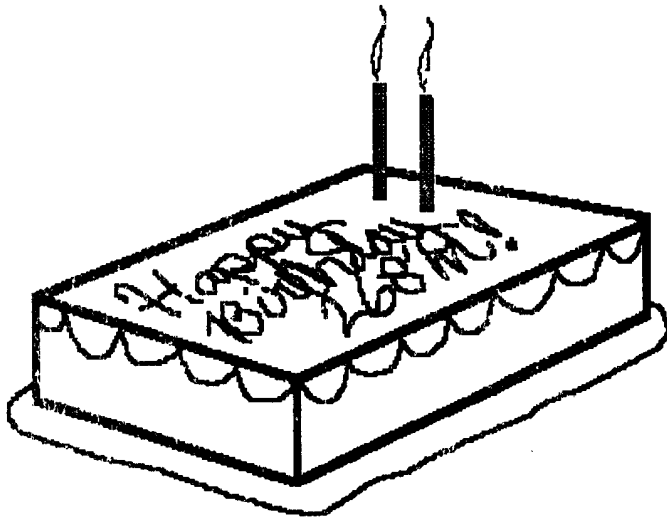


the world of 68' micros

Supporting Tandy Color Computer Disk BASIC, CoCo OS-9, and OS-9/68000

The end of our second year!



The beginning of our third!

CONTENTS

<i>The Editor Speaks</i>	2
<i>Letters to the Editor</i>	3
<i>Basic09 Subroutines</i>	4
(article) Michael Graffam	
<i>Hilights from the Past</i>	5
(article) Robert Gault	
<i>Industrial OS-9 User...</i>	10
(column) F.G. Swygert	
<i>The Hardware Hacker</i>	10
(column) Dr. Marty Goodman	
<i>Operating System-Nine</i>	11
(column) Rick Ulland	
<i>Programming the System IV</i>	13
(article) David Wordell	
<i>CoCo Laser Show</i>	14
(article) Steve Noskovicz	
<i>Basic09 in Easy Steps</i>	20
(column) Chris Dekker	
<i>micronotes</i>	22
<i>Advertiser's Index</i>	25

NOTE: "microdisk" will now be a quarterly publication. The last disk will be sent out soon (last one for volume 2).

POSTMASTER:

If undeliverable return to:
FARNA Systems PB
Box 521
Warner Robins, GA 31099

the world of 68' micros

Published by:
FARNA Systems

P.O. Box 321
Warner Robins, GA 31099-0321

Editor: F. G. Swygert

Subscriptions:

\$25/year (8 issues) US; \$32/year for Canada/Mexico (\$13 US, \$17 C/M for six months-four issues). Overseas \$45/year (\$23 for four issues) AIR; \$37/year, \$18 six months for surface mail.

microdisk: \$40 per year, \$21 six months, or \$6 per issue. Overseas add \$10/year, \$5/six months, \$1/single issue for air mail delivery. Contains programs and source listings from magazine; not stand-alone.

Advertising Rates:

\$15 1/6 page, \$20 1/4 page, \$35 1/2 page, \$60 full page, copy ready. Add \$10 for special placement, \$10 for typesetting (\$5 1/4 or less). Dot matrix will be typeset if deemed unacceptable and submitter billed. 10% discount for four or more appearances.

All trademarks/names property of their respective owners.

Any and all contributions welcomed. Submission constitutes warranty on part of the author that the work is original and not copywritten by another party. All opinions expressed herein are those of the individual writers, not necessarily the publisher or editor. FARNA Systems reserves the right to edit or reject any submitted material without explanation. Renumeration discussed on an individual basis.

Back issues are \$4 per copy. Overseas add \$1 each for surface, \$2.50 airmail delivery.

Newsstand/bulk orders available. Dealers should contact the publisher for details.

Problems with delivery, change of address, subscriptions, or advertisers should be sent to the publisher with a short description.

The publisher is available for comment via e-mail at dsrtfox@Delphi.com. The Delphi CoCo and OS-9 SIGs on Delphi are also frequented (The Delphi SIGs are still sponsored by Falsoft).

ENTIRE CONTENTS COPYRIGHT
1995, FARNA Systems

The editor speaks...

F.G. Swygert

I must start out by apologizing for the many spelling errors in the last issues editorial. I was rushed getting the magazine out (always am after a fest!), and forgot to run the spell checker on it and several other articles. I'm not usually that bad with spelling anyway, but the rush writing job makes for a few missed keys.

There is a follow-up on the Rambler. About 2500 miles after the wreck in Chicago, the upper front suspension joint failed. Apparently, it had been cracked during the wreck. This was impossible to determine by visual inspection. I had noticed a "popping" sound when I applied brakes hard, but suspected a tie-rod end (which I was preparing to replace). One afternoon as I was turning the car around in the driveway to go somewhere, the joint gave way and the left front wheel collapsed.

Now, if you know anything about front suspensions, you will realize that a car can't be steered with one collapsed front wheel! If this had happened while I was driving, especially on a curve, I may not be here writing this for you today. And I drove the car pretty hard on a couple of occasions after the wreck.

Since the collapse occurred in the front yard, the repair was relatively easy. I rebuilt the front suspension five or six years ago, so knew exactly what to do. On the road it would have been next to impossible, even if the car and I did survive.

I don't believe anything really happens without reason, especially something as "lucky" as this. This was a small miracle (I really do drive the car a little hard a times!). If you're a believer, thank God for me again!

Well, here we are! Still going after two years. I admit that at times it is difficult to put a good magazine together. But I have tried. And many of you have made very good

contributions. I was going through my files just a few days ago. I will be using some more of that material soon! So if you sent something in and think I forgot about you, well, you're probably right. Some articles were filed and forgotten in the rush to put each issue together. I'll attempt to correct that over the next few issues.

If there are still things you dislike about this magazine, please let me know. I need the criticism in order to make improvements. And if you have something that may be of interest to others, please send that in also, no matter how trivial it may appear to be. There are some beginners out there still, and people who are new to the CoCo and OS-9 or OSK. Just because it was mentioned in one of the old magazines doesn't mean it shouldn't be repeated!

One important item for future issues: I go bi-monthly beginning with this issue. The reason is just the time and effort to mail each issue. I won't reduce content any. So you should receive the same amount of information as you receive now, just bi-monthly instead of roughly every six weeks. Maybe I can do better on getting them out in time this way! Please let me know what you think about this... will I loose any subscribers over it? I'll try to make future issues a few pages more also.

And I apologize that this issue is late, and that I have indeed missed an issue! There are indeed only seven issues for volume two instead of eight. I'm sorry, but the last fest, the car trouble, a divorce, and move (because of!) just prevented the eighth issue this time around.

Well, that about covers everything for another issue. Please enjoy your summer and have safe vacations!

Letters to the Editor

Almost forgot to renew my subscription! I can't afford to miss an issue.

In a past issue you chided a few drop-outs for not expressing their needs, desires, or wants as far as articles were concerned, so here are mine:

1. A series on OS-9 assembly language programming.
2. One or two articles on using/interfacing Multi-Vue.
3. An article or two on setting up and using KBCOM.

If you can find interested writers, it would be most rewarding for at least this one subscriber.

Jim Kirby
32658 Meadowbrook
Livonia, MT 48154

Well readers, are there any takers on these articles? Maybe Rick will cover MV better in some of his later articles. The OS-9 assembly language idea is really good, hope one of you will take the bait and write something on it!

This is a good time to mention that you don't have to be an accomplished writer to submit a good article. I'll assist with editing, you just write down your ideas and thoughts on a subject so I'll have something to work with!!

Help! I've been unable to get the CoCo 3 emulator to work on my 486DX2/66 with 8MB or RAM. The green screen comes up scrambled! According to the VGA compatibility test it should work fine. Works great on my wife's 486DLC/40. Right now I'm building a 486DX/50 as a dedicated CoCo 2/3 if it runs the

emulators. Also, is there a way to connect the Commodore 1084 or 1802 monitor to the CoCo 3's RGB port? I have both and would like to try.

Enclosed is a check for my renewal. You put out a first rate magazine and make up a big part of the CoCo community.

Ron Shively
820 Garden Street
Beatrice, NE 68341

Ron, several people have had problems with the CC3 emulator. The problem is in the video card. Jeff Vavasour has apparently corrected the problem. Please write him and ask for a 'fixed' disk!

The 1084 Commodore monitor can be used directly with the CoCo 3 by using the RGBA port. Wire it wire to wire to the CoCo port. You just need the pin-outs for the RGBA port, which is in the owners manual. I'm unfamiliar with the 1802, but should be the same.

I hate to admit it, but I went out and bought a new computer, a Packard Bell Pentium model. I don't have room to keep two computers set up, so was excited to see the cover article about the emulator. However, I do have some questions:

1. My new computer only has a 3.5" drive. Do I need a 5.25" also?
2. I see that I can run DECB and OS-9. Does the emulator keep track of the different file formats for the two systems?
3. I guess I need more information on the CoCo 2 emulator. Is this a prerequisite to the CoCo 3 emulator?

Jay Duke

11642 Pines Trail
Roscommon, MI 48653-9703

Jay, you will need to get a 5.25" drive for your computer. This shouldn't be hard to do if you have a 5.25" bay free. The emulator creates virtual disks on the hard drive for DECB and OS-9, and keeps the two separate easily. And yes, you should get the CoCo 2 emulator and play with it first, but you can write Jeff and get the CoCo 3 emulator first.

Hi, Frank. I was just reading the May issue, and noticed your reply to the first letter to the editor, in which you assert that, under Extended ADOS-3, an 80-track drive will be limited to 78 file entries. You should be able to have 128 file entries, since sectors 3 through 18 on Track 17 (16 sectors) are available to accommodate 8 file entries each.

Art Flexser
(artflexser@delphi.com)

Thanks for the correction Art! I was actually thinking that the 80 track disks followed the same format as the 35/40 track schemes. But since Tandy never had a standard for an 80 track drive, I guess that wouldn't be a limitation!

Writing subroutines for Basic09.

This article is meant to be a general overview of writing external subroutines for Basic09. I will cover a few different routines. The routines I will cover are string handling routines. I have uploaded many subroutines to Delphi's OS-9 Sig (ProgrammersDen, search for Basic09, Subroutines), and have written a few especially for this article.

I have chosen to do string handling for 2 reasons, first off, compared to other languages Basic09 (and just about every other Basic variant) has, in my opinion pathetic string handling ability, and second, they are easier to illustrate to novice programmers, and therefore make for better tutorials. With all that said, lets get to the fun part!

The following examples (Lstr, Rstr, Midstr) will replace a portion of a string (much the same way that LEFT\$, RIGHT\$ and MID\$ isolate portions of strings). The syntax for Lstr and Rstr are the same, you run the routine with the original string and the new string as parameters to the program. RUN LStr("Hello", "J") would make "Hello" become "Jello". RStr would replace at the right side of the string. The code for LStr and RStr is as follows:

```
PROCEDURE LStr
0000 PARAM orgstr:STRING[200]
000C PARAM newstr:STRING[200]
0018 DIM buf1:STRING[200]
0024
0025 buf1=RIGHT$(orgstr,LEN (orgstr)-
LEN(newstr))
0037 orgstr=newstr+buf1
0043 END
```

```
PROCEDURE RStr
0000 PARAM orgstr:STRING[200]
000C PARAM newstr:STRING[200]
0018
0019 DIM buf1:STRING[200]
0025 buf1=LEFT$(orgstr,LEN (orgstr)-
LEN(newstr))
0037 orgstr=buf1+newstr
0043 END
```

Ok, all we have left to do is make up a Midstr. The syntax for Midstr will of course have to be changed, and we will

have to include where in string to replace. I have made two versions of Midstr and Mstr. Midstr's syntax is like this: "RUN Midstr(old\$, start_pos, end_pos, newstr\$)". This allows you to specify the starting and ending positions of the string to replace. RUN Midstr("My name is mike", 3, 8, "") will create the string "My is mike", RUN MStr("My name is mike", 3, "aaaa") will create the string "Myaaaaame is mike". MStr is different in that it uses the length of newstr\$ to determine the ending position. MStr is used as follows: "RUN MStr(old\$, start_pos, newstr\$)".

```
PROCEDURE MidStr
0000 PARAM orgstr:STRING[200]
000C PARAM strpos:INTEGER
0013 PARAM endpos:INTEGER
001A PARAM newstr:STRING[200]
0026 DIM buf1,buf2:STRING[200]
0036
0037 buf1=MID$(orgstr,1,strpos-1)
0048 buf2=MID$(orgstr,endpos+1,
LEN(orgstr))
005B orgstr=buf1+newstr+buf2
```

```
PROCEDURE Mstr
0000 PARAM orgstr:STRING[200]
000C PARAM strpos:INTEGER
0013 DIM endpos:INTEGER
001A PARAM newstr:STRING[200]
0026 DIM buf1,buf2:STRING[200]
0036 endpos=LEN(newstr)
003F
0040 buf1=MID$(orgstr,1,strpos-1)
0051 buf2=MID$(orgstr,endpos+1,
LEN(orgstr))
0064 orgstr=buf1+newstr+buf2
```

The next set of routines is string removal functions. These include Lrem Rrem and Mrem. These routines will remove portions of strings defined by position and length. The syntax for Lrem and Rrem is as follows: RUN LRem("Michael", 1) would make the string "ichael". Using Rrem would make "Michae".

```
PROCEDURE LRem
```

```
0000 PARAM instr:STRING[200]
000C PARAM num:INTEGER
0013
0014 DIM i:INTEGER
001B DIM temp:STRING[200]
0027
0028 temp=MID$(instr,num+1,LEN
(instr))
003B instr=temp

PROCEDURE RRem
0000 PARAM instr:STRING[200]
000C PARAM num:INTEGER
0013
0014 DIM i:INTEGER
001B DIM temp:STRING[200]
0027
0028 temp=LEFT$(instr,LEN(instr)-
num)
0039 instr=temp
0041 END
```

MRem will remove from the middle of a string its syntax is: RUN MRem (org\$, start, end). org\$ is the original string to be processed, start is the starting position in the string, and end is the ending position.

```
PROCEDURE MRem
0000 PARAM instr:STRING[200]
000C PARAM startpos,endpos:
INTEGER
0017 DIM temp:STRING[200]
0023 DIM temp1:STRING[200]
002F temp=LEFT$(instr,startpos-1)
003E temp1=RIGHT$(instr,LEN (instr)-
endpos)
004F instr=temp+temp1
```

Another routine is a run-length encoding system, this encoder can be used on any string that does not include numbers. It will take a string say, "AAAAAAAABBBBBBBBCCCCDDDDDEF" and make it "7A9B4C4DEF". It is important to note that E and F are stored as-is because including a number would make the string bigger, runs of 2 or more are stored (runs of 2 won't be compressed any, but it makes the routine simpler, so I did it that way). TCom is a program to call encode and decode a file. It will work well on any string with long "runs" of characters, a string like "Hello my name

is Mike" wouldn't get crunched at all, this encoder is meant primarily for "bit-mapped" image data, where there will likely be long runs of the same character.

```

PROCEDURE TCom
0000 (* This program uses a subroutine
to compress text files *)
003B DIM switch:STRING[1]
0047 DIM infile,outfile:STRING[80]
0057 DIM in,out:BYTE
0062 DIM line:STRING[200]
006E DIM a,b,c,i:INTEGER
0081 PRINT "C- Compress File"
0095 PRINT "E- Expand File"
00A7 INPUT ">?",switch
00B1
00B2 INPUT "Input File Name? ",infile
00CB INPUT "Output File Name?
",outfile
00E5
00E6 OPEN #in,infile:READ
00F2 CREATE #out,outfile:WRITE
00FE
00FF WHILE EOF(#in)=FALSE DO
010B READ #in,line
0115 RUN encode(switch,line)
0124 WRITE #out,line
012E ENDWHILE
0132
0133 CLOSE #in
0139 CLOSE #out

