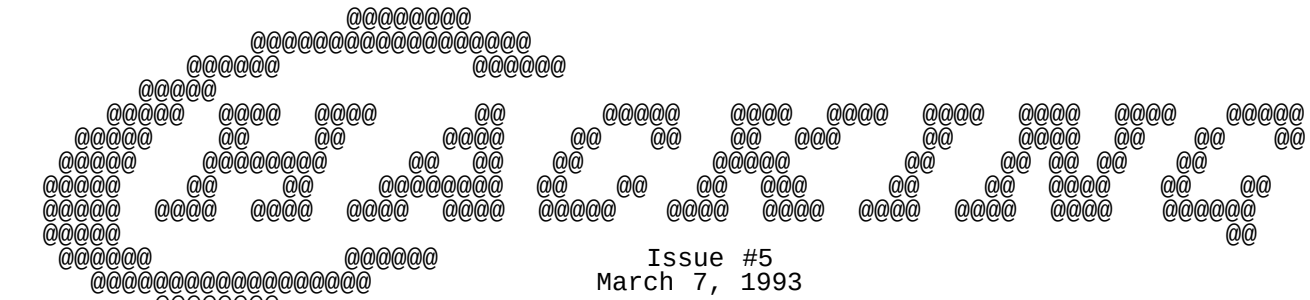

Issue #5
March 7, 1993

Editor's Notes:
by Craig Taylor

It seems that each issue of C= Hacking has always began with a "Sorry, It's late but here it is message." - Well, this one will start out again like that - This issue was originally scheduled to be out the middle of January but due to several delays in obtaining articles and my delaying trying to debug the multi-tasking source code it's been held up until now.

My apologies to the authors who have had their articles into me on time - school is coming first for me and having to do a lot of coding for several classes was the major contributing factor to the delays.

Now, after the apologies are out of the way - Let's take a look at what has happened since last time I wrote.

- RUN magazine is no longer with us.

As one of the last hold-outs I was expecting RUN magazine to keep on printing until the Commodore 64/128's really did die out but apparently the publishers decided it wouldn't be so. This leaves the Twin Cities magazine as the only US magazine in publication for the Commodore (6502 based) computers that I am aware of. Speaking of Twin Cities (not sure if he's combining the 64/128 or just coming out with seperate Twin Cities magazines) does anybody know or have any information on when the next issue will be out? Or has my lastest issue just not been sent out?

As I was writing this I got the latest issue of Twin Cities which has expanded to C=64 coverage also. The new issue looks very nice, about 53 pages of so of good decent material. I'd recommend get a subscription for those of you who are looking to still hear about new Commodore products.

I'd like to get people's reactions on the demise of RUN and what will people will think will probably be the main source of information for C= owners. A lot of people reading this magazine are on the comp.sys.cbm newsgroup but I'm wondering about individuals who do not have access to such a newsgroup and do not have access to the internet. Let me know what you think - hopefully through a friend w/ access to the internet. Sort of a catch-22 I guess.

- A Mail-Server has been setup to automate sending issue requests.

The full details of how to use the Mail-Server is in a documentation file contained within but this mail-server (whose source code is available for anyone who wishes to see it written in VAX DCL code) also allows file requests which will be uuencoded and sent to you. I am trying to have all of the programs in each issue available via request as for some people it is a minor pain trying to extract and compile the programs contained within.

- I saw a note recently that the speed-up board work was still being done.

Does anybody know anything further about this? I'm interested in this and how it would be carried out / done but aside from an occasional post here and there about it I actually hear very little.

- There is also work on an Ansi C compiler being done.

Recently a group of people (about 9 currently) are working on a C compiler for the C=64 and C=128 which will eventually support the full ANSI C library. A large list of extensions have been proposed and the compiler will probably be released as either shareware or possibly, public domain.

Ack! - This magazine keeps growing. The last issue was approx. somewhere around 3000 lines, this one is just a tad over 6000. I'm sure that we're not suffering the quality just because of the

quantity. :-) Be sure to take a look at the previous back issues available via the Mail-Server and don't be afraid to suggest comments or suggestions. While usually the authors are too busy to take ideas for new programs we always welcome to hear how useful you find certain programs included herein etc.

Also I am looking for articles on any type of software project, hardware project or general theory articles that you would like to submit. Just leave me a message via email at "duck@pembvax1.pembroke.edu". Note also that I've just signed up for a GENIE account and can be reached there via C.TAYLOR37 once my account is approved.

==========

Please note that this issue and prior ones are available via anonymous FTP from ccosun.caltech.edu under pub/rknop/hacking.mag in addition to the mailserver which is documented in this issue.

==========

==========

In this Issue:

==========

Mail-Server Documentation

by Craig Taylor (duck@pembvax1.pembroke.edu)

What is a mail-server?

A mailserver is an automated job that will scan my mail file for messages with a subject line of "MAILSERV" and will then automatically carry out certain operations within the body of the mail message. This makes it easier on me and you. Easier for me so that I don't have to deal with 50+ messages each month asking for files to be sent out and also insures that your files that you requested will be sent within 24 hours. In addition it allows

files to be more easily sent and accessed in case you are not able to extract the source files from C= Hacking.

If you have FTP access please see the Editor's Notes at the start for information on R. Knop's FTP site and how to access it as you may find using that somewhat quicker to use.

How to use the mail-server / What it is.

This mail-server is intended to help me keep track / more easily update my mailing list of individuals who wish to sub-scribe or get back-issues of C= Hacking mailed to them.

To use it simply send a message to "duck@pembvax1.pembroke.edu" (me) with a subject line of "MAILSERV" and then with one of the following commands in the body of the mail message:

Currently the following commands are supported:

```
  help              - sends current documentation f file list
  send iss<number>. - sends issue # (1-4 currently). Remember the period!!
  subscribe         - subscribe to the mailing list automatically
 *subscribe catalog - subscribes to a list that will be sent out
                      everytime the catalog changes.
  catalog           - show list of available source /uuencoded binaries
  psend name        - send uuencoded binary.
```

Commands no longer supported:

```
  status            - returns the current commands (this list)
                      (use the help file)
```

Please note that the mailserver is only run at 2:00 AM EST.

Catalog List - Last update February 27, 1993.

```
  iss1.                 - C= Hacking, Issue #1
  iss2.                 - C= Hacking, Issue #2
  iss3.                 - C= Hacking, Issue #3
  iss4.                 - C= Hacking, Issue #4
  iss5.                 - C= Hacking, Issue #5
  contents.lis          - Content Listing of Issues #1-4
  mailserv.012493       - VAX/DCL Mailserver .share file

 *invasion1.sfx         - Space Invasion Source (Starting with Issue 4)
 *bmover.sfx            - Geos 128 Banking with Banks 2f3 (Issue #2)
 *vdc-bg.sfx            - Use of 64K VDC RAM in Geos (Issue #3)
 *c64.zip               - C64 Emulator for IBM
 *lrr.sfx               - Little Red Reader (from C= Hacking #4)

  -- Temporary Files -> Or files that will be deleted as needed for space

 *zedv075.sfx           - Zed-128 Text Editor
 *ramdosii.sfx          - REU Dos for the C=128 Allows > 512k REU.

  NOTE: Files marked with "*" should be requested via PSEND - they will be
        sent to you in uuencoded form. They may _not_ be requested via SEND.
```

==========

The Demo Corner: Stretching Sprites
by Pasi 'Albert' Ojala (po87553@cs.tut.fi)
Written: 16-May-91 Translation/Revision 01-Jun-92, Dec-92

(All timings are in PAL, principles will apply to NTSC too)

You might have heard that it is possible to expand sprites to more than twice their original size. Imagine a sprite scroller with 6-times expanded sprites. However, there is no need to expand all of them equally. Using this technique, it is possible to make easy sinus effects and constantly expanding and shrinking letters.

The VIC (video interface controller) may be fooled in many things. One of them is the vertical expansion of sprites. If you clear the expand flag and then set it back straight away, VIC will think it has only displayed the first one of the expanded lines. If we do the trick again, VIC will continue to display the same data again and again. But why does VIC behave like this ?

Logic gates will tell the truth

It is not really a bug, but a feature. The hardware design to implement the

vertical enlargement was just as simple as possible. Those, who do not care about hardware should skip this part... The whole y-enlargement is handled with five simple logical ports. Each sprite has an associated Set-Reset flip-flop to tell whether to jump to the next sprite line (add three bytes to the data counter) or not.

Let's call the state of the flip-flop Q and the inputs R (reset) and S (set). The function of a SR flip-flop is quite simple: if R is one, Q goes to zero, if S is one, Q goes to one. Otherwise the state of the flip-flop does not change. In this case the flip-flop is Set, if either the Y-enlargement bit is zero or the state of the flip-flop is zero at the end of a scan line. The flip-flop is reset, if both the state and the Y-enlargement are ones at the end of the line.

When you clear the bit in the vertical expansion register, the flip-flop will be set regardless of the electron beam position on the scan line. If you set the bit again before the end of the line, the flip-flop will be cleared and VIC will be displaying the same sprite line again. In other words, VIC will think that it is starting to display the second line of the expanded sprite row. This way any of the lines in any of the sprites may be stretched as wanted.

```
 .---- Current flipflop state (if one, enables add to sprite pointer)
 |  .---- Y-expansion bit.
 |  |  .--- End of line pulse (briefly one at end of line)
 |  |  |  .--- Next state (What state will become under these conditions)
 |  |  |  |
 0  0  0  1
 0  0  1  1
 0  1  0  no change
 0  1  1  1
 1  0  0  1             Clear $D017 -> flip-flop is set
 1  0  1  1
 1  1  0  no change     Set $D017   -> flip-flop resets at the end of line
 1  1  1  0
```

So, simply, at any time, if vertical expand is zero, the add enable is set to one. At the end of the line - before adding - the state is cleared if vertical expand is one.

Even odder ?

Something very weird happens when we clear the expansion bit right when VIC is adding three to the sprite image counters. The values in the counters will be increased only by two, and the data is then read from the wrong place.

Normally the display of a sprite ends when VIC has shown all of the 21 lines of the sprite (the counter will end up to $3f). If there has been a counter mixup, $3f is not reached after 21 lines and VIC will go on counting and will display the sprite again, now normally. If we fool the counter only once, the counter value $3f is reached when the sprite is displayed twice.

Fiddling

I don't think the distorted counter effect can be used for anything, but there is many things where the variable stretching could be used. When you open the borders, you can be sure that there is a constant amount of time, if you stretch the sprites to the whole lenght of the area. You may stretch only the first and last lines, stretch the other lines by a constant or using a table, or using a variable table or any of the combinations possible.

A raster routine is a must

Because you have to access the VIC registers on each line during the stretch, you need some kind of routine which can do other kinds of tricks besides the stretch. You can open the side borders and change the background color and maybe you have to shift the screen (and the bad lines with it) downwards. [See previous C=Hacking Issues for talk about raster interrupts.]

Look at the demo program. In the beginning of the raster routine there is first some timing, then a loop that lasts exactly 46 clock cycles. It takes exactly one scan line to execute. Inside the loop we first do the necassary modifications to the vertical scroll register, then we change the background color and then we open the side borders. And finally we handle the stretching using the stretch data, where a zero-bit means that the corresponding sprite will be stretched. A one-bit means that VIC is allowed to go to the next line of the sprite data.

Stretching takes time

Besides showing the stretched sprites we need time to generate the stretching data, unless of course, the stretch is constant. We have to have 20 one-bits for each sprite in our table. It is not feasible to determine the state of each byte in the table, instead you clear the table and plot the needed bits.

The routine is quite straightforward, but many optimizations may be applied to make it faster. First we load Y with the stretch of the first line (the y-coordinate of the data). Then we use it as an index to the table and plot the right bit and increase Y with the expansion value. Then we do it again until we have all of the 20 bits scattered to the table. The last sprite line will then stretch until we stop the stretching, because the last line is not allowed to be drawn.

Speed is everything

The calculation itself is easy, but optimizing the routine is not. If all of the sprites are stretched equally (by integer amounts) and from the same position, the routine is the fastest possible. You can also have variable and smooth stretch. Smooth stretch uses other than integer expansion values and thus also needs more processor time. If each sprite has to be stretched individually, you need much more time to do it.

The fastest routine I have ever written uses some serious selfmodification tricks. There are also some other tricks to speed up the stretch, but they are all secret ones.. :-) Well, what the h*ck, I will include it anyway. By the time you read this I have already made a faster routine..

You can speed up that routine (by 17%) by unrolling the inner loop, but you have to use a different addressing mode for ORA (zero-page). You also need to place some restrictions to the tables used.. If you unroll both loops, you can get ~25% faster routine than the Fore!-version.

Demo program

I tried to collect all of the main principles of stretching and raster routines to the demo program. I use the term "raster routine" when the execution is tightly synchronized to the electron beam and to the screen display. The program may be unclear in places, but I wanted to keep it as short as possible. The routine opens the side borders, scrolls the screen vertically, changes the background color and stretches the sprites.

The stretcher routine allows different y-position and amount of expansion for each sprite. This routine uses 1/8 fractions to do the counting, and so it is much too slow to use in a real demo. VIC registers are initialized from a table, instead of setting them separately. Interrupt position is one line above the sprites. The program does not open the top or bottom borders. (I usually use a NMI to open the vertical borders, so that I only need to use one raster-IRQ position.)

I tried to make a NTSC version, but I couldn't get it to synchronize. There are also less cycles available so you can't stretch all of the sprites individually in NTSC (with this routine that is..).

Fast-stretch from Megademo92 (part: Fore!)

```
SINPOS          Stretch sinus index
SINSPEED        Stretch sinus index speed
YSINPOS         Y-sinus index
YSINSPEED       Y-sinus index speed
MASK            Bit mask for passess (usually $01,$02,$04,$08,$10..)

YSINUS          Y-sinus table
STRETCH         Sprite line sizes   (LSB of the address must be 0)
SIZET           Sprite size/2 table (LSB of the address must be 0)
DATA            Stretch data table (cleared before this routine)
```

[xx] marks selfmodification. For example loop counter, bit mask and index to the stretch and size data tables are stored straight in the code.

```
0b90    lda #$06        ; Number of sprites-1 (here I used only 7 sprites)
0b92    sta $0b96
0b95    ldx #$[ff]      ; Load counter
0b97    clc             ; Clear carry for adc
```

```
0b98    lda SINPOS,x    ; Stretch sinus position
0b9b    sta $0bd1       ; Set low bytes of indices
0b9e    sta $0bb8
0ba1    adc SINSPEED,x  ; Add stretch sinus speed (carry is not set)
0ba4    and #$7f        ; Table is 128 bytes (twice)
0ba6    sta SINPOS,x    ; Save new sinus position
0ba9    lda YSINPOS,x   ; Get the Y sinus position
0bac    adc YSINSPEED,x ; Add Y sinus speed
0baf    sta YSINPOS,x   ; Save new Y sinus position
0bb2    tay             ; Position to index register
0bb3    lda YSINUS,y    ; Get Y-position from table (can be 256 bytes long)
0bb6    sec             ; adc either sets or clears carry, we have to set it
0bb7    sbc SIZET[1e]   ; Subtract size of the sprite/2 to get the sprite
0bba    clc             ;  to stretch from the middle.
0bbb    tay             ; MaxSize/2 < Y-sinus < AreaHeight-MaxSize/2
0bbc    lda MASK,x      ; Get the ora-mask for this pass
0bbf    sta $0bcb       ; Store mask
0bc2    sta $0bdb
0bc5    ldx #$13        ; 19 lines here + 1 after
0bc7    lda DATA,y      ; Load & ora-mask & store
0bca    ora #[$01]
0bcc    sta DATA,y
0bcf    tya
0bd0    adc STRETCH[1e],x ; Add the stretch from the table (carry is not set)
0bd3    tay
0bd4    dex             ; decrease counter
0bd5    bne $0bc7       ; Do the 19 lines
0bd7    lda DATA,y      ; Load & ora-mask & store the 20th line
0bda    ora #[$01]
0bdc    sta DATA,y
0bdf    dec $0b96       ; Next sprite(s)
0be2    bpl $0b95
0be4    rts
```

Timings:
```
-------
clear 128 bytes: 514  + 12 cycles       8.16 lines
7 passes       : 3820 + 12 cycles       60.6 lines = 8.66 lines/pass
```

The unrolled clear routine consists of one load (lda #$00) and 128 store instructions (sta $nnnn). 12 cycles are counted for JSR/RTS.

Stretching of 8 sprites would take slightly less than 80 lines, which is one fourth of the total raster time. Displaying a 128-line high stretcher takes about 130 lines (counting sprite setup and synchronization), scroller couple of lines more. Total 212 lines leaves 100 lines (6300 cycles) free for other activities in a PAL system. In a NTSC system you would have only 50 lines left.

A simple basic routine to create the stretch data:
```
-------------------------------------------------
a=0:for f=0 to 127:a=a+Height*(2+sin(f*PI/64)):poke Table+f,a:
poke Table+f+128,a:a=a-int(a):next f
```

This will also handle the 'rounding'. Because of this we don't have to handle fractions in the stretcher routine. The use of a table also gives the opportunity to have a separate size for each sprite line. The table does not need to be a sinus, it could have triangle or any other 'waveform' as long as the minimum value in the table (sprite line size) is 1.

A basic routine to do the size/2 table:
```
--------------------------------------
a=0:for f=0 to 19:a=a+peek(Table+f):next f: rem get the size in position 0
for f=0 to 127:poke STable+f,a/2:a=a-peek(Table+f)+peek(Table+f+20):next f
```

Stretcher program

```
YSCROLL= $CF00 ; Vertical scroll table (moves bad lines)
STRETCH= $CF80 ; Stretch table
COLORS=  $CE80 ; Table for background colors
YCOORD=  $0380 ; Sprite y-positions (eight bytes)
HEIGHT=  $0388 ; Sprite stretches   (eight bytes)
YPOS=    52    ; Sprite y-coordinate
SPRCOL=  2     ; Sprite colors


*= $C000
```

```
        SEI             ; Disable interrupts
        LDA #$7F
        STA $DC0D       ; Disable timer interrupts
        LDA #<IRQ       ; Our own interrupt handler
        STA $0314
        LDA #>IRQ
        STA $0315
        LDX #$3E        ; We create a sprite to cassette buffer
LOOP    LDA SPRITE,X
        STA $0340,X
        DEX
        BPL LOOP
        LDX #7
LOOP2   LDA #$D         ; Set the sprite image pointers
        STA $07F8,X
        LDA #SPRCOL     ; Set sprite colors
        STA $D027,X
        DEX
        BPL LOOP2
        LDX #$26
LOOP3   LDA VIDEO,X     ; Init VIC
        STA $D000,X
        DEX
        BPL LOOP3
        LDX #$7F        ; Create the y-scroll table
LOOP4   TXA             ;  and clear the color table
        AND #$07
        ORA #$10        ; Non-blank screen
        STA YSCROLL,X
        LDA #$00
        STA COLORS,X
        DEX
        BPL LOOP4
        STA $3FFF
        LDX #23         ; Create a color table
LOOP5   LDA BACK,X
        STA COLORS+8,X
        STA COLORS+32,X
        STA COLORS+56,X
        STA COLORS+80,X
        STA COLORS+96,X
        DEX
        BPL LOOP5
        JSR CHANGE      ; Init sprite sizes and y-positions
        CLI             ; Enable interrupts
        RTS

IRQ     LDX #$01
        LDY #$08        ; 'normal' $D016
        NOP             ; Timing
        NOP
        NOP
        BIT $EA         ; (Add NOP's etc. for NTSC)
LOOP6   LDA YSCROLL-1,X ; Move the screen (bad lines)      5
        STA $D011                                          4
        LDA COLORS,X    ; Load the background color        4
        DEC $D016       ; Open the border                  6
        STA $D021       ; Set the background color         4
        STY $D016       ; Screen to normal                 4
        LDA STRETCH,X   ; Stretch the sprites              4
        STA $D017                                          4
        EOR #$FF                                           2
        STA $D017                                          4
                        ; (Add NOP for NTSC     +2)
        INX             ; Increase counter                 2
        BPL LOOP6       ; Loop 127 times                 + 3
                                                         ---
        LDA #1          ; Ack the raster interrupt       =46
        STA $D019                                        +17(sprites)
                                                         ---
        JSR DOSTRETCH   ; New stretch                    =63(whole)

        JMP $EA31

SPRITE  BYT 0,0,0,3,$FB,0,7,$7E          ; An Example sprite
        BYT 0,$35,$DF,0,$1D,$77,0,$B7
        BYT $5D,0,$BD,$83,$7E,$EF,1,$DE
        BYT $BB,1,$78,$AE,3,$70,$EB,0
        BYT 0,$BA,3,$60,$EE,3,$D8,$FB
        BYT 2,$F6,$FE,$83,$BD,$9F,$BA,0
        BYT $37,$EE,0,$3D,$FB,0,7,$7E
```

```
        BYT 0,3,$DF,0,0,0,0

VIDEO   BYT $E8,YPOS,$20,YPOS,$50,YPOS,$80,YPOS,$B0,YPOS
        BYT $E0,YPOS,$10.YPOS,$40,YPOS,$C1,$18,YPOS-1,0,0
        BYT $FF,8,$FF,$15,1,1,$FF,$FF,$FF,0,0,0,0,0,0,0,1,10
        ; Init values for VIC - sprites, interrupts, colors

BACK    BYT 0,$B,$C,$F,1,$F,$C,$B   ; Example color bars
        BYT 0,6,$E,$D,1,$D,$E,6
        BYT 0,9,2,$A,1,$A,2,9

DOSTRETCH
        LDX #31             ; Clear the table
        LDA #0              ; (Unrolling will help the speed,
LOOP7   STA STRETCH,X       ;  because STA nnnn,X is 5 cycles
        STA STRETCH+32,X    ;  and STA nnnx is only 4 cycles.)
        STA STRETCH+64,X
        STA STRETCH+96,X
        DEX
        BPL LOOP7
        STA REMAIND+1       ; Clear the remainder
        LDA #7
        STA COUNTER+1       ; Init counter for 8 loops
        LDA #$80
        STA MASK+1          ; First sprite 7, mask is $80
COUNTER LDX #$00            ; The argument is the counter
        LDY YCOORD,X        ; y-position
        LDA HEIGHT,X        ; Height of one line (5 bit integer part)
        STA ADD+1
        LDX #20             ; Handle 20 lines
LOOP8   LDA STRETCH+2,Y
MASK    ORA #$00
        STA STRETCH+2,Y     ; Set a one-bit
        STY YADD+1
REMAIND LDA #0
        AND #7              ; Previous remainder
ADD     ADC #0              ;  add to the height
        STA REMAIND+1       ; Save the new value
        LSR
        LSR
        LSR
        CLC                 ; Take the integer part
YADD    ADC #0
        TAY                 ; New value to y-register
        DEX
        BNE LOOP8
        LSR MASK+1          ; Use new mask
        DEC COUNTER+1       ; Next sprite
        BPL COUNTER

CHANGE  LDA #$00
        ASL                 ; Sprite height changes with 2x speed
        AND #$3F
        TAY                 ; 64 bytes long table
        INC CHANGE+1        ; Increase the counter
        LDX #7              ; Do eight sprites
LOOP9   LDA SINUS,Y
        LSR
        LSR
        CLC                 ; Use the same sinus as y-data
        ADC #8
        STA HEIGHT,X        ; Sprite height will be from 1 to 3 lines
        TYA
        ADC #10             ; Next sprite enlargement will be 10 entries
        AND #$3F            ;  from this
        TAY
        DEX
        BPL LOOP9
        LDX #7
        LDA CHANGE+1
        AND #$3F
        TAY
LOOP10  LDA SINUS,Y         ; Y-position
        STA YCOORD,X
        TYA
        ADC #10             ; Next sprite position is 10 entries from this one
        AND #$3F
        TAY
        DEX
        BPL LOOP10
        RTS
```

```
SINUS   BYT $20,$23,$26,$29,$2C,$2F,$31,$34 ; A part of a sinus table
        BYT $36,$38,$3A,$3C,$3D,$3E,$3F,$3F
        BYT $3F,$3F,$3F,$3E,$3D,$3C,$3A,$38
        BYT $36,$34,$31,$2F,$2C,$29,$26,$23
        BYT $20,$1C,$19,$16,$13,$10,$E,$B
        BYT 9,7,5,3,2,1,0,0,0,0,0,1,2,3,5,7
        BYT 9,$B,$E,$10,$13,$16,$19,$1C
```

Stretching sprites demo program basic loader (PAL)

```
1 S=49152
2 DEFFNH(C)=C-48+7*(C>64)
3 CH=0:READA$,A:PRINTA$:IFA$="END"THENPRINT"<clr>":SYS49152:END
4 FORF=0TO31:Q=FNH(ASC(MID$(A$,F*2+1)))*16+FNH(ASC(MID$(A$,F*2+2)))
5 CH=CH+Q:POKES,Q:S=S+1:NEXT:IFCH=ATHEN3
6 PRINT"CHECKSUM ERROR":END
100 DATA 78A9648D1403A9C08D1503A23EBD96C09D4003CA10F7A207A90D9DF807A9029D, 3614
101 DATA 27D0CA10F3A226BDD5C09D00D0CA10F7A27F8E0DDC8A290709109D00CFA9009D, 3897
102 DATA 80CECA10F08DFF3FA217BDFCC09D88CE9DA0CE9DB8CE9DD0CE9DE0CECA10EB20, 5281
103 DATA 67C15860A201A008EAEAEA24EABDFFCE8D11D0BD80CECE16D08D21D08C16D0BD, 4699
104 DATA 80CF8D17D049FF8D17D0E810E0EE19D02014C14C31EA00000003FB00077E0035, 3394
105 DATA DF001D7700B75D00BD837EEF01DEBB0178AE0370EB0000BA0360EE03D8FB02F6, 3628
106 DATA FE83BD9FBA0037EE003DFB00077E0003DF00000000E834203450348034B034E0, 3015
107 DATA 3410344034C118330000FF08FF150101FFFFFF00000000000000010A000B0C0F, 1859
108 DATA 010F0C0B00060E0D010D0E060009020A010A0209A21FA9009D80CF9DA0CF9DC0, 1876
109 DATA CF9DE0CFCA10F18D4DC1A9078D35C1A9808D45C1A200BC8003BD88038D51C1A2, 4314
110 DATA 14B982CF09009982CF8C5AC1A900290769008D4DC14A4A4A186900A8CAD0E24E, 3430
111 DATA 45C1CE35C110CDA9000A293FA8EE68C1A207B99EC14A4A1869089D880398690A, 3474
112 DATA 293FA8CA10ECA207AD68C1293FA8B99EC19D800398690A293FA8CA10F1602023, 3622
113 DATA 26292C2F313436383A3C3D3E3F3F3F3F3F3E3D3C3A383634312F2C292623201C, 1654
114 DATA 191613100E0B090705030201000000000001020305070 90B0E101316191C0000, 296
200 DATA END,0
```

Uuencoded C64 exutable for stretching sprites (PAL)

```
begin 644 stretch.64
M`0@-"`$`4[(T.3$U,@`F"`(`EJ5(*$,ILD.K-#BJ-ZPH0[$V-"D`40@#`$-(?
MLC`ZAT$D+$$ZF4$D.HM!)+(B14Y$(J>9(I,B.IXT.3$U,CJ``(@(!`"!1K(P/
MI#,Q.E&RI4@HQBC**$$D+$:L,JHQ*2DIK#$VJJ5(*,8HRBA!)"Q&K#*J,BDI:
M*0"I"`4`0TBR0TBJ43J74RQ1.E.R4ZHQ.H(ZBT-(LD&G,P#!"`8`F2)#2$5#F
M2U-532!%4E)/4B(Z@``."60`@R`W.$$Y-C0X1#$T,#-!.4,P.$0Q-3`S03(S3
M14)$.39#,#E$-#`P,T-!,3!&-T$R,#=!.3!$.41&.#`W03DP,CE$+"`S-C$TP
M`%L)90"#(#(W1#!#03$P1C-!,C(V0D1$-4,P.40P,$0P0T$Q,$8W03(W1CA%4
M,$1$0SA!,CDP-S`Y,3`Y1#`P0T9!.3`P.40L(#,X.3<`J`EF`(,@.#!#14-!B
M,3!&,#A$1D8S1D$R,3="1$9#0S`Y1#@X0T4Y1$$P0T4Y1$(X0T4Y1$0P0T4Y1
M1$4P0T5#03$P14(R,"P@-3(X,0#U"6<`@R`V-T,Q-3@V,$$R,#%!,#`X14%%?
M045!,C1%04)$1D9#13A$,3%$,$)$.#!#14-%,39$,#A$,C%$,#A#,39$,$)$G
M+"`T-CDY`$(*:`"#(#@P0T8X1#$W1#`T.49&.$0Q-T0P13@Q,$4P144Q.40P4
M,C`Q-$,Q-$,S,45!,#`P,#`P,#-&0C`P,#<W13`P,S4L(#,S.30`CPII`(,@V
M1$8P,#%$-S<P,$(W-40P,$)$.#,W145&,#%$14)",#$W.$%%,#,W,$5",#`P<
M,$)!,#,V,$5%,#-$.$9",#)&-BP@,S8R.`#<"FH`@R!&13@S0D0Y1D)!,#`SN
M-T5%,#`S1$9",#`P-S=%,#`P,T1&,#`P,#`P,#!%.#,T,C`S-#4P,S0X,#,TX
M0C`S-$4P+"`S,#$U`"D+:P"#(#,T,3`S-#0P,S1#,3$X,S,P,#`P1D8P.$9&B
M,34P,3`Q1D9&1D9&,#`P,#`P,#`P,#`P,#`P,3!!,#`P0C!#,$8L(#$X-3D`<
M=@ML`(,@,#$P1C!#,$(P,#`V,$4P1#`Q,$0P13`V,#`P.3`R,$$P,3!!,#(PK
M.4$R,49!.3`P.40X,$-&.41!,$-&.41#,"P@,3@W-@##"VT`@R!#1CE$13!#0
M1D-!,3!&,3A$-$1#,4$Y,#<X1#,U0S%!.3@P.$0T-4,Q03(P,$)#.#`P,T)$G
M.#@P,SA$-3%#,4$R+"`T,S$T`!`,;@"#(#$T0CDX,D-&,#DP,#DY.#)#1CA#=
M-4%#,4$Y,#`R.3`W-CDP,#A$-$1#,31!-$$T03$X-CDP,$$X0T%$,$4R-$4LQ
M(#,T,S``70QO`(,@-#5#,4-%,S5#,3$P0T1!.3`P,$$R.3-&03A%138X0S%!C
M,C`W0CDY14,Q-$$T03$X-CDP.#E$.#@P,SDX-CDP02P@,S0W-`"J#'``@R`RJ
M.3-&03A#03$P14-!,C`W040V.$,Q,CDS1D$X0CDY14,Q.40X,#`S.3@V.3!!\
M,CDS1D$X0T$Q,$8Q-C`R,#(S+"`S-C(R`/<,<0"#(#(V,CDR0S)&,S$S-#,V*
M,S@S03-#,T0S13-&,T8S1C-&,T8S13-$,T,S03,X,S8S-#,Q,D8R0S(Y,C8R+
M,S(P,4,L(#$V-30`0PUR`(,@,3DQ-C$S,3`P13!",#DP-S`U,#,P,C`Q,#`PR
M,#`P,#`P,#`Q,#(P,S`U,#<P.3!",$4Q,#$S,38Q.3%#,#`P,"P@,CDV`$\-E
,`R`"#($5.1"PP```>
`
end
size 1362
```

Rob Hubbard's Music: Disassembled, Commented and Explained
by Anthony McSweeney (u882859@postoffice.utas.edu.au)

[Ed's Note: I questioned this article concerning copyright problems and he has assured me that it is legal to present it in entirity like this as it is past a certain # of years. Accordingly I'm presenting it and any concerns

should be taken up with him and not myself.]

