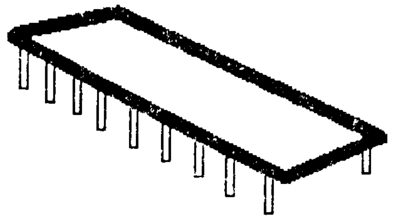
February 1995 *Vol. 2 Number 5* *$4.50 Canada, $4.00 US*

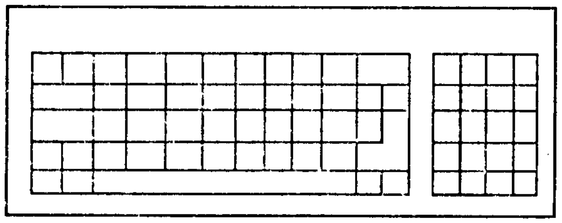
the world of 68' micros

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

You may soon be able to use one of these with your CoCo... *through the bit-banger!*



Auto-boot OS-9 without losing DECB!



Puppo Keyboard Adapters available again soon!!!

(See "micro notes" for details)

CONTENTS

NOTICE: AS OF 01 JANUARY 1995, SUBSCRIPTION RATES WILL BE:
US: $25/year, $13/ 6 mths
Overseas: $45/year, $23/ 6 mths (air)
$37/ year, $18/ 6 mths (surface)
APO/FPO: $30/ year, $16/ 6 mths
(1st class US mail)
Canada/Mexico: $32/year, $17/ 6 mths
(no change for microdisk)

POSTMASTER:
If undeliverable return to:
FARNA *Systems* PB
Box 321
Warner Robins, GA 31099

Address Correction Requested

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. microdisk contains programs and source listings from each 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.

The publisher welcomes any and all contributions. 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 recevied a lot of little notes with renewals.. I thank you all for those... but no real letters of general interest except the one printed to the right. Do let me know what's going on out there!! I need your input so I'll know what to do in the future, and if anything needs correcting.

I've made a few changes. I decided we needed a slightly different look on the cover, so changed the title. What do you think? I also changed the "end of story" marker a bit. I may want to change the cover again next year. If you have any ideas, sketch them out and send them in! The only restrictions are that I need the info I have on the front, including the table ofcontents. If I move the contents inside, we lose 1/3 of a page. Of course, that might not be so bad... what do you think?

I do have some bad news about the programming contest. I did get some great entries, but not enough. I'm not backing out of this entirely, but I can't justify the prizes I had announced either. I only received between six and eight entries... I had expected at least a dozen! I am going to be sending four general prizes to those who entered... I haven't decided exactly what they will be yet.

Until next time, keep those CoCos and OS-9 (68K) machines running, and welcome to the first issue of 1995!!

[illegible]

Letters to the Editor

According to the last issue of "UpTime" I should have two issues of "68' micros" added to my current subscription. I am renewing at this time for another year anyway.

I have a question about the "Darts" program in the May 1994 issue. I finally got around to typing it in and got a syntax error in line 20. After retyping several times, I tried deleting the line and got a function error and a program lock-up. I am not a programmer so can you tell me what I am doing wrong or is there an error in the program?

Laura Boyce
Box 5699
Phoenix, AZ 85010

Laura, I'm not printing this to embarass you, please don't take it that way! I remember how it was as a beginner also.

There is no error in the program that I am aware of. Please send me a printout or hand written listing of at least the offending line (line 20) and four lines before and after this. The entire listing as you typed it would be best. I have no idea exactly how you typed the program, and it could be an error in a line other than 20.

I do see one thing I did that may be confusing you. Since it is hard to see 38 actual asterics and 9 actual spaces, I typed the phrases in parentheses. Could you be typing those phrases instead of what they say? Line 20 should be:

```
20 HSCREEN2: HCLS10:A$"*******
*******************************": B$="
": HPRINT(2,0),B$: NEXT: HPRINT
(2,23), A$: FOR X=1TO5: SOUND
RND (25,5),1: NEXT
```

The fault is at least as much mine as yours... I had not found the best method to print program listings at the time "Darts" was printed. As you can see by the above line, it is hard to determine how many asterics and spaces are used though. I do hope this solves the problem!

< 268'm >

The Seven-line demo: an amazing achievment with DECB!

Program by John Kowalski, comments by Art Flexser and Rick Adams

```
2 REM ** 2WAYSCRL.BAS
4 REM **by SockMaster
10 POKE65497,0 : HSCREEN2 : FOR
G=0 TO 15 : READ A : PALETTE G, A
: NEXT
20 DATA0,19,22,50,54,52,38,37,44,
45,41,13,11,25,27,26
30 FOR G=0 TO 319 STEP .5 : HSET
(G, RND(191), RND(15)) : NEXT
40 C=1 : S=40 : FOR G=15 TO 1
STEP -1 : HCOLORG
50 HCIRCLE (310-G*5,48), S : HPAINT
(310-G*5,48) : S= S-2.6 : HLINE (RND
(110)+210, RND(80)+104)-(RND (110)
+210,RND(80)+ 104), PSET, BF:NEXT
60 Q=65439 : W=0 : E=127
70 FORG=0TO127 : PALETTE W,W :
POKE Q,G:::::::::::     POKE Q, E-G :
NEXT : GOTO 70
```

NOTE: There should be eleven (11) colons (:) and five (5) spaces after "POKE Q,G" *in line 70.*

This little 7-line BASIC program (ignoring the first two REM lines) does a neat trick: it scrolls the top half of the graphics screen leftward at the same time as the lower half of the screen scrolls rightward. The program was recently uploaded to the CoCo interest group on Internet. It is attributed to John Kowalski of Montreal, also known as Sock Master, who has also recently released a shareware terminal program called Twilight Term.

The most surprising thing about this program is that it is entirely in BASIC (the data values in the second line are palette color values, not machine-language code). Most experienced programmers would assume that this sort of two-way scrolling, if it were possible to do at all, would absolutely require a machine-language routine. But Sock Master manages it in BASIC, using some very clever programming.

Here's a line-by-line breakdown of what the program does:

10-20: set up the palette colors.

30-50: draw some things on the graphics screen. (Not much point in scrolling a blank screen, right?) Specifically, line 30 draws a background of randomly colored and randomly positioned dots. Then, lines 40 and 50 draw some filled circles and boxes in various colors.

60: assigns values to some constants.

70: where the fun begins.

Line 70 LOOKS really strange. What are all those colons and spaces for? And why is there a palette statement that assigns color 0 to palette 0 over and over again, when once should seemingly be sufficient? And what are those pokes to location 65439 doing?

This is what is going on: Location 65439 ($FF9F) is the GIME's horizontal scroll register. As the value poked into it increases from 0 to 127, the screen scrolls more and more to the left, with portions that disappear on the left side reappearing on the right. That explains what happens in the top half of the screen. A rightward scroll would result from decreasing the value in the horizontal scroll register from 127 to 0. But, how in the world can the bottom of the screen be made to scroll rightward at the same time the top half is scrolling to the left? Here is where Sock Master displays some real ingenuity.

What is necessary is to time the two pokes in line 70 very carefully, so that the first one, which scrolls the screen one notch to the left of its previous position, occurs at the beginning of each 60th-of-a-second redrawing of the screen, while the second one, which scrolls the screen one notch to the right, occurs just as the bottom half of the screen starts to get drawn, approximately 1/120th of a second later. Here is where the PALETTE W,W statement comes in. Its purpose is not to assign a color at all, but to take advantage of an incidental feature of the PALETTE statement.

In the ROM code for the PALETTE command, there is a machine-language SYNC instruction that tells the processor to wait for an interrupt that indicates a new frame of the screen is about to be drawn. This SYNC enables the pokes in line 70 to be synchronized with the redrawing of the screen each 60th of a second.

Try removing the PALETTE command from line 70. You will see that the screen now jiggles all over the place due to the lack of sychronization with the two pokes.

It should now be clear that the purpose of all those odd-looking colons and spaces in the middle of line 70 is to adjust the delay so that the second poke occurs just as the bottom half of the screen starts to get drawn. If you add more colons, the dividing point of the screen will be lower than the halfway point; deleting some of the colons will move the dividing point upward. A FOR/NEXT loop can be used instead of the colons and spaces, but does not allow the same degree of fine adjustment of the dividing point as using colons and spaces does. I tried substituting FOR I=1 to 2:NEXT and got a dividing point that was about a quarter of an inch too high, while changing the upper limit to 3 made the dividing point about the same amount too low.

A final point is that this technique of changing the value of a graphics register after part of the screen has been drawn can also be used with the palette registers, so as to allow different sets of colors to be used in (say) the top half and bottom halves of the screen.

There have been a few demonstration programs that display all 64 colors that the GIME can generate on a single screen, even though the graphics mode employed is supposed to allow only 16 colors; these work by switching palette contents 4 times per frame in synchrony with the display timing. Hmmm, I wonder if you could do THAT with a BASIC-only program?

Art Flexser

The combination of colons and spaces to provide a small delay is very clever... the space gives a smaller delay than the colon. It takes a very small amount of time to scan the space, whereas with the colon, the ROM code in the BASIC interpreter does the overhead involved in setting up for scanning a new BASIC command, which takes a bit longer. So colons are used for coarse adjustment of the timing, and spaces are used for fine adjustment.

Rick Adams

< 268'm >

57,600 Bits Per Second... *through the bit-banger!*

The start of a new terminal program... we hope! *Earl W. Casper*

I'm developing a 57.6K baud DECB terminal package. When the editor heard about it, he said maybe some of you might want to help. I've got a working version that only runs at 57.6K baud. To send a control character you press CTRL and then the character. To send the password you press F2 and it appears. If you press ENTER then the password is sent. If you press anything else then you just return to the terminal. The primitive autodailer and password program are written in BASIC so someone should be able to improve on it pretty easily. Hopefully someone will add the lower baud rates with parity and such. Of couse what the system really needs is x-modem, y-moden, and z-modem.

There are two jobs that I plan to tackle. At present, interrupts are disabled while a byte is being sent. At 57.6K baud it only takes .0002 seconds to send a byte, but if a byte starts coming in while a byte is going out you get a bad character on the screen. I make so many mistakes typing that I watch the screen for my mistakes anyway. I can't tell which ones are mine and which ones are from the program. Anyway, I plan to fix that problem like Kottke did in his bitbang program in an earlier article.

Right now I'm using Robert Gault's patches to disk EDTASM because I never got around to putting macros into my Quick Assembler that assembles and loads a typical 3000 line program in 10 seconds, and macros are really handy when you are writing repetitive code. I'm going to use my Quick Assembler to add high resolution characters like V-Term uses, because of its excellent graphics debugging capabilities. I'm not sure, but I think I can get the same good kind of results with a two color high res screen as I get now with a hardware character screen. Sixteen colors will be spectacular, and I hope it will only be a little slower and a little harder to read.

There are two tricks that I use to make this program work so well. Since the CoCo uses a clock rate of 2 NTSC ticks, there are 3,579,545/57,600/2=31 CoCo memory cycles between bits. Sending characters is quite straightforward. The problem in receiving is taking care of the byte and getting ready for the next one in time. I adjust the pointer and store it in the address field of an STA instruction while the byte is coming in. Then all I have to do after the last bit comes in is load the byte with an LDA instruction, store it with that STA instruction we were talking about, acknowledge the interrupt with an LDA instruction, restore the A register with another LDA instruction, and return from the interrupt with an RTI instruction. I think I could squeeze another ten memory cycles out of there, but it works fine already, so why do it.

The other trick is in displaying the characters. I map the high res buffer into the 64K space, and then copy the characters from the input buffer to the high res buffer in a tight loop using the B register to keep track of the horizontal position on the screen. When I get to a new line I move the high res buffer down 80 spaces instead of scrolling the screen up 80 spaces. When I'm storing characters in the high res buffer I also store it in the line just above the high res buffer. That way I have a copy of the whole screen above the high res buffer after a while, so I can move the high res buffer up one screen when the new line comes and start all over. Hmm, maybe that was more than one trick. Anyway, the EDTASM source will be on the micro-disk if you are interested.

Hopefully I'll be submitting more articles as others and I complete more of the program. The program works fine as it is, but it will be getting better and better if everyone helps out.

```
576DIAL.BAS:
10 '57,600 BY EARL CASPER
20 'COPYRIGHT 1994 BY EARL
CASPER
30 '
40 WIDTH 80
50 CLEAR 200,&H6000
60 LOADM"57,600"
70 PRINT"1. DELPHI"
80 PRINT"2. COCONUT"
90 PRINT"3. TERMINAL"
100 AD$="AT\Q0 DT8925880"
110 PW$="password2"
120 K$=INKEY$
130 IF K$="" THEN 120
140 IF K$="3"THEN 190
150 IF K$="2"THEN 190
160 IF K$<>"1"THEN END
170 AD$="AT\Q0 DT2584528"
180 PW$="password1"
190 EXEC&H6000'Initialize
200 DEFUSR1=&H6003 'Send a
string to the modem
210 DEFUSR2=&H6009 'Send a
string to the terminal screen
220 IF K$="3"THEN 240
230 T$=USR1(AD$+CHR$(13))
240 EXEC&H6006 'Enter the
terminal program
250 EXEC&H600C 'Change to
BASIC screen
260 PRINT PW$;
270 K$=INKEY$
280 IF K$=""THEN 270
290 IF K$="Q" OR K$="q" THEN
END
300 PRINT STRING$(LEN(PW$),
CHR$(8));
310 IF K$<>CHR$(13)THEN 240
320 T$=USR2("Password sent.")
330 T$=USR1(PW$+CHR$(13))
340 GOTO 240
```

EDITOR: Earl has done all the groundwork here for a high-speed DECB terminal program. Now a little help is needed from you! Do you think you might have some ideas to improve this program? Maybe you have a terminal program started and could use the high-speed portion as an addition? Please let Earl (and myself) know!

