTL1V       RMB   1             HOLD CNTL1 VAL
00032 D 0020                CTL2V       RMB   1             HOLD CNTL2 VALUE
00033 D 0021                CTL1A       RMB   2             HOLD CNTL1 ADDR
00034 D 0023                CTL2A       RMB   2             HOLD CNTL2 ADDR
00035 D 0025                COFFSET     RMB   2             OFFSET IN BUFFER
00036   0168                BUFLEN      EQU   90*4
00037 D 0027                BUFFER      RMB   BUFLEN
00038 D 018F                MEMSIZE     EQU   .
00039   000D 03                         FCB   READ.+WRITE. DRIVER MODE
00040   000E 42454550       BPRNAM      FCS   /BEEPER/      Program Name
00041   0014 01                         FCB   1             EDITION
00042                       ***********
00043                       *  Entry points
00044                       *
00045   0015                ENTER
00046 W 0015 16000F                     LBRA  INIT
00047 W 0018 160042                     LBRA  READ
00048 W 001B 160047                     LBRA  WRITE
00049   001E 1600A0                     LBRA  GETSTAT
00050   0021 16009D                     LBRA  PUTSTAT
00051   0024 16009C                     LBRA  TERM
00052   0027                INIT
00053                       *******
00054                       *  U ADDRESS OF DEVICE STATIC STORAGE
00055                       *  Y ADDRESS OF DEVICE DESCRIPTOR MODULE
00056                       *
00057   0027 AEA813                     LDX   CNTL1,Y       Move the address of cntl1 byte
00058   002A AFC821                     STX   CTL1A,U       from the D.Descriptor to stati
00059   002D A684                       LDA   ,X            save the value of the cntl1 by
00060   002F A7C81F                     STA   CTL1V,U
00061   0032 8A08                       ORA   #$08          set one of the bits
00062   0034 A784                       STA   ,X            that turns on sound
00063   0036 AEA815                     LDX   CNTL2,Y       Move the address of cntl2 byte
00064   0039 AFC823                     STX   CTL2A,U       from the D.Desc to static stor
00065   003C A684                       LDA   ,X            save the value of the cntl2 by
00066   003E A7C820                     STA   CTL2V,U
00067   0041 84F7                       ANDA  #$FF-$08      set the other bit
00068   0043 A784                       STA   ,X            that turns on sound
00069   0045 8D08                       BSR   INITBUF       Initialize the sound buffer
00070   0047 5F                         CLRB                CLEAR CARRY
00071   0048 E7C825                     STB   COFFSET,U     Coffset is a two byte field
00072   004B E7C826                     STB   COFFSET+1,U
00073   004E 39                         RTS                 RETURN
```

```
00074          ************
00075          *   Put something that won't sound too bad
00076          *   into the sound buffer
00077          *
00078   004F                INITBUF
00079   004F 30C827                 LEAX  BUFFER,U
00080   0052 CC0167                 LDD   #BUFLEN-1
00081   0055                INITLOOP
00082   0055 E78B                   STB   D,X
00083   0057 830001                 SUBD  #1
00084   005A 2CF9                   BGE   INITLOOP
00085   005C 39                     RTS
00086   005D                READ
00087          *********
00088          * U ADDRESS OF DEVICE STATIC STORAGE
00089          * Y ADDRESS OF PATH DESCRIPTOR
00090          * RETURN CHARACTER READ IN A
00091          *
00092   005D AE41                   LDX   V.PORT,U
00093   005F A684                   LDA   ,X          get the value in the D/A regis
00094   0061 44                     LSRA
00095   0062 44                     LSRA              Shift out the low order bytes
00096   0063 5F                     CLRB              Clear carry
00097   0064 39                     RTS
00098   0065                WRITE
00099          ********
00100          *  U DEVICE STATIC STORAGE
00101          *  Y PATH DESCRIPTOR
00102          *  A VALUE TO WRITE
00103          *
00104   0065 6DC826                 TST   COFFSET+1,U If coffset isn't zero
00105   0068 263F                   BNE   DEFINE      we are in the process of filli
00106   006A 6DC825                 TST   COFFSET,U   buffer.  We have to tst both C
00107   006D 263A                   BNE   DEFINE
00108   006F 4D                     TSTA              If the character to write is 0
00109   0070 272F                   BEQ   SDEFINE     the sound buffer
00110          ********
00111          *  LOOP THROUGH BUFFER
00112          *
00113   0072 30C827                 LEAX  BUFFER,U    the address of the sound buffe
00114   0075 3402                   PSHS  A           SAVE COUNT
00115          * ORCC #INTMASKS Shut off interrupts
00116   0077                CYCLE
00117   0077 CC0167                 LDD   #BUFLEN-1   Offset in buffer
00118   007A 3406                   PSHS  D           Save offset
00119   007C                WLOOP
00120   007C A68B                   LDA   D,X         get a byte out of buffer
00121   007E 3410                   PSHS  X
00122          **** on second thought it would have been
00123          **** better to just do a leax BUFFER,U later instead
00124          **** of saving this value here
00125   0080 AE41                   LDX   V.PORT,U    The address of the D/A registe
00126   0082 3402                   PSHS  A           Build the byte to store in the
00127   0084 A684                   LDA   ,X          register
00128   0086 8403                   ANDA  #%00000011
00129   0088 AAE0                   ORA   ,S+
00130   008A A784                   STA   ,X          Store the new D/A value
00131   008C 3510                   PULS  X           recover the buffer address
00132          ***** see note
00133   008E ECE4                   LDD   ,S          get the new offset in buffer
00134   0090 830001                 SUBD  #1          decriment the offset
00135   0093 EDE4                   STD   ,S
00136   0095 2CE5                   BGE   WLOOP       if it isn't negative send the
00137   0097 3262                   LEAS  2,S         Clear stack
00138   0099 6AE4                   DEC   ,S          decriment repeat count
00139   009B 26DA                   BNE   CYCLE       cycle if not zero
00140          * ANDCC #$FF-INTMASKS
00141   009D 3261                   LEAS  1,S         CLEAR STACK
00142   009F 5F                     CLRB
00143   00A0 39                     RTS
00144
00145   00A1                SDEFINE
00146   00A1 CC0168                 LDD   #BUFLEN
00147   00A4 EDC825                 STD   COFFSET,U
00148   00A7 5F                     CLRB              CLEAR CARRY
00149   00A8 39                     RTS
```

```
00150          **********
00151          *  Load the Sound buffer
00152          *
00153    00A9              DEFINE
00154    00A9 48                    LSLA              Prepare the value
00155    00AA 48                    LSLA
00156    00AB 3402                  PSHS  A           save it
00157    00AD ECC825                LDD   COFFSET,U   Current offset
00158    00B0 830001                SUBD  #1
00159    00B3 EDC825                STD   COFFSET,U   Update offset
00160    00B6 30C827                LEAX  BUFFER,U
00161    00B9 308B                  LEAX  D,X         location to store this byte at
00162    00BB 3502                  PULS  A           get the byte
00163    00BD A784                  STA   ,X          store it
00164    00BF 5F                    CLRB
00165    00C0 39                    RTS
00166    00C1              GETSTAT
00167    00C1              PUTSTAT
00168    00C1 5F                    CLRB
00169    00C2 39                    RTS
00170    00C3              TERM
00171          ********
00172          *  U DEVICE STATIC STORAGE
00173          *
00174    00C3 AEC821                LDX   CTL1A,U
00175    00C6 A6C81F                LDA   CTL1V,U
00176    00C9 A784                  STA   ,X          restore the original ctl1 valu
00177    00CB AEC823                LDX   CTL2A,U
00178    00CE A6C820                LDA   CTL2V,U
00179    00D1 A784                  STA   ,X          restore the original ctl2 valu
00180    00D3 5F                    CLRB
00181    00D4 39                    RTS
00182    00D5 A60D8D                EMOD
00183    00D8              BPREND   EQU   *

00000 error(s)
00003 warning(s)
00FF 00255 program bytes generated
0172 00370 data bytes allocated
16D2 05842 bytes used for symbols
```

TBEEP2

```
PROCEDURE TBEEP2
 0000      (* ------------------------------------------------
 0032      (* TBeep2 is a test driver for the device driver
 0062      (* BEEPER.  It loads BEEPER with a wave form,
 008F      (* then sends it a few more characters to test the tone.
 00C7      (* ------------------------------------------------
 00F9      (*
 00FC      (* Note is an array which contains the values which will be sent to
 013F      (* the D/A to form a note
 0158      (*
 015B      DIM NOTE(360):BYTE
 0167      DIM I,J,K:INTEGER
 0176      DIM SOUND:INTEGER \(* Path number for A/D
 0193      DIM MAGNITUDE,FREQ:INTEGER \(* variables used to form the waveform
 01C4      DIM C:BYTE \(* a utility one-byte variable
 01E9      DIM CMD:STRING \(* waveform command
 0203
 0204      OPEN #SOUND,"/BEEP":WRITE
 0214      DEG  \(* Use degrees for angles
 022F      (* Initialize Note to zeros
 024A      FOR I=1 TO 360
 025B        NOTE(I)=0
 0266      NEXT I
 0271
 0272      (*
 0275      (* Build waveform
 0286      (*
 0289      LOOP
 028B        RUN GFX("ALPHA") \(* make screen printable
 02B0        (* Get parameters for a sin wave
 02D0        INPUT "MAGNITUDE* ",MAGNITUDE
 02E3        INPUT "FREQUENCY: ",FREQ
 02F6        (*
 0309        (* add the sin wave to the wave in NOTE
```

```
0320          (*
0323          FOR I=1 TO 360
0334            NOTE(I)=NOTE(I)+MAGNITUDE*(1+SIN(I*FREQ))
0358          NEXT I
0363          (*
0366          (* Display the graph
037A          (*
037D          RUN GFX("MODE",0,1)
038F          FOR I=1 TO 180
039F            J=NOTE(I*2)-2
03B0            K=J+4
03BB            IF J<0 THEN J=0
03CD            ENDIF
03CF            J=J*2
03DA            K=K*2
03E5            IF J>192 THEN J=192
03F7            ENDIF
03F9            IF K>192 THEN K=192
040B            ENDIF
040D            RUN GFX("LINE",I,J,I,K)
042D          NEXT I
0438          (*
043B          (* Display a little bit of the next cycle
0464          (* to demonstate the the wave is continuous
048F          (*
0492          FOR I=181 TO 255
04A2            J=NOTE((I-180)*2)-2
04B6            K=J+4
04C1            IF J<0 THEN J=0
04D3            ENDIF
04D5            J=J*2
04E0            K=K*2
04EB            IF J>192 THEN J=192
04FD            ENDIF
04FF            IF K>192 THEN K=192
0511            ENDIF
0513            RUN GFX("LINE",I,J,I,K)
0533          NEXT I
053E          (*
0541          (* There is no prompt because the screen is full of
0574          (* graphics, but enter Y<CR> A<CR>, or N<CR> after
05A6          (* the graph has been drawn
05C1          (*
05C4          INPUT CMD
05C9        EXITIF CMD="Y" THEN
05D6        ENDEXIT
05DA          IF CMD<>"A" THEN
05E7            FOR I=1 TO 360 \(* The waveform is bad,
0610              NOTE(I)=0 \(* zero it and start over
0634            NEXT I
063F          ENDIF
0641          RUN GFX("CLEAR")
064E        ENDLOOP
0652        RUN GFX("ALPHA")
065F        RUN GFX("QUIT")
066B        C=0 \(* a zero tells the driver to use the next 360 characters
06 B        PUT #SOUND,C \(* to build a new form
06CB        PUT #SOUND,NOTE \(* send the new form
06E9        PRINT "STARTING SOUND"
06FB        (*
06FE        (* Send a few beeps of different lengths
0726        (*
0729        FOR I=100 TO 250 STEP 50
073E          C=I
0746          PUT #SOUND,C
0750          GOSUB 100
0754          PRINT "END OF LOOP ",I
0768        NEXT I
0773        END
0775        (*
0778        (* Delay a little
0789        (*
078C 100
0790        FOR J=1 TO 500
07A1        NEXT J
07AC        RETURN
```

COLUMN FIFTEEN

THE OS-9 SEMINAR

I went to the OS-9 users seminar last summer, so did almost every person I've heard of in the OS-9 community. It was interesting walking through the exhibit hall and listening to the speakers. The thing that makes me willing to go halfway across the country to take part in the seminar this summer is the fun I had last year talking with other OS-9 people. Most of us, myself included, spend our lives in a world where every other microcomputer user thinks the world ends right past PC-DOS and CPM. Last summer I fairly wallowed in the pleasure of being with hundreds of people who shared my interest in OS-9. We argued, agreed, complained, puzzled, and applauded about things that are dear to OS-9 users (and not many others).

If you need a practical reason to spend a long weekend in Des Moines, bring a question with you. If you have been itching to show the person on the Microware hotline a problem that he can't reproduce, he'll be there. Go demonstrate the problem yourself. If you want to suggest that OS-9 badly needs a WALL command you can probably find someone important and back him into a corner about it.

All the important vendors were there last year -- I assume they'll be back. If they come, you'll be able to check the Smoke Signal version of OS-9 for compatibility with other versions. Try a few things on the GIMIX III. I hope Privac comes again; their graphics board is much more impressive in motion than in an advertisement. I imagine there'll be a bunch of new vendors there showing CoCo products.

The vendors and Microware staff notwithstanding, the best place to look for answers will be standing or sitting beside you (very likely at breakfast or some other improbable time). Last year I found the other users at the Seminar a mine of useful information. If you are a vendor, go to the Seminar even if you don't have a booth. It is a great place to test the water.

The Seminar is a businesslike affair, but it is also something of a party: Jeanne Kaplan's party. Everyone who has dealt with Microware for any length of time knows that Jeanne is a consummate organizer. Last year everything ticked along smoothly despite the fact that she must have been slowed down a little by the child she was about to have. Last year Microware hosted a banquet and a fancy brunch. The Governor of Iowa came and gave us a little talk over dinner. Ken Kaplan handed out prizes to individuals who had made particularly distinguished contributions to the OS-9 community. At the brunch more prizes were handed out. I wonder what is in store for us this year.

Microware is going to give the Users Group some software for a raffle. I don't know just how it will be organized yet, but the plan is to hand the prizes out at the Sunday morning brunch.

Last year we heard a lot about the new 68000 version of OS-9. This year we may be able to see one in action. That's not official from Microware, but there are signs that it may be ready.

I guess it sounds like I'm advertising the Seminar. I suppose I am. I wouldn't miss it for the world, and I hope I'll see you there.

OFLEX

Just today I received a copy of OFlex. This program runs Flex as a process in an OS-9 Level Two system. I'm afraid it's been too long since I used Flex with any regularity for me to give the program a good workout. Still, I ran a few Flex programs and checked out the interface to OS-9.

I remembered from "The Soul of a New Machine" that Adventure was an important test used on new hardware. I have a version of Adventure which runs under Flex, so I ran through a dozen rooms or so with it and grabbed two or three treasures ... no problem. I compiled a Pascal program using the TSC Pascal compiler with no difficulties except some trouble remembering how to use Flex.

Part of the OFlex package is a program called XCOPY that runs under OFlex. XCopy can copy from OS-9 files to Flex files and back. I tried every combination I could think of and couldn't make it fail. That brings up the one important failing I could find in OFlex; there is no FORMAT utility. I guess FORMAT is too near the hardware to run in what amounts to a virtual machine.

OFlex can read and write Flex disks. It can also format files on an OS-9 disk so the files can be treated as Flex disks by OFlex. The files are accessed through a command called ASNDISK. Using ASNDISK, files can be associated with each disk number (1 through 4). This is a useful feature for Flex. I shudder to think of the problem it would be dealing with a hard disk full of Flex files. With OFlex the hard disk can be broken up into many smaller virtual disks giving manageable bunchs of files to work with.

OFlex isn't reentrant. This is sad, but, as I remember it, many Flex programs change flags and pointers inside Flex. Because it isn't reentrant, each instance of OFlex running under OS-9 needs a full 60K, but, if the memory is available, many users can run OFlex on the same machine. This could be viewed as an easy way of getting multi-user Flex.

OFlex is licensed from TSC and Frank Hogg Labs. As far as I can tell it is regular Flex with modified I/O which feeds into OS-9. It ran the programs I tried flawlessly, but I know of several Flex programs (I've written some myself) which use memory-mapped I/O directly instead of going through Flex. They won't work under OFlex. Anyhow, If you have OS-9 and you wish you

could run most of your old Flex programs, or at least read the old disks, DFlex will do what you need. If you have no particular need for OS-9 but figure DFlex might be an improved way to run Flex, you must be very brave. It is an improvement over regular Flex in several ways, but one day a program you desperately want to run won't work with this mutation of Flex. In any case try DFlex with your software before you rely on it.

NEW MANUALS

I got a stack of new OS-9 Manuals last week. I'm not an authority on most of the OS-9 Manuals, but I've practically memorized the System Programmer's Manual. The new manual is a big improvement over the old one. There is a section on memory management for Level Two and a section on pipes with a few assembly language examples. The Level Two Service Requests are in with the other requests, not isolated in an appendix. Speaking of Service Requests, the manual goes into a good deal more detail than it used to on some of them. The explanation of Chain takes more than two pages. Exit takes about a page and a quarter, as does Intercept.

The new manual contains lots of useful snippets of code demonstrating tricky points. I was particularly pleased to see five chunks of about ten lines each that cover the most obscure parts of an interrupt driven device driver. I believe those chunks of code were taken straight out of the ACIA device driver.

Microware has been producing steadily better manuals for the last two years. The new Systems manual is their best so far. If it had been available last January, I might never have seen a need for this column.

C FUNCTIONS

I have been working on a program to model a problem in distributed systems for a course I am taking. I needed some functions to manipulate floating point numbers as a separate mantissa and exponent. I spent most of an evening fussing around with assembler before I gave up and wrote the functions mostly in C. It was such a frustrating experience that I decided to include them in this column. I wrote frexp and modf to duplicate functions that are part of the UNIX math library.

Frexp returns the mantissa of val as a double less than one, and stores the exponent in the integer pointed to by eptr. The exponent is for a power of two; that is, the number was (val=x*2**exp).

Modf separates a double into an integer part and a fractional part. The integer part is stored at the address in ptr (as a double), and the fractional part is returned (also as a double).

I wrote most of the code for these functions in C because I couldn't do it in assembler. I certainly tried, but Microware C uses lots of internal subroutines and a special static storage location called flacc (floating point accumulator) to do floating point calculations. I had lots of trouble finding the floating point number and returning the number to the caller. As you can see from the programs, my solution was to use C to do everything in modf, and to find val and return a value in frexp.

THE BUTTERFLY

It looks like the Computer Science Department here at the University of Rochester is going to get a computer called a Butterfly. It is named after the network used to connect its processors together. The Butterfly that will be coming here has 128 68000 microprocessors. Each 68000 has at least 512K of memory and, potentially, its own buss. They are all able to read and write one another's memory. I hear that this computer will have the fastest instruction rate in the world. Of course, instruction rates are an almost meaningless measure, but won't that be a marvelous computer to develope parallel algorithms on! It's coming with a UNIX-like operating system, but I can't help but wonder whether it could run OS-9.

DYNASPELL

Last summer at the OS-9 Users Seminar I met Dale Puckett at dinner -- before we were both elected as Users Group officers. I had been a loyal user of Dynaspell, a program written by Dale Puckett, but I wasn't entirely happy with it. In fact I had written a very mixed short review of it in this column. During dinner I made Dale sit through a careful explanation of my criticism of his program, and a long discussion of what I thought a spelling checker should do.

Dale was very patient with me. He even encouraged me to go into more depth about my ideas for the perfect spelling checker. I told him that I would write a new, more complete review of Dynaspell if he would send me a version that deserved fresh consideration. Some months later I got a package from Dale including something pretty close to my dream spelling checker. We went through some iterations working out various problems. Now I owe Dynaspell a review. I have been very slow about writing that review, so let me summarize here. I'll go into more depth another month. Dynaspell isn't perfect, but I haven't been able to find any bugs in the latest version. It is much faster than the early version I had. It is able to look near misses up in its dictionary and suggest corrections when it suspects a spelling error.

My remaining complaint about Dynaspell is that the new features don't go far enough. The "look up" feature isn't as selective as a would like. It often finds more possible spellings for a word than it can fit on the screen. On the other hand it sometimes doesn't search widely enough to find the correct spelling for me. I

also wish it would give me the features of a screen oriented text editor when it finds a spelling editor. Dynaspell has a mode in which spelling errors can be viewed in context, but the context it shows is a screenfull of the document up to and including the word in error. I would like to be able to move forward and backward throught the document, and to change words other than the one in error.

I used my early copy of Dynaspell because I need a spelling checked badly and it was the best I had. I use it more often and more happily now. It is one of the best spelling checkers I know: mainframe programs included.

A NICE EXPERIENCE

Early last summer I bought a TeleVideo 970 terminal. They were just becoming available on the market; in fact, I had a hard time finding one. It seems the boat bringing a large shipment in from overseas had sunk. I'm not certain I believe that, but it was definitely difficult to find one to buy. I finally found one, got it home, and started using it. Nice terminal. Big screen, nice keyboard. Almost too flexible.

After about a week I started finding bugs. A few commands didn't work right. I called the number in the manual and talked to an engineer. The next day I got a package via Federal Express with new firmware ROMS. That wasn't the end of the problems with the terminal. I'm one of those annoying people who reads the entire manual then tries all the strange combinations of commands just to see what they will do, and the 970 has a manual about two thirds of an inch thick. The last time I called them I told them that I needed a feature which was documented in the manual, but which the errata with the manual said was not implemented (downloadable fonts). Without a complaint they sent me a whole new logic board which supports that feature.

I don't think I would recommend the TeleVideo terminal to most OS-9 users. The terminal costs over a thousand dollars. That makes it hard to justify when a adequate terminal only costs five or six hundred dollars. For those who take terminals seriously, it is worth what it costs. It supports ANSI standard and VT52 control sequences, and includes about every feature I can imagine except full graphics (they say that's coming).

The best thing about the 970 is the excellent support TeleVideo gives. Many large vendors seem to lose interest after they sell you their product. TeleVideo has gone out of their way for me again and again.

TRICKS FOR LEVEL TWO

I just learned about OS9P3 in the new OS-9 System Programmer's Manual. I have often wished for an easy way to add System Service Requests to OS-9. Under Level One, it isn't too hard, but under Level Two it has required either slight of hand or very strange practices. Only modules running in the system address space can add Service Requests, but OS-9 doesn't include a way to run a process in the system address space. I have run device drivers and file managers just to add Service requests, and considered renaming OS9P2 as OS9P21 and adding my own OS9P2 which will link to and call OS9P21.

Microware has included something like that last trick in Level Two. After OS9P2 is finished initializing (all it does is set up a list of Service Requests) it tries to find OS9P3. There is no OS9P3 unless the user adds it to the boot file, so it generally fails to find the module, but if it finds OS9P3 it executes it as a system module. This opens up lots of interesting possibilities.

Other interesting possibilities are suggested by the SS.SIG and SS.Relea SetStat codes. SS.SIG instructs OS-9 to send a specified signal when data is ready from a path. The easy use for this is to wait for output from several paths at once. This is especially good for things like "modem" programs that need to wait for input from two paths simultaneously. Without this SetStat the only way to handle that problem was to poll both paths.

It isn't difficult to write a program that polls a number of paths. In fact, polling is the way most of the more primitive microcomputer operating systems work. The problem with polling is that it wastes tremendous amounts of CPU power. I seldom type faster than 2 characters per second. If a program has to poll for my input it will look for something to read thousands of times before it gets anything.

With SS.SIG it should be possible to do a couple of SetStats and wait for a signal. While an OS-9 program waits it uses essentially nothing but memory. This should make modem programs and other programs with similar problems much more efficient.

The other use I can think of for SS.SIG is to solve the problem that devices can't be preempted. If you have a system with more than one terminal you have probably noticed that if you send a message to another terminal, the message waits until the user at the other terminal types a carriage return. That's because there is a program (e.g. the shell) trying to read from that terminal. Until the read is finished OS-9 won't allow any process to write to it. SS.SIG gives us a way to break that deadlock by not leaving a read active.

I have included a trivial program which demonstrates the use of the SS.SIG setstat with this Column. It doesn't do anything useful -- just copies lines from standard input to standard output. The exciting thing is that it works! I ran tstssig on one terminal; typed a few lines into it to make certain that it worked; left it at its prompt, and went to my other terminal. I typed