Introduction:

How do you introduce someone like Rob Hubbard?? He came, he saw and he conquered the '64 world. In my estimation, this one man was resposible for selling more '64 software than any other single person. Hell! I think that Rob Hubbard was responsible for selling more COMMODORE 64's than any other person! I certainly bought my '64 after being blown away by the Monty on the Run music in December 1985. In the next few years, Rob would totally dominate the '64 music scene, releasing one hit after another. I will even say that some really terrible games sold well only on the strength of their brilliant Rob Hubbard music (eg. KnuckleBusters and W.A.R.).

So how did Rob achieve this success? Firstly (of course) he is a superb composer and musician, able to make the tunes that bring joy to our hearts everytime we hear them! (also consider the amazing diversity of styles of music that Rob composed). Secondly, he was able to make music which was suited to the strengths and limitations of the SID chip. Just recall the soundfx used at the beginning of Thrust, or in the Delta in-game music. Perhaps the biggest limitation of SID must be the meagre 3 channels that can be used, but most Hubbard songs appear to have four, five or even more instruments going (just listen to the beginning of Phantoms of the Asteriods for example... that's only one channel used!!). I could really go on for (p)ages identifying the outstanding things that Rob Hubbard did, so I will finally mention that Rob's coding skills and his music routines were a major factor in his success.

The First Rob Hubbard Routine:

Rob Hubbard created a superb music routine from the very first tune which was released (Confuzion). Furthermore, Rob used this routine to make music for a very long time, only changing it _slightly_ over time. The sourcecode that I present here was used (with slight modifications) in: Confuzion, Thing on a Spring, Monty on the Run, Action Biker, Crazy Comets, Commando, Hunter Patrol, Chrimera, The Last V8, Battle of Britain, Human Race, Zoids, Rasputin, Master of Magic, One Man & His Droid, Game Killer, Gerry the Germ, Geoff Capes Strongman Challenge, Phantoms of the Asteroids, Kentilla, Thrust, International Karate, Spellbound, Bump Set and Spike, Formula 1 Simulator, Video Poker, Warhawk or Proteus and many, many more! All you would need to do to play a different music is to change the music data at the bottom, and a few lines of the code.

This particular routine has been ripped off by many famous groups and people over the years, but I don't think that they were ever generous enough to share it around. Can you remember The Judges and Red Software?? They made the famous Red-Hubbard demo, and used it in Rhaa-Lovely and many of their other productions. I'm sure that the (Atari) ST freaks reading this will love Mad Max (aka Jochen Hippel), and remember the BIG demo which featured approx 100 Rob Hubbard tunes converted to the ST. Although I hate to admit it, I decided to start sharing around my own sourcecode after receiving the amazing Protracker sourcecode (340K!) on the Amiga (thanks Lars Hamre). That made me shameful to be selfish, especially after I learned alot of from it. Why don't YOU share around your old sourcecodes too!

The particular routine that is included below was ripped from Monty on the Run, and it appeared in memory from $8000 to about $9554. The complete routine had code for soundfx in it, which I have taken out for the sake of clarity. Although the routine is really tiny - a mere 900 or 1000 bytes of code, there are some amazingly complex concepts in it which require alot of explanation if you don't know much about computer music or SID. Fortunately for you, I have put in excellent label names for you, and also alot of really helpful and amazing comments. In fact, I think this sourcecode must have a much better structure and comments than Rob Hubbard's original!!! I think that the best way to understand the sourcecode is to study it, and figure out what is going on using the comments.

In addition to the comments in the source, there are *3* descriptions of the routine in this article. The first tells you how to use the music routine when it's viewed as an already assembled 'module'. The second goes through an overview of the music and instrument data format, and is great for getting an overall feel for what the code is doing. The third description looks at the various sections of the code, and how they come together.

How to use the sourcecode:

```
    jsr    music+0 to initialize the music number in the accumulator
```

```
    jsr    music+3 to play the music
    jsr    music+6 to stop the music and quieten SID
```

  The music is supposed to run at 50Hz, or 50 times per second. Therefore PAL users can run the music routine off the IRQ like this:

```
    lda    #$00     ; init music number 0
    jsr    music+0
    sei             ; install the irq and a raster compare
    lda    #<irq
    ldx    #>irq
    sta    $314
    stx    $315
    lda    #$1b
    sta    $d011
    lda    #$01
    sta    $d01a
    lda    #$7f
    sta    $dc0d
    cli
loop =*
    jmp    loop     ; endless loop (music is now playing off interrupt :-)

irq =*
    lda    #$01
    sta    $d019
    lda    #$3c
    sta    $d012

    inc    $d020    ; play music, and show a raster for the time it takes
    jsr    music+3
    dec    $d020

    lda    #$14
    sta    $d018
    jmp    $ea31
```

  If this method is used on NTSC machines, then the music will be running at 60Hz and will sound much to fast - possibly it might sound terrible. I'm afraid you'll have to put up with this unless YOU are good enough to make a CIA interrupt at 50Hz. As I havn't had to worry about NTSC users before, perhaps someone will send me the best way to do this...

[Ed. Note: You could also keep a counter for the IRQ and don't execute it every 6 interrupt. This will make it the right speed although the best solution is for modifying the CIA to 50Hz like he mentions above.]

How the music data is arranged:
****

1. The music 'module' contains one or more songs.

  Each RH music is made up of a 'bunch' of songs in a single module. Thus the 'module' can have the title music, in-game music, and the game-over music all using the same playroutine (and even the same instruments :). The source that appears below only has the one song in it, and the music number is automatically set to 0 as a result (line 20). The label 'songs' is where you want to look for the pointers to the songs if you want to change this.

2. Each song is made up of three tracks.

  We all know that there are only 3 channels on the SID chip, so there are also 3 tracks - one for each channel. When I said 'pointers to the songs' above, I was therefore referring to 'pointers to the three tracks that make up the song'...hence we are looking at the label 'songs' again. Each track needs a high and low pointer, so there are 6 bytes needed to point to a song.

3. Each track is made up of a list of pattern numbers

  Each track consists of a list of the pattern numbers in the order in which they are to be played. Here we are looking at the labels 'montymaintr1' and 'montymaintr2'and 'montymaintr3'. Therefore I can tell you that the initial patterns played in this song are $11, $12 and $13 on channels 1,2 and 3 respectively. The track is either ended with a $ff or $fe byte. A $ff means that the song needs to be looped when the end of the track is reached (like the monty main tune), while a $fe means that the song is only to be played once. The current offset into the track is often called the current POSITION for that track.

4. A pattern consists of a sequence of notes.

A pattern contains the data that says when the notes should be played, how long they should be played for, at what pitch, with what instrument, should there be ADSR, should there be bending (portamento) of the notes etc. Each pattern is ended with a $ff byte, and when this is encountered, the next pattern in the track will be played. Each note has up to a 4 byte specification.

- The first byte is always the length of the note from 0-31 or 0-$1f in hex. You will notice that the top three bits are not used for the length of the note, so they are used for other things.
- Bit#5 signals no release needed. - Bit#6 signals that this note is appended to the last one (no attack/etc). - Bit#7 signals that a new instrument or portamento is coming up.

- The second byte is an optional byte, and it holds the instument number to use or the portamento value (ie a bended note). This byte will be needed according to whether bit#7 of the first byte is set or not...ie if the 1st byte was negative, then this byte is needed.
  - If the second byte is positive, then this is the new instrument number.
  - If the second byte is negative, then this is a bended note (portamento). and the value is the speed of the portamento (except for bits #7 and #0) Bit #0 of the portamento byte determines the direction of the bend.
    - Bit#0 = 0 then portamento is up.
    - Bit#0 = 1 then portamento is down.

- The third byte of the specification is the pitch of the note. A pitch of 0 is the lowest C possible. A pitch of 12 or $C(hex) is the next highest C above that. These pitches are denoted fairly universally as eg. 'C-1' and for sharps eg. 'D#3'. Notice that this routine uses pitches of higher than 72 ($48) which is c-6 :-)

- The fourth byte if it exists will denote the end of the pattern. ie. If the next byte is a $ff, then this is the end of the pattern.

NOTE: I have labelled the various bytes with numbers for convenience. Bear in mind that some of these are optional, so if the second byte is not needed, then I will say that the pitch of the note coming up is the 'third byte', even though it isn't really.

Okay, here are some examples:

eg. $84,$04,$24 means that the length of the note is 4 (from the lower 5 bits of the first byte), that the instrument to use is instrument number 4 (the second byte, as indicated by bit #7 of the first byte), and that the pitch of the note is $24 or c-3.
eg. $D6,$98,$25,$FF means that the length of the note is 22 ($16), that this note should be appended to the last, that the second byte is a portamento (as both 1st and 2nd bytes -ve!), that the portamento is going up (as bit#0 = 0) with a speed of 24 ($18), that the pitch of the note is $25 or c#3, and that this is the end of the pattern.

It doesn't get any harder than that!! Did you realise that this is exactly the way that Rob Hubbard made the music!! He worked out some musical ideas on his cheap (musical) keyboard, and typed the notes into his assembler in hex, just like this.

5. The instruments are an 8 byte data structure.

You are looking at the label 'instr' at the bottom of the sourcecode. The 8 bytes which come along first are instrument numnber 0, the next 8 define instrument number 1, etc. Here are the meanings of the bytes, but I suggest that you check out your programming manuals if you are unfamiliar with these:

- Byte 0 is the pulse width low byte, and
- Byte 1 is the pulse width high byte. (also see byte 7).

- Byte 2 is the control register byte.
  This specifies which type of sound should be used; sine, sawtooth etc.

- Byte 3 is the attack and decay values, and
- Byte 4 is the sustain and release values.
  The note's volume is altered according to these values. When the attack and decay are over, the volume of the note is held at the sustain level. When length of a note is over, a release is done through the 'gate' bit in SID.

- Byte 5 is the vibrato depth for the instrument.

- Byte 6 is the pulse speed.
  Timbre is created by changing the shape of the waveform each 50th of a second, and this is the most common way of achieving it. The shape of

  the pulse waveform changes from square to a very rectangular at a speed
  according to this byte.
  N.B. if you are interested in how the pulse value number works, then
  e-mail me sometime, as I found this out (exhaustively) a few days ago!

- Byte 7 is the instrument fx byte, and is the major thing which changes
  between different music routines. Each bit in this byte determines whether
  this instrument will have a certain effect in it.
  - Bit#0 signals that this is a drum. Drums are made from a noise channel
    and also a fast frequency down, with fast decay. Bass drums use a square
    wave, and only the first 50th of a second is a noise channel. This is
    the tell-tale instrument that gives away a Rob Hubbard routine! Hihats
    and other drums use noise all the time.
  - Bit#1 signals a 'skydive'. This is a slower frequency down, that I think
    sounds like somebody yelling as they fall out of a plane .. AHHHHhhhhgh..
    ..hence I call it a skydive!!
  - Bit#2 signals an octave arpeggio. It's a very limited arpeggio routine in
    this song. Listen for the arpeggio and the skydive when combined, which
    is used alot in Hubbard songs.
  - All the other bits have no meaning in this music, but were used alot in
    later music for the fx.

A big reason that I presented this early routine, was because there was not
too much in the way of special fx to confuse you. As a result, you can
concentrate on the guts of the code instead of the special fx :-)

How the sourcecode works:
****

  The routines at the top of the sourcecode are concerned with turning the
music on and off, and you will see that this is done through a variable called
'mstatus' (or music status). If mstatus is set to $C0, then the music is
turned off and SID is quietened, thereafter mstatus is set to $80 which means
that the music is still off, but SID doesn't need to be quietened again. When
the music is initialized, then mstatus is given a value of $40 which kicks in
the further initialization stuff. If mstatus is any other value, then the
music is being played. For any of the initialization stuff to have any meaning
to you, you ofcourse have to understand the rest of the playroutine :-)

  After we have got past the on/off/init stuff, we are at the label called
'contplay' at around line 100. The first thing you should notice is that this
is the start of a huge loop that is done *3* times - once for each channel.
The loop really *is* huge, as it ends right on the last few lines of the code
:-)

  Now that we are talking about routines within the loop, we are talking about
these routines being applied to the channels independantly. There are 2 main
routines within the loop, one is called NoteWork, and the other is called
SoundWork. NoteWork checks to see whether a new note is needed on this
channel, and if it is, then the notedata is fetched and stuff is initialized.
If no note is needed, then SoundWork is called which processes the instruments
and does the portamento.

  NoteWork first checks the speed at which the notes are fetched. If the delay
is still occurring, then new notes are not needed and soundwork is called.
N.B. that the speed for Monty on the Run is 1, which means that a note of
length $1f will last for 64 calls to the routine (ie just over a second). If
the speed of the song is reset, then NoteWork decrements the length of the
current note. When the length of the current note hits $ff (-1) then a new
note is needed, otherwise SoundWork is jumped to.

  The data for a new note is collected at the label 'getnewnote'. In the
simplest case, this involves getting the next bytes of data from the current
pattern on this channel, but if the end of the pattern is reached, then the
next pattern number is fetched by reference to the current position within
this channel's track. In an even more complex situation, the end of a track is
reached, and the current position needs to be reset to 0 before the next
pattern number can be found.

  You can see quite clearly in this part of the routine where the length of
the note is collected, and it is determined whether a 2nd byte is needed,
where the pitch is collected, and the end of the song checked. You can also
see where some of the data is collected from the current instrument and jammed
into the SID registers.

  SoundWork is called if no new notes are needed, and it processes the
instruments and does the portamento etc. This part of the routine is neatly
expressed in sections which are really well commented and quite easy to
understand.

- The first thing that occurs in SoundWork is that the 'gate bit' of SID is
  set when the length of the note is over - this causes a release of the note.

- The vibrato routine is quite inefficient, but it's pretty good for 1985!
  Ofcourse vibrato is implemented by raising and lowering the pitch of the
  note ever-so-slightly causing the note to 'float'. The amount of the
  vibrato is determined in the current instrument.

- The pulsework routine changes the pulsewidth between square wave and very
  rectangular wave according to the pulsespeed in the current instrument.
  (ie. it changes the sound of the instrument and thus alters the 'timbre')
  The routine goes backwards and forwards between the two; and switches
  when one extremity is reached. It's interesting to note that the current
  values of the pulse width are actually stored in the instrument :-)

- Portamento is achieved by adding/subtracting an amount of frequency to
  the current frequency each time this part of the routine is called.

- The instrument fx routines are also really easy to figure out, as they are
  well commented. Both the drums and the skydive do a very fast frequency
  down, so it is the most significant byte of the frequency which is reduced
  .. and not 16-bit maths (math?!) The arpeggio is only an octave arpeggio,
  so for the first 50th of a second, the current note is played, and for
  the next 50th of a second, current note+12 is played, followed by the
  current note again etc.
  (If you don't know what an arpeggio is..it's generally when the notes of)
  (a chord are played individually in a rapid succession. It produces a   )
  ('full' sound depending on the speed of the arpeggio. In most cases the)
  (note is changed 50 times per second, which gives a very nice sound. If)
  (you have listened to some computer music, then you will have definately)
  (listened to an arpeggios all the time, even if you don't realize it!   )

Final Thoughts:
****

  Bounce I'm finally near the end of this article! It has been alot of work to try to explain this routine, but I'm glad that I've done it *grin* If you have any questions then please feel free to e-mail me, or even e-mail Craig if it's after August 1993 and I'll make sure that I leave a forwarding address with him. Also, please feel free to e-mail me and tell me what you think of this article. I will only be bothered writing more of the same if I know that someone is finding them useful/interesting. Also e-mail me if you are interested in Amiga or ST music too, as I've done alot on both of those machines.

  I'm not sure whether Craig will be putting the actual sourcecode below this text, or in some kind of Appendix. In either case, I SHALL take all legal responsibilty for publishing Rob Hubbard's routine. Craig was quite reluctant to publish the routine in his net-mag because of copyright reasons. As a post-graduate law student that will be working as a commercial lawyer (attourney for Americans :) specializing in copyright/patents for computer software/hardware starting August this year, I don't believe that there are any practical legal consequences for me.