Write to:
Earl W. Casper
P.O. Box 60576
Phoenix, AZ 85082

(continued on next page)

Assembly listing for 57,600 bps driver:

```
        TITLE   57,600 BY EARL CASPER

*       COPYRIGHT 1994 BY EARL CASPER

        ORG     $6000

DEBOUN  EQU     $86C

BITIN   MACRO
        BRN     *               3 3
        LBRN    0               3 8
        ROR     $FF22           7 15
        ROR     TEMPIN          7 22
        ENDM

        LBRA    INIT
        BRA     STROUT
        NOP
        LBRA    TERM
        BRA     STRIN
        NOP
        BRA     BASSCR
        NOP
        RMB     1
IFRONT  RMB     2
IREAR   RMB     2

STROUT  LDB     ,X
        BEQ     STREX
        LDX     2,X
STROLP  LDA     ,X+
        JSR     CHROUT
        CLRA
STRWAI  DECA
        BNE     STRWAI
        DECB
        BNE     STROLP
STREX   RTS

STRIN   LDB     ,X
        BEQ     STREX
        LDX     2,X
        LDU     IREAR
        LEAU    1,U
STRILP  LDA     ,X+
        STA     ,U+
        DECB
        BNE     STRILP
        LEAU    -1,U
        STU     IREAR
        RTS

BASSCR  LDY     #$10/8*$6C00  #$6C000/8
        STY     $FF9D
        RTS

DSPMMU  RMB     1
DSPTSK  RMB     1

FIRDSP  LDA     $FFA1
        STA     DSPMMU
        LDA     $FFA9
        STA     DSPTSK
        LDA     #$6600/$200  #$66000/$2000
        STA     $FFA1
        STA     $FFA9

        LDU     CURLOC
        LDY     SCRLOC
        LDA     $FE08
        STA     24*80*2+1,U
        TFR     U,D
FASCNT  SUBD    #80*2
        CMPD    #$2000
        BHS     FASCNT
        ASRA
        RORB
        NEGB
        RTS

FASDSP  CMPX    #BUFEND
        BNE     FASXOK
        LDX     #BUF
        BRA     XFIXED
FASXOK  LEAX    1,X
XFIXED  LDA     ,X
        ANDA    #$7F
        CMPA    #$20
        BHS     FASCHR
        CMPA    #8
        BNE     FASNBK
        CMPB    #80
        BEQ     FASEX
        LDA     #$20
        STA     ,-U
        STA     24*80*2,U
        INCB
        BRA     FASEX
FASNBK  CMPA    #$D
        BNE     FASEX
        LDA     #$20
FASCLR  STA     24*80*2,U
        STA     ,U++
        DECB
        BNE     FASCLR
        BRA     FASLIN
FASCHR  STA     24*80*2,U
        STA     ,U++
        DECB
        BNE     FASEX
FASLIN  LDB     #80
        LEAY    80*2/8,Y
        CMPU    #$2000+24*80*2
        BLO     FASUOK
        LDU     #$2000
        LDY     #$10/8*($6600+$5*2) ($66000+$50*2)/8
FASUOK  STY     $FF9D
FASEX   CMPX    IREAR
        BNE     FASDSP
        RTS

LASDSP  STU     CURLOC
        STY     SCRLOC
        LDA     $FE08
        ORA     #$C0
        STA     24*80*2+1,U
        LDA     #$20
LASCLR  STA     24*80*2,U
        STA     ,U++
        DECB
        BNE     LASCLR
        LDA     DSPMMU
        STA     $FFA1
        LDA     DSPTSK
        STA     $FFA9
        RTS

KX      RMB     2
KB      RMB     1
KCC     RMB     1
```

Announcing the

4th Annual "Last" Chicago CoCoFest

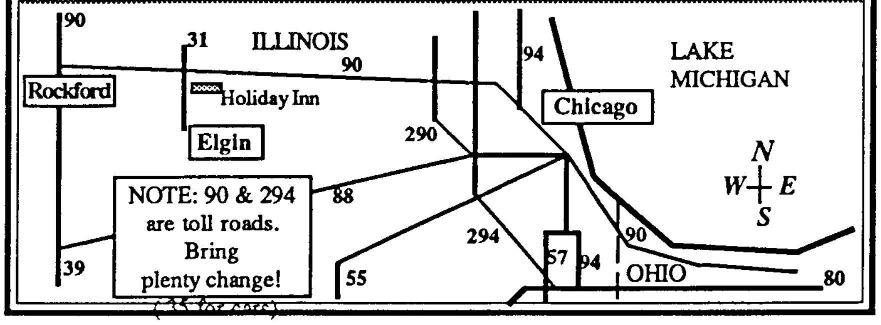
April 29 & 30, 1995

at the

Elgin Holiday Inn

a Holidome Indoor Recreation Center

345 West River Road, Elgin, IL



Show Hours:
Sat., April 29 10:00AM-6:00PM
Sun., April 30 10:00AM-4:00PM

Admission:
$5.00 in advance, $8.00 at door
(2 day pass only, order before 16 April)

Address for advance tickets:
Tony Podraza, Fest Coordinator
119 Adobe Circle
Carpentersville, IL 60110-1101

Reservations:
1-708-695-5000
1-800-465-4329

Sponsored by:
Glenside CoCo Club

Vendor Information:
Booth Price: $35 ($30 for 2nd)
Member Price: $30
(Membership can be purchased at registration -- $15 per year)
Reservations must be received no later than 3/25/95. Deposit of $20 per booth required with balance due due no later than 4/26/95. Balance received after 4/16/95 subject to a 20% late fee. Vendor setup Sat, April 29 5:30AM-9:45AM

```
01500
01510 CURLOC  RMB     2
01520 SCRLOC  RMB     2
01530 FASSP   RMB     2
01540
01550 INIT    STS     FASSP
01560         LDS     #STACK
01570         CLR     $FFD9           SPEED UP
01580
01590         LDX     #DEBOUN  KEYBOARD DEBOUNCE
01600         STX     $11B
01610
01620         CLR     CTRL            CONTROL
01630
01640         ORCC    #$50    INSTALL FIRQ HOOK
01650         LDA     JMP
01660         STA     $FEF4
01670         LDX     #FIRQ
01680         STX     $FEF5
01690         ANDCC   #$AF
01700
01710         LDA     #$35  ENABLE FIRQ ON FALLING CD
01720         STA     $FF21
01730
01740         LDX     #BUF
01750         STX     IFRONT
01760         STX     IREAR
01770         LDU     #$2000
01780         STU     CURLOC
01790         LDY  #$10/8*($6600+$5*2)#($66000+$50*2)/8
01800         STY     SCRLOC
01810         LBSR    FIRDSP
01820
01830         LDA     #$20    CLEAR SCREEN
01840         LDB     $FE08
01850         LDU     #$2000
01860 INICLR  STD     24*80*2,U
01870         STD     ,U++
01880         CMPU
                      #$2000+24*80*2
01890         BLO     INICLR
01900
01910         LDB     #80   POINT AT FIRST LAST LINE
01920         LDU     #$2000
01930         LDY  #$10/8*($6600+$5*2)#($66000+$50*2)/8
01940
01950         LBSR    LASDSP
01960         LDS     FASSP
01970         RTS
01980
01990 TERM    STS     FASSP
02000         LDS     #STACK
02010         LDX     IFRONT
02020         LDY     SCRLOC
02030         STY     $FF9D
02040 LOOP    JSR     [$A000]  CHECK KEYBOARD
02050         CMPA    #4
02060         BEQ     EXIT
02070         TSTA
02080         BNE     OUTPUT
02090
02100 FASLP   CMPX    IREAR   CHECK FOR INPUT
```

```
02120          LBSR    FIRDSP
02130          LBSR    FASDSP
02140          LBSR    LASDSP
02150          BRA     FASLP
02160
02170 EXIT     STX     IFRONT
02180          LDS     FASSP
02190          RTS
02200
02210 OUTPUT   CMPA    #189
02220          BNE     NCTRL
02230          STA     CTRL
02240          BRA     LOOP
02250
02260 NCTRL    TST     CTRL
02270          BEQ     NCTRL2
02280          ANDA    #$1F
02290 NCTRL2   CLR     CTRL
02300          JSR     CHROUT
02310          BRA     LOOP
02320
02330 CHROUT   PSHS    B,A,CC  DISABLE INTERRUPTS
02340          ORCC    #$50
02350
02360          LDB     #8              EIGHT BITS
02370
02380          CLR     $FF20           START BIT
02390
02400          ROLA                    2 2
02410          LBRN    0               5 7
02420          NOP                     2 9
02430          NOP                     2 11
02440
02450 OUT      LBRN    0               5 5
02460          LBRN    0               5 10
02470          LBRN    0               5 15
02480          STA     $FF20           5 20
02490
02500          RORA                    2 2
02510          NOP                     2 4
02520          NOP                     2 6
02530          DECB                    2 8
02540          BNE     OUT             3 11
02550
02560          LDA     #2              3 3
02570          LBRN    0               5 8
02580          LBRN    0               5 13
02590          NOP                     2 15
02600          STA     $FF20           5 20
STOP BIT
02610
02620          PULS    CC,A,B,PC ENABLE INTERRUPTS
02630
02640 FIRQ     STX     TEMPX           6 6
02650          BRN     *               3 9
02660          LBRN    0
02670
02680          BITIN
02690          LDX     IREAR           6 6
02700          BRN     *               3 9
02710
02720          BITIN
02730          CMPX    #BUFEND                 4 4
02740          NOP                     2 6
02750          BNE     OK              3 9
02760
02770          BITIN
02780          LDX     #BUF            3 3
02790          BRN     *               3 6
02800          BRA     FIXED           3 9
02810
```

```
02820 OK       BITIN
02830          LEAX    1,X             5 5
02840          NOP                     2 7
02850          NOP                     2 9
02860
02870 FIXED    BITIN
02880          STX     STA+1           6 6
02890          BRN     *               3 9
02900
02910          BITIN
02920          STX     IREAR           6 6
02930          BRN     *               3 9
02940
02950          BITIN
02960          LDX     TEMPX           6 6
02970          BRN     *               3 9
02980
02990          BITIN
03000          STA     TEMPA           5 5
03010          NOP                     2 7
03020          NOP                     2 9
03030
03040          BITIN                   7 7
03050          LDA     TEMPIN          5 12
03060 STA      STA     $8000           5 17
                                   DUMMY OPERAND
03070          LBRN    0               5 22
03080          NOP                     2 24
03090          NOP                     2 26
03100          LDA     $FF20           5 31
                           ACKNOWLEDGE INTERRUPT
03110
03120          LDA     TEMPA           5 5
03130          RTI                     6 11
03140
03150 JMP      JMP     *               DUMMY JUMP
03160 CTRL     RMB     1
03170 TEMPA    RMB     1
03180 TEMPX    RMB     2
03190 TEMPIN   RMB     1
03200          RMB     $100
03210 STACK    EQU     *
03220 BUF      RMB     $FFF
03230 BUFEND   RMB     1
03240
03250          END     TERM
```

< 268'm >

Thanks for patronizing our advertisers! Please let them know when you see their ads in 268'm!

BASIC in Color

Fred Remin

BASIC Support for the CoCo!

reprinted with permission from ***CoCo-Link***

Last issue I started this column for all beginners and maybe some not so beginners as well. The main gist of the first article was some suggestions for setting up your system to ensure that you enjoy more fully your computing time and a quick way to determine what type of system you own.

If you followed my step by step instructions and suggestions you should now be sitting in a very conducive environment and be fully aware as to what you have connected to your system. If not, then may I suggest that you once again read the last article, in particular the quick methods of determining how much memory you have and the Basic version.

Loading a Program

What I will cover next is loading a program into your computer and getting it running. The first thing to have a look at is are you using a tape or disk based system.

If you are using a tape system then first determine if the program is BASIC or BINARY. To do this first have a look at the instructions that came with the program tape. The instructions should tell you to either CLOAD (indicating a BASIC program) or CLOADM (a binary program) a file (many times this is written on the cassette label); i.e.:

```
CLOAD"programname"   or
CLOADM"programname"
```

After typing the "CLOAD" or "CLOADM" command, your cursor will change to a flashing "S", indicating that the computer is searching the tape for your file. It would be a good idea to make sure the tape recorder is playing now! When the file is found, the "S" will change to "F" and the program should load.

Once the program has loaded into your computer one of two things will happen. If the program has an "autoexecute" capability, then it will automatically start running. If, however, you end up with your normal green screen and flashing cursor, we need to give the computer an additional command to start the program.

If you used CLOAD to load the program, just type RUN and then press the ENTER key. If you used CLOADM to load, type EXEC and press the ENTER key.

Disk drive users would follow basically the same instructions without the "C" in CLOAD (LOAD) and CLOADM (LOADM). Use the exact same commands to start the program.

What does all this mean? Well, basically CLOAD means "cassette load". CLOADM means "cassette load machine language" (a machine language file is a binary file). RUN means to "run" a BASIC program while EXEC means to execute a binary program. The same rules apply to disk based programs.

Errors in Loading

The most common error to occur when loading programs from a tape based system is the dreaded I/O error. This means an Input or Output error has occurred and usually happens when you have tried to load a program when the tape is positioned inside a program rather than at the beginning of the tape or between programs. It will also occur when the tape player is dirty or the sound level is incorrectly adjusted.

The following are some of the tried and true fixes for I/O errors:

1. Rewind the tape to the beginning of the program or just before it. This can be done by using the tape counter (write the numbers down for each program!). You can also use the "SKIPF" command. This will skip the current program and stop at the beginning of the next. You can also type MOTOR ON <ENTER> and unplug the earphone jack from the recorder. The speaker will emit an awful screech -- that's you program. Type MOTOR OFF <ENTER> to stop the tape.

2. Clean the tape recorder. A head cleaning tape will work, but a more thorough job can be done with a cotton swab and some alcohol. Soak the swab in alcohol then wipe down the head and all rollers and guides the tape comes into contact with. Rubbing alcohol will not harm any of the rubber rollers... and they need cleaning too! Alcohol evaporates fast, so within a minute or two you're ready to try again.

3. Adjust the volume on your recorder. A good place to start is about half open. The Tandy CTR recorders with level of 1-10 usually work best between 7 and 8. Others will simply have to be trial and error. Start in the middle and keep trying a little louder until the tape loads successfully.

4. As a last resort, turn the tape recorder over. What happens is that the tape physically shifts a little in the recorder, thus moving where the head is on the tape. Sometimes this slight movement is enough, sometimes not.

For a disk system the I/O error usually means a garbled program, dirty drive heads, or dirty contacts on the disk controller and/or cables. There isn't much you can do about a garbled disk except get out your backup disk and make another copy. I cannot strees enough the importance of backups! The VERY FIRST thing you should do when you get a new disk or tape program is to make a backup! Note that some (but not all) stereo cassette decks will reproduce tapes just fine in the high-speed dubbing mode (my Sears bookshelf system won't, but my Kenwood rack system will).

You can't get to the disk drive heads with a cotton swab. You must use a special cleaning disk. All computer stores and many office supply stores carry cleaning disks for 5-1/4" and 3-1/2" disks. Get the one you need and follow the instructions.

Very often the problem lies with dirty contacts on the disk controller or cable. Turn your computer off and remove the disk controller. Clean the contacts on each end with a pencil eraser. You may want to take the plastic case apart for better access to the top contacts. There is a screw under the label in the center. Remove this then snap the case apart. There is usually no problem on the disk drive end of the cable, but it wouldn't

(continued on page 10)

The Industrial OS-9 User... by *Ed Gresick*

G-*WINDOWS*

One question often asked is where is G-WINDOWS used. Certainly it can be used as a simple user interface or GUI as I use it. The other end of the spectrum is the industrial user who will use the G-WINDOWS environment as a means of controlling industrial processes. Many are using ControlCalc from RTWare. Others are writting their own applications.