```
Echo Hi there >/term
```

on the other terminal and it appeared immediately on the terminal running tstssig. I

went back to the terminal running tstssig and typed a blank. The blank caused a signal to be sent to tstssig letting it proceed to the I$ReadLn. Once the read was "up" /term was locked. I tried to send another message to /term and found that I had to wait until I typed a carriage return on /term before the message was delivered and the echo command completed.

I wonder whether the SS.SIG trick should be used as a matter of policy when long waits for input are expected.

```
                                nam   tstssig
                                ttl   Test SSIG set stat
          *-------------------------------------------------------*
          *          Test SS.SIG SetStat Service request.         *
          *    This program will copy lines from standard input   *
          *    to standard output without tying the device        *
          *    used for standard input up with a read, or using   *
          *    excessive amounts of CPU time by polling the       *
          *    standard input path.                               *
          *                                                       *
          *    tstssig has no practical use that I can think of.  *
          *-------------------------------------------------------*
                                IFP1              use os9defs
                                ENDC
     0011             Type     set   Objct+Prgrm
     0081             Revs     set   ReEnt+1
     0001             StdOut   set   1
     0000             StdIn    set   0
     0004             SSCode   set   4           code used to indicate input wa
     0064             LineSiz  set   100
     00C8             Stacksiz set   200
     0000 87CD0072             mod   TstLen,TstNam,Type,Revs,Entry,MemSize
     000D 54737473    TstNam   fcs   /Tstssig/
     0014 01          Edition  fcb   1
     0015 3D3DBE      Prompt   fcs   /==>/
     0003             PromptL  equ   *-Prompt
          *-------------------------------------------------------*
          *                    Static Storage                     *
          *-------------------------------------------------------*
   D 0000             IntNo    rmb   1           Save the signal from the trap
   D 0001             Line     rmb   LineSiz     Storage for a line to echo
   D 0065                      rmb   Stacksiz
   D 012D             MemSize  equ   .


     0018             Entry
          ********
          *  Set up signal intercept trap
          *
     0018 308D0050             leax  Trap,PCR    Address of Interrupt trap code
     001C 103F09               OS9   F$Icpt
     001F             Loop
     001F 308DFFF2             leax  Prompt,PCR
     0023 108E0003             ldy   #PromptL
     0027 8601                 lda   #StdOut
     0029 103F8A               OS9   I$Write     Write the prompt
     002C 2536                 bcs   Error
     002E             StrtRead
     002E 8600                 lda   #StdIn
     0030 C601                 ldb   #SS.Ready
     0032 103F8D               OS9   I$GetStt    any data ready?
     0035 2516                 bcs   DoSSIG      No; wait for a signal
     0037             DoEcho
     0037 3041                 leax  Line,U
     0039 108E0064             ldy   #LineSiz
     003D 8600                 lda   #StdIn
     003F 103F8B               OS9   I$ReadLn    Read a line
     0042 2520                 bcs   Error
     0044 8601                 lda   #StdOut
     0046 103F8C               OS9   I$WritLn     and echo it back out
     0049 2519                 bcs   Error
     004B 20D2                 bra   Loop        Go prompt for the next line


     004D             DoSSIG
     004D C61A                 ldb   #SS.SSIG    setstat function code
     004F 8E0004               ldx   #SSCode
     0052 103F8E               OS9   I$SetStt
     0055 8E0000               ldx   #0
     0058 103F0A               OS9   F$Sleep     Sleep until an interrupt comes
     005B D600                 ldb   IntNo
     005D C104                 cmpb  #SSCode
     005F 27CD                 beq   StrtRead
     0061 43                   coma              set carry
     0062 2000                 bra   Error
```

```
00073   0064            Error
00074   0064 C1D3               cmpb  #E$Eof
00075   0066 2601               bne   Exit
00076   0068 5F                 clrb            EOF isn't an error
00077   0069            Exit
00078   0069 103F06             OS9   F$Exit
00079        ***************
00080        *  Trivial Interrupt trap
00081        *
00082   006C            Trap
00083   006C E7C4               stb   IntNo,U     save the interrupt code
00084   006E 3B                 rti
00085   006F 8AD34F             emod
00086   0072            TstLen  equ   *
```

FREXP

```
 1 double
 2 frexp(val,iptr)
 3  double val;
 4  int *iptr;
 5  {
 6     register double *rp;
 7     int exp;
 8
 9     rp = &val;
10     /*  at this point U contains  the address of val */
11 #asm
12  ldb 7,U get C exponent
13  addb #128
14  sex
15  std ,S save exp
16  lda #128
17  sta 7,U
18 #endasm
19
20  *iptr = exp;
21  return(val);
22  }
```

MODF

```
 1 /* modf returns the positive fractional part of val.
 2    and stores the integer part in the double pointed to
 3    by ptr.
 4 */
 5 #define MAXLONG 134217727
 6
 7 double
 8 modf(val,ptr)
 9  double val, *ptr;
10
11  {
12     double tmp;
13
14     if(val > MAXLONG)
15        {
16           *ptr = val;
17           return(0.0);
18        }
19
20     tmp = (long)val; /* truncate to int by coercion to long*/
21     *ptr = val - tmp;
22     return(tmp);
23  }
```

COLUMN SIXTEEN

STANDARDS

Several months ago I mentioned Smoke's special version of OS-9 Level Two in this column. The questions I posed about its compatibility with Microware OS-9 stirred up a lot of commotion, but thanks to Don Williams' intervention no blood was shed. Smoke Signal has agreed to give customers a choice of the accelerated Smoke version of OS-9 or the Microware version. I think Smoke Signal deserves much credit for offering their customers this alternative. Some, perhaps most, people who use OS-9 need extra speed enough to take the risk associated with a version of OS-9 not just like everyone else's. Cautious people (like me) can ask Smoke to send them the Microware version of OS-9.

It probably seems strange that I, a person who likes to fuss with operating systems, should get so worked up about changes to OS-9. After all, I enjoy adding non-standard features to OS-9; I even publish some of them in this column.

Let me examine the question of standards from a few points of view. There are things to be said for ignoring standards: mostly that ignoring existing standards is the way new, improved ones are born. However, consumers find standards convenient, and producers typically find standards crucial.

Good examples of standards that beg to be ignored can be found in the busses invented in the early days of microcomputers. Engineers I know agree that the S-100 bus is poorly designed. They would love to be able to make a few changes to its specifications. Our own SS-50 bus has gone through some evolution, but extending the address space beyond a megabyte will require further changes to the standard.

I don't know hardware very well, but I imagine electrical engineers learn to work around standards about the same way programmers do. Strict adherence to standards even when they have been outgrown often results in a "kludge." Either the old code is left there and a new structure built on top of it, or it is entirely replaced with code that does things "right," and adherence to the standard is added on as a special case, something ugly hanging off the side of the new idea. Both of these solutions look like poor design.

IBM is a good example of a company, in fact an industry, caught on a horns of a standard. Years ago they invented the 360 architecture, a computer architecture that they used for all their computers. The idea of having a line of compatible computers caught on nicely. Later, they extended the 360 architecture to include virtual memory and a few other goodies, giving the 370 architecture. It was also quite successful. Customers seemed to appreciate being able to move to more powerful computers without rewriting any software. Most recently, IBM produced XA, an extension of the 370 architecture which 370 customers can move to relatively painlessly.

While these hardware changes were going on, operating systems were being improved. Programs that ran under MFT (an old operating system for 360s) should run with no important changes under the latest version of MVS. This level of compatibility exists only because IBM has stuck grimly to its standards. This practice has brought them success, but not critical acclaim. I know operating system experts who pretend to feel sick when MVS is mentioned -- with some justification. That operating system contains layer after layer of history. In some places the complexity is so thick it is practically impossible to figure out what the programmer was trying to do. I imagine that, if the effort which goes into adapting MVS and 370 architecture to modern needs were directed toward designing new hardware and software, the result would be much faster and more useful than IBM's current 370-type products. I bet there are numerous engineers and computer scientists at IBM who yearn to junk the old standards in favor of something better.

Standards like S-100, SS-50, and 360/370 architecture have tied manufacturers to dinosaurs. They can't depart from their standards without hurting, and perhaps losing customers. The big computer and software manufacturers probably have mixed feeling about standards. The consumers of their products feel about the same way.

It is hard to resist a sexy new computer or piece of software. The non-standard offerings are frequently faster and in various ways better than the more conservative ones. The problem is that non-standard computers or operating systems are risky. The excitement of being the only person in the state with some fast, elegant operating system fades fast when you have troubles with software availability.

We are lucky to be using hardware and software that have good standards. CoCo users are dealing with only one vendor and one machine. It is a shame Tandy didn't decide to use the same disk format all the other OS-9 systems do, but at least that problem is well known. It should be easy to exchange software and hardware between CoCos.

The SS-50 bus is also a good standard which has been carefully respected by the vendors that support it. I ran my Gimix disk controller board with a SWTPc CPU board and memory boards from three different sources for about a year with no trouble. If all those manufacturers hadn't respected the SS-50 standard, I couldn't have done that.

Microware OS-9 is solid across all the machines I know of. It is even possible to move from Level One to Level Two without changing software (provided the programs were written to appropriate standards). An OS-9 user can trade from a CoCo to a Helix to a Gimix III system without rewriting any programs except where they use special I/O features of each computer (like graphics on the CoCo). A software house can use their Gimix III system with its high speed and debugging facilities to develop software which will run on a CoCo. Usually we can order software without paying attention to the manufacturer of our machine.

The standards within OS-9 are as important as the interface to user programs. The device drivers and other system modules I include with the column occasionally should run on any OS-9 system with suitable hardware. I rely on Microware to stick with the interfaces between system modules that they have specified. If I ever find the money for it, I will be able to buy a graphics board for my system. If the vendor is selling it for the OS-9 market, it will come with software to hook it into my system. That software will almost certainly work because its author wrote it and tested it on a system with the same interfaces between system modules as mine.

Programmers have the most to gain from carefully followed standards. If someone buys a program that doesn't run on his computer, he will complain -- maybe return the program. This is a problem for the consumer, but for the author of that program it is a disaster. Imagine what it would feel like to spend thousands of hours creating a masterpiece of a program, then discover that it would only run on a few of the computers you had counted on for your market. With Microware OS-9 on any supported computer a programmer can be confident that that won't happen.

Programmers would like to see more standards in the OS-9 world. I have wished and worked for a standard terminal interface for a year now. It is a shame that each programmer who wants to sell his programs has to invent a way to adapt his program to whatever kind of terminal it might encounter. A standard here would save days in program development time for each program that used it, encourage more programmers to use terminal features supported by the standard, and give purchasers confidence that a program would work with their terminals.

STANDARDS THAT ARE THE USER'S RESPONSIBILITY

If your system comes to you nonstandard in some way, you should complain to the person responsible. Once you have it, it's your baby. You can generate additional standards to simplify your system, or let chaos grow in your system.

Several areas come to mind as good places to institute standards. Directory structure is an especially good place to devise a standard. If you write a lot of programs, you may need a naming convention. A set of standards for documentation might help keep it up-to-date.

There are two policies that can be used to guide the construction of directory structures. The directories can be arranged by what the contents are (programs, text, spread sheet info.), or by what they are for (sort programs, household, User Group files). Each method has its charm. I use both, each where it seems appropriate, but I wish I had decided early which way I wanted to go and stuck with it. Sometimes I have to search for minutes before I find a file I haven't used in a few months.

It is a good question whether documentation for a project should be in the same directory with the source of programs for that project, in a sibling of that directory dedicated to documentation for several projects (or just for a single project), or in a directory which is the child of the directory with the source in it.

Some people think that directories should contain either only other directories, or only data files. I don't think I like that idea, but I can see some value in it.

Program names deserve serious thought. The shorter they are the faster they can be typed. It is easier to type L than LIST, but the shorter names are the more cryptic they become. LOOK or LOGOFF could also be abbreviated L. It has to be clear what the abbreviation stands for. It makes sense to me to give short names to frequently used programs. The names of the commands will stay fresh in the mind if they are frequently used even if they aren't very mnemonic. Less frequently used programs should have longer names both to save short names for more frequently used commands, and to jog the memory about their function.

THE USERS GROUP

The OS-9 Users Group plans to submit a list of "requirements" to Microware at the OS-9 Seminar this summer. If you have spotted a flaw in Microware's software that you think is of general interest, or would like to suggest that a new feature should be added to one of their products, this would be a good way to bring it to Microware's attention. Submit your suggestion in writing to the Users Group early enough that it will reach us at least a few weeks before the Seminar. Please keep it to about a page or less. We will have copies of all the suggestions available at the Users Group booth at the seminar. The suggestions will be discussed at the Users Group meeting and those about which we can reach a consensus will be given to Microware. We will try to get an official response to each suggestion from Microware -- something like: impossible, not interested, will do, wonderful suggestion, or already done.

COLUMN SEVENTEEN -- THE FIRST STEP INTO OS-9

There has been some call recently for information for the beginning user of OS-9. Color Computer users new to OS-9 feel swamped by the number of details involved in the operating system. This column is an attempt to make OS-9 seem simpler to new users.

The OS-9 operating system has started to develop a reputation for complexity and obscurity -- in other words, user hostility. It is an unjust accusation. The thing that makes OS-9 appear confusing is the way it is presented. There are many subtle features in the operating system, and a large array of utilities. The manuals that come with it could help but don't. The OS-9 manuals were written as reference manuals, not tutorials. They drop everything on you at once. A new OS-9 user who is experienced with computers or very brave should read the manuals, wrap his mind around the whole thing, and sit down at the computer to enjoy OS-9. That is the quick, brute force, way to learn OS-9, but if it doesn't work for you, I recommend a gentler approach.

My copy of CoCo OS-9 includes about fifty commands. All these commands are important to at least some people, but most of them are only confusing to to new OS-9 users. The entire English language includes more than a hundred thousand words, but most people only use fewer than twenty thousand of them, and it is possible to communicate with a vocabulary of a thousand words or less. Operating systems like Unix and OS-9 are much like English in that respect. Of all the commands available under OS-9 about a dozen are really necessary. The bare minimum set of OS-9 commands are:

- backup
- copy
- del
- dir
- edit
- format
- free
- list
- rename
- shell

The shell is the program which processes the commands you type into OS-9 and runs the other commands. Several commands are built into the shell. They are:

- chd
- chx
- ex
- w
- kill
- setpr

The only shell commands that you really need to know are cho and chx, If you mean to do assembly language programming you will also need:

- asm
- debug

If you will be using BasicO9 you will need:

- BasicO9
- RunB
- GFX

Of all these commands there are four that need explanation especially badly. Format needs to be discussed because it is dangerous; if it is used carelessly it can destroy important information. BACKUP is a relatively fast way to copy an entire disk (it is a good thing to get into the habit of doing this); perhaps a careful discussion of BACKUP will encourage people to use it more. Explaining DIR is a good excuse to say a few things about directories: an important feature of OS-9. CHX ano CHD also relate to directories, and seem straightforward. What they are supposed to do matters less to a person with a OS-9 on a small computer than their unofficial side effects.

FORMAT

The format command is the first one to use. Until a disk has been formatted it is unusable to OS-9. The format command writes a pattern on the disk which marks the disk off into sectors (which amount to pigeonholes for OS-9 to store data in). After writing the pattern format checks the disk to make certain the pattern is recorded correctly on the disk. If it isn't, format will note that the sectors where the errors occurred are faulty, and those sectors won't be used to store data. Format also writes some information which will be used to manage files on that disk. In the process of doing all this the format program completely erases the disk. If the disk is fresh out of a box of new disks you can feel certain that there is nothing on the disk that you care about, but, if it is one you are recycling, be careful. After format is started any data that was on that disk is gone forever.

Put the disk you want to format in the drive you aren't using for the system disk (I'm going to assume you have your system disk in the drive OS-9 calls /DO, and the disk you want to format in drive /D1). Invoke the format command by typing FORMAT /D1 at the OS-9 prompt. The command line should look like:

```
OS9:FORMAT /D1
```

to which you should get the response:

```
COLOR COMPUTER FORMATTER
FORMATTING DRIVE  /D1
Y (YES) OR  N (NO)
READY?
```

This is format giving you a chance to change your mind. It is also a way for you to format disks if you only have one drive, by asking format to format the disk in drive /D0 and replacing the system disk with the disk you want to format in at this point. In either case double check that you are about to format the correct disk. If you want to be especially safe take your system disk out of drive /D0 at this point even if you are formatting the disk in drive one. There is no danger of format writing on the wrong disk, but you can't be too careful. If you reply N to the READY? prompt format will quit immediately leaving the disk intact. If you reply Y, there will be a pause (23 seconds on my CoCo), then format will prompt you for a name for the disk. The prompt will look like:

```
DISK NAME:
```

At this point enter the name you have assigned to the disk. The name can be up to 32 characters long and may include blanks. Follow the disk name with an ENTER. Format will now check the disk. As it checks each track on the disk it will write the track number to the screen in hexadecimal (base 16). If you have a thirty five track drive, the numbers will be from 000 to 022. Then format will print the message:

```
NUMBER OF GOOD SECTORS: $000276
```

If the number is smaller than 276 (a base 16 number which is 630 in decimal) some sectors were faulty.

If you want to demonstrate to yourself that format did something to the disk try the FREE command on the new disk. Enter the command FREE /D1. The command line should look like:

```
OS9:FREE /D1
```

The response should be something like:

```
disk name CREATED ON 84/01/24
CAPACITY: 630 SECTORS (1-SECTOR
CLUSTERS)
620 FREE SECTORS, LARGEST BLOCK
620 SECTORS
```

Where "disk name" in the first line of the response will be the name you gave the disk when you formatted it.

BACKUP

The next command to use after the format command is BACKUP. It is crucial to have a backup copy of each software distribution disk you have. If you make an error that damages the only disk with an significant piece of software on it you will have to wait until you can get a replacement for the disk before you can use your computer again. Even if the time wasted waiting for the replacement disk isn't important to you, consider that replacement disks cost money.

Backup is a relatively fast way to create an exact copy of a disk. It has many options, but the simplest way to use the command is to just give the command BACKUP. The command line should look like:

```
OS9:BACKUP The response will be:
READY TO BACKUP FROM /D0 TO /D1
?:
```

At this point put the disk you want to copy in /D0 and a formatted disk which has nothing you want to keep on it in drive /D1. Then check the disk in /D1 ... BACKUP will erase anything that's on that disk. When you are certain everything is OK type Y. Now BACKUP will double check with you by telling you the name of the disk in drive /D1. The message will look like:

```
THE DISK
 IS BEING SCRATCHED
OK ?:
```

If you reply Y to this, the backup from the disk in /D0 to the disk in /D1 will take place. The disk in /D1 will become an exact copy of the disk in /D0 right down to the disk's name.

The BACKUP command takes what seems like a long time to run. There are two things that can speed it up. One is to use the -V option which prevents the copy from being verified. I don't suggest that anyone use this option. The other way to speed BACKUP up is to instruct OS-9 to give it extra memory to run in. BACKUP can use extra memory to run more quickly. BACKUP ran for one minute 58 seconds when I started it with the command line:

```
OS9:BACKUP
```

Normally BACKUP uses 19 pages of memory. If you give it more -- say 100 pages -- with the command line:

```
OS9:BACKUP #100
```

it runs in one minute 48 seconds. It is also quieter because the heads on the disks don't load and unload as often.

DIR

The command which tells you what files are one your disks is the the Dir (short for directory) command. If you just type DIR after booting OS-9 you will get a response like

```
DIRECTORY OF . 23:55:08
OS9BOOT   CMDS      SYS
DEFS      STARTUP
```

This means that you are listing the current directory which is known by the pseudonym "." at 11:55:08 in the evening. The files in that directory are OS9BOOT, CMDS, SYS, DEFS, and STARTUP. Now, in fact only OS9BOOT and STARTUP are normal files, the other three files are subdirectories. Subdirectories are such an interesting topic that they were the subject of their own column some months ago, and won't be covered any more than absolutely necessary

here. To find out more about the files than their names use the command DIR E.

```
OS9:DIR E
```

which will respond:

```
 DIRECTORY OF .  23:59:57

CREATED   ON     OWNER      NAME
ATTR        START           SIZE
==============================
83/06/02  1921      0    OS9BOOT
------WR           A        3032
83/06/02   1956       0     CMDS
D-EWREWR          3C         6A0
83/06/02   2002       0      SYS
D-EWREWR         164          A0
83/06/02   2002       0     DEFS
D-EWREWR         17F          C0
83/06/02  2003      0    STARTUP
----R-WR        1F5          E
```

then it will stop because the screen is full. When you are ready to continue hit any key ... I usually press the space bar. That was the end of the directory, so all you get after you let the output continue is a few blank lines and a new OS9 prompt.

Two of the fields in the DIR E output are of no special interest until you become an advanced OS-9 user: OWNER, and START. The first two fields for each file are the date and time the file was created. The date is in the usual YY/MM/DD format and the time is in HHMM format with hours ranging from 00 to 23. The attributes field contains information about what the file can be used for. The main thing now is that files with a D as the first character in the attribute field are directories. Files with a dash as the first character in their attribute field are normal files.

The other option which can be used with the DIR command is X. The X option is a short hand way to get the directory of the execution directory; that is, the directory OS-9 searches for programs, like the commands, you ask it to run. The command line:

```
DIR X
```

will give you a rather long list of all the files in your execution directory. If you haven't written any of your own programs, this will be a list of all the commands and utility programs which came with OS-9. You will probably have to press the space bar in the middle of the output of this command. It is more than one page long.

CHX AND CHD

Chx stands for Change Execution Directory, Chd for Change Data Directory. OS-9 expects to find all commands, whether they are part of the operating system or something you wrote, in the execution directory. All files that you don't mean to execute are looked for in the data directory. (There are ways around both of these restrictions, but let's skip that for now.) After you boot OS-9 you will find that the execution directory is /D0/CMDS and the data directory is /D0. If you have a second drive (I have been assuming that you do) you will probably want to use that for data. The command:

```
CHD /D1
```

will cause all future references to data files to look for them on /D1.

To speed OS-9 up, the location of the directory file on the disk is kept in memory. This leads to the side effect of the Chd and Chx commands. When you read the directory OS-9 goes directly to the directory's location on disk and starts reading ... imagine what would happen if you fooled OS-9 by changing disks. You change disks and type a command like

```
LIST FOO
```

or even just DIR. Your operating system will start reading where the directory is supposed to be. Since the disk with a directory at the selected spot is sitting in its envelope and some other disk is in the drive, OS-9 will find something unexpected where the directory was. The result could be any of several error messages. The solution to this problem is to always give OS-9 a chance to find the directories on a new disk by giving it Chd and Chx commands as necessary when you change disks.

There is one last tricky thing about the Chx/Chd commands' special use. If you keep things simple it will seem that you only need to use the Chx command, but this is just a special case. I suggest that you learn how to make directories and use them when you can, but, until you start using them, the new disks you use to store data will only have the directory FORMAT automatically creates (called the "root directory"). The root directory is always at the same location on a disk. Because of this special fact about the root directory OS-9 is always able to find it, and changing disks that only have the root directory on them won't cause any trouble. The execution directory is usually not the root directory, so this special case doesn't generally apply to it.

The set of commands I have mentioned in this column might be considered a "starter set" for OS-9. The dozens of commands I left out are certainly worth learning, but you can get OS-9 working with these few.

OOPS

I neglected to mention a few months ago that OFlex as reviewed in this column is available only from Gimix. Richard Don, the salesman for Gimix, explained the geneology of OFlex to me. It is Flex by TSC adapted by Richard Hogg to run under OS-9. Gimix provides enhanced disk Device Drivers to support Flex's requirements, and made some enhancements to Richard Hogg's design. Anyone who takes out licenses from TSC and Richard Hogg can sell OFlex, but the version I reviewed has features added by Gimix.

COLUMN EIGHTEEN

MY LIFE

I'm afraid this month's column will be a little short. I just bought a house. Nothing major wrong with it, but I'm living in the first floor while I fix up the upstairs. Piles of boxes are everywhere, and it seems like everything I need is in a box at the center of an unknown pile. This disorder has not helped me get a lot of computing done.

I don't mean to turn this column into a diary, but there are a few other important items. A kitten is helping me write this. I got him to help make my house seem homelike, but he likes to help type. I enjoy his help, but I hope he will switch to sleep-in-the-lap mode soon.

This fall I will finally become a full-time graduate student. I have been studying Computer Science part time for years, but it seemed that the field was moving ahead faster than I was learning it. It is a scary business going back to college after being a working man for years, but I'm fairly quivering with eagerness. I have one more column to write as a free man, then I will be a student. I think I can get permission to keep writing this column. I hope my studies add some spice to my writing.

NON-STANDARD HARDWARE

A fair amount of the OS-9 mail I get asks about special versions of OS-9. Many people have old SWTPc systems they would like to run OS-9 on. There are also a few people with home-brew 6809 systems who'd like to port OS-9. The news for these people is mostly bad.

There used to be a SWTPc version of OS-9 Level One, but I don't think it is sold any more. OS-9 Level Two is sold only through hardware manufacturers, and SWTPc hasn't licensed it. If you have your own home-brew design, you can license OS-9 from Microware, but the price is ridiculous unless you mean to sell it.

Two years ago (or more) Microware used to sell a generic version of OS-9 Level One. You could buy it directly from Microware and adapt it to whatever system you wanted. I guess a few people must have purchased that version of OS-9 and tied up Microware's hotline for days with the trouble they had getting it going. The effort they had to put into helping people use the generic OS-9 was more than Microware could afford, so they dropped the product. This policy seems to be mainly a way of avoiding piracy. The theory is that if the people who sell the hardware have to buy the right to sell OS-9, they will see to it that people buy an operating system instead of stealing it.

Officially there is no way to get OS-9 for your SWTPc, or home-brew machine. Unofficially, there are ways. An important features of OS-9 is its hardware independence. The clock and I/O devices are handled by drivers. The interfaces to the drivers are general enough that any reasonable hardware can be accommodated. Microware will sell the source to several device drivers and a few clock drivers. With a copy of OS-9 for any machine, a working OS-9 to build the new OS-9 on, and a collection of source from Microware it should be possible for an experienced programmer to adapt OS-9 to any 6809-based machine I have heard of.

It isn't hard to buy a copy of OS-9 to customize. Try a few manufactures. When I was building crazy systems I did a lot of business with AAA Chicago Computing; they might be able to help you.[6] You don't care what version of OS-9 you get unless you can get one that is already partly compatible with your system. You'll have to write a clock driver, a disk driver, and, if you use an unusual serial chip, a SCF driver. If you want to adapt Level Two, you'll have to buy a version of OS-9 that is designed for the memory management hardware you have. Memory management is done in the OS-9 kernel (OS9P1). It isn't easy to adapt without lots of source code ... the kind of source Microware sells as part of an OEM license ... very expensive.

If anyone has OS-9 running on unsupported hardware let me know. Microware doesn't officially want to support you, but they might not object if we set up a function of the Users Group to help you out. If there is enough interest, maybe we can find a reliable source of adaptable OS-9. In any case, I'll report any tips you send me in this column.

DIRECTORIES AS FILES

A directory is a special type of file, If they are handled correctly, they can be opened and used without much trouble. If you try to list or dump a directory file, you will have trouble. Directory files can only be opened using the directory access mode; and Dump, List, Copy, and most other OS-9 utilities don't use this mode.

The easiest thing to do with a directory is to simply read it and copy it to standard output. The program called DList copies the current directory to standard output. You can see the contents of the current data directory by assembling DList and typing

```
OS9: DList ! Dump
```

Directories contain many unprintable characters, so if you don't use Dump to format the output you will get gibberish on the screen. You may even make your terminal do strange things.

---

[6] It occurs to me that Radio Shack sells the least expensive OS-9 around. Microware has a few versions of Level One for Motorolla systems that they can sell. The Radio Shack OS-9 has a non-standard disk format that you can avoid by buying the more expensive Motorolla software.

I have a directory with only the file containing DList in it. I ran DList with the command line:

```
OS9:DList ! Dump >tmp
```

The contents of Tmp are listed in Figure 6.

```
Addr  0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0 2 4 6 8 A C E
----  ----  ----  ----  ----  ----  ----  ----  ----------------
0000  2EAE 0000 0000 0000 0000 0000 0000 0000  ................
0010  0000 0000 0000 0000 0000 0000 0000 01B9  ...............9
0020  AE00 0000 0000 0000 0000 0000 0000 0000  ................
0030  0000 0000 0000 0000 0000 0000 0000 054D  ...............M
0040  746D F000 0000 0000 0000 0000 0000 0000  tmp.............
0050  0000 0000 0000 0000 0000 0000 0000 0765  ................
0060  444C 6973 F443 4830 B700 0000 0000 0000  DListCH07.......
0070  0000 0000 0000 0000 0000 0000 0000 0763  ...............c
```

Figure 6: Hex dump of a directory

Each entry in the directory takes two lines in the dump. The first two entries are self-referencing. The .. entry is first; the name ".." (2EAE) is at the beginning of the entry. The disk address of the file descriptor (0001B9) for the parent of this directory is at the end of the entry. The second entry is for "." (AE) which is the alias for this directory. The address associated with that (00054D) points to the file descriptor for this directory.

The entry for tmp is for the file I put the dump into. The last entry looks like it is for DListCH07, but if you look at the hex part of the dump you will see that the high bit of the "t" is on, meaning that it is the last character in the string. The characters "CH07" are an artifact of a previous use of the entry for the file SCRATCH07.

DList could be changed to edit the contents of the directory before passing it to standard output. The first two entries are always for "." and "..". There is usually no need to notice them. There can also be null entries in the directory. When a file is deleted the first byte in the directory entry is set to $00 making it into a null entry. DList could check for null entries and suppress them.

DList2 is an enhanced version of DList. It uses I$Seek to skip the first two entries, then copies all entries that don't start with $00 to standard output.

The next feature to add would be formatting the output so it could be read without using Dump. The address of the file descriptor isn't likely to be worth seeing often, so the final program, ld, just prints the file names. A useful directory list program needs to be able to list the contents of directories other than the current default so I added that function. Ld can take a directory name on the command line. It wouldn't be too hard to add the x option by opening the directory with the execution attribute, but that's a function I didn't add. The program determines whether a directory name was given by checking the length of the parameter area. If the parameter area is only one byte long, it only contains a carriage return, otherwise it contains a directory name terminated with a carriage return.

It would be nice to add the "e" option to the ld command, but an extended directory involves lots of numbers and dates. The code to format all that information would make a long program. Instead, I have written a BasicO9 program that takes the output of DList2 and generates a more extensive report. There is room for lots of improvement in DFormat; I only print the file name, creation date, last modified date, and file size, and I don't sort the list in any special order. Improvements like these are, as they say, "left for the reader."

Each directory entry contains the disk address of the file descriptor sector for the file. The file descriptor contains all the interesting information about a file. We need to read the file descriptor, but all we know is its disk address, and the only way to get at a particular sector on a disk is with physical-sector I/O. Normally physical-sector I/O is done by opening a device; e.g. dump /D0@. Since there is no easy way to find out the name of the drive the directory is on, the /D0@ type of trick isn't useful. There is an interesting variation on physical-sector I/O which I haven't been able to find documented anywhere. If you open the file @, it will open the drive the data directory is on for physical I/O. If you open it for execution, it will open the drive with the execution directory on it for physical I/O.

Since DList2 is feeding this program, and DList2 can only read the current data directory, DFormat assumes the directory is on the same disk as the data directory.

I used a useful trick from the UNIX ls command. Directory files are indicated by a "/" after them in the listing from DFormat.

If you have a UNIX-like sort program available the combination of DList2 and DFormat can be made even more useful. By sorting on the various fields in the output from DFormat you can get the listing alphabetically by name, by increasing size, or in chronological order.

If you have Run8 type DFormat in, save it, and pack it. Then use it with a command line like:

```
OS9:DList2 ! DFormat
```

If you don't have RunB, you'll need to use something like:

```
OS9:DList2 ! basic09 DFormat
```

DLIST PROGRAM

```
Microware OS-9 Assembler 2.1   07/13/84 12:07:52                Page 001
DList - List the Current Directory

00001                                   nam   DList
00002                                   ttl   List the Current Directory
00003                                   IFP1
00005                                   ENDC
00006    0011                 type      set   PRGRM+OBJCT
00007    0081                 Revs      set   REENT+1
00008    0000 87CD0043                  mod   MEnd,Name,Type,Revs,Entry,Memsize
00009
00010 D  0000                 DPath     rmb   1           Directory path number
00011 D  0001                 Buffer    rmb   32          buffer for directory entries
00012 D  0021                 Stack     rmb   200
00013 D  00E9                 Memsize   equ   .
00014
00015    000D 444C6973        Name      fcs   /DList/
00016    0012 01              Version   fcb   1
00017    0013 2EA0            Dirname   fcs   /. /
00018
00019    0015                 Entry
00020    0015 8681                      lda   #DIR.+READ.
00021    0017 308DFFF8                  leax  Dirname,PCR
00022    001B 103F84                    OS9   I$Open
00023    001E 251D                      bcs   Error
00024    0020 9700                      sta   DPath
00025    0022                 RLoop
00026    0022 3041                      leax  Buffer,U
00027    0024 108E0020                  ldy   #32
00028    0028 103F89                    OS9   I$Read
00029    002B 250B                      bcs   TEof
00030    002D 8601                      lda   #1          Std output
00031    002F 103F8A                    OS9   I$Write
00032    0032 2509                      bcs   Error
00033    0034 9600                      lda   DPath
00034    0036 20EA                      bra   RLoop
00035    0038                 TEof
00036    0038 C1D3                      cmpb  #E$EOF
00037    003A 2601                      bne   Error
00038    003C 5F                        clrb
00039    003D                 Error
00040    003D 103F06                    OS9   F$Exit      return
00041    0040 69C250                    EMOD
00042    0043                 MEnd      equ   *

00000 error(s)
00000 warning(s)
$0043 00067 program bytes generated
$00E9 00233 data bytes allocated
$223F 08767 bytes used for symbols
```

DLIST2 PROGRAM

```
Microware OS-9 Assembler 2.1   07/13/84 12:08:17                Page 001
DList2 - List the Current Directory