  I would have given an arm or a leg for a commented Rob Hubbard sourcecode in the past, so I hope you enjoy this valuable offering.

```
;rob hubbard
;monty on the run music driver

;this player was used (with small mods)
;for his first approx 30 musix

.org $8000
.obj motr

 jmp initmusic
 jmp playmusic
 jmp musicoff


;===================================
;init music

initmusic =*

  lda #$00         ;music num
  ldy #$00
  asl
  sta tempstore
```

```
  asl
  clc
  adc tempstore    ;now music num*6
  tax

- lda songs,x      ;copy ptrs to this
  sta currtrkhi,y  ;music's tracks to
  inx              ;current tracks
  iny
  cpy #$06
  bne -

  lda #$00         ;clear control regs
  sta $d404
  sta $d40b
  sta $d412
  sta $d417

  lda #$0f         ;full volume
  sta $d418

  lda #$40         ;flag init music
  sta mstatus

  rts


;====================================
;music off

musicoff =*

  lda #$c0         ;flag music off
  sta mstatus
  rts


;====================================
;play music

playmusic =*

  inc counter

  bit mstatus      ;test music status
  bmi moff         ;$80 and $c0 is off
  bvc contplay     ;$40 init, else play


;==========
;init the song (mstatus $40)

  lda #$00         ;init counter
  sta counter

  ldx #3-1
- sta posoffset,x  ;init pos offsets
  sta patoffset,x  ;init pat offsets
  sta lengthleft,x ;get note right away
  sta notenum,x
  dex
  bpl -

  sta mstatus      ;signal music play
  jmp contplay


;==========
;music is off (mstatus $80 or $c0)

moff =*

  bvc +            ;if mstatus $c0 then
  lda #$00
  sta $d404        ;kill voice 1,2,3
  sta $d40b        ;control registers
  sta $d412

  lda #$0f         ;full volume still
  sta $d418
```

```
  lda #$80          ;flag no need to kill
  sta mstatus       ;sound next time

+ jmp musicend      ;end


;==========
;music is playing (mstatus otherwise)

contplay =*

  ldx #3-1          ;number of chanels

  dec speed         ;check the speed
  bpl mainloop

  lda resetspd      ;reset speed if needed
  sta speed


mainloop =*

  lda regoffsets,x ;save offset to regs
  sta tmpregofst   ;for this channel
  tay


;check whether a new note is needed

  lda speed         ;if speed not reset
  cmp resetspd      ;then skip notework
  beq checknewnote
  jmp vibrato

checknewnote =*

  lda currtrkhi,x  ;put base addr.w of
  sta $02          ;this track in $2
  lda currtrklo,x
  sta $03

  dec lengthleft,x ;check whether a new
  bmi getnewnote   ;note is needed

  jmp soundwork    ;no new note needed


;==========
;notework
;a new note is needed. get the pattern
;number/cc from this position

getnewnote =*

  ldy posoffset,x  ;get the data from
  lda ($02),y      ;the current position

  cmp #$ff         ;pos $ff restarts
  beq restart

  cmp #$fe         ;pos $fe stops music
  bne getnotedata  ;on all channels
  jmp musicend

;cc of $ff restarts this track from the
;first position

restart =*

  lda #$00         ;get note immediately
  sta lengthleft,x ;and reset pat,pos
  sta posoffset,x
  sta patoffset,x
  jmp getnewnote


;get the note data from this pattern

getnotedata =*

  tay
```

```
  lda patptl,y     ;put base addr.w of
  sta $04          ;the pattern in $4
  lda patpth,y
  sta $05

  lda #$00         ;default no portamento
  sta portaval,x

  ldy patoffset,x  ;get offset into ptn

  lda #$ff         ;default no append
  sta appendfl

;1st byte is the length of the note 0-31
;bit5 signals no release (see sndwork)
;bit6 signals appended note
;bit7 signals a new instrument
;     or portamento coming up

  lda ($04),y      ;get length of note
  sta savelnthcc,x
  sta templnthcc
  and #$1f
  sta lengthleft,x

  bit templnthcc   ;test for append
  bvs appendnote

  inc patoffset,x  ;pt to next data

  lda templnthcc   ;2nd byte needed?
  bpl getpitch

;2nd byte needed as 1st byte negative
;2nd byte is the instrument number(+ve)
;or portamento speed(-ve)

  iny
  lda ($04),y      ;get instr/portamento
  bpl +

  sta portaval,x   ;save portamento val
  jmp ++

+ sta instrnr,x    ;save instr nr

+ inc patoffset,x

;3rd byte is the pitch of the note
;get the 'base frequency' here

getpitch =*

  iny
  lda ($04),y      ;get pitch of note
  sta notenum,x
  asl              ;pitch*2
  tay
  lda frequenzlo,y ;save the appropriate
  sta tempfreq     ;base frequency
  lda frequenzhi,y
  ldy tmpregofst
  sta $d401,y
  sta savefreqhi,x
  lda tempfreq
  sta $d400,y
  sta savefreqlo,x
  jmp +

appendnote =*

  dec appendfl     ;clever eh?


;fetch all the initial values from the
;instrument data structure

+ ldy tmpregofst
  lda instrnr,x    ;instr num
  stx tempstore
  asl              ;instr num*8
```

```
  asl
  asl
  tax

  lda instr+2,x    ;get control reg val
  sta tempctrl
  lda instr+2,x
  and appendfl     ;implement append
  sta $d404,y

  lda instr+0,x    ;get pulse width lo
  sta $d402,y

  lda instr+1,x    ;get pulse width hi
  sta $d403,y

  lda instr+3,x    ;get attack/decay
  sta $d405,y

  lda instr+4,x    ;get sustain/release
  sta $d406,y

  ldx tempstore    ;save control reg val
  lda tempctrl
  sta voicectrl,x


;4th byte checks for the end of pattern
;if eop found, inc the position and
;reset patoffset for new pattern

  inc patoffset,x  ;preview 4th byte
  ldy patoffset,x
  lda ($04),y

  cmp #$ff         ;check for eop
  bne +

  lda #$00         ;end of pat reached
  sta patoffset,x  ;inc position for
  inc posoffset,x  ;the next time

+ jmp loopcont


;==========
;soundwork
;the instrument and effects processing
;routine when no new note was needed

soundwork =*

;release routine
;set off a release when the length of
;the note is exceeded
;bit4 of the 1st note-byte can specify
;for no release

  ldy tmpregofst

  lda savelnthcc,x ;check for no release
  and #$20         ;specified
  bne vibrato

  lda lengthleft,x ;check for length of
  bne vibrato      ;exceeded

  lda voicectrl,x  ;length exceeded so
  and #$fe         ;start the release
  sta $d404,y      ;and kill adsr
  lda #$00
  sta $d405,y
  sta $d406,y


;vibrato routine
;(does alot of work)

vibrato =*

  lda instrnr,x    ;instr num
```

```
  asl
  asl
  asl               ;instr num*8
  tay
  sty instnumby8    ;save instr num*8

  lda instr+7,y     ;get instr fx byte
  sta instrfx

  lda instr+6,y     ;get pulse speed
  sta pulsevalue

  lda instr+5,y     ;get vibrato depth
  sta vibrdepth
  beq pulsework     ;check for no vibrato

  lda counter       ;this is clever!!
  and #7            ;the counter's turned
  cmp #4            ;into an oscillating
  bcc +             ;value (01233210)
  eor #7
+ sta oscilatval

  lda notenum,x     ;get base note
  asl               ;note*2
  tay               ;get diff btw note
  sec               ;and note+1 frequency
  lda frequenzlo+2,y
  sbc frequenzlo,y
  sta tmpvdiflo
  lda frequenzhi+2,y
  sbc frequenzhi,y

- lsr               ;divide difference by
  ror tmpvdiflo     ;2 for each vibrdepth
  dec vibrdepth
  bpl -
  sta tmpvdifhi

  lda frequenzlo,y ;save note frequency
  sta tmpvfrqlo
  lda frequenzhi,y
  sta tmpvfrqhi

  lda savelnthcc,x ;no vibrato if note
  and #$1f          ;length less than 8
  cmp #8
  bcc +

  ldy oscilatval

- dey               ;depending on the osc
  bmi +             ;value, add the vibr
  clc               ;freq that many times
  lda tmpvfrqlo     ;to the base freq
  adc tmpvdiflo
  sta tmpvfrqlo
  lda tmpvfrqhi
  adc tmpvdifhi
  sta tmpvfrqhi
  jmp -

+ ldy tmpregofst   ;save the final
  lda tmpvfrqlo    ;frequencies
  sta $d400,y
  lda tmpvfrqhi
  sta $d401,y


;pulse-width timbre routine
;depending on the control/speed byte in
;the instrument datastructure, the pulse
;width is of course inc/decremented to
;produce timbre

;strangely the delay value is also the
;size of the inc/decrements

pulsework =*

  lda pulsevalue   ;check for pulsework
```

```
  beq portamento   ;needed this instr

  ldy instnumby8
  and #$1f
  dec pulsedelay,x ;pulsedelay-1
  bpl portamento

  sta pulsedelay,x ;reset pulsedelay

  lda pulsevalue   ;restrict pulse speed
  and #$e0         ;from $00-$1f
  sta pulsespeed

  lda pulsedir,x   ;pulsedir 0 is up and
  bne pulsedown    ;1 is down

  lda pulsespeed   ;pulse width up
  clc
  adc instr+0,y    ;add the pulsespeed
  pha              ;to the pulse width
  lda instr+1,y
  adc #$00
  and #$0f
  pha
  cmp #$0e         ;go pulsedown when
  bne dumpulse     ;the pulse value
  inc pulsedir,x   ;reaches max ($0exx)
  jmp dumpulse

pulsedown =*

  sec              ;pulse width down
  lda instr+0,y
  sbc pulsespeed   ;sub the pulsespeed
  pha              ;from the pulse width
  lda instr+1,y
  sbc #$00
  and #$0f
  pha
  cmp #$08         ;go pulseup when
  bne dumpulse     ;the pulse value
  dec pulsedir,x   ;reaches min ($08xx)

dumpulse =*

  stx tempstore    ;dump pulse width to
  ldx tmpregofst   ;chip and back into
  pla              ;the instr data str
  sta instr+1,y
  sta $d403,x
  pla
  sta instr+0,y
  sta $d402,x
  ldx tempstore


;portamento routine
;portamento comes from the second byte
;if it's a negative value

portamento =*

  ldy tmpregofst
  lda portaval,x   ;check for portamento
  beq drums        ;none

  and #$7e         ;toad unwanted bits
  sta tempstore

  lda portaval,x   ;bit0 signals up/down
  and #$01
  beq portup

  sec              ;portamento down
  lda savefreqlo,x ;sub portaval from
  sbc tempstore    ;current frequency
  sta savefreqlo,x
  sta $d400,y
  lda savefreqhi,x
  sbc #$00         ;(word arithmetic)
  sta savefreqhi,x
```

```
  sta $d401,y
  jmp drums

portup =*

  clc              ;portamento up
  lda savefreqlo,x ;add portval to
  adc tempstore    ;current frequency
  sta savefreqlo,x
  sta $d400,y
  lda savefreqhi,x
  adc #$00
  sta savefreqhi,x
  sta $d401,y


;bit0 instrfx are the drum routines
;the actual drum timbre depends on the
;crtl register value for the instrument:
;ctrlreg 0 is always noise
;ctrlreg x is noise for 1st vbl and x
;from then on

;see that the drum is made by rapid hi
;to low frequency slide with fast attack
;and decay

drums =*

  lda instrfx      ;check if drums
  and #$01         ;needed this instr
  beq skydive

  lda savefreqhi,x ;don't bother if freq
  beq skydive      ;can't go any lower

  lda lengthleft,x ;or if the note has
  beq skydive      ;finished

  lda savelnthcc,x ;check if this is the
  and #$1f         ;first vbl for this
  sec              ;instrument-note
  sbc #$01
  cmp lengthleft,x
  ldy tmpregofst
  bcc firstime

  lda savefreqhi,x ;not the first time
  dec savefreqhi,x ;so dec freqhi for
  sta $d401,y      ;drum sound

  lda voicectrl,x  ;if ctrlreg is 0 then
  and #$fe         ;noise is used always
  bne dumpctrl

firstime =*

  lda savefreqhi,x ;noise is used for
  sta $d401,y      ;the first vbl also
  lda #$80         ;(set noise)

dumpctrl =*

  sta $d404,y


;bit1 instrfx is the skydive
;a long portamento-down from the note
;to zerofreq

skydive =*

  lda instrfx      ;check if skydive
  and #$02         ;needed this instr
  beq octarp

  lda counter      ;every 2nd vbl
  and #$01
  beq octarp

  lda savefreqhi,x ;check if skydive
```

```
  beq octarp        ;already complete

  dec savefreqhi,x ;decr and save the
  ldy tmpregofst   ;high byte freq
  sta $d401,y


;bit2 instrfx is an octave arpeggio
;pretty tame huh?

octarp =*

  lda instrfx      ;check if arpt needed
  and #$04
  beq loopcont

  lda counter      ;only 2 arpt values
  and #$01
  beq +

  lda notenum,x    ;odd, note+12
  clc
  adc #$0c
  jmp ++

+ lda notenum,x    ;even, note

+ asl              ;dump the corresponding
  tay              ;frequencies
  lda frequenzlo,y
  sta tempfreq
  lda frequenzhi,y
  ldy tmpregofst
  sta $d401,y
  lda tempfreq
  sta $d400,y


;==========
;end of dbf loop

loopcont =*

  dex              ;dbf mainloop
  bmi musicend
  jmp mainloop

musicend =*

  rts


;====================================
;frequenz data
;====================================

frequenzlo .byt $16
frequenzhi .byt $01
 .byt $27,$01,$38,$01,$4b,$01
 .byt $5f,$01,$73,$01,$8a,$01,$a1,$01
 .byt $ba,$01,$d4,$01,$f0,$01,$0e,$02
 .byt $2d,$02,$4e,$02,$71,$02,$96,$02
 .byt $bd,$02,$e7,$02,$13,$03,$42,$03
 .byt $74,$03,$a9,$03,$e0,$03,$1b,$04
 .byt $5a,$04,$9b,$04,$e2,$04,$2c,$05
 .byt $7b,$05,$ce,$05,$27,$06,$85,$06
 .byt $e8,$06,$51,$07,$c1,$07,$37,$08
 .byt $b4,$08,$37,$09,$c4,$09,$57,$0a
 .byt $f5,$0a,$9c,$0b,$4e,$0c,$09,$0d
 .byt $d0,$0d,$a3,$0e,$82,$0f,$6e,$10
 .byt $68,$11,$6e,$12,$88,$13,$af,$14
 .byt $eb,$15,$39,$17,$9c,$18,$13,$1a
 .byt $a1,$1b,$46,$1d,$04,$1f,$dc,$20
 .byt $d0,$22,$dc,$24,$10,$27,$5e,$29
 .byt $d6,$2b,$72,$2e,$38,$31,$26,$34
 .byt $42,$37,$8c,$3a,$08,$3e,$b8,$41
 .byt $a0,$45,$b8,$49,$20,$4e,$bc,$52
 .byt $ac,$57,$e4,$5c,$70,$62,$4c,$68
 .byt $84,$6e,$18,$75,$10,$7c,$70,$83
 .byt $40,$8b,$70,$93,$40,$9c,$78,$a5
 .byt $58,$af,$c8,$b9,$e0,$c4,$98,$d0
```

```
 .byt $08,$dd,$30,$ea,$20,$f8,$2e,$fd


regoffsets .byt $00,$07,$0e
tmpregofst .byt $00
posoffset  .byt $00,$00,$00
patoffset  .byt $00,$00,$00
lengthleft .byt $00,$00,$00
savelnthcc .byt $00,$00,$00
voicectrl  .byt $00,$00,$00
notenum    .byt $00,$00,$00
instrnr    .byt $00,$00,$00
appendfl   .byt $00
templnthcc .byt $00
tempfreq   .byt $00
tempstore  .byt $00
tempctrl   .byt $00
vibrdepth  .byt $00
pulsevalue .byt $00
tmpvdiflo  .byt $00
tmpvdifhi  .byt $00
tmpvfrqlo  .byt $00
tmpvfrqhi  .byt $00
oscilatval .byt $00
pulsedelay .byt $00,$00,$00
pulsedir   .byt $00,$00,$00
speed      .byt $00
resetspd   .byt $01
instnumby8 .byt $00
mstatus    .byt $c0
savefreqhi .byt $00,$00,$00
savefreqlo .byt $00,$00,$00
portaval   .byt $00,$00,$00
instrfx    .byt $00
pulsespeed .byt $00
counter    .byt $00
currtrkhi  .byt $00,$00,$00
currtrklo  .byt $00,$00,$00


;====================================
;monty on the run main theme
;====================================

songs =*
 .byt <montymaintr1
 .byt <montymaintr2
 .byt <montymaintr3
 .byt >montymaintr1
 .byt >montymaintr2
 .byt >montymaintr3


;====================================
;pointers to the patterns

;low pointers
patptl =*
 .byt <ptn00
 .byt <ptn01
 .byt <ptn02
 .byt <ptn03
 .byt <ptn04
 .byt <ptn05
 .byt <ptn06
 .byt <ptn07
 .byt <ptn08
 .byt <ptn09
 .byt <ptn0a
 .byt <ptn0b
 .byt <ptn0c
 .byt <ptn0d
 .byt <ptn0e
 .byt <ptn0f
 .byt <ptn10
 .byt <ptn11
 .byt <ptn12
 .byt <ptn13
 .byt <ptn14
 .byt <ptn15
 .byt <ptn16
```

```
  .byt <ptn17
  .byt <ptn18
  .byt <ptn19
  .byt <ptn1a
  .byt <ptn1b
  .byt <ptn1c
  .byt <ptn1d
  .byt <ptn1e
  .byt <ptn1f
  .byt <ptn20
  .byt <ptn21
  .byt <ptn22
  .byt <ptn23
  .byt <ptn24
  .byt <ptn25
  .byt <ptn26
  .byt <ptn27
  .byt <ptn28
  .byt <ptn29
  .byt <ptn2a
  .byt <ptn2b
  .byt <ptn2c
  .byt <ptn2d
  .byt 0
  .byt <ptn2f
  .byt <ptn30
  .byt <ptn31
  .byt <ptn32
  .byt <ptn33
  .byt <ptn34
  .byt <ptn35
  .byt <ptn36
  .byt <ptn37
  .byt <ptn38
  .byt <ptn39
  .byt <ptn3a
  .byt <ptn3b

;high pointers
patpth =*
  .byt >ptn00
  .byt >ptn01
  .byt >ptn02
  .byt >ptn03
  .byt >ptn04
  .byt >ptn05
  .byt >ptn06
  .byt >ptn07
  .byt >ptn08
  .byt >ptn09
  .byt >ptn0a
  .byt >ptn0b
  .byt >ptn0c
  .byt >ptn0d
  .byt >ptn0e
  .byt >ptn0f
  .byt >ptn10
  .byt >ptn11
  .byt >ptn12
  .byt >ptn13
  .byt >ptn14
  .byt >ptn15
  .byt >ptn16
  .byt >ptn17
  .byt >ptn18
  .byt >ptn19
  .byt >ptn1a
  .byt >ptn1b
  .byt >ptn1c
  .byt >ptn1d
  .byt >ptn1e
  .byt >ptn1f
  .byt >ptn20
  .byt >ptn21
  .byt >ptn22
  .byt >ptn23
  .byt >ptn24
  .byt >ptn25
  .byt >ptn26
  .byt >ptn27
  .byt >ptn28
```

```
 .byt >ptn29
 .byt >ptn2a
 .byt >ptn2b
 .byt >ptn2c
 .byt >ptn2d
 .byt 0
 .byt >ptn2f
 .byt >ptn30
 .byt >ptn31
 .byt >ptn32
 .byt >ptn33
 .byt >ptn34
 .byt >ptn35
 .byt >ptn36
 .byt >ptn37
 .byt >ptn38
 .byt >ptn39
 .byt >ptn3a
 .byt >ptn3b


;==================================
;tracks
;==================================

;track1
montymaintr1 =*
 .byt $11,$14,$17,$1a,$00,$27,$00,$28
 .byt $03,$05,$00,$27,$00,$28,$03,$05
 .byt $07,$3a,$14,$17,$00,$27,$00,$28
 .byt $2f,$30,$31,$31,$32,$33,$33,$34
 .byt $34,$34,$34,$34,$34,$34,$34,$35
 .byt $35,$35,$35,$35,$35,$36,$12,$37
 .byt $38,$09,$2a,$09,$2b,$09,$0a,$09
 .byt $2a,$09,$2b,$09,$0a,$0d,$0d,$0f
 .byt $ff

;track2
montymaintr2 =*
 .byt $12,$15,$18,$1b,$2d,$39,$39
 .byt $39,$39,$39,$39,$2c,$39,$39,$39
 .byt $39,$39,$39,$2c,$39,$39,$39,$01
 .byt $01,$29,$29,$2c,$15,$18,$39,$39
 .byt $39,$39,$39,$39,$39,$39,$39,$39
 .byt $39,$39,$39,$39,$39,$39,$39,$39
 .byt $39,$39,$39,$39,$39,$39,$39,$39
 .byt $39,$39,$39,$39,$39,$01,$01,$01
 .byt $29,$39,$39,$39,$01,$01,$01,$29
 .byt $39,$39,$39,$39,$ff

;track3
montymaintr3 =*
 .byt $13,$16,$19
 .byt $1c,$02,$02,$1d,$1e,$02,$02,$1d
 .byt $1f,$04,$04,$20,$20,$06,$02,$02
 .byt $1d,$1e,$02,$02,$1d,$1f,$04,$04
 .byt $20,$20,$06,$08,$08,$08,$08,$21
 .byt $21,$21,$21,$22,$22,$22,$23,$22
 .byt $24,$25,$3b,$26,$26,$26,$26,$26
 .byt $26,$26,$26,$26,$26,$26,$26,$26
 .byt $26,$26,$26,$02,$02,$1d,$1e,$02
 .byt $02,$1d,$1f,$2f,$2f,$2f,$2f,$2f
 .byt $2f,$2f,$2f,$2f,$2f,$2f,$2f,$2f
 .byt $0b,$0b,$1d,$1d,$0b,$0b,$1d,$0b
 .byt $0b,$0b,$0c,$0c,$1d,$1d,$1d,$10
 .byt $0b,$0b,$1d,$1d,$0b,$0b,$1d,$0b
 .byt $0b,$0b,$0c,$0c,$1d,$1d,$1d,$10
 .byt $0b,$1d,$0b,$1d,$0b,$1d,$0b,$1d
 .byt $0b,$0c,$1d,$0b,$0c,$23,$0b,$0b
 .byt $ff


;==================================
;patterns
;==================================

ptn00 =*
 .byt $83,$00,$37,$01,$3e,$01,$3e,$03
 .byt $3d,$03,$3e,$03,$43,$03,$3e,$03
 .byt $3d,$03,$3e,$03,$37,$01,$3e,$01
 .byt $3e,$03,$3d,$03,$3e,$03,$43,$03
```

```
  .byt $42,$03,$43,$03,$45,$03,$46,$01
  .byt $48,$01,$46,$03,$45,$03,$43,$03
  .byt $4b,$01,$4d,$01,$4b,$03,$4a,$03
  .byt $48,$ff

ptn27 =*
  .byt $1f,$4a,$ff

ptn28 =*
  .byt $03,$46,$01,$48,$01,$46,$03,$45
  .byt $03,$4a,$0f,$43,$ff

ptn03 =*
  .byt $bf,$06
  .byt $48,$07,$48,$01,$4b,$01,$4a,$01
  .byt $4b,$01,$4a,$03,$4b,$03,$4d,$03
  .byt $4b,$03,$4a,$3f,$48,$07,$48,$01
  .byt $4b,$01,$4a,$01,$4b,$01,$4a,$03
  .byt $4b,$03,$4d,$03,$4b,$03,$48,$3f
  .byt $4c,$07,$4c,$01,$4f,$01,$4e,$01
  .byt $4f,$01,$4e,$03,$4f,$03,$51,$03
  .byt $4f,$03,$4e,$3f,$4c,$07,$4c,$01
  .byt $4f,$01,$4e,$01,$4f,$01,$4e,$03
  .byt $4f,$03,$51,$03,$4f,$03,$4c,$ff

ptn05 =*
  .byt $83,$04,$26,$03,$29,$03,$28,$03
  .byt $29,$03,$26,$03,$35,$03,$34,$03
  .byt $32,$03,$2d,$03,$30,$03,$2f,$03
  .byt $30,$03,$2d,$03,$3c,$03,$3b,$03
  .byt $39,$03,$30,$03,$33,$03,$32,$03
  .byt $33,$03,$30,$03,$3f,$03,$3e,$03
  .byt $3c,$03,$46,$03,$45,$03,$43,$03
  .byt $3a,$03,$39,$03,$37,$03,$2e,$03
  .byt $2d,$03,$26,$03,$29,$03,$28,$03
  .byt $29,$03,$26,$03,$35,$03,$34,$03
  .byt $32,$03,$2d,$03,$30,$03,$2f,$03
  .byt $30,$03,$2d,$03,$3c,$03,$3b,$03
  .byt $39,$03,$30,$03,$33,$03,$32,$03
  .byt $33,$03,$30,$03,$3f,$03,$3e,$03
  .byt $3c,$03,$34,$03,$37,$03,$36,$03
  .byt $37,$03,$34,$03,$37,$03,$3a,$03
  .byt $3d

ptn3a =*
  .byt $03,$3e,$07,$3e,$07,$3f,$07
  .byt $3e,$03,$3c,$07,$3e,$57,$ff

ptn07 =*
  .byt $8b
  .byt $00,$3a,$01,$3a,$01,$3c,$03,$3d
  .byt $03,$3f,$03,$3d,$03,$3c,$0b,$3a
  .byt $03,$39,$07,$3a,$81,$06,$4b,$01
  .byt $4d,$01,$4e,$01,$4d,$01,$4e,$01
  .byt $4d,$05,$4b,$81,$00,$3a,$01,$3c
  .byt $01,$3d,$03,$3f,$03,$3d,$03,$3c
  .byt $03,$3a,$03,$39,$1b,$3a,$0b,$3b
  .byt $01,$3b,$01,$3d,$03,$3e,$03,$40
  .byt $03,$3e,$03,$3d,$0b,$3b,$03,$3a
  .byt $07,$3b,$81,$06,$4c,$01,$4e,$01
  .byt $4f,$01,$4e,$01,$4f,$01,$4e,$05
  .byt $4c,$81,$00,$3b,$01,$3d,$01,$3e
  .byt $03,$40,$03,$3e,$03,$3d,$03,$3b
  .byt $03,$3a,$1b,$3b,$8b,$05,$35,$03
  .byt $33,$07,$32,$03,$30,$03,$2f,$0b
  .byt $30,$03,$32,$0f,$30,$0b,$35,$03
  .byt $33,$07,$32,$03,$30,$03,$2f,$1f
  .byt $30,$8b,$00,$3c,$01,$3c,$01,$3e
  .byt $03,$3f,$03,$41,$03,$3f,$03,$3e
  .byt $0b,$3d,$01,$3d,$01,$3f,$03,$40
  .byt $03,$42,$03,$40,$03,$3f,$03,$3e
  .byt $01,$3e,$01,$40,$03,$41,$03,$40
  .byt $03,$3e,$03,$3d,$03,$3e,$03,$3c
  .byt $03,$3a,$01,$3a,$01,$3c,$03,$3d
  .byt $03,$3c,$03,$3a,$03,$39,$03,$3a
  .byt $03,$3c,$ff

ptn09 =*
  .byt $83,$00,$32,$01,$35,$01,$34,$03
  .byt $32,$03,$35,$03,$34,$03,$32,$03
  .byt $35,$01,$34,$01,$32,$03,$32,$03
```

```
  .byt $3a,$03,$39,$03,$3a,$03,$32,$03
  .byt $3a,$03,$39,$03,$3a,$ff

ptn2a =*
  .byt $03,$34,$01,$37,$01,$35,$03,$34
  .byt $03,$37,$03,$35,$03,$34,$03,$37
  .byt $01,$35,$01,$34,$03,$34,$03,$3a
  .byt $03,$39,$03,$3a,$03,$34,$03,$3a
  .byt $03,$39,$03,$3a,$ff

ptn2b =*
  .byt $03,$39,$03,$38,$03,$39,$03,$3a
  .byt $03,$39,$03,$37,$03,$35,$03,$34
  .byt $03,$35,$03,$34,$03,$35,$03,$37
  .byt $03,$35,$03,$34,$03,$32,$03,$31
  .byt $ff

ptn0a =*
  .byt $03
  .byt $37,$01,$3a,$01,$39,$03,$37,$03
  .byt $3a,$03,$39,$03,$37,$03,$3a,$01
  .byt $39,$01,$37,$03,$37,$03,$3e,$03
  .byt $3d,$03,$3e,$03,$37,$03,$3e,$03
  .byt $3d,$03,$3e,$03,$3d,$01,$40,$01
  .byt $3e,$03,$3d,$03,$40,$01,$3e,$01
  .byt $3d,$03,$40,$03,$3e,$03,$40,$03
  .byt $40,$01,$43,$01,$41,$03,$40,$03
  .byt $43,$01,$41,$01,$40,$03,$43,$03
  .byt $41,$03,$43,$03,$43,$01,$46,$01
  .byt $45,$03,$43,$03,$46,$01,$45,$01
  .byt $43,$03,$46,$03,$45,$03,$43,$01
  .byt $48,$01,$49,$01,$48,$01,$46,$01
  .byt $45,$01,$46,$01,$45,$01,$43,$01
  .byt $41,$01,$43,$01,$41,$01,$40,$01
  .byt $3d,$01,$39,$01,$3b,$01,$3d,$ff

ptn0d =*
  .byt $01,$3e,$01,$39,$01,$35,$01,$39
  .byt $01,$3e,$01,$39,$01,$35,$01,$39
  .byt $03,$3e,$01,$41,$01,$40,$03,$40
  .byt $01,$3d,$01,$3e,$01,$40,$01,$3d
  .byt $01,$39,$01,$3d,$01,$40,$01,$3d
  .byt $01,$39,$01,$3d,$03,$40,$01,$43
  .byt $01,$41,$03,$41,$01,$3e,$01,$40
  .byt $01,$41,$01,$3e,$01,$39,$01,$3e
  .byt $01,$41,$01,$3e,$01,$39,$01,$3e
  .byt $03,$41,$01,$45,$01,$43,$03,$43
  .byt $01,$40,$01,$41,$01,$43,$01,$40
  .byt $01,$3d,$01,$40,$01,$43,$01,$40
  .byt $01,$3d,$01,$40,$01,$46,$01,$43
  .byt $01,$45,$01,$46,$01,$44,$01,$43
  .byt $01,$40,$01,$3d,$ff

ptn0f =*
  .byt $01,$3e,$01
  .byt $39,$01,$35,$01,$39,$01,$3e,$01
  .byt $39,$01,$35,$01,$39,$01,$3e,$01
  .byt $39,$01,$35,$01,$39,$01,$3e,$01
  .byt $39,$01,$35,$01,$39,$01,$3e,$01
  .byt $3a,$01,$37,$01,$3a,$01,$3e,$01
  .byt $3a,$01,$37,$01,$3a,$01,$3e,$01
  .byt $3a,$01,$37,$01,$3a,$01,$3e,$01
  .byt $3a,$01,$37,$01,$3a,$01,$40,$01
  .byt $3d,$01,$39,$01,$3d,$01,$40,$01
  .byt $3d,$01,$39,$01,$3d,$01,$40,$01
  .byt $3d,$01,$39,$01,$3d,$01,$40,$01
  .byt $3d,$01,$39,$01,$3d,$01,$41,$01
  .byt $3e,$01,$39,$01,$3e,$01,$41,$01
  .byt $3e,$01,$39,$01,$3e,$01,$41,$01
  .byt $3e,$01,$39,$01,$3e,$01,$41,$01
  .byt $3e,$01,$39,$01,$3e,$01,$43,$01
  .byt $3e,$01,$3a,$01,$3e,$01,$43,$01
  .byt $3e,$01,$3a,$01,$3e,$01,$43,$01
  .byt $3e,$01,$3a,$01,$3e,$01,$43,$01
  .byt $3e,$01,$3a,$01,$3e,$01,$43,$01
  .byt $3f,$01,$3c,$01,$3f,$01,$43,$01
  .byt $3f,$01,$3c,$01,$3f,$01,$43,$01
  .byt $3f,$01,$3c,$01,$3f,$01,$43,$01
  .byt $3f,$01,$3c,$01,$3f,$01,$45,$01
  .byt $42,$01,$3c,$01,$42,$01,$45,$01
  .byt $42,$01,$3c,$01,$42,$01,$48,$01
```

```
  .byt $45,$01,$42,$01,$45,$01,$4b,$01
  .byt $48,$01,$45,$01,$48,$01,$4b,$01
  .byt $4a,$01,$48,$01,$4a,$01,$4b,$01
  .byt $4a,$01,$48,$01,$4a,$01,$4b,$01
  .byt $4a,$01,$48,$01,$4a,$01,$4c,$01
  .byt $4e,$03,$4f,$ff

ptn11 =*
  .byt $bf,$06,$56,$1f,$57,$1f,$56,$1f
  .byt $5b,$1f,$56,$1f,$57,$1f,$56,$1f
  .byt $4f,$ff

ptn12 =*
  .byt $bf,$0c,$68,$7f,$7f,$7f,$7f,$7f
  .byt $7f,$7f,$ff

ptn13 =*
  .byt $bf,$08,$13,$3f,$13,$3f,$13,$3f
  .byt $13,$3f,$13,$3f,$13,$3f,$13,$1f
  .byt $13,$ff

ptn14 =*
  .byt $97,$09,$2e,$03,$2e,$1b,$32,$03
  .byt $32,$1b,$31,$03,$31,$1f,$34,$43
  .byt $17,$32,$03,$32,$1b,$35,$03,$35
  .byt $1b,$34,$03,$34,$0f,$37,$8f,$0a
  .byt $37,$43,$ff

ptn15 =*
  .byt $97,$09,$2b,$03,$2b,$1b,$2e,$03
  .byt $2e,$1b,$2d,$03,$2d,$1f,$30,$43
  .byt $17,$2e,$03,$2e,$1b,$32,$03,$32
  .byt $1b,$31,$03,$31,$0f,$34,$8f,$0a
  .byt $34,$43,$ff

ptn16 =*
  .byt $0f,$1f,$0f,$1f,$0f,$1f,$0f,$1f
  .byt $0f,$1f,$0f,$1f,$0f,$1f,$0f,$1f
  .byt $0f,$1f,$0f,$1f,$0f,$1f,$0f,$1f
  .byt $0f,$1f,$0f,$1f,$0f,$1f,$0f,$1f
  .byt $ff

ptn17 =*
  .byt $97,$09,$33,$03,$33,$1b,$37,$03
  .byt $37,$1b,$36,$03,$36,$1f,$39,$43
  .byt $17,$37,$03,$37,$1b,$3a,$03,$3a
  .byt $1b,$39,$03,$39,$2f,$3c,$21,$3c
  .byt $21,$3d,$21,$3e,$21,$3f,$21,$40
  .byt $21,$41,$21,$42,$21,$43,$21,$44
  .byt $01,$45,$ff

ptn18 =*
  .byt $97,$09,$30,$03,$30,$1b,$33,$03
  .byt $33,$1b,$32,$03,$32,$1f,$36,$43
  .byt $17,$33,$03,$33,$1b,$37,$03,$37
  .byt $1b,$36,$03,$36,$2f,$39,$21,$39
  .byt $21,$3a,$21,$3b,$21,$3c,$21,$3d
  .byt $21,$3e,$21,$3f,$21,$40,$21,$41
  .byt $01,$42,$ff

ptn19 =*
  .byt $0f,$1a,$0f,$1a,$0f,$1a,$0f,$1a
  .byt $0f,$1a,$0f,$1a,$0f,$1a,$0f,$1a
  .byt $0f,$1a,$0f,$1a,$0f,$1a,$0f,$1a
  .byt $0f,$1a,$0f,$1a,$0f,$1a,$0f,$1a
  .byt $ff

ptn1a =*
  .byt $1f,$46,$bf,$0a,$46,$7f,$7f,$ff

ptn1b =*
  .byt $1f,$43,$bf,$0a,$43,$7f,$ff

ptn1c =*
  .byt $83,$02,$13,$03,$13,$03,$1e,$03
  .byt $1f,$03,$13,$03,$13,$03,$1e,$03
  .byt $1f,$03,$13,$03,$13,$03,$1e,$03
  .byt $1f,$03,$13,$03,$13,$03,$1e,$03
  .byt $1f,$03,$13,$03,$13,$03,$1e,$03
  .byt $1f,$03,$13,$03,$13,$03,$1e,$03
  .byt $1f,$03,$13,$03,$13,$03,$1e,$03
```

```
 .byt $1f,$03,$13,$03,$13,$03,$1e,$03
 .byt $1f,$ff

ptn29 =*
 .byt $8f,$0b,$38,$4f,$ff

ptn2c =*
 .byt $83,$0e,$32,$07,$32,$07,$2f,$07
 .byt $2f,$03,$2b,$87,$0b,$46,$83,$0e
 .byt $2c,$03,$2c,$8f,$0b,$32,$ff

ptn2d =*
 .byt $43,$83,$0e,$32,$03,$32,$03,$2f
 .byt $03,$2f,$03,$2c,$87,$0b,$38,$ff

ptn39 =*
 .byt $83,$01
 .byt $43,$01,$4f,$01,$5b,$87,$03,$2f
 .byt $83,$01,$43,$01,$4f,$01,$5b,$87
 .byt $03,$2f,$83,$01,$43,$01,$4f,$01
 .byt $5b,$87,$03,$2f,$83,$01,$43,$01
 .byt $4f,$01,$5b,$87,$03,$2f,$83,$01
 .byt $43,$01,$4f,$01,$5b,$87,$03,$2f
 .byt $83,$01,$43,$01,$4f,$01,$5b,$87
 .byt $03,$2f

ptn01 =*
 .byt $83,$01,$43,$01,$4f,$01,$5b,$87
 .byt $03,$2f,$83,$01,$43,$01,$4f,$01
 .byt $5b,$87,$03,$2f,$ff

ptn02 =*
 .byt $83,$02,$13,$03,$13,$03,$1f,$03
 .byt $1f,$03,$13,$03,$13,$03,$1f,$03
 .byt $1f,$ff

ptn1d =*
 .byt $03,$15,$03,$15,$03,$1f,$03,$21
 .byt $03,$15,$03,$15,$03,$1f,$03,$21
 .byt $ff

ptn1e =*
 .byt $03,$1a,$03,$1a,$03,$1c,$03,$1c
 .byt $03,$1d,$03,$1d,$03,$1e,$03,$1e
 .byt $ff

ptn1f =*
 .byt $03,$1a,$03,$1a,$03,$24,$03,$26
 .byt $03,$13,$03,$13,$07,$1f,$ff

ptn04 =*
 .byt $03,$18,$03,$18,$03,$24,$03,$24
 .byt $03,$18,$03,$18,$03,$24,$03,$24
 .byt $03,$20,$03,$20,$03,$2c,$03,$2c
 .byt $03,$20,$03,$20,$03,$2c,$03,$2c
 .byt $ff

ptn20 =*
 .byt $03,$19,$03,$19,$03
 .byt $25,$03,$25,$03,$19,$03,$19,$03
 .byt $25,$03,$25,$03,$21,$03,$21,$03
 .byt $2d,$03,$2d,$03,$21,$03,$21,$03
 .byt $2d,$03,$2d,$ff

ptn06 =*
 .byt $03,$1a,$03,$1a
 .byt $03,$26,$03,$26,$03,$1a,$03,$1a
 .byt $03,$26,$03,$26,$03,$15,$03,$15
 .byt $03,$21,$03,$21,$03,$15,$03,$15
 .byt $03,$21,$03,$21,$03,$18,$03,$18
 .byt $03,$24,$03,$24,$03,$18,$03,$18
 .byt $03,$24,$03,$24,$03,$1f,$03,$1f
 .byt $03,$2b,$03,$2b,$03,$1f,$03,$1f
 .byt $03,$2b,$03,$2b,$03,$1a,$03,$1a
 .byt $03,$26,$03,$26,$03,$1a,$03,$1a
 .byt $03,$26,$03,$26,$03,$15,$03,$15
 .byt $03,$21,$03,$21,$03,$15,$03,$15
 .byt $03,$21,$03,$21,$03,$18,$03,$18
 .byt $03,$24,$03,$24,$03,$18,$03,$18
 .byt $03,$24,$03,$24,$03,$1c,$03,$1c
 .byt $03,$28,$03,$28,$03,$1c,$03,$1c
```

```
 .byt $03,$28,$03,$28

ptn3b =*
 .byt $83,$04,$36,$07
 .byt $36,$07,$37,$07,$36,$03,$33,$07
 .byt $32,$57,$ff

ptn08 =*
 .byt $83,$02,$1b,$03,$1b,$03,$27,$03
 .byt $27,$03,$1b,$03,$1b,$03,$27,$03
 .byt $27,$ff

ptn21 =*
 .byt $03,$1c,$03,$1c,$03,$28,$03,$28
 .byt $03,$1c,$03,$1c,$03,$28,$03,$28
 .byt $ff

ptn22 =*
 .byt $03,$1d,$03,$1d,$03,$29,$03,$29
 .byt $03,$1d,$03,$1d,$03,$29,$03,$29
 .byt $ff

ptn23 =*
 .byt $03,$18,$03,$18,$03,$24,$03,$24
 .byt $03,$18,$03,$18,$03,$24,$03,$24
 .byt $ff

ptn24 =*
 .byt $03,$1e,$03,$1e,$03,$2a,$03,$2a
 .byt $03,$1e,$03,$1e,$03,$2a,$03,$2a
 .byt $ff

ptn25 =*
 .byt $83,$05,$26,$01,$4a,$01,$34,$03
 .byt $29,$03,$4c,$03,$4a,$03,$31,$03
 .byt $4a,$03,$24,$03,$22,$01,$46,$01
 .byt $30,$03,$25,$03,$48,$03,$46,$03
 .byt $2d,$03,$46,$03,$24,$ff

ptn0b =*
 .byt $83,$02,$1a,$03,$1a,$03,$26,$03
 .byt $26,$03,$1a,$03,$1a,$03,$26,$03
 .byt $26,$ff

ptn0c =*
 .byt $03,$13,$03,$13,$03,$1d,$03,$1f
 .byt $03,$13,$03,$13,$03,$1d,$03,$1f
 .byt $ff

ptn26 =*
 .byt $87,$02,$1a,$87,$03,$2f,$83,$02
 .byt $26,$03,$26,$87,$03,$2f,$ff

ptn10 =*
 .byt $07,$1a,$4f,$47,$ff

ptn0e =*
 .byt $03,$1f,$03,$1f,$03,$24,$03,$26
 .byt $07,$13,$47,$ff

ptn30 =*
 .byt $bf,$0f,$32,$0f,$32,$8f,$90,$30
 .byt $3f,$32,$13,$32,$03,$32,$03,$35
 .byt $03,$37,$3f,$37,$0f,$37,$8f,$90
 .byt $30,$3f,$32,$13,$32,$03,$2d,$03
 .byt $30,$03,$32,$ff

ptn31 =*
 .byt $0f,$32
 .byt $af,$90,$35,$0f,$37,$a7,$99,$37
 .byt $07,$35,$3f,$32,$13,$32,$03,$32
 .byt $a3,$e8,$35,$03,$37,$0f,$35,$af
 .byt $90,$37,$0f,$37,$a7,$99,$37,$07
 .byt $35,$3f,$32,$13,$32,$03,$2d,$a3
 .byt $e8,$30,$03,$32,$ff

ptn32 =*
 .byt $07,$32,$03
 .byt $39,$13,$3c,$a7,$9a,$37,$a7,$9b
 .byt $38,$07,$37,$03,$35,$03,$32,$03
 .byt $39,$1b,$3c,$a7,$9a,$37,$a7,$9b
```

```
  .byt $38,$07,$37,$03,$35,$03,$32,$03
  .byt $39,$03,$3c,$03,$3e,$03,$3c,$07
  .byt $3e,$03,$3c,$03,$39,$a7,$9a,$37
  .byt $a7,$9b,$38,$07,$37,$03,$35,$03
  .byt $32,$af,$90,$3c,$1f,$3e,$43,$03
  .byt $3e,$03,$3c,$03,$3e,$ff

ptn33 =*
  .byt $03,$3e
  .byt $03,$3e,$a3,$e8,$3c,$03,$3e,$03
  .byt $3e,$03,$3e,$a3,$e8,$3c,$03,$3e
  .byt $03,$3e,$03,$3e,$a3,$e8,$3c,$03
  .byt $3e,$03,$3e,$03,$3e,$a3,$e8,$3c
  .byt $03,$3e,$af,$91,$43,$1f,$41,$43
  .byt $03,$3e,$03,$41,$03,$43,$03,$43
  .byt $03,$43,$a3,$e8,$41,$03,$43,$03
  .byt $43,$03,$43,$a3,$e8,$41,$03,$43
  .byt $03,$45,$03,$48,$a3,$fd,$45,$03
  .byt $44,$01,$43,$01,$41,$03,$3e,$03
  .byt $3c,$03,$3e,$2f,$3e,$bf,$98,$3e
  .byt $43,$03,$3e,$03,$3c,$03,$3e,$ff

ptn34 =*
  .byt $03,$4a,$03,$4a,$a3,$f8,$48,$03
  .byt $4a,$03,$4a,$03,$4a,$a3,$f8,$48
  .byt $03,$4a,$ff

ptn35 =*
  .byt $01,$51,$01,$54,$01
  .byt $51,$01,$54,$01,$51,$01,$54,$01
  .byt $51,$01,$54,$01,$51,$01,$54,$01
  .byt $51,$01,$54,$01,$51,$01,$54,$01
  .byt $51,$01,$54,$ff

ptn36 =*
  .byt $01,$50,$01,$4f
  .byt $01,$4d,$01,$4a,$01,$4f,$01,$4d
  .byt $01,$4a,$01,$48,$01,$4a,$01,$48
  .byt $01,$45,$01,$43,$01,$44,$01,$43
  .byt $01,$41,$01,$3e,$01,$43,$01,$41
  .byt $01,$3e,$01,$3c,$01,$3e,$01,$3c
  .byt $01,$39,$01,$37,$01,$38,$01,$37
  .byt $01,$35,$01,$32,$01,$37,$01,$35
  .byt $01,$32,$01,$30,$ff

ptn37 =*
  .byt $5f,$5f,$5f
  .byt $47,$83,$0e,$32,$07,$32,$07,$2f
  .byt $03,$2f,$07,$2f,$97,$0b,$3a,$5f
  .byt $5f,$47,$8b,$0e,$32,$03,$32,$03
  .byt $2f,$03,$2f,$47,$97,$0b,$3a,$5f
  .byt $5f,$47,$83,$0e,$2f,$0b,$2f,$03
  .byt $2f,$03,$2f,$87,$0b,$30,$17,$3a
  .byt $5f,$8b,$0e,$32,$0b,$32,$0b,$2f
  .byt $0b,$2f,$07,$2c,$07,$2c,$ff

ptn38 =*
  .byt $87
  .byt $0b,$34,$17,$3a,$5f,$5f,$84,$0e
  .byt $32,$04,$32,$05,$32,$04,$2f,$04
  .byt $2f,$05,$2f,$47,$97,$0b,$3a,$5f
  .byt $5f,$84,$0e,$32,$04,$32,$05,$32
  .byt $04,$2f,$04,$2f,$05,$2f,$ff

ptn2f =*
  .byt $03,$1a,$03,$1a,$03
  .byt $24,$03,$26,$03,$1a,$03,$1a,$03
  .byt $18,$03,$19,$03,$1a,$03,$1a,$03
  .byt $24,$03,$26,$03,$1a,$03,$1a,$03
  .byt $18,$03,$19,$03,$18,$03,$18,$03
  .byt $22,$03,$24,$03,$18,$03,$18,$03
  .byt $16,$03,$17,$03,$18,$03,$18,$03
  .byt $22,$03,$24,$03,$18,$03,$18,$03
  .byt $16,$03,$17,$03,$13,$03,$13,$03
  .byt $1d,$03,$1f,$03,$13,$03,$13,$03
  .byt $1d,$03,$1e,$03,$13,$03,$13,$03
  .byt $1d,$03,$1f,$03,$13,$03,$13,$03
  .byt $1d,$03,$1e,$03,$1a,$03,$1a,$03
  .byt $24,$03,$26,$03,$1a,$03,$1a,$03
  .byt $18,$03,$19,$03,$1a,$03,$1a,$03
  .byt $24,$03,$26,$03,$1a,$03,$1a,$03
```

```
 .byt $18,$03,$19,$ff


;===================================
;instruments
;===================================

instr =*
 .byt $80,$09,$41,$48,$60,$03,$81,$00
 .byt $00,$08,$81,$02,$08,$00,$00,$01
 .byt $a0,$02,$41,$09,$80,$00,$00,$00
 .byt $00,$02,$81,$09,$09,$00,$00,$05
 .byt $00,$08,$41,$08,$50,$02,$00,$04
 .byt $00,$01,$41,$3f,$c0,$02,$00,$00
 .byt $00,$08,$41,$04,$40,$02,$00,$00
 .byt $00,$08,$41,$09,$00,$02,$00,$00
 .byt $00,$09,$41,$09,$70,$02,$5f,$04
 .byt $00,$09,$41,$4a,$69,$02,$81,$00
 .byt $00,$09,$41,$40,$6f,$00,$81,$02
 .byt $80,$07,$81,$0a,$0a,$00,$00,$01
 .byt $00,$09,$41,$3f,$ff,$01,$e7,$02
 .byt $00,$08,$41,$90,$f0,$01,$e8,$02
 .byt $00,$08,$41,$06,$0a,$00,$00,$01
 .byt $00,$09,$41,$19,$70,$02,$a8,$00
 .byt $00,$02,$41,$09,$90,$02,$00,$00
 .byt $00,$00,$11,$0a,$fa,$00,$00,$05
 .byt $00,$08,$41,$37,$40,$02,$00,$00
 .byt $00,$08,$11,$07,$70,$02,$00,$00

.end
```

```
================================================================================
ZPM3 and ZCCP Enhancements for CP/M Plus from Simeon Cran
by Randy Winchester (randy@mit.edu)

Operating System Components

The CP/M Plus operating system consists of three modules.  The CCP (Console
Command Processor), is the part of CP/M that you see when you first boot the
system.  The CCP prints the A> disk prompt, accepts user input, and loads
commands from disk.

The BDOS (Basic Disk Operating System) handles the CP/M functions of disk,
console, and printer input/output, and the tasks of file management.

The BIOS (Basic Input Output System) does the real input/output work for the
BDOS. The BIOS contains the code customized for the CP/M hardware that you're
using. On the C128, the BIOS contains the routines for driving the 40 and 80
column screens, using the REU as a RAM drive, and reading/writing several
different disk formats on 1571 and 1581 drives.  The BIOS can be thought of as
a collection of device drivers that are specific to your computer.


What's New - BIOS-R6

BIOS-R6 (C128 BIOS modified by Randy Winchester and others) is the latest of
the modified versions of the C128 CP/M BIOS.  Most of the changes to the BIOS
result in faster processing speed. For example, all the code for driving a 40
column screen has been removed.  Almost everyone using CP/M is going to be
using it in 80 columns anyway.  Cutting this code takes a big load off the
system and increases overall speed by about 15%.  Similarly, the interrupt
driven RS232 has been set from 300 to 75 baud.  The higher the baud rate, the
more processor time is required to service RS232.  Since the RS232 code is
always running, decreasing the baud rate frees up cycles that the processor
needs to service RS232.  This doesn't affect the operation of terminal programs
which explicitly set the baud rate when they start up.

Other features of BIOS-R6 include a screen dump function, commented source to
assist the programmer in producing customized systems, and support for
additional disk formats.  Some of the new disk formats include Commodore's
standard 1581 CP/M format, MAXI 71 (398K on 5.25" disks), and GP 1581 (796K on
3.5" disks).

C128 CP/M programmers who want to add or change operating system features
should try to make changes to the BIOS.  For one thing, BIOS source code is
available, but not available for the BDOS or CCP.  (Source code is not
available for the BDOS and CCP replacements mentioned in this article either).
Another reason is that the BDOS and CCP are intended to be "invariable"
operating system components - that is, they are identical for different
computers that run CP/M Plus.  A study of the BIOS source code will reveal
segments of code that can be removed if they aren't needed, and will provide
```