ControlCalc is available in two versions; a full development package which includes G-VIEW and a Run-Time package. Simply, ControlCalc is a spreadsheet with a graphics interface and some added capabilities. In addition to allowing manual input of data, data may be input to cells directly from ports which interface with the outside world. Other cells may have their outputs directed to other ports which interface to equipment outside of the computer. Recalculation of the cells may be triggered by events, etc.

Under G-WINDOWS, it is easy to monitor and control a process. A schematic or pictorial view of the process, machinery, etc. can be displayed. This can show the information necessary for the operator monitoring and/or controlling the process. Further, the operator may enter changes to the process to control it. These may be as simple as GO/STOP or ACCEPT/REJECT to continuosly variable rotary or linear controls - in effect, an analog inputs. Inputs may be via the keyboard or a mouse or some other pointing device such as a touch screen. In fact, the latter is popular in 'dirty'environments and where minimum operator training is necessary.

Many users are using multiple displays, each displaying and controlling adifferent process or different portion of a singleprocess. To my knowledge,most hardware supporting G-WINDOWS will also support multiple graphics cardsand monitors. The most popular combination to date use touch screens aspointing devices. I'll outline two applications I'm familiar with. DELMAR isproviding the hardware and writing additional I/O drivers. The customersare writing their own software and designing, building and/or providing additional hardware/machinery as required. Both projects are in the development stage.

The first, and intended to be sold to other Companies, I'll call an 'industrial laundry' product. This customer will be offering a computer, the software and necessary sensors and controllers to interface with existingequipment being used in 'industrial laundries'. Depending on the product being 'laundered', the process may require two or three stages. The first stage does a preliminary, brute force laundering. The effluent is analyzed and the information used to control the mix being used for laundering. The data is also sent to the next stage. This stage does the final laundering.As in the first stage, the effluent is monitored and used to control the cleaning mix. After the final laundering, the product is washed and the washeffluent is analyzed. This determines whether the product is clean and what adjustments must be made to the preceeding stages. Each stage will have it'sown display and use touch screens for manual inputs. ControlCalc is being used to write the application software. Custom gadgets (the do-hickies on the screen to represent various things) are prepared with G-VIEW.

While interrupt handling isn't critical, OS-9 was selected because of itsmulti-tasking capability and re-entrent code. The customer stated that the combination of OS-9, G-WINDOWS, G-VIEW and ControlCalc provides him withtools and performance unavailable with other operating systems - that theyhave investigated other OSs thoroughly.

This product is being tested at a customer's site. The advantages to theircustomers will be lower cleaning material costs and the potential eliminationof re-laundering products.

The second application isn't as far along. It is a cutting application not unlike cutting cloth in the garmet industry for clothing. Interrupt speed and handling is important. The 'material' is man-made and manufactured by this Company. It is cut and assembled into the final products which they sell. The pattern is irregular and varies according to the specific end product. The products are expensive, the field very competitive and they dominate it - and they wish to continue to so. The material is cut on an X-Y machine with a working area 4'x 4'. Cutting is by means of a laser knife and they are achieving cutting speeds up to 1400 inches per minute.

The X-Y machines are currently controlled by AutoCad programs running on 486 machines. The final size of the cut material is critical. Tolerances must be held to a few thousandths of an inch over 45 inches. Because the materialisn't perfectly homogenous, the final size is affected by assembly. They currently have a reject rate of about 25%. The variations in the material are known and each sheet has the characteristics of that batch in bar code form which is entered into the computer prior to processing that sheet. Also there are stresses introduced in the material while it is being cut. They can measure the stresses and they know the effects, but the program runningon the 486 can't handle them unless they slow cutting rate substantially.

For comments or questions, Ed can be reached via this magazine or:

E-mail:
EDELMAR@delphi.com
76576,3312@CompuServe.com

U.S. Mail:
PO Box 78
Middletown, DE 19709

Telephone:
302-378-2555 Voice
302-378-2556 FAX

MM/1 and OSK support from BlackHawk Enterprises

Hardware:

MM/1 Serial Cards	$35
MM/1 Midi Cards	$45
68340 acelerators	$325
SCSI Tape drives	call
SCSI Hard drives	call
BGFX in stock!	$45
RAM prices	call
Floppy Drives	call

Coming Soon - Modems, CD-ROM

Software:

PixUtils	$25
DeskTop for MM/1	$79
Fontasee	$35
Paint for MM/1	$79

New Software on the way!

Now available - ~~COMPLETE~~

MM/1 Systems!

(call for pricing)

BlackHawk Enterprises, Inc.

P.O. Box 10552

Enid, OK 73706-0552

Phone 405-234-2347

Internet:

nimitz@delphi.com

OS-9 and G-WINDOWS have been tested for several months. They have written a simple test control routine with mostly text output and are displaying only a simple deviation curve with graphics. Recently, they added a second VGA card and monitor to control a second machine. Preliminary results show satisfactory performance from one SYSTEM V controlling two X-Y cutting machines and it appears the SYSTEM V, running at 25 MHz, is loafing with the X-Y cutting machines running at maximum capacity. The reject rate of material cut on these two machines has dropped to under 1% (from around 25%). The next step will be to integrate the measurements of the material into the program. This will eliminate the cutting process (into squares or rectangles) permitting them to feed the X-Y cutting machines from continuous strip material.

Discussions with their engineers show a complete satisfaction with this approach. They have some more testing and expansion they wish to try. When they finally determine the limits, they plan on purchasing ControlCalc to write their final software. Their target is one computer controlling up to four X-Y machines. Currently, 18 X-Y cutting machines are in use. The benefits they expect are - reduced reject rate, fewer operators (one for eachcomputer controlling 3 or 4 X-Y machines) and increased capacity with the existing hardware. I don't know exactly what each window will display, but from their questions, I expect the data being logged will also be shown in a graphics format (bars, curves, etc.) to assist detecting trends and permit input and/or control via a touch screen.

As you may've suspected, the Company is very conservative. If the letters IBM aren't associated with the computer, it isn't supposed to be used. I believe (hope?) that when their engineers have finished this prototype work station, they will have the evidence (dollar savings) to convince management to look away from IBM / MS-DOS and in some new directions.

< 268'm >

(continued from page 8)

hurt to clean those contacts as well. You can spray some tuner cleaner or electrical contact cleaner in the female connectors on the cable and the CoCo port.

The next most common error for both tape and disk users is the FM (File Mode) error. What the computer is telling you is that you have tried to load a BASIC program as a machine language file or vice-versa. This comes about by trying to CLOAD a program when you should CLOADM.

Saving a File

The same general procedures used for loading a file are used to save one. Of course, instead of using CLOAD or CLOADM you use CSAVE or CSAVEM, dropping the "C" for disk systems. If using tape, don't forget to let the tape run 3-4 counter numbers between saves. This makes for fewer I/O errors later! Do the same at the beginning of a tape, even a new one. Disk systems keep up with where the files are on the disk for you, so no problems there!

Saving a machine language file to tape is a little more complicated than simply typing CSAVEM. You need to know the START, END, and EXEC addresses of the program also. To find these, type the following while the program is loaded in memory but not executing:

To find the START address:

```
PRINT PEEK(487)*256+PEEK(488)
<ENTER>
```
write down the number.

To find the END address:

```
PRINT PEEK(126)*256+PEEK(127)-1
<ENTER>
```
write down the number.

To find the EXEC address:

```
PRINT PEEK(157)*256+PEEK(158)
<ENTER>
```
write down the number.

Now the machine language program can be saved by typing:

```
CSAVEM"programname", START, END, EXEC <ENTER>
```

Simply insert the number you wrote down in place of START, END, and EXEC.

You should now be able to successfully load and save both BASIC and machine language (binary) programs from tape or disk. If you are still having problems please do not hesitate to drop me a note or give a call. I will be happy to help out if I can. Next issue we'll start some programming! < 268'm >

The Hardware Hacker

Dr. Marty Goodman

PALs and GALs explained

Several of my articles here have involved referring to PALs and GALs in Color Computer applications, such as the upgrade for the old Multipak Interface and the upgrade to the DS69A digitizer. Several folks have asked me to explain to them just what "PALs" and "GALs" are, and how they are similar to the familiar related programmable chips, the PROM, the UV-EPROM, the EEPROM, and the "flash EPROM."

PALs vs GALs:

PALs (Programmable Array Logic) and GALs (Generic Array Logic) represent two successive generations of technology that make wiring up logic circuits more of a matter of programing a chip rather than a matter of wiring together numerous separate chips that each have specific logic gates on them.

PALs are the earlier technology (late 1970's). These came in dozens of varieties with fixed numbers and varieties of outputs and inputs. Programing them consisted of physically blowing fuses inside the PAL, so they could be programing only ONCE. In this respect, a PAL resembles the old PROMs, which are memory chips that are programed by physically blowing fuses inside the chip.

GALs are a more recent technology, which represents a number of major improvements. GALs are electrically erasable, so can be reprogrammed. In this respect they resemble EEPROMs. They cost less than PALs. They tend to be faster (have shorter logic gate delays) than the old PALs. GALs are more flexible. One 16V8 GAL chip can (depending on how it is programmed) be directly substituted for any one of two dozen different PAL chips. Thus, all but the two most flexible PAL chips (the 16R8 and 16L8) are today totally obsolete, and the 16R8 and 16L8 will soon disappear, for all of their functions can be done by a 16V8 GAL chip faster and more cheaply.

GALs sometimes go by different names depending on the manufacturer. AMD calls their GALS "PALCE" devices. ICT calls their GALs "PEELs".

GALs vs EPROMs:

The important thing that differentiates PALs and GALs from PROMs and EPROMs of various sorts is that what one is programing in a PAL or a GAL is the WIRING of an ARRAY of LOGIC GATES inside the PAL or GAL. Programing a GAL is literally comparable to physically soldering up a bunch of small scale logic gates for a specific function. With a PROM or EPROM, one is typically dealing just with computer memory. To confuse the issue, I must admit that PROMs have been cleverly used to substitute for logic in certain design situations. But mainly, one is putting binary 1 and 0 information into a PROM or EPROM that later is read by a computer.

PALs and GALs, in contrast, are employed to replace bunches of small scale logic chips (such as 74-series or 4000 series) in order to greatly reduce chip count, decrease design costs, provide greater flexibility for later design changes, increase the speed of logic, or all four of these consideration together. Depending on the program, one GAL can replace as many as a half dozen or more small scale logic chips.

Programing PALs and GALs:

Programing GALs is a rather sophisticated operation. First one chooses a particular GAL based on the capabilities it offers, typically number of available input and output pins. Then one uses a programing LANGUAGE (such as "CUPL" or "PALASM") to define which pins of the GAL are inputs and which are outputs. One then uses the language to specify, among the outputs, which outputs are simple combinatorial logic outputs (such as the output of a 74LS00 AND gate), which are tri-state outputs (such as the output of a gate in a 74LS125 chip), and which outputs behave like the outputs of one of a variety of flip-flop modules (such as "D-flip flops" or "R-flip flops" or "JK-flip flops"). If one has specified some of the outputs as flip-flops, one pin of the GAL becomes a clock input pin.

If one has specified one or more outputs as tri-state, one pin of the GAL becomes the output-enable input. After specifying pin functions, one literally writes Boolean Algebra equations that define exactly what all the output pins do based on information that appears at the input pins. When one is done, one ASSEMBLES these specifications using the PAL-assembler, and an output in the form of a "JED fusemap" is produced. It is this fusemap that is then used by the GAL programer to burn that logic into the chip.

GAL programmers (the physical device that burns the fuse map into the chip) are electronically more complicated than EPROM programers, for they have to accommodate a much wider variety of chips AND a more complicated set of procedures for programing each chip. These and other factors have resulted in GAL programmers being considerably more expensive than EPROM programers. Prices tend to start at $700 for CHEAP GAL programers. Such programers tend to be "device programers" that will program EPROMs and controller chips, too. Around this time I suspect that some hobbyist-type inexpensive GAL programmers for PC compatibles that just handle three or four of the more common (such as 16V8, 20V10, 22V10) of the hundreds of different GAL chips may be appearing, costing considerably less. But I off hand don't know of any specific offerings like this.

Let's look at a simple application for a GAL chip: The upgrade PAL (GAL) chip for the Multipak Interface. Below is my object code written in CUPL assembly language specifying the 16V8 I supply for upgrading the Multipak Interface:

```
Partno      None;
Name        MPAKCC3;
Revision    01;
Date        6/12/91;
Designer    Marty Goodman;
Company     Cheshire Cat;
  Assembly    None;
  Location    None;

  /*********************/
/* This is the logic for the CoCo 3 */
/* UPgrade PAL for the Multipak */
/* Interface , Tandy Cat # 26-3024 */
/***************************/
/*    Taget Devices: P16V8       */
/***************************/
/* * Inputs: define inputs */
Pin 1 = FF;
Pin 2 = A7;
Pin 3 = A6;
Pin 4 = A5;
Pin 5 = A4;
Pin 6 = A3;
Pin 7 = A2;
Pin 8 = A1;
Pin 9 = A0;
Pin 11 = RW;
Pin 12 = E;
Pin 13 = Q;
Pin 18 = SLENB;
Pin 19 = CTS;

/* * Outputs: define outputs */
Pin 14 = ENBUS;
Pin 15 = ENREAD;
Pin 16 = LOADREG;
Pin 17 = unused;

/* * Logic: */
!ENBUS   = !CTS # !SLENB # (!FF & !A7 &
A6);
!LOADREG = !RW & !Q & E & !FF & !A7 &
A6 & A5 & A4 & A3 & A2 & A1 & A0;
!ENREAD  = RW & E & !FF & !A7 & A6 &
A5 & A4 & A3 & A2 & A1 & A0;
```

The 16V8 GAL chip that I specified is a 20 pin chip. 16 of those pins can be specified as inputs, and up to 8 of those pins could be specified as outputs (hence the designation 16V8). In this application, I specified that three of the pins (14, 15, and 16) would be outputs, 14 pins would be inputs, and one (pin 17) would be left unused. Pin 10 is ground on this chip, and pin 20 is Vcc. I defined the logic for the output pins based on what I knew of how the Multipak Interface functions, on the schematic for the Multipak Interface, AND on the helpful logic equations provided in Tandy's service manuals for the new and the old Multipak Interface devices. In CUPL's language, "!" means "inverted" (equivalent to a line over a logic function, or an asterisk in front of the logic function, in other logic notations), "&" means a logical "AND", and "#" means a logical "OR". Tho not used in this design, "$" in CUPL notation means "Exclusive Or". When I feed the above design specs into a CUPL assembler, I get out the following JED fusemap:

```
CUPL      2.15b Serial# 6-00005-083
Device    g16v8s Library DLIB-h-24-9
Created   Wed Jun 12 09:13:55 1991
Name      MPAKCC3
Partno    None
Revision  01
Date      6/12/91
Designer  Marty Goodman
Company   Cheshire Cat
Assembly  None
Location  None
*QP20
*QF2194
*G0
*F0
*L0768 10100111011101110111011001
010110
*L1024 10100111011101110111011101
010101
*L1280 11111101111111111111111111
111111
*L1312 111111111101111111111111111
11111
*L1344 101001111111111111111111111
11111
*L2048 000000000111001011110110011
10110
*L2080 101001100000000000000000000
00000
*L2112 000000001110001111111111111
11111
*L2144 111111111111111111111111111
11111
*L2176 1111111111111110
*C1BB8
* 9E90
```

If you feed this fusemap into to a GAL programer, and program a 16V8 GAL with the data, you will have the GAL (or "PAL" if you prefer to call it that) which will successfully upgrade a Tandy Multipak Interface. The fusemap specifies to the GAL programer where to "burn" and where to leave connected links in the internal logic array of the GAL chip.