```

```

PROCEDURE encode
0000 (* This program is a run-length
encoder *)
003A PARAM switch:STRING[1]
0046 PARAM line:STRING[200]
0052 DIM comline:STRING[200]
005E comline=""
0065 DIM strpos:INTEGER
006C strpos=1
0073 DIM a,b,c,i:INTEGER
0086 DIM table:STRING[10]
0092 table="0123456789"
00A3 DIM char:STRING[1]
00AF
00B0 IF switch="C" OR switch="c"
THEN
00C5 c=1
00CC FOR strpos=1 TO LEN(line)
00DE char=MID$(line,strpos,1)
00EC IF SUBSTR(char,table)>0 THEN
00FC PRINT #3," Run-Length Encoding
Error -"
0120 PRINT #3,"Numerals not allowed
in input file."
014C PRINT #3,"Unconditional Abort."
0169 PRINT #3,"Error #001"
017B END
017D ENDIF
017F IF char=MID$(line,strpos+1,1)

```

```

THEN
0195 c=c+1
01A0 ELSE
01A4 IF c=1 THEN
01B0 comline=comline+char
01BC c=1
01C3 ELSE
01C7 comline=comline+STR$(c)+char
01D8 c=1
01DF ENDIF
01E1 ENDIF
01E3 NEXT strpos
01EE line=comline
01F6 END
01F8 ENDIF
01FA
01FB IF switch="E" OR switch="e"
THEN
0210 num$=""
0218 table="0123456789"
0229 FOR strpos=1 TO LEN(line)
023B char=MID$(line,strpos,1)
0249 IF SUBSTR(char,table)>0 THEN
0259 num$=num$+char
0265 ELSE
0269 IF num$<>"0" THEN
0276 FOR i=1 TO VAL(num$)
0289 comline=comline+char
0295 NEXT i
02A0 num$=""
02A8 ELSE
02AC comline=comline+char
02B8 ENDIF
02BA ENDIF
02BC NEXT strpos
02C7 line=comline
02CF END
02D1 ENDIF
02D3
02D4 PRINT #3,"Run-Length Encoding
Error- "
02F7 PRINT #3," run
encode({option},string)"
0321 PRINT #3," Where {option} is either
c (condense) e (expand)"
035A PRINT #3," and string is the string
to process."
0387 PRINT #3," Unconditional Abort."
03A4 PRINT #3,"Error #001"
03B6 END

```

The following text can expanded by Tcom/encode to produce image data that could be plotted to the screen to form bitmapped graphics. A friend and I did this over a modem, running encode on the strings compressed them down and made it alot quicker than sending each character over one at a time.

51.

```

28.14*9.
26.18*7.
23.24*4.
22.26*3.
20.30*.
19.7*18.7*
19.5*22.5*
19.3*26.3*
19.3*26.3*
19.3*26.3*
19.3*26.3*
20.4*23.3*.
22.3*20.3*3.
.50*
.50*
.50*
.50*
.50*
.50*
.2*46.2*
51.
(The above is a "q" sitting sideways)

```

The final set of routines are Rev, Toupper, Tolower, and Caps. Rev will reverse a string, its syntax is RUN Rev(string\$) ("Hello" will become "olleH", and "bob" becomes "bob", I havent found out what causes that bug yet).

Toupper and Tolower will convert a string to all upper or all lower case. Caps will capitalize every word in a string.

```

PROCEDURE Rev
0000 PARAM str:STRING[100]
000C DIM temp$:STRING[100]
0018 DIM i:INTEGER
001F
0020 temp$=""
0027 FOR i=LEN(str) TO 1 STEP -1
003F temp$=temp$+MID$(str,i,1)
0051 NEXT i
005C
005D str=temp$

```

```

PROCEDURE Toupper
0000 PARAM instr:STRING[200]
000C DIM strpos:INTEGER
0013 DIM temp$:STRING[200]
001F
0020 (* convert term variable to uppercase
*)
0048 temp$=""
004F strpos=1
0056
0057 REPEAT
0059 IF ASC(MID$(instr,strpos,1))>96
AND ASC(MID$(instr,strpos,1))<123
THEN

```

```

007A temp$=temp$+CHR$(ASC(MID$(instr,strpos,1))-32)
0091 ELSE
0095 temp$=temp$+MID$(instr, strpos,1)
00A7 ENDIF
00A9 strpos=strpos+1
00B4 UNTIL strpos>LEN(instr)
00C1
00C2 instr=temp$
00CA END
00CC

```

```

PROCEDURE Toupper
0000 PARAM instr:STRING[200]
000C DIM strpos:INTEGER
0013 DIM temp$:STRING[200]
001F
0020 (* convert term variable to uppercase *)
0048 temp$=""
004F strpos=1
0056
0057 REPEAT
0059 IF ASC(MID$(instr,strpos,1))<91
AND ASC(MID$(instr,strpos,1))>64
THEN
007A temp$=temp$+CHR$(ASC(MID$(instr,strpos,1))+32)
0091 ELSE
0095 temp$=temp$+MID$(instr,strpos,1)
00A7 ENDIF
00A9 strpos=strpos+1
00B4 UNTIL strpos>LEN(instr)
00C1
00C2 instr=temp$
00CA END
00CC

```

```

PROCEDURE Caps
0000 PARAM instr:STRING[200]
000C DIM strpos:INTEGER
0013 DIM temp$:STRING[200]
001F
0020 strpos=0
0027 temp=""
002E REPEAT
0030 strpos=strpos+1
003B IF MID$(instr,strpos,1) <> "" THEN
004E r$=MID$(instr,strpos,1)
005C RUN toupper(r$)
0066 temp=temp+r$
0072 REPEAT
0074 strpos=strpos+1
007F temp=temp+MID$(instr,strpos,1)
0091 UNTIL MID$(instr,strpos,1)=""
" OR strpos>LEN(instr)
00AC ENDIF
00AE UNTIL strpos>LEN(instr)
00BB instr=temp
00C3 END

```

At this point I will mention that while

all these routines can be ran like they are now, it is usually a good idea to modify them and retro-fit them to work in a "GOSUB/RETURN" situation. This eliminates cluttering up the module directory, and if done right, in my opinion makes for "cleaner" code overall. I prefer having all my subroutines internal to the program, except of course when the program is getting big and the source won't fit into my favorite editor's buffer. At that point splitting up the source is a good idea just to keep it managable.

I have written many other subroutines, the ones listed here are all very simple. Some of my other ones, like Strpro (a string processor, it will take a string, and up to a 100 subscript array of data and process the string according to "rules" outlined in the data array, it is much like a small batch language) and Fuzzy (a fuzzy logic routine for Basic09 (which is still in the processes of being written)) are much more complicated. All my subroutines are available on disk for free. All I want is the money for the disk and shipping charges (\$4). If you send me money for the disk with the subroutines on it, please include a note stating what the money is for, and the disk format you would like it on (180k/360k 5.25's or 720k 3.5). I can also email them to you if you are on Delphi (or if you supply an Internet address).

USMail:
Michael Graffam
25 Pine Echo Drive
Poughkeepsie NY 12601

E-mail:
Illusionist@Delphi.com
Or Michael Graffam over FIDO
(OS-9 Echo).

My home phone number is
(914)-471-7438

Programming the System IV

(continued from page 13)

```

This is key.c *****

#include <stdio.h>
#include <keybd.h>

main()
{
int keybd[2]; /* array to be passed
back from key() function */
while(1) {
key(keybd); /* Go and get the key */

switch(keybd[1]) {
case 'A': printf("You hit Up Arrow\n");
break;
case 'B': printf("You hit Dn Arrow\n");
break;
case 'C': printf("You hit Rt Arrow\n");
break;
case 'D': printf("You hit Lt Arrow\n");
break;
default : printf("You hit an invalid
key\n");
exit(0);
}
}
}

```

```

This is keybd.h *****

key(keybd)
int keybd[]; /* Tell function it's
getting array */

{
char *str1 = "tmode noecho";
char *str2 = "tmode echo";
char inchar;

system(str1); /* shut off echo */
keybd[1] = 0; /* clear keybd[] so
prior keys not passed
back again */
inchar = '\x1B'; /* initialize - read
can't see escape */
read(0,&inchar,1); /* read from
keyboard and store */
if (inchar == '\x1B') { /* read failed
to change inchar */
read(0,&inchar,1); /* escape found
- read second char */
if (inchar == '\x5B') {
read(0,&inchar,1); /* second
char was [ - parse key */
keybd[1] = inchar; /* array being
passed back */
}
}
}
system(str2); /* turn echo back on */
}

```

Hi-lights from the Past, Part 1

Robert Gault

Support programs MLFINDER.BIN and MLCODE.BAS

There have been many excellent Cocomagazines which have disappeared into the mists of the past. They have left a legacy of marvelous programs and informative articles for the Cocomagazine. Unfortunately these magazines are long out of print and the authors have long left the Cocomagazine scene.

While many of us fondly remember these articles, there are new members in the Cocomagazine community who (having received their computers as gifts, inheritances, or purchases at computer swaps) don't have the benefit of these magazine gems.

Starting with this article, I will attempt to present the essence from some of the better of these articles. This will be done only if it is possible to present significantly original code and text. I do respect copyright laws.

You will need to use both Basic and machine language for this series so I recommend that you acquire an Editor/Assembler. For those who can't find one, I'll start with a diversion. What follows is a program MLCODE.BAS which scans memory from the start of an ml routine to the end and converts the code into a Basic loader/driver. The output is in ASCII so it can be easily merged into other Basic programs as you will see. Notice that the data is in hexadecimal format (takes less space than decimal), is limited to ten items per Basic code line (for ease of reading and typing), and includes error checking simple minded though it is.

To use MLCODE.BAS you need the starting, ending, and execution address of the routine to be converted. MLFINDER will give you that information and more. This program also indicates the location on disk of the code segments which is great for hackers.