hints as to new features that can be added.

The distribution package, BIOS-R6.LBR includes documentation, source code, utilities, and support files. BIOS-R6.LBR also contains the latest version of ZPM3. [Ed. Note: The files mentioned in this article can be found via anonymous FTP or via the mailserver through the "psend" command.]

ZPM3 Features

ZPM3 is a replacement BDOS by Simeon Cran. Since the BDOS is supposed to be "invariable," why would anyone want to replace it? The answers to that are pretty typical - bug fixes, speed enhancements, and new features! ZPM3 interacts with the BIOS and CCP in most of the same ways as the standard Digital Research BDOS, and for the most part appears to be a clone of the standard BDOS. The standard BDOS was coded in 8080 assembly to make it compatible with machines that use the older slower 8080 processor. Very few (if any) CP/M Plus machines used the 8080. ZPM3 is coded in faster, compact Z80 assembly language, for the Z80 processor that is at the heart of most CP/M Plus computers (including the C128).

The ZPM3 documentation details fixes to several bugs that have plagued CP/M Plus since day one. Although the bugs sound somewhat obscure, there's no telling when one might cause problems.

ZPM3 is much faster than standard CP/M Plus. The increased speed should be obvious after using it for a short time.

The new features offered by ZPM3 are remarkable. Three closely related features are enhanced command line editing, a history buffer that stores and recalls multiple commands, and Automatic Command Prompting. These features work in concert to provide a flexible and convenient command line interface. Command line editing now has 20 control key functions for moving or deleting by characters or whole words. The most recent command lines (up to 250 characters) are stored in the history buffer, and can be recalled and reused, or reedited if necessary. Automatic Command Prompting is best appreciated if seen in action. It's similar to command line completion in Unix, except that it's automatic, with matching responses coming directly from the history buffer. If you've recently entered a long command line with lots of options, and need to reuse it (or edit it slightly first), typing the first few unique characters will bring back the entire command from the history buffer if it's still intact. Automatic Command Prompting is so radical that it might take some getting used to. If you don't think you can get used to it, it can be shut off.

The latest version of ZPM3, ZPM3N08.ARK, is included inside BIOS-R6.LBR, and can also be found as a separate file.

ZCCP Documentation, Version 1.0

The remainder of this article will describe ZCCP and how to configure a system disk to get a fully functional ZPM3/ZCCP system up and running. BIOS-R6 and ZPM3 both come with enough documentation to keep you busy for hours, but ZCCP has never been distributed by itself, because up until this article, there has not been any documentation for it. Most of the documentation that follows was figured out through experimentation and later verified by Simeon Cran.

ZCCP Features

This documentation is provided to assist the user in getting a ZCCP system up and running. It is not an exhaustive course on Z- System or ZCPR. The following list details which ZCPR features are provided with ZCCP, and which ones aren't.

* ZCPR 3.3 compatibility. ZCCP can run a wide range of utilities an applications created for ZCPR 3.3 and ZCPR 3.4.

* TCAP. A Z3T termcap file describing terminal characteristics can be loaded into the system. Z-System programs make use of the TCAP for output to the screen - a big improvement over the old method of patching individual programs with terminal control codes. TCAP files are loaded by the ZCCP LOADSEG command.

* Named directories. User areas can be assigned names. Up to 12 user areas can be assigned names. Named Directory Registers (*.NDR files) are loaded by the ZCCP LOADSEG command.

* Command Search Path. ZCCP will search for commands along a user defined search path. Up to six path elements (directories) can be defined.

* Environment block. Contains TCAP, Named Directory, and Path information. Also includes a map of active disk drives and other system information. The environment block can be viewed with the Z-System SHOW utility.

* Flow control. Conditional processing for batch files. Relies on Z-System IF.COM for setting the flow state. Other flow control commands (FI, ELSE, XIF, OR, AND) are resident.

* Multiple commands can be entered on the command line. The command line buffer will hold up to 225 characters. Commands should be separated by semicolons.

* Extended Command Processor. If a command is not a built-in flow command, resident command, or located on disk along the search path, the command line is passed to an extended command processor. A typical extended command processor is ARUNZ, a sophisticated batch file executor with alias features. To use a program as an extended command processor, rename it to CMDRUN.COM and place it in the ROOT directory of your boot disk.

* Error handler. In the event that the extended command processor can't handle a command, control is passed to an error handler. Error handlers give information about the error (instead of the useless CP/M "?" message) and allow the command line to be edited and reused.

* Resident commands. The following commands are built in:
  CLS - clears the screen
  NOTE - text following the NOTE command is treated as a comment.
  FI   - Flow control: terminate the current
  IF level ELSE - Flow control: toggle the flow state
  XIF - Flow control: exit all pending IF levels
  OR   - Flow control: OR IF tests to set flow state
  AND - Flow control: AND IF tests to set flow state

* Shell stack. Up to four shell levels can be defined. Z-System provides a choice of several different shells. Applications such as terminal programs and word processors can also be assigned shell status.

* ZCCP uses the LOADSEG command for direct loading of RSX files that have not been GENCOMed. Example: LOADSEG SAVE.RSX loads SAVE.RSX.

There are some things that Z3Plus will do that ZCCP won't do.

- ZCCP does not support a Flow Command Package (FCP). It relies on the transient IF command. Other flow commands (FI, ELSE, XIF, OR, AND) are resident in ZCCP.

- A Resident Command Package (RCP) is not implemented. CLS and NOTE are resident in ZCCP. All other commands must be loaded from disk. This isn't as much of a handicap as it might sound if you have a fast RAM drive, such as a CBM 17xx REU, Quick Brown Box, or RAMLink.

- ZCCP can not load type 4 programs (used with ZCPR 3.4). It loads standard COM files at 100H, and type 3 programs that load higher in memory. Most type 4 programs have type 3 or COM equivalents.

- ZCCP can not reexecute loaded programs. This trick is usually performed on Z-Systems with a GO command that jumps to 100H. Since ZCCP also loads at 100H, a GO command would only restart ZCCP.

The Files

Three files are included in ZCCP.ARK:

File name	Size	Description
CCP .COM	3k	ZCCP replacement for CCP.COM
LOADSEG .COM	3k	Loader for named directories and termcaps
ZINSTAL .ZPM	1k	Segment containing environment information

Getting Started - Preparing a Boot Disk

Format a Commodore CP/M format 5.25 or 3.5 inch disk. ZCCP must be booted from device 8 (CP/M drive A).

Copy the files from ZCCP.ARK to user area 0 of the newly formatted disk.

Copy CPM+.SYS to user 0 of the boot disk. The CPM+.SYS must have been generated using the BDOS segments from ZPM3.

Locate a copy of a Z-System alias utility. A good one is SALIAS16, although others should work also. Copy it to user 0 of the boot disk.

At this point, hit the reset switch and boot the system with the new disk. After the system boots, you won't be able to do much with it. The only resident commands are CLS and NOTE, and ZCCP can only locate commands if they are prefixed with the drive and user number.

The next step is to create a startup alias. When ZCCP boots, it looks for a file named STARTZPM.COM and executes commands from it. STARTZPM.COM is created with a ZCPR alias utility. Here is a listing of a STARTZPM.COM created with SALIAS:

```
     ============================================================

     A0>SALIAS STARTZPM

     15:                 ; Logs the ROOT directory (A15) on the
                         ; current drive.

     QD F/F              ; Installs Quick Brown Box ramdisk driver.

     LOADSEG NAMES.NDR C128-XBR.Z3T
                         ; LOADSEG loads the Named Directory Register
                         ; and TCAP.
                         ; Directories can now be referred to by
                         ; name, as in the next command:

     SETPTH10 /C COMMANDS REU 1581 $$$$ $$0 ROOT
                         ; SETPTH sets the command search path.
                         ; The /c option first clears any existing path.
                         ; Directories are then listed in the
                         ; order searched.  In this case, COMMANDS
                         ; is a 64K QBB ramdisk (drive/user F0) where
                         ; frequently used commands are stored.  REU is
                         ; a 1750 REU (drive/user M0).  1581 is a 1581
                         ; drive, (drive/user C15) where some 700K
                         ; of utilities and applications are
                         ; located.  $$$$ refers to the currently
                         ; logged drive and user area.  $$0 refers
                         ; to user area 0 of the current drive.
                         ; The ROOT directory is on drive A, user
                         ; 15, where startup utilities and system
                         ; files can be found.

     1571 [AB            ; This speeds up 1571 disk drives A and B
                         ; by shutting off the redundant write verify.

     AUTOTOG ON          ; Turns on keyboard control of ZPM3 Auto
                         ; Command Prompting.  Auto Command
                         ; Prompting is toggled by entering CTRL-Q.

     COMMANDS:           ; Logs the commands directory.

     IF ~EXIST CP.*      ; Test to see if commands are loaded.
                         ; This line reads:  "If the CP command
                         ; does not exist . . ." and sets the flow
                         ; state to true if the file doesn't exist.
        QD I/F           ; ". . . then initialize the QBB . . ."
        C1:CP C1:*.* F0:
                         ; ". . . copy all of the commands in
                         ; drive/user C1 to the commands (F0)
                         ; directory . . ."
     FI                  ; ". . . end if."

     ROOT:               ; Log the root directory (A15).

     CP C:ZF*.* M0:      ; Copy ZFILER.COM and ZFILER.CMD to the
                         ; REU directory (M0).

     VERROR              ; Install VERROR error handler.

     DATE S              ; Set the system time and date.

     ZF                  ; Invoke ZFILER as a shell.

     ============================================================
```

Of course, your STARTZPM alias will vary depending on the hardware you need to support, your software preferences, and your work habits. This alias is close

to the upward size limit that ZCCP can handle based on the capacity of the multiple command buffer. At the very least, I recommend an alias that will set up a search path and load a TCAP.

Actually, I put the cart before the horse in this example. If you try to reboot your system with the LOADSEG command as listed, you'll notice that you don't have a NAMES.NDR file. There isn't one distributed with ZCCP either. Z-System utilities won't let you edit the NDR either, since the buffer for it hasn't been created yet. This turned out to be a nasty chicken/egg situation, hopefully solved by the inclusion of a sample NAMES.NDR file containing simply A0:SYSTEM and A15:ROOT.

At this point, you should have a mostly functioning ZCCP system disk. Press reset and boot it up. You might want to correct any problems with it or tweak it to perfection before moving on.

List of Z-System Utilities for ZCCP

Some of the following utilities are essential, others are nice to have. The version numbers listed are the latest known versions at the time that this documentation was written. Utilities can be found on ZNode BBSs, and some of them are available on Simtel20 or its mirror sites. Some of the more important utilities will be uploaded to cco.caltech.edu.

SALIAS16 - already mentioned in the example above. SALIAS (or one of the other ZCPR alias utilities) are essential.

ARCOPY - not a ZCPR utility, but one of the best CP/M file copiers ever.

SD138B - excellent DIRectory utility. SD offers many different types of sorts, list formats, etc., displays date stamps, and supports output to a file.

MKDIR32 - utility for manipulating directory names and Named Directory Register (*.NDR) files.

ERASE57 - erases files.

ZFILER10 - a file management shell that can launch applications. It is programmable in that it can execute user defined macros from a file. Multiple files can be "tagged" and operated on by other programs. ZFILER is an excellent program, sort of a GUI desktop without the slow graphics.

C128-XGR - a library of eXtended GRaphics termcaps for the C128. This file is essential if you want to use any ZCPR programs that need a TCAP. These termcaps are the first for the C128 that implement character graphics, standout mode, and control of blinking reverse, and underline modes.

SETPTH10 - used to set the command search path. Essential!

VERROR17 - error handler that displays the command line for reediting. VERROR17 is the only error handler that I found that works with ZCCP.

ZEX50 - Z-System EXecutive is a powerful batch file processor that replaces the CP/M SUBMIT command.

LBRHLP22 - Z-System Help utility displays help files. Help files can be crunched (*.HZP), and/or loaded from a HELP.LBR library.

ARUNZ09 - runs an alias script from a text file. ARUNZ is frequently used as an extended command processor. To use ARUNZ (or any other executable utility) as an extended command processor, rename it to CMDRUN.COM.

VLU102 - Video Library Utility views or extracts files from libraries. Versions of VLU above 1.02 do not work reliably with ZPM3/ZCCP.

Z33IF16 - is the IF.COM discussed in the section on flow control.

SHOW14 - displays an immense amount of information about your Z-System. SHOW also includes a memory patching function.

ZCNFG24 - configures Z-System program options. Most Z-System programs are distributed with a configuration (*.CFG) file that produces a menu of configuration options when run with ZCNFG.

ZP17 - Z-System Patch utility edits files, disk sectors, or memory, and includes a built-in RPN calculator and number base converter.

ZMAN-NEW - This is a manual describing Z-System features in depth. It is based on earlier versions of Z-System, and is a little dated, but otherwise contains information that you won't find anywhere else. Not everything in the manual applies to operation of ZPM3/ZCCP, but with the documentation presented here, you should be able to get a good idea of what works and what doesn't.

ZCCP Technical Notes

ZCCP is a replacement CCP that implements ZCPR 3.3. It loads at 100H and is stored in the bank 0 CCP buffer for fast reloading as does the standard CCP. By contrast, Z3Plus loads into high memory and can be overwritten by transient commands, requiring reloading Z3Plus from disk. Because ZCCP replaces the CCP, a ZCCP system has more TPA (transient program area) than a Z3Plus system. A ZCCP system on the C128 has more than 57K of TPA, almost the same amount as a standard C128 CP/M system.

This should be enough information to get started with ZPM3/ZCCP. Set up a boot disk, experiment with some Z-System utilities, read ZMAN-NEW, and get some applications running. You'll agree that ZPM3/ZCCP breaths new life into CP/M.

==========

Multi-Tasking on the C=128 - Part 1
by Craig Taylor (duck@pembvax1.pembroke.edu)

I. Introduction / Package Over-view..

This article will detail the multi-tasking kernal which I have written butt is still in the debugging stage . The documentation is being released now as C= Hacking has been delayed for a month while this article and a few others were in the process of being shaped. The source code listings, binaries, and a few sample programs will be in the next issue of C= Hacking as well as available on the mailserver and on R. Knop's FTP site when they are available..

The Commodore 128 does not support TRUE multi-tasking in that the processor handles swapping from task to task. Rather the package will make use of the interrupts occuring sixty times a second to determine when to switch tasks.. The Commodore 128 greatly simplifies things as in addation to the interrupts it also has the provision to relocate zero page and the stack page. So the package basically works by intercepting the IRQ vector, taking a look at the current job, saving the stack pointer, finding the next active job, loading the stack page and registers and resuming the normal IRQ as if nothing had ever happened.

Unfortunatly Commodore never thought of having multiple programs in memory executing at any given time. Hence, problems will occur with file accesses, with memory contention, and with an over-all slowdown in speed. The package will detail how to handle device contentions, but it's recommended that programmers make use of the C= 128 kernal call LKUPLA $ff59 containing the logical file number they wish to use in .A; if the carry flag is set upon return then it is safe to use, else find another one as another program is using it. However, note that if you have multiple programs doing this then you may have problems with one grabbing a logical file number after the other process has checked for it. Multi-tasking is fun 'eh? Problems like this will be examined when we get into semaphores later in this article..

Craig Bruce's Dynamic Memory Allocation article in the second issue of C= Hacking should provide a very strong basis for a full-blown memoryy manager. With minor modifications (basically just changing the initial allocations so that the package is not killed) it should be able to work. Also it will need changes to make sure that processes don't try to allocate at the same time. So a memory manager is not too much of a problem. Details of what changes will be necessary shall be in the next issue.

What is a process? What is a program? I've been using the terms almost inter-changebly throughout this article at this point. Basically I'm calling them the same. A process, or program is defined as a program with it's own executable section, it's own data sections, and it's own stack and zero page. (Note, however, that the multi-tasking package does not support relocation of the zero page although this is likely to change). The "kernal" of the multi-tasker is basically that part of the package which governs which process is executed or switched to next. Semaphores will be examined in detail later; they function as flags for processes too know when it is safe to execute something, and serve as signals betweenn them.

Future versions of the package, (even though I know it does not exist out side of my house yet), will support pipes and a more strongly typed kernal so that processes may be prioritized.

II. A Look At Multi-Tasking

The introduction introduced some basic elements of multi-tasking but I'll repeat them here, defining them so that this article can be clear as some of the concepts can get a bit confusing.

Background - A process is said to be in the "backgr ound" if it is not the foreground task and may or may not have input devi ces associated with it.

Foreground - A process is said to be "foreground" if it is the main active process and is holding the keyboard and screen display captive (ie: the user is actually working within it).

Kernal - A small section of code that performs low-leval work that is needed by any programs in memory..

Multi-Tasking - Execution of more than one process at any given time.

Priority - A value associated with each process that determines how often, and possibly when a process is executed.

Process - The space in memory taken up by executable program code, any associated data, the stack and the registers associated and currently in use by it, including the current PC (program counter)..

Semaphores - Values that are globally accessed by processes to share and communicate information between each other and the kernal.

Some CPU's have available a multi- tasking mode (the 386 and 486 are the most famaliar ones that come to mind), y et the 8502 chip contained inside the Commodore 128 was first designed before 1985 and lacks multi-tasking. It would be nice if such a multi-tasking CPU in the 6502 family did exist but it would also create problems with the 6502 style architecture and wouldd produce severe compatibility problems.

So how is the C=128 supposed to do multi-tasking? Well, we'll "simulate" it..

Basically if we had two programs in machine language:

```
            Program 1:                            Program 2:
          - lda #65    ; the "A" character      - lda #64    ; the "@" character
            jsr $ffd2  ; print it                 jsr $ffd2  ; print it
            jmp -                                 jmp --
```

And we wanted them to multi-task we'd expect something like the following:

@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@A@AA

It's unlikely that you'll get that in many multi-tasking environments, even non-simulated ones. Since we're only going to be switching tasks every 1/60 of a second then we're more likely to see an output similair to this:

@@@@@@AAAAAAA@@@@@@@AAAAAAA@@@@@@@@AAAAAAA@@@@@@@AAAAAAAA@@@@@@@@AAAAAA@@@@@@@

So that it seems a process will run for about 1/60 of a second beforee switching to the next one.

We run into problems however. The KERNAL in the C128 that contains most off the file handling, screen manipulations, and keyboard input routines. It was never designed with the idea of multi-tasking in mind. So we're gonna have code running in the KERNAL in two spots for the two differant processes and it's more than likely we'll end up with something like:

@@@@@@@@<and then a BRK or some strange error or never-never-land>>

There's got to be some way to fix it - There is - It's called a semaphore..

A semaphore is a value that is checked before access is granted to another group of memory locations. The semaphore is basically requested via the following:

```
        request_semaphore       sei
                                 ldx semaphore
```

```
                                dex
                                beq +
                                cli
                              - ldy #$ff
                                dey
                                bne -
                              + inc semaphore
                                cli
```

Now the request_semaphore has to disable interrupts to prevent another task from changing the semaphore value before this routine has had a chance. The actual code for the request_semaphore will be very similair to the above.

Using a similair routine that performs the opposite - setting the semaphore to a zero value when finished we can dictate what program has control over what device or what memory areas.

The semaphores will be used to govern access to the KERNAL routines which manipulate the locations in zero page etc, they'll also be used to manage the memory manager when it is implemented as it'd be awkward for it to allocate the same block of memory to two or more processes.