This particular application is a relatively simple one, because it is "purely combinatorial": It does not use "registers" (flip-flops). It's also simple in that all of the outputs are "simple outputs". None are "tri-stateable". The PAL in the Multipak interface is used to create three signals (ENBUS, ENREAD, and LOADREG), which in turn are used to disable the data bus when "forbidden" parts of the data bus are being read (GIME registers) or when the SLENB line is active, and to decode for the internal Multipak Interface register at $FF7F, to allow one to point to given slots using software commands.

Let's look at just one of those signals, the LOADREG signal, which is used to decode address $FF7F for the internal Multipak slot select register. The LOADREG signal is active LOW in the Multipak Interface circuit. Thus, we need to specify a signal that will go LOW only when address $FF7F is on the Color Computer system bus. Additionally, we need to specify that the signal is active only when a WRITE command is being specified, and this in turn means that the R/*W line (represented here as RW) must be low). There's one other thing we need to do: Make sure that LOADREG is active ONLY when the Q clock is low and the E clock is high.

To fully understand the line that specifies the LOADREG output, you also need to know that in the Multipak Interface circuit, the 8 high order address lines (A8 thru A15) of the CoCo system bus are fed into a 74LS30 8 input NAND gate, and the resulting output signal from that 74LS30 (the signal I call "FF" in the GAL design information) is fed into the Multipak PAL on pin 1. This "FF" signal is active low only when the two high order digits of the four digit hex address on the CoCo system bus is "FF".

Now, look at the line in which I specify the logic of the LOADREG signal:

!LOADREG = !RW & !Q & E & !FF & !A7 & A6 & A5 & A4 & A3 & A2 & A1 & A0;

Look at the left side of the equal sign. I've put a "!" in front of the name of the signal, to specify that this is a signal that will be active when low. Now look to the right side of the equal sign. Here I specify that the signal will be active when RW, FF, and Q are low (for you can see "!" in front of those inputs in the logic description), and when E is high. This sets the signal to be active low when the R/W line is specifying a WRITE command, when the 8 high order bits of the address bus are high (FFxx), and when the E clock is high and the Q clock is low. Let's now look at the rest of the decoding logic specified in this line. You can see that LOADREG is active low when A0, A1, A2, and A3 are all HIGH (because none of those have a "!" in front of them in the specification). This means that the four low order address lines must be HIGH for LOADREG to be valid. This in turn further decodes LOADREG to address FFxF. Now look at A4, A5, A6, and A7. Note that only A7 has a "!" in front of it. This means that LOADREG will be active when A4, A5, and A6 are all high, and A7 is low. This in turn specifies a "7" for that particular digit of the HEX address. Putting it all together, we see that LOADREG is specified to be active low when address $FF7F is on the CoCo address bus, and when a valid "WRITE" is being requested. It's further narrowed down to the proper part of the CoCo timing cycle (using the E and Q clock information) to allow us to write valid data into the register in the Multipak that specifies what slots are selected. The LOADREG signal is used to clock data into a 74LS374 chip, which in turn determines what slots the *CTS, *SCS, and *CART data lines are hooked up to.

Note two things here: (1) It's easy to change the address being specified by just changing the programing of the GAL chip. This allows for easy upgrading and alteration of the design logic. While the logic for LOADREG is not changed when one goes from the original Multipak PAL to the version upgraded for the CoCo3, the logic for the other signals is changed, to lock out the address range $FF80 - $FFBF used by the GIME chip of the CoCo 3. (2) To specify the logic that this GAL accomplishes, one would have had to use three or more small scale logic chips, instead of this one PAL or GAL chip.

NOTE, also, that the original Tandy Multipak Interface used a 14L4 PAL chip for address decoding, for which I substituted a 16V8 GAL chip. 14L4 chips just ceased to be available around the time I first laid out the logic equations for my Multipak Interface upgrade PAL chip. I actually did compile a fuse map for an upgraded 14L4, and burned a few 14L4's for some dealers. But 14L4's turned out to be more expensive than 16V8's, and soon were unattainable. So I switched over to using the more modern (faster, cheaper, and re-programmable) 16V8 chips. Tandy apparently actually was unable to get hold of 14L4's when it came time for it to supply "official" multipak upgrade chips, or may simply have gotten a better deal on another (equally obsolete, slow, one-time-programmable, but adequate) chip, the Phillips Signetics PLS153. Thus, upgrades that Tandy provided will be Phillips Signetics PLS153 PAL chips, while almost all of the upgrade chips I provided to dealers and individuals are in the form of 16V8 GAL chips. My upgrade should work as well (or better than, because the faster logic in the 16V8 may make for more reliable operation) the official Tandy upgrade.

< 268'm >

Comments and questions may be sent in care of 68'Micros or directly to :

Dr. Marty Goodman
1633 Bayo Vista Avenue
San Pablo, CA 94806

E-mail: martygoodman@delphi.com

Operating System -Nine

Russell Hoffman

More on memory management

My apologies to those who were waiting for the final half of the memory management tutorial I'd planned for last month- it vanished into the great hard drive crash of 94. Since it's certain to be a while before that text is readable again, this month is a quick summary of what's freshest on my mind- hard drive installation and backup.

Cobbler:

MODULES is a collection of files with names like ddd0_40ds.dd (default device=d0=40trk dblside.device descriptor) which have to be collected into a single, simple file the 'bootstrap' loader can handle, the job we call 'making a boot'.

Cobbler appears to provide a way out of the hassle of maintaining bootlists and deciphering cryptic filenames. It's both handy and a fair pun (as geeks go) but cobbler was only intended as a fast snapshot of existing boots or trivial mods of them. After five or six cobblers you have a boot that will be difficult to recreate if it's ever lost. Backing up a step means digging around for a 2 year old ex-boot.... you get my drift.

Of course, you probably thought of all this and carefully save a copy of each modified module.....in /dd/SYS/MODULES where you can't get if the hard drive explodes:-) In my case, the need to restore the drive was a result of yet ANOTHER blown 6309 halt line (and the corresponding blown fuse). Nice to find the old wind_lst6809.io file tucked away with the descriptors and stuff.

In the beginning, there were Disk Drives:

OS-9 names disk drives and disk drive like devices as floppies /d#, hard drives /h#, or ramdisks /r#. These logical names connect to drive descriptors using the same name, merged into 'os9boot'. These descriptor modules then call the driver assigned to each device type with the fine info, and all of the above consult the manager RBF to actually make a decision.

Floppy drives

Each floppy drive has a descriptor, which calls (for Tandy type controllers) the cc3disk driver. Almost everyone uses a modified cc3disk which can adapt to disk basic and MS-DOS formatted disks. Tandy supplied most obvious descriptors in SYS/BOOTMAKER/MODULES and most modified drivers use these Tandy style descriptors. At least two will be merged in boot, (plus default drive) these may need to be replaced or modified. For configurations not included (like 80trk /d0) an existing descriptor may be modified 'on the fly' using dmode, and then saved to MODULES. Remember a /dd copy if floppy based.

Hard Drives

Are very similar to floppies in basic structure, since they share RBF.mn. The basic division between CoCo hard drives probably lies in how the drive configuration is stored. SASI (Disto original) or MFM/RRL (Burke&Burke) store drive info in an internal table. The original OS-9 format can follow this, so after making a descriptor, initializing the drive is exactly like initializing a floppy.

SCSI drives record vital info on the drive itself during format- (KenTon or Disto option) which makes preformatted drives real easy to install. Mounting an old hard drive is reduced to plugging it in vs. having to premodify descriptors to match a long lost data table. Roll yer own usually means a disk basic utility program to add the scsi data sectors after a formatting the device, which means you ***do*** need that table!

Hard drive driver flavors-- Tandy and Disto-- share the name cchdisk, while Burke&Burke supplied a real collection, all named bbSOMETING. (how well it spells 'BBFHDISK' indicates it's capability (thus size- from level one tiny to who wants to format a floppy anyway?)). Each drive has a descriptor, similar to the floppy module, and how they are created varies by vendor.

Tandy included descriptors for all of their hard drive systems in the 'Development System' MODULES dir, along with their cc3hdisk (crc=$FCC2DB). Other systems require knowledge of at least the number of 'cylinders' and 'heads'- additional info is desirable. Unless this is a reprint, find a floppy with DOCS/HARDWARE/TABLES which includes most typical drives (all Seagate, some ancient, **********)

Disto provides a generic 5meg descriptor, and a custom 'dmode' (sometimes filenamed 'hmode') command. This boot can be cobblered, but it's still a good idea to save the new descriptor(s) to MODULES (see floppy text above) for later use. One feature of the Disto driver not available from dmode- the multipak slot to be used- is byte $15 (set with ded to $80+slot#-1).

Burke & Burke supplied a 'ddmaker' utility to create custom descriptors, which are ezgenned (ezgen typically comes on any B&B disk) into a boot. While the B&B only supported two drives, a completely separate controller can be added to add two more. The process also adds an independent copy of the bb*disk driver for the second controller (except Nitros9), with suitable descriptors.

The final wrinkle

To this structure, add the 'default device'. This is simply one of the above descriptors (any kind of drive), with the logical name changed to /dd, which means one piece of hardware ends up with two descriptors so two logical names. Any program that has to store an internal file (say, a high score) or access a SYS file knows where you keep this sort of stuff automatically, by asking for the default device.

In the bootlist, it's usually placed right under the corresponding driver- which takes a little extra time using utilities like Ezgen (which have to open and close space instead of just replace), but a later ident -s will then show what sort of boot it is.

The default device is one of those neat ideas that didn't require a major opsys change to implement, but it is a recent invention. Older software was often coded to look for a specific drive, which means programs written for floppies wouldn't transport to hard drives. Use DEd to search these programs for the strings D0 and D1 (they will usually be readable ASCII, even in the executable), and change these to DD. One example, ComputerWare's old basic compiler- many expensive apps like a payroll package where written and sold. In all cases, editing eight characters moved the entire package to a CoCo3/hard drive.

CoCo OS-9 itself 'features' this sort of hardcoding- both in cc3go and init. There is one minor difference to look out for. OS-9 kind of thows a curve here, in that the last character in a string has the high bit set. To change, use hex mode and check your work in the ascii window (ie- entering 44 c4 shows "DD").

What you should do about it.

We've discussed crash boots before. Excellent plan. (ed: Rick mentioned this when his hard drive crashed recently... he wasn't exactly prepared!) From the top, ***remove*** the hard drive /dd, then add a floppy /dd in it's spot under cc3disk. Check that cc3go (root dir) and init (in os9boot) have been changed to dd and not h0, since only the dd name will follow your changes automatically.

To this boot, you'll need at least a minimal cmds dir with shell and grfdrv. I'd suggest adding the disk utilities, including your hard drive restore program- and a Gen util along with SYS/ MODULES. This will use so much room, this 360K is useless for normal purposes but it's perfect when the drive fades away.

From this base, a 'real' hd boot is easy. Cobbler a new disk and copy cc3go to it. The edit os9boot, rip out the floppy /dd, move down below *hdisk, insert hard / dd. These two files are the entire boot disk!

The next time the operating systems needs to be modified, boot with a copy of the 'crash' boot, and modify/test that first. If it does blow up and garbage a directory, it will probably hit /dd...just another floppy. 'Provisional' modules under test can be left in floppy SYS/ MODULES until they've been proven useful... or left on the (abandoned) floppy as a record of *why* you gave up last time. With an added benifit the main modules directory never has any untested pieces in it to be used accidentally at some later date.

Simple changes may not require this particular dance-- for example inserting /h1 is such a trivial task (after practice) there's little need to test. At least save a copy of the new descriptor to the 'crash' MODULES dir for future repairs.

The disk.

First a digression. As this column has matured, the number of utilities and other programs discussed has expanded. Long time readers may remember my original intent to stick pretty close to what was commonly available. Limiting ourselves to only Tandy release didn't work out to well, so 'disk of the month' was started as an alternate source for the fantasic programs coco users have contributed to the community.

Now the scene is changing, as new subscribers (and even new users) start looking for the same programs. Add reprints and folks who want multiple sets and DOM is getting cumbersome. So the old ones have been grouped into logical sets- we've even added a few app disks (which arrive ready to run, not dearc). Write or email CoNect for the full list. With more to offer, we need a multiset deal- I'll try $5.00 the first set plus $3.00 per additional, per order.

Back to the task at hand- Pulling this CoCo back together required several pd and shareware utilities, now collected on the 'SysAd' package.

What they are, and why I needed them:

dEd is probably the most popular disk editor for Lvl2, allowing ascii and hexadecimal input. Anything modpatch can do in ram (with the ensuing cobblers, saves, etc) can be done directly to os9boot, the disk file- saving a few steps. This is also an excellent way to scan old programs (it has a search function) for those pesky hardcoded /D0 and /D1 references.

Dmode allows you to change floppy descriptors on the fly- you can temporarily 'borrow' another systems boot and at least reach your floppys quickly, even when the borrowed boot is one of Tandy's 35 specials. (Disto sasi users pull the same trick on the hard drive).

ipatch & makpatch are the standard way to distribute mods for copyrighted code. Makpatch is used to compare two files, and the differences (stored as an .ipc file) can be fed to ipatch later. Obvious when installing others patches, it's useful at home.... I have 2 or 3 versions of some programs, but usually only use one. Rather than keep all three online, makpatch created patch files to modify the (already modified) version I usually use into the others. This sounds a little goofy, but it's faster to recreate the program than dig around the back room for the archive when I need an old version.

label is one of those simple utils you can't imagine being left out. It changes the disk name recorded at format (you used blank, didn't you?) into something meaningful.

megaread isn't for the weak of heart. There's nothing wrong with the utiltiy itself- it reports how long your hard drive takes to transfer one megabyte, and does it well. Improving the figure usually requires reformatting with a different interleave and other tricks, which is where the condolences come in.

Save is an obvious choice.