00001                                   nam   DList2
00002                                   ttl   List the Current Directory
00003                                   IFP1
00005                                   ENDC
00006   0011                    type    set   PRGRM+OBJCT
00007   0081                    Revs    set   REENT+1
00008   0000 87CD0057                   mod   MEnd,Name,Type,Revs,Entry,Memsize
00009
00010 D 0000                    DPath   rmb   1           Directory path number
00011 D 0001                    Buffer  rmb   32          buffer for directory entries
00012 D 0021                    Stack   rmb   200
00013 D 00E9                    Memsize equ   .
00014
00015   000D 444C6973           Name    fcs   /DList2/
00016   0013 01                 Version fcb   1
00017   0014 2EA0               Dirname fcs   /. /
00018
00019   0016                    Entry
00020   0016 8681                       lda   #DIR.+READ. file access mode
00021   0018 308DFFF8                   leax  Dirname,PCR file name "."
00022   001C 103F84                     OS9   I$Open
00023   001F 2530                       bcs   Error
00024   0021 9700                       sta   DPath       save the path number
00025   0023 3440                       pshs  U           save U
00026   0025 CE0040                     ldu   #32*2
00027   0028 8E0000                     ldx   #0
00028   002B 103F88                     OS9   I$Seek      skip over . and .. entries
00029   002E 3540                       puls  U           restore U
00030   0030 251F                       bcs   Error
00031   0032                    RLoop
00032   0032 3041                       leax  Buffer,U
00033   0034                    RLoop2
00034   0034 108E0020                   ldy   #32
00035   0038 103F89                     OS9   I$Read
00036   003B 250F                       bcs   TEof
00037   003D 6D41                       tst   Buffer,U    null entry?
00038   003F 27F3                       beq   RLoop2      yes: skip it and read again
00039   0041 8601                       lda   #1          Std output
00040   0043 103F8A                     OS9   I$Write
00041   0046 2509                       bcs   Error
00042   0048 9600                       lda   DPath       directory path
00043   004A 20E6                       bra   RLoop       read again
00044   004C                    TEof
00045   004C C1D3                       cmpb  #E$EOF      Is this EOF?
00046   004E 2601                       bne   Error       no; error
00047   0050 5F                         clrb              yes; return happy
00048   0051                    Error
00049   0051 103F06                     OS9   F$Exit      return
00050   0054 C80070                     EMOD
00051   0057                    MEnd    equ   *

00000 error(s)
00000 warning(s)
$0057 00087 program bytes generated
$00E9 00233 data bytes allocated
$224E 08782 bytes used for symbols
```

```
Microware OS-9 Assembler 2.1   07/13/84 12:08:29                Page 001
ld - List Files in a Directory

00001                                   nam   ld
00002                                   ttl   List Files in a Directory
00003                                   IFP1
00005                                   ENDC
00006    0011                   type    set   PRGRM+OBJCT
00007    0081                   Revs    set   REENT+1
00008    0000 87CD0072                  mod   MEnd,Name,Type,Revs,Entry,Memsize
00009
00010 D  0000                   DPath   rmb   1           Directory path number
00011 D  0001                   Buffer  rmb   32          buffer for directory entries
00012 D  0021                   Stack   rmb   200
00013 D  00E9                   Memsize equ   .
00014
00015    000D 6CE4              Name    fcs   /ld/
00016    000F 01                Version fcb   1
00017    0010 2EA0              Dirname fcs   /. /
00018
00019    0012                   Entry
00020    0012 10830001                  cmpd  #1          length of parameter string
00021    0016 2204                      bhi   DirNGivn
00022         *                 If more than one byte of parameters
00023         *                 assume file name on command line.  Otherwise use "."
00024    0018 308DFFF4                  leax  Dirname,PCR use "." as directory
00025    001C                   DirNGivn
00026    001C 8681                      lda   #DIR.+READ. file access mode
00027    001E 103F84                    OS9   I$Open
00028    0021 2532                      bcs   Error
00029    0023 9700                      sta   DPath       save the path number
00030    0025 3440                      pshs  U           save U
00031    0027 CE0040                    ldu   #32*2
00032    002A 8E0000                    ldx   #0
00033    002D 103F88                    OS9   I$Seek      skip over . and .. entries
00034    0030 3540                      puls  U           restore U
00035    0032 2521                      bcs   Error
00036    0034                   RLoop
00037    0034 3041                      leax  Buffer,U
00038    0036                   RLoop2
00039    0036 108E0020                  ldy   #32
00040    003A 103F89                    OS9   I$Read
00041    003D 2511                      bcs   TEof
00042    003F 6D41                      tst   Buffer,U    null entry?
00043    0041 27F3                      beq   RLoop2      yes: skip it and read again
00044    0043 8D13                      bsr   Edit        Prepare file name for printing
00045    0045 8601                      lda   #1          Std output
00046    0047 103F8C                    OS9   I$WritLn
00047    004A 2509                      bcs   Error
00048    004C 9600                      lda   DPath       directory path
00049    004E 20E4                      bra   RLoop       read again
00050    0050                   TEof
00051    0050 C1D3                      cmpb  #E$EOF      Is this EOF?
00052    0052 2601                      bne   Error       no; error
00053    0054 5F                        clrb              yes; return happy
00054    0055                   Error
00055    0055 103F06                    OS9   F$Exit      return
00056    0058                   Edit
00057    0058 5F                        clrb
00058    0059                   ELoop
00059    0059 6D85                      tst   B,X
00060    005B 2B07                      bmi   ELoopX
00061    005D 270B                      beq   EError      A name can't end in a null
00062    005F 5C                        incb
00063    0060 C11D                      cmpb  #29
00064    0062 25F5                      blo   ELoop       A name can't be more than 29 b
00065    0064                   ELoopX
00066    0064 860D                      lda   #$0D        <CR>
00067    0066 5C                        incb
00068    0067 A785                      sta   B,X
00069    0069 39                        rts
```

```
00070   006A             EError
00071   006A 860D                 lda   #$0D
00072   006C A784                 sta   ,X          Return null line for errors
00073   006E 39                   rts
00074   006F 6FABBC               EMOD
00075   0072             MEnd     equ   *

00000 error(s)
00000 warning(s)
$0072 00114 program bytes generated
$00E9 00233 data bytes allocated
$2299 08857 bytes used for symbols
```

DFORMAT PROGRAM

```
PROCEDURE DFormat
 0000       TYPE dirfmt=name:STRING[29]; lsn(3):BYTE
 001B       TYPE SegLFmt=SLsn(3):BYTE; SegLen:INTEGER
 0031       TYPE fdfmt=attr:BYTE; owner:INTEGER; ModDate(5),LinkCt,
                 FileSize(4),CDate(3):BYTE; SegList(48):SegLFmt
 0070       DIM DirEnt:dirfmt
 0079       DIM FD:fdfmt
 0082       DIM Real_LSN:REAL
 0089       DIM i,errnum:INTEGER
 0094       DIM PPath:BYTE \REM Physical IO  path number
 00B6       OPEN #PPath,"@":READ
 00C2       LOOP
 00C4         GET #0,DirEnt
 00CD         ON ERROR GOTO 10
 00D3         REM Change name from assembler string format to
 0102         REM Basic09 string format by plunking a $00 at the end of it.
 013E         FOR i=1 TO 29
 014E         EXITIF ASC(MID$(DirEnt.name,i,1))>127 THEN
 0164           DirEnt.name=LEFT$(DirEnt.name,i)
 0177         ENDEXIT
 017B         NEXT i
 0186         REM change the LSN of the FD sector from three bytes
 01B9         REM to a real number
 01CC         Real_LSN=DirEnt.lsn(3)+256*(DirEnt.lsn(2)+256*
                       DirEnt.lsn(1))
 01F4         SEEK #PPath,Real_LSN*256
 0203         GET #PPath,FD
 020D         PRINT DirEnt.name;
 0216         IF FD.attr>127 THEN
 0225           PRINT "/";
 022B         ENDIF
 022D         PRINT "  "; FD.ModDate(1); "/"; FD.ModDate(2); "/";
                    FD.ModDate(3); "  "; FD.ModDate(4);
 0265         PRINT "-"; FD.ModDate(5); "  "; FD.CDate(1); "/";
                    FD.CDate(2); "/"; FD.CDate(3);
 029C         PRINT "  "; FD.FileSize(4)+256*(FD.FileSize(3)+256*
                     (FD.FileSize(2)+256*FD.FileSize(1)))
 02D2       ENDLOOP
 02D6 10    REM error handler
 02E9       errnum=ERR
 02EF       IF errnum=211 THEN  \REM end of file
 0309         CLOSE #PPath
 030F         END
 0311       ELSE
 0315         PRINT "Error number "; errnum
 032A         END
 032C       ENDIF
 032E       END
```

MORE GAMES WITH DIRECTORIES

Last month I discussed reading from directory files. This month I'll stay with directories and add some additional tricks.

The directory formatting command at the end of this column is a useful version of the DIR command. It doesn't illustrate any ideas that weren't covered in last column, but it is a single program that is faster to use than the pipeline of programs I presented last month.

I have found that C is a good language to write quick system level programs. Of course, assembly language still has some advantages over any high-level language; not least that almost everyone with OS-9 has an assembler. A functional directory command in assembler would be just too long for one month's column, and not interesting enough to devote several months to. So the first program for this column is an integrated directory formatting command. It is written in C. It could be translated to BasicO9 without too much trouble, but that would require loading BasicO9 every time you want to list a directory. Sorry, people witnout C.

Radio Shack is selling Microware C at an impressively low price. It is a good investment.

Think of dr as a good starting point. It is easy to get it to sort its output. Adding the ability to select only files that meet certain criteria for display is harder but useful enough to be worth the effort. Working this up into a full-screen command environment is something I've been promising myself time to do ..., but I haven't yet.

You can write directories as well as read them. There are good reasons to do this. Renaming files is one reason. The rename command simply writes a new name over the old one in the directory. Deleting and creating files are other reasons to write into directory files, but RBFMan takes care of those operations. Most other things you would want to change about a file involve writing into the file descriptor sector for the file. That's just as easy as writing the directory. Easier.

There is an easy way to make C read a directory file, but there is no equivalent method for updating directory files. The combination of attributes required to write into a directory can be used from assembler, or from the lower level parts of C, but it seems Microware wanted to make it a bit tricky to mess with directories. Before I continue let me add to their implicit warning. If you are not brave and experienced don't even think of updating a directory file!

Writing on directory files is a dangerous thing to do. If you make a mistake you can loose files, or even mess up the structure of the entire disk. DON'T jump in and try programs that write to the directory on an important disk.

After making certain that your program doesn't damage the directory under normal circumstances, think about extraordinary situations. How does the program behave if the system crashes right in the middle of the change? Can trouble start if two programs try to make a change at the same time? What will a program reading the directory while you make your change see?

Another area where you can get in trouble and discover interesting new possibilities hidden in the OS-9 file structure is the possibility for having several directory entries pointing at the same file.

There is a link count in each file descriptor sector. This count will always be one in normal OS-9 systems, but the field offers a way to tell OS-9 (RBFMan) that there are two or more directory entries pointing at a file.

This trick will certainly cause DCHECK to have fits. If you link two directory files to one another (not just with the .. file name) DCHECK will loop between the two directories forever. Even if you don't get this extreme DCHECK will note that more than one file is using the clusters belonging to the file with which you're playing. I have a deadly fascination with this trick of linking to a file several times. The parts to put it togetner are all there, but for some reason Microware hasn't built it into OS-9 yet.

My bet is that the reason for multiple links to files remaining dormant in OS-9 is the recovery problem this feature creates. It is impossible to update the link count in the file descriptor and change the number of directory entries pointing to a file simultaneously. There is always some way to crash the system between the two operations -- pulling the plug will work.

If the link count is greater than the number of directory entries actually linked to the file, the file will eventually be left around with no directory entries pointing at it. The disk space for the file will be allocated and there will be no easy way to return them.

If the link count is smaller than the number of directory entries linked to the file the result is worse. Eventually there will be a directory entry pointing to a file that isn't there. The sectors that used to belong to the file could be part of another file or just free; in either case the result is chaos.

It looks impossible. There is trouble whether the file descriptor is updated before or after the directory. There are two solutions.

One possibility is to live with the problem. An experienced user can fuss around with the allocation map and directory entries, and repair a damaged disk. Most of the work can be automated. Computers don't crash often. Chances are they won't crash in the middle of a directory operation....

The alternative is to use "stable storage" tricks. Every time OS-9 starts up look for evidence of a crash, and every

time you update a directory prepare for one. This slows directory updates, systems startup, and even disk mounts; but it prevents users from having to worry about recovery.

Neither method sounds OS-9-like. I use the "live with the problem" method. I've never had reason to regret it, but I am prepared for the worst. The "stable storage" method is interesting ... worth a brief discussion.

Here is a way to reliably update a directory:

1. Copy the entire directory including file descriptors to a special spot, with its address known to recovery routines (in a table located at some known spot).

2. Update the copy of the directory.

3. Put the address of the old directory in the same table as the address of the new one with a mark indicating that it is old.

4. Put the address of the updated directory in the directory's parent.

5. Remove the new directory from the table.

6. Delete the old directory removing it from the table.

Step 4 involves a single operation that changes the directory structure visible to the public. Until step 4 is executed no program knows about the change. After step 4 there is a consistent updated directory.

Recovery works as follows:

- *IF THERE ISN'T ANYTHING IN THE "TABLE"*

  no recovery necessary

- *IF THERE IS A POINTER MARKED "OLD" AND NO NEW POINTER*

  delete the old directory

- *IF THERE IS ONLY A NEW DIRECTORY IN THE TABLE*

  delete it.

- *IF BOTH POINTERS ARE IN THE TABLE*

  continue from step 4 in the update procedure

Things fall apart again if two processes might simultaneously update the directory, or the file descriptors attached to it. If that is permitted the protocol gets complicated. Too complicated for this column.

I'm not going to try to present a program implementing stable storage this month. Just a simple program to squeeze the null entries out of a directory.

DR PROGRAM

```
#include <stdio.h>
#include <ctype.h>
#include <modes.h>
#include <direct.h>


static struct dirent DirEntry;
static FILE *fopen(), *dir, *disk;
/*-------------------------------------------------------*
 *    dr                                directory read   *
 *    Read and format all the important fields in a      *
 *    directory entry and the attached FDs.               *
 *    To allow formatting, sorting, and searching programs*
 *    the best access to this data it is just printed    *
 *    without titles.                                    *
 *    There are no options.  A directory name may be given*
 *    as a command line argument.  If it isn't the current*
 *    data directory will be listed.                     *
 *-------------------------------------------------------*/
main(argc,argv)
 int argc;
 char *argv[];
 {
    char temp[120];
    char device[30];
    register int i;


    pflinit();
    argv++;                         /* bump past program name in args*/
    if (argc > 1)
       strcpy(temp,*argv);
    else
       strcpy(temp,".");            /* default directory */

    if((dir = fopen(temp, "d")) == NULL) /* open the directory */
       {
          fprintf(stderr,"%s can't be read as a directory\n",temp);
          exit(1);
       }


    strcpy(device,"@");  /* default device is data directory device */
    if(temp[0] == '/')
       {
          i = 0;
          do
             device[i] = temp[i];
          while (isalnum(temp[++i]) || temp[i] == '.' || temp[i] == '_');
          device[i++] = '@';
          device[i] = '\0';
          fprintf(stderr,"Device: %s\n",device);
       }

   /*-------------------------------------------*
    *  Open the device containing the directory  *
    *  we're about to list.                      *
    *--------------------------------------------*/

   if((disk = fopen(device,"r")) == NULL)
       {
          fprintf(stderr,"Error %d opening device s\n",ferror(disk),device);
          exit(1);
       }

    fread(&DirEntry, sizeof DirEntry, 1, dir); /* skip . entry */
    fread(&DirEntry, sizeof DirEntry, 1, dir); /* skip .. entry */

    /*-------------------------------------------*
     * Read and format directory entries until EOF*
     * Null entries are ignored by putEntry.      *
     *--------------------------------------------*/
    while (fread(&DirEntry, sizeof DirEntry, 1, dir) != NULL)
       putEntry(DirEntry.dir_name,DirEntry.dir_addr);

    exit(0);
 }


 putEntry(Name,Address)
```

```
81      char *Name, *Address;
82      {
83         char CName[30];
84         long LSN;
85
86         if(Name[0] == '\0')
87            return; /* Null entry */
88
89         fixname(CName,Name); /* change OS-9 string (high-bit)
90                                 to C format string */
91
92         l3tol(&LSN,Address,1); /* make LSN usefull */
93
94         printf("%s",CName);    /* reformatted file name */
95
96         expansion(LSN);        /* rest of the information */
97
98         return;
99      }
100
101 static struct fildes FD;
102
103  expansion(LSN) /* print everything interesting about a file */
104     long LSN;
105     {
106
107        if(fseek(disk, LSN*256, 0) == EOF)
108           {
109              fprintf(stderr, "Disk seek error %d\n",ferror(disk));
110              exit(1);
111           }
112        if(fread(&FD, sizeof FD, 1, disk) == NULL)
113           {
114              fprintf(stderr,"Disk read error %d\n",ferror(disk));
115              exit(1);
116           }
117
118        format_attr(FD.fd_att);
119        printf(" %u",FD.fd_own);
120        format_date(FD.fd_date,5);
121        printf(" %d %ld",FD.fd_link, FD.fd_fsize);
122        format_date(FD.fd_dcr,3);
123        printf("\n");
124        return;
125     }
126
127
128
129  fixname(goodname,badname) /* convert from OS-9 string to C string */
130     char *goodname,*badname;
131     {
132        register int i;
133
134        i = 0;
135        do
136           {
137              *goodname++ = *badname & '\x7f';
138           }
139        while ((*badname++ > 0) && (++i <= 29));
140
141        *goodname = '\0';
142        return;
143     }
144
145  format_attr(attr) /* print file attributes */
146     char attr;
147     {
148
149        if(attr & S_IFDIR) /* is it a directory? */
150           printf("7 [ d");
151        else
152           printf(" [");
153
154        if(attr & S_ISHA E)
155           printf("-ps");
156
157        if(attr & S_IOEXEC)
158           printf("-pe");
159
160        if(attr & S_IOWRITE)
161           printf("-pw");
162
```

```
163         if(attr & S_IOREAD)
164            printf("-pr");
165
166         if(attr & S_IEXEC)
167            printf("-e");
168
169         if(attr & S_IWRITE)
170            printf("-w");
171
172         if(attr & S_IREAD)
173            printf("-r");
174         printf("]");
175         return;
176      }
177
178   format_date(date,x) /* print a date in readable form */
179      char *date; /* yymmdd (hhmm) */
180      int x; /* number of entries in the date array */
181      {
182         char *month_name();
183
184         printf(" (%d %s 19%02d",date[2], month_name(date[1]), date[0]);
185         if(x >= 5)
186            printf(" %d:%02d", date[3], date[4]);
187         printf(")");
188         return;
189      }
190
191
192   char *month_name(n) /* return name of n-th month */
193      int n;
194      {
195         static char *name[] =
196            {
197               "illegal month",
198               "January",
199               "February",
200               "March",
201               "April",
202               "May",
203               "June",
204               "July",
205               "August",
206               "September",
207               "October",
208               "November",
209               "December"
210            };
211
212         return(( n < 1 || n > 12) ? name[0] : name[n]);
213      }
```

DIRSQZ PROGRAM