III. Multi-Tasking Function Calls (Package Calls)

OffSet	Name	Notes
$00	SetUp	.C=0 Init Package, .C=1 Uninstall Package (including Kernal re-direction).
$03	SpawnProcess	On return: .C=0 parent, .C = 1 child
$06	ChangePriority	.A = new foreground priority, .X = new background
$09	KillThisProc	Kills Calling Process (no return)
$0c	KillOtherProc	Kills Process # .A
$0f	RequestSemaph	Requests Semaphore #.X
$12	ReleaseSemaph	Releases Semaphore #.X
$14	GetProcInfo	Returns Process Information, Input=#A of 0: Process Id of 1: Process Foreground Priority of 2: Process Background Priority of 3+ Other Information To Be Decided Later
$17	PipeInit	.AY - Address of Pipe, .X = Size/8 Return: .X - Pipe #
$1a	WritePipe	.AY - Address of Null Term. Data, .X = Pipe #
$1d	ReadPipe	.AY - Address to Put Data .X=Pipe #
$20	....	\
$2e	....	\ More Routines for the future.

IV. Availibility of the Packagee

The package should be available at the time of the next issue. A further examination of how the routines work shall be examined along with the source code.

Errors popped up in developing it and rather than delay C= Hacking any further I decided to go ahead and release the above information so that individuals can start developing appropriate routines. In addition, please note that PIPEs _may_ or may not be supported in the next issue. I have not fully made up my mind yet on them.

V. Referencess

Born to Code in C, Herbert Schildt, Osborne-McGraw Hill, p.203-252.

Notes from Operating Systems Course, Pembroke State Univ, Fall '92.

==========

LITTLE RED READER: MS-DOS file reader/WRITER for the C128 and 1571/81.

by Craig Bruce (csbruce@neumann.uwaterloo.ca)

1. INTRODUCTION

This article is a continuation of the Little Red Reader article from last issue. The program has been extended to write MS-DOS files, in addition to reading them. The program still works drive-to-drive so you'll still need two disk drives (either physical or logical) to use it. The program has also been extended to allow MS-DOS files to be deleted and to allow the copying of Commodore-DOS files between CBM-DOS disks (this makes it more convenient to use the program with a temporary logical drive like RAMDOS). Also, since I have recently acquired a CMD FD-4000 floppy disk drive, I know that this program works with MS-DOS disks with this drive (but only for the 720K format).

The program still has the same organization as last time: a menu-oriented user-interface program written in BASIC that makes use of a package of MS-DOS disk accessing routines written in machine language. Oh, this program is Public Domain Software, so feel free to distribute and/or mangle it as you wish. Just note any manglings on the "initializing" screen so people don't blame me.

The program runs on either the 40 or 80-column screens, but you will get much better performance from the BASIC portion of the program by being in 80-column mode and FAST mode. A modification that someone might want to make would be to spread-out the display for the 80-column screen and add color to the rather bland display.

2. USER GUIDE

LOAD and RUN the "lrr.128" BASIC program file. When the program is first run, it will display an "initializing" message and will load in the binary machine language package from the "current" Commodore DOS drive (the current drive is obtained from PEEK(186) - the last device accessed). The binary package is loaded only on the first run and is not reloaded on subsequent runs if the package ID field is in place.

The system is designed to have two file selection menus: one for the MS-DOS disk drive, and one for the Commodore-DOS disk drive (which may be a logical disk drive). The idea for copying is that you select the files in one of these menus, and then program knows to copy them to the disk for the other menu. This idea of having two selection menus is also very consistent with the original program.

2.1. MS-DOS MENU

When the program starts, the MS-DOS menu of the program is displayed. It looks like:

```
MS-DOS  MS=10:1581  CBM=8  FREE=715000

NUM  S  TRN  TYP  FILENAME  EXT  LENGTH
---  -  ---  ---  --------  ---  ------
  1  *  ASC  SEQ  HACK4     TXT  120732
  2     BIN  PRG  RAMDOS    SFX   34923

D=DIR M=MSDEV F=CBMDEV C=COPY Q=QUIT
T=TOGGLE R=REMOVE X=CBMCPY /=MENU +-=PG
```

except that immediately after starting up, "<directory not loaded>" will be displayed rather than filenames. The menu looks and operates pretty much as it did in the last issue of C= Hacking. The only differences are that the number of bytes free on the drive are displayed (which is useful to know when writing files) and there are some more commands.

The directory ("D"), change ms-dos device ("M"), change commodore file device ("F"), toggle column contents ("T"), copy ms-dos files to cbm-dos disk ("C"), quit ("Q"), paging ("+" and "-"), column change (SPACE or RETURN), and the cursor movement commands all work the same as before. They are all sticks to use to flog the beast into submission. The new commands are: "R" (remove == delete), "/" (change menu), and "X" (copy CBM files == "Xerox").

The remove command is used to delete selected files from the MS-DOS disk. After selecting this option, you will get an annoying "are you sure" question and the the selected files will quickly disappear and the changes will finally be written to disk. Deleting a batch of MS-DOS files is much quicker than deleting Commodore-DOS files since MS-DOS disks use a File Allocation Table rather than the linked list of blocks organization that CBM uses. In order to make the BASIC program execute quicker, after deleting, the original order of the filenames in the directory listing will be changed. Be forewarned that the delete operation is non-recoverable.

The change menu command is used to move back and forth between the Commodore-DOS and MS-DOS menus.

2.2. COMMODORE-DOS MENU

The Commodore-DOS menu, which displays the names of the Commodore files selected for various operations, looks and works pretty much the same as the MS-DOS menu:

```
CBMDOS  MS=10:1581  CBM=8  FREE=3211476

NUM  S  TRN  FILENAME         T  LENGTH
---  -  ---  ---------------- -  ------
```

```
   1  *  BIN  LRR-128          P    9876
   2     ASC  COM-HACKING-005  S  175412

  D=DIR M=MSDEV F=CBMDEV C=COPY Q=QUIT
  T=TOGGLE R=REMOVE X=CBMCPY /=MENU +-=PG
```

You'll notice, however, that the filetype field ("T" here) is moved and is unchangable. Also, the file lengths are not exact; they are reported as the block count of the file multiplied by 254. This menu is not maintained for files being copied to the CBM-DOS disk from an MS-DOS disk. You'll have to re-execute the Directory instruction to get an updated listing.

The "D" (directory) command has local effect when in this menu. The Commodore-DOS directory will be loaded from the current CBM device number. Note that in order for this to work, the CBM device must be number eight or greater (a disk drive). Originally, the subroutine for this command was written using only GET#'s from the disk and was very slow. It was modified, however, to call a machine language subroutine to read the information for a directory entry from the directory listing, and hence the subroutine now operates at a tolerable speed.

The "C" (copy) command also has a different meaning when in this menu. It means to copy the selected CBM files to the MS-DOS disk. See details below.

The copy CBM files ("X") command is used to copy the files in the CBM-DOS menu to another CBM-DOS disk unit. Select the files you want to copy and then press X. You will then be asked what device number you want to copy the files to. The device can be another disk drive or any other device (except the keyboard). Using device number 0 does not mean the "null" device as it does with copying MS-DOS to CBM. If you are copying to a disk device and the file already exists, then you will be asked if you wish to overwrite the file. You cannot copy to the same disk unit. Also, all files are copied in binary mode (regardless of what translation you have selected for a file).

The copy CBM files command was included since all of the low-level gear needed to implement it (specifically "commieIn" and "commieOut" below) was also required by other functions. This command can be very convenient when working with RAMDOS. For example, if you only had a 1571 as device 8 but you have a RAM expander and have installed RAMDOS as device 9, then you would copy MS-DOS files to RAMDOS using the MS-DOS menu, and then you would go to the Commodore-DOS menu ("/"), read the directory, select all files, insert an Commodore-DOS diskette into your 1571, and then use "X" to copy from the RAMDOS device to the 1571.

The remove command ("R") does not work for this directory. You can SCRATCH your CBM-DOS files your damn self.

2.3. COPY CBM-DOS TO MS-DOS

Before you can copy selected CBM-DOS files to an MS-DOS disk, the MS-DOS disk directory must be already loaded (from the MS-DOS menu). This is required since the directory and FAT information are kept in memory at all times during the execution of this program.

When you enter copy mode, the screen will clear and the name of each selected file is displayed as it is being copied. If an error is encountered on either the MS-DOS or CBM-DOS drive during copying, an error message will be displayed and copying will continue (after you press a key for MS-DOS errors). Please note that not a whole lot of effort was put into error recovery.

To generate an MS-DOS filename from an CBM-DOS filename, the following algorithm is used. The filename is searched from right to left for the last "." character. If there is no "." character, then the entire filename, up to 11 characters, is used as the MS-DOS filename. Characters 9 to 11 will be used as the extension. If there is a "." character, the all characters before it, up to eight, will be used as the MS-DOS filename and all characters after the final ".", up to three, will be used as the MS-DOS extension.

Then, the newly generated MS-DOS filename is scanned for any extra "." characters or embedded spaces. If any are found, they are replaced by the underscore character ("_", which is the backarrow character on a Commodore display). Finally, all trailing underscores are removed from the end of both the filename and extension portions of the MS-DOS filename. Also, all characters are converted to lowercase PETSCII (which is uppercase ASCII) when they are copied into the MS-DOS filename. Note that if the Commodore filename is not in the 8/3 format of MS-DOS, then something in the name may be lost. Some examples of filename conversion follow:

CBM-DOS FILENAME	MS-DOS FILENAME
"lrr.bin"	"lrr.bin"

```
"lrr.128.bin"          "lrr_128.bin"
"hello there.text"     "hello_th.tex"
"long_filename"        "long_fil.ena"
"file 1..3.s__5"       "file_1.s"
```

It would have been time-consuming to have the program scan the MS-DOS directory for a filename already existing on the disk, so LRR will put multiple files on a disk with the same filename without complaining. This also gets rid of the problem of asking you if you want to overwrite the old file or generate a new name. However, in order to retrieve the file from disk on an MS-DOS machine, you will probably have to use the RENAME command to rename the first versions of the file on the disk to something else so MS-DOS will scan further in the directory for the last version of the file with the same filename. There is no rename command in LRR because I never thought of it in time. It would have been fairly easy to put in.

The date generated for a new MS-DOS file will be all zeros. Some systems interpret this as 12:00 am, 01-Jan-80 and others don't display a date at all for this value.

The physical copying of the file is done completely in machine language and nothing is displayed on the screen while this is happening, but you can follow things by looking at the blinking lights and listening for clicks and grinds.

Since the FAT and directory are maintained in RAM during the entire copying process and are only flushed to disk after the entire batch of files are copied, copying is made more efficient, since there will be no costly seek back to track 0 after writing each file (like MS-DOS does). If you have a number of small files to copy, then they will be knocked off in quick succession, faster than many MS-DOS machines will copy them.

To simplify the implementation, the current track of disk blocks for writing is not maintained like it is for reading. Also, a writing interleave of 1:1 is used for a 1571, which is not optimal. However, since writing is such a slow operation anyway, and since the 1571 is particularly bad by insisting on verifying blocks, not much more overhead is introduced than is already present.

An interesting note about writing MS-DOS disks is that you can terminate LRR in the middle of a copy (with STOP+RESTORE) or in the middle of copying a batch of files, and the MS-DOS disk will remain in a perfectly consistent state afterwards. The state will be as if none of the files were copied. The reason is that the control information (the FAT and directory) is maintained internally and is flushed only after copying is all completed. But don't terminate LRR while it is flushing the control information.

Here is a table of copying speeds for copying to 1571, 1581, and CMD FD-4000 disk units with ASC and BIN translation modes. All figures are in bytes/second, which includes both reading the byte from a C= disk and writing it to the MS-DOS disk. The average speed for either the read or write operation individually will be twice the speed given below. These results were obtained from copying a 156,273 byte text file (the text of C= Hacking Issue #4).

FROM \ TO:	FD-bin	FD-asc	81-bin	81-asc	71-bin	71-asc
RAMLink	2,332	2,200	2,332	2,200	1,594	1,559
RAMDOS	1,070	1,053	1,604	1,600	1,561	1,510
FD4000	-	-	1,645	1,597	1,499	1,464
JD1581	1,662	1,619	-	-	1,474	1,440
JD1571	1,050	1,024	953	933	-	-

These figures are for transfer speed only, not counting the couple of seconds of opening files and flushing the directory. Note that all my physical drives are JiffyDOS-ified, so your performance may be slower. I am at a loss to explain why an FD-4000 is so much slower than a 1581 for copying from a RAMDOS file, but the same speed or better for copying from anything else.

Since I don't have access to an actual MS-DOS machine, I have not tested the files written onto an MS-DOS disk by LRR, except by reading them back with LRR and BBR. I do know, however, that earlier encarnations of this program did work fine with MS-DOS machines.

3. MS-DOS ROOT DIRECTORY

It was brought to my attention that I made a mistake in the pervious article. I was wrong about the offset of the attributes field in a directory entry. The layout should have been as follows:

OFFSET	LEN	DESCRIPTION
0..7	8	Filename

8..10	3	Extension
11	1	Attributes: $10=Directory, $08=VolumeId
12..21	10	<unused>
22..25	4	Date
26..27	2	Starting FAT entry number
28..31	4	File length in bytes

4. FILE COPYING PACKAGE

As was mentioned above, Little Red Reader is split into two pieces: a BASIC front-end user interface program and a package of machine language subroutines for disk accessing. The BASIC program handles the menu, user interaction, and most of the MS-DOS directory searching/modifying. The machine language package handles the hardware input/output, File Allocation Table and file structure manipulations.

The file copying package is written in assembly language and is loaded into memory at address $8000 on bank 0 and requires about 13K of memory. The package is loaded at this high address to be out of the way of the main BASIC program, even if RAMDOS is installed.

This section of the article is presented in its entirety, including all of the information given last time.

4.1. INTERFACE

The subroutine call interface to the file copying package is summarized as follows:

ADDRESS	DESCRIPTION
PK	initPackage subroutine
PK+3	msDir (load MS-DOS directory/FAT) subroutine
PK+6	msRead (copy MS-DOS to CBM-DOS) subroutine
PK+9	msWrite (copy CBM-DOS to MS-DOS) subroutine
PK+12	msFlush subroutine
PK+15	msDelete subroutine
PK+18	msFormat subroutine [not implemented]
PK+21	msBytesFree subroutine
PK+24	cbmCopy (copy CBM-DOS to CBM-DOS) subroutine
PK+27	cbmDirent (read CBM-DOS directory entry) subroutine

where "PK" is the load address of the package ($8000).

The parameter passing interface is summarized as follows:

ADDRESS	DESCRIPTION
PV	two-byte package identification number ($CB, 132)
PV+2	errno : error code returned
PV+3	MS-DOS device number (8 to 30)
PV+4	MS-DOS device type ($00=1571, $FF=1581)
PV+5	two-byte starting cluster number for file copying
PV+7	low and mid bytes of file length for copying
PV+9	pointer to MS-DOS directory entry for writing
PV+11	CBM-DOS file block count
PV+13	CBM-DOS file type ("S"=seq, "P"=prg, etc.)
PV+14	CBM-DOS filename length
PV+15	CBM-DOS filename characters (max 16 chars)

Where "PV" is equal to PK+30. Additional subroutine parameters are passed in the processor registers.

The MS-DOS device number and device type interface variables allow you to set the MS-DOS drive and the package identification number allows the application program to check if the package is already loaded into memory so that it only has to load the package the first time the application is run and not on re-runs. The identification sequence is a value of $CB followed by a value of 132.

4.1.1. INIT_PACKAGE SUBROUTINE

The "initPackage" subroutine should be called when the package is first installed, whenever the MS-DOS device number is changed, and whenever a new disk is mounted to invalidate the internal track cache. It requires no parameters.

4.1.2. MS_DIR SUBROUTINE

The "msDir" subroutine will load the directory, FAT, and the Boot Sector parameters into the internal memory of the package from the current MS-DOS

device number. No (other) input parameters are needed and the subroutine returns a pointer to the directory space in the .AY registers and the number of directory entries in the .X register. If an error occurs, then the subroutine returns with the Carry flag set and the error code is available in the "errno" interface variable. The directory entry data is in the directory space as it was read in raw from the directory sectors on the MS-DOS disk.

4.1.3. MS_READ SUBROUTINE

The "msRead" subroutine will copy a single file from the MS-DOS disk to a specified CBM-Kernal logical file number (the CBM file must already be opened). If the CBM logical file number is zero, then the file data is simply discarded after it is read from the MS-DOS file. The starting cluster number of the file to copy and the low and mid bytes of the file length are passed in the PV+5 and PV+7 interface words. The translation mode to use is passed in the .A register ($00=binary, $FF=ascii) and the CBM logical file number to output to is passed in the .X register. If an error occurs, the routine returns with the Carry flag set and the error code in the "errno" interface variable. There are no other output parameters.

Note that since the starting cluster number and low-file length of the file to be copied are required rather than the filename, it is the responsibility of the front-end application program to dig through the raw directory sector data to get this information. The application must also open the Commodore-DOS file of whatever filetype on whatever device is required; the package does not need to know the Commodore-DOS device number.

4.1.4. MS_WRITE SUBROUTINE

The "msWrite" subroutine copies a single file from a specified CBM-Kernal logical file number to a MS-DOS file. The MS-DOS device number and type are set above. A pointer to the MS-DOS directory entry in the buffer returned by the "msDir" call must be given in interface word PV+9 and the translation mode and CBM lfn are passed in the .A and .X registers as in the "msRead" routine. An error return is given in the usual way (.CS, errno). Otherwise, there are no return values.

It is the responsibility of the calling program to initialize the MS-DOS directory entry to all zeros and then set the filename and set the starting cluster pointer to $0FFF. This routine will update the starting cluster and file length fields of the directory entry when it finishes. The internal "dirty flags" are modified so that the directory and FAT will be flushed on the next call to "msFlush".

4.1.5. MS_FLUSH SUBROUTINE

The "msFlush" subroutine takes no input parameters other than the implicit msDevice and msType. If "dirty" (modified), the FAT will be written to the MS-DOS disk, to both physical replicas of the disk FAT. Then, each directory sector that is dirty will be written to disk. After flushing, the internal dirty flags will be cleared. An error return is given in the usual way. There are no other output parameters. If you call this routine and there are no dirty flags set, then it will return immediately, without any writing to disk.

4.1.6. MS_DELETE SUBROUTINE

The "msDelete" subroutine will deallocate all File Allocation Table entries (and hence, data clusters) allocated to a file and mark the directory entry as being deleted (by putting an $E5 into the first character of the filename). The file is specified by giving the pointer to the directory entry in interface word at PV+9. After deallocating the file data, the internal "dirty" flag will be set for the FAT and the sector that the directory entry is on, but nothing will be written to disk. There is no error return from this routine. It is the responsibility of the calling routine to eventually call the "msFlush" routine.

4.1.7. MS_FORMAT SUBROUTINE

The "msFormat" subroutine is not implemented. It's intended function was to format the MS-DOS disk and generate and write the boot sector, initial FAT, and initial directory entry data.

4.1.8. MS_BYTES_FREE SUBROUTINE

The "msBytesFree" subroutine will scan the currently loaded MS-DOS File Allocation Table, count the number of clusters free, and return the number of bytes free for file storage on the disk. There are no input parameters and the bytes free are returned in the .AYX registers (.A=low, .Y=mid, .X=high byte). The subroutine has no error returns and does not check if an MS-DOS directory is actually loaded.

4.1.9. CBM_COPY SUBROUTINE

The "cbmCopy" subroutine will copy from an input CBM-Kernal logical file number given in the .A register to an output CBM-Kernal lfn given in the .X register, in up to 1024 byte chunks. File contents are copied exactly (no translation). This routine does not care if the lfn's are on the same device or not, but the input device must be a disk unit (either logical or physical). An error return is given in the usual way.

4.1.10. CBM_DIRENT SUBROUTINE

The "cbmDirent" subroutine reads the next directory entry from the CBM-Kernal lfn given in .A and puts the data into interface variables. Of course, the lfn is assumed to be open for reading a directory ("$"). The block count is returned in the word at PV+11, the first character of the filetype is returned at PV+13, the number of characters in the filename is returned in PV+14, and the filename characters are returned in bytes PV+15 to PV+30. An error return is given in the usual way.

This routine assumes that the first two bytes of the directory file have already been read. The first call to this routine will return the name of the disk. The end of a directory is signalled by a filename length of zero. In this case, the block count returned will be the number of blocks free on the disk.

4.2. IMPLEMENTATION

This section presents the code that implements the MS-DOS file reading and writing package. It is here in a special form; each code line is preceded by the % symbol. The % sign is there to allow you to easily extract the assembler code from the rest of this magazine (and all of my ugly comments). On a Unix system, all you have to do is execute the following command line (substitute filenames as appropriate):

```
grep '^%' Hack5 | sed 's/^% //' | sed 's/^%//' >lrr.s
```

```
% ; Little Red Reader/Writer utility package by Craig Bruce, 31-Jan-92
% ; Written for C= Hacking Net-Magazine; for C-128, 1571, 1581
%
```

The code is written for the Buddy assembler and here are a couple setup directives. Note that my comments come before the section of code.

```
% .org $8000
% .obj "lrr.bin"
%
% ;====jump table and parameters interface ====
%
% jmp initPackage ;()
% jmp msDir       ;( msDevice, msType ) : .AY=dirAddr, .X=direntCount
% jmp msRead      ;( msDevice, msType, startCluster, lenML,.A=trans,.X=cbmLfn )
% jmp msWrite     ;( msDevice, msType, writeDirent, .A=trans, .X=cbmLfn )
% jmp msFlush     ;( msDevice, msType )
% jmp msDelete    ;( writeDirent )
% jmp msFormat    ;( msDevice, msType )
% jmp msBytesFree ;( ) : .AYX=bytesFree
% jmp cbmCopy     ;( .A=inLfn, .X=outLfn )
% jmp cbmDirent   ;( .A=lfn )
%
% .byte $cb,132   ;identification (location pk+30)
```

These interface variables are included in the package program space to minimize unwanted interaction with other programs loaded at the same time, such as the RAMDOS device driver.

```
% errno           .buf 1
% msDevice        .buf 1
% msType          .buf 1    ;$00=1571, $ff=1581
% startCluster    .buf 2
% lenML           .buf 2    ;length medium and low bytes
% writeDirent     .buf 2    ;pointer to dirent
% cdirBlocks      .buf 2    ;cbm dirent blocks
% cdirType        .buf 1    ;cbm dirent filetype
% cdirFlen        .buf 1    ;cbm dirent filename length
% cdirName        .buf 16   ;cbm dirent filename
%
```

This command is not currently implemented. Its stub appears here.

```
% msFormat = *
```

```
%    brk
%
% ;====global declaraions====
%
% kernelListen = $ffb1
% kernelSecond = $ff93
% kernelUnlsn  = $ffae
% kernelAcptr  = $ffa2
% kernelCiout  = $ffa8
% kernelSpinp  = $ff47
% kernelChkin  = $ffc6
% kernelChkout = $ffc9
% kernelClrchn = $ffcc
% kernelChrin  = $ffcf
% kernelChrout = $ffd2
%
% st = $90
% ciaClock = $dd00
% ciaFlags = $dc0d
% ciaData  = $dc0c
%
```

These are the parameters and derived parameters from the boot sector. They
are kept in the program space to avoid interactions.

```
% clusterBlockCount .buf 1        ;1 or 2
% fatBlocks         .buf 1        ;up to 3
% rootDirBlocks     .buf 1        ;up to 8
% rootDirEntries    .buf 1        ;up to 128
% totalSectors      .buf 2        ;up to 1440
% firstFileBlock    .buf 1
% firstRootDirBlock .buf 1
% fileClusterCount  .buf 2
% lastFatEntry      .buf 2
%
```

The cylinder (track) and side that is currently stored in the track cache
for reading.

```
% bufCylinder     .buf 1
% bufSide         .buf 1
```

These "dirty" flags record what has to be written out for a flush operation.

```
% fatDirty        .buf 1
% dirDirty        .buf 8  ;flag for each directory block
% formatParms     .buf 6
%
```

This package is split into a number of levels. This level interfaces with the
Kernal serial bus routines and the burst command protocol of the disk drives.

```
% ;====hardware level====
%
```

Connect to the MS-DOS device and send the "U0" burst command prefix and the
burst command byte.

```
% sendU0 = *  ;( .A=burstCommandCode ) : .CS=err
%    pha
%    lda #0
%    sta st
%    lda msDevice
%    jsr kernelListen
%    lda #$6f
%    jsr kernelSecond
%    lda #"u"
%    jsr kernelCiout
%    bit st
%    bmi sendU0Error
%    lda #"0"
%    jsr kernelCiout
%    pla
%    jsr kernelCiout
%    bit st
%    bmi sendU0Error
%    clc
%    rts
%
%    sendU0Error = *
%    lda #5
```

```
%    sta errno
%    sec
%    rts
%
```

Toggle the "Data Accepted / Ready For More" clock signal for the burst transfer protocol.

```
% toggleClock = *
%    lda ciaClock
%    eor #$10
%    sta ciaClock
%    rts
%
```

Wait for a burst byte to arrive in the serial data register of CIA#1 from the fast serial bus.

```
% serialWait = *
%    lda #$08
% -  bit ciaFlags
%    beq -
%    rts
%
```

Wait for and get a burst byte from the fast serial bus, and send the "Data Accepted" signal.

```
% getBurstByte = *
%    jsr serialWait
%    ldx ciaData
%    jsr toggleClock
%    txa
%    rts
%
```

Send the burst commands to "log in" the MS-DOS disk and set the Read sector interleave factor.

```
% mountDisk = *  ;() : .CS=err
%    lda #%00011010
%    jsr sendU0
%    bcc +
%    rts
% +  jsr kernelUnlsn
%    bit st
%    bmi sendU0Error
%    clc
%    jsr kernelSpinp
%    bit ciaFlags
%    jsr toggleClock
%    jsr getBurstByte
%    sta errno
%    and #$0f
%    cmp #2
%    bcs mountExit
```

Grab the throw-away parameters from the mount operation.

```
%    ldy #0
% -  jsr getBurstByte
%    sta formatParms,y
%    iny
%    cpy #6
%    bcc -
%    clc
```

Set the Read sector interleave to 1 for a 1581 or 4 for a 1571.

```
%    ;** set interleave
%    lda #%00001000
%    jsr sendU0
%    bcc +
%    rts
% +  lda #1            ;interleave of 1 for 1581
%    bit msType
%    bmi +
%    lda #4            ;interleave of 4 for 1571
% +  jsr kernelCiout
%    jsr kernelUnlsn
%    mountExit = *
```

```
%    rts
%
```

Read all of the sectors of a given track into the track cache.

```
% bufptr = 2
% secnum = 4
%
% readTrack = *  ;( .A=cylinder, .X=side ) : trackbuf, .CS=err
%    pha
%    txa
```

Get the side and put it into the command byte. Remember that we have to flip the side bit for a 1581.

```
%    and #$01
%    asl
%    asl
%    asl
%    asl
%    bit msType
%    bpl +
%    eor #$10
% +  jsr sendU0
%    pla
%    bcc +
%    rts
% +  jsr kernelCiout      ;cylinder number
%    lda #1               ;start sector number
%    jsr kernelCiout
%    lda #9               ;sector count
%    jsr kernelCiout
%    jsr kernelUnlsn
```

Prepare to receive the track data.

```
%    sei
%    clc
%    jsr kernelSpinp
%    bit ciaFlags
%    jsr toggleClock
%    lda #<trackbuf
%    ldy #>trackbuf
%    sta bufptr
%    sty bufptr+1
```

Get the sector data for each of the 9 sectors of the track.

```
%    lda #0
%    sta secnum
% -  bit msType
%    bmi +
```

If we are dealing with a 1571, we have to set the buffer pointer for the next sector, taking into account the soft interleave of 4.

```
%    jsr get1571BufPtr
% +  jsr readSector
%    bcs trackExit
%    inc secnum
%    lda secnum
%    cmp #9
%    bcc -
%    clc
%    trackExit = *
%    cli
%    rts
%
```

Get the buffer pointer for the next 1571 sector.

```
% get1571BufPtr = *
%    lda #<trackbuf
%    sta bufptr
%    ldx secnum
%    clc
%    lda #>trackbuf
%    adc bufptr1571,x
%    sta bufptr+1
%    rts
%
```

```
% bufptr1571 = *
%    .byte 0,8,16,6,14,4,12,2,10
%
```

Read an individual sector into memory at the specified address.

```
% readSector = *  ;( bufptr ) : .CS=err
```

Get and check the burst status byte for errors.

```
%    jsr getBurstByte
%    sta errno
%    and #$0f
%    cmp #2
%    bcc +
%    rts
% +  ldx #2
%    ldy #0
%
```

Receive the 512 sector data bytes into memory.

```
%    readByte = *
%    lda #$08
% -  bit ciaFlags
%    beq -
%    lda ciaClock
%    eor #$10
%    sta ciaClock
%    lda ciaData
%    sta (bufptr),y
%    iny
%    bne readByte
%    inc bufptr+1
%    dex
%    bne readByte
%    rts
%
% oldClock = 5
%
```

Write an individual sector to disk, from a specified memory address.

```
% writeSector = *  ;( bufptr, .A=track, .X=side, .Y=sector ) : .CS=err
%    pha
%    sty secnum
```

Get the side into the burst command byte

```
%    txa
%    and #$01
%    asl
%    asl
%    asl
%    asl
%    ora #$02
%    bit msType
%    bpl +
%    eor #$10
% +  jsr sendU0
%    pla
%    bcc +
%    rts
```

Send rest of parameters for burst command.

```
% +  jsr kernelCiout      ;track number
%    lda secnum           ;sector number
%    jsr kernelCiout
%    lda #1               ;sector count
%    jsr kernelCiout
%    jsr kernelUnlsn
%    sei
%    lda #$40
%    sta oldClock
%    sec
%    jsr kernelSpinp      ;set for burst output
%    sei
%    bit ciaFlags
%    ldx #2
%    ldy #0
```

```
%
```