Stream..... first, it's ***not*** free, it's shareware. (Bruce Isted, $25). Second, it's definately worth $25. Give it a look. Besides speed, a nice feature is the ability to extract single files from an archive (or maybe your MODULES dir:-). Some online releases had a doc problem- if you tried before and couldn't get it working try files -e ! stream -bv /d1.

vfy is a little difficult to describe- just think of it as a header editor. You can modify everything from ram requests to the edition number given, without resorting to manually tweaking bytes. And, it verifies:-)

In the interest of completeness, a few other 'system administration' utilities from other sets- iconedit, pc/rsdos, undelete, and possibly others are on the CoNect disk.

vrn was to be on the last DOM, now it's here. This driver supports the sort of RAM swaps and 'virtual irq' used by programs like Flight Simulator and King's Quest3. It's most pedestrian application is running these two programs under a more or less normal boot (for those with a meg or better).

< 268'm >

Comments and questions may be sent in care of 68'micros or directly to the author at:

Rick Ulland
449 South 90th
West Allis, WI 53214
E-mail is rickuland@delphi.com

OS-9/OSK Answers!

Joel Mathew Hegberg

Tackling TERMCAP... PART II

In the last issue, I gave the source code for an entire program which uses the TERMCAP (TERMinal CAPabilities) library to control the cursor movement and decode input from the user. The program simply clears the screen and allows the user to type anywhere on the screen, using the arrow keys to move the cursor. Now, it's time to explain how the program works! I will assume you have the previous issue on hand to refer back to while I explain, and I have to assume that those reading this understand the concept of pointers to (i.e. addresses of) data.

I like to break the TERMCAP learning process down into four categories — required variables and functions, initialization, outputting data, and inputting data. Not every program will have both input and output via termcap, but all termcap programs must have an initialization sequence and the required variables and functions. For instance, your program may just display fancy character patterns on the user's screen for entertainment (a screen-saver, perhaps). This would not require and termcap-supported input functions, so those could be omitted.

Required Variables and Functions

The first thing you need in your termcap program is '#include <termcap.h>' so all the library functions will work! Next, we need a couple variables to remember how many lines and columns are available on the user's terminal, and we also need to define a variable called "ospeed", which I'll explain a little later. These three variables can be defined as type 'short' (2 byte integers) like this:

```
        short lines,columns,ospeed;
```

At this point, we need to decide what terminal capabilities our program needs in order to do what we want it to do. (Remember, a list of terminal capabilities is found on pages 8-31 through 8-37 in the "Using Professional OS-9" manual.) For each capability, our program will need to have a character pointer assigned to use it. For example, our program will need to have "cl" (CLear screen), "cm" (Cursor Move), "ku" (Keypad Up), "kd" (Keypad Down), "kl" (Keypad Left), "kr" (Keypad Right), and "kb" (Keypad Backspace). The naming convention used under TERMCAP is simply using the two-character capability name and capitalizing them. So, we define our capability string pointers like this:

```
    char *CL,
        *CM,
        *KU,
        *KD,
        *KL,
        *KR,
        *KB;
```

In addition, the termcap library routines (tgoto in particular) require three more variables be defined — PC_ (for Pad Character... note this is not a pointer), *UP (move cursor UP a line), and *BC (for Backspace Character string). You will note they are in the program (see previous issue).

So now we've just set up some pointer variables... the question you should be asking is, "What do they point to?" I'm glad you asked! They point to character strings stored in a termcap buffer, that is yet another required variable definition. Our initialization routine will fill this buffer with all the character strings our program needs to control the user's terminal and point our pointer variables to those strings. So, how big should our buffer be? The examples I've seen have it set to 400 bytes, but you may want to change this if you have problems so it's best to make it a #define:

```
  #define TCAPSLEN 400
  char tcapbuf[TCAPSLEN];
```

Alright, that's all for required variables. Now, we just need to write one function which is required by the termcap library called user_tputc(). The exact name isn't important, so feel free to rename it. It outputs a single character (passed to it as a parameter) to the user's terminal. Here's what you'll find in the program we're working on:

```
/* writes one character to terminal. */
/*needed by tputs() library function*/
int user_tputc(c)
char c;
{
  return (write(STDOUT,&c,1));
}
```

Initialization

The initialization of our termcap program is a pretty straight-forward process. We need to find out what type of terminal the user is on, read the information about the terminal from the /dd/SYS/termcap file (or from the environment variable TERMCAP, if defined), extract only the capabilities we want and store them in our termcap buffer (tcapbuf) while pointing our string pointers to them, and find out how many lines and columns are on the user's terminal. We also have to do quite a bit of error checking as well. We need to handle conditions where the user's TERM environment variable may not be defined, the terminal type is unknown, not all of our needed terminal capabilities are defined in the termcap file, and our termcap buffer (tcapbuf) is too small. We now begin learning about the termcap library functions. There are four functions used for initialization — tgetent(), tgetnum(), tgetflag(), and tgetstr(). The first function, tgetent(), grabs the raw termcap entry and stores it in a temporary buffer. It is important to realize that this is a raw form of the termcap entry that is unusable for input/output. We want to extract the termcap strings our program needs and store them in a usable form in our termcap buffer (tcapbuf). (My manuals recommend having the temporary buffer at least 1024 bytes or longer to accomodate large termcap entries.) So for our initialization routine, let's define a temporary buffer (tcbuf), a pointer for the user's terminal type (*term_type), and a couple generic pointers (*temp and *ptr):

```
chartcbuf[1024],*term_type,*temp,*ptr;
```

Looking back at the previous issue, we can see how term_type= getenv ("TERM") is used to find out the user's terminal type (or if one isn't defined, in which case an error is printed). We use this term_type pointer in our tgetent() call to grab the raw termcap entry and detect if the terminal type is unknown. The tgetent() function is smart enough to determine if the user has defined a TERMCAP environment variable to hold their termcap entry.

```
if (tgetent(tcbuf,term_type)<=0)
{
  fprintf(stderr,"Unknown terminal type
'%s'.\n",term_type);
  exit(1);
}
```

Now we start filling up our termcap buffer (tcapbuf) with the capability strings that we need, and set up our pointers to them. While filling tcapbuf, we have to keep track of where the next string needs to go in tcapbuf, so we'll use our generic pointer 'ptr' for this task. tgetstr() automatically adjusts our pointer for us and returns a pointer to the location of the string stored in our buffer. This makes extracting our needed capabilities very easy:

```
  /* read the termcap entry */
  ptr=tcapbuf;
  if (temp=tgetstr("PC",&ptr))
PC_=*temp;
  CL=tgetstr("cl",&ptr);
  CM=tgetstr("cm",&ptr);
  KU=tgetstr("ku",&ptr);
  KD=tgetstr("kd",&ptr);
  KL=tgetstr("kl",&ptr);
  KR=tgetstr("kr",&ptr);
  KB=tgetstr("kb",&ptr);
    UP=tgetstr("up",&ptr);
```

If any of our needed terminal capabilities do not exist, their pointers are set to NULL. We can then check to make sure we have everything we need by looking at our pointers and making sure all of them are not NULL:

```
  /* make sure we have everything */
  if (!(CL && CM && KU && KD &&
KL && KR && KB && UP))
  {
    fprintf(stderr,"Incomplete termcap
entry.\n");
    exit(1);
  }
```

To discover the number of lines and columns the user has on his terminal, we use tgetnum(), since we're getting a number rather than a string. tgetnum() does not touch our tcapbuf. This is the same for tgetflag(), which returns 1 for true or 0 for false for a boolean termcap capability. Only tgetstr() updates tcapbuf and our generic pointer:

```
  lines=tgetnum("li");
  columns=tgetnum("co");
  ospeed=-1; /* no padding */

  if (lines<1 || columns<1)
  {
    fprintf(stderr,"No rows or
columns!\n");
    exit(1);
  }
```

What the heck is that 'ospeed' variable? Let me try to explain. Some older terminals required a delay after each special operation (like moving the cursor, clearing the screen, etc.) before more data could be sent. The delays are in the form of "pad characters" which are sent out for a certain duration of time. How long are the pad characters sent to the terminal? That's what ospeed determines. ospeed stands for Operating SPEED of the terminal, and represents the baud rate. Under OS-9, advanced program-mers will recall that baud rates are given values 0 through 16 (5=300, 7=1200, 10=2400, 14=9600, etc.). You can obtain the value for the user's terminal by making a call to _gs_opt() and looking at _sgs_baud. Or, you can do what I did, and make the assumption that the user is using a relatively modern terminal that doesn't require padding and set ospeed to -1. With ospeed set to -1, no padding is sent to the users terminal during special operations.

Finally, we just need to check to make sure our generic pointer hasn't gone past the end of our terminal capabilities buffer (tcapbuf)! This is an awkward error, since if it has gone past the buffer, there's no telling what it could have overwritten in our program. The best thing is just get out quickly.

```
  if (ptr>=&tcapbuf[TCAPSLEN])
  {
    fprintf(stderr,"Terminal description too
big!\n");
    exit(1);
  }
```

This takes care of initialization! Our tcapbuf now contains all our needed terminal capability strings and our pointers are initialized. In the next issue, I'll explain how to do I/O using our termcap example program, so be sure to keep the all your 68'Micros issues on hand!

Boisy Pitre recently sent me some source code to help K-Windows programmers determine if the user is, in fact, running on a K-Windows terminal. He also included another routine to return the edition number of K-Windows on the system. The latter routine will be of interest to non K-Windows as well, since it can (with few changes) work for any module in memory. My thanks to Boisy for allowing me to reprint this code here!

```
#include <stdio.h>
main()
{
  if (isKWindowsDevice(1) == 1) {
    printf("stdout is a KWindows
device!\n");
  } else {
    printf("stdout is NOT a KWindows
device!\n");
  }
  printf("Windio #%d\n",
KWindowsEdition());
}
/*
 * this function returns 1 if the
 * current path is open
 * under a K-Windows device,
 * else it returns 0
 *
 * Boisy G. Pitre — 12/8/94
 */
#include <module.h> isKWindowsDevice
(path)
int path;
{
  char device[32], driver[8];
  mod_dev *descAddr;
  mod_dev *modlink();
 /*get the name of the path's device*/
  if (_gs_devn(path, device) == -1) {
    /* failed to get device */
    return(0);
  }
  /* attempt to link to path's device
     ONLY if it's a descriptor */
  if ((descAddr = modlink(device,
mktypelang(MT_DEVDESC,
ML_ANY))) == -1) {
    /*failed to link to path's device*/
    return(0);
  }
  munlink(descAddr);
  /* unlink to be nice */
  /* copy driver string and check
     to see if it's windio */
  strncpy(driver, (int)descAddr + descAddr-
>_mpdev, 6);
  driver[6] = '\0';
  if (strucmp(driver, "WINDIO") != 0) {
    return(0);
  }
 /*it's most likely a K-W window*/
  return(1);
}
int strucmp (str1, str2)
register char *str1, *str2;
{
  register char *p;
  char *index();
  if ((p = index (str1,'\n')) != NULL)
*p ='\0';
  if ((p = index (str2,'\n')) != NULL)
*p ='\0';
  while (toupper (*str1) == toupper (*str2)
&& *str1 && *str2) {
    ++str1;
    ++str2;
  }
  if (*str1 == *str2)  return(0);
  if (*str1 > *str2)  return(1);
  return(-1);
}
/*
* this function returns the edition number
* of windio or -1 if the windio driver cannot
* be found
*
* Boisy G. Pitre — 12/8/94
*/
int KWindowsEdition()
{
  struct modhcom *windioDrv;
  struct modhcom *modlink();
  /* attempt to link to windio driver */
  if ((windioDrv = modlink("windio",
mktypelang(MT_DEVDRVR, ML_ANY)))
== -1) {
    /* failed to link to windio driver */
    return(-1);
  }
  munlink("windio");
/* unlink to be nice */
  return((int)windioDrv->_medit);
}
```

< 268'm >

Any comments, questions, or source code to be included in Joel's column may be sent in care of 68'Micros or directly to Joel at:

Joel Mathew Hegberg
936 N. 12th Street
Dekalb, IL 60115
E-mail : joelhegberg@delphi.com

Programming in "C"

Time for a little STRUCTURE!

P.J. Ponzo

If we wish to keep a record of, say, friends ... including name, address, birthdate, etc. we might declare:

```
char name[20], address[40], birthdate[15];
```

where name, for example, is an array of 20 characters which is meant to hold the name of one such friend, and address holds 40 chars, etc.

If we want 100 such records, we could use:

```
char name[100][20], address[100][40], birthdate[100][15];
```

where name[0], name[1],name[2], etc. are each arrays of 20 characters, etc.

We would print our list via a statement like:

```
for (i=0; i<100; i++) printf("\n%s %s %s",name[i],address[i],birthdate[i]);
```

Since the birthdate has the form "Nov 6, 1934" (for example) we would need to extract the last number (1934), and perform a subtraction, in order to determine his (her?) age ... and would need all such numbers in order to deduce the average age of our friends. So we might declare birthdate to be a trio of objects: an array of characters (to hold the birth-month, like "Nov"), an integer (to hold the birth-day) and a second integer (to hold the birth-year) ... that way we could perform some arithmetic on the integer parts of the birthdate.

```
char name[100][20], address[100][40];
char birth_month[100][4];
int birth_day[100], birth_year[100];
```

The above would do it. Sufficiently many arrays (of characters and integers) for 100 friends, each with names of 19 characters (or less) and addresses of 39 characters (or less) and 3 characters for the birth_month (we'll need the terminating '\0' in each char array!)... and 100 birth_days and birth_years (remember that birth_day[0] to birth_day[99] is 100 birth-days!). It would be nice to have a DATA TYPE which held one such record, with name, address, etc. and mixed chars and ints !!

Let's welcome the STRUCTURE ...

We invent a structure called date which includes a 3 character month (hence month[4]) and two ints (day and year). We are accustomed to saying int x; and char y;, meaning that x is of DATA TYPE int and y is of DATA TYPE char. SO, we will want to say date birth; meaning that birth is of DATA TYPE date meaning a collection of objects: a char month[4], an int day and an int year.

But, if date is a structure (with the elements mentioned above) then we will refer to it as struct date, to inform the compiler that date is no ordinary guy but is, in fact, a struct ... with all the rights and privileges thereto appertaining!

SO, we declare birth to be such a structure (called date) via: struct date birth;
see? struct date go together!

So why wouldn't we just define a structure called birth, which has the 3 members: month[4] and int day and int year ?? Because we will want to use this struc for other dates, like: struct date death;
Too morbid? Then how about:
struct date WhenWeMet;

Now, WhenWeMet is of DATA TYPE date too, containing the same three elements of char month[4] and int day and int year !! (Note: some compilers may not recognize names like WhenWeMet but may restrict names to, say, 8 characters or less).