```
 1 #include <stdio.h>
 2 #include <direct.h>
 3 #include <modes.h>
 4
 5 static struct dirent DirEntry;
 6 static int dir; /* path number */
 7
 8 /* DirSqz */
 9 /*       This program can be used to press the null
10          entries out of large directories that have
11          been hit with many deletions.
12 */
13 main()
14  {
15     long strt_ptr, end_ptr, backup();
16
17 pflinit();
18     if((dir = open(".", S_IFDIR+S_IREAD+S_IWRITE)) == NULL)
19         {
20            fprintf(stderr,"Error opening the directory %d\n",ferror(dir));
21            exit(1);
22         }
23     strt_ptr = sizeof DirEntry * 2; /* point past . and .. */
24     getstat(2, dir, &end_ptr); /* length of the file */
25     end_ptr -= sizeof DirEntry; /* point back from end */
26     end_ptr = backup(end_ptr,strt_ptr);
27
28     for(;strt_ptr < end_ptr; strt_ptr += sizeof DirEntry)
29         {
30            lseek(dir,strt_ptr,0);
31            read(dir, &DirEntry, sizeof DirEntry);
32            if(DirEntry.dir_name[0] == '\0')
33                {
34                   end_ptr = backup(end_ptr, strt_ptr);
35                             /* leaves DirEntry with good data */
36                   if(end_ptr <= strt_ptr)
37                      break;
38                   lseek(dir, strt_ptr, 0);
39                   write(dir, &DirEntry, sizeof DirEntry);
40                   lseek(dir, end_ptr, 0);
41                   write(dir, "", -1);
42                   end_ptr = backup(end_ptr,strt_ptr);
43                }
44         }
45     exit(0);
46  }
47
48
49
50
51  long backup(end_ptr,strt_ptr)
52     long end_ptr, strt_ptr;
53     {
54        for(;end_ptr >= strt_ptr;end_ptr -= sizeof DirEntry)
55            {
56               lseek(dir, end_ptr, 0);
57               read(dir, &DirEntry, sizeof DirEntry);
58               if(DirEntry.dir_name[0] != '\0')
59                  break;
60            }
61        return(end_ptr);
62     }
```

A REVIEW OF O-F

A few weeks ago I spent most of a Saturday hooking my old SWTPC FLEX machine to my new machine as a remote computer so I could use it to write a FLEX-format disk. It felt rather odd using my "smart terminal" program to communicate with a machine less than a foot away. The process involves shuffling disks drives back and forth, and much opening and shutting of cabinets. I don't like it much. My new machine has GIMIX software switching, so I can run FLEX on it, but even the remarkable GIMIX CPU board can't run both operating systems at once. On occasion I have uploaded a file from one OS to an IBM and then downloaded it with the other OS, accomplishing a change of disk format from FLEX to OS-9 or vice-versa. These methods are all inelegant, ad hoc solutions to a problem. Dr. Matthew Scudiere has come up with a much cleaner solution: He has written an OS-9/FLEX copy program called O-F.

GENERAL SYSTEM DESCRIPTION

This OS9/FLEX copy program is a BASIC09 program which allows the user to convert an OS-9 format disk into a hybrid form which can be read and written by FLEX. In the process of doing this it makes the disk inaccessible to OS-9 except as an entire disk (i.e. /Dn@) but O-F is able to copy files to and from the hybrid disk, and read the FLEX directory. The disk that results from the reformatting is enough like standard FLEX format that FLEX doesn't know the disk isn't one of its own.

LIMITATIONS

Only freshly formatted, single sided 5 or 8 inch disks with no bad sectors can be used, and there is no way to use a disk which is in real FLEX format (formatted by the FLEX NEWDISK or FORMAT program). The FLEX to OS-9 copy part of the program expands tab characters into strings of blanks by default, but there is an option which causes the file to be copied intact. Of course, this program doesn't make any attempt to convert FLEX programs into OS-9 programs. That is work for other programs.

OPERATION

In order to run O-F you must first start Basic09. The version I tested was in source form, so I had to load it and run it. If it is distributed as Basic09 I-Code it should be possible to just run it. The program lists 7 options:

```
0    Directions
1    FLEX Directory
2    Copy FLEX text file to OS9
3    Copy OS9 path to FLEX
4    Delete FLEX File
5    Reformat OS9 Disk
6    Exit program
```

and prompts for a selection. "Directions" produces a quick summary of the function of the program, about half a screen full. "FLEX Directory" lists the basic information in the directory of a pseudo-FLEX disk: file name, Begin, End, Size, and date. It also gives the number of sectors used on the disk, and the number of sectors left. The "Copy FLEX text file to OS9" dialogue is:

```
FLEX Compatible source Drive ID
--
FLEX file name to copy --
Copy to OS9 destination path --
```

The "Compatible source Drive ID" is the device name for the disk that has been reformatted; that wasn't too clear to me. The "Copy OS9 path to FLEX" dialogue is:

```
Drive ID --
FLEX File name to write (Caps)
--
Copy FROM OS9 SOURCE path --
```

To delete a FLEX file, select 4, then:

```
Flex compatible source Drive ID
--
FLEX file name to delete (use
proper case) --
```

The dialogue for reformatting a disk is very cautious:

```
Drive ID --
  Are you sure? --
Overwrite -- <old volume name>
  Are you sure? --
5-in or 8-in disk? --
```

I tried reformatting and writing on 5 inch disks (SS/SD, SS/DD, 40 track and 80 track), and 8 inch disks of all permutations. It worked on the 5 inch disks, and on SS single and double density 8 inch disks. I was able to read psuedo-FLEX files created by O-F from FLEX without any trouble. O-F had no trouble reading files written by FLEX on disks reformated by O-F. The reformatted disks were also fully usable in FLEX. FLEX truly thinks the reformatted disk is one of its own. One nice touch is that the program puts two entries in the OS-9 root directory of the reformatted disk:

```
** NO OS9 Files Allowed **        (This is a FLEX copy disk)
```

These entries appear if you do a DIR command on the reformatted disk, letting you know very quickly that this disk is special.

EVALUATION

This is a competent and very useful program. It is especially well equipped with error messages and informative text. In fact, although it came without a manual, I was able to follow the built-in directions without any trouble. I do hope that a manu-

al is available by the time this program hits the market. A program without a manual seems somehow unbalanced even if it is usable without documentation. A nice extra is that it appears that this program may be distributed in source form.

O-F works by tricking FLEX. This together with the variety of disk formats that FLEX might use forces the program have some odd restrictions. The most serious limitation is the restriction to specially formatted disks. It certainly would be nice to be able to drag out a four year old FLEX disk and read it with this program. The restriction to single sided disks is reasonable in the context of copying files from one format to the other. For some people the most important limitation will be the language requirement. Since this is a Basic09 program, you must have Basic09 to be able to run it. It could be a measure of the desperate need for a program like this one that it is being hustled out in Basic09 form.

One of O-F's strongest points is the cautious approach it takes to the user. This program doesn't know how to deal with double sided disks, but it doesn't just tell you so, it won't let you use them. You get a message clearly telling you that double sided disks are not-ok if you try. Similar messages appear if you try to use a disk that is flawed in a number of other ways.

SUMMARY

O-F is available from DATA-COMP. It isn't really a program of general interest ... there are probably some OS-9 users who don't have FLEX or friends with FLEX. Those people have very little use for this program. The group of people this program should prove most useful to are the owners of software-switching machines. Using this program they can conveniently transfer data between operating systems. There are a lot of FLEX users out there -- our close relatives in the computer world. It is good to be able to exchange disks with them even if we have to be the ones to provide the disks.

REVIEW OF OS-9 CIS COBOL

OVERVIEW

COBOL is a big language, an old language, and an extremely popular language. Some languages were designed to be compiled and run on small computers; COBOL was not. COBOL is vehemently detested by many people involved with computers, but, despite all the nasty publicity it gets, COBOL is probably the most used computer language in the world. If you need to hire an experienced programmer for a business application, you will find the hunting best if you snoot for a COBOL programmer. COBOL was one of the first compiled languages developed for computers (around 1960), and it has been being (arguably) improved since then. The fully "improved" version of COBOL is an enormous language whose compiler is fully capable of needing the best part of a megabyte of memory to run properly.

There are standards against which any version of COBOL should be measured. ANSI (American National Standards Institute) has defined a COBOL standard which constitutes the official definition of the language. CIS COBOL was written to conform to the ANSI standard definition of COBOL.

To quote the manual: "CIS COBOL is ANSI COBOL as given in 'American National Standard Programming Language COBOL' (ANSI X3.23 1974)." It includes level 1 of the ANSI definition of COBOL along with a few parts of level 2. This doesn't mean that CIS COBOL is the version of the language you may have used on a mainframe computer, but it does mean that if you don't use the enhancements that CIS COBOL includes, the programs you write using it will run essentially unmodified on any other computer that runs level 2 or higher of ANSI COBOL. Also, since CIS COBOL is compiled to intermediate code, programs written in it can be run on any computer that has the appropriate interpreter. If you read the adds in BYTE, you will see that CIS COBOL is implemented for many computers.

I didn't test CIS COBOL exhaustively for conformance to the standard, but I did write a few programs in it. I am used to IBM's VS-COBOL, and a version of UNIVAC COBOL; both are highly enhanced versions of higher levels of ANSI COBOL than CIS COBOL. It took me a while to learn which of my favorite programming tricks aren't possible under level 1 of ANSI COBOL, but, after I learned the limitations I had to live with, I found that I could write programs with no more difficulty than I usually experience when writing in COBOL. I wish I had been able to transfer a program from the IBM to my micro and compile it, but I don't know of any real programs written to be compiled by ANSI level 1 COBOL. Transferring a program in the other direction is no problem.

There is far too much to CIS COBOL for me to say with certainty that it all works, but I understand that the language has actually been successfully tested against a set of standard test programs.

ENHANCEMENTS

Standard COBOL doesn't support the interactive microcomputer environment very well, but CIS COBOL includes enhancements to the ACCEPT and DISPLAY statements that make it relatively easy to display screens of data, and accept data from fields defined on the screen. Information can be accepted from, or displayed at, a particular cursor location. An input field can be defined as numeric only, in which case any inappropriate characters (like "A") won't be accepted. When a field is filled with data, the cursor automatically jumps to the beginning of the next field. There are special keys which jump the cursor forward and backward a field at a time. Special function keys can be defined. They act like a carriage return (terminate entry into a screen), but a program can determine whether a screen was terminated by a carriage return or a function key, and which function key was used. The location of the cursor when carriage return was pressed is also available. The net effect of these enhancements is that it is fairly easy to write CIS COBOL programs that accept and display screens of data.

In addition to the usual COBOL file organizations (including ISAM), CIS COBOL allows an organization they call "line sequential." Line sequential files are variable length record files, in which the records are terminated by carriage returns. This makes it easy to read and write files that Pascal would call "files of text." The most generally important examples of files of this type are files created by text editors, and line by line output to a terminal or printer. The other access modes supported by CIS COBOL are: sequential, relative, and indexed. The names of files can be specified at run time using statements like:

```
SELECT FILE-15 ASSIGN TO
FILE-15-NAME.
...
ACCEPT FILE-15-NAME.
OPEN INPUT FILE-15.
```

In addition to the standard ANSI debug features, CIS COBOL has a respectable interactive debugger. The commands available under this debugger are:

```
P - Display the current program counter
G - Set a breakpoint
X - Single step
D - Display data at specified offset in data division
A - Change memory (ASCII)
S - Set block for display or change
/ - Display block
. - Change bytes in block
T - Trace paragraphs
L - Write CR,LF
M - Define a debug macro
$ - End a macro definition
C - Display a specified character
; - precedes a comment (for describing macros)
```

The interactive debugger can be used on any COBOL program by including +D on the command line that invokes the program, e.g., RunC +D test.int. This means that you can use the debugger on a program without having to do anything special when you compile it.

Microware has included eight subroutines in the COBOL run time system which can be called from a COBOL program. MOVE-BLOCK is a procedure that can be used to do a high speed move of a block of data. ABORT terminates the program with an error code. CHAIN makes the standard OS-9 F$Chain system call available. The FUN-KEY subroutine can be used after a ACCEPT statement to find out if a function key was pressed instead of the carriage return key, and which one. DATE returns the date and, optionally, the time. SHELL invokes a shell, passing it a specified string. CHX and CHD change the execution and data directories for the program.

The subroutines in the run time system are called by number. CIS COBOL can also call subroutines which are either COBOL I-code, or object code. The CALL statement looks like:

```
CALL "/D0/SUBLIB/TEST.SUB.1"
USING ...
  ON OVERFLOW ....
```

The called program is loaded into memory if it is not already there, and, depending on whether the module header indicates that it is I-code or object code, interpreted or executed. If there is no room in memory for the new module, the ON OVERFLOW clause in the CALL statement gets control. The CANCEL verb unlinks a subroutine, freeing the memory it is using.

In addition to these methods of calling external subroutines, CIS COBOL supports program segmentation, which can be used to divide the program into sections that will remain on disk until they are needed. Segments use memory efficiently at the cost of extra disk I/O by sharing a common pool of overlay memory.

In addition to supporting ANSI COBOL level 1, including:

```
The Nucleus
Table Handling
Sequential Input and Output
Relative Input and Output
Indexed Input and Output
Segmentation
Library (Copy)
Inter-program communication
debug
```

CIS COBOL supports parts of level 2 of ANSI COBOL including:

- Nested IF
- PERFORM UNTIL
- The START statement for Relative and Indexed I/O
- Full level 2 Inter-program communication

LIMITATIONS

I was disappointed with some of the restrictions of the low level of COBOL implemented for CIS COBOL, but not very surprised. I am more upset by some problems with terminal support, and the CONFIG utility that is used to customize the run time package for a particular type of terminal.

The features of advanced levels of COBOL that I missed most were AND and OR in IF statements. It is possible to do without boolean operations in IF statements, but I am not used to having to work around a limitation like that. Another very popular feature which is missing in CIS COBOL is the SORT statement. A surprising number of production COBOL programs include at least one sort, and it would be hard to eliminate a sort from a program without a major redesign.

The run time system which interprets the COBOL intermediate code also includes routines for terminal control. It is customized for a terminal by a utility program called CONFIG. I was not impressed with CONFIG. My favorite terminal uses the ANSI standard terminal control sequences ... CONFIG was clearly not written with my terminal in mind. I struggled for two evenings trying to get RunC configured for my TeleVideo with no success. Finally, I gave up and turned to my H-19, which was much more like what CONFIG wanted ... I had COBOL running in ten minutes. There were

three fundamental problems with CONFIG's handling of my TeleVideo's control sequences. CONFIG expected most terminal control strings to be no more than three characters long; several of the ANSI strings are longer than that. CONFIG simply can't deal with the ANSI direct cursor positioning sequence; I circumvented that problem by pretending that my terminal didn't have a direct cursor positioning command, and specifying relative positioning. CONFIG can only deal with commands that move the cursor one row or column at a time in relative positioning mode. Since the ANSI strings that cause the cursor to move one row or column are three characters long, this is a slow way to adjust the cursor position. The clear-screen sequence for my terminal is four characters long; so I couldn't use it. RunC tries to fake a clear-screen somehow, but it makes a real mess of it. The clear-screen sequence somehow came out as a string of thousands of <bell> characters. I understand that a more recent version of CONFIG than the one I have allows a four character string for the clear-screen sequence. I think that would have made it possible for me to get my TeleVideo working with COBOL.

CONFIG forms a trap for the unwary user. Once you start into it there is no turning back. If you change your mind about the response you just keyed in, you have to wait until you reach the end of the entire (long) string of questions, and ask to be allowed to change a large subset of your answers. When you are going through CONFIG to fix a mistake or change an existing terminal description to fit a new terminal, you have to fill in the correct answer to each question. There is no way to select a default, or keep the old value It is true that CONFIG is not likely to be a heavily used utility, but I found it so hard to use that I would much rather have written a few subroutines to support my terminals.

Once I got the screen support working, I found that I wasn't pleased with the way it worked. I believe that when the cursor leaves a numeric field, the field should be right justified and zero filled. The screen handling package in CIS COBOL seems to agree with me to some extent. If you enter a "." in an integer field it will right justify and zero fill, but if you exit the field with a carriage return (ending the entire screen) or down arrow (moving to the next field), a test for numeric in the program will indicate that the field is not numeric. If the field has editing characters in it the field is inclined to end up left justified and zero filled.

I am used to getting useful, english error messages from COBOL; CIS COBOL gives error messages with numeric codes in them indicating what the error is. Even after I looked up the error, it wasn't clear what the problem was. For instance, when I hadn't declared a variable it told me that there was a type mismatch in the statement using the undeclared variable. When I tried to use AND and OR, it gave me the same error. I ended up treating the error message as "something's wrong around here."

BENCHMARKS

I ran two benchmarks against this COBOL: one for speed at numeric processing (the sieve), the other for speed in handling ISAM files. I adjusted the prime number program from the January 1983 BYTE slightly to fit ANSI level one, and ran it. This version of COBOL would have fallen nearly at the bottom of the chart given in that BYTE, between Microsoft COBOL and RMCOBOL. It took 541 seconds to find the first 1899 primes. I could have made the program run somewhat faster by using indexing instead of subscripting, but that would have spoiled the benchmark. I have to admit that I felt silly writing a Eratosthenes Sieve program in COBOL. Testing COBOL for its ability to find prime numbers is like testing programmers for their ability to read Latin; they may be able to do it but it is hardly relevant. I ran that benchmark because it is the most used benchmark for microcomputer languages, but I also ran another non-standard, but, I think, more relevant, benchmark.

I constructed a benchmark program which gives a good measure of the speed with which the language handles indexed I/O. Indexed I/O is very important to the group of users who might use COBOL. Interpreting the results of a benchmark that involves I/O is a little tricky. Certainly the file structure the language uses is very important, especially with a large indexed file; but the access time for the disk is an important factor, and the time the operating system takes for a context switch is somewhat important.

I built a file 10,000 records long of 55 byte records with five byte keys and then read it randomly reading two records alternately from each end. It took 2615 seconds to build the file and 3233 seconds to read the file (it would, of course, have been possible to read it faster if I had read sequentially). I ran these benchmarks on a GIMIX system with a CM 5000 Winchester (a file that size would not have fit on my 8" floppies). I used OS-9 Level Two on a 2 mhz 6809. The performance would have been much worse if I had used a floppy instead of a Winchester, and somewhat better if I had used GMX-III.

I compiled three COBOL programs on the same machine I ran the benchmarks on. A simple merge program which I haven't included with this review took 45 seconds to compile, the sieve compiled in 35 seconds, and the ISAM test program took 43 seconds.

SUMMARY

It is possible to get past the problems with CONFIG, to learn to live with the primitive error messages, and to feel comfortable with the screen handling conventions. What is left is a substantial implementation of an old, but useful language. I don't think everyone should run out and buy this package, but, for a few people, it could be uniquely useful. If you want to use a group of COBOL programs on microcomputer, it would certainly be easier to convert them from one level of

COBOL to another than to translate them into an entirely different language. CIS COBOL would be a good teaching tool for schools unable to afford time on a machine with a full-blown COBOL compiler. It should be relatively easy to find programmers who can work in COBOL. With CIS COBOL, a microcomputer could be used as a development environment for COBOL programs, though the low level of CIS COBOL would prevent this in most cases. Perhaps the most significant advantage of CIS COBOL over other languages is that programs written in CIS COBOL can be moved in I-Code form to a variety of other machines and operating systems, and run without source code. UCSD Pascal has shown that this is an asset even though it can't generally run under a normal operating system.

CIS COBOL was written by Micro Focus Limited. Microware wrote a run time package for it that allows any program written in CIS COBOL, including CIS COBOL itself, to be run under OS-9. By writing a run time package for CIS COBOL, and arranging to license it for OS-9, Microware made a large collection of business software available to OS-9 users. If you are looking for a nice accounting system, payroll, MRP system, or whatever, check with Microware. They have a long list of vendors offering programs which run under the CIS COBOL run time system.

Some small number of people will find Microware's version of CIS COBOL just what they need. If you think you are one of those people, I recommend that you get the manual before you commit to the language. The manuals won't be any help to you if you don't know COBOL, but, if you do, they will leave you with an accurate impression of the language, and either leave you impatient to get the software, or disappointed about some important missing feature (most likely sort).

COBOL TEST PROGRAM