I will, as needed, present both assembly source code and Basic drivers produced by MLCODE.BAS for future parts to this series.

```
MLCODE.BAS
10 PMODE0:PCLEAR1
20 CLS:PRINT"SELECT OPTION:"
30 PRINT"1) ROUTINE IS IN MEMORY"
40 PRINT"2) PROGRAM IS ON DISK":PRINT
50 INPUT" 1 OR 2";S:IFS<1 OR S>2 THENRUN
60 PRINT:IFS=1THEN110
70 CLS:INPUT"SOURCE DRIVE NUMBER =";D0:DRIVE D0
80 PRINT:PRINT"  FILE <MUST> BE /BIN":PRINT
90 PRINT"DO NOT USE /EXT:DR":INPUT"WHAT IS THE FILE NAME
```

```
";FILES
100 LOADM FILES
110 INPUT"WHAT IS THE START ADDRESS IN HEX$ "; START$: START=VAL("&H"+START$)
120 INPUT"WHAT IS THE END ADDRESS IN HEX$ "; EN$: EN=VAL("&H"+EN$)
130 INPUT"WHAT IS THE EXECUTION ADDRESS IN HEX$ ";EX$
140 INPUT"PRINT TO SCREEN OR DISK <0 OR 1> ";BUF
150 IFBU=1 THEN PRINT: INPUT"DESTINATION DRIVE =";D1:DRIVE D1
160 LI=10
170 IF BU=0 THEN190
180 OPEN"O",#BU,FILES+"/MLC"
190 PRINT#BU,LI,"REM ";FILES: LI=LI+10
200 PRINT#BU,LI;"LI=80":LI=LI+10
210 PRINT#BU,LI;"FOR M=&H"; START$;" TO &H";EN$;" STEP10: SUM=0":LI=LI+10
220 PRINT#BU,LI;"FOR I=0TO9: READAS: VA=VAL("&H"; CHR$(34); CHR$(34);"+AS):SUM=SUM+VA: POKE M+I,VA: NEXT: READ CHK: IF SUM<>CHK THEN PRINT"; CHR$(34); "ERROR IN LINE"; CHR$(34);"LI: END"
230 LI=LI+10: PRINT#BU,LI; "LI=LI+10: NEXT"
240 LI=LI+10
250 PRINT#BU,LI;"SAVEM "; CHR$(34); FILES; CHR$(34);"&H"; START$; "&H"; EN$;"&H"; EX$: LI=LI+10
260 PRINT#BU,LI;"END":LI=LI+10
270 FOR M=START TO EN STEP 10
280 PRINT#BU, LI;"DATA ";LI=LI+10
290 SUM=0:FORI=0TO9
300 V=PEEK(M+I):SUM=SUM+V:PRINT #BU,USING"%%"; HEX$(V): IF M+I =EN THEN GOTO330 ELSE IF I<>9 THEN PRINT#BU," ";NEXT ELSE NEXT
310 PRINT#BU,"";SUM:NEXT
320 CLOSE #BU:END
330 FORJ=1+1 TO9:PRINT#BU,"00";:NEXTJ:PRINT#BU,"";SUM:CLOSE #BU
00100 TITLE MLF - MLFINDER
00110 * MLFinder by Robert Gault
00120 * Usage: Find Load and Execution addresses of ml programs stored on
00130 * disk. Report the track and sector locations where code is
00140 * stored. Report to printer or screen.
00150 * For syntax see HELP message below.
00160
00170 * GENERAL METHOD:
00180 * GET FILE NAME, OPEN FILE SEQUENTIAL FOR INPUT THEN
00190 * READ LENGTH OF SECTION, GET START ADDRESS, ADD LENGTH TO START
00200 * READ CHARACTERS FOR LENGTH, CHECK FOR END OR NEXT SECTION.
```

```
00210
00220 CR EQU $0D CARRIAGE RETURN
00230 SPACE EQU $20
00240
00250 DEVNUM EQU $6F 0=SCREEN, 1-N=DISK, -1=CASS., -2=PRINTER
00260 BUFLG EQU $70 BUFFER IN FLAG
0=DATA -1=EMPTY
00270 ZERO EQU $8A ALWAYS=$0000
00280 GETNCH EQU $9F GET NEXT CHARACTER
00290 GETCCH EQU $A5 GET CURRENT CHARACTER
00300 TRACK EQU $EC DSKCON VARIABLES
00310 SECTOR EQU $ED " "
00320 DFLTY EQU $957 DISK FILE TYPE 0=BAS, 1=DATA, 2=ML, 3=TXT
00330 CLOSEF EQU $A42D CLOSE FILE
00340 ERROR EQU $AC46 BASICERROR ROUTINE
00350 BASIC EQU $AC73 PRINT CR, OK
00360 COMMA EQU $B26D TEST FOR COMMA
00370 GETNM EQU $B73D EVALUATE POS NUMBER RETURN IN REG.X
00380 PRINTS EQU $B99C PRINT STRING TO SCREEN
00390 *****
*****
*****
00400 * MACROS - DEFINED FOR CLARITY IN SUBSEQUENT CODE
00410 *****
*****
*****
00420 PUTCHR MACRO
00430 JSR [$A002] OUTPUT CHARACTER VIA $6F DEVICE
00440 ENDM
00450
00460 SETDEV MACRO
00470 LDA MODE POINT DEVICE TO PRINTER OR SCREEN
00480 STA DEVNUM
00490 ENDM
00500
00510 RSETDV MACRO
00520 LDA DIMAGE RESET DEVICE TO DISK FILE
00530 STA DEVNUM
00540 ENDM
00550
00560 TSTPTR MACRO
00570 LDA $FF22 TEST FOR PRINTER READY
00580 BITA #1 RS232 INPUT LINE
00590 BEQ \A
00600 LEAX <OFFLIN-1,PCR
00610 JMP PRINTS
00620 OFFLIN FCC /Printer is off line!/
00630 FCB 0
00640 \A EQU *
00650 ENDM
00660
```

```

00670 *****
*****
00680
00690 ORG SCB EXTENDED BASIC
SCRATCH PAD
00700 LOFFST RMB 2 LOAD OFFSET
00710 LENGTH RMB 2 LENGTH OF
LOAD BLOCK
00720 LDADDR RMB 2 LOAD ADDRESS
00730 MODE RMB 1 OUTPUT PATH:
0=SCREEN, FE=PRINTER
00740 LOCAT RMB 2 !
    POINTER TO DISK SECTOR
00750 TRSEC RMB 2 TRACK, SECTOR
00760 DOS RMB 1 DOS INDICATOR:
0=1.0 1=1.1
00770 DIMAGE RMB 1 IMAGE OF $6F
00780
00790 *****
*****
00800
00810 ORG $E00
00820 START CLR MODE RESET LINE
PRINTER FLAG
00830 CLR DOS
00840 A@ JSR GETNCH
00850 LBEQ HELP
00860 CMPA #'H HELP
00870 LBEQ HELP
00880 CMPA #'P FOR LINE PRINTER
00890 BNE B@
00900 TSTPTR
00910 LDA #-2
00920 STA MODE INDICATE LINE
PRINTER
00930 BRA A@
00940 B@ LDX $C004
00950 CMPX #$D66C DOS1.0
00960 BEQ C@
00970 INC DOS
00980 CMPX #$D75F DOS1.1
00990 BEQ C@
01000 LDB #20 I/O ERROR; CLOSEST
TO UNKNOWN DOS
01010 JMP ERROR
01020 C@ EQU *
01030
01040 TST DOS
01050 BNE A@
01060 JSR $CEDF DEFAULT .BIN &
GET FILE NAME
01070 JSR $C959 OPEN SEQUENTIAL
FILE FOR INPUT
01080 BRA B@
01090 A@ JSR $CFBB
01100 JSR $CA07
01110 B@ EQU *
01120
01130 LDA DEVNUM GET PATH #
01140 STA DIMAGE SAVE IT
01150 LDD DFLTYP DISK FILE TYPE
CODE
01160 SUBD #$200 ML+BIN
01170 LBNE $A616 BAD FILE MODE
ERROR CALL
01180 STD LOFFST REG.D=0; CLEAR
LOAD OFFSET
01190 STD LOCAT CLEAR DISK
SECTOR LOCATION
01200 * NOW LOOK FOR AN OFFSET
LOAD AS IN "LOADM FILE,&H2000"
01210 JSR GETCCH MORE PARAMETER?
01220 BEQ A@
01230 JSR COMMA TEST FOR COMMA
01240 JSR GETNM GET NUMBER;
MAKE INTEGER IN REG.X
01250 STX LOFFST UPDATE OFFSET
LOAD VALUE
01260 A@ JSR $A5C7 GETCCH WITH
ERROR CHECK; NO MORE ENTRIES
01270
01280 MAINLP LBSR GET1BF GET
CHARACTER FROM BUFFER; BLOCK
CODE
01290 PSHS A SAVE IT; 0=NEXT
SEGMENT, FF=END
01300 LBSR GET2BF GET 2
CHARACTERS FROM BUFFER
01310 STD LENGTH BLOCK LENGTH
FOR LOADING
01320 LDX TRACK READ DISCON
01330 STX TRSEC SAVE DATA FOR
REPORT
01340 TFR D,X SAVE BLOCK LENGTH
01350 LBSR GET2BF 2 MORE
CHARACTERS; LOAD POINTER
01360 ADDD LOFFST ADD START
OFFSET TO POINTER
01370 STD LDADDR LOAD ADDRESS
01380 LDA ,S+ RETRIEVE BLOCK CODE
AND SET Z FLAG
01390 BNE CLOSE CLOSE FILE
01400
01410 A@ TST DOS SKIP THROUGH
ONE CODE BLOCK
01420 BNE B@
01430 JSR $C597 GET BYTE FROM FILE
01440 LDB BUFLG
01450 LBNE $C334 INPUT PAST END
OF FILE
01460 BRA C@
01470 B@ JSR $C5C4
01480 LDB BUFLG
01490 LBNE $C352 EOF
01500 C@ LEAX -1,X
01510 BNE A@ END OF SECTION YET
?
01520
01530 LEAX LOAD,PCR POINT TO
MESSAGE
01540 SETDEV POINT OUTPUT TO
CORRECT DEVICE
01550 LDD LDADDR GET VALUE
01560 BSR PRINT
01570 LEAX END,PCR POINT TO
MESSAGE
01580 LDD LDADDR LOAD ADDRESS
01590 ADDD LENGTH
01600 SUBD #1
01610 BSR PRINT
01620 LEAX TRKMSG,PCR POINT TO
MESSAGE
01630 CLRA
01640 LDB TRSEC GET TRACK VALUE
01650 BSR PRINT
01660 LEAX SECTR,PCR POINT TO
MESSAGE
01670 CLRA
01680 LDB TRSEC+1 GET SECTOR
VALUE
01690 BSR PRINT
01700 LEAX LOC,PCR POINT TO
MESSAGE
01710 LDD LOCAT GET POINTER TO
SECTOR
01720 ADDD #5 ACCOUNT FOR
SUBHEADER
01730 PSHS D SAVE TRUE LOCATION
IN SECTOR

```

NOW AVAILABLE!

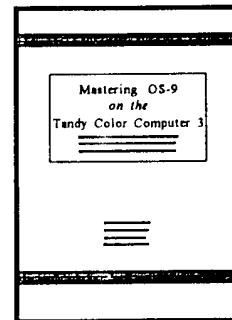
The long awaited update of Paul Ward's "Start OS-9"...

"Mastering OS-9"

Edited and revised by Francis G. Swygert

This new edition contains revised, easier to follow tutorials, an index, revised information articles, several NEW informative articles (including Rick Ulland on making an OS-9 boot disk), and several extra utilities on disk! 262 softbound pages (5.5"x8.5").

Price for book and disk is only \$35 post paid (US)
(Canada add \$2 for s&h, overseas add \$10 for airmail)



FARNA Systems

Box 321

Warner Robins, GA 31099-0321

912-328-7859


```

01740 BSR PRINT
01750 RSETDV RESET DEVICE TO
CORRECT DISK PATH
01760 PULS D RECOVER LOCATION
01770 ADDD LENGTH ADD BLOCK
LENGTH
01780 STD LOCAT SAVE NEW VALUE
01790 BRA MAINLP
01800
01810 CLOSE JSR CLOSEF
01820 SETDEV
01830 LEAX <XFER,PCR
01840 LDD LDADDR
01850 BSR PRINT
01860 LDA #CR FLUSH PRINTER
01870 BRA FINI
01880
01890 PRINT PSHS D PRINT TO SCREEN
AS INDEXED BY REG. X
01900 A@ LDA ,X+ READ MESSAGE
01910 BEQ C@
01920 PUTCHR SEND REG.A TO
CORRECT PATH
01930 BRA A@
01940 C@ PULS A GET MSB
01950 BSR D@
01960 PULS A GET LSB
01970 D@ TFR A,B
01980 LSRA MSN
01990 LSRA
02000 LSRA
02010 LSRA
02020 BSR E@
02030 TFR B,A
02040 ANDA #SOF LSN
02050 E@ CMPA #9
02060 BLS F@
02070 ADDA #7
02080 F@ ADDA #'0
02090 FINI PUTCHR
02100 RTS
02110
02120 GET2BF BSR A@ GET TWO
CHARACTERS FROM BUFFER
02130 A@ BSR GET1BF STORE RESULT
IN D REG.
02140 EXG A,B
02150 RTS
02160
02170 GET1BF TST DOS
02180 BNE A@
02190 JMP SCCE2 GET CHAR FROM
BUFFER
02200 A@ JMP $CDBC
02210
02220 LOAD FCB CR
02230 FCC /LOAD:/
02240 FCB 0
02250 END FCC /END:/
02260 FCB 0
02270 XFER FCC /XFER:/
02280 FCB 0
02290 TRKMSG FCC /TRACK:/
02300 FCB 0
02310 SECTR FCC /SECTOR:/
02320 FCB 0
02330 LOC FCC /LOCATION:/
02340 FCB 0
02350
02360 HELP LEAX <MESSAGE,PCR
02370 A@ LDA ,X+

```

```

02380 BEQ B@
02390 PUTCHR
02400 BRA A@
02410 B@ JMP BASIC
02420 MESSAGE FCB CR
02430 FCC "USAGE: [ ]=OPTIONAL"
02440 FCB CR
02450 FCC / EXEC:"NAME"[,offset]/
02460 FCC /prints info to screen/
02470 FCB CR
02480 FCC / EXEC:P"NAME"[,offset]/
02490 FCC /sends info to printer/
02500 FCB CR,0
02510 END START

```

```


Output of MLCODE.BAS to create
MLFINDER.BIN
10 REM MLFINDER
20 LI=80
30 FOR M=&HE00 TO &H100D
STEP10:SUM=0
40 FOR I=0TO9:READA$:VA=VAL
("&H"+A$):SUM=SUM+VA:POKE
M+I,VA:NEXT:READ CHK:IFSUM<>CHK
THEN PRINT"ERROR IN LINE"LI:END
50 LI=LI+10:NEXT
60 SAVEM "MLFINDER", &HE00, &H100D,
&HE00
70 END
80 DATA F , D1, F , D6, 9D, 9F, 10, 27, 1 ,
72, 939
90 DATA 81, 48, 10, 27, 1 , 6C, 81, 50, 26,
28, 652
100 DATA B6, FF, 22, 85, 1 , 27, 1B, 30, 8C,
2 , 861
110 DATA 7E, B9, 9C, 50, 72, 69, 6E, 74, 65,
72, 1207
120 DATA 20, 69, 73, 20, 6F, 66, 66, 20, 6C,
69, 844
130 DATA 6E, 65, 21, 0 , 86, FE, 97, D1, 20,
C8, 1224
140 DATA BE, C0, 4 , 8C, D6, 6C,!
27, C , C , D6, 1125
150 DATA 8C, D7, 5F, 27, 5 , C6, 14, 7E, AC,
46, 1080
160 DATA D , D6, 26, 8 , BD, CE, DF, BD, C9,
59, 1370
170 DATA 20, 6 , BD, CF, BB, BD, CA, 7 , 96,
6F, 1280
180 DATA 97, D7, FC, 9 , 57, 83, 2 , 0 , 10,
26, 901
190 DATA 97, A6, DD, CB, DD, D2, 9D, A5,
27, 8 , 1541
200 DATA BD, B2, 6D, BD, B7, 3D, 9F, CB,
BD, A5, 1625
210 DATA C7, 17, 0 , BC, 34, 2 , 17, 0 , B0,
DD, 884
220 DATA CD, 9E, EC, 9F, D4, 1F, 1 , 17, 0
, A5, 1190
230 DATA D3, CB, DD, CF, A6, E0, 26, 60,
D , D6, 1593
240 DATA 26, B , BD, C5, 97, D6, 70, 10, 26,
B4, 1146
250 DATA 89, 20, 9 , BD, C5, C4, D6, 70, 10,
26, 1140
260 DATA B4, 9C, 30, 1F, 26, E4, 30, 8D, 0
, 8E, 1012
270 DATA 96, D1, 97, 6F, DC, CF, 8D, 4A,
30, 8D, 1452
280 DATA 0 , 89, DC, CF, D3, CD, 83, 0 , 1
, 8D, 1253

```

```

290 DATA 3D, 30, 8D, 0 , 89, 4F, D6, D4, 8D,
34, 1085
300 DATA 30, 8D, 0 , 88, 4F, D6, D5, 8D, 2B,
30, 1063
310 DATA 8D, 0 , 88, DC, D2, C3, 0 , 5 , 34,
6 , 965
320 DATA 8D, 1E, 96, D7, 97, 6F, 35, 6 , D3,
CD, 1273
330 DATA DD, D2, 20, 85, BD, A!
4 , 2D, 96, D1, 97, 1504
340 DATA 6F, 30, 8C, 51, DC, CF, 8D, 4 , 86,
D , 1099
350 DATA 20, 26, 34, 6 , A6, 80, 27, 6 , AD,
9F, 799
360 DATA A0, 2 , 20, F6, 35, 2 , 8D, 2 , 35,
2 , 693
370 DATA 1F, 89, 44, 44, 44, 8D, 4 , 1F,
98, 768
380 DATA 84, F , 81, 9 , 23, 2 , 8B, 7 , 8B, 30,
655
390 DATA AD, 9F, A0, 2 , 39, 8D, 0 , 8D, 3
, 1E, 866
400 DATA 89, 39, D , D6, 26, 3 , 7E, CC, E2,
7E, 1144
410 DATA CD, BC, D , 4C, 4F, 41, 44, 3A, 0
, 20, 784
420 DATA 45, 4E, 44, 3A, 0 , 20, 58, 46, 45,
52, 614
430 DATA 3A, 0 , 20, 54, 52, 41, 43, 4B, 3A,
0 , 521
440 DATA 20, 53, 45, 43, 54, 4F, 52, 3A, 0
, 20, 586
450 DATA 4C, 4F, 43, 41, 54, 49, 4F, 4E, 3A,
0 , 659
460 DATA 30, 8C, D , A6, 80, 27, 6 , AD, 9F,
A0, 1032
470 DATA 2 , 20, F6, 7E, AC, 73, D , 55, 53,
41, 939
480 DATA 47, 45, 3A, 20, 20, 20, 5B, 20,
5D, 542
490 DATA 3D, 4F, 50, 54, 49, 4F, 4E, 41, 4C,
D , 688
500 DATA 20, 20, 20, 20, 20, 45, 58, 45,
43, 485
510 DATA 3A, 22, 4E, 41, 4D, 45, 22, 5B, 2C,
6F, 661
520 DATA 66, 66, 73, 65, 74, 5D, 20, 20, 20,
20, 757
530 DATA 70, 72, 69, 6E, 74, 73, 20, 69, 6E,
66, 1021
540 DATA 6F, 20, 74, 6F, 20, 73, 63, 72, 65,
65, 932
550 DATA 6E, D , 20, 20, 20, 20, 20, 45,
58, 472
560 DATA 45, 43, 3A, 50, 22, 4E, 41, 4D, 45,
22, 631
570 DATA 5B, 2C, 6F, 66, 66, 73, 65, 74, 5D,
20, 907
580 DATA 20, 20, 73, 65, 6E, 64, 73, 20, 20,
69, 774
590 DATA 6E, 66, 6F, 20, 74, 6F, 20, 70, 72,
69, 945
600 DATA 6E, 74, 65, 72, D , 0 , 00, 00, 00,
00, 454

```


Robert Gault
 832 N. Renaud
 Grosse Pointe Woods, MI 48236
 313-881-0335
 <ab282@detroit.freenet.org>

The Industrial OS-9 User

F. G. Swygert

Specifications of 680x0 series microprocessors

Several people have recently asked about the differences between various Motorola processors. The information below was picked up off the Internet describing the features of the current generation of 680x0 chips.