Inside a STRUCTure

Before we see how to define such a structure as date, we use it! It will have 3 members: month, day, year. We declare:

```
struct date birth[100], WhenWeMet[100];
```

so each of birth[0], birth[1], etc. and WhenWeMet[0], WhenWeMet[1], etc. are structs of TYPE date. We refer to birth[i].month, birth[i].day and birth [i].year and to WhenWeMet [i].month, WhenWeMet[i].day and WhenWeMet [i].year for i=0, 1, 2, ... up to the number of friends we have (and each reflects the 3 members month, day and year of the date structure ... OK?)

To input all this information we might use:

```
printf("\n How many friends do you have");
scanf("%s",&number);
for (i=0; i<number; i++) {
  printf("\n For friend %d",i);
     printf("\n Enter Month of Birth ");
scanf("%s",&birth[i].month);
     printf("\n Enter Day   of Birth ");
scanf("%s",&birth[i].day);
     printf("\n Enter Year  of Birth ");
scanf("%s",&birth[i].year);
}
```

```
printf("\n How many friends do you have");
scanf("%s",&number);
```

Here we ask for the number of friends, and store it in number (note the &number!):

```
printf("\n How many friends do you have");
scanf("%s",&number);
for (i=0; i<number; i++) {
```

Then, we go through each of number of friends, asking questions:

```
for (i=0; i<number; i++) {
  printf("\n For friend %d:",i+1);
```

We remind the user which friend we're working on by printfing ...

For friend 1: then For friend 2: etc.

(We don't refer to "friend 0:" ... that would be insulting ...so we print the numbers i+1 rather than i).

```
  printf("\n For friend %d",i);
     printf("\n Enter Month of Birth ");
scanf("%s",&birth[i].month);
```

We ask Enter Month of Birth and put the answer into &birth[i].month (for the ith friend) and, as required by scanf(),we use the &ddress !

```
printf("\n Enter Month of Birth ");
scanf("%s",&birth[i].month);
     printf("\n Enter Day   of Birth ");
scanf("%s",&birth[i].day);
     printf("\n Enter Year  of Birth ");
scanf("%s",&birth[i].year);
```

... and so on, for the birth.day and birth.year. Unfortunately, this won't (quite) work ... did you see why?

We'll repeat the program excerpt:

```
printf("\n How many friends do you have");
scanf("%s",&number);
for (i=0; i<number; i++) {
  printf("\n For friend %d",i);
     printf("\n Enter Month of Birth ");
scanf("%s",&birth[i].month);
     printf("\n Enter Day   of Birth ");
scanf("%s",&birth[i].day);
     printf("\n Enter Year  of Birth ");
scanf("%s",&birth[i].year);
}
```

number should have a %d (for an integer)

birth[i].day should have a %d (for an integer)

birth[i].year should have a %d (for an integer)

... but there's something else ...

```
     printf("\n Enter Month of Birth ");
scanf("%s",&birth[i].month);
```

scanf() will put the 3 characters typed at the keyboard, say "Nov",

into the memory reserved for birth[i].month, but won't put in a '\0'! WE must put it in (... while we're praying that the user doesn't type november, which is much too long to fit into the 4 bytes we've reserved for the month!).

We could initialize all the bytes in the birth.month to '\0' via:

```
for (i=0; i<100; i++)     { /* for all 100 friends*/
   for (j=0; j<4; j++)  {/* for each of 4 bytes*/
      birth[i].month+j='\0'; /*set byte to '\0'*/
   }                  /* end of inner "for"  */
}                     /* end of outer "for"  */
```

... then (provided the user doesn't type more than a 3 character month!) we've got all the '\0' string terminators we'll need.

This little ritual is necessary because scanf() is meant as a general-purpose input ... ints and floats and chars etc. A special-purpose input ... just for strings of chars ... would be smart enough to append the '\0'.

The stdio.h library of C-functions will contain such a function. gets(&sam) will get a string and put it into the address &sam. Just like scanf() requires a pointer to the memory location where the input is to be stored, so does gets(). We write:

```
1 printf("\n How many friends do you have "); scanf("%d",&number);
2 for (i=0; i<number; i++) {
3    printf("\n For friend %d",i);
4      printf("\n Enter Month of Birth "); gets(&birth[i].month);
5      printf("\n Enter Day   of Birth "); scanf("%d",&birth[i].day);
6      printf("\n Enter Year  of Birth "); scanf("%d",&birth[i].year);
7 }
```

and the gets() in line 4 will collect each chararacter typed, and when a \newline is typed (the Return or Enter key) it will exchange it for a '\0' ... and put everything into the memory location indicated.

Defining a STRUCTure

It's about time we defined our struct date:

```
struct date  {
    char month[4];
    int day;
    int year;
};
```

Note the structure of a structure:

```
struct name  {
    — all the —
    — members —
    — go here —
};
```

We give it a name (like date) so we can declare other objects to be of this DATA TYPE (remember birth and WhenWeMet ?)... and an opening and closing { and } ... and the various members (like char month[4], etc.)... and a final;

```
struct date  {
    char month[4];
    int day;
    int year;
};
```

This final is meaningful!

Because a struct date is to be used just like int or char, and because we usually say int x; or char x; (with a final ;), then we terminate a structure definition with a ; and expect to be able to say struct date {

```
          — etc —
     } x;
```

(note the similarity with int x; etc.)

SO, for our earlier example, we could say:

```
struct date  {
    char month[4];
    int day;
    int year;
} birth[100],WhenWeMet[100];
```

... and we've defined our struct date AND declared birth[ ] and WhenWeMet[ ] to be such structures ... all at once!

Now let's return to the record of our "friends", which includes nameand address as well as some dates. For each "record" we will define another structure ... let's call it "record" (what else?)

```
struct record  {
    char name[20];
    char address[40];
    struc date birth;
    struc date WhenWeMet;
} friend[100];
```

Note that we've not only defined the struct called record but we've also declared 100 such structures, using the "final" friend[100]; friend[0] and friend[1] and friend[2] etc. are ALL of type record hence contain members name, address, birth and WhenWeMet.

The first two (name and address) are character arrays BUT the last two (birth and WhenWeMet) are ... SURPRISE (!) structures of TYPE date !!!

We now have two structures defined:

```
struct date  {            struct record  {
  char month[4];              char name[20];
  int day;                  char address[40];
  int year;                   struc date birth;
};                     struc date WhenWeMet;
                            } friend[100];
```

Earlier we declared 200 structures of TYPE date (namely birth[100]and WhenWeMet[100]). We also referred to the 3 members of birth[47] (for example) as birth[47].month, birth[47].day and birth[47].year.

Now we won't need to define these 200 structures (sorry about that!) since struct date birth and struct date WhenWeMet are embedded in the record structure ... structures within structures!! STRUCTures within STRUCTures ?? NOBODY SAID THIS WAS EASY!

```
struct date  {            struct record  {
  char month[4];              char name[20];
  int day;                  char address[40];
  int year;                   struc date birth;
};                     struc date WhenWeMet;
                            } friend[100];
```

To input (for example) the name of the friend[47], we'd say:

```
printf("\n  Name please : "); gets(&friend[47].name);
```

... and ... to input the birth.month we'd say:

```
printf("\n Month of Birth : "); gets(&friend[47].birth.month);
```

Note the use of gets() (to automatically append the '\0'). Note, too, the very logical way we refer to the member of a structure within a structure... friend [47].birth.month:

```
main()  {       /* not-too-useful-program*/
  int number, i;
  struct date  {
      char month[4];
      int day;
      int year;
  };
  struct record {
      char name[20];
      char address[40];
      struct date birth;
  } friend[100];
     printf("\nHow many friends : "); scanf("%d",&number);
  for (i=0; i<number; i++) {
     printf("\n For friend %d",i+1);
           printf("\nName ? "); gets(&friend[i].name);
           printf("\nAddress ? "); gets(&friend[i].address);
        printf("\nMonth of birth ? "); gets(&friend[i].birth.month);
        printf("\nDay of birth ? "); scanf("%d",&friend[i].birth.day);
        printf("\nYear of birth ? "); scanf("%d",&friend[i].birth.year);
  }

     printf("\nSUMMARY of your %d friends",number);
  for (i=0; i<number; i++) {
     printf("\nName:%s",friend[i].name);
   printf("\nAddress:%s",friend[i].address);
     printf("\nBorn on %s %d,%d",
   friend[i].birth.month,friend[i].birth.day,
friend[i].birth.year);
  }
}
```

giving a (typical) printout:

```
Name:Peter Ponzo
Address:49 Margaret S., Waterloo, Ont.
Born on Nov 6,1934
```

POINTERS to STRUCTURES!

One sometimes feels frustrated in writing an elaborate function which does the most wonderful things, only to get from such a function a single int or float or char (you can't return(a,b,c,d,e) at the end of the function, but only return(a) (!@#$%). BUT, the function can create an elaborate structure which houses all the wonderful things, then

ROMY-16

EPROM EMULATOR

- Emulates ROMS (2716-27010) or RAMs
- in 8- and 16- bit systems.
- Window/menu driven interface. Provides 8 hardware breakpoints for 8-
- bit systems.
  $195 (2716-27256) or **$245** (2716-
- 27010), 90 day warranty.
- 15 day money-back guarantee. Optional assembler, disassembler, and ROM debugger, add $100.

Universal Microprocessor Simulator/Debugger V2.1

- Simulates Z8, Z80, 64180, 8048, 8051 8085, 6800, 6801, 6805, 6809, 68HC11,
- 6303, 6502 & 65C02. Assembler, Disassembler, & Windowed
- Symbolic Simulator. Supports on-board debug through RS232.
  $100 each CPU [illegible]

6809 Single Board

- Supports 8K RAM, 8K ROM.
- Two 6821 PIAs connect 32 bits of I/O to outside world.
- No jumpers for 2732, 2764, and
- 6116.
- Two interrupt signals on CPU bus.
- Size is 2.75"x5". **$60** each board. For an integrated development system with assembler, disassembler, and on-board debugger please add $70.

68HC11 Microcontroller Development System

- Eight channel 8-bit A/D converter. 32K ROM and 32K RAM.
- $120 each SBC, to complete with assembler, disassembler, BASIC inter-preter and on-board debugger add $70

J&M Microtek, Inc.
83 Saman Road, W Orange, NJ 07052
Tel: 201-325-1892 Fax: 201-736-[illegible]

return(a) where a is a pointer to the structure!

In fact we've used such a function ... one which returns a pointer. Before I tell you which function it is (from the stdio.h library), let's see how such a function should be declared.

Consider: char f(); which declares f() to be a function which returns a char. Then, to declare a function which returns a pointer to a char it would be sensible to use the format: char *f(); ... right?

SO ... if the function f() were to return a pointer to a structure called sam, we'd declare it with:
sam *f(); ... right?

And what if sam was typedefined to be a structure, as in:
typedef SomeStructure SAM; (where we have agreed to use capitals)
Then the function f() would be declared:
SAM *f(); ... right?
NOW ... remember when we used:
FILE *fopen(); ???

In fact, when we fopen() a file on a disk, the operating system returns a pointer to a structure (called FILE) and this structure contains all the wonderful things we need to know about the file ... and that's why we also declare this pointer fp: FILE *fp, *fopen(); and say (subsequent to the above declaration): fp=fopen(); so we assign to fp the pointer returned by fopen().

Then, when we want to get a character from this file, we need only pass to getc() this pointer (as in getc(fp)) and now the getc() function will be able to extract all the wonderful things it needs.. from the structure!

NOW, if sam *f(); is the way we declare f() to be a function which returns a pointer to an object of TYPE sam (which could be an int or a float or a struct etc.), then how should we declare f to be a pointer to a function which returns an object of TYPE sam ????: sam (*f)();
says that f is now the pointer ... because we used (*f) ...and the thing it points to is *f (remember that int *x; declares x to be a pointer to an int, and the int is *x) SO, since (*f) is to be a function, we say (*f)()... and we've seen this curious notation before too!

Suppose that ptr is a pointer to a structure (declared using *ptr so that (*ptr) IS the structure itself). Suppose the structure had a member called name.
Then we'd refer to this member as: (*ptr).name
(as in (*ptr).name="George";)

Another (simpler) notation for the same thing is:
ptr -> name (use this only if ptr is a pointer)

... and if the structure had a member birth which was itself a structure containing a member called month, then we can use the notation:
ptr->birth.month (which means (*ptr).birth.month)
as in ptr->birth.month="May";

AND, if birth happened to be a pointer too, we'd use:
ptr->birth->month

MORAL???
Use sam.birth if sam is a structure.
Use sam->birth if sam is a pointer to a structure.

< 268'm >

P.J.Ponzo
Dept. of Applied Math
Univ. of Waterloo
Ontario N2L 3G1
CANADA

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 Convertor 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

Basic09 In Easy Steps

Chris Dekker

Optimizing you programs.

As every programmer knows there are many ways to reach a certain goal and although all of them get the job done (we hope!!), not all methods are equal in elegance and processing speed. Elegance in this respect means that a program should have a straight forward and logical design, without lots of twists and turns. This not only looks better; it usually means that you have an easier time debugging your code and, just as important, it makes it a lot easier to maintain the code later on.

Now, I know writing elegant code is not that easy; especially for beginners. I, too, sometimes have to suppress the desire to rewrite functional code because it looks nowhere near as good as it did some time ago. So,, if things don't go smoothly at first: don't despair; practise still makes (almost?) perfect.

Having said all that, the best way to avoid problems is not to get into bad habits in the first place. Unfortunately DECB is not a good environment for getting into the right habits. Most notably because you can not build up a program as a collection of separate modules as you can with Basic09.

This doesn't seem a big deal up front, but the larger your programs get; the more likely it is you loose track of a variable, which ends up holding the wrong number, leading to unreliable results, etc. The worst part is that if you discover this after a year or so, you may spend a week trying to track down GOTOs, GOSUBs, etc. and generally trying to find out what happens where.

If you consider this the thrill of a lifetime: go ahead, make your day. As long as you can keep your programs running there is nothing really wrong with it. Personally, though, I consider it a waste of my time if I have to spend that much time trying to track down an error. I think I should be able to track down a faulty module within a minute, at least 90% of the time.

My perspective on these matters may be slightly different than that from most of you. For instance, the source code for Level II graphics and CoCoTop combined runs over 600K. The amount of source code I have written for the various packages and personal use is around 2 Megabytes (not counting old versions) and still expanding.

All together this is a lot of software to maintain. My main point, though, is that as little as a few years ago I didn't have the faintest idea it would balloon like this. The same thing might happen to (some of) you, so you better get it right from the start.

So far I have talked only about the "why"s. I guess it is eminently more useful for you to talk about the "how"s and "what"s. So here goes. The first thing you should do is patently low-tech: sit down with a piece of paper and write down what you want the program to do and how you expect to accomplish this. This is called flow charting.