```
** CIS COBOL V4.4                    Test.CBL                    PAGE: 0001
**
        IDENTIFICATION DIVISION.
        PROGRAM-ID. FIRST-TEST.PROGRAM.
        AUTHOR. PETER DIBBLE.
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER. GIMIX.
        OBJECT-COMPUTER. GIMIX.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
             SELECT INPUT-1 ASSIGN ":CI:"
                  ORGANIZATION IS LINE SEQUENTIAL.
             SELECT MERGE-FILE ASSIGN MERGE-NAME.
             SELECT TEMP-FILE ASSIGN "MERGE.TEMP".
        DATA DIVISION.
        FILE SECTION.
        FD  INPUT-1;
             RECORD 40;
             BLOCK 5;
             LABEL RECORDS ARE STANDARD.
        01  INPUT-1-LINE                    PIC X(40).
        FD  MERGE-FILE;
             RECORD 20;
             BLOCK 10;
             LABEL RECORDS ARE STANDARD.
        01  MERGE-LINE                      PIC X(20).
        FD  TEMP-FILE;
             RECORD 20;
             BLOCK 10;
             LABEL RECORDS ARE STANDARD.
        01  TEMP-LINE                       PIC X(20).
        WORKING-STORAGE SECTION.
        01  IN-THIS                         PIC X(40) VALUE SPACES.
        01  LINE-FMT REDEFINES IN-THIS.
             02  KEEP-THIS.
                  04  FILLER                  PIC X(19).
                  04  CARRIAGE-RETURN         PIC X.
             02  FILLER                     PIC X(20).
        01  MERGE-THIS                      PIC X(20) VALUE SPACES.
        01  FILE-STAT                       PIC X VALUE "0".
        PROCEDURE DIVISION.
        START-UP.
       *  PARAMETERS ARE GIVEN IN THE FIRST RECORD OF STD. INPUT
             OPEN INPUT INPUT-1.
             READ INPUT-1 INTO MERGE-NAME.
             OPEN INPUT MERGE-FILE.
             OPEN OUTPUT TEMP-FILE.
             DISPLAY "MERGING STANDARD INPUT WITH ", MERGE-NAME.
```

```
           READ INPUT-1 INTO IN-THIS;
                AT END MOVE HIGH-VALUES TO IN-THIS.
           PERFORM FIX-IN.
           READ MERGE-FILE INTO MERGE-THIS;
                AT END MOVE HIGH-VALUES TO MERGE-THIS.
       MAIN-SECTION.
           PERFORM MERGE-LOOP UNTIL FILE-STAT EQUAL TO "1".
           MOVE "0" TO FILE-STAT.
           CLOSE MERGE-FILE.
           OPEN OUTPUT MERGE-FILE.
           CLOSE TEMP-FILE.
           OPEN INPUT TEMP-FILE.
           PERFORM READ-TEMP.
           PERFORM COPY-TEMP-TO-MERGE UNTIL FILE-STAT EQUAL TO "1".
           STOP RUN.
       MERGE-LOOP.
           PERFORM PICK-NEXT.
           WRITE TEMP-LINE.
       END-MERGE-LOOP.
           EXIT.
       PICK-NEXT.
           IF KEEP-THIS < MERGE-THIS
                THEN
                     PERFORM FIX-IN
                     MOVE KEEP-THIS TO TEMP-LINE
                     READ INPUT-1 INTO IN-THIS;
                          AT END PERFORM END-IN
                ELSE
                     MOVE MERGE-THIS TO TEMP-LINE
                     READ MERGE-FILE INTO MERGE-THIS;
                          AT END PERFORM END-MERGE.
       PICK-NEXT-END.
           EXIT.
       END-IN.
           MOVE HIGH-VALUES TO IN-THIS.
           IF MERGE-THIS = HIGH-VALUES
                THEN
                     MOVE "1" TO FILE-STAT.
       END-MERGE.
           MOVE HIGH-VALUES TO MERGE-THIS.
           IF IN-THIS = HIGH-VALUES
                THEN
                     MOVE "1" TO FILE-STAT.
       FIX-IN.
           MOVE X"0D" TO CARRIAGE-RETURN.
       COPY-TEMP-TO-MERGE.
           WRITE MERGE-LINE.
           PERFORM READ-TEMP.
       END-COPY-TEMP-TO-MERGE.
           EXIT.
       READ-TEMP.
           READ TEMP-FILE; AT END PERFORM END-TEMP.
           MOVE TEMP-LINE TO MERGE-LINE.
       END-READ-TEMP.
           EXIT.
       END-TEMP.
           MOVE "1" TO FILE-STAT.
       END-INPUT.
           MOVE HIGH-VALUES TO IN-THIS.
       END-MERGE-IN.
           MOVE HIGH-VALUES TO MERGE-THIS.
       END-PROGRAM.
           EXIT.
** CIS COBOL V4.4 REVISION 0                                    URN rp/
** COMPILER COPYRIGHT (C) 1978,1981 MICRO FOCUS LTD
** ERRORS=00000 DATA=00791 CODE=00489 DICT=00654:01229/01883 GSA FLAG
```

COBOL SIEVE

```
** CIS COBOL V4.4                    siev.cbl                    PAGE: 0001
**
        IDENTIFICATION DIVISION.
        PROGRAM-ID. SIEVE.
        AUTHOR. PETER DIBBLE.
        ENVIRONMENT DIVISION.
        WORKING-STORAGE SECTION.
        77  PRIME                          PIC 9(5) COMP.
        77  PRIME-COUNT                    PIC 9(5) COMP.
        77  I                              PIC 9(4) COMP.
        77  K                              PIC 9(5) COMP.
        01  BIT-ARRAY.
            03  BIT OCCURS 8191 TIMES      PIC 9 COMP.
        PROCEDURE DIVISION.
        START-UP.
            DISPLAY "TEN ITERATIONS".
            PERFORM SIEVE THROUGH SIEVE-END.
            DISPLAY "PRIMES FOUND: ", PRIME-COUNT.
            STOP RUN.
        SIEVE.
            MOVE ZERO TO PRIME-COUNT.
            MOVE 1 TO I.
            PERFORM INIT-BITS 8191 TIMES.
            MOVE 1 TO I.
            PERFORM SCAN-FOR-PRIMES THROUGH END-SCAN-FOR-PRIMES
                8191 TIMES.
        SIEVE-END.
            EXIT
        INIT-BITS.
            MOVE 1 TO BIT (I).
            ADD 1 TO I.
        END-INIT-BITS.
            EXIT.
        SCAN-FOR-PRIMES.
            IF BIT (I) = 0
                THEN
                    GO TO NOT-PRIME.
            ADD I I 1 GIVING PRIME.
      *     DISPLAY PRIME.
            ADD I PRIME GIVING K.
            PERFORM STRIKOUT UNTIL K > 8191.
            ADD 1 TO PRIME-COUNT.
        NOT-PRIME.
            ADD 1 TO I.
        END-SCAN-FOR-PRIMES.
            EXIT
        STRIKOUT.
            MOVE 0 TO BIT (K).
            ADD PRIME TO K.
        END-PROGRAM.
            EXIT.
** CIS COBOL V4.4 REVISION 0                                URN rp/
```

COBOL BENCHMARK PROGRAM

```
** CIS COBOL V4.4                    Bnch.CBL                    PAGE: 0001
**
        IDENTIFICATION DIVISION.
        PROGRAM-ID. ISAM-BENCHMARK
        AUTHOR. PETER DIBBLE.
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER. GIMIX.
        OBJECT-COMPUTER. GIMIX.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
             SELECT ISAM-FILE-1 ASSIGN "ISAM.FILE";
                  ORGANIZATION IS INDEXED;
                  ACCESS MODE IS SEQUENTIAL;
                  RECORD KEY IS ISAM-KEY-1.
             SELECT ISAM-FILE-2 ASSIGN "ISAM.FILE";
                  ORGANIZATION IS INDEXED;
                  ACCESS MODE IS RANDOM;
                  RECORD KEY IS ISAM-KEY-2.
        DATA DIVISION.
        FILE SECTION.
        FD   ISAM-FILE-1;
             DATA RECORD ISAM-RECORD-1.
        01   ISAM-RECORD-1.
             03   ISAM-KEY-1                  PIC 9(9) COMP-3.
             03   FILLER                      PIC X(50).
        FD   ISAM-FILE-2;
             DATA RECORD ISAM-RECORD-2.
        01   ISAM-RECORD-2.
             03   ISAM-KEY-2                  PIC 9(9) COMP-3.
             03   FILLER                      PIC X(50).
        WORKING-STORAGE SECTION.
        77   KEY-NO                          PIC 9(9) COMP-3 VALUE 0.
        77   HI-NUMBER                       PIC 9(9) COMP-3.
        77   LO-NUMBER                       PIC 9(9) COMP-3.
        77   DATE                            PIC XXX VALUE "004".
        01   WORK-DATA.
             03   WORK-KEY                    PIC 9(9) COMP-3.
             03   I-DATA                      PIC X(50).
        01   SYSTEM-DATE.
             03   YEAR                        PIC 99.
             03   MONTH                       PIC 99.
             03   DAY                         PIC 99.
        01   SYSTEM-TIME.
             03   HOUR                        PIC 99.
             03   MINUTE                      PIC 99.
             03   SECOND                      PIC 99.
        PROCEDURE DIVISION.
        START-UP.
             OPEN OUTPUT ISAM-FILE-1.
             MOVE "ASSORTED DATA: NAME, ADDRESS, ETC, OR WHATEVER" TO
                  I-DATA.
             ADD 1 KEY-NO GIVING LO-NUMBER.
             MOVE KEY-NO TO WORK-KEY.
             DISPLAY "START BUILD".
             CALL DATE USING SYSTEM-DATE, SYSTEM-TIME.
             DISPLAY "TIME " HOUR, ":", MINUTE, ":", SECOND
```

```
            PERFORM ADD-RECORD 10000 TIMES.
            CLOSE ISAM-FILE-1
            DISPLAY "BUILD DONE".
            CALL DATE USING SYSTEM-DATE, SYSTEM-TIME.
            DISPLAY "TIME " HOUR, ":", MINUTE, ":", SECOND
            MOVE WORK-KEY TO HI-NUMBER.
            DISPLAY "READ STARTING".
            OPEN INPUT ISAM-FILE-2.
            PERFORM TEST-READS 2500 TIMES.
            CLOSE ISAM-FILE-2.
            CALL DATE USING SYSTEM-DATE, SYSTEM-TIME.
            DISPLAY "TIME " HOUR, ":", MINUTE, ":", SECOND
            DISPLAY "READ DONE".
            STOP RUN.
        ADD-RECORD.
            ADD 1 TO WORK-KEY.
            WRITE ISAM-RECORD-1 FROM WORK-DATA;
                INVALID KEY PERFORM ERROR-1.
        ERROR-1.
            DISPLAY "INVALID KEY: ", ISAM-KEY-1.
        TEST-READS.
            PERFORM READ-HIGH.
            PERFORM READ-HIGH.
            PERFORM READ-LOW.
            PERFORM READ-LOW.
        READ-HIGH.
            MOVE HI-NUMBER TO ISAM-KEY-2, WORK-KEY.
            READ ISAM-FILE-2; INVALID KEY PERFORM ERROR-2.
            SUBTRACT 1 FROM WORK-KEY GIVING HI-NUMBER.
        ERROR-2.
            DISPLAY "INVALID KEY: ", WORK-KEY.
        READ-LOW.
            MOVE LO-NUMBER TO WORK-KEY, ISAM-KEY-2.
            READ ISAM-FILE-2; INVALID KEY PERFORM ERROR-2.
            ADD 1 WORK-KEY GIVING LO-NUMBER.
        END-PROGRAM.
            EXIT.
** CIS COBOL V4.4 REVISION 0                                    URN rp/
** COMPILER COPYRIGHT (C) 1978,1981 MICRO FOCUS LTD
** ERRORS=00000 DATA=00705 CODE=00703 DICT=00612:01271/01883 GSA FLAG
```

REVIEW OF SOFTWARE BY CLEARBROOK SOFTWARE GROUP

DEDIT

Overview

DEdit is a screen-oriented editor which is intended for use on non-text files. It displays the contents of a file (or an entire disk) in what amounts to "dump" format for inspection and modification. I think it should be possible to configure DEdit for any terminal that has direct cursor positioning support.

Details

DEdit is a minimal, but adequate, editor. It has a set of 16 one-keystroke commands consisting of five cursor control keys (up,down,left,right, and home), eight editor control keys (exit, reread sector, next sector, previous sector, write sector, and read specified sector), three keys to control a "find" facility (find, again, and abort find). The remaining two commands control the edit "windows."

DEdit can display either two or three windows at a time. The main window is a hexadecimal format display of a sector of the file being edited. To the right of the hexadecimal display is an ASCII display of the printable characters in the sector. The third window, positioned near the top of the screen, is the binary representation of the character which the cursor is positioned to. There is a command to move the cursor from window to window, and a command to turn the binary window on and off.

The characters corresponding to the 16 commands are specified by a table in the DEdit program who's location is given in the documentation. I found the choice of command characters strange, and promptly changed them to something I could remember; it was fairly easy to change the table with debug (I could have used DEdit).

Another table in the DEdit program describes the characteristics of your terminal. DEdit is supplied configured for the TVI 910, but can be altered to work with most other terminals. The terminal must support direct cursor positioning, and a clear screen/home cursor sequence that is no more than four bytes long.

Limitations

There are no serious drawbacks to DEdit. It works reliably, and has enough features to be useful. DEdit seems to have been designed to be used for emergency repairs to directories, and other special purposes. This kind of use doesn't call for a feature packed editor; still, I am disappointed in the bare-bones approach Clearbrook took to this problem.

The three main uses I would have for this kind of editor are placing special characters in text files, fussing with directories (unerasing files), and zapping modules. For zapping text files it would be nice to be able to eliminate the hex window for quick scanning of a file. Perhaps a one byte hex window could be kept at the top of the screen the way the binary window is. Fussing with directories would be much easier if the editor would format the directory in a meaningful way -- the format is right there in the System Programmer's Manual, but I appreciate programs that make things easier for me. The module format is another one that could be displayed more meaningfully. The module header could be separated from the rest of the module and the parts of it labeled. Disassembled format would be another nice feature.

Even a bare-bones editor needs a good manual. DEdit's manual is not good. For anyone but a brave and experienced hacker the poor documentation could entirely rule out use of this program. The commands are all described on one page together with directions for changing the command characters. The terminal description table is described in two and a half pages in sufficient detail that any experienced assembly language programmer should be able to configure the program for a terminal ... eventually. I spent quite a while learning that the numeric fields were unsigned. Despite the fact that one- byte fields are almost always signed in 6809 machine language, the unsigned nature of these fields was never mentioned. Perhaps it should have been obvious to me (after all, cursor positions are never negative), but I used about ten minutes figuring this out.

Summary

For someone who needs a disk editor right now, DEdit would be a useful utility. For anyone who can wait, I would recommend hanging on; this program has more features than the various free ones I know of, but not enough to make it worth the money to someone who will only use it occasionally.

BT9

Overview

BT9 is a module which maintains files in a binary tree format. The BT9 module can be called from Basic09, or any other language which can use the Basic09 calling sequence.

Details

Fourteen commands can be passed to BT9 allowing random access, and sequential access (forward or backwards) to files built and maintained through BT9. The commands are:

```
- Read file header
        (done after open)
- Write file header
        (done before close)
- Read a record
- Write a record
- Add a record
- Quick add
- Update current record
- Delete current record
- Find = key
- Find >= key
- Find record with lowest key
- Find record with highest key
- Find next higher record
- Find next lower record
```

The documentation is marginal, but there is a demonstration program which is a great help.

Limitations

There is a tremendous problem with this program. They picked the wrong file structure. A binary tree isn't even the data structure of choice for use in high speed memory when the data is not static. A plain binary tree like what BT9 uses will structure data in the worst possible way when the data is added in sorted order. On disk the idea of a binary tree is just horrifying.

Trees from a family called bichromatic trees have the characteristic that they don't grow long branches causing poor performance. Most data structures texts include AVL trees and 2-3 trees from this family. The most popular, and probably the best, type of tree to use on disks is the B-tree. A B-tree is something like a binary tree, but it doesn't have nasty habits like the binary tree, and it is much faster than the binary tree when the data is on disk.

After accusing this program of using the wrong data structure it hardly seems reasonable to bother with any other problems, but there is one other major problem that would be a good deal easier to fix. BT9 takes a rather casual attitude toward disk errors. They are reported to the user, but not reflected to the calling program in any way. This sloppy treatment of errors makes it impossible for a program to attempt any form of automatic recovery from disk errors.

Summary

I can only imagine using this module to solve a problem in a quick and dirty way. It would not be very difficult to write a module which was upwardly compatible with this one but used a B-tree data structure.

[7] Since this review was written Clearbrook has announced a package that supports B-Trees.

D-SERIES UTILITIES -- DDIR, DDEL, DCOPY AND DATTR

Overview

I bet every OS-9 user has been waiting for these programs. They are generalizations on the DIR, DEL, COPY, and ATTR commands which work on multiple files.

Details

All the D-Series command include a <file spec> option as one of the parameters. The <file spec> can include part of a file name, an attribute of the file (ie D S PR PW PE R W and E), a user number, or 0 to indicate the user's own files. These can be combined with AND, OR, NOT and parenthesis. The <file spec> is used by the D-Series commands to select the files which will be acted on.

The DDIR command gives a simple (one column) list of all the files which meet the selection criteria in a specified directory and, optionally, in lower level directories. The output of of DDIR is formatted by indenting for each level into the directories which makes it quite readable.

The DDel command deletes all the files meeting the selection criteria. It can optionally prompt before deleting each file, list the files as they are deleted, and search the specified directory and all lower level directories.

The DAttr can change the attributes and owner of files meeting the selection criteria. The options are the same as for DDel with the addition of a set of options to specify the new file attributes and owner. Only a superuser (user 0) can change the owner of files he doesn't own.

The DCopy command copies all files meeting the selection criteria from a selected directory to another selected directed. The options are:

- prompt before copying
- list files as they are copied
- copy files in specified and lower directories
- make directories
- delete files which already exist in destination directory

Problems and Limitations

The file selection criteria used with these commands are not as elaborate as the criteria the equivalent UNIX commands use. In particular there are no wild cards or pattern matching options.

If any command line parameters are specified for the DDir, DDel, and DAttr commands the directory name must be included on the command line. Most OS-9 commands

use "." as the default directory, and the D-Series commands do if no parameters are given. It seems inconsistent that they don't always allow the directory to be defaulted.

The DDIR command sometimes[8] returns with an error 211 (end of file). It is my theory that this is returned from the last read of the directory, and is entirely innocent, but shouldn't be returned to the shell.

minor complaint about the documentation: every time "pr pw" should have been in the documentation, "pr pr" was there instead.

Summary

The D-Series commands are very useful. They are not without faults, but, especially for systems with large capacity disks (lots of files), they are almost essential.

Reconsideration

I sent this review to Clearbrook Software Group for their comments. They returned the letter which I have included with this review, revisions to the documentation for the editor, a new editor, and a config program for the editor.[9]

The documentation is much improved, although it still suffers from having been written by a knowledgeable person. They include directions for recovering a deleted file which are useful, and what appears to be a screen dump of a dedit screen which is also nice to have. With the original documentation, I spent a while trying to use DEdit before I realized that I had it configured wrong.

I had trouble with the "config_dedit" program they sent me; it sometimes just stopped. It was a packed Basic09 program so I couldn't tell whether it was an incompatibility with my Basic09, or something else.

The new version of DEdit is identical to the one I already had. I ran the OS-9 compare utility, cmp, against them and found they only differed in the bytes that are changed when the terminal configuration is done.

The improvements to the documentation are nice, and the configuration program seems like it might make it a good deal easier to set DEdit up for a terminal, if it would work.

---

[8] This problem (error 211) has since been fixed

[9] The letter isn't included here, but it detailed the improvements they had made, and said that DEDIT did what it was intended to do very nicely.

A REVIEW OF DYNACALC FOR OS-9

OVERVIEW

DYNACALC is a very capable electronic spread sheet program. It is enough like all the other spread sheet programs (visiclones) so anyone familiar with one of them should be able to adjust to DYNACALC very quickly. DYNACALC is not a great leap beyond all other electronic spread sheets, but it is a very good example of the current state of the art.

A electronic spread sheet program makes the terminal appear to be looking at a section of a large grid. The "cells" in the grid can each contain a number, equation, or character string. The equations usually operate on the contents of other cells (A column of cells might contain monthly expenses, and another cell somewhere on the grid might contain the sum of all the cells in the column of expenses.) The special thing about electronic spread sheets as opposed to paper spread sheets is that when a number or equation on an electronic spread sheet is changed, all the cells that depend on that value are updated to reflect the change. This is a simple idea, but such a good idea that I know of many people who have purchased computers just to be get at this kind of program.

SOME DETAILS

DYNACALC is a large 6809 assembly language program which has been available under FLEX since last year. It is now also available under OS-9. The OS-9 version doesn't seem like warmed over FLEX code; it seems to have been designed for OS-9. It is reentrant (the same module can be used by any number of simultaneous users), and uses standard input and output. Practically any CRT type terminal can be supported. In fact, the warranty for DYNACALC says that if you have a CRT terminal with at least 80 characters per line and direct cursor addressing:

> If your terminal has the required characteristics, but you are unable to configure DYNACALC to work properly (using the INSTALL utility), send us your original DYNACALC diskette and a copy of the operator's manual for your CRT. We will either make it work on your terminal, at no extra charge to you, or refund your full DYNACALC purchase price.

That is a very impressive commitment! If you have several users on your system with different types of terminal, you can get DYNACALC to support them all concurrently if you have each terminal type use a different data directory, and put the appropriate terminal file in each directory.

DYNACALC can save the contents of a spread sheet in a file that can be read by other programs. I wouldn't call the files easy to use, but they aren't impossible to use either, and the format is clearly documented. DYNACALC's saved data is hard to use because the format of the file reflects DYNACALC's flexible attitude towards the user - it will take any sort of data scattered around anywhere you like. If you want to create a file for DYNACALC to use as data for a spread sheet, you don't have to cope with the vicissitudes of humans. It is a relatively simple job to create data files for DYNACALC.

An excellent help facility is an integral part of the program, though you can remove it to save space if you want. Most of the time you can type a "?" to access a screen of terse explanations of your options. The help screens do not take the place of reading the manual, but they can provide a quick jog of the memory. There are also 12 error codes which I wish all visiclones had. Spread sheets can take on some of the attributes of complicated programs, especially hard to find bugs. Imagine trying to debug a program with only one error message like "Sorry, I can't do that," "Say Wha?" or whatever.

My copy of DYNACALC came the terminal files listed in Figure 7. I recognize SWTPC, Hazeltine, Adds, Heathkit/Zenith, ADM, and Televideo in there. Even if your terminal isn't in that list, you can use the INSTALL.DC utility to build a terminal file for your terminal.

ct_82	ct_82_92	c8200	c8200_92
c82_w	c82_w_92	h_1400	h_1420
h_1500	adds_vpt	adds_3a	h_19
act_iv	adm_3a	ansi_	tv_912
pe_550	info_100	iq_120	tv_950

Figure 7: Dynacalc Terminal Support

A particularly strong point of DYNACALC is the set of powerful functions it supports, including basic math (trig, log/exp, square root, max/min, pi, int, round, and absolute value), "group" functions (sum, average, standard deviation, net present value, choose, lookup, and index), and a bunch of miscellaneous functions. Choose selects the nth entry from a list, lookup is the standard visiclone lookup function, and index is like lookup except that it scans for an exact match instead of greater than. Many of DYNACALC's functions work with either character strings or numbers. This expands the usefulness of the functions substantially.

DYNACALC has commands which move rows and columns around, and do insert and delete operations on them. The fanciest

command in this family is the sort command, which allows you to sort rows or columns based on the values in a column or row respectively.

I have never been entirely pleased with the speed of any program. Of course I wish DYNACALC ran faster, but I don't remember using a spread sheet program on a microcomputer that ran faster.

LIMITATIONS AND PROBLEMS

The only real problem with DYNACALC is with its terminal support, and I'm not sure it could have been done much better without losing generality. The terminal support problem is not a major one. In fact, I imagine that after a few months of using the program I will feel nothing but affection for it.

It is hard to choose characters to use as arrow keys. DYNACALC uses curly and square brackets as cursor control keys by default. This is a good choice if you want to drive it with a disk file, but not very intuitive. If you like this choice as little as I did you can change it with INSTALL. Unfortunately install only allows you to use single characters as control keys; my terminal, like most terminals, sends escape sequences when the arrow keys are pressed.

Screen updating is not as fast and smooth as it is on machines that have integral screen support. I understand that a 9600 baud terminal can't possibly compete with memory mapped video, but I believe that, if the insert and delete character and line facilities on my terminal were used, the screen could be updated more quickly. It would have been hard to make DYNACALC support more advanced terminals while still supporting "dumb" terminals, but I wish it had been done.

SUMMARY

DYNACALC is a fine program, but although it seems to have been written by a programmer familiar with OS-9, it doesn't make the fullest use of the power of OS-9. I wish DYNACALC could use all available memory instead of just 64K, and I wish printing was handled by a separate process so I could start a copy of a sheet printing, then continue work on the original. Extended memory probably could have been used under Level Two without degrading the program under Level One, and multiple processes are supported by both levels of OS-9.

I find myself expecting a great deal of DYNACALC. My carping at its terminal support (which is in many ways unusually good), and pushing for support of fancy OS-9 features is a reflection of my very high opinion of the program.

I know people who find it reasonable to buy a personal computer just to have an electronic spread sheet. DYNACALC is an excellent spread sheet program. It can help with any number of business problems, simple problems in the sciences, and just plain showing off the computer to the uninitiated. I think DYNACALC is a program which should be included in the toolkit of most OS-9 users. One warning, spreadsheet programs tend to be popular. I am afraid that I will have to wait for a crack at my machine more often now that I have DYNACALC on it.

REVIEW OF DYNAMITE

OVERVIEW

Dynamite is a disassembler for the 6809/6800 sold by Computer Systems Center. The version I tested runs under OS-9, but there are other versions for FLEX and Uni-FLEX. Disassemblers are able to convert a file of executable object (machine) code into a program in assembly language. It is important to realize that Dynamite won't work on intermediate code, such as Basic09 packed files, and it won't always convert object files into the original language. Dynamite can convert an executable object module generated by any language into assembly language. Even if the program was written in a higher level language like Pascal or C, Dynamite will only produce assembler.

If you have reliable software and don't like to dig around in your system much, you have no need for Dynamite. Don't waste your money. If you would like to fix (modify) your software, or just want to understand it as only someone with the source code can, Dynamite, or some other disassembler, is valuable. I have disassembled many pages of code by hand. Those hours of work qualify me to say that disassembly is just the type of work which should be left to computers.

SOME DETAILS

Dynamite can be used to get a quick look at source that could have generated an object file. The command:

```
DYNAMITE filename a
```

will disassemble the module in the file called filename and send its output, which looks like the the output of an assembler, to the terminal. The "a" option tells Dynamite to give the ascii equivalent of each printable character it encounters during the disassembly. This simple disassembly is enough in many cases. If the module is more complicated than is easy to understand without meaningful labels, the next step is to help Dynamite do a better job of decoding the module until its output is understandable.

Table 2: Dynamite Label Classes

D	Direct references
L	PCR references
X	Extended references
$	Hex constant
&	Decimal constant
@	Decimal or Hex constant depending on magnitude
¬	ASCII constant
!	System function name

Table 3: Dynamite Addressing Modes

```
#1 - one byte immediate (any register)
#D - Immediate with Accumulator D
#X,#Y,#U,#S - Immediate with other registers
X,Y,U,S - Indexed by X,Y,U, or SP
D - Direct page
E - Extended addressing
R - Relative
```

Dynamite doesn't distinguish between data and instructions while disassembling. This results in some very strange output as blocks of constants are disassembled. Even the name of the program pointed to in the module header is decoded into assembly language instructions. The "a" option makes it easy to find the data areas, and Dynamite can be told where they are either through its standard input or in its command file. Once Dynamite knows where the data areas are, it will stop disassembling them as instructions. Instead, it will label the entries in the data area, and disassemble them into constants (fcb,fcc,...).

When Dynamite is run without any guidance, it invents names for everything it encounters that might have had a name in the original program. Addresses, offsets, and immediate data all are given names. Names for immediate data and offsets are useful. Names for offsets in PCR instructions are VERY useful because, although different references to a location will have different PCR offsets, Dynamite resolves them to the same name.

An assembly language program more than about a page long is hard to read unless it has meaningful names. Dynamite gives names that consist of a letter and a number. More meaningful names can be assigned by using a label file.

Dynamite can use two classes of files with label definitions in the form of equates. It always uses a "system name" file which contains the names used for each OS9 call. When the instruction:

```
OS9 I$Open
```

is decoded the "I$Open" comes from the system name file. The second file full of label definitions is the "label file." The label file's name has to be given in the Dynamite command line. Each line in the label file is of the form:

```
label EQU value class
```

for example:

```
Init EQU $24 L
```

Where Init is the label, $24 is the value and "L" is the class. Initially eight label classes are defined: see Table 2. These classes are sufficient for a simple disassembly, but I found myself defining additional classes very soon. A class is defined by putting some labels in the label file with that class. All the unused letters A..Z can be used as new classes. For example, when I disassemble modules from OS-9, I usually have to define labels for offsets in the System Direct Page, and the process descriptor. For the System Direct Page the D class is fine, but for the process descriptor I have to define a new class. I usually use P.

Dynamite will use its default classes of labels wherever they are appropriate unless it is given instructions to use another class of label. A good disassembler needs to be able to assign labels to values very specifically. Although 8 is the offset of the P$User in the process descriptor control block, it wouldn't generally be a good idea to assign the name P$User to the value 8 throughout a program. Dynamite gives you two ways to limit the scope in which a label is used. A class of label is activated by a command of the form:

```
<mode> <class> [<offset>]
<range>
```

The modes are listed in Table 3. The class is a default class, or one defined in the label file. The offset is added to a value before the proper label is looked up, then included in the disassembly listing. This could be used to generate instructions like:

```
lda #CR+$80
```

in the disassembly. The range gives the range of offsets from the start of the module being disassembled over which the mapping given by this command is in effect.

Commands can come either from standard input after Dynamite is started, or from a command file.

If the reason for disassembling a module is to learn how it works, the listing generated by Dynamite should be enough. If the goal is to revise the original program, Dynamite can generate a file which contains source which can be assembled with the Microware standard assembler, or any compatible assembler to give a module identical to the original.

The OS-9 version of Dynamite expects to disassemble 6809 instructions from a file with modules in OS-9 format, but there is an option which causes it to disassemble a file into 6800 instructions and another option which tells it to expect to find the module in Motorola or FLEX format instead of the usual OS-9 format.

OPERATION

I use Dynamite to sort of chew away at the edges of a program until I have it reduced to an understandable listing. First I let Dynamite have its head, and produce a listing using all its defaults. Using this listing, I start building the labels and commands files. At first I just define the data areas and a few labels. Then I go through a cycle of running Dynamite then using the output to refine and extend the contents of the commands and labels files until the listing satisfies me. Then I ask Dynamite to generate a file with the source in it. This file is the best I can do with Dynamite. It isn't well formatted, and has no comments. The final polishing has to be done with an editor.

Please realize that if you disassemble proprietary software (such as Dynamite itself) the same laws and moral obligations that should prevent you from passing out copies of the original program apply to the disassembled program.

LIMITATIONS

When I first tried to use Dynamite, I had a terrible time. I blamed the documentation. Determined not to be unfair, I sat down and read the manual from start to finish. I won't say it was easy reading, but once I had chewed my way through it I understood how to use Dynamite. The manual is a little brief for the manual of a program that does such tricky work, but it is complete. It is not set up to be skipped through!

Dynamite's advertising might lead a person to believe that disassembling a module with Dynamite is easy. You run Dynamite against a file and it falls apart into neat code. This is not true at all... disassembling a module is hard. You have to figure out all the tricks the person who wrote the program used. This is not too hard to do for a short, simple program, but long tangled modules are much harder to disassemble than they are to read in commented source form, and some modules are hard to understand even when the original source is in front of you.

It seems a little silly to design a disassembler with the ability to insert comments in its output, but Dynamite is such a complete product that I am a little disappointed that there is no way to include a "comment file" in the input for Dynamite. I understand that Computer Systems Center is working on this shortcoming.

SUMMARY

I am very impressed with Dynamite. It does about as good a job of helping a person to disassemble a module as it can do. For example, if Dynamite finds that a label falls in the middle of an instruction, it

throws in an ORG to adjust the PC so the label falls at the start of an instruction. This keeps data areas from throwing the disassembly out of whack; usually if there is a data area in a program, there is a reference to the first instruction after the data area which Dynamite can use to get itself lined up again if it hasn't been told that the data area is there and has gotten itself wrapped around the axle by trying to turn data into instructions.

Dynamite is designed to be useful for several different types of disassembly. The quick disassembly can be done without building any files. The most important information can be supplied interactively. Used this way Dynamite can produce a usable listing in just a few minutes. The full power and flexibility of the program shows up when a higher quality listing is the goal. Dynamite lends itself to the process of successive refinements that leads to a clear disassembly.

I don't recommend Dynamite for every OS-9 user. In fact, I imagine there are not many OS-9 users who have a need for this type of software, but for those who need a disassembler, Dynamite is everything it should be.

A REVIEW OF RMS

RMS (Record Management System) is a primitive, but useful tool for organizing and processing data. It isn't a database system, or even a polished record management system, but, nevertheless, I rather like it.

OVERVIEW

RMS stores data using at least two files. The _rms file contains data. It must be formatted in advance using the RMSNEW utility. The _dic file contains a description of the data in the _rms file. The _dic file must be created with a text editor before any data can be placed in the _rms file. A third file type _ndx (index) is used when a _rms file must be sorted on some key other than the one designated in the dictionary file. Many _ndx files can be generated, one for each ordering of the file. Index files can be created with the INDEX utility, or any other program that generates a file with a key value on each line.

RMS has to know many things about your terminal before it can be used. A file called rms_trm must be build with a text editor and placed in the root directory of /DO, or the directory which will be the default data directory when RMS is run. The rms_trm file must contain the hexadecimal representations of 88 bytes of data including 31 terminal characteristics and command codes.

SOME DETAILS

RMS saves information in record groups consisting of one "primary" record and any number of "secondary" records related to the primary record. The secondary records aren't required, but they are important when a variable amount of information is to be associated with each primary record.

I use primary and secondary records in the database of Prairie Home Companion (an excellent program on Public Radio each Saturday evening) programs I keep. Some information about each week's show fits nicely in the primary record: a date, and a comment to act as a title for that week's show. I maintain secondary records to save the names of the guests, notes on each monolog, and notes on each "advertisement." I use the secondary records because although I could probably put a ceiling on the number of guests, monologs and adds that might occur in a program, the ceilings would have to be much higher than the usual numbers. RMS assumes that all fields will have data in them when it allocates space for a record; so leaving space for data that isn't usually needed would waste lots of file space. Since I only use as many secondary records as I need, they use space comparatively efficiently.

The dictionary file associated with each RMS file defines the structure of the data in the file and the way the records are displayed on the screen. If secondary records are used, the dictionary file contains the formats for primary and secondary records.

The first line in the dictionary file contains the title for the primary records. This title is displayed on the screen when the RMS editor is being used to edit a primary record. Lines following the title line are used to define fields in the record, one field per line. The first field is the "key" for the record. The key can be used to select records for editing very quickly. The line defining a field contains the name of the field, the length of the field, the type of data to be stored in it (alphanumeric, numeric, money, or date), the prompt to use in the editor, and various data validation options. The field can be made optional, a minimum length can be specified, and a range or list of acceptable values can be given.

The dictionary file I use for my Prairie Home Companion file demonstrates some of the features of RMS. I include it here as an example.

```
"P r a i r i e   H o m e   C o m p a n i o n"
DATE       8    D    "Date aired:" ;
COMMENT1   50   A*   "Comments: " ;
COMMENT2   50   A*   "          " ;
POINT1     15   A*   "Special Notes: " ;
POINT2     15   A*   "              " ;
$
"D e t a i l s"
DATE       8    D    "Date aired:" ;
TYPE       1    A    " Type (Sponsor, Powdermilk,Monolog, Guest, Other):" [S,P,M,O,G] ;
SUBJECT    50   A*   "Subject: " ;
SUBJECT2   40   A*   "         " ;
SUBJECT3   40   A*   "         " ;
$
```

Figure 8: Sample RMS Definition

The only fields on which I used validation are the date fields, which RMS validates for possible dates, and the TYPE field, which I only permit to take one of five values. RMS formats the data on the screen using a few simple rules. The fields are placed in order starting in the upper left corner and working left-to-right and top-to-bottom. RMS won't split a field and its prompt between two lines. It is possible to have some effect on the screen

format by adding leading blanks to prompts, but not much. Trying to do something radical -- like add a blank line -- makes a terrible mess on the screen.

The dollar signs mark the end of each record definition.

The lengths of the primary and secondary records are 138 bytes and 139 bytes. I kept the lengths about the same because RMS allocates only one size of record. It must leave space for the largest possible record; so, when two record formats are used, the difference in size between the two records is wasted for each small record stored in the file.

I used an important trick on the secondary record. Since RMS only understands two record formats, I used the secondary format for five different types of records. The TYPE field in a secondary record indicates the meaning of the information in the rest of that record.

The RMS editor is used to add, delete, and update records in an RMS file. It is also able to search through the file either sequentially, or by key.

The RMS report writer is powerful, but not versatile. It takes as input an rms file, a report specification file, and sometimes an index file. The report specification file is something like a program. The language used reminds me of RPG. It can contain commands which select or exclude records. Any number of lines can be printed for each record selected. Special lines can be printed at the start of a report, at the end of a report, at the end of a group of records, and at the top of each page. Page breaks happen when a page is full, or (optionally) after each primary, or secondary record is processed. There are no arithmetic commands in the report generator, but various accumulators are kept: the number of selected records, the number of selected groups, the number of selected secondary records, and totals and subtotals for each numeric field.

By default reports are generated with records sorted in ascending order on their key. Other orders can be specified by using an index file.

Index files can be generated by the INDEX utility. INDEX produces a file that contains a list of record key values. If records are selected from the RMS file in the order specified in the index file they will be in the order specified when INDEX was run to create the index file. For example:

```
INDEX phc Points Point1
```

would generate an index file called Points which could be used to sort the phc RMS file in ascending order on the POINT1 field. Index files can be edited to generate orderings that are beyond INDEX's abilities.

The RMSCOPY utility can be used to copy RMS files, but it can also do much more. RMSCOPY can be used to add fields to a file, remove fields, or merge similar RMS files.

FLAWS

Setting RMS up is exceptionally difficult. It took me hours to get the _trm file right. The worst part of my problem was that RMS didn't help me uncover problems, it just wouldn't work. I have a terminal which uses ANSI standard control sequences which some programs have trouble with. Other people might not have quite as hard a time as I did.

The documentation keeps referring to file names with dots in them, but RMS always uses underscores. I called the program's author to ask about this. It seems that when RMS was written for OS-9 dots weren't allowed in file names. I assume that there is a FLEX version of RMS which uses dots where the manual says they should be. Since OS-9 now permits dots in file names, RMS could be adjusted to fit its manual, or the document could be updated to reflect the use of underscores. That neither of these things has been done indicates a negligent attitude that is disturbing.

It is practically impossible to format the screen any way other than the way RMS wants it. This would be easier to take if I liked the way RMS formats the screen. I prefer to use up the whole screen, and RMS packs the fields as close together as possible.

RMS's file structure is wasteful of disk space. Since it can't handle variable-length fields or records, it uses more space per record than is necessary in almost every case. It also has to format the entire file before any records can be placed in it. It would be more consistent with OS-9 conventions to start off with a small file and enlarge it as required.

Index files aren't automatically updated. That means that if you generate an index file, then insert or delete records in the _rms file, the index file is out of date and has to be made over again. It is easy to forget to make new index files, and RMS doesn't do anything to make it easier.

SUMMARY

I find RMS useful, but frustrating. It is not a database program; it doesn't even pretend to be. Before I could discover how useful RMS is I had to get it set up and got used to its limitations. These were so discouraging that I almost gave up on the program. I'm glad I didn't. I use the RMS editor as tool for searching quickly through large files, and generating reports on the contents of those files. I wish RMS could deal with multiple keys, but, for many applications, one key is plenty. As a report generator RMS is quite good, including all the most commonly used features. It would be better if there was some way to do arithmetic, but I'm surprised how well I can make do with what's there.

I had heard that RMS was inclined to crash, but I haven't been able to get it to do anything unexpected except when I messed up its _trm file, or tried to get it to format the screen in a way contrary to its nature.

RMS is not a highly polished program. In fact, it's primitive. Not primitive in a sloppy sense ... more simple and rough-hewn ... like a well build log cabin. It makes me want to write a real database program for OS-9, but, since I probably won't get around to that, I expect that RMS will continue to get a moderate amount of use around here.

REVIEW OF RMA AND RLINK

RMA (Relocating Macro Assembler) and RLINK (Relocating Linker) are new programs from Microware. They are required for C (and probably for future languages from Microware), and are currently bundled with C. Those who already have the C compiler from Microware shouldn't consider purchasing RMA/RLINK -- they already have them under the names c.asm and c.link.

OVERVIEW

It is easier to explain RLINK's purpose than RMA's. RLINK takes one or more files created by RMA and turns them into an executable module. RMA is a tool which makes writing large programs easier with a moderately good macro facility and a variety of tools which permit a program to be divided into several pieces which can be assembled separately.

This separate assembly is the really important part of RMA. With separate assembly it is easy to build a library of procedures which can be called from any program. Structured programming requires that each procedure be as independent of other procedures as possible. It is much easier to do this when each module has clear connections to other modules -- in particular, any shared data should be noted; RMA makes it easy to isolate procedures, and makes it hard to hide shared data.

SOME DETAILS

RMA includes the usual conditional assembly statements:

- *FAIL*

  Generates an assembler error and a message.

- *IF/ELSE/ENDC*

  Do just what they should. ELSE is optional.

- *REPT/ENDR*

  repeats a set of statements a specified number of times.

These statements can be used in the body of a program, or in macros. Macros amount to procedures, or specially defined instructions which can be used very much as if they were 6809 instructions. A macro is defined by the MACRO/ENDM statements. A macro can be given parameters which are referred to within the macro by a backslash followed by a number: \1 would be the first parameter, \2 the second, etc. The number of parameters given is available through the special operator \#, and the length of any parameter is available through the operator \Ln where n is the number of the argument whose length is in question.

```
Swap MACRO                    exchanges bytes in memory
*       arg1 -- points to memory location
*       arg2 -- another location
*       arg3 -- the number of bytes to swap (a constant)
          IFNE \# - 3           check the number of args.
          FAIL Swap: must have exactly three arguments
          ENDC
          pshs A,B,X,Y
          leas -1,S             Make work space on stack
          leax \1,U             address of first variable
          leay \2,U             address of second variable
          ldb #\3               number of bytes to swap
          ble \@Lx              if none; stop
\@Lp      lda B,X
          sta ,S
          lda B,Y
          sta B,X
          lda ,S
          sta B,Y
          decb
          bne \@Lp
\@Lx      leas 1,S              clear work space
          puls A,B,X,Y
          ENDM
```

Figure 9: RMA Macro

When a macro needs unique labels, RMA offers the \@ operator. This operator returns an @ followed by a number unique to each invocation of each macro.

A sample RMA macro can be found in Figure 9. This macro could be invoked with the statement:

```
Swap Var1,Var2,20
```

which could be used as many times as necessary in a program with Swap defined.

The Separate Assembly Facility

RMA includes statements which define three different "program sections."

The PSECT section contains program code and constants. RMA can only deal with one PSECT per assembly. The PSECT statement includes all the data given in the MOD statement in ASM except the module length, but only the entrypoint argument to PSECT is an address. The parameters are:

- *NAME*

  Up to 20 byte name for the PSECT

- *TYPELANG*

  the type/language for the PSECT

- *ATTRREV*

  the attribute (ReEnt ?) and revision level of the PSECT

- *EDITION*

  the edition number to be used for the executable module.

- *STACKSIZE*

  The estimated size of the stack for this procedure.

- *ENTRY*

  The Label used for the first instruction to be executed in the PSECT.

If the PSECT is the mainline segment of the program being written, all the arguments must have values; for example:

```
PSECT Example,Prgrm+Objct,
      ReEnt+1,1,250,EntryPt
```

Procedures which are used as subroutines must have zeros in some fields; for example:

```
PSECT SubProc,0,0,0,100,0
```

The PSECT section contains only constant data: instruction mnemonics, OS9, fcc, fdb, fcs, fcb, rzb (reserve zero-value bytes, VSECT, ENDSECT, and END. In particular rmb is not allowed in a PSECT.

The VSECT section reserves memory locations. It has two forms:

```
VSECT DP
```

reserves space in the direct page, and just

reserves space outside the direct page. The VSECTs are used for the variables that would normally be addressed off the U register in an OS-9 program Normally only the rmb instruction is used in a VSECT, but for elaborate programs it is possible to have variables automatically initiallized. If you are willing to include the initiallization code in your program (it is included with RMA) you can use fcc, fdb, fcs, fcb, and rzb in a VSECT along with rmb. It is important that there is no official way to know where variables allocated in a VSECT will be relative to other variables. Your program will be able to find its variables, but finding relationships between the addresses of variables at assembly time is hard.

As many VSECTs as convenient can appear in a PSECT.

If VSECT is used inside the PSECT, as it usually is, it will cause the linker to allocate space for the variables in it. If a VSECT is placed outside the PSECT it will make the variables in the VSECT known in the code, but not allocate any storage. This is a useful trick for cases when you know that a block of variables has already been allocated and you want access to all of them. I haven't tried this, and I can't find it in the manual, but Microware declares it will work.

A CSECT is just a way to assign values to names. They are used extensively in the DEFS files for RMA. Only the rmb statement can be used in a CSECT. If the CSECT statement is given an argument, that argument is the starting value in the CSECT, otherwise the values in the CSECT start at zero.

Every program sector must be terminated with an ENDSECT. A PSECT can contain other sectors, but in general sectors should not be nested.

A label can be made globally available by following it with a colon ":" when it is defined. If a label isn't global, it is only known in the PSECT where it is defined. If a label isn't global, it can be used to represent a different thing in each, separately assembled, file.

Speaking of labels: RMA permits labels up to nine characters long and always distinguishes upper and lower case letters.

The files that are produced by RMA, called relocatable files, can be decoded by a program called RDUMP which is included with RMA. RDUMP can give anything from a quick summary to an exhaustive dump of information about symbols referenced and defined in the file being investigated.

SOME INTERNALS

Since RMA has no way of telling what offsets RLINK will assign to variables defined in VSECTS, it is often unable to use the small-offset forms of the indexed instructions. References to data in VSECTS are assembled as 16 bit offsets. RMA records

information about variables defined and used in a PSECT which is used by RLINK. RLINK goes through the files it is linking filling in the blanks left by RMA.

RLINK accepts a list of files to link and libraries to use. It will link all the files on the command line even if the mainline PSECT doesn't reference anything in them. If there are any references left unresolved, RLINK will search the library(s) for the PSECTs needed to resolve the references. A library is simply a group of PSECTs merged together: the MERGE command does this nicely. PSECTS in a library can call one another, but, since the library is read sequentially, unresolved references must be to PSECT further along in the file, or in another library which will be searched later.

LIMITATIONS

I haven't been able to discover an easy way to have RMA calculate the length of a group of variables in a VSECT. The concept of a useful data position counter ("." in ASM) doesn't exist in RMA. There are several counters (Direct Page, Uninitialized data, and initiallized data), and, in any case, the linker has the last word on addresses. I got used to this problem, and I can't think of any way for Microware to design it out of RMA without introducing other problems, but it is a serious problem. The lack of a "." caused other habits I have to generate errors as well.

RMA's inability to determine offsets in a VSECT causes the 16 bit offset instructions to be used more than they are in programs assembled with ASM. These instructions are relatively long and slow. At first this really upset me, but my experience and Microware's indicates that it isn't a significant problem. I converted several very large (5000 to 10000 lines of code) programs from ASM to RMA and they generally got a little smaller. Microware declares that they have converted Basic09 from ASM to RMA, and that it got a little smaller through the conversion. I attribute the small decrease in size to better coding habits that RMA encourages. Still, in the last analysis, programs assembled by ASM can be made to run faster than programs assembled by RMA.

This is really nit-picking, but the command line option which should set the width of the listing which RMA can produce doesn't work. It's not that important, but little problems like that could give a less forgiving person than me a bad impression that would spoil the excellent job done on the really important parts of the product.

I found several problems in the first copy of RMA that I got, some of them quite serious. I now have edition five. If you have an earlier edition, I would strongly recommend getting an update. If you mean to use c.asm as a stand-alone assembler, you should also see to it that you have an up-to-date revision. The problems were tricky things that wouldn't generally show up with correct code, but I haven't been able to uncover any bugs other than the problem with the width of the listing in the current revision of RMA

Converting programs from the standard assembler to RMA is not as simple as one might think. To start with the standard DEFS files won't work, and Microware didn't include complete DEFS files with RMA. I frequently use "." ... that had to be dealt with. RMA can't handle as many symbols as the standard assembler before the symbol table overflows. This meant that I couldn't just convert a program into RMA, I had to use RMA. A large program MUST be broken down into several PSECTs and assembled in pieces then linked.

SUMMARY

I think RMA/RLINK is wonderful. I am a serious assembly language programmer. I write large programs that take a long time to assemble, and have quantities of chunks of code that I "USE" in assembler programs to prevent myself from having to rewrite commonly used procedures. RMA lets me build libraries, and assemble only the small part of a program that I change. I also care about structured programming, and RMA lets me use that discipline for assembly language programs.

Assembly language procedures to be called from C must be written in RMA, and I have been able to call C procedures from RMA programs. RMA comes with the C compiler, but the documentation that is included in the C manual isn't sufficient to make full use of c.asm/c.link. The information I have given in this review may supplement the C manual enough, but, if not, I would recommend purchasing a copy of the RMA/RLINK manual from Microware.

The standard assembler is easier to use for short and simple programs. RMA has a lot more power, and is correspondingly harder to use. Nevertheless, if you are serious about assembler, RMA/RLINK is important to have. Even if the added structure doesn't mean anything to you, the large amounts of time that you won't spend waiting for big programs to assemble will be worth the investment in money and time that RMA requires.

INDEX

ACIA ... 48
Active Process Queue ... 58
ANSI Protocol ... 39
Application Programs ... 35

B-Trees ... 131
BasicO9 ... 15, 35
BasicO9 Installation ... 77
Binary Trees ... 131
Boot file ... 9
  Construction ... 9
BT9 ... 131
Buffers ... 26
Busy Waiting ... 25
BWord ... 60

C Asm ... 145
C Functions - Floating Point ... 96
C Language ... 96, 113
C Link ... 145
Cache ... 49
Calc ... 21
Carry Bit ... 29
Changing Disks ... 78
Changing I/O Configuration ... 74
CharCt ... 60
CHD ... 37
CHX ... 37
CIS COBOL ... 123
Clearbrook Software Group ... 131
COBOL ... 123
COBOL Debugger ... 124
CoCO ... 62, 77, 81
CoCo Disk Driver ... 87
Commands ... 9
  ATTR ... 37
  Backup ... 103-104
  CHD ... 37, 103, 105
  CHX ... 37, 48, 103, 105
  DCHECK ... 113
  DELDIR ... 9
  DIR ... 55, 103-104
  DSAVE ... 9
  EX ... 48
  Format ... 103
  IDENT ... 9
  Kill ... 48
  LOAD ... 78
  LOGIN ... 56
  MAKDIR ... 36
  MDIR ... 78
  MFREE ... 78
  OS9GEN ... 9
  PRINTERR ... 9
  PWD ... 9
  PXD ... 9
  Setpr ... 48
  TSMON ... 56
  UNLINK ... 78
  W ... 48
Compuserve ... 77, 81
Concurrency problems ... 16
  Busy Waiting ... 25, 57
  Execution Sequence ... 25
  Lockout ... 25

D-Series ... 132
D/A Converter (CoCo) ... 89
Data Base ... 141
Data Directory ... 36
DAttr ... 132
DCopy ... 132
DDel ... 132
DDir ... 132
Debugger ... 35
Debugging ... 50
Dedit ... 131
DefsList ... 27
Device Descriptor ... 74
Device Driver ... 6
  Beeper ... 82
  Logical ... 48
  Null ... 6
  Null Program ... 7
  OS-9 Service ... 26
  Service Routine ... 57
  Static Storage ... 49
Device Drivers ... 26
Device Independance ... 74
Directories ... 36
  Anonymous ... 37
  Changing Disks ... 78
  CHD ... 37
  CHX ... 37
  Data Directoy ... 36
  Default directories ... 105
  Dir command ... 104
  Directories as files ... 107
  DirSqz ... 118
  Dr ... 115
  Execution Directory ... 36
  Formatted listing ... 113
  Location on Disk ... 48
  MAKDIR ... 36
  Multiple Links to a File ... 113
  Reading Directories ... 107
  Root ... 36
  Updating ... 113
DirSqz ... 118
Disk Contention ... 26
Dispatcher ... 57
Dissassemblers ... 137
Dr ... 115
Driver.Device ... 57
Driver2 ... 22
DYNACALC ... 135
Dynaform ... 35
Dynamite ... 137
DynaSpell ... 35, 96
DynaStar ... 35

Edit ... 35
Editors ... 88
Enqueue/Dequeue ... 16
Entry Point ... 28
Execute Only files ... 56
Execution Directory ... 36
Execution file attribute ... 108
Execution Sequence ... 25

File Attributes ... 55
File Descriptor ... 108
File Security ... 55
File Sharing ... 55
File.Attributes ... 55
Filter ... 39
Flex ... 95
Frank Hogg Labs ... 35
Fujitsu ... 47

Generic OS-9 ... 107
GETSTAT/PUTSTAT ... 39
Gimix ... 47, 81
GIMIX III ... 6
GOTOXY ... 39
Graphics ... 61
Graphics hardware ... 87

I/O Managers ... 74
I/O redirection ... 73
I/O System ... 73
IFP1/ENDC ... 27
Initial Program ... 56
Initialization Table ... 75
Interrupt Service Routine ... 57
Interrupts ... 57, 89

Isam ... 47, 123

IBM Group ... 47
Johnson, Dan (D.P.) ... 87

Level II Memory Requirements ... 81
Locker ... 18
Locking ... 16-17
Lockout ... 25

Macros ... 145
Memory for Level II ... 87
Memory Management ... 107
Mice ... 88
Mix ... 25
Module Type ... 27
Modules ... 16
  Data Modules ... 16
  Entry Point ... 28
  Locking ... 16-17
  Memory requirement ... 78
  Module Size ... 17
  Module Type ... 27
  Reentrant ... 27
  Revision ... 27
  Shared ... 16
  Storage Requirement ... 28
  Version Number ... 28
MOTD ... 56
Multi-Processors ... 88
Multitasking ... 82
MUMPS ... 41

Non-Standard Hardware ... 107

O-F ... 121
OFlex ... 95, 105
OS9P1 ... 107
OS9P3 ... 97

P/V ... 16
Parameter String terminator ... 28
Paramod ... 42
Pascal ... 5, 35-36
  Brief Review ... 35
  Microware ... 5
  Release 2.0 (Microware) ... 36
  Virtual Storage ... 5
Password ... 56
Password file ... 56
Pipes ... 59
  Algorithm Partition via ... 60
  Bug in PIPEMan ... 60
  Direct from programs ... 60
  Example of a Pipeline ... 60
  Filters ... 59
  Internals ... 61
  Interprocess ... 62
  Overview ... 59
  Shell ... 59
  Words (Filter) ... 59
POpt ... 27, 30
Porting OS-9 ... 107
Printer Options ... 27
Priority ... 26
Privac ... 47
Processes ... 10
  Address Space ... 12
  Background ... 48
  Busy Waiting ... 25, 57
  Concurrency problems ... 16
  Concurrent ... 15
  Execution Sequence ... 25
  Getting a good Mix ... 25
  Interprocess Communication ... 15
  Lockout ... 25
  Parameter Area ... 11, 15
  Priority ... 26
Program length ... 27
Programs ... 13
  Beeper ... 84
  Beeper2 ... 90
  BWord ... 63
  Calc ... 21
  CharCt ... 64
  COBOL Benchmark ... 129
  Cobol Sieve ... 128
  Cobol Test ... 126
  DFormat ... 112
  DirSqz ... 118
  DList ... 109
  DList2 ... 110
  Dr ... 115
  Driver One ... 14
  Driver Two ... 22
  FRexp ... 100
  Getting a good Mix ... 25
  Grapher ... 66
  Help_B ... 45
  Id ... 111
  Locker ... 18
  Modf ... 100
  ParamMod ... 44
  POpt ... 27, 30
  Rast ... 70
  Sound ... 83
  StrtTask ... 67
  StrtTask One ... 13
  TBeep2 ... 92
  TestBeep ... 86
  TstSSig ... 99
  Vcia ... 51
Protection ... 55

Rasterization ... 61
Record Management ... 141
Recovery ... 113
Reentrant ... 16, 27
Revision ... 27
RLINK ... 145
RMA ... 145
RMS ... 141
RunB ... 60

School ... 59
Schools ... 41
Separate Assembly ... 145
Service Requests ... 10
  F$Fork ... 11
  F$Send ... 57
  Fork ... 10
  Link ... 16
  SetStat SS.SIG ... 97
Shared modules ... 16
Shell ... 28
SHELL Commands ... 48
SHELL Push ... 48
Smoke Signal Broadcasting ... 47, 81, 101
Sort ... 47, 60
Sound Generation ... 81
Spelling Checker ... 96
Split Screen ... 49
Spread Sheet ... 135
SS.SIG Setstat ... 97
Stable Storage ... 113
Standard Error ... 73
Standard Error Path ... 73
Standard I/O Paths ... 73
Standard Input ... 73
Standard Output ... 73
Standards ... 39, 101
Startup File ... 56, 73
Storage Requirement ... 28
Suspend State ... 57
  refid=Proces.Suspend State ... 57
SWTPc ... 107

Tano Dragon ... 47
Televideo ... 97
Terminal Commands ... 40

Terminal Handling ... 123
Terminal Support ... 39
Time Sharing ... 56
Tone Generation ... 89
Translation Lower to Upper Case ... 28
Tuning ... 26

Uniq ... 60
UNIX ... 36
USE ... 27
User Number ... 56
User Seminar ... 47
Users Group ... 29, 47, 62

Vcia ... 51
Version Number ... 28
VTerm ... 49

Waveform ... 82

XOn/XOff ... 9