	MC68000	MC68020	MC68030	MC68040
>> Introduced	1979	1984	1987	1989
>> Internal Data Bus	32 bit	32 bit	2x32 bit	1x32, 1x64
>> External Data Bus	16 bit	32 bit	32 bit	32 bit
>> Internal Address Bus	32 bit	32 bit	32 bit	32 bit
>> Addressable Memory	16 MB	4 GB	4 GB	4 GB
>> Virtual Memory	No	Yes	4 GB	4 GB
>> Clock Speed (MHz)	8-16	16-33	16-50	25-40
>> Instruction Cache	No	256 byte	256 byte	4 KB
>> Data Cache	No	No	256 byte	4 KB
>> Math coprocessor	No	Optional	Optional	Yes
>> Support dual processors	No	No	No	Yes
>> Memory Management	No	No	Yes	Yes
>> Pipelining	No	Moderate	Moderate	Yes

The Hardware Hacker

Dr. Marty Goodman

Putting the 'smartwatch' under the CoCo 3 DECB ROM

Many people have used real-time clocks on their CoCos. A popular one is the Dallas Semiconductor "smartwatch". This is usually used under the Disk BASIC ROM in the disk controller. The smartwatch will not (for reason's even I am not clear on) work properly if you just socket the CoCo 3's BASIC ROM and try to put the smartwatch between the CoCo 3 Basic ROM and the

socket.

There is, however, a minor cheat you can do to make it work that way. It involves bending OUT the chip enable pin (pin 20) of the smartwatch, and hooking THAT to the *CTS line. At smartwatch into the (nonexistant) disk controller ROM socket.

You can avoid actually bending a pin on the actual physical

smartwatch by doing that with an added socket you insert into this mess. Of course, you need to make sure that pin 20 of the CoCo 3 BASIC ROM does connect to the pin 20 spot on the original socket.

Well folks, I've spent the last few months living, sleeping, and eating serial ports. So I figured to do an article on serial ports....truth is, it's this or another few months of Russell Hoffman!

The Hardware:

Under OS9, you can *technically* use any hardware you have a driver for with any software. You can set up a multiuser system using the bitbanger printer port.... or hook a serial printer to a Fast232. Both would be wrong. The application software doesn't even know what's out there, but different jobs require different capabilities.

The biggest confusion about serial ports concerns their speed(s). There are actually three speeds to consider. 'Thruput' (sic) is how fast bytes can actually be transferred to the application. A stock CoCo with a Tandy serial port hovers around 600cps, and this figure goes up with patching, to about 1200 by full-bore Nitros9. Confusion comes when folks ignore this figure, and instead talk about the 'connect speed', which is how fast a character is sent. If both sides were always ready to go and noone ever waited for anything, thruput would be 1/10th connect speed (start bit, 8 bits, 1 stop, so 9600bps is 960cps).

With data compression, we get one last befuddlement- there is less to send after it's compressed, so the compressor must be fed faster than the modem connect speed. With two levels of handshake to slow it down, this 'port speed' is never attained, but manufacturers like to brag about it.

The other side of the problem is how much this process loads the cpu itself. As an example, the backpanel 'bitbanger' port. All this port does is connect the signals to a pia. The cpu has to watch carefully, and time each signal change itself, building up bits until there is a whole character. It's a stone slow process, and at 1200 bps the coco is completely tied up. If anything else happens, at all, characters are dropped and the machine might go down. Still, it is possible to grab a \$20 modem and try out the online world cheaply. Calling a free bbs, you do have the other windows around for &etc, just make sure nobodies sending at the time! Users of commercial services will want to use a DECB program, which can operate this software port at higher speeds by really taking over the cpu. The bitbanger has one advantage- it's free.

The next step is some sort of 6551 serial port. The cpu no longer has to time the bit stream- it can pick up a whole byte after acia builds it. With one of these, you can get about 700cps (1200 under nitros9). Which will connect up to 19200bps- only the fastest, compression modems can't be served. You are limited while using the port- it's not hard to hiccup a one byte buffer. For multitasking via remote terminals, it has a painful flaw. Initially moderate cpu load goes way up as baud rate increases above 2400 and by the time the serial link is running fast enough to be useful the apps it's linking have slowed beyond reason. Available cheaply (\$10-\$20 used to \$45 new).

Last stage is (currently) a 16550 serial port. These store up to 16 bytes in internal registers. Part of this is used to increase the time before the cpu is called, and the remainder handles some overflow in case of slow irq service or packet delays. CPU load is way down, to the point BBS or local host systems can think of one as a cpu upgrade. A fast 115k top connect, and faster 5000cps thruput. There is also a fancy \$80 price tag, and it's a bit of an i/o space hog. Just like a V8.

Software:

With some sort of hardware going, it's off to work! Let's start with the weird stuff. OS9 was written to operate over a terminal- forget graphics, a text console (imagine, a keyboard and screen connected directly to the computer!) was a bit odd. As a result, no special software is needed to use a terminal- except a copy of tsmon and login, which Tandy hid in the DevPak. There are free (and better) ones PD. (Note the console needs grf/windint and a pile of custom descriptors).

Now, lots of folks have tried this before, and it does work -with problems. Serial port overhead is compounded, since the CoC isn't just transferring files around, but actually running applications. Old multiuser boxes like the GIMIX used 'smart' ports to unload the cpu. As a result, a half-dozen users were able to use the same 6809. The 16550 isn't as good as the 256 byte smart cards I've seen, but it heads that way. A standard rs232 pak is perhaps ok for a single remote user, but once the cpu is asked to do very much the port overhead really begins to show.

Or not. Comparing remote users against a normal CoCo, the CoCo just looks slow. Turns out the console is a heavy load. The screen doesn't update that rapidly and there is lots of cpu load. We have pia's to read and

mice to compare. With a faster-than-Tandy serial port, the console user slows overall speed more than another terminal user. It's scary.

Beware especially the software mouse. On my system, the console runs MultiVue, so the main desk has a pretty interface. This is usually OK, since a single user won't be showing the gshell screen while doing something else. I'd log in from the shop, and there was a big difference in speed that couldn't be accounted for. Turned out to be the mouse! Mouse at left top, or console pointed away from MVue, system speed doubles. Try mousing around during a gfx printer dump.

Which brings us to another use for serial ports- the backpanel mouse has the same troubles as the backpanel serial port. And it's fixed the same way- using the 'smouse' serial mouse drivers and a hardware serial port. Which leaves the keyboard/screen...But I digress.

Software Setup:

Setting up OS9 programs for use over a terminal really depends on the program. At best, you'll be changing emulations on the terminal pretty often, since CoCo is a bit short on standards. The first set of programs are those that simply 'do what they do'. All the stock os9 utilities were written for simple terminals in the first place, and ASCII emulation is fine. Some (like proc) even display better under simple ASCII. There are plenty of old non-CoCo apps which work fine over ASCII....they are the ones that were always stone slow on a text screen.

Then there is stuff written for/on a CoCo, using Tandy's window codes. A distressing number of CoCo apps demand a Tandy screen. If your terminal is another OS-9 CoCo this isn't much of a problem, otherwise you'll need a DECB (SubEthra) or MS-DOS (Northern Xposure) terminal program with OS-9 emulation. The apps that need OS-9 screens range from the utilitarian DED on up. If you missed out on the build level two from rogue thing, you can at least call home and play a game or two. Try merging a CoCo Artist graphic to /t2.... (Interesting 'emulation' note- SuperComm just passes the codes on- emulate by default. Which makes the version shipped with Fast232 slower in 'os9 emulation' (legally dumped to scf) than the direct write ASCII and ANSI emulator screens.)

The last app group is the nicest- over the years there have been various standards for

defining a data file describing the display codes to be used over any i/o device. If you've been reading the whole mag, Joel's covered termcap well...but the CoCo predates termcap. It's not completely hopeless, some old apps (DynaStar) support it's precursor, termset. Others have their own custom display data file format, which can be corrupted in various ways to suit a non-OS-9 terminal. No two 6809 apps seem to follow the same 'standard', so you might find yourself setting the same thing up 24 different ways- and most of the entries in these old term files probably refer to terminals long since extinct, although vt100 terminals display ANSI perfectly well.

If you are the prudent (paranoid?) sort, note the standard CoCo practice of loading lots of merged together utilities at boot bypasses much of OS-9's security- stuff like format and attr have to be separated from the common herd. The best approach is to log in as a public user and try to kill you system. Make it a hobby!

On to 'normal' telecom....from os9's view, this isn't normal at all, in fact the system very much wants to be the server. Getting OS-9 to dummy up enough to make a good terminal is difficult, but the work has been done and there are quite a few good public domain telecom programs for OS-9. In fact, they are good enough there are very few commercial ones! Of course those neat programs are available free.... from electronic sources.

This is perhaps the biggest hurdle- I well remember attempting to download a 'real' program with DeskMate2 (ASCII buffer capture only). CoNect will of course offer the traditional disk of stuff (\$5) for those who are quite stuck.

Another problem is the general style of OS-9 telecom. There isn't a single all inclusive program... instead, a collection of utilities is needed. First up, download protocols. Most terminal programs have a quite a few built in, but even here, OS-9's modular style shows- with 'external' download utilities. Sometimes the user has to physically clear out and run them manually, while some terminals (Supercomm) call rz and sz automatically (Really. Anytime a zmodem header or ready to receive appears in the data stream, the up/download window pops up). Pretty cool- but beware, zmodem is very slow on a CoCo, and probably works best at 2400. Note that Omen's sz/rz is shareware.

Even after the file has been downloaded, there is possibly more to do. Many files are archived, and a separate dearchiver is needed to 'burst' them. Older files generally fall into three main groups, arc, pak, and ar. Recently, the OS-9 CoCo has joined the rest of the world with lha (which bursts lzh files). There is also a much improved ar for OS-9-only files, and an unzipper.

But we aren't done yet- Even if you never stray beyond the confines of the friendly local bbs, internet access is so universal you'll eventually have to send or receive a binary file through a ASCII-only mailer. UUencode and Uudecode

will morph anything to/from legal, printable ASCII.

The easiest way into telecommunications is to subscribe to one of the big, flashy online services. Some even offer custom interface software, unfortunately none of this is available for the CoCo. With a few exceptions, you don't *have* to use the flashy interface, and the traditional 'terminal emulator' (we have plenty of those) works fine. Not because the basic terminal software is that good- these services invest alot of time and money in accomodating as many types of machines as possible. By the same token, almost any download protocol is supported, and 'binary' (non-text) files usually arrive in usable form. Obviously, some methods are better (or more popular) than others but many things will work.

A small BBS can be thought of as a limited version of a big BBS....you will be ushered to an ez/friendly menu which leaves little doubt what you can do.

An alternative to the traditional BBS is connecting directly to a machine that is 'on' the internet. This is just like logging into your CoCo- there isn't any structure, instead you are dumped uncerimoniously at a command line prompt. A bit of a pain, but with many schools and other institutions offering access this is becoming more popular everyday, and you can't beat the price. Check around... uwm has started a community machine (omnifest) with a low low \$25 a year fee (Look for my new E-mail address soon).

There is one major difference- you must adapt to the machine, not the other way around. Probably the host will be vaxen, Sun flavored unix, or something similar, so you'll have to brush up on simple unix a bit (see below). Simple ASCII terminals will probably give you access to the command line, expect many applications to require VT100 or better. With many CoCo terminals ending in the ANSI range, this could be a problem. KBCom (Eddie Kuns) not only does a good enough vt100 to display properly, it trips the 'screen type detectors' on autosensors. Kinda fun to be compliant. With vt100, the full screen applications will work.

Some systems do have a menu driven front end, but it might not pop up on it's own. Try some obvious choices like menu or ez, and you may get lucky. No menus? The unix manual system is left. Type 'man command' to see the manual

pages for 'command'.

Learn emacs 'key bindings'. Not intuitive at all, but close to universal- everybody has emacs. Other programs (newsgroup readers and the like) often use emacs style controls as well.

There are at least two CoCo versions of micro emacs that use them, so practice is cheap. Some very typical programs:

- man * - manual page(s) for *
- ls - short data directory
- dir - extended data directory
- emacs - full screen editor
- more - file lister
- pine - email reader
- elm
- nn - usenet news reader
- tin
- logout - opposite of login

Many files will be uucoded anyway, and so can be downloaded with a simple buffer capture. One quirk, anytime more gets to send it's prompt, it doesn't add a line feed. When you are going thru the buffer cleaning up the dump, make sure to check before line deleting any more? prompts.

If you've ftp'ed a file to 'your' unix box, there is always sz (which usually does x and y modem also). More work for the man command- or look over the docs for the CoCo version. Only difference is *all* the switches work from a unix box.

Want to move your CoCo keyboard? Get a

Puppo PC/XT Adapter

Allows use of any PC/XT compatible keyboard with your CoCo. Easy access to OS-9, DECB, and other options. Hold space bar on power-up to skip menu. Switchable and most auto-sensing keyboards supported.

Only \$72.50 post paid!

\$100.00 post paid with keyboard!

FARNA Systems

Box 321

Warner Robins, GA 31099-0321

NOTE: Keyboards will be 101 key. FARNA reserves the right to substitute 84 key models if necessary without notice.

Programming the System IV (PT68K4)

David Wordell

How to read the Function Row Keys and the Cursor Key Pad Keys

For those of you who do not know, I have a PT68K4 machine, running OS-9/68000 Version 2.4. This is essentially the System IV in a kit form. This article describes some of the things I have been learning to do with this fantastic machine.

After I got home from one of our club meetings, I was fired up. Several club members had helped me to find the solution to a problem I had been working on for some time. The problem was "How to read the Function Row Keys and the Cursor Key Pad Keys on my PT68K4 and use them in a C program". Those are the "F" keys, F1 through F12, the four arrow keys, and the delete, end, home, insert, page down and page up keys. The problem stems from the fact that these "special" keys send a three character code instead of a single character code like the a, b, c, d, etc. All the special keys send Escape (1B), Square Bracket, [, (5B), plus some capital letter such as A, (41). The numbers in parenthesis are the hexadecimal values. The up arrow will send Escape, Square Bracket, A. The down arrow will send Escape, Square Bracket, B, or ^[B. I'm using the ^ caret to signify the Escape character, hex value 1B. The right arrow will send ^[C, and the left arrow will send ^[D. The rest of the special keys are just like the arrow keys, with the last character being the determining factor. We made use of this unique sequence as you will soon see.

Lets look at the page with the program on it. First you will see that we include the `stdio.h` file as usual. Then we include `keybd.h`, a header file I created from what I had learned at the club meeting. We'll get to that in a minute. Next the `main()` function where the program begins. In the first line of the program we declare an integer array with room for two integers. One for the integer we want to store and one for the NULL that C will put at the end of the array. This array will be passed to the `key()` function which is actually in the header file `keybd.h`. In this example we create an endless loop, to search for the key, with the statement `while(1)`. Since we did not use a variable in the parentheses, it can never become zero, or false, to stop the loop.