Write the 512 bytes for the sector.

```
%    writeByte = *
%    lda ciaClock
%    cmp ciaClock
%    bne writeByte
%    eor oldClock
%    and #$40
%    beq writeByte
%    lda (bufptr),y
%    sta ciaData
%    lda oldClock
%    eor #$40
%    sta oldClock
%    lda #8
% -  bit ciaFlags
%    beq -
%    iny
%    bne writeByte
%    inc bufptr+1
%    dex
%    bne writeByte
%
```

Read back the burst status byte to see if anything went wrong with the write.

```
%    clc
%    jsr kernelSpinp
%    bit ciaFlags
%    jsr toggleClock
%    jsr serialWait
%    ldx ciaData
%    jsr toggleClock
%    txa
%    sta errno
%    and #$0f
%    cmp #2
%    cli
%    rts
%
```

This next level of routines deals with logical sectors and the track cache
rather than with hardware.

```
% ;====logical sector level====
%
```

Invalidate the track cache if the MS-DOS drive number is changed or if a new
disk is inserted. This routine has to establish a RAM configuration of $0E
since it will be called from RAM0. Configuration $0E gives RAM0 from $0000 to
$BFFF, Kernal ROM from $C000 to $FFFF, and the I/O space over the Kernal from
$D000 to $DFFF. This configuration is set by all application interface
subroutines.

```
% initPackage = *
%    lda #$0e
%    sta $ff00
%    lda #$ff
%    sta bufCylinder
%    sta bufSide
%    ldx #7
% -  sta dirDirty,x
%    dex
%    bpl -
%    sta fatDirty
%    clc
%    rts
%
```

Locate a sector (block) in the track cache, or read the corresponding physical
track into the track cache if necessary. This routine accepts the cylinder,
side, and sector numbers of the block.

```
% sectorSave = 5
%
% readBlock = *  ;( .A=cylinder,.X=side,.Y=sector ) : .AY=blkPtr,.CS=err
```

Check if the correct track is in the track cache.

```
%    cmp bufCylinder
%    bne readBlockPhysical
%    cpx bufSide
%    bne readBlockPhysical

If so, then locate the sector's address and return that.

%    dey
%    tya
%    asl
%    clc
%    adc #>trackbuf
%    tay
%    lda #<trackbuf
%    clc
%    rts
%

Here, we have to read the physical track into the track cache.  We save the
input parameters and call the hardware-level track-reading routine.

%    readBlockPhysical = *
%    sta bufCylinder
%    stx bufSide
%    sty sectorSave
%    jsr readTrack

Check for errors.

%    bcc readBlockPhysicalOk
%    lda errno
%    and #$0f
%    cmp #11    ;disk change
%    beq +
%    sec
%    rts

If the error that happened is a "Disk Change" error, then mount the disk and
try to read the physical track again.

% +  jsr mountDisk
%    lda bufCylinder
%    ldx bufSide
%    ldy sectorSave
%    bcc readBlockPhysical
%    rts
%

Here, the physical track has been read into the track cache ok, so we recover
the original input parameters and try the top of the routine again.

%    readBlockPhysicalOk = *
%    lda bufCylinder
%    ldx bufSide
%    ldy sectorSave
%    jmp readBlock
%

Divide the given number by 18.  This is needed for the calculations to convert
a logical sector number to the corresponding physical cylinder, side, and
sector numbers that the lower-level routines require.  The method of repeated
subtraction is used.  This routine would probably work faster if we tried to
repeatedly subtract 360 (18*20) at the top, but I didn't bother.

% divideBy18 = *  ;( .AY=number ) : .A=quotient, .Y=remainder
%    ;** could repeatedly subtract 360 here
%    ldx #$ff
% -  inx
%    sec
%    sbc #18
%    bcs -
%    dey
%    bpl -
%    clc
%    adc #18
%    iny
%    tay
%    txa
%    rts
%
```

Convert the given logical block number to the corresponding physical cylinder, side, and sector numbers. This routine follows the formulae given in the previous article with a few simplifying tricks.

```
% convertLogicalBlockNum = *  ;( .AY=blockNum ) : .A=cyl, .X=side,.Y=sec
%    jsr divideBy18
%    ldx #0
%    cpy #9
%    bcc +
%    pha
%    tya
%    sbc #9
%    tay
%    pla
%    ldx #1
% +  iny
%    rts
%
```

Copy a sequential group of logical sectors into memory. This routine is used by the directory loading routine to load the FAT and Root Directory, and is used by the cluster reading routine to retrieve all of the blocks of a cluster. After the given starting logical sector number is converted into its physical cylinder, side, and sector equivalent, the physical values are incremented to get the address of successive sectors of the group. This avoids the overhead of the logical to physical conversion. Quite a number of temporaries are needed.

```
% destPtr = 6
% curCylinder = 8
% curSide = 9
% curSector = 10
% blockCountdown = 11
% sourcePtr = 12
%
% copyBlocks = *  ;( .AY=startBlock, .X=blockCount, ($6)=dest ) : .CS=err
%    stx blockCountdown
%    jsr convertLogicalBlockNum
%    sta curCylinder
%    stx curSide
%    sty curSector
%
%    copyBlockLoop = *
%    lda curCylinder
%    ldx curSide
%    ldy curSector
%    jsr readBlock
%    bcc +
%    rts
% +  sta sourcePtr
%    sty sourcePtr+1
%    ldx #2
%    ldy #0
```

Here I unroll the copying loop a little bit to cut the overhead of the branch instruction in half. (A cycle saved... you know).

```
% -  lda (sourcePtr),y
%    sta (destPtr),y
%    iny
%    lda (sourcePtr),y
%    sta (destPtr),y
%    iny
%    bne -
%    inc sourcePtr+1
%    inc destPtr+1
%    dex
%    bne -
```

Increment the cylinder, side, sector values.

```
%    inc curSector
%    lda curSector
%    cmp #10
%    bcc +
%    lda #1
%    sta curSector
%    inc curSide
%    lda curSide
%    cmp #2
%    bcc +
```

```
%    lda #0
%    sta curSide
%    inc curCylinder
% +  dec blockCountdown
%    bne copyBlockLoop
%    clc
%    rts
%
```

Convert a given cluster number into the first corresponding logical block number.

```
% convertClusterNum = *  ;( .AY=clusterNum ) : .AY=logicalBlockNum
%    sec
%    sbc #2
%    bcs +
%    dey
% +  ldx clusterBlockCount
%    cpx #1
%    beq +
%    asl
%    sty 7
%    rol 7
%    ldy 7
% +  clc
%    adc firstFileBlock
%    bcc +
%    iny
% +  rts
%
```

Read a cluster into the Cluster Buffer, given the cluster number. The cluster number is converted to a logical sector number and then the sector copying routine is called. The formula given in the previous article is used.

```
% readCluster = *  ;( .AY=clusterNumber ) : clusterBuf, .CS=err
%    jsr convertClusterNum
%
%    ;** read logical blocks comprising cluster
%    ldx #<clusterBuf
%    stx 6
%    ldx #>clusterBuf
%    stx 7
%    ldx clusterBlockCount
%    jmp copyBlocks
%
```

Write a logical block out to disk. The real purpose of this routine is to invalidate the read-track cache if the block to be written is contained in the cache.

```
% writeLogicalBlock = *  ;( .AY=logicalBlockNumber, bufptr ) : .CS=err
%    jsr convertLogicalBlockNum
%    cmp bufCylinder
%    bne +
%    cpx bufSide
%    bne +
%    pha
%    lda #$ff
%    sta bufCylinder
%    sta bufSide
%    pla
% +  jsr writeSector
%    rts
%
% writeClusterSave .buf 2
%
```

Write a cluster-ful of data out to disk from the cluster buffer. This routine simply calls the write logical block routine once or twice, depending on the cluster size of the disk involved.

```
% writeCluster = *  ;( .AY=clusterNumber, clusterBuf ) : .CS=err
%    jsr convertClusterNum
%    ldx #<clusterBuf
%    stx bufptr
%    ldx #>clusterBuf
%    stx bufptr+1
%    sta writeClusterSave
%    sty writeClusterSave+1
%    jsr writeLogicalBlock
```

```
%    bcc +
%    rts
% +  lda clusterBlockCount
%    cmp #2
%    bcs +
%    rts
% +  lda writeClusterSave
%    ldy writeClusterSave+1
%    clc
%    adc #1
%    bcc +
%    iny
% +  jsr writeLogicalBlock
%    rts
%
```

This next level of routines deal with the data structures of the MS-DOS disk format.

```
% ;====MS-DOS format level====
%
% bootBlock = 2
%
```

Read the disk format parameters, directory, and FAT into memory.

```
% msDir = *  ;( ) : .AY=dirbuf, .X=dirEntries, .CS=err
%    lda #$0e
%    sta $ff00
%
```

Read the boot sector and extract the parameters.

```
%    ;** get parameters from boot sector
%    lda #0
%    ldy #0
%    jsr convertLogicalBlockNum
%    jsr readBlock
%    bcc +
%    rts
% +  sta bootBlock
%    sty bootBlock+1
%    ldy #13              ;get cluster size
%    lda (bootBlock),y
%    sta clusterBlockCount
%    cmp #3
%    bcc +
%
```

If a disk parameter is found to exceed the limits of LRR, error code #60 is returned.

```
%    invalidParms = *
%    lda #60
%    sta errno
%    sec
%    rts
%
% +  ldy #16              ;check FAT replication count, must be 2
%    lda (bootBlock),y
%    cmp #2
%    bne invalidParms
%    ldy #22              ;get FAT size in sectors
%    lda (bootBlock),y
%    sta fatBlocks
%    cmp #4
%    bcs invalidParms
%    ldy #17              ;get directory size
%    lda (bootBlock),y
%    sta rootDirEntries
%    cmp #129
%    bcs invalidParms
%    lsr
%    lsr
%    lsr
%    lsr
%    sta rootDirBlocks
%    ldy #19              ;get total sector count
%    lda (bootBlock),y
%    sta totalSectors
%    iny
```

```
%    lda (bootBlock),y
%    sta totalSectors+1
%    ldy #24              ;check sectors per track, must be 9
%    lda (bootBlock),y
%    cmp #9
%    bne invalidParms
%    ldy #26
%    lda (bootBlock),y
%    cmp #2               ;check number of sides, must be 2
%    bne invalidParms
%    ldy #14              ;check number of boot sectors, must be 1
%    lda (bootBlock),y
%    cmp #1
%    bne invalidParms
%
```

Calculate the derived parameters.

```
%    ;** get derived parameters
%    lda fatBlocks        ;first root directory sector
%    asl
%    clc
%    adc #1
%    sta firstRootDirBlock
%    clc                  ;first file sector
%    adc rootDirBlocks
%    sta firstFileBlock
%    lda totalSectors     ;number of file clusters
%    ldy totalSectors+1
%    sec
%    sbc firstFileBlock
%    bcs +
%    dey
% +  sta fileClusterCount
%    sty fileClusterCount+1
%    lda clusterBlockCount
%    cmp #2
%    bne +
%    lsr fileClusterCount+1
%    ror fileClusterCount
% +  clc
%    lda fileClusterCount
%    adc #2
%    sta lastFatEntry
%    lda fileClusterCount+1
%    adc #0
%    sta lastFatEntry+1
%
%    ;** load FAT
%    lda #<fatbuf
%    ldy #>fatbuf
%    sta 6
%    sty 7
%    lda #1
%    ldy #0
%    ldx fatBlocks
%    jsr copyBlocks
%    bcc +
%    rts
%
%    ;** load actual directory
% +  lda #<dirbuf
%    ldy #>dirbuf
%    sta 6
%    sty 7
%    lda firstRootDirBlock
%    ldy #0
%    ldx rootDirBlocks
%    jsr copyBlocks
%    bcc +
%    rts
% +  lda #<dirbuf
%    ldy #>dirbuf
%    ldx rootDirEntries
%    clc
%    rts
%
```

This routine locates the given FAT table entry number and returns the value stored in it. Some work is needed to deal with the 12-bit compressed data structure.

```
% entryAddr = 2
% entryWork = 4
% entryBits = 5
% entryData0 = 6
% entryData1 = 7
% entryData2 = 8
%
% locateFatEntry = *  ;( .AY=fatEntryNumber ) : entryAddr, entryBits%1
```

Divide the FAT entry number by two and multiply by three because two FAT entries are stored in three bytes. Then add the FAT base address and we have the address of the three bytes that contain the FAT entry we are interested in. I retrieve the three bytes into zero-page memory for easy manipulation.

```
%    sta entryBits
%    ;** divide by two
%    sty entryAddr+1
%    lsr entryAddr+1
%    ror
%
%    ;** times three
%    sta entryWork
%    ldx entryAddr+1
%    asl
%    rol entryAddr+1
%    clc
%    adc entryWork
%    sta entryAddr
%    txa
%    adc entryAddr+1
%    sta entryAddr+1
%
%    ;** add base, get data
%    clc
%    lda entryAddr
%    adc #<fatbuf
%    sta entryAddr
%    lda entryAddr+1
%    adc #>fatbuf
%    sta entryAddr+1
%    ldy #2
% -  lda (entryAddr),y
%    sta entryData0,y
%    dey
%    bpl -
%    rts
%
% getFatEntry = *  ;( .AY=fatEntryNumber ) : .AY=fatEntryValue
%    jsr locateFatEntry
%    lda entryBits
%    and #1
%    bne +
%
```

If the original given FAT entry number is even, then we want the first 12-bit compressed field. The nybbles are extracted according to the diagram shown earlier.

```
%    ;** case 1: first 12-bit cluster
%    lda entryData1
%    and #$0f
%    tay
%    lda entryData0
%    rts
%
```

Otherwise, we want the second 12-bit field.

```
%    ;** case 2: second 12-bit cluster
% +  lda entryData1
%    ldx #4
% -  lsr entryData2
%    ror
%    dex
%    bne -
%    ldy entryData2
%    rts
%
% fatValue = 9
%
```

Change the value in a FAT entry. This routine is quite similar to the get
routine.

```
% setFatEntry = *  ;( .AY=fatEntryNumber, (fatValue) )
%    jsr locateFatEntry
%    lda fatValue+1
%    and #$0f
%    sta fatValue+1
%    lda entryBits
%    and #1
%    bne +
%
%    ;** case 1: first 12-bit cluster
%    lda fatValue
%    sta entryData0
%    lda entryData1
%    and #$f0
%    ora fatValue+1
%    sta entryData1
%    jmp setFatExit
%
%    ;** case 2: second 12-bit cluster
% +  ldx #4
% -  asl fatValue
%    rol fatValue+1
%    dex
%    bne -
%    lda fatValue+1
%    sta entryData2
%    lda entryData1
%    and #$0f
%    ora fatValue
%    sta entryData1
%
%    setFatExit = *
%    ldy #2
% -  lda entryData0,y
%    sta (entryAddr),y
%    dey
%    bpl -
%    sty fatDirty
%    rts
%
```

Mark the directory sector corresponding to the given directory entry as being
dirty so it will be written out to disk the next time the msFlush routine is
called.

```
% dirtyDirent = *  ;( writeDirent )
%    sec
%    lda writeDirent
%    sbc #<dirbuf
%    lda writeDirent+1
%    sbc #>dirbuf
%    lsr
%    and #$07
%    tax
%    lda #$ff
%    sta dirDirty,x
%    rts
%
% delCluster = 14
%
```

Delete the MS-DOS file whose directory entry is given. Put the $E5 into
its filename, get its starting cluster and follow the chain of clusters
allocated to the file in the FAT, marking them as unallocated (value $000)
as we go. Exit by marking the directory entry as "dirty".

```
% msDelete = *  ;( writeDirent )
%    ldy #$0e
%    sty $ff00
%    lda writeDirent
%    ldy writeDirent+1
%    sta 2
%    sty 3
%    lda #$e5
%    ldy #0
%    sta (2),y
%    ldy #26
```

```
%    lda (2),y
%    sta delCluster
%    iny
%    lda (2),y
%    sta delCluster+1
% -  lda delCluster+1
%    cmp #5
%    bcc +
%    jmp dirtyDirent
% +  tay
%    lda delCluster
%    jsr getFatEntry
%    pha
%    tya
%    pha
%    lda #0
%    sta fatValue
%    sta fatValue+1
%    lda delCluster
%    ldy delCluster+1
%    jsr setFatEntry
%    pla
%    sta delCluster+1
%    pla
%    sta delCluster
%    jmp -
%
% flushBlock = 14
% flushCountdown = $60
% flushRepeats = $61
% flushDirIndex = $61
%
```

Write the FAT and directory sectors from memory to disk, if they are dirty.

```
% msFlush = *  ;( msDevice, msType ) : .CS=error
%    lda #$0e
%    sta $ff00
%    lda fatDirty
%    beq flushDirectory
%    lda #0
%    sta fatDirty
%
%    ;** flush fat
```

Flush both copies of the FAT, if there are two; otherwise, only flush the one.

```
%    lda #2
%    sta flushRepeats
%    lda #1
%    sta flushBlock
%
%    masterFlush = *
%    lda fatBlocks
%    sta flushCountdown
%    lda #<fatbuf
%    ldy #>fatbuf
%    sta bufptr
%    sty bufptr+1
% -  lda flushBlock
%    ldy #0
%    jsr writeLogicalBlock
%    bcc +
%    rts
% +  inc flushBlock
%    dec flushCountdown
%    bne -
%    dec flushRepeats
%    bne masterFlush
%
%    ;** flush directory
%    flushDirectory = *
%    lda firstRootDirBlock
%    sta flushBlock
%    lda rootDirBlocks
%    sta flushCountdown
%    lda #0
%    sta flushDirIndex
%    lda #<dirbuf
%    ldy #>dirbuf
%    sta bufptr
```

```
%    sty bufptr+1
% -  ldx flushDirIndex
%    lda dirDirty,x
%    beq +
%    lda #0
%    sta dirDirty,x
%    lda flushBlock
%    ldy #0
%    jsr writeLogicalBlock
%    dec bufptr+1
%    dec bufptr+1
% +  inc flushBlock
%    inc flushDirIndex
%    inc bufptr+1
%    inc bufptr+1
%    dec flushCountdown
%    bne -
%    clc
%    rts
%
% bfFatEntry = 14
% bfBlocks = $60
%
```

Count the number of free FAT entries (value $000) from entry 2 up to the highest FAT entry available for cluster allocation. Then multiply this by the number of bytes per cluster (either 512 or 1024).

```
% msBytesFree = *  ;( ) : .AYX=fileBytesFree
%    ldy #$0e
%    sty $ff00
%    lda #2
%    ldy #0
%    sta bfFatEntry
%    sty bfFatEntry+1
%    sty bfBlocks
%    sty bfBlocks+1
% -  lda bfFatEntry
%    ldy bfFatEntry+1
%    jsr getFatEntry
%    sty 2
%    ora 2
%    bne +
%    inc bfBlocks
%    bne +
%    inc bfBlocks+1
% +  inc bfFatEntry
%    bne +
%    inc bfFatEntry+1
% +  lda bfFatEntry
%    cmp lastFatEntry
%    lda bfFatEntry+1
%    sbc lastFatEntry+1
%    bcc -
%    ldx clusterBlockCount
% -  asl bfBlocks
%    rol bfBlocks+1
%    dex
%    bne -
%    lda #0
%    ldy bfBlocks
%    ldx bfBlocks+1
%    rts
%
```

This is the file copying level. It deals with reading/writing the clusters of MS-DOS files and copying the data they contain to/from the already-open CBM Kernal file, possibly with ASCII/PETSCII translation.

```
% ;====file copy level====
%
% transMode = 14
% lfn = 15
% cbmDataPtr = $60
% cbmDataLen = $62
% cluster = $64
%
```

Copy the given cluster to the CBM output file. This routine fetches the next cluster of the file for the next time this routine is called, and if it hits the NULL pointer of the last cluster of a file, it adjusts the number of valid

```
file data bytes the current cluster contains to FileLength % ClusterLength
(see note below).

% copyFileCluster = *  ;( cluster, lfn, transMode ) : .CS=err

Read the cluster and setup to copy the whole cluster to the CBM file.

%    lda cluster
%    ldy cluster+1
%    jsr readCluster
%    bcc +
%    rts
% +  lda #<clusterBuf
%    ldy #>clusterBuf
%    sta cbmDataPtr
%    sty cbmDataPtr+1
%    lda #0
%    sta cbmDataLen
%    lda clusterBlockCount
%    asl
%    sta cbmDataLen+1
%

Fetch the next cluster number of the file, and adjust the cluster data length
for the last cluster of the file.

%    ;**get next cluster
%    lda cluster
%    ldy cluster+1
%    jsr getFatEntry
%    sta cluster
%    sty cluster+1
%    cpy #$05
%    bcc copyFileClusterData
%    lda lenML
%    sta cbmDataLen
%    lda #$01
%    ldx clusterBlockCount
%    cpx #1
%    beq +
%    lda #$03
% +  and lenML+1

The following three lines were added in a last minute panic after realizing
that if FileLength % ClusterSize == 0, then the last cluster of the file
contains ClusterSize bytes, not zero bytes.

%    bne +
%    ldx lenML
%    beq copyFileClusterData
% +  sta cbmDataLen+1
%
%    copyFileClusterData = *
%    jsr commieOut
%    rts
%

Copy the file data in the MS-DOS cluster buffer to the CBM output file.

% cbmDataLimit = $66
%
% commieOut = *  ;( cbmDataPtr, cbmDataLen ) : .CS=err

If the the logical file number to copy to is 0 ("null device"), then don't
bother copying anything.

%    ldx lfn
%    bne +
%    clc
%    rts

Otherwise, prepare the logical file number for output.

% +  jsr kernelChkout
%    bcc commieOutMore
%    sta errno
%    rts
%

Process the cluster data in chunks of up to 255 bytes or the number of data
bytes remaining in the cluster.
```

```
%    commieOutMore = *
%    lda #255
%    ldx cbmDataLen+1
%    bne +
%    lda cbmDataLen
% +  sta cbmDataLimit
%    ldy #0
% -  lda (cbmDataPtr),y
%    bit transMode
%    bpl +
```

If we have to translate the current ASCII character, look up the PETSCII value in the translation table and output that value. If the translation table entry value is $00, then don't output a character (filter out invalid character codes).

```
%    tax
%    lda transBuf,x
%    beq commieNext
% +  jsr kernelChrout
%    commieNext = *
%    iny
%    cpy cbmDataLimit
%    bne -
%
```

Increment the cluster buffer pointer and decrement the cluster buffer character count according to the number of bytes just processed, and repeat the above if more file data remains in the current cluster.

```
%    clc
%    lda cbmDataPtr
%    adc cbmDataLimit
%    sta cbmDataPtr
%    bcc +
%    inc cbmDataPtr+1
% +  sec
%    lda cbmDataLen
%    sbc cbmDataLimit
%    sta cbmDataLen
%    bcs +
%    dec cbmDataLen+1
% +  lda cbmDataLen
%    ora cbmDataLen+1
%    bne commieOutMore
```

If we are finished with the cluster, then clear the CBM Kernal output channel.

```
%    jsr kernelClrchn
%    clc
%    rts
%
```

The file copying main routine. Set up for the starting cluster, and call the cluster copying routine until end-of-file is reached. Checks for a NULL cluster pointer in the directory entry to handle zero-length files.

```
% msRead = *  ;( cluster, lenML, .A=transMode, .X=lfn ) : .CS=err
%    ldy #$0e
%    sty $ff00
%    sta transMode
%    stx lfn
%    lda startCluster
%    ldy startCluster+1
%    sta cluster
%    sty cluster+1
%    jmp +
% -  jsr copyFileCluster
%    bcc +
%    rts
% +  lda cluster+1
%    cmp #$05
%    bcc -
%    clc
%    rts
%
% inLfn = $50
% generateLf = $51
% cbmDataMax = $52
% reachedEof = $54
```

```
% prevSt = $55
%

Set the translation and input logical file number and set up for reading
from a CBM-Kernal input file.

% commieInInit = *  ;( .A=transMode, .X=inLfn )
%    sta transMode
%    stx inLfn
%    lda #0
%    sta generateLf
%    sta reachedEof
%    sta prevSt
%    rts
%

Read up to "cbmDataMax" bytes into the specified buffer from the established
CBM logical file number.  The number of bytes read is returned in
"cbmDataLen".  If end of file occurs, "cbmDataLen" will be zero and the .Z
flag will be set.  Regular error return.

% commieIn = *  ;( cbmDataPtr++, cbmDataMax ) : cbmDataLen, .CS=err, .Z=eof

Establish input file, or return immediately if already past eof.

%    lda #0
%    sta cbmDataLen
%    sta cbmDataLen+1
%    ldx reachedEof
%    beq +
%    lda #0
%    clc
%    rts
% +  ldx inLfn
%    jsr kernelChkin
%    bcc commieInMore
%    sta errno
%    rts
%

Read next chunk of up to 255 bytes into input buffer.

%    commieInMore = *
%    lda #255
%    ldx cbmDataMax+1
%    bne +
%    lda cbmDataMax
% +  sta cbmDataLimit
%    ldy #0
% -  jsr commieInByte
%    bcc +
%    rts
% +  beq +
%    sta (cbmDataPtr),y
%    iny
%    cpy cbmDataLimit
%    bne -
%

Prepare to read another chunk, or exit.

% +  sty cbmDataLimit
%    clc
%    lda cbmDataPtr
%    adc cbmDataLimit
%    sta cbmDataPtr
%    bcc +
%    inc cbmDataPtr+1
% +  clc
%    lda cbmDataLen
%    adc cbmDataLimit
%    sta cbmDataLen
%    bcc +
%    inc cbmDataLen+1
% +  sec
%    lda cbmDataMax
%    sbc cbmDataLimit
%    sta cbmDataMax
%    bcs +
%    dec cbmDataMax+1
% +  lda reachedEof
```

```
%    bne +
%    lda cbmDataMax
%    ora cbmDataMax+1
%    bne commieInMore

Shut down reading and exit.

% +  jsr kernelClrchn
%    lda cbmDataLen
%    ora cbmDataLen+1
%    clc
%    rts
%

Read a single byte from the CBM-Kernal input logical file number.  Translate
character into ASCII and expand CR into CR+LF if necessary.  Return EOF if
previous character returned was last from disk input channel.

% commieInByte = *  ;( ) : .A=char, .CS=err, .Z=eof, reachedEof
%    ;** check for already past eof
%    lda reachedEof
%    beq +
%    brk
%    ;** check for generated linefeed
% +  lda generateLf
%    beq +
%    lda #0
%    sta generateLf
%    lda #$0a
%    clc
%    rts
%    ;** check for eof
% +  lda prevSt
%    and #$40
%    beq +
%    lda #$ff
%    sta reachedEof
%    lda #0
%    clc
%    rts
%    ;** read actual character
% +  jsr kernelChrin
%    ldx st
%    stx prevSt
%    bcc +
%    sta errno
%    jsr kernelClrchn
%    rts
%    ;** translate if necessary
% +  bit transMode
%    bpl +
%    tax
%    lda transBufToAscii,x
%    beq commieInByte

Note here that the translated character is checked to see if it is a carriage
return, rather than checking the non-translated character, to see if a
linefeed must be generated next.  Thus, you could define that a Commodore
carriage return be translated into a linefeed (for Unix) and no additional
unwanted linefeed would be generated.

%    cmp #$0d
%    bne +
%    sta generateLf
%    ;** exit
% +  ldx #$ff
%    clc
%    rts
%
% firstFreeFatEntry = $5a
%

Search FAT for a free cluster, and return the cluster (FAT entry) number.  A
global variable "firstFreeFarEntry" is maintained which points to the first
FAT entry that could possibly be free, to avoid wasting time searching from
the very beginning of the FAT every time.  Clusters are allocated in
first-free order.

% allocateFatEntry = *  ;( ) : .AY=fatEntry, .CS=err
% -  lda firstFreeFatEntry
%    cmp lastFatEntry
```

```
%    lda firstFreeFatEntry+1
%    sbc lastFatEntry+1
%    bcc +
%    rts
% +  lda firstFreeFatEntry
%    ldy firstFreeFatEntry+1
%    jsr getFatEntry
%    sty 2
%    ora 2
%    bne +
%    lda firstFreeFatEntry
%    ldy firstFreeFatEntry+1
%    clc
%    rts
% +  inc firstFreeFatEntry
%    bne -
%    inc firstFreeFatEntry+1
%    jmp -
%
% msFileLength = $5c  ;(3 bytes)
%
```

Allocate a new cluster to a file, link it into the file cluster chain, and write the cluster buffer to disk in that cluster, adding "cbmDataLen" bytes to the file.

```
% msWriteCluster = *  ; (*) : .CS=err
%    ;** get a new cluster
%    jsr allocateFatEntry
%    bcc +
%    rts
%    ;** make previous fat entry point to new cluster
% +  sta fatValue
%    sty fatValue+1
%    lda cluster
%    ora cluster+1
%    beq +
%    lda cluster
%    ldy cluster+1
%    ldx fatValue
%    stx cluster
%    ldx fatValue+1
%    stx cluster+1
%    jsr setFatEntry
%    jmp msClusterNew
```

Handle case of no previous cluster - make directory entry point to new cluster.

```
% +  lda writeDirent
%    ldy writeDirent+1
%    sta 2
%    sty 3
%    ldy #26
%    lda fatValue
%    sta (2),y
%    sta cluster
%    iny
%    lda fatValue+1
%    sta (2),y
%    sta cluster+1
%
%    ;** make new fat entry point to null
%    msClusterNew = *
%    lda #$ff
%    ldy #$0f
%    sta fatValue
%    sty fatValue+1
%    lda cluster
%    ldy cluster+1
%    jsr setFatEntry
%    ;** write new cluster data
% +  lda cluster
%    ldy cluster+1
%    jsr writeCluster
%    bcc +
%    rts
%    ;** add cluster length to file length
% +  clc
%    lda msFileLength
%    adc cbmDataLen
```

```
%    sta msFileLength
%    lda msFileLength+1
%    adc cbmDataLen+1
%    sta msFileLength+1
%    bcc +
%    inc msFileLength+2
% +  clc
%    rts
%
```