Your flow chart doesn't have to be such a beautiful design that you need a computer to draw one: chances are no one will ever lay a hand (or eye) on it anyway. It is, however, an important tool to visualize how the various portions of your program will interact. Generally speaking this process helps most by identifying which chunks of code will be used over and over again.

This allows you to start by writing a separate module for each of these chunks. The two main advantages of this approach are: A) you have to debug this code only once and B) if you have to make a change in that code you will have to do this only once, too, to implement the change throughout the program.

I know that you can achieve the same effect by using subroutines, but with modules you have the added advantage of dealing with local variables. A local variable is a variable that is only visible to the module it belongs to. This way you don't have to worry about running out of variable names; using a name twice, so you get your signals crossed, etc. Some examples of this type of modules are: modules for file access, for handling screen functions, keyboard input, menus, etc.

Now that you have these special modules in place (or at least laid out) you start thinking about the next layer. These are usually modules that call the aforementioned modules. In a record keeping program you could write a module to add records, one to edit them, a module to sort records, etc.

At the top of our little pyramid would be your main module. Sometimes this module is referred to as the primary module although, technically speaking, under Basic09 this is incorrect. This main module usually carries the name of your program. It can best be used for things like handling the main menu (if there is one), opening paths to various devices and most of the error handling. Of course it also contains the code to call the other modules that make up your program.

I find that this way of structuring programs works very well for larger efforts: it is easy to maintain and expand. Of course if you want to write a small utility, setting up a program in this way is a waste of time and effort. Just where to draw the line is a matter of personal taste and tends to become a lot clearer after some practise (and mistakes).

Coming back to our flow charts: there are two mistakes you can easily (but shouldn't) make. A) don't draw your chart five minutes before you start typing in the code. Personally I like to do such a job a few days or a week in advance. This gives you some time to let the design sink in and it is also easier to make changes to the overall design if you haven't written a byte of code yet. Remember a big mistake at this level WILL come back to haunt you. B) Don't think that you are gifted enough to do without a decent design. Sooner or later we all reach a level where, if you don't do your "homework", you can only screw up. With a lot of extra code (and effort) the program may be semi-reliable, but that is still nothing to brag about.

Is starting out right all you can do to optimize a program? No, there are lots of things to keep in mind when writing the actual code. For instance Y=X^2 gives the same result as Y=X*X but it takes your CoCo about 40 times as long to figure it out. Not all operations can be sped up that dramatically, but it does drive the point home.

There are actually quite a lot of little improvements to be made along those lines. I could spell them out for you here, but they are all neatly grouped together in chapter 12 of your Basic09 manual so you might as well read that. There you will find about 3 pages worth of tips plus a table ranking the execution speed of the various operations. After that it is up to you to implement them.

That leaves for this article just a few tips you won't find in the manual. For one thing the manual correctly states that quickest way to transfer data between data structures, arrays, etc. is to use the LET assignment operator. Or, in case you want to transfer data to or from a disk, to use the PUT and GET statements.

These methods work great mainly on account of reduced processing overhead. The only drawback is that LET, GET and PUT transfer entire arrays in one block. But what if your buffer is only half full? Well, you could use a loop to transfer the contents record for record. This works fine if your records are 100 bytes or more long. But for short records (not to mention byte transfers) processing overhead is big enough to substantially slow down your program.

Fortunately there are a few escape hatches. The above mentioned restrictions are built into Basic09, not OS-9 itself. As a substitute for the LET command you can use the F$move system call or, and this may be easier, the ML subroutines presented in part 8 of this series.

To get around the GET/PUT restrictions we use another system call. This call (I$read or I$write (whichever one is appropriate) allows us to transfer bytes to and from a device without checking what we are actually transferring. This last feature is very important to avoid system crashes. Although setting up these system calls is a little extra work they allow you to control the size of the transfer down to the last byte.

First you must set up a data structure for the 6809's registers (as I have shown you before); then open a path to a file and replace a GET statement by the following code:

```
    regs.a=path
    regs.x=ADDR(buffer)
    regs.y=?? (number of bytes you want to
read)
    RUN syscall($89,regs) (I$read call)
    IF LAND(regs.cc,1)=1 THEN
    ERROR regs.b \ ENDIF
```

The only other consideration is that your software has to calculate a valid value for regs.y. This depends on what your program needs to do and shouldn't be too hard to figure out. If an error occurs, this routine will report it to Basic09 in the same way as the GET statement so no special precautions are necessary there.

To replace the PUT statement: use the same routine but change $89 to $8A, this is the code for I$write. OS-9 has two other system calls for data transfer ($8B and $8C) but they correspond to Basic09's READ (INPUT) and WRITE statements (performing checks on the data) so they are not all that useful in raw data transfers.

Another area where you can get some more speed, for certain programs as much as 5%, is to avoid calling gfx2. This utility matches the commands you give it with the appropriate codes and performs error checking on the variables you want to pass. However if you don't mind doing some extra work up front, your program will do fine without it.

The extra work usually amounts to setting up a data structure or a simple BYTE array to hold the codes and values you want to send to the device driver. This device driver is the part of OS-9 that actually executes your commands and it doesn't care whether it gets those commands from gfx2 or from your program, as long as gets the right values.

All of this sounds very difficult, but is surprisingly easy to implement. For instance you want to close an overlay window. You can do so with the command RUN gfx2("owend") or you can use the following code:

```
  DIM owend:INTEGER
  owend=$1B23 \ PUT #1,owend
```

In both cases the result will be the same: your overlay window closes or %@#!! #195; meaning there was no overlay window to begin with. The reason is simple enough: if gfx2 gets a "owend" command it will send two bytes ($1B and $23) via stdout (path #1) to the device driver (exactly what we did).

By the same reasoning RUN gfx2(path,"owend") translates into PUT #path, owend. As an added bonus you will actually conserve memory if you use more than two "owend" statements in your program, because you have to initialize owend only once.

There are a number of other codes that you can set up in the same way. For instance cursor ON/OFF, reverse video ON/OFF, etc. The actual codes for this can be found in the windows section (in the back) of your OS9 manual.

All the way at the other end of the scale for this way of accessing the screen is the following example. I use it frequently because I find it the most reliable way to convert the type of a window. Basically it replaces the following commands:

```
  RUN gfx2("dwend")
  RUN gfx2("dwset",2,0,0,80,24,1,0,0)
  RUN gfx2("select")
```

Theoretically you should be looking at an 80 column screen after executing these commands, but that doesn't always happen. For some reason there are times that the computer's hardware and software no longer agree with each other after these commands. OS9 insists that you are looking at a 80 column text window, while the CoCo's hardware happily displays a 40 column text window. I am not sure why this happens but I do know it is very annoying because, generally speaking, your program crashes in a hurry. The most reliable way to avoid this mess seems to be to send your escape codes (as these commands are usually called) directly to your screen driver. The same way as I described above. Your code looks something like this:

```
  DIM switch(14):BYTE; i:INTEGER
  FOR i=1 TO 14
  READ switch(i)
  NEXT i
  PUT #1,switch
  DATA 27,36,27,32,2,0,0,80,24,1,0,0,
  27,33
```

As you can see this code does the same thing as the above commands. 27,36 = $1B24 = dwend; 27,32 = $1B20 = dwset; 27,33 = $1B21 = select. The other codes, to set up the new window, are the same as in the "dwset" command for gfx2.

Note that the length of "switch" is critical here: it MUST be 14 bytes. A device driver reacts to everything it gets (or doesn't get) and you could easily lock up your computer by sending the wrong number of bytes.

As a byline: there is a chance that this approach is not 100% reliable either. Generally this occurs when your program tries to access the new window right after the PUT statement. This seems to be caused by the hardware falling behind the processor. Your screen gets updated 60 times per second. This is very fast by our standards but to your CoCo that is once per 30000 clock cycles in which it can do a lot of work.

So far I have been successful solving this problem by inserting one line of code right after the PUT statement. This code puts your program to sleep until the screen has been updated:

```
  regs.x=2 \RUN syscall($0A,regs)
```

Well,, that's about it in this neck of the woods. You can find the codes to replace other gfx2 commands in the windows section of your OS9 manual. You will find equivalents there for all basic graphics and windowing functions, along with the number of bytes you have to send to the device driver and their meaning. Till next time, enjoy your CoCo.

< 268'm >

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

Chris Dekker
RR #4
Centreville, NB E05 1H0
CANADA

Don't forget to see his software ad in the advertiser's section!

AUTO BOOT OS-9 FROM POWER-ON *Steve Hilton*

Burn a 27C256 EPROM to replace your CoCo 3 ROM with changes to autoload OS-9 from power-on

I wanted to auto-boot without taking away anything from the original RS DECB system. So, with that in mind, I came up with the idea to replace the area of CoCo 3 ROM used by the graphic data of the 3 Microware programmers (Sorry, boys!).

Track 34 of your OS-9 system disk holds the modules REL, BOOT, and OS9P1. This code is loaded by the "DOS" command into location $2600. The code is entered and after a bit of housekeeping is RELocated to its rightful position, $ED00. OS9P1 then takes over and loads the rest of the boot from your system disk using the BOOT module.

As you may know, the CoCo 3 ROM code uses the CTRL and ALT key depression when powering up to display the Microware boys. I took over the code for this test and directed the system to jump to BASIC when it saw the CTRL and ALT keys depressed. Without the keys being pressed, the system jumps to our ROM based OS-9 bootloader.

The actual code changes to ROM are small, as listed below, but you must arrange to get your $1200 bytes of track 34 into position starting at $4405 (This is where the data for the 3 amigos started). The code changes for making a new ROM are:

Relative Addr	Original	New
4038	C3	D6
4039	6C	05
40B9	27	10
40BA	07	27
40BB	31	01
40BC	3F	33
40BD	26	31
40BE	F2	3F
40BF	16	26
40C0	01	F0
40C1	2E	12
41F0	4F	30
41F1	B7	8C
41F2	FE	B5
41F3	ED	CC
41F4	97	43
41F5	71	C8
41F6	B7	ED
41F7	FF	00
41F8	DE	16
41F9	C6	FE
41FA	09	C7
4405	Here begins the Track 34 data.	4F
		53
		20
		5B
		12

CONTINUES -

Replacing the ROM in your Color Computer 3 might be a little much for some people. You must use a desoldering tool to take out the old ROM. (This can be simplified by cutting the unit out first and then using a desoldering tool to clean up). How ever you do it, be careful not to damage the foil traces.

Install a 28 pin socket to hold the EPROM. Put your boot disk in /D0 and close the door. Turn on the power and watch as the screen will flash green for just a split second and then the blue screen with "OS9 BOOT" will appear.

What about track 34? No longer needed. I used the disk editor, DED, to replace the sectors back into the GAT table. I'm working on a patch to OS9GEN to skip the part where track 34 gets loaded.

If you don't own or have access to an eprom burner, I would be happy to make one for anyone wishing. You pay for the EPROM and the return shipping. 27C256 EPROMs cost about $5.00 and shipping would run under a dollar, (USMAIL). So, send $6.00 to:

Steve Hilton
612 Chateau Circle
Burnsville, MN 55337

You'll receive a standard version of the OS-9 bootloader. If you want a doctored up version, send me a message saying what changes you want. If it's just a couple of bytes, I can handle that, but I don't have the time to make major changes, and I can't be held responsible if your changes don't work as anticipated. Good luck!

Infocom Games on the MM/1! *Boisy Pitre*

I used to be a big Infocom game fan when I had my CoCo and used DECB (long ago). When I moved to OS-9, I didn't play them anymore since doing so meant going back to DECB.

While fiddling around on chestnut, I ran across an Infocom game interpreter that Brian White had ported to run under OSK. After downloading it, I found that Infocom sells "Lost Treasures of Infocom" Vols. I and II, a compendium of their original games. After calling around a few local software shops, I located a place which was selling both volumes. Although I just picked up Vol II, there are a ton of games on both:

Vol I: Deadline, Suspect, MoonMist, Spellbreaker, Ballyhoo, Infidel, Hitchhiker's guide to the Galaxy, Starcross, the Witness, Planetfall, Suspended, Stationfall, Enchanter, The Lurking Horror, Zork I, Zork II, Zork III, Beyond Zork, Zork Zero, Sorcerer, Suspended

Vol II: A Mind Forever Voyaging, Bureaucracy, Cutthroats, Hollywood Hijinx, Plundered Hearts, Nord and Burt, Sherlock Holmes, Titan, Trinity, Wishbringer, Boarderzone, Seastalker

The store that carried them said that they are no longer a hot item. Vol I was reduced from $69.96 to $39.95. Vol II was reduced from $39.95 to $19.95.

All that was necessary was to insert the 3.5" MS-DOS disk in my MM/1, use pcf to copy the .DAT files from each of the four disks, and voila! I now have 12 games sitting in my GAMES/INFOCOM directory which can be played at my whim. The interpreter works flawlessly! I now have Infocom games on my MM/1, and it looks really cool.

(*editor:* I'm not sure as to the availability of the Infocom game packs listed above. This message was posted to Delphi in April of 1994. You may have to search a bit for the disks!)

micro notes

Many of you have wondered about **monitors that will work with the CoCo 3, MM/1, and an IBM clone.** Well, I ran across an ad for some refurbished, guaranteed .26 dot pitch **Sony Multiscans** that sync from 15.7KHz (CoCo, MM/1, CGA) to 36KHz (standard 480x640 VGA). Price is $175 plus shipping. These guys are also trying to get rid of some refurb CGA monitors. These will work with a CoCo if all you need is an 80 column display, but the colors are all wrong and you only get eight of them (plus black and white). Won't do for games at all!! Call 314-937-0335 or write Rich, Pikul & Associates, 101 Glenfield Drive, Festus, MO 63028. Shipping should be around $15 each.

Another bargain find: **Citizen 120D serial printers for only $40** ($15 for parallel port model)! These are decent nine pin printers with a 120 cps print speed, about the same as a DMP-130. The best thing is that they have an RS-232 serial port built in! You will need to make a cable, but that's pretty easy. No **serial/parallel convertor needed!!** The same company has great deals on Citizen 180D ($20 and up) and 200GX ($90, only parallel available) models also.

I haven't checked, but I believe the serial board for the 120D also fits the 180D, so a 180D serial should be $40-$50. Call Prescott Electronics, 1-800-298-6484 (714-492-6304 in CA). Reference their ad in "Compu-Mart".

The new Congress has a sticky problem... revising the 1976 copyright law to protect digitally recorded material. This will affect transmission and reproduction of digital material, and could have an affect on BBS systems.

The old law is hindered by the unwieldy "legalese" language it is written in. Also, it could not be foreseen in 1976 that the means for electronic transmission would be so widespread and easy to use for the general public.

The new law will close many loopholes in the old. It will also be illegal to digitally reproduce documents. A big problem is that the working groups advising lawmakers on changes do not truly understand the technology involved. Do any of us??