Now we call the `key()` function, sending the array "keybd" by putting it inside the parentheses. Let's look at the function. It's name is `key` and we indicate that it will receive an argument called `keybd`. The next line indicates that `keybd` will be an integer array. Then we enter the body of the function. First we declare two pointers to character strings and create the strings "tmode noecho" and "tmode echo". Next we declare a character variable named `inchar`.

This is the part I think you will love. It should open many doors for you C programmers. There is a built in function called `system()` that in essence, allows you to open a shell, issue an OS-9 command, and return to your program. I was having a problem with the reading of keys. What ever key I hit, it was printed to the screen. There are many instances that I don't want this to happen. I used the `system()` function to send the string "tmode noecho" to OS-9, just as if I had typed it on the keyboard to a shell. No more keys being printed to the screen. Bye the way, on the CoCo you would use "tmode -echo" but "tmode noecho" is the OS-9/68000 version of this.

Next we clear the integer array to zero, always a good practice, to be sure it does not contain a previous character. Then we initialize the character variable to the escape character. We discovered, with previous experiments, that the `read()` function that we are about to use did not seem to be able to read the escape character. If the variable `inchar` had contained some other value and an escape character was sent, `inchar` would retain the previous value. Now put your reverse hat on. If the above is true, and we make the value of `inchar` 1B, the escape character value, and we receive an escape character, `inchar` will still have the 1B value after the read. Not because it read the escape character, but because it did not read it and retained the previous value. On the other hand, if any other character is read, it will change the value of `inchar` and it will contain the new value after the read. Sort of like turning a "bug" into a "feature" isn't it.

Anyway, we now check for the value

of `inchar` to be 1B with the `if` statement. If the statement is true, we read another character. If not, this was not one of the special keys and we are "out of here", back to the main program. For now, assume that `inchar` was 1B. If the key was one of the special keys, the next character should be the "[" or 5B. We use another `if` statement to check for this. Again, if no, we are "out of here". If yes, we read the last character. This will determine which one of the special keys we saw. We then make the integer array, `keybd[1]`, value the same as the last character read. This will be passed back to the main body of the program. One last task before leaving. We use the `system()` function to send the string "tmode echo", turning the echo back on, before returning to the main program.

Now we are back at the main program and we have a value representing the key pressed in the array, `keybd[1]`. Using the `switch()` function to compare the value, we make four "cases" for the four arrow keys. Any other value we will call an "invalid key" for this program. With this example, we now have a workable function, which will allow reading any of the special keys from the keyboard of a PT68K4 or System IV, and using them in any program that we wish.

In order to compile this example program you must create the header file called `keybd.h` and place it in your `DEFS` directory with all your other header files. The "#include <keybd.h>" statement in the program will then take care of the rest.

Once again I must thank Ed Gresick for giving me a copy of the keyboard codes for a System IV, at the last Atlanta Fest. Without Ed's generous help I would not have a clue as to what the keyboard codes were. Thanks Ed. I really appreciate the doors you have opened for me.

David Wordell
833 Woodhaven Ln
GrandPrairie, TX 75052

(program listings on page 6)

Controlling a laser light show using a CoCo...

Editor: *Many of the regular Atlanta CoCoFest vendors have seen the laser light show at Stone Mountain Park in Atlanta. The laser light show is actually controlled by several Zilog Z-80 based microcontrollers. Why can't the CoCo do this? Well, it can... Steve explains a lot of theory in this and upcoming articles. Even if you aren't interested in building your own laser show equipment, he'll explain a lot of graphics theory also...*

I have been working on a low cost but very sophisticated laser graphics system for some time, which is based on the CoCo. I spent about 3-4 years on my software to get a very versatile capability, anything the big boys do, just not as much or as fast...even some things some big boys can't do...and I'm an RF engineer turned digital.

I got hooked in about 1985 after seeing what one of my co-workers (software type) had developed with a U. of Illinois professor over a period of years. He now has his own business doing large laser light shows. In fact, the first company he founded got the government contract to do laser light shows on Grand Coulee Dam! He seemed to know what he was doing on the software end but I thought I could help with hardware.

I had some ideas I wanted to try to speed things up so I started to play around. I got some stuff and after a year realized I had developed the basis for a small and inexpensive but "full function" laser graphics system. Now, I found that the "big boys" are all trying to do more, bigger, faster, etc. Mine couldn't compete with these but I figured there must be people wanting to spend \$\$ on this stuff but not the \$50K+ needed for the "professional" systems.

Obviously there are many ways to configure something like this but I decided to design a small (1/2 cu. ft.) self contained, laser graphics system. Plug it in, turn it on, watch the show! The images displayed are vector images, basically dot-to-dot drawings. The vast majority of functions are carried out in software. The only hardware other than the computer is three 8 bit D/A's, three scanners and the associated driver amps.

I'll focus first on the muscles of laser

graphics, the scanners. The connecting lines between dots are drawn in hardware, that is, real, moving mechanical pieces, chunks of iron-like hardware. This key component is the galvanometer, or galvo, also called a scanner. A galvo looks like a small motor, about one to three inches on a side. The galvo shaft, however, does not go around and around, but has a limited travel from the central resting point. It is just like the old D'Arsenval (sp) meter movements. A spring holds the shaft in the central resting position and current through the coil causes the shaft to rotate away from center to a new position. More current causes it to go further from center. What we have is a current-to-angle converter. One of the parameters for a galvo is its "degrees per ampere"; how many degrees the shaft moves for a given current. The ones I use are about 35 degrees per amp., however, they are not moved that much when in use and probably couldn't take it. I move them only about seven degrees and that takes 200 milliamps. The mirror goes on the shaft, parallel to it, forming a little tennis racket shape.

There is one thing here which helps us and probably the only thing in nature where you get something for nothing. Because the angle of reflection is equal to the incident angle when reflecting off a mirror, the angle the light beam is deflected is twice that of the mirror movement. Move the mirror five degrees and the beam goes ten. If you don't care for further explanation, skip the next paragraph.

Start by thinking of a ray of light coming in perpendicular (also called normal) to the mirror. It will be reflected back on itself, also normal to the mirror. Now rotate the mirror, let's say, five degrees. The incoming ray is now five degrees off the normal, because the normal moved that much. The reflected ray will be five degrees off, but on the other side of normal, so it moved ten degrees. Double your money for free; all thanks to Snell, or somebody.

There are two other types of scanners you may see advertised. These are not used in laser graphics except for some special effect. The first is the resonant scanner. It looks just like the galvo, and

works much the same, except that it is made to operate at one single frequency of vibration or oscillation and one amplitude or excursion. It just swing back and forth at one rate.

The second is the rotating polygon scanner. This is a motor which rotates at one constant speed, on one direction. The mirror, however, has many faces. The faces all point outward. Like putting mirrors on the outside of a cylinder. As it rotates, each face takes a turn at sweeping the beam through the same line, in saw-tooth fashion.

Now, how do we get pictures out of this, you ask. Well, one galvo and one mirror allows the movement of the beam anywhere along a line. That is, the dot of light from the laser can be moved, or swept, along one axis. By using two galvos, with shafts at right angles, and the mirrors turned correctly, you get an X-Y space on the screen. One galvo positions the dot horizontally, the other vertically.

The astute reader may notice that the current-to-angle-to-screen concept introduces an error from the fact that the distance moved on the screen is the tangent of the angle, not the angle itself. Also, that the tangent is a non-linear function. I think everyone ignores this for the small angles we use. One galvo is manufactured to give tangent correction, but I'm sure no one uses it for graphics.

The drive electronics for these things is basically a power op-amp., voltage-to-current driver. It is deceptively simple, however. The basic mechanical galvo will overshoot and ring, or oscillate about the new position and slowly dye out. To prevent this, some form of feedback must be employed to get maximum speed without overshoot. When you draw something like, say a square, the dot, and hence the galvos must move to the new point and stop, not passing it up. This is called critical damping and without going into all the control / feedback theory, this is prevented by using the feedback signal. There are two kinds of feedback in use. The most common is position feedback. The galvo generates a signal voltage proportional to its position. The other, velocity feedback and the one I use, generates, as you would



BlackHawksoft

244 S. Randall Road • Suite #172 Elgin, Il. 60123
(708)742-3084 eves & ends • MO, Check, COD; US funds
Shipping included for US, Canada, & Mexico

MM/1 Products (OS9-68000)

COMING SOON!!! CDF - CD-Rom file Manager! Unlock a wealth of files on CD with the MM/1!

VCDP \$50.00 - New Virtual CD Player allows you to play audio CDs on your MM/1!! Graphical interface emulates a physical CD player. Requires SCSI interface and NEC CD-Rom reader.

KLOCK \$20.00 - Optional CUCKOO on the hour and half hour!! Continuously displays the digital time and date on the /term screen or on all open screens. Requires I/O board, audio cable, and speakers.

WAVES ~~vr 1.5~~ \$30.00 - Now supports 8SVX and .WAV files !! Allows you to save and play all or any part of a sound file. Merge files together or split into pieces. Record, edit and save files with ease. Change playback/record speed. Convert Mono to Stereo and vice-versa!! Record and Play requires I/O board, cable, and audio equipment.

SOUND CABLE \$10.00 - Connects MM/1 sound port to stereo equipment for recording and playback.

GNOP \$5.00 - GNOP is the AWARD-WINNING version of PONG(tm) exclusively for the MM/1 !! You'll go crazy trying to beat the clock and keep that @\$%& little ball in line! Professional Pong-ists everywhere swear by (at) it !!! Requires MM/1, mouse, and lots of patience.

Coco Products (DECB)

HOME CONTROL \$20.00 - Put your old TRS-80 Color Computer Plug 'n' Power controller back on the job with your Coco 3!! Control up to 256 modules, 99 events!!

HI & LO-RES JOYSTICK ADAPTER \$27.00 - Tandy Hi-Res adapter or NO adapter at the flip of a switch!!

KEYBOARD CABLE \$25.00 - Five foot extender cable for Coco 2 and 3. Custom lengths available.

MYDOS \$15.00 - CUSTOMIZABLE! EPROM-ABLE! The commands Tandy left out. Optional 6 ms. disk drive speed. Supports double-sided and forty track drives. Set CMP or RGB palettes on power-up. Power-up in any screen. Speech and Sound Cartridge supported. Point and click mouse directory and MORE. For all Coco 3 with Disk Basic 2.1. More options than you can shake a joystick at!!

DOMINATION \$18.00 - MULTI-PLAYER STRATEGY GAME! Battle other players armies to take control of the planet. Play on a hi-res map. Become a Planet-Lord today! Requires CoCo 3, one disk, and joystick or mouse

SMALL GRAFX ETC.

- "Y" & "TRI" cables. Special 40 pin male/female end connectors, priced EACH CONNECTOR ----- \$6.50
- Rainbow 40 wire ribbon cable, per foot ----- \$1.00
- Hitachi 63C09E CPU and Socket ----- \$13.00
- 512K Upgrades, with RAM chips ----- \$72.00
- MPI Upgrades
 - For all large MPIs (PAL chip) ----- \$10.00
 - For small #26-3124 MPI (satellite board) ----- \$10.00
- Serial to Parallel Converter with 64K buffer, cables, and external power supply ----- \$50.00
- 2400 baud Hayes compatible external modems ----- \$40.00

ADD \$2.00 S&H TO EACH ORDER

SERVICE, PARTS, & HARD TO FIND SOFTWARE WITH COMPLETE DOCUMENTATION AVAILABLE. INKS & REFILL KITS FOR CGP-220, CANON, & HP INK-JET PRINTERS, RIBBONS & Ver. 6 EPROM FOR CGP-220 PRINTER (BOLD MODE), CUSTOM COLOR PRINTING.

**TERRY LARAWAY, 41 N.W. DONCEE DRIVE
BREMERTON, WA 98310 206-692-5374**

INTRODUCING THE MM/1B!

A new machine based on a board produced by Kreider Electronics featuring :

- o 16 bit PC AT I/O Bus with 5 slots
 - o MC68306 CPU at 16.67 MHz
 - code compatible with 68000
 - 2.4 MIPS
 - o 0.5MB to 16MB DRAM (4 SIMM sockets)
 - o IDE Hard Disk Interface (2 drives max)
 - o 1.44MB Floppy Interface (2 drives max)
 - o 2 16 byte FIFO serial ports (up to 115K baud)
 - o Bi-directional parallel port
 - o On board RS232 buffers
 - o Battery Backed Real Time Clock
 - o AT Keyboard Interface & Standard AT power connector
 - o Baby AT Size Footprint
 - o BASIC (resembles Microsoft Basic)
 - o MGR - graphical windowing environment, full documentation!
 - o "Personal" OSK V3.0 (Industrial with RBF)
 - Display drivers: Tseng 4K, generic inexpensive VGA
 - SCSI card support: Future Domain 1680 & Adaptec AAH 15XX
 - o OSK version 2.4 including network file manager, PCF, SCF, SBF, RBF, Pipeman, RamDisk, MW C Compiler version 3.2 with r68/168, MW Basic, MW Debug, MW Programmers Toolkit (Mail, print spooler, UMacs)
 - o UUCP package from Bob Billson
 - o Ghostscript (PostScript interpreter)
 - o Many other utilities and tools
- Pricing as low as \$400!**
(motherboard, Personal OSK, & MGR, no RAM)



BlackHawk Enterprises, Inc.

P.O. Box 10552
Enid, OK 73706-0552
Phone 405-234-2347
Internet: nimitz@delphi.com

guess, a signal voltage proportional to the (angular) velocity of the galvo. Suffice it to say that this signal can be put into the driver circuitry in such a way to keep the ringing out, but let the galvo get quickly to where it is going. My galvos take 4 milli-seconds to move from point to point.

Yes, there are not-so-mechanical ways to move a laser beam like acusto-optic, Kerr cells, but you're getting into the bucks and I don't know much about them anyway, but I believe there are some use of them.

So this all describes a voltage-to-position output device for two axes. Input an X and Y voltage and you get a spot at that point (and a line from the previous point).

This brings us to one more aspect of the projector, the "blanker". It would be nice, at times, to be able to move between points and NOT draw with light, so we need a means of blocking, or blanking the beam. The first inclination might be to put another galvo in there with a black vane to move in and out of the beam - and this is the predominant method of intensity control - now we have voltage-to-intensity. Being cheap, I went back to what is practically a folk art requirement of the beginning laser graphics student, the speaker. Yes that thing inside the speaker box which makes the music come out - to the audiophile, the driver - you know, the thing with the big magnet and the voice coil of fine wire and the dark blue paper cone that seems to invite the pressure of your finger....

Since these things move fast, everyone who gets interested in drawing with laser light thinks of them and attempts to use them to do the deflecting - and usually with measured results - it's a tough battle I only thought about. Having majored in control theory, I knew the galvo route was far easier; particularly after finding the lower cost velocity feedback galvos adequate.

A small 1.5 inch dia. speaker, with an added vane, works nicely, not only to turn the beam on and off, but to vary the intensity. Add another voltage-to-current amp and you have the projector part of the battle done. The rest...oh yea, except for the three D/As, is all in software.

You know, I have this theory that hardware and software are interchangeable just like matter and energy. You can do it one way or the other...but that's

another article. I'll move to the software, after I digress a bit here to cover related hardware.

You say you'd like to play with this yourself? It helps to have a laser and there are many places to get them; I'm not covering that here.