Copy a CBM-Kernal file to an MS-DOS file, possibly with translation.

```
% msWrite = *  ;( msDevice, msType, writeDirent, .A=trans, .X=cbmLfn ) :.CS=err
%    ldy #$0e
%    sty $ff00
%    ;** initialize
```

Set input file translation and logical file number, init cluster, file length,
FAT allocation first free pointer (to cluster #2, the first data cluster).

```
%    jsr commieInInit
%    lda #0
%    sta cluster
%    sta cluster+1
%    sta firstFreeFatEntry+1
%    sta msFileLength
%    sta msFileLength+1
%    sta msFileLength+2
%    lda #2
%    sta firstFreeFatEntry
%
%    ;** copy cluster from cbm file
% -  lda #<clusterBuf
%    ldy #>clusterBuf
%    sta cbmDataPtr
%    sty cbmDataPtr+1
%    lda clusterBlockCount
%    asl
%    tay
%    lda #0
%    sta cbmDataMax
%    sty cbmDataMax+1
%    jsr commieIn
%    bcc +
%    rts
% +  beq +
%    jsr msWriteCluster
%    bcc -
%    rts
%
%    ;** wrap up after writing - set file length, dirty flag, exit.
% +  lda writeDirent
%    ldy writeDirent+1
%    sta 2
%    sty 3
%    ldx #0
%    ldy #28
% -  lda msFileLength,x
%    sta (2),y
%    iny
%    inx
%    cpx #3
%    bcc -
%    jsr dirtyDirent
%    clc
%    rts
%
```

This level deals exclusively with Commodore files.

```
% ;===== commodore file level =====
%
```

Copy from an input disk logical file number to an output lfn, in up to 1024
byte chunks. This routine makes use of the existing "commieIn" and
"commieOut" routines. No file translation is available; binary translation is
used for both commieIn and commieOut.

```
% cbmCopy = *  ;( .A=inLfn, .X=outLfn )
%    ldy #$0e
%    sty $ff00
```

```
%    stx lfn
%    tax
%    lda #0
%    jsr commieInInit
% -  lda #<clusterBuf
%    ldy #>clusterBuf
%    sta cbmDataPtr
%    sty cbmDataPtr+1
%    lda #<1024
%    ldy #>1024
%    sta cbmDataMax
%    sty cbmDataMax+1
%    jsr commieIn
%    bcs +
%    beq +
%    lda #<clusterBuf
%    ldy #>clusterBuf
%    sta cbmDataPtr
%    sty cbmDataPtr+1
%    jsr commieOut
%    bcs +
%    jmp -
% +  rts
%
```

Read a single directory entry from the given logical file number, which is assumed to be open for reading a directory ("$"). The data of the directory entry are returned in the interface variables.

```
% cbmDirent = *  ;( .A=lfn )
```

Initialize.

```
%    ldy #$0e
%    sty $ff00
%    tax
%    jsr kernelChkin
%    bcc +
%    cdirErr = *
%    lda #0
%    sta cdirFlen
%    sta cdirBlocks
%    sta cdirBlocks+1
%    rts
%    ;** get block count
% +  jsr cdirGetch
%    jsr cdirGetch
%    jsr cdirGetch
%    sta cdirBlocks
%    jsr cdirGetch
%    sta cdirBlocks+1
%    ;** look for filename
%    lda #0
%    sta cdirFlen
% -  jsr cdirGetch
%    cmp #34
%    beq +
%    cmp #"b"
%    bne -
%    jsr kernelClrchn
%    rts
%    ;** get filename
% +  ldy #0
% -  jsr cdirGetch
%    cmp #34
%    beq +
%    sta cdirName,y
%    iny
%    bne -
% +  sty cdirFlen
```

Look for and get file type.

```
% -  jsr cdirGetch
%    cmp #" "
%    beq -
%    sta cdirType
```

Scan for end of directory entry, return.

```
% -  jsr cdirGetch
```

```
%    cmp #0
%    bne -
%    jsr kernelClrchn
%    rts
%
```

Get a single character of the directory entry, watching for end of file (which would indicate error here).

```
% cdirGetch = *
%    jsr kernelChrin
%    bcs +
%    bit st
%    bvs +
%    rts
% +  pla
%    pla
%    jsr kernelClrchn
%    jmp cdirErr
%
% ;===== data =====
%
```

This is the translation table used to convert from ASCII to PETSCII. You can modify it to suit your needs if you wish. If you cannot reassemble this file, then you can sift through the binary file and locate the table and change it there. An entry of $00 means the corresponding ASCII character will not be translated. You'll notice that I have set up translations for the following ASCII control characters into PETSCII: Backspace, Tab, Linefeed (CR), and Formfeed. I also translate the non-PETSCII characters such as {, |, ~, and _ according to what they probably would have been if Commodore wasn't so concerned with the graphics characters.

```
% transBuf = *
%        ;0   1   2   3   4   5   6   7   8   9   a   b   c   d   e   f
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$14,$09,$0d,$00,$93,$00,$00,$00 ;0
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;1
% .byte $20,$21,$22,$23,$24,$25,$26,$27,$28,$29,$2a,$2b,$2c,$2d,$2e,$2f ;2
% .byte $30,$31,$32,$33,$34,$35,$36,$37,$38,$39,$3a,$3b,$3c,$3d,$3e,$3f ;3
% .byte $40,$c1,$c2,$c3,$c4,$c5,$c6,$c7,$c8,$c9,$ca,$cb,$cc,$cd,$ce,$cf ;4
% .byte $d0,$d1,$d2,$d3,$d4,$d5,$d6,$d7,$d8,$d9,$da,$5b,$5c,$5d,$5e,$5f ;5
% .byte $c0,$41,$42,$43,$44,$45,$46,$47,$48,$49,$4a,$4b,$4c,$4d,$4e,$4f ;6
% .byte $50,$51,$52,$53,$54,$55,$56,$57,$58,$59,$5a,$db,$dc,$dd,$de,$df ;7
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;8
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;9
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;a
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;b
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;c
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;d
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;e
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;f
%
```

This is the translation table used to convert from PETSCII to ASCII. You can modify it to suit your needs, similar to the ASCII to PETSCII table. An entry of $00 means the corresponding PETSCII character will not be translated. You'll notice that I have set up translations for the following PETSCII control characters into ASCII: Delete (into Backspace), Tab, Carriage Return (into CR+LF), and ClearScreen (into Fordfeed). Appropriate translations into the ASCII characters {, }, ^, _, ~, \, and | are also set up.

```
% transBufToAscii = *
%        ;0   1   2   3   4   5   6   7   8   9   a   b   c   d   e   f
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$09,$00,$00,$00,$0d,$00,$00 ;0
% .byte $00,$00,$00,$00,$08,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;1
% .byte $20,$21,$22,$23,$24,$25,$26,$27,$28,$29,$2a,$2b,$2c,$2d,$2e,$2f ;2
% .byte $30,$31,$32,$33,$34,$35,$36,$37,$38,$39,$3a,$3b,$3c,$3d,$3e,$3f ;3
% .byte $40,$61,$62,$63,$64,$65,$66,$67,$68,$69,$6a,$6b,$6c,$6d,$6e,$6f ;4
% .byte $70,$71,$72,$73,$74,$75,$76,$77,$78,$79,$7a,$5b,$5c,$5d,$5e,$5f ;5
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;6
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;7
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;8
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;9
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;a
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;b
% .byte $60,$41,$42,$43,$44,$45,$46,$47,$48,$49,$4a,$4b,$4c,$4d,$4e,$4f ;c
% .byte $50,$51,$52,$53,$54,$55,$56,$57,$58,$59,$5a,$7b,$7c,$7d,$7e,$7f ;d
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;e
% .byte $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$7e ;f
%
% ;====bss storage (size=11,264 bytes)====
```

%

This is where the track cache, etc. are stored. This section requires 11K of storage space but does not increase the length of the binary program file since these storage areas are DEFINED rather than allocated with ".buf" directives. The Unix terminology for this type of uninitialized data is "bss".

```
% bss = *
% trackbuf   = bss
% clusterBuf = trackbuf+4608
% fatbuf     = clusterBuf+1024
% dirbuf     = fatbuf+1536
% end        = dirbuf+4096
```

5. USER-INTERFACE PROGRAM

This section presents the listing of the user-interface BASIC program. You should be aware that you can easily change some of the defaults to your own preferences if you wish. In particular, you may wish to change the "dv" and "dt" variables in lines 25 and 26. This program is not listed in the "%" format that the assembler listing is since you can recover this listing from the uuencoded binary program file. The listing is here in its entirety.

```
10 print chr$(147);"little red reader 128 version 1.00"
11 print : print"by craig bruce 09-feb-93 for c=hacking" : print
12 :
20 cd=peek(186):if cd<8 then cd=8 : rem ** default cbm-dos drive **
25 dv=9:dt=0 :  rem ** ms-dos drive, type (0=1571,255=1581)
26 if dv=cd then dv=8:dt=0 : rem ** alternate ms-dos drive
27 :
30 print "initializing..." : print
40 bank0 : pk=dec("8000") : pv=pk+30
50 if peek(pv+0)=dec("cb") and peek(pv+1)=132 then 60
55 print"loading machine language routines..." : bload"lrr.bin",u(cd)
60 poke pv+3,dv : poke pv+4,dt : sys pk
70 dim t,r,b,i,a$,c,dt$,fl$,il$,x,x$
71 cm$="dmftc+-q "+chr$(13)+chr$(145)+chr$(17)+chr$(157)+chr$(29)+chr$(19)
72 cm$=cm$+chr$(147)+"/rnx"+chr$(92)
75 dl=-1 : cf=-1 : me=0 : ca=0 : ma=0
80 dim di$(1,300),cl(128),sz(128),dp(128),cn$(300)
90 if dt=255 then dt$="1581" :else dt$="1571"
100 fl$=chr$(19)+chr$(17)+chr$(17)+chr$(17)+chr$(17)
110 il$=fl$:fori=1to19:il$=il$+chr$(17):next
120 goto 500
130 :
131 rem ** load ms-dos directory **
140 print"loading ms-dos directory..." : print
150 sys pk : sys pk+3
160 dl=0
170 rreg bl,dc,bh,s : e=peek(pv+2)
180 if (s and 1) then gosub 380 : dl=-1 : return
190 print"scanning ms-dos directory..." : print
200 db=bl+256*bh
205 sys pk+21 : rreg bl,x,bh : ma=bl+bh*256+x*65536
210 if dc=0 then 360
220 for dp=db to db+32*(dc-1) step 32
230 if peek(dp)=0 or peek(dp)=229 then 350
240 if peek(dp+11) and 24 then 350
250 dl=dl+1
```

Line 260 sets the default selection, translation, and filetypes for MS-DOS files. Change to your liking.

```
260 d$=right$(" "+str$(dl),3)+"     asc  seq  " : rem ** default sel/tr/ft **
270 a$="" : fori=0to10 : a$=a$+chr$(peek(dp+i)) : next
280 a$=left$(a$,8)+"  "+right$(a$,3)
290 print dl; a$
300 d$=d$+a$+"  "
310 cl(dl)=peek(dp+26)+256*peek(dp+27)
320 sz=peek(dp+28)+256*peek(dp+29)+65536*peek(dp+30)
330 di$(0,dl)=d$+right$("    "+str$(sz),6)
335 dp(dl)=dp
340 sz(dl)=sz
350 next dp
360 return
370 :
371 rem ** report ms-dos disk error **
380 print chr$(18);"ms-dos disk error #";mid$(str$(e),2);
390 print " ($";mid$(hex$(e),3);"), press key.";chr$(146)
400 getkey a$ : return
```

```
410 :
411 rem ** screen heading **
420 print chr$(147);chr$(18);
421 if me=0 then print"ms-dos";:x=ma:else print"cbmdos";:x=ca
422 print chr$(146);"  ms=";mid$(str$(dv),2);":";dt$;
430 print"  cbm=";mid$(str$(cd),2);"  free=";mid$(str$(x),2)
440 print : return
450 :
451 rem ** screen footing **
460 print il$;"d=dir m=msdev f=cbmdev c=copy q=quit   "
470 print     "t=toggle r=remove x=cbmcpy /=menu +-=pg";
480 return
490 :
491 rem ** main routine **
500 t=1 : c=0
501 r=0
510 if me=0 then mf=dl:mc=2 : else mf=cf:mc=1
520 gosub 420
521 if me<>0 then 542
530 print "num  s  trn  typ  filename  ext  length"
540 print "---  -  ---  ---  --------  ---  ------"
541 goto 550
542 print "num  s  trn  filename         t  length"
543 print "---  -  ---  ---------------- -  ------"
550 gosub 460
560 b=t+17 : if b>mf then b=mf
570 print fl$;: if t>mf then 590
580 for i=t to b : print di$(me,i) : next
590 if mf<0 then print chr$(18);"<directory not loaded>";chr$(146)
591 if mf=0 then print chr$(18);"<no files>";chr$(146)
600 if mf<=0 then 660
610 print left$(il$,r+5);chr$(18);
620 on c+1 goto 630,640,650
630 print spc(4);mid$(di$(me,t+r),5,3) : goto 660
640 print spc(7);mid$(di$(me,t+r),8,5) : goto 660
650 print spc(12);mid$(di$(me,t+r),13,5) : goto 660
660 getkey a$
670 i=instr(cm$,a$)
680 if mf>0 then print left$(il$,r+5);di$(me,t+r)
690 if i=0 then 600
700 on i goto 760,1050,1110,950,1150,1000,1020,730,860,860,770,790,810,830,850
705 on i-15 goto 500,713,1400,713,1500,713
710 stop
711 :
712 rem ** various menu options **
713 me=-(me=0)
714 goto500
730 print chr$(147);"have an awesome day." : bank15
740 end
760 if me=1 then gosub 420 : gosub 2500 : goto 500
765 gosub 420 : gosub 140 : goto 500
770 r=r-1 : if r<0 then r=b-t
780 goto 600
790 r=r+1 : if t+r>b then r=0
800 goto 600
810 c=c-1 : if c<0 then c=mc
820 goto 600
830 c=c+1 : if c>mc then c=0
840 goto 600
850 r=0 : c=0 : goto 600
860 if mf<=0 then 600
870 x=t+r : on c+1 gosub 890,910,930
880 print left$(il$,r+5);di$(me,x) : goto 600
890 if mid$(di$(me,x),6,1)=" " then x$="*" :else x$=" "
900 mid$(di$(me,x),6,1)=x$ : return
910 if mid$(di$(me,x),9,1)="a" then x$="bin" :else x$="asc"
920 mid$(di$(me,x),9,3)=x$ : return
930 if mid$(di$(me,x),14,1)="s" then x$="prg" :else x$="seq"
940 mid$(di$(me,x),14,3)=x$ : return
950 if mf<=0 then 600
960 for x=1 to mf
970 on c+1 gosub 890,910,930
980 next x
990 goto 520
1000 r=0:if b=mf then t=1 : goto 510
1010 t=t+18 : goto 510
1020 if mf<=0 then 660
1025 r=0:if t=1 then t=mf-(mf-int(mf/18)*18)+1 : if t<=mf then 510
1030 t=t-18 : if t<1 then t=1
1040 goto 510
1050 print il$;chr$(27);"@";
```

```
1060 input"ms-dos device number (8-30)";dv
1061 if cd=dv thenprint"ms-dos and cbm-dos devices must be different!":goto1060
1070 x=71 : input"ms-dos device type  (71/81)";x
1080 if x=8 or x=81 or x=1581 then dt=255:dt$="1581" :else dt=0:dt$="1571"
1090 poke pv+3,dv : poke pv+4,dt : sys pk : dl=-1 : ma=0
1100 goto 500
1110 print il$;chr$(27);"@";
1120 input "cbm-dos device number (0-30)";cd
1130 if cd=dv thenprint"ms-dos and cbm-dos devices must be different!":goto1120
1140 cf=-1 : ca=0 : goto 500
1141 :
1142 rem ** copy files **
1150 if me=1 then 2000
1151 print chr$(147);"copy ms-dos -> cbm-dos":print:print
1160 if dl<=0 then fc=0 : goto 1190
1170 fc=0 : for f=1 to dl : if mid$(di$(0,f),6,1)="*" then gosub 1200
1180 next f
1190 print : print"files copied =";fc;" - press key"
1191 getkey a$ : goto 520
1200 fc=fc+1
1210 x$=mid$(di$(0,f),19,8)+"."+mid$(di$(0,f),29,3)
1220 cf$="":fori=1tolen(x$):if mid$(x$,i,1)<>" " then cf$=cf$+mid$(x$,i,1)
1230 next
1231 if right$(cf$,1)="." then cf$=left$(cf$,len(cf$)-1)
1232 cf$=cf$+","+mid$(di$(0,f),14,1)
1240 print str$(fc);". ";chr$(34);cf$;chr$(34);tab(20);sz(f)"bytes";
1245 print tab(35);mid$(di$(0,f),9,3)
1250 cl=cl(f) : lb=sz(f) - int(sz(f)/65536)*65536
1260 if cd>=8 then dopen#1,(cf$+",w"),u(cd) :else if cd<>0 then open 1,cd,7
1265 if cd<8 then 1288
1270 if ds<>63 then 1288
1275 x$="y" : print "cbm file exists; overwrite (y/n)";
1280 close 1 : input x$ : if x$="n" then fc=fc-1 : return
1285 scratch(cf$),u(cd)
1286 dopen#1,(cf$+",w"),u(cd)
1288 if cd<8 then 1320
1300 if ds<20 then 1320
1310 print chr$(18)+"cbm disk error: "+ds$ : fc=fc-1 : close1 : return
1320 poke pv+6,cl/256 : poke pv+5,cl-peek(pv+6)*256
1330 poke pv+8,lb/256 : poke pv+7,lb-peek(pv+8)*256
1340 tr=0 : if mid$(di$(0,f),9,1)="a" then tr=255
1346 x=1 : if cd=0 then x=0
1350 sys pk+6,tr,x
1355 rreg x,x,x,s : e=peek(pv+2)
1356 if (s and 1) then gosub 380 : fc=fc-1
1360 if cd<>0 and cd<8 then close1
1370 if cd>=8 then dclose#1 : if ds>=20 then 1310
1380 return
1398 :
1399 rem ** remove ms-dos file **
1400 print chr$(147);"remove (delete) selected ms-dos files:":print
1401 if me<>0 then print"ms-dos menu must be selected!" : goto2030
1402 a$="y":input"are you like sure about this (y/n)";a$
1403 print:if a$="n" then goto 520
1410 if dl<=0 then fc=0 : goto 1440
1420 fc=0 : f=1
1425 if mid$(di$(0,f),6,1)="*" then gosub 1470 : fc=fc+1 : f=f-1
1430 f=f+1 : if f<=dl then 1425
1434 print"flushing..."
1435 sys pk+12
1440 print : print"files removed =";fc;" - press key"
1445 sys pk+21 : rreg a,x,y : ma=a+y*256+x*65536
1450 getkey a$ : goto 500
1470 print"removing ";chr$(34);mid$(di$(0,f),19,13);chr$(34)
1490 poke pv+10,dp(f)/256 : poke pv+9,dp(f)-peek(pv+10)*256
1492 sys pk+15
1494 di$(0,f)=di$(0,dl):sz(f)=sz(dl):dp(f)=dp(dl):cl(f)=cl(dl)
1495 dl=dl-1
1496 return
1498 :
1499 rem ** copy cbm files **
1500 print chr$(147);"copy cbm-dos to cbm-dos:":print
1501 if cf<=0 then print"commodore directory not loaded" : goto 2030
1502 x=0 : input"device number to copy to";x : print
1503 if x<=0 or x>=64 then print"bad device number!" : goto 2030
1504 if x=cd then print"cannot copy to same device" : goto 2030
1505 for f=1 to cf : if mid$(di$(1,f),6,1)<>"*" then 1570
1506 print di$(1,f) : open1,cd,2,cn$(f)+",r"
1507 if x<8 then open 2,x,7 : goto1550
1508 cf$=cn$(f)+","+mid$(di$(1,f),31,1)+",w"
1509 open2,x,3,cf$
```

```
1510 if ds<>63 then 1530
1511 close2
1512 x$="y":input"file exists: overwrite (y/n)";x$ : if x$="n" then 1560
1520 scratch(cn$(f)),u(x)
1525 open2,x,3,cf$
1530 if ds>20 then print chr$(18);"cbm dos error: ";ds$ : goto1560
1550 sys pk+24,1,2
1560 close1 : close2
1570 next f
1580 print : print"finished - press a key" : getkey a$ : goto510
1998 :
1999 rem ** copy cbm-dos to ms-dos **
2000 print chr$(147);"copy cbm-dos to ms-dos:" : print : print
2010 if dl>=0 then 2035
2020 print"ms-dos directory must be loaded first"
2030 print : print"press any key" : getkey a$ : goto 510
2035 fc=0
2036 for f=1 to cf : if mid$(di$(1,f),6,1)<>"*" then 2045
2040 fc=fc+1 : c$=cn$(f)
2041 printmid$(str$(fc),2);" ";mid$(di$(1,f),14,16);mid$(di$(1,f),34);":";
2042 gosub2050 : print left$(m$,8);".";right$(m$,3)
2043 tr=0 : if mid$(di$(1,f),9,1)="a" then tr=255
2044 gosub2100
2045 next
2046 print"flushing..." : sys pk+12
2047 sys pk+21 : rreg a,x,y : ma=a+y*256+x*65536
2048 print: print"files copied =";fc : goto2030
2049 :
2050 x=instr(c$,".") : if x=0 then m$=c$+"           " : goto2090
2055 x=len(c$)+1 : do : x=x-1 : loop until mid$(c$,x,1)="."
2060 m$=left$(left$(c$,x-1)+"        ",8)
2070 x$=mid$(c$,x+1)+"   "
2080 m$=m$+x$
2090 m$=left$(m$,11)
2091 fori=1to11:x$=chr$(asc(mid$(m$,i,1))and127):if x$="."orx$=" " then x$="_"
2092 mid$(m$,i,1)=x$ : next i
2093 i=8 : do while i>1 and mid$(m$,i,1)="_" : mid$(m$,i,1)=" " : i=i-1 : loop
2094 i=11 : do while i>8 and mid$(m$,i,1)="_" : mid$(m$,i,1)=" " : i=i-1 : loop
2098 return
2099 :
2100 fori=0to0
2105 for dp=db to db+32*(dc-1) step 32
2110 if peek(dp)=0 or peek(dp)=229 then 2140
2120 next dp
2130 print"no free ms-dos directory entires" : return
2140 next i
2160 fori=1tolen(m$):pokedp+i-1,asc(mid$(m$,i,1)) and 127:next
2170 fori=11to31:poke dp+i,0:next
2180 pokedp+26,255:pokedp+27,15
2190 poke pv+10,dp/256:poke pv+9,dp-peek(pv+10)*256
2200 open1,cd,2,c$
2300 sys pk+9,tr,1 : rreg x,x,x,s
2301 close1
2305 if s and 1 then e=peek(pv+2) : gosub380 : return
```

Line 2310 sets the default MS-DOS selection, translation, and filetype after copying to MS-DOS disk, based on the CBM-DOS filetype. Change to your liking.

```
2310 x$="     asc  seq  ":if tr=0 then x$="     bin  prg  "
2320 dl=dl+1 : d$=right$(" "+str$(dl),3)+x$
2330 d$=d$+left$(m$,8)+"  "+right$(m$,3)
2340 cl(dl)=peek(dp+26)+256*peek(dp+27)
2350 sz=peek(dp+28)+256*peek(dp+29)+65536*peek(dp+30)
2360 di$(0,dl)=d$+right$("        "+str$(sz),8)
2370 dp(dl)=dp
2380 sz(dl)=sz
2395 return
2498 :
2499 rem ** load commodore dos directory **
2500 print"loading commodore dos directory..." : print
2501 if cd<8 then print"cbmdos device must be >= 8!" : goto2030
2505 open1,cd,0,"$0":get#1,a$,a$ : cf=-1 : q$=chr$(34)
2506 do
2507 sys pk+27,1 : b=peek(pv+11)+256*peek(pv+12) : t$=chr$(peek(pv+13))
2510 x=peek(pv+14)
2520 if x=0 then exit
2530 x$="" : for i=pv+15 to pv+15+x-1 : x$=x$+chr$(peek(i)) : next
2575 cf=cf+1
2590 if cf=0 then print"disk="q$x$q$ : print : goto2650
2600 cn$(cf)=x$
2610 a$=left$(x$+"                ",17)+t$+right$("       "+str$(b*254),8)
```

Lines 2620 and 2625 set the default CBM-DOS selection and translation modes based on the filetype. Change to your liking.

```
2620 di$(1,cf)=right$("  "+str$(cf),3)+"     asc  "+a$
2625 if t$<>"s" then mid$(di$(1,cf),9,3)="bin"
2630 print di$(1,cf)
2650 loop
2670 ca=b*256 : close1 : return
```

6. UUENCODED FILES

Here are the binary executables in uuencoded form. The CRC32s of the two files are as follows:

```
crc32 = 3896271974 for "lrr-128"
crc32 = 2918283051 for "lrr.bin"
```

The "lrr.128" file is the main BASIC program and the "lrr.bin" file contains the machine lanugage disk-accessing routines.

```
begin 640 lrr-128
M`1PS'`H`F2#'*#$T-RD[(DQ)5%1,12!2140@4D5!1$52(#$R."!615)324].
M(#$N,#`B`&D<"P"9(#H@F2)"62!#4D%)1R!"4E5#12`P.2U&14(M.3,@1D]2
M($,]2$%#2TE.1R(@.B"9`&\<#``Z`*L<%`!#1++"*#$X-BDZBR!#1+,X(*<@
M0T2R."`Z((\@*BH@1$5&055,5"!#0DTM1$]3($12259%("HJ`.8<&0!$5K(Y
M.D14LC`@.B`@CR`J*B!-4RU$3U,@1%))5D4L(%194$4@*#`],34W,2PR-34]
M,34X,2D`'!T:`(L@1%:R0T0@IR!$5K(X.D14LC`@.B"/("HJ($%,5$523D%4
M12!-4RU$3U,@1%))5D4`(AT;`#H`/AT>`)D@(DE.251)04Q)6DE.1RXN+B(@
M.B"9`&`=*`#^`C`@.B!02[+1*"(X,#`P(BD@.B!05K)02ZHS,`")'3(`BR#"
M*%!6JC`ILM$H(D-"(BD@KR#"*%!6JC$ILC$S,B"G(#8P`,D=-P"9(DQ/041)
M3D<@34%#2$E.12!,04Y'54%'12!23U5424Y%4RXN+B(@.B#^$2),4E(N0DE.
M(BQ5*$-$*0#J'3P`ER!05JHS+$16(#H@ER!05JHT+$14(#H@GB!02P`.'D8`
MAB!4+%(L0BQ)+$$D+$,L1%0D+$9,)"Q)3"0L6"Q8)`!('D<`0TTDLB)$3494
M0RLM42`BJL<H,3,IJL<H,30U*:K'*#$W*:K'*#$U-RFJQR@R.2FJQR@Q.2D`
M:!Y(`$--)+)#322JQR@Q-#<IJB(O4DY8(JK'*#DR*0"/'DL`1$RRJS$@.B!#
M1K*K,2`Z($U%LC`@.B!#0;(P(#H@34&R,`#!'E``AB!$220H,2PS,#`I+$-,
M*#$R."DL4UHH,3(X*2Q$4"@Q,C@I+$-.)"@S,#`I`.D>6@"+($14LC(U-2"G
M($14)+(B,34X,2(@.M4@1%0DLB(Q-3<Q(@`/'V0`1DPDLL<H,3DIJL<H,3<I
MJL<H,3<IJL<H,3<IJL<H,3<I`#,?;@!)3"2R1DPD.H%)LC&D,3DZ24PDLDE,
M)*K'*#$W*3J"`#T?>`")(#4P,`!#'X(`.@!E'X,`CR`J*B!,3T%$($U3+41/
M4R!$25)%0U1/4ED@*BH`C!^,`)DB3$]!1$E.1R!-4RU$3U,@1$E214-43U)9
M+BXN(B`Z()D`GA^6`)X@4$L@.B">(%!+JC,`IQ^@`$1,LC``Q1^J`/X)($),
M+$1#+$)(+%,@.B!%LL(H4%:J,BD`YQ^T`(L@*%,@KR`Q*2"G((T@,S@P(#H@
M1$RRJS$@.B".``\@O@"9(E-#04Y.24Y'($U3+41/4R!$25)%0U1/4EDN+BXB
M(#H@F0`@(,@`1$*R0DRJ,C4VK$)(`%`@S0">(%!+JC(Q(#H@_@D@0DPL6"Q"
M2"`Z($U!LD),JD)(K#(U-JI8K#8U-3,V`&$@T@"+($1#LC`@IR`S-C``@2#<
M`($@1%"R1$(@I"!$0JHS,JPH1$.K,2D@J2`S,@"A(.8`BR#"*$10*;(P(+`@
MPBA$4"FR,C(Y(*<@,S4P`+L@\`"+(,(H1%"J,3$I(*\@,C0@IR`S-3``QR#Z
M`$1,LD1,JC$`"R$$`40DLLDH(B`BJL0H1$PI+#,IJB(@("`@($%30R`@4T51
M("`B(#H@CR`J*B!$149!54Q4(%-%3"]44B]&5"`J*@`V(0X!022R(B(@.B"!
M2;(PI#$P(#H@022R022JQRC"*$10JDDI*2`Z(((`4B$8`4$DLL@H020L."FJ
M(B`@(JK)*$$D+#,I`%\A(@&9($1,.R!!)`!Q(2P!1"2R1"2J022J(B`@(@"2
M(38!0TPH1$PILL(H1%"J,C8IJC(U-JS"*$10JC(W*0"^(4`!4UJRPBA$4*HR
M."FJ,C4VK,(H1%"J,CDIJC8U-3,VK,(H1%"J,S`I`.$A2@%$220H,"Q$3"FR
M1"2JR2@B("`@("*JQ"A36BDL-BD`[R%/`410*$1,*;)$4`#](50!4UHH1$PI
MLE-:``8B7@&"($10``PB:`&.`!(B<@$Z`#<B<P&/("HJ(%)%4$]25"!-4RU$
M3U,@1$E32R!%4E)/4B`J*@!D(GP!F2#'*#$X*3LB35,M1$]3($1)4TL@15)2
M3U(@(R([RBC$*$4I+#(I.P"1(H8!F2`B("@D(CO**-(H12DL,RD[(BDL(%!2
M15-3($M%62XB.\<H,30V*0"?(I`!H?D@020@.B".`*4BF@$Z`,`BFP&/("HJ
M(%-#4D5%3B!(14%$24Y'("HJ`-0BI`&9(,<H,30W*3O'*#$X*3L``R.E`8L@
M346R,""G()DB35,M1$]3(CLZ6+)-03K5()DB0T)-1$]3(CLZ6+)#00`L(Z8!
MF2#'*#$T-BD[(B`@35,](CO**,0H1%8I+#(I.R(Z(CM$5"0[`%DCK@&9(B`@
M0T)-/2([RBC$*$-$*2PR*3LB("!&4D5%/2([RBC$*%@I+#(I`&,CN`&9(#H@
MC@!I(\(!.@"$(\,!CR`J*B!30U)%14X@1D]/5$E.1R`J*@"X(\P!F2!)3"0[
M(D0]1$E2($T]35-$158@1CU#0DU$158@0SU#3U!9(%$]455)5"`@("(`[2/6
M`9D@("`@(")4/51/1T=,12!2/5)%34]612!8/4-"34-062`O/4U%3E4@*RT]
M4$<B.P#S(^`!C@#Y(^H!.@`2).L!CR`J*B!-04E.(%)/551)3D4@*BH`("3T
M`52R,2`Z($.R,``H)/4!4K(P`$\D_@&+($U%LC`@IR!-1K)$3#I-0[(R(#H@
MU2!-1K)#1CI-0[(Q`%DD"`*-(#0R,`!K)`D"BR!-1;.Q,""G(#4T,@";)!("
MF2`B3E5-("!3("!44DX@(%194"`@1DE,14Y!344@($585"`@3$5.1U1((@#+
M)!P"F2`B+2TM("`M("`M+2T@("TM+2`@+2TM+2TM+2T@("TM+2`@+2TM+2TM
M(@#5)!T"B2`U-3``!24>`ID@(DY532`@4R`@5%).("!&24Q%3D%-12`@("`@
M("`@(%0@($Q%3D=42"(`-24?`ID@(BTM+2`@+2`@+2TM("`M+2TM+2TM+2TM
M+2TM+2TM("T@("TM+2TM+2(`/R4F`HT@-#8P`%HE,`)"LE2J,3<@.B"+($*Q
M348@IR!"LDU&`',E.@*9($9,)#LZ((L@5+%-1B"G(#4Y,`"3)40"@2!)LE0@
MI"!"(#H@F2!$220H344L22D@.B""`,@E3@*+($U&LS`@IR"9(,<H,3@I.R(\
M1$E214-43U)9($Y/5"!,3T%$140^(CO'*#$T-BD`\25/`HL@34:R,""G()D@
MQR@Q."D[(CQ.3R!&24Q%4SXB.\<H,30V*0`#)E@"BR!-1K.R,""G(#8V,``;
M)F("F2#(*$E,)"Q2JC4I.\<H,3@I.P`S)FP"D2!#JC$@B2`V,S`L-C0P+#8U
M,`!8)G8"F2"F-"D[RBA$220H344L5*I2*2PU+#,I(#H@B2`V-C``?2:``ID@
```