CompuServe has lowered it's fees! For those who frequent CIS this is good news. If you are currently on one of the other services, such as my much favored "Delphi" (second choice being GEnie), you won't be flocking over the CIS -- it is STILL the most expensive! The new rates are $9.95 per month (actually up $1) and $4.80 per hour for all connect speeds (down from $9.60).

Do you need to transfer files easily between a PC and your CoCo? I recently received a flyer from **Elite Software** for their **Elite*Xfer** software. This is a PC program that allows for easy transfer of files between a PC and CoCo (DECB only). I should know... I have (and regularly use) a copy!! Elite*Xfer will even work with a high density 5-1/4" floppy drive! And it formats CoCo disk also. The regular price for this software is $69.95, but Elite has reduced the price to $29.95 for a limited time! This offer should be good at least through February, so act now! **Elite also continues to offer:** Elite*Calc ($29.95), Elite*Word ($29.95), Elite*Word/80 ($29.95), Elite*Spel and Elite*Spel/80 ($14.95), and Elite*File ($29.95). All except the "/80" programs will run on any CoCo model. Elite*Calc uses a software screen to display over 32 columns, Word and File use the standard 32 column screens. All will run on a CoCo 3. You can order any four titles (including Elite*Xfer) for $89.95. To order call 1-800-745-8491 or write to: Elite Software, Box 11224, Pittsburg, PA 15238-0224. VISA and MasterCard are accepted.

Many of you have asked about the availability of **Puppo keyboard adapters** and/or circuit boards. Well, I have some good news! I have been in contact with the person who built these things for Bob Puppo, and he has **received permission to make more!** But I need at least six confirmed orders to get them into production! The cost will be **$70 each plus $3 S&H. For $100 total, you will get the adapter AND an 84 key XT keyboard** (so you don't have to worry about finding one)! The keyboard has 10 function keys to the left of the main keys and the arrow keys are not separate but are within the numeric keypad.

ANY XT compatible keyboard will work. The keyboard of choice is the 101 key models with a physical switch between XT and AT modes on the bottom, but MOST (I can't guarantee all!) "auto-switching" keyboards will work as well. If you are seriously interested, send at least a $20 deposit by the end of February. You will receive your adapter (and keyboard) no later than the end of April 1995.

"micro notes" is for news on anything that may be of interest to readers, personal classified ads, and new product releases. If you see anything that may be of interest to other readers, have something for sale, or need an item, please write in and let us know! Vendors: be sure and let us know when you have a new product even if you don't already advertise!!

WANTED

1. OS-9 version of Checkbook Plus with docs.
2. May thru November 1989 issues of Nine-Times disk magazine.
3. 2400 baud modem with docs and cable.
4. July 89 and newer Rainbow on Disk

L.T. Day
Box 32332
Columbus, OH 43232

NEW PRODUCTS *from* FARNA Systems!

CoCo Family Recorder/OS-9 1.0
If you are into genealogy, then the CoCo Family Recorder is the *absolute best* program for the CoCo 3. The OS-9 version is nearly identical to the DECB version in appearance, but takes advantage of many OS-9 features such as pop-up windows for entering data. DECB users can send their *original* CCFR disk (it will be returned) to get the OS-9 version for only $20.00. Others must pay the regular price of $28.50. Requires at least one 40 track double sided drive (FD-502) or larger. Can be shipped on 3.5" 720K disk if requested.

FARNA Systems
Box 321
Warner Robins, GA 31099
$2.50 shipping and handling per order. Canada S&H $4.00; Overseas $7.00

NOTICE: WE CAN NOW ACCEPT CREDIT CARDS!!! **Visa and MasterCard (only!) can now be accepted for payment. There is a 6% service charge and a minimum charge of $25. Cards are accepted through a third party and will be billed through "FS Printing".**

OS-9 Point Of Sale 1.0
Designed specifically for the small business that needs more than one check-out station but can't afford the $7,500.00 or more for an MS-DOS based system. Easy to use, menu driven software uses OS-9's multi-user/tasking features, eliminating the high cost of networks. Has all necessary features to replace your cash register and keep track of your sales and inventory automatically. Supports multiple serial ASCII terminals.Current price is only $62.50.

FARNA *Systems*

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

Software, Books, and Hardware for all OS-9/OSK Systems!

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
Sound recorder for CoCo3. Record any sound for easy play-back in your BASIC or M/L programs.

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

Mind Games - $7.50
Collection of 9 classic games. Run from included RAM disk w/512K.

Cross Road II - $7.50
Simple Tic-Tac-Toe, but with amazing sound and graphics! Sound recorded with Digi-Tech.

Space Intruders - $14.50
Looks just like Atari's classic "Sapce Invaders"! CoCo 1/2 and 3.
Donut Dilemma - $14.50
Climb, jump, and ride elevators to top of Donut factory to shut it down! 10 levels. CC 1,2,3.
Rupert Rythym - $14.50
Collect Rupert's stolen notes, then work our correct sequence. Great action adventure! **Get Space Intruders, Donut Dilemma, and Rupert Rythym for only $33.50! Save $10!**

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.*)

OS-9 Point of Sale - $62.50
Maintain inventory, print invoices, customer catalog, etc. Multi-user capable under Level II. Supports ASCII terminals. Basic09 required. Simple menu driven interface.

Books:

Tandy's Little Wonder - $22.50
140 page softbound book with history and technical info for all CoCo models. Schematics, peripherals, upgrades, modifications, repairs, much more- all described in detail! Vendors, clubs, BBSs also listed.

OS-9 Quick Reference Guides Level II (Revision 2) - $7.50
68K (based on 2.3) - $10.50
Get that bulky manual offyour desk! These handy QRGs have all the most needed information in a 5.5"x 8.5" desk-top size. Includes command syntax, error codes, special keys functions, etc.

CoCo Hardware:

DigiScan Video Digitizer - $150
Capture images from VCR, camcorder, or TV camera. No MPI required- uses joystick ports. CoCoMax3, Max 10, ColorMax 3 compatible. Special order- allow 90 days for delivery. Send $75 deposit.

Ken-Ton SCSI Hard Drive System and Components

Complete, ready to run, "plug and play" 85MB system. Top quality drive, case, and ps. Send how much space for DECB, OS-9 --- $550.00
No-Drive Kit: controller, OS-9 drivers, RGB-DOS in ROM, 2 pos. "Y" cable, and drive cable (specify type). Seagate N series drive with ROM rev. 104 or greater needed. ---- $250.00 (state how much of drive to be used for OS-9)

Controller only -------------------- $135.00
OS-9 Drivers ---------------------- $25.00
RGB-DOS (for DECB access) --------- $35.00
** $50 for RGB-DOS and OS-9 drivers when purchased together with a controller **
"Y" cable, $25 for two position, $35 for three.
Drive cables - specify direct to drive or SCSI case type connector ---------------- $25.00

Add $2.50 S&H per order. Canada/Mexico add $4.00; Overseas $7.00

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!

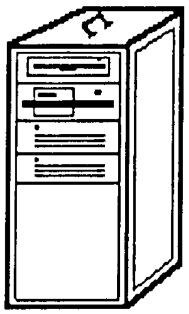
Mailing Service: If you send catalogs or letter correspondence to 200 or more persons at once, we can do all work for you for about the same cost of your materials alone! How much is your time worth???

Contact Frank Swygert at above address/phone for quotes

for all your CoCo hardware needs, connect with

CoNect

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



NEW PRODUCT: **The Cube**
This tower enclosure was designed specifically for the CoCo and peripherals, even an MPI if desired! Four drive bays, two will hold a pair of 3.5" drives sideways. Easy access, carry handle mounted on top!

Mini RS-232 Port: Don't let the name fool you! This is a full featured serial port, supporting the signals needed for flow control as well as the basic 4. Jumper blocks allow readdressing or swapping DSR/DCD. No custom cables or hardware widgets needed here! Y cable users will need to add $9.95 for a power supply. **$49.95**

Hitachification: CoNect will install a Hitachi 63B09E CPU and a socket into your CoCo. Machine MUST be in working condition! The 68B09E will be returned unharmed. 90 day limited warranty. Chip and installation only **$29.95**

REPAIRS: We can repair most damaged CoCos, even those with bad traces where a 68B09 was removed. Costs vary with damage. Bad 68B09 sockets repaired for only $40! **Inquire BEFORE sending your computer.**

NEW FOR 1995 FROM DISTO!

1."Inside 2-Meg": A techinical 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, MPROM 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 517-747-4851

SUB-ETHA SOFTWARE

By arrangement with **StrongWare**, Sub-Etha Software is proud to bring you...

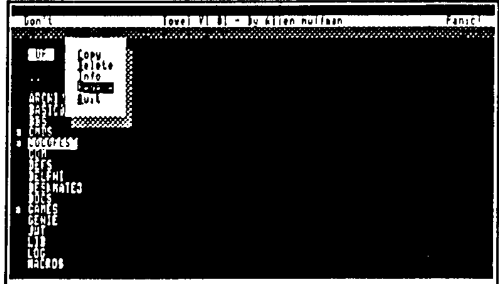
Soviet Bloc - Called the best Tetris(tm)-like game by many, this is a bigger version of the now-classic falling shapes puzzle game. RS-DOS Req: CoCo 3, Joystick/Orchestra-90 Pak optional ________ $19.95

GEMS - "Columns" of colors fall as you change the order of the colors. Match three in a row, column, or diagonal at the bottom and those colors disappear. Sounds simple, doesn't it? RS-DOS Req: CoCo 3, Joystick/ Orchestra-90 Pak optional ________ $24.95

Copy Cat - Simon says "match the sequence of tones as the colored diamonds flash". Great for building memory skills. RS-DOS Req: CoCo 3 ________ $ 9.95
OSK Req: MM/1 or 100% K-Windows Systems ________ $14.95

HFE (Hprint Font Editor) - A fantastic editor for those HPRINT fonts with lots of options. Create your own character set which you can LOADM and us in your own programs. Also creates and edits fonts compatible with MiniBanners! RS-DOS Req: CoCo 3, Joystick optional ________ $19.95

Font Collection - A collection of 18 useable HPRINT or MiniBanners fonts. (Importable to OS-9 for use with MiniBanners09 as well!) RS-DOS Req: ________ $ 9.95



Towel! V1.01 by Allen Huffman - NEW VERSION! A program no intergalactic hitchhiker should be without! Use mouse or keyboard hot-keys to perform common disk and file commands from pull-down menus. Tag multiple files for Copy, Delete, Rename, etc., and even point 'n click a Backup, Cobbler, Dcheck, or whatever. User definable menu for custom options. Runs under the EthaWin interface (included) on a high-speed text screen. All commands/colors configurable. OS9 Req: CoCo 3, OS-9 Level 2 ________ $19.95

Sub-Etha Software
P.O. Box 152442
Lufkin, TX 75915

Add $2.50 S&H. Texas residents add 8.25% tax. Write us for more info!

Sub-Etha Originals:

MultiBoot V1.03 by Terry Todd & Allen Huffman - Type "DOS" and bring up a list of up to sixteen OS9BOOT files! No more floppy swapping. A serious must-have for intense OS-9 users. OS9 Req: CoCo 3, OS-9 Level 2 ________ $19.95

1992 CoCoFest SIMULATOR V1.02 by Allen Huffman - NEW VERSION now uses compressed graphics and has scoring. Take a walking tour of the '92 Atlanta CoCoFest with this graphics adventure. 16-level digitized photos of the event and a text parser (ie, "get the box of disks") to let you interact. Runs on a 640x192 graphics screen. OS9 Req: 512K CoCo 3, OS-9 Level 2, 500K+ of Disk Space ________ $ 9.95
OSK Req: MM/1 or 100% K-Windows System ________ $14.95

Write-Right by Joel Mathew Hegberg - Featureful word processor for the MM/1 with what you would expect from a "real" word processor. What you see is what you get! ________ $54.95

Etha-GUI by Joel Mathew Hegberg - A neat program launcher for the MM/1 which includes handy desktop utilities like a phone dialer and nifty screen savers...even a trash can. Point and click icons to run programs. ________ $34.95

Quality OS-9 Software from

ColorSystems

NEW! K-Windows Chess for MM/1
Play chess on your MM/1..........................$24.95

NEW! 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

To order send check or money order to:

Color Systems
P.O. Box 540
Castle Hayne, NC 28429
(916) 675-1706

Call or write for a free catalog! Demo disks also available.
NC Residents please add 6% sales tax
Owned and operated by ZacK C. Sessions

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 ---------------------------- $20.00
Access your SmartWatch from DECB! New function added to access date/time from BASIC (DATE$). Only $15.00 with any other purchase!

RGBOOST -- $15.00
Make the most of your HD6309 under DECB! Uses new 6309 functions for a small gain in speed. Compatible with all programs tested to date! Only $10.00 with any other purchase!

Robert Gault
832 N. Renaud
Grosse Pointe Woods, MI 48236
313-881-0335
Add $4 shipping & handling per order

Summertime is "off" season for a lot of CoCoists. If you are one of those, look forward to new releases and upgrades this fall. If you use your CoCo all year 'round, the following titles are currently available:

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 •••
RR #4 Centreville, NB
E0J 1H0, CANADA
Phone 506-276-4841

User-friendly Level II Programs!

Peripheral Technology Specials

486SLC/33MHz Motherboard w/CPU	$139.00
486SLC/50MHz IBM, ISA,CPU, 0K	$199.00
486SLC/66MHz IBM, VESA, CPU, Math	$299.00
IBM boards - Made in USA - 3YR warranty	
1MB SIMM 70ns DRAM	$47.00
356MB Samsung IDE Drive	$229.00
420MB Connor IDE Drive	$275.00
546MB Maxtor IDE Drive	$379.00
IDE/Floppy/Serial/Parallel	$24.95
1.44MB TEAC Floppy	$49.95
Mini Tower, 200W, LED readout	$79.00
Panasonic Dual Speed CD ROM	$169.00
VGA Card ET4000-1MB, 1280x1024	$99.00
VGA Monitor WEN .28mm 1024x768	$249.00

UPS Ground $7.00 on most items except Tower & monitor ($12.00 UPS Ground).

1250 E. Piedmont Rd. 404/973-2156
Marietta, GA 30062 FAX: 404/973-2170

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 Send manual or RomPak to prove ownership	$25.00
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!' 256/512 byte sectors, multipak support	$25.00

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 TTY/ANSI/VT100/K-Windows/Binary Emulation	$50.00
Upgrade from TasCOM (Send TasCOM manual please)	$30.00

7 Greenboro Cres
Ottawa, ON K1T 1W6
CANADA
(613)736-0329

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

Internet mail: cmckay@northx.isis.org

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.

ADVERTISER'S INDEX:

NOTE:
Volume 2 numbers 4 and 5 issues of microdisk will be delivered on the same disk following this issue.

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 ACRTC 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

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.

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

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