First, the cheapest, yet lotsa bang-for-the-buck, is a three motor (or more) cycloid system (two isn't bad to start), but all you get is patterns, more motors, more neaterest patterns!! Each motor has a mirror on the end of the shaft that is almost perpendicular, about 2 or 3 degrees off is good (remember the beam moves twice as far as the mirror - angle of incidence equals angle of reflection). The mirror is sitting sorta like a wheel on the motor shaft but it has a little wobble and one of them makes a circle on the screen. For the first one I tried, I used child's modeling clay to attach the mirror to the shaft and I think its still attached.

The incoming laser beam is placed slightly off axis so the reflected beam goes back past it and to the screen. With addition of the second motor/mirror, the beam is directed back away from the laser and the third, back toward the laser again. You will be restricted by the motor size when it comes to positioning the laser and motors since the beam has to clear the motor bodies. Also, the first mirror can be the smallest, but each successive mirror must be big enough to intercept the entire pattern from the one(s) before it. So the idea here is to get the mirrors as close as possible, yet get the beam past the motor housings. You change the motor speeds and directions - have fun. Just remember, motors have a finite lifetime, cheap ones wear out fast when run continuously. I also know of a product which is a real laser and a wobbly mirror with two coils which you hook up to your stereo and it draws neat designs directly from the music.

Now, you say, fergit-it, I want to do cheap VECTOR graphics, like the big boys...not just loopity-de-loops...we're talkin' words and things...dad-blast-it there MUST be a way to do it for just a couple-o bucks!

Well, EVERYBODY (even me) who wanted to do it said something like "HEY, Speakers move fast, I'll start there and improve on that idea." This is the folk art part, though maybe it should be called folklore. Well, it can sorta be done. The speakers can't get very close together and if you start making linkages it gets to

be a Rube Goldberg nightmare. A company called BEI makes fast piezo moved mirrors, but they have deflection WAY under a degree. All the really serious fellas get tired of fooling with speakers and quickly move to galvos, without exception.

The workhorse of the laser entertainment industry is the General Scanning G120 galvo. These are probably the fastest, position feedback galvos, but are in the kilo-buck range. There are more affordable ones that DO give usable graphics. They start at about \$130. A word from the experienced here. Once you decide to get out of cycloids and into vector graphics were talking dedicating your life to writing SOFTWARE. Unless, of course, you want to buy (eek! there's that word) software from someone like me. With the big boys, like MKW, were back to talking BUCKS!

I recently found out about a guy in Fla. who claims to have a full, open loop X-Y galvo set for about \$130. (made in Japan...its about time) He claims that they are "almost" critically damped - an absolute necessity for vector graphics, though he is doing the pattern thing and isn't too worried about it. You can make up for a little (maybe 5-10%) overshoot in software, but this will mean that you are not getting the full speed out of the thing. Course, if

that's all you got, maybe you'll take it. I've only talked to him and, like I

say, he wasn't too sure about some things. He also has some GS Galvos for around \$100, but they have no feedback so they can't be damped. Back EMF feedback alone won't do enough, I tried.

Rob Elkins
RK Manufacturing
4018 North 30th Ave.
Hollywood, Fl. 33020
(305) 963-2948 (machine)

What DO I use? Well, that's sorta my secret, but they are General Scanning Galvos. They are slower units for strip chart recorders and they are velocity feedback only. They're rated at 100 Hz but I'm getting 150Hz 3dB BW (sine wave). I get a 4ms. critically damped step response which is good enough to get a five letter word, block letters, with minimal flicker. I use current drive with velocity and voltage (back EMF) feedback. They were about \$130 each new 2-3 years ago. The fella who got me

interested in laser graphics gave me two and said, "here, I found these in some surplus house, you can't really do anything with them, but you're welcome to goof around"

Now I get to the things more closely related to the software and the CoCo. Individual images are drawn (entered) on the CoCo's X-Pad. A single push of the pen stores the X and Y coordinates of a single dot or vector. Thought I don't use 3D images, those who do, will also have the Z coordinate in there; "drawing" the third dimension gets a bit tricky and I'm not going into it here. Drawing a 2D image is not a particularly complex function, so I wrote my own software for it and Basic is fast enough. I have two ml routines for speed. One puts a drawing grid on the screen and the other tracks the end of the image as points are edited in / out.

Though it generally works quite well, the X-Pad will hunt between adjacent pixels occasionally, so a little sluggishness in the software is a bit of an advantage. To make the drawing of written words and other smooth images easier, I added a switch on a cord to duplicate the pen's tip switch. It is held in the other hand and allows the image to be drawn without requiring pressure on the pen tip, which is awkward to maintain. When holding the switch down, vectors are captured at about two per second as you "write" with the pen. I also replaced the ball point pen cartridge with a steel rod, smoothed on the end. The tip is a little larger than the pen tip and easier to slide around. You are usually looking at the screen when drawing, so there's no advantage in having the pen write. In fact, it is a disadvantage since at times it is helpful to draw the image on a piece of paper, first, to allow seeing it and making adjustments before committing it to RAM. Then you can trace over the paper on the X-Pad.

This gets you two bytes for each vector, an X and a Y, but there's one more thing to consider. At times we will need to have disconnected lines in our image, so we need to turn off the beam and move the scanners. It is also usual to turn off the beam at the end of every image so there is no line to the next image when multiple images are displayed. A third byte per vector holds the intensity. When drawing images, the intensity is usually just either on or off and this is toggled with the B key.

For a blanked line, I use one of those obscure POKEs, into some graphics variable, to get a dotted line on the screen.

When you need to turn off the beam within an image, you may need to put some extra vectors at one point to make sure the beam gets there before blanking, then you shoot a couple of blank vectors to make sure it is off before it moves away. Sometimes you have to experiment to find out how many of these extra vectors are needed. The delay between vectors can be different for each image and depends on what the image is, so the number of blanks required varies. The time required is the time for the scanners to stop and for the blanker to move in / out.

One bit of the intensity byte is an end-of-image flag so the display processor knows when the image is done. This makes it also a status byte, but I use the terms interchangeably.

So now we have an image (a vector list) in RAM and on the screen (I draw lines on the screen between points as I captured vectors).

The display size is 256 x 256. Since the PMODE 4 screen is only 192 high this could be a problem, but there aren't many images which need to go below that. Many of the computer generated patterns do, but its better to see them on the projector anyway; it's usually necessary to adjust the inter vector delay to get the desired result.

I have a routine I call the circle generator which uses the sine and cosine functions to generate what are technically lissajous patterns, but get quite complex due to variable k-factors which are manually entered. This produces some interesting patterns including multi point stars and some very interesting quasi-animated thingys. The X-coord. is the sine and the Y-coord is the cosine of the angle. However, to get more than just circles out of it, the angle has a multiplier which is different for the sine and cosine. The three things entered at the start are:

Degrees per step of the angle which normally starts at zero and stops at 360.

The sine (horiz) factor placed like so: $\text{Sin}(\text{hk} \cdot \text{angle})$.

The cosine (vert) factor placed like so: $\text{Cos}(\text{vk} \cdot \text{angle})$.

Specifying these (as well as the degrees per step) gets to be a real science and very large values of hk and vk, in the thousands, can produce the best re-

sults. I must point out that the effect of some of these is completely different on the laser projector than on the more static screen. This is also done in Basic since floating point is required and the image is created only once so there is no real time requirement.

As I said above, the time between vectors is a variable for each image. If you are drawing a square, you want the beam to get to each vertex before proceeding to the next, for nice square corners. In this case, you want a delay equal to the step response of the scanners. If you draw a circle, you don't want the scanners stopping or even slowing significantly (except for a special effect), so the delay is made small. This way, the scanners are constantly "chasing" the vectors coming from the D/As and a smooth curve is drawn.

If you need both in an image, you can set the delay short. In the places where you need full stops, you can put several vectors in the same place to wait for the scanners to catch up.

Since I elected to have this "inter-vector delay" determined by the software calculation loop timing, delays are typically over 255 requiring this to be one of the few word variables.

Frequently, images need adjustment to get what you want, so typical select then move, delete, insert functions are available. Here is where another ml routine speeds things up by scanning through the image's vector list to find the nearest one for editing. The algorithm I use here is to scan through the X coords until one is found within 4, then see if the Y-coord. is within 4. This finds the first point within a 4X4 box and works fine.

The last aspect of images is saving them for later use. Of course, that's on disk, and I simply use SAVEM and the transfer address is irrelevant. To make them re locatable, I go back and change the load address in the file to \$0000 allowing me to use the LOADM offset as the load address. In thinking about this now, I realize I could save a disk write if I left the load address as is and simply read it before re-loading the image and calculating the offset to put it where I want.

The display processor is the code which is running when an image is being displayed by the laser projector. It displays the images, applies all the motion parameters, and has the ability to inter-

pret a list of commands to make these images come and go, move and jump and spin....all in about 2K of code.

Several images can be displayed at a time and each is controlled independent of the others. The 6809 is really well suited to the complexity I have chosen; just enough registers and capability. I needed four pointer registers and some twos complement math.

There is one thing to be noted here before going on to motions. The coordinates in an image's vector list are never changed once it is drawn and editing is complete. Starting with an X or Y coordinate from the vector list, the appropriate parameters are applied in a temporary variable then the final value is sent to the scanner and discarded. The scanner is the bit bucket. If they were changed with every calculation, the accumulated error from the limited precision would distort the image after a few passes. Images exist in what is called "Image Space" or the "Image Universe". That is to say that images are stored full size, centered on the screen and un-rotated. The origin (0,0) is usually in the center of the image in image space. This is important because this is the spot on the image which gets moved to the desired location, and it is the center of the size changes and of rotations.

When we go over...er translate...or is it transform to the display space (or universe), we want to simulate real world actions so we will want to select which images to display, move them around, change their size, and rotate them.

These operations can not be performed simultaneously, so the order of operations must be given some thought or strange things may result. I elected to do the rotations first so that perspective and depth cue were always seen in full force. This is actually not natural since small things have little perspective. I do this to enhance these very interesting effects and nobody has objected yet. Also, the calculations are done real time, one vector (point) at a time, as they are sent to the scanners.

To see what can happen by selecting the wrong order, lets look at moving an image before scaling it. Remember bytes and 256. The way I did it, the center of image space is 128,128. Image coordinates can go from (0,0) for the lower left to (255,255) for the upper right. If an image is large (and remember they usually all are) and we add a move offset

such that the resulting coordinate either goes over 255 or under zero, the result will do what we refer to as "wrap". The resultant byte will still be between zero and 255, but it will enter the other end, so to speak. Now, there will, of course, be a carry, but I would have to be using double precision to account for it. That takes time, and that we ain't got when using the 6809 at under 1MHz and doing all this stuff. So, by scaling before moving I stay inside the universe. Basically this means "DON'T WRAP". When you do wrap, that which exits the universe on one side enters on the opposite side. Since everything is done point by point, this tears an image apart. I have a mathematically closed number system of 0-255.

Motion calculations are done on the fly, one vector (point) at a time, as they are sent to the scanners. Moving an image around the screen is simply a matter of adding an X and Y offset to the X and Y coordinate, of each vector in the image, respectively. There is an X offset for moving horizontally and a Y offset for moving vertically. The move offset is the last to be applied which greatly helps avoiding wrap.

Changing the size, or scaling an image, is accomplished by multiplying each coordinate by the scale factor which goes from 0 to 1 (0 to 255 at the processor level). Since the image in image-space is at the origin, scaling the coordinate values changes its size. There are independent X and Y scale factors allowing special effects.

Now comes the neat stuff. I only to 2D images, but they do rotate into the third dimension and there are some tricks to get a full 3D thingy on the screen.

I'm not going to go into full detail on the rotation calculations here. I will provide technical information in another document. There are some things which work in concert to make a very nice package. To calculate the rotated X & Y coordinates, sine and cosine functions are required. By restricting the rotations to a fixed number of steps things can be greatly simplified. A fixed number of angles suggests a look-up table and 256 entries is a nice number.

Something very nice happens when this is done. An eight bit pointer register makes an ideal trig table pointer in this case. This register contains the angle and the entry it points to is the cosine in my case. Since the sine and cosine

functions both repeat after one cycle and the pointer register will also "auto-repeat" the table, the pointer register makes the ideal trig table pointer. There is absolutely no overhead required to baby sit the pointer to keep it within bounds. Increment or decrement it without consideration for any overflow, you always get the proper value. This gives a step size of 1.4 degrees which gives very smooth rotations.

Also, only one table is needed since the sine and cosine are identical except for being shifted by 90 degrees from each other. In this case 90 degrees is 64, so subtracting 64 from the pointer, points it to the sine. Again, over/under flow is ignored.

To rotate an image, each vector is taken one at a time. The image space X coordinate is multiplied by the appropriate value from the trig table. If needed for the particular rotation, it is added to a similar product of the Y coordinate and proper trig value, then sent to the scanner. The exact steps are different for rotations about the three axes and are spelled out in the technical section following this paper. The convention I selected for the axes puts the X axis horizontally, pointing right on the screen, the Y axis vertically, pointing up on the screen, and the Z axis coming out of the screen.

This was the biggest challenge of the display processor. The 6809 has an unsigned multiply, so I had to build a signed multiply - the first attempt resulted in images being torn apart when rotated. The routine simply looks at the sign of the operands, forces them to be positive, multiplies and makes the answer the proper sign. I also made it a macro.

When rotated, the images have perspective otherwise there would be virtually no illusion of rotation. That is, parts of the object which have been rotated further away from the observer should look smaller; and closer parts should look larger. This tends to be one of the more esoteric concepts in graphics, but it can be simplified, as I will show.

The first thing which is needed is the Z distance of the newly rotated point, provided it has been rotated out of the X-Y plane into the Z dimension. This is calculated using the same trig table. Now, true perspective is calculated by dividing the X and Y coordinate value by the Z distance (with appropriate selection of the screen and viewer locations).

Division is a lengthy process for the 6809, so I developed an effective and quick approximation which I call "25% Linear Perspective".

Taking a simplistic view of perspective, I reasoned this way. If I make the Z value go positive and negative, it can be used to increase and decrease the size of the image as it moves toward or away from the viewer. This is just like the overall scale function described above (applied to single vectors) and is done by changing the distance from the point to the center of the object. Because the center of the object is at the origin (0,0) this simply means increasing and decreasing the coordinates. The simplest and fastest way to do this is by addition. I take the Z coordinate, divide it by four (this is just two shifts right) so it only effects the size a small amount, and add it to the both the X and Y coordinates for that point.

I also included what is called "depth cueing". This is a somewhat un-natural but effective technique used to further enhance the the depth illusion where the parts farther away get dimmer. For this, the Z coordinate is added to the intensity byte before sending it to the blanker. The polarity is such that moving away dims the intensity.

Although not a motion in the classical sense, one final parameter which is part of the motion parameters is intensity. By allowing modification of the entire image's intensity, I can fade images in and out.

The 6809 has four pointer registers and that is just what I need. One is used to point to the image X, Y and status bytes when stepping through the image and applying the motions. The second points to the image's parameters.

The third points to the trig table. The fourth point to the commands to be described later.

Keep in mind that all the motion operations are always applied which means that an image which is displayed centered on the screen ,full size, and unrotated,;

P

1- Has zero added to all the coordinates.

2- Has all coordinates scaled by one.

3- Has been rotated zero degrees with zero perspective added.

With everything I've described so far, there are eight motion parameters for an image 2-Move, 2-Scale, 3-Rotate, 1-In-

tensity. Because it is always better, when possible, to have the computer do the work for you rather than the other way around, I added two more concepts to make the simpler motions easy.

The ability to re-define the center of the image on the fly allows flexibility. The image can be rotated around any point, not just the center that was defined when it was drawn. This makes ten motion parameters.

The last group of parameters yields considerable capability for a small additional amount of calculation. These are the auto increments.

With this scheme, it is easy to see that if I wanted to slowly move an image across the screen, I would have to update the move offsets frequently and in small steps, so the image is drawn over and over, each time in a new position. All these move offsets would have to be stored somewhere and could amount to a considerable amount of data. If I have the display processor move things for me, I only have to specify how. The easiest kind of motion to describe is constant motion, where the change in position per time is constant.

This requires that only a speed be specified. So, I have increment parameters for the eight motion parameters. Each time an image is displayed, the increments are added to the respective motion parameters to move them to the next position. If the X move increment is equal to one, the image is drawn one step to the right of the previous position, resulting in a constant speed of one, to the right. This means only one command is needed to start a motion. The image can move across the screen, grow/shrink, spin or fade in/out. With the eight bit pointer into the trig table wrapping itself, rotations do the same thing - one command - go until the speed is set to zero. This is nifty. This brings the number of motion parameters for one image to 18.

Now, producing images is quite a trivial matter, at least that's the way it seems to me. Also, once the display processor is functional all the motion capability is there. However, the choreography (converting desired motions into computer commands) is the killer. Anyone familiar with GRASS?

Eighteen parameters to keep track of for each image does not seem like a big number. After all we have a computer here and that's a drop in the bucket.

Well, for the computer that's correct.

It can keep track of thousands of numbers and find and use them when instructed. The problem here is the Brain-Computer link. By defining commands to change motion parameters and coding a parser to grab them and implement them, we have created a software processor. Hence the name display processor. There are eighteen things we can do with an image at any given time and the first thing one may be inclined to do is make some kind of command line interpreter or assembler. However, this is cumbersome at best. The control of several images traveling in complex movements related to simulate a simple real world situation turns out to be very complex at times. An assembler only relieves a small amount of the burden. A better way would be with a graphical interface and that's still in the future. One issue I've struggled with here is how to specify the motion speeds in a non textual way. As they say, that's a whole 'nuther subject.

So, there you have it. I have two types of systems. The full system is disk based for image and show development and to run my "big" shows. It has the graphics tablet and can manually display any image or any complete show off disks or automatically run anything on disk, just keep puttin' disks in. The other is the 1/2 cu. foot box. Everything's inside - the laser, scanners, CoCo board and power supply. Show is on EEPROM. A couple of these can be put on a bus and cued from the masterto display many internal sequences. I haven't finished that software yet.

I must be honest and admit that I'm still impressed at what I get out of the CoCo (so was my "professional" buddy who got me started in this about ten years ago); but I'm told that I'm easily impressed.

What do I do with them? Well, school dances and parties in the area. My son has a D.J business and he uses one sometimes. I also occasionally give a demo lecture to a school or Boy Scout group.

I haven't talked with any other small system developers, so I can't make any comparisons with what anyone else is doing.

I am always looking for someone interested in getting involved. I'm hoping that there may be someone in the area. I'm doing all the hardware, software, and mechanical design and could use some help.

Basic09 In Easy Steps

Chris Dekker

Advanced Programming: Memory Management

bbs0912.txt

As promised earlier, this time around we will take a look at memory management. Managing your computer's memory is an important part of medium sized and big programs written for the CoCo. The main reason here is that the CoCo's processor can address only 64K at any given moment. This is due to the fact that the 6809 (and 6309) processor have only 16 pins available to form an address with. This gives us $2^{16} = 65536$ possible combinations or addresses. A shorthand notation for this is 64K. Since this is a hardware limitation, there is no way we can work our way around this. Even if you install 2 Megabytes of memory, your CoCo's processor will access only 64K of it.

To get access to every byte in those 2 Megabytes we must do two things. First we must add an extra 5 address lines at the hardware level. The circuits that do this are usually referred to as Dynamic Address Translation hardware or DAT for short.

Secondly we must have a piece of software that divides the computer's memory into blocks and then drives the DAT hardware so that it will switch the correct blocks into the 64K address space available to the processor. Although it may need some patches to expand it's horizons, this piece of software is already part of the OS-9 operating system.

I won't explain in detail how this works, but basically what OS-9 does is the following: It divides the available memory into 8K blocks and then sets up and maintains internal tables which show which blocks are free, which ones are used and to what process they belong. When you want to run a process, OS-9 loads certain values into registers in the DAT hardware that cause the microprocessor to access the correct blocks. When you want to access a different block all OS-9 has to do is copy that block's number to a DAT register and you're all set to access the bytes inside that block.

The main thrust of memory management for Basic09 programs is switching the correct block(s) into and out of the microprocessor's address space. For small programs this is no big deal since they will easily fit into the 64K space. For somewhat bigger programs there may be problems reflected by an error 43 message. The reason for this grey area (may or may not work) is that Basic09 will automatically (try to) switch the code that it needs to complete a task into the processor's 64K address space. If it succeeds you will never know it happened, or you will be

looking at error 43.

In order to understand this better you should know the following: on the CoCo memory is cut into 8K blocks. This means that the processor can access 8 of those blocks simultaneously. But no more than that. If it needs to access a 9th block one of the 8 has to go to make room for the new one. Those 8K blocks can not be divided. Even if you load a small utility like Makdir (34 bytes) into a block; using that utility will cause the entire block to be switched into the processor's address space.

Of course this is a waste of $(8192-34)8158$ bytes. Reducing this waste is the idea behind merging a dozen or so of such small utilities into a single file. This allows you to access all of them "for the price of one".

So how does Basic09 fit into this scheme of things? For starters we have to keep in mind that in order to run Basic09 programs, either Basic09 itself (3 blocks long) or Runb (2 blocks) must be in your 64K address space. Unless you start them up using the ex modifier, a copy of Shell will also be in your address space. This means that you may have used up half of your address space before typing in one byte of code. Add to that the workspace for Basic09 and a copy of gfx2 and syscall and your 64K could be filled up without doing anything useful.

As you can see it is important to try to merge as many programs and utilities as possible to optimize the use of our 8K blocks. This can be thought of as a first step in memory management. Note that when you are merging modules, you can not go 1 or 2 bytes over the 8K (8192/\$2000 bytes) limit or you will waste another 8K block.

Before we dig any deeper into the matter I want to point out a difference between Basic09 and Runb. With Basic09 all program code and data space is located inside Basic09's workspace. Once a program is packed and runs with Runb it behaves like any other OS-9 process: with it's program code and data space separated into individual blocks. This means that you can effectively manage your memory by manipulating your program code, it's data space (buffers, etc.) or both.

Let's first take a look at managing program code. Let's assume that you know up front that you are working on a large project. Sometimes a program starts small, but after continually adding code it gets so big that it has to be split up. This isn't easy and it is almost impossible to give hints on how to

that in an article like this. If you think there is a chance that will happen: assume the worst and split up the code before you start writing it.

In some instances there is an easy way out for managing your program code. For instance for accounting programs you can split the code into totally separate programs: one for data entry/editing, one for making printouts, one for compiling cashflow statements, etc. You can then run these programs one after another or simultaneously in different shells.

Unfortunately, this easy way out is not always available. Programs like CoCoTop and the canvas program from level II graphics are much too big to fit into a 64K workspace, but still must appear as a single program to the user.

Such code isn't all that hard to write, but it takes some ingenuity to debug. Usually one ends up debugging the subroutine modules as stand alone programs. You can then link them to the main program when they are up and running (by replacing some DIM statements by PARAM, etc.).

Before I tell you how such programs are structured, let's see how much space we have to work with. You must write these programs from the point of view that they are running under Runb. Assembling and running the source code with Basic09 is only a temporary measure. To make the best use of our 64K space we can merge Shell, Runb, gfx2 and syscall into one file. This leaves us with room for a few small utilities (like link and load which makes booting up easier) and still stay within two 8K blocks. Of course your program takes at least 1 block, which accounts for three of the eight available blocks.

How the other five are used depends partly on how you start your program from the OS9: prompt. Let's say we have a program called cashflow that is 8000 bytes long. There are several ways to start this program. For instance:

```
cashflow <ENTER>
ex runb cashflow <ENTER>
runb #24k cashflow <ENTER>
```

All three possibilities will start the program, but memory requirements are different. All three cases use the three blocks I mentioned above. On top of that the first case requires three more blocks: one for the shell that launches the program and two (16K) is the default data space for Runb.

The second case requires two additional blocks because the "ex" kills the shell that

starts the program. Note that in this case the program cashflow must exit by starting a new shell or you may end up looking at a blank screen with nowhere to go but the reset button. After winding down everything else you can end your program with the following code:

```
SHELL "shell"  
BYE
```

The 3rd case requires four extra blocks, for a total of seven, because it also reserves extra data space. Of course modifying the allotted data space is only useful if your program can use it or get by without it.

Now that we have exhausted all possibilities outside modifying the actual program code we will take a look at that too. To make full use of the CoCo's address space we must (or at least try to) split the program into 8K chunks (of packed code). We can do this by writing modules that are (almost) 8K long or by merging modules into various 8K blocks.

The easiest way to achieve this is to merge the packed modules into files that are as close to 8K in length as possible. Your main program should then include some code to load these files. Since OS9 loads each file into a separate block we are all set to take advantage of the extra memory.

As I mentioned earlier Basic09 will try to switch whatever module it needs into our address space. So the only extra code we have to write is the code to get rid of unwanted blocks. This has been made easy for us through the KILL command. If you KILL a module Basic09 not only unlinks it, it will also switch the block which holds the module out of your address space.

To run and unlink a module called "edit" your code can look like this:

```
RUN edit(parameters)  
KILL "edit"  
REM check for errors that edit may return
```

Although this code will work, there is a chance that it will get you into trouble. If your main program calls "edit" in this fashion a number of times, you may end up staring at an error 43 message. For no obvious reason I might add. I mostly use a slightly modified version of this code that doesn't report the problem.

```
DIM module:string[10]  
module="edit"  
RUN module(parameters)  
KILL module
```

Your "main program" becomes a very simple affair in these setups. It basically holds the code to set up and initialize common variables and to run the various subroutines. It can also do error handling and file handling, so you won't have to duplicate that code in every subroutine. This module usually also holds the code for a main menu. All these functions make it the central hub of your code: each subroutine module that exits comes back to this module.

Since this module usually comes nowhere near filling the 8K block you can merge a number of small, often used, subroutines with it in the same block. These can be routines for keyboard input, menu handling, or ML subroutines that perform a specific task. An added advantage to doing so, is that your subroutines don't have to KILL these modules every time they call them.

Along the same lines: if a subroutine module happens to RUN another subroutine that does not reside in the same block as the "main program" it must issue a KILL command for that module too. If it doesn't, your address space will gradually fill up and you will be looking at an error 43.

There is one other sticking point that may throw off your grand scheme. OS9 loads every module only once. If you have two programs that use the same module, that module will be loaded along with the first program. If you load the second program later on and run it, this program will have to switch a block of program 1 into its address space before it will run. If your address space is already filled, you get to see error 43 once more. In this case you must UNLINK both programs and, once unlink reports an error #221, reload program 2 and run it again.

And then there is managing your data space. Sometimes even a small program needs a lot of dataspace. A prime example would be dupdisk, which backs up floppy disks in one pass. For a DS 40-track disk this amounts to 360K, which is of course much bigger than our addressing space.

So once again we must switch blocks of memory. But this time it isn't as easy as using RUN and KILL statements. This time we must content with system calls (and a bug or two).

Before we dig into the code I will give you some examples of what you can do.

I already mentioned dupdisk. This program uses two different types of memory management. First it finds out how large the disk is that it has to copy. Then it tries to reserve that much space for itself. If successful, it copies the disk into that dataspace block for 8K block. To copy the contents of the data space to another disk the whole process gets reversed.

Another process is used by homepac's mailer. This program reads its datafile into graphics buffers, 60 records per buffer. Although this may look strange, it allows the program to let OS9 do all the dirty work such as (de)allocating buffers, keeping track of them, etc. All the program has to do is convert record numbers into buffer numbers.

A third way of managing your data is one that I used for Homepac's check program. This involves no system calls or block switching, but requires some extra Basic09 code. In effect this program extends its buffer into the datafile on the diskdrive. I suppose one could call it a virtual buffer. Using this technique I could size the buffer so it will fit within Runb's default data space, yet the program can access very large quantities of data.

Of course there are a few more ways of dealing with data: Quickletter for instance, has an 18000 byte main buffer and that's all you get to work with. However, since few of us regularly write documents larger than that, this is not really a big problem.

At the other end of the scale you could tell your program to use a diskfile as its buffer and process that one record at a time. This gives you vast amounts of space, but questionable performance because of disk access times (especially for floppy drives).

I wanted to show you some programming examples here too, but the article has become too large. So that will have to wait until next time.

Chris can be reached in care of
this magazine or directly at:

Chris Dekker
RR #4

Centreville, NB E05 1H0
CANADA

micro notes

(Almost) FREE GENie signup!

To sign up, just follow these simple steps:

1. Set your communications software for half duplex (local echo), at 300, 1200, 2400 baud.
2. Dial toll free: 1-800-638-8369, or in Canada, 1-800-387-8330. Upon connection, enter HHH
3. At the U# prompt, enter JOINGENIE then press <RETURN>
4. When asked to enter a code enter MSC524. This will give you a \$50 credit toward your first month's usage.
5. Have a major credit card ready. In the U.S. you may also use your checking account number. For additional information call 1-800-638-9636 or E-mail TANDYS on GENie or hogan@genie.geis.com on Internet.

FOR SALE: 512K CoCo 3, CM-8 RGB Monitor, FD-501 double sided drives (two!), DMP-105 printer with stand, 300 baud modem, two joysticks, hi-re interface, and mouse - all in excellent condition with dust covers. Rainbow Magazine from 2/87 through 1/90 with some Rainbow on Disk. Many ROM Paks and approximately 50 disks full of programs. All for \$275! Please call 9-5 EST 1-800-283-5412, after 6pm EST 508-833-0765. Ask for Ed. or write: Ed Eslzari
28 Cromwell Road
Sandwich, MA 02563

EFFO (EUROPEAN FORUM FOR OS-9)

EFFO stands for European Forum For OS-9 and was founded in 1988. Its main goal is to support Microware's OS-/68K. Support primarily consists of providing communication between users and those who do not yet but would profit by doing so.

EFFO is independent from and not commercially related to any company. Members are companies offering OS-9 compatible hardware, system programmers, computer clubs and end users such as private computer owners, research institutes and university departments.

EFFO provides a collection of public domain software that is of general interest and that helps to make OS-9 more attractive to programmers and users. It also coordinates ports of (mainly UNIX) software (e.g. to avoid redundant work) and makes all the nuts and bolts of managing an operating system available to everybody so that "the wheel isn't re-invented" all the time.

The EFFO software pool disks contain ready-to-use software that has been thoroughly tested. The software is maintained and updated continuously. All disks come with printed

installation and use documentation. 23 disks (up to 1992) also have OS-9/6809 software. EFFO also has a printed forum: the journal 'OS-9 International', devoted to OS-9 related topics. Every issue includes the most recent version of EFFO's public domain software list. This list will also be made available on the EFFO bulletin board and through regular postings to international network boards.

For more info, contact EFFO at:
EFFO
P.O. Box
CH-8606 Greifensee
Switzerland
FAX +41 1 940 38 90
e-mail: effo@effo.ch

This is also the address where the editorial staff of 'OS-9 International' and all active EFFO members can be reached in case of questions concerning particular articles or software packages.

Last but not least we invite you to join EFFO. As a regular member you get some price reduction on our PD disks and a one-year subscription to 'OS-9 International'. Contact us for US pricing.



*Software, Books, and Hardware for
all OS-9/OSK Systems!*
ADD \$2.50 S&H
(\$4.00 Canada, \$10.00 Overseas)

Box 321
Warner Robins, GA 31099
Phone 912-328-7859
Internet: dsrtfox@delphi.com

CoCo DECB Software:

CoCo Family Recorder - \$17.50

Genealogy program for CoCo 3. Requires 2 drives, 80 col. monitor.

NEW! OS-9 Version - \$32.50

DigiTech Pro - \$12.50

Record any sound for easy play-back in your BASIC or M/L programs. CoCo 3 512K.

ADOS: Support for double sided drives, 40/80 tracks, faster formatting, much more!

Original (CoCo 1/2) - \$15.00

ADOS 3 (CoCo 3) - \$25.00

Extended ADOS 3 - \$30.00 (ADOS 3 req., RAM drives, support for 512K-2MB)

ADOS 3/Ext. Combo - \$50.00

CoCo OS-9 Software:

Patch OS-9 - \$7.50

Automated program installs most popular/needed patches for OS-9 Level II. 512K and two 40T/DS (or larger) drives required. (128K/35T users can install manually- state 35T.)

Pixel Blaster - \$12.50

High speed graphics tools for OS-9 Level II. Easily speed up your game programming with this tool kit and subroutines!

OS-9 Point of Sale - \$62.50

Maintain inventory, print invoices, customer catalog, etc. Multi-user capable. Supports ASCII terminals. Menu driven. CoCo3.

Books:

NEW!! Mastering OS-9 - \$30.00

This is the long awaited update of Paul Ward's "Start OS-9". New format is easier to read, has easier to follow tutorials, and updated information files. Comes complete with disk, which has a few added utilities.

Tandy's Little Wonder - \$22.50

History, technical info, schematics, peripherals, upgrades, modifications, repairs, much more- all described in detail for all CoCo

CoCo Hardware:

DigiScan Video Digitizer - \$150: Capture images from VCR, camcorder, or TV camera. No MPI required- uses joystick ports. CoCo Max3, Max 10, Color Max 3 compatible. Special order- allow 90 days for delivery. Send \$75 deposit, remainder due before delivery.

Puppo Keyboard Adapters - \$70: Use IBM PC/XT keyboards with your CoCo! Mounts in your CoCo case with no soldering. 101 key keyboards available for \$30 with order.

FARNA Systems Publishing Services

Type Setting and Printing: We can prepare professional typeset manuals, books, booklets, catalogs, and sales flyers for you - we can print or you reproduce as needed from a master set! Very reasonable prices - inquire!

Mailing Service: If you send catalogs or letter correspondence to 200 or more persons at once, we can do all work for you for about the same cost of your materials alone! How much is your time worth???

*Contact Frank Swygert at
FARNA Systems for quotes*

for all your CoCo hardware needs, connect with

CoNect

449 South 90th Street
Milwaukee, WI 53214
E-mail: rickuland @delphi.com

The main problem with OS9 under a CoCo is the serial port. With a one character buffer, it's hard to do much before the serial port needs service. Windows and OS2 have the same problem. Or did, until National released the 16550 uart- 16 bytes of internal fifo buffering gives multitasking systems time to do some.

Announcing Fast232

Tandywith Sacia

bps load ___ thruput

2400 28.3 sec 237 cps

9600 73.6 sec 938 cps

57600 not available

Fast232

load ___ thruput

25.3 sec 235 cps

31.4 sec 950 cps

32.6 sec 5373cps

Local machines with faster modems can now be connected to properly (or improperly at 115K!). More down to earth, Fast232 is a ROMPak sized case, which will accept a daughterboard (once I get the case to close) to give two ports in a very 'concise' package. OS9 drivers by Randy Wilson. Free bonus software! The pd release of 'SuperComm' (Dave Phillipsen and Randy Wilson). All software includes 6809 and 6309 versions.

Fast232	\$79.95
Second port	\$45.00

NEW FOR 1995 FROM DISTO!

1. "Inside 2-Meg": A technical booklet that fully describes how the DISTO 2-Meg Upgrade kit works. Includes schematic, PAL listing, theory and chip by chip circuit explanations. \$20 + \$2.50 S/H.

2. "Blank Board Kit": Includes blank virgin boards (no components) of the SCII, SCI, 4IN1, MEBII, MPR0M and Mini Controller. Collect all the components and make your own! \$29.90 + \$4.50 S/H.

3. Call for other DISTO products in stock
(limited quantities available)

DISTO

1710 DePatie

St. Laurent, QC H4L 4A8

CANADA

Phone 418-747-4851

The 6th Annual Atlanta CoCoFest

September 30 & October 1, 1995
Northlake Holiday Inn, Atlanta, Georgia

Show Hours:

Sat., Sept 30: 9:00AM - 5:00PM

Sun., Oct 1: 9:00AM - #:00PM

Admission:

\$10.00 for both days (no one day passes)

Reservations:

Northlake Holiday Inn

1-800-465-4329 or 404-938-1026

Sponsored by:

Atlanta Computer Society

PO Box 80694

Atlanta, GA 30366

BBS: 404-636-2991

Wittman Computer Products

873 Johnson Road

Churchville, NY 14428

Phone 716-494-1506 : Fax 716-293-1207

Internet: ww2150@acsprl.acs.brockport.edu

K-Windows Chess for MM/1

Play chess on your MM/1.....\$24.95

X-10 Master Control for MM/1

Use MM/1 to control you home!.....\$29.95

Variations of Solitaire

Pyramid, Klondike, Spider, Poker and Canfield

MM/1.....\$29.95 CoCo3.....\$19.95

OS-9 Game Pack

Othello, Yahtzee, KnightsBridge, Minefield,
and Battleship

MM/1.....\$29.95 CoCo3.....\$19.95

WPSHell

An OS-9 Word Processing Point and Click Interface

CoCo3.....\$14.95

Using AWK with OS-9

Includes V2.1.14 of GNU AWK for OS-9/68000

MM/1.....\$14.95

NEW! WCP306 Computer!

MC68306 16.67MHz CPU, code compatible with MC68000.

Four SIMM sockets, 512K-16MB memory. IDE hard disk interface, floppy interface, two serial ports, parallel port, real-time clock, all built into motherboard! Designed to use 16 bit AT expansion cards (six slots) and standard AT keyboard, power supply, and case. Comes with "Personal OS-9/68000" and MGR, a graphical user interface with complete documentation. Only \$400 for bare board as described above!

Call or write for a free catalog! Demo disks also available.

Owned and operated by William L. Wittman, Jr.

CoCoTop version 1.0	\$24.95
CoCoTop version 1.1	\$19.95
CoCoTop 1.1 + Tools 3	\$34.95
OScopy/RScopy	\$10.00
TOOLS 3 version 1.1	\$29.95
Quickletter version 2.0	\$19.95
Accounting level 2	\$34.95
Investing level 2	\$24.95
Level II graphics 1.2	\$34.95

upgrades only \$5.00 (return original disk)

Shipping+handling: US/Canada \$3.00 all others \$5. Prices in US dollars Send cheque or money order NO COD'S. Call or write for Canadian dollar prices. Mention the name of this magazine in your order and you will receive a free bonus disk!

C. Dekker ... User-friendly Level II Programs!

RR #4 Centreville, NB

E0J 1H0, CANADA

Phone 506-276-4841



EDTASM6309 Version 2.02 -----

\$35.00

This is a major patch to Tandy's Disk EDTASM to support Hitachi 6309 codes. Supports all CoCo models. CoCo 3 version uses 80 column screen, 2MHz. YOU MUST ALREADY OWN TANDY'S DISK EDTASM TO MAKE USE OF THIS PRODUCT. It WILL NOT work with a disk patched cartridge EDTASM.

CC3FAX -----

\$35.00

Extensive modification to WEFAX (Rainbow, 1985) for 512K CoCo 3. Uses hi-res graphics, holds full 15 min. weather fax image in memory. Large selection of printer drivers. Requires shortwave receiver and cassette cable (described in documentation)

HRSDOS -----

\$25.00

Move programs and data between DECB and OS-9 disks. Supports RGB-DOS for split DECB/OS-9 hard drives. No modifications to system modules (CC3Disk or HDisk) required.

DECB SmartWatch Drivers -----

\$20.00

Access your SmartWatch from DECB! New function added to access date/time from BASIC (DATES). Only \$15.00 with any other purchase!

Robert Gault

832 N. Renaud

Grosse Pointe Woods, MI 48236

313-881-0335

Add \$4 shipping & handling per order

Mid Iowa and Country CoCo Club

(non-profit)

If you want support, we're here for you! MI&CCC publishes the UPGRADE disk magazine, now in its tenth year, five as a national publication. We've grown to be one of the largest CoCo outreaches in the world! We have subscribers in over 40 states and five provinces of Canada, as well as in Australia and England.

Your MI&CCC membership brings you:

1. A year's subscription to the UPGRADE disk magazine (requires CoCo 3 and one disk drive), 8-10 issues per year. This is a news magazine, not a software disk.
2. Access to our shareware/public domain/orphanware library - we keep only the best!
3. Optional Christian sub-chapter that gathers Christian oriented software for those interested.
4. ROM burning and other support.

Say you saw this ad in "68 micros" and receive a bonus disk along with your new membership!

Mid Iowa & Country CoCo Club

Terry Simons, Treas./Editor

1328 48th, Des Moines, IA 50311

Please include your phone number and system information

The OS-9 User's Group, Inc.

Working to support OS-9 Users

Membership includes the Users Group newsletter, MOTD, with regular columns from the President, News and Rumors, and "Straight from the Horse's Mouth", about the use of OS-9 in Industrial, Scientific and Educational institutions.

Annual Membership Dues:

United States and Canada	Other Countries
25.00 US	30.00 US

The OS-9 Users Group, Inc.
6158 W. 63d St. Suite 109
Chicago, IL 60638
USA

Northern Xposure Quality Products from North of the Border

OS-9 Level II Color Computer 3 Software	
NitrOS-9 v1.20 Call or write for upgrade info or new purchase procedure. Requires Hitachi 6309 CPU	\$29.95
Shanghai:OS-9 Introductory price	\$25.00
Send manual or RomPak to prove ownership	
Thexder:OS-9 Send manual or RomPak to prove ownership	\$29.95
Smash! Breakout-style arcade game	\$29.95
Rusty Launch DECB/ECB programs from OS-9!	\$20.00
Matt Thompsons SCSI System v2.2 'It flies!'	\$25.00
256/512 byte sectors, multipak support	

Disk Basic Software	
Color Schematic Designer v3.0 New lower price	\$30.00
Oblique Triad Software	Write for catalogue

Color Computer 3 Hardware	
Hitachi 6309 CPU (normally 'C' model, may be 'B')	\$15.00
SIMM Memory Upgrade Runs Cooler! 512k \$44.95 0K \$39.95	
Sound Digitizing cable	\$15.00

OS-9/68000 Software	
OSTerm 68K v2.2 External transfer protocol support	\$50.00
TTY/ANSI/VT100/K-Windows/Binary Emulation	
Upgrade from TasCOM (Send TasCOM manual please)	\$30.00

7 Greenboro Cres
Ottawa, ON K1T 1W6
CANADA
(613)736-0329
Internet mail: cmckay@northx.isis.org

*All prices in U.S. funds.
Check or MO only.
Prices include S&H*

Don't have a subscription to "microdisk"? Don't want to pay the high price for back issues? You can now get the complete Volume 1 of microdisk for \$30 (plus \$2.50 S&H)! That's an \$18 savings over back issues and a \$10 savings over the subscription price! Just write and tell us you want the entire volume 1.

All files will be on as few disks as possible, not separate disks for each issue. "microdisk" is not a stand-alone product, but a companion to this magazine. Subscriptions are \$40 per year. Single issues are \$6 each in US, \$45/\$6.50 in Canada. Overseas add \$10 per year, \$1 each for airmail.

ADVERTISER'S INDEX:

<i>Atlanta CoCoFest</i>	23
<i>BlackHawk Enterprises</i>	15
<i>C. Dekker</i>	24
<i>CoNect</i>	23
<i>Delmar Company</i>	BC
<i>DISTO</i>	23
<i>FARNA Systems</i>	9,22,25
<i>HawkSoft</i>	15
<i>Mid Iowa CoCo Club</i>	24
<i>Northern Xposure</i>	25
<i>OS-9 User's Group</i>	25
<i>Robert Gault</i>	24
<i>Small GrafX Etc.</i>	15
<i>Wittman Computer Products</i>	24

**Don't have a subscription yet?
WHAT ARE YOU WAITING FOR?!**
Subscribe today!
(Details inside front cover)

For superior OS-9 performance, the
SYSTEM V

Provides a 68020 running at 25 MHz, up to 128 MBytes of 0 wait-state memory, SCSI and IDE interfaces, 4 serial and 2 parallel ports, 5 16-bit and 2 8-bit ISA slots and much more. The **SYSTEM V** builds on the design concepts proven in the **SYSTEM IV** providing maximum flexibility and inexpensive expandability. Optionally available at 33 MHz.

AN OS-9 FIRST - the MICROPROCESSOR is mounted on a daughter board which plugs onto the motherboard. This will permit inexpensive upgrades in the future when even greater performance is required.

G-WINDOWS benchmark performance index of a **SYSTEM V** running at 25 MHz with a standard VGA board is 0.15 seconds faster than a 68030 running at 30 MHz with an ACRTE video board (85.90 seconds vs 86.05 seconds).

For less demanding requirements, the
SYSTEM IV

The perfect, low cost, high-quality and high performance OS-9 computer serving customers world-wide. Designed for and accepted by industry. Ideal low-cost work-station, development platform or just plain fun machine. Powerful, flexible and expandable inexpensively. Uses a 68000 microprocessor running at 16 MHz.

G-WINDOWS - a PROVEN WINNER

FOR OS-9

G-WINDOWS is now
AVAILABLE for OS-9000 from
delmar co

Available for the **SYSTEM IV** and **SYSTEM V** computers, the PT68K4 board from Peripheral Technology and the CD68X20 board from Computer Design Services. Resolutions from 640 x 480 x 256 to 1024 x 768 x 256. Support for multiple VGA cards running different processes, different portions of the same process or both.

Support for generic VGA boards and **SUPER VGA** boards including ET4000 (Tseng Labs), OTI067 (Oak), CT452 and CT453 (Chips and Technology), GENOA, WD90C11 (Paradise) and S3 (S3 Inc.) for modes from 640 x 480 to 1280 x 1024 depending on board.

Distributor of MICROWARE SYSTEMS CORPORATION Software

This ad was prepared and printed using QuickEd under OS-9.

delmar co

PO Box 78 - 5238 Summit Bridge Road - Middletown, DE 19709
302-378-2555 FAX 302-378-2556