MIC<I.\HH1$DD*$U%+%2J4BDL."PU*2`Z((D@-C8P`*0FB@*9(*8Q,BD[RBA$
M220H344L5*I2*2PQ,RPU*2`Z((D@-C8P`*XFE`*A^2!!)`"^)IX"2;+4*$--
M)"Q!)"D`Y":H`HL@34:Q,""G()D@R"A)3"0L4JHU*3M$220H344L5*I2*0#T
M)K("BR!)LC`@IR`V,#``/R>\`I$@22")(#<V,"PQ,#4P+#$Q,3`L.34P+#$Q
M-3`L,3`P,"PQ,#(P+#<S,"PX-C`L.#8P+#<W,"PW.3`L.#$P+#@S,"PX-3``
M9B?!`I$@2:LQ-2")(#4P,"PW,3,L,30P,"PW,3,L,34P,"PW,3,`;"?&`I``
M<B?'`CH`DR?(`H\@*BH@5D%224]54R!-14Y5($]05$E/3E,@*BH`HB?)`DU%
MLJLH346R,"D`JR?*`HDU,#``UB?:`ID@QR@Q-#<I.R)(059%($%.($%715-/
M344@1$%9+B(@.B#^`C$U`-PGY`*````H^`*+($U%LC$@IR"-(#0R,"`Z((T@
M,C4P,"`Z((D@-3`P`!HH_0*-(#0R,"`Z((T@,30P(#H@B2`U,#``-"@"`U*R
M4JLQ(#H@BR!2LS`@IR!2LD*K5``^*`P#B2`V,#``6"@6`U*R4JHQ(#H@BR!4
MJE*Q0B"G(%*R,`!B*"`#B2`V,#``>R@J`T.R0ZLQ(#H@BR!#LS`@IR!#LDU#
M`(4H-`.)(#8P,`">*#X#0[)#JC$@.B"+($.Q34,@IR!#LC``J"A(`XD@-C`P
M`+XH4@-2LC`@.B!#LC`@.B")(#8P,`#0*%P#BR!-1K.R,""G(#8P,`#P*&8#
M6+)4JE(@.B"1($.J,2"-(#@Y,"PY,3`L.3,P`!,I<`.9(,@H24PD+%*J-2D[
M1$DD*$U%+%@I(#H@B2`V,#``02EZ`XL@RBA$220H344L6"DL-BPQ*;(B("(@
MIR!8)+(B*B(@.M4@6"2R(B`B`%TIA`/**$1))"A-12Q8*2PV+#$ILE@D(#H@
MC@"/*8X#BR#**$1))"A-12Q8*2PY+#$ILB)!(B"G(%@DLB)"24XB(#K5(%@D
MLB)!4T,B`*LIF`/**$1))"A-12Q8*2PY+#,ILE@D(#H@C@#>*:(#BR#**$1)
M)"A-12Q8*2PQ-"PQ*;(B4R(@IR!8)+(B4%)'(B`ZU2!8)+(B4T51(@#[*:P#
MRBA$220H344L6"DL,30L,RFR6"0@.B".``TJM@.+($U&L[(P(*<@-C`P`!PJ
MP`.!(%BR,2"D($U&`#0JR@.1($.J,2"-(#@Y,"PY,3`L.3,P`#PJU`."(%@`
M1BK>`XD@-3(P`&,JZ`-2LC`ZBR!"LDU&(*<@5+(Q(#H@B2`U,3``=BKR`U2R
M5*HQ."`Z((D@-3$P`(@J_`.+($U&L[(P(*<@-C8P`,`J`012LC`ZBR!4LC$@
MIR!4LDU&JRA-1JNU*$U&K3$X*:PQ."FJ,2`Z((L@5+.R348@IR`U,3``V2H&
M!%2R5*LQ."`Z((L@5+,Q(*<@5+(Q`.,J$`2)(#4Q,`#X*AH$F2!)3"0[QR@R
M-RD[(D`B.P`>*R0$A2)-4RU$3U,@1$5624-%($Y534)%4B`H."TS,"DB.T16
M`&(K)02+($-$LD16(*>9(DU3+41/4R!!3D0@0T)-+41/4R!$159)0T53($U5
M4U0@0D4@1$E&1D5214Y4(2(ZB3$P-C``CBLN!%BR-S$@.B"%(DU3+41/4R!$
M159)0T4@5%E012`@*#<Q+S@Q*2([6`#/*S@$BR!8LC@@L"!8LC@Q(+`@6+(Q
M-3@Q(*<@1%2R,C4U.D14)+(B,34X,2(@.M4@1%2R,#I$5"2R(C$U-S$B`/\K
M0@@27(%!6JC,L1%8@.B"7(%!6JC0L1%0@.B">(%!+(#H@1$RRJS$@.B!-0;(P
M``DL3`2)(#4P,``>+%8$F2!)3"0[QR@R-RD[(D`B.P!&+&`$A2`B0T)-+41/
M4R!$159)0T4@3E5-0D52("@P+3,P*2([0T0`BBQJ!(L@0T2R1%8@IYDB35,M
M1$]3($%.1"!#0DTM1$]3($1%5DE#15,@35535"!"12!$249&15)%3E0A(CJ)
M,3$R,`"C+'0$0T:RJS$@.B!#0;(P(#H@B2`U,#``J2QU!#H`P"QV!(\@*BH@
M0T]062!&24Q%4R`J*@#2+'X$BR!-1;(Q(*<@,C`P,`#\+'\$F2#'*#$T-RD[
M(D-/4%D@35,M1$]3("T^($-"32U$3U,B.IDZF0`8+8@$BR!$3+.R,""G($9#
MLC`@.B")(#$Q.3``3RV2!$9#LC`@.B"!($:R,2"D($1,(#H@BR#**$1))"@P
M+$8I+#8L,2FR(BHB(*<@C2`Q,C`P`%<MG`2"($8`@RVF!)D@.B"9(D9)3$53
M($-/4$E%1"`](CM&0SLB("T@4%)%4U,@2T59(@"5+:<$H?D@020@.B")(#4R
M,`"A+;`$1D.R1D.J,0#.+;H$6"2RRBA$220H,"Q&*2PQ.2PX*:HB+B*JRBA$
M220H,"Q&*2PR.2PS*0`)+L0$0T8DLB(B.H%)LC&DPRA8)"DZBR#**%@D+$DL
M,2FSL2(@(B"G($-&)+)#1B2JRBA8)"Q)+#$I``\NS@2"`#@NSP2+(,DH0T8D
M+#$ILB(N(B"G($-&)++(*$-&)"S#*$-&)"FK,2D`62[0!$-&)+)#1B2J(BPB
MJLHH1$DD*#`L1BDL,30L,2D`C2[8!)D@Q"A&0RD[(BX@(CO'*#,T*3M#1B0[
MQR@S-"D[HS(P*3M36BA&*2)"651%4R([`*@NW029(*,S-2D[RBA$220H,"Q&
M*2PY+#,I`-<NX@1#3+)#3"A&*2`Z($Q"LE-:*$8I(*L@M2A36BA&*:TV-34S
M-BFL-C4U,S8`$2_L!(L@0T2QLC@@IR#^#2,Q+"A#1B2J(BQ7(BDL52A#1"D@
M.M4@BR!#1+.Q,""G()\@,2Q#1"PW`",O\02+($-$LS@@IR`Q,C@X`#<O]@2+
M($13L[$V,R"G(#$R.#@`:B_[!%@DLB)9(B`Z()D@(D-"32!&24Q%($5825-4
M4SL@3U9%4E=2251%("A9+TXI(CL`DB\`!:`@,2`Z((4@6"0@.B"+(%@DLB).
M(B"G($9#LD9#JS$@.B".`*,O!07R*$-&)"DL52A#1"D`O2\&!?X-(S$L*$-&
M)*HB+%<B*2Q5*$-$*0#/+P@%BR!#1+,X(*<@,3,R,`#B+Q0%BR!$4[,R,""G
M(#$S,C``&#`>!9D@QR@Q."FJ(D-"32!$25-+($524D]2.B`BJD13)"`Z(-$9#
MLD9#JS$@.B"@,2`Z((X`0C`H!9<@4%:J-BQ#3*TR-38@.B"7(%!6JC4L0TRK
MPBA05JHV*:PR-38`;#`R!9<@4%:J."Q,0JTR-38@.B"7(%!6JC<L3$*KPBA0
M5JHX*:PR-38`EC`\!512LC`@.B"+(,HH1$DD*#`L1BDL.2PQ*;(B02(@IR!4
M4K(R-34`K3!"!5BR,2`Z((L@0T2R,""G(%BR,`"],$8%GB!02ZHV+%12+%@`
MV#!+!?X)(%@L6"Q8+%,@.B!%LL(H4%:J,BD`^#!,!8L@*%,@KR`Q*2"G((T@
M,S@P(#H@1D.R1D.K,0`0,5`%BR!#1+.Q,""O($-$LS@@IR"@,0`U,5H%BR!#
M1+&R.""G(/X/($@.B"+($3L;(R,""G(#$S,3``.S%D!8X`03%V!3H`8#%W
M!8\@*BH@4D5-3U9%($U3+41/4R!&24Q%("HJ`)@Q>`69(,<H,30W*3LB4D5-
M3U9%("A$14Q%5$4I(%-%3$5#5$5$($U3+41/4R!&24Q%4SHB.ID`SS%Y!8L@
M346SL3`@IR"9(DU3+41/4R!-14Y5($U54U0@0D4@4T5,14-4140A(B`Z((DR
M,#,P``,R>@5!)+(B62(ZA2)!4D4@64]5($Q)2T4@4U5212!!0D]55"!42$E3
M("A9+TXI(CM!)``:,GL%F3J+($$DLB).(B"G((D@-3(P`#8R@@6+($1,L[(P
M(*<@1D.R,"`Z((D@,30T,`!%,HP%1D.R,"`Z($:R,0!Z,I$%BR#**$1))"@P
M+$8I+#8L,2FR(BHB(*<@C2`Q-#<P(#H@1D.R1D.J,2`Z($:R1JLQ`)4RE@5&
MLD:J,2`Z((L@1K.R1$P@IR`Q-#(U`*@RF@69(D9,55-(24Y'+BXN(@"T,IL%
MGB!02ZHQ,@#A,J`%F2`Z()DB1DE,15,@4D5-3U9%1"`](CM&0SLB("T@4%)%
M4U,@2T59(@`-,Z4%GB!02ZHR,2`Z(/X)($$L6"Q9(#H@34&R0:I9K#(U-JI8
MK#8U-3,V`!\SJ@6A^2!!)"`Z((D@-3`P`$XSO@69(E)%34]624Y'("([QR@S
M-"D[RBA$220H,"Q&*2PQ.2PQ,RD[QR@S-"D`@#/2!9<@4%:J,3`L1%`H1BFM
M,C4V(#H@ER!05JHY+$10*$8IJ\(H4%:J,3`IK#(U-@",,]0%GB!02ZHQ-0#*
M,]8%1$DD*#`L1BFR1$DD*#`L1$PI.E-:*$8ILE-:*$1,*3I$4"A&*;)$4"A$
M3"DZ0TPH1BFR0TPH1$PI`-8SUP5$3+)$3*LQ`-PSV`6.`.(SV@4Z`/TSVP6/
M("HJ($-/4%D@0T)-($9)3$53("HJ`"<TW`69(,<H,30W*3LB0T]062!#0DTM
M1$]3(%1/($-"32U$3U,Z(CJ9`&`TW06+($-&L[(P(*<@F2)#3TU-3T1/4D4@
M1$E214-43U)9($Y/5"!,3T%$140B(#H@B2`R,#,P`(PTW@58LC`@.B"%(D1%
M5DE#12!.54U"15(@5$\@0T]062!43R([6"`Z()D`P#3?!8L@6+.R,""P(%BQ

```
MLC8T(*<@F2)"040@1$5624-%($Y534)%4B$B(#H@B2`R,#,P`/0TX`6+(%BR
M0T0@IR"9(D-!3DY/5"!#3U!9(%1/(%-!344@1$5624-%(B`Z((D@,C`S,``C
M->$%@2!&LC$@I"!#1B`Z((L@RBA$220H,2Q&*2PV+#$IL[$B*B(@IR`Q-3<P
M`$@UX@69($1))"@Q+$8I(#H@GS$L0T0L,BQ#3B0H1BFJ(BQ2(@!D->,%BR!8
MLS@@IR"?(#(L6"PW(#H@B3$U-3``C37D!4-&)+)#3B0H1BFJ(BPBJLHH1$DD
M*#$L1BDL,S$L,2FJ(BQ7(@"<->4%GS(L6"PS+$-&)`"P->8%BR!$4[.Q-C,@
MIR`Q-3,P`+<UYP6@,@#W->@%6"2R(EDB.H4B1DE,12!%6$E35%,Z($]615)7
M4DE412`H62].*2([6"0@.B"+(%@DLB).(B"G(#$U-C``"C;P!?(H0TXD*$8I
M*2Q5*%@I`!DV]06?,BQ8+#,L0T8D`$TV^@6+($13L3(P(*<@F2#'*#$X*3LB
M0T)-($1/4R!%4E)/4CH@(CM$4R0@.B"),34V,`!=-@X&GB!02ZHR-"PQ+#(`
M:388!J`Q(#H@H#(`<38B!H(@1@"B-BP&F2`Z()DB1DE.25-(140@+2!04D53
M4R!!($M%62(@.B"A^2!!)"`Z((DU,3``J#;.!SH`RS;/!X\@*BH@0T]062!#
M0DTM1$]3(%1/($U3+41/4R`J*@#Z-M`'F2#'*#$T-RD[(D-/4%D@0T)-+41/
M4R!43R!-4RU$3U,Z(B`Z()D@.B"9``TWV@>+($1,L;(P(*<@,C`S-0`Z-^0'
MF2)-4RU$3U,@1$E214-43U)9($U54U0@0D4@3$]!1$5$($9)4E-4(@!C-^X'
MF2`Z()DB4%)%4U,@04Y9($M%62(@.B"A^2!!)"`Z((D@-3$P`&PW\P=&0[(P
M`)LW]`>!($:R,2"D($-&(#H@BR#**$1))"@Q+$8I+#8L,2FSL2(J(B"G(#(P
M-#4`LS?X!T9#LD9#JC$@.B!#)+)#3B0H1BD`[3?Y!YG**,0H1D,I+#(I.R(@
M(CO**$1))"@Q+$8I+#$T+#$V*3O**$1))"@Q+$8I+#,T*3LB.B([``\X^@>-
M,C`U,"`Z()D@R"A-)"PX*3LB+B([R2A-)"PS*0`Y./L'5%*R,"`Z((L@RBA$
M220H,2Q&*2PY+#$ILB)!(B"G(%12LC(U-0!#./P'C3(Q,#``23C]!X(`9CC^
M!YDB1DQ54TA)3D<N+BXB(#H@GB!02ZHQ,@"2./\'GB!02ZHR,2`Z(/X)($$L
M6"Q9(#H@34&R0:I9K#(U-JI8K#8U-3,V`+8X``B9.B"9(D9)3$53($-/4$E%
M1"`](CM&0R`Z((DR,#,P`+PX`0@Z`/(X`@A8LM0H0R0L(BXB*2`Z((L@6+(P
M(*<@322R0R2J(B`@("`@("`@("`@(B`Z((DR,#DP`"`Y!PA8LL,H0R0IJC$@
M.B#K(#H@6+)8JS$@.B#L(/P@RBA#)"Q8+#$ILB(N(@!!.0P(322RR"C(*$,D
M+%BK,2FJ(B`@("`@("`@(BPX*0!8.18(6"2RRBA#)"Q8JC$IJB(@("`B`&4Y
M(`A-)+)-)*I8)`!U.2H(322RR"A-)"PQ,2D`L3DK"(%)LC&D,3$Z6"2RQRC&
M*,HH320L22PQ*2FO,3(W*3J+(%@DLB(N(K!8)+(B("(@IR!8)+(B7R(`R#DL
M",HH320L22PQ*;)8)"`Z(((@20`&.BT(2;(X(#H@ZR#](SFQ,2"O(,HH320L
M22PQ*;(B7R(@.B#**$TD+$DL,2FR(B`B(#H@2;))JS$@.B#L`$4Z+@A)LC$Q
M(#H@ZR#](SFQ.""O(,HH320L22PQ*;(B7R(@.B#**$TD+$DL,2FR(B`B(#H@
M2;))JS$@.B#L`$LZ,@B.`%$Z,P@Z`%PZ-`B!2;(PI#``?#HY"($@1%"R1$(@
MI"!$0JHS,JPH1$.K,2D@J2`S,@"=.CX(BR#"*$10*;(P(+`@PBA$4"FR,C(Y
M(*<@,C$T,`"F.D@(@B!$4`#2.E((F2).3R!&4D5%($U3+41/4R!$25)%0U1/
M4ED@14Y425)%4R(@.B".`-HZ7`B"($D`!CMP"(%)LC&DPRA-)"DZET10JDFK
M,2S&*,HH320L22PQ*2D@KR`Q,C<Z@@`>.WH(@4FR,3&D,S$ZER!$4*I)+#`Z
M@@`W.X0(ET10JC(V+#(U-3J71%"J,C<L,34`83N.")<@4%:J,3`L1%"M,C4V
M.I<@4%:J.2Q$4*O"*%!6JC$P*:PR-38`<#N8")\Q+$-$+#(L0R0`C3O\")X@
M4$NJ.2Q44BPQ(#H@_@D@6"Q8+%@L4P"4._T(H#$`MSL!"8L@4R"O(#$@IR!%
MLL(H4%:J,BD@.B"-,S@P(#H@C@#N.P8)6"2R(B`@("`@05-#("!315$@("(Z
MBR!44K(P(*<@6"2R(B`@("`@0DE.("!04D<@("(`$3P0"41,LD1,JC$@.B!$
M)++)*"(@(JK$*$1,*2PS*:I8)``P/!H)1"2R1"2JR"A-)"PX*:HB("`BJLDH
M320L,RD`43PD"4-,*$1,*;+"*$10JC(V*:HR-3:LPBA$4*HR-RD`?3PN"5-:
MLL(H1%"J,C@IJC(U-JS"*$10JC(Y*:HV-34S-JS"*$10JC,P*0"D/#@)1$DD
M*#`L1$PILD0DJLDH(B`@("`@("`@(JK$*%-:*2PX*0"R/$()1%`H1$PILD10
M`,`\3`E36BA$3"FR4UH`QCQ;"8X`S#S""3H`]3S#"8\@*BH@3$]!1"!#3TU-
M3T1/4D4@1$]3($1)4D5#5$]262`J*@`C/<0)F2),3T%$24Y'($-/34U/1$]2
M12!$3U,@1$E214-43U)9+BXN(B`Z()D`5SW%"8L@0T2S.""G()DB0T)-1$]3
M($1%5DE#12!-55-4($)%(#X](#@A(B`Z((DR,#,P`(4]R0F?,2Q#1"PP+"(D
M,"(ZH2,Q+$$D+$$D(#H@0T:RJS$@.B!1)++'*#,T*0"+/<H)ZP#$/<L)GB!0
M2ZHR-RPQ(#H@0K+"*%!6JC$Q*:HR-3:LPBA05JHQ,BD@.B!4)++'*,(H4%:J
M,3,I*0#3/<X)6++"*%!6JC$T*0#A/=@)BR!8LC`@IR#M`!<^X@E8)+(B(B`Z
M(($@2;)05JHQ-2"D(%!6JC$UJEBK,2`Z(%@DLE@DJL<HPBA)*2D@.B""`",^
M#PI#1K)#1JHQ`$L^'@J+($-&LC`@IR"9(D1)4TL](E$D6"11)"`Z()D@.B")
M,C8U,`!:/B@*0TXD*$-&*;)8)`"9/C(*022RR"A8)*HB("`@("`@("`@("`@
M("`@("`B+#$W*:I4)*K)*"(@("`@("`@(JK$*$*L,C4T*2PX*0#'/CP*1$DD
M*#$L0T8ILLDH(B`@(JK$*$-&*2PS*:HB("`@("!!4T,@("*J020`[CY!"HL@
M5"2SL2)3(B"G(,HH1$DD*#$L0T8I+#DL,RFR(D))3B(`_CY&"ID@1$DD*#$L
MB0T8I``0_6@KL`!H_;@I#0;)"K#(U-B`Z(*`Q(#H@C@ `````
`
end
begin 640 lrr.bin
M`(!,#(),7(-,A89,\X=,,(5,Z(1,/8!,HX5,2XA,@(C+A``````````````
M`,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
M````2*D`A9"M(8`@L?^I;R"3_ZE5(*C_))`P#ZDP(*C_:""H_R20,`(88*D%
MC2"`.&"M`-U)$(T`W6"I""P-W/#[8""3@*X,W""*@(I@J1H@6X"0`6`@KO\D
MD##.&"!'_RP-W""*@"";@(T@@"D/R0*P):` ()N`F56`R,`&D/48J0@@6X"0
M`6"I`2PB@#`"J00@J/\@KO]@2(HI`0H*"@HL(H`0`DD0(%N`:)`!8""H_ZD!
M(*C_J0D@J/\@KO]X&"!'_RP-W""*@*GVH(J%`H0#J0"%!"PB@#`#($:!(%Z!
ML`GF!*4$R0F0ZQA88*GVA0*F!!BIBGU5@84#8``($`8.!`P""B";@(T@@"D/
MR0*0`6"B`J``J0@L#=SP^ZT`W4D0C0#=K0S<D0+(T.GF`\K0Y&!(A`2**0$*
M"@H*"0(L(H`0`DD0(%N`:)`!8""H_Z4$(*C_J0$@J/\@KO]XJ4"%!3@@1_]X
M+`W<H@*@`*T`W<T`W=#X104I0/#RL0*-#-RE!4E`A06I""P-W/#[R-#=Y@/*
MT-@8($?_+`W<((J`().`K@S<((J`BHT@@"D/R0)88*D.C0#_J?^-2H"-2X"B
M!YU-@,H0^HU,@!A@S4J`T!#L2X#0"XB8"AAIBJBI]AA@C4J`CDN`A`4@\8"0
M&:T@@"D/R0OP`CA@(*:`K4J`KDN`I`60VV"M2H"N2X"D!4PF@J+_Z#CI$K#Z
MB!#W&&D2R*B*8"!L@J(`P`F0"$B8Z0FH:*(!R&"&"R!^@H4(A@F$"J4(I@FD
M"B`F@I`!8(4,A`VB`J``L0R1!LBQ#)$&R-#TY@WF!\K0[>8*I0K)"I`2J0&%
M"N8)I0G)`I`&J0"%">8(Q@O0NQA@..D"L`&(KCZ`X`'P!PJ$!R8'I`<8;42`
MD`'(8"#C@J+VA@:BG(8'KCZ`3)&"('Z"S4J`T` L2X#0"DBI XU*@(U+@&@@
```

```
MC(%@```@XX*B]H8"HIR&`XTK@XPL@R`0@Y`!8*T^@,D"L`%@K2N#K"R#&&D!
MD`'(("#8*D.C0#_J0"@`"!^@B`F@I`!8(4"A`.@#;$"C3Z`R0.0!ZD\C2"`
M.&"@$+$"R0+0\:`6L0*-/X#)!+#FH!&Q`HU!@,F!L-M*2DI*C4"`H!.Q`HU"
M@,BQ`HU#@*`8L0+)"="_H!JQ`LD"T+>@#K$"R0'0KZT_@`H8:0&-18`8;4"`
MC42`K4*`K$.`..U$@+`!B(U&@(Q'@*T^@,D"T`9.1X!N1H`8K4:`:0*-2("M
M1X!I`(U)@*GVH*"%!H0'J0&@`*X_@""1@I`!8*GVH*:%!H0'K46`H`"N0(`@
MD8*0`6"I]J"FKD&`&&"%!80#1@-JA02F`PHF`QAE!(4"BF4#A0,8I0)I]H4"
MI0-IH(4#H`*Q`ID&`(@0^&`@1(2E!2D!T`BE!RD/J*4&8*4'H@1&"&K*T/JD
M"&`@1(2E"BD/A0JE!2D!T`^E"84&I0<I\`4*A0=,Q82B!`8))@K*T/FE"H4(
MI0<I#P4)A0>@`KD&`)$"B!#XC$R`8#BM)X#I]JTH@.FF2BD'JJG_G4V`8*`.
MC`#_K2>`K"B`A0*$`ZGEH`"1`J`:L0*%#LBQ`H4/I0_)!9`#3-.$J*4.('2$
M2)A(J0"%"84*I0ZD#R"2A&B%#VB%#DP(A:D.C0#_K4R`\"ZI`(U,@*D"A6&I
M`84.K3^`A6"I]J"@A0*$`Z4.H`@$(.0`6#F#L9@T/#&8=#?K46`A0ZM0("%
M8*D`A6&I]J"FA0*$`Z9AO4V`\!"I`)U-@*4.H``@$(/&`\8#Y@[F8>8#Y@/&
M8-#=&&"@#HP`_ZD"H`"%#H0/A&"$8:4.I`\@=(2$`@4"T`;F8-`"YF'F#M`"
MY@^E#LU(@*4/[4F`D-NN/H`&8"9ARM#YJ0"D8*9A8*5DI&4@_X*0`6"I]J"<
MA6"$8:D`A6*M/H`*A6.E9*1E('2$A62$9<`%D!RM)8"%8JD!KCZ`X`'P`JD#
M+2:`T`6N)8#P`H5C(#6&8*8/T`(88"#)_Y`$C2"`8*G_IF/0`J5BA6:@`+%@
M)`X0!JJ]]HCP`R#2_\C$9M#L&*5@96:%8)`"YF$XI6+E9H5BL`+&8Z5B!6/0
MQ"#,_QA@H`Z,`/^%#H8/K2.`K"2`A62$94RAAB#JA9`!8*5ER060]!A@A0Z&
M4*D`A5&%5(558*D`A6*%8Z94\`2I`!A@IE`@QO^0!(T@@&"I_Z93T`*E4H5F
MH``@((>0`6#P!Y%@R,1FT/&$9ABE8&5FA6"0`N9A&*5B96:%8I`"YF,XI5+E
M9H52L`+&4Z54T`:E4@53T+@@S/^E8@5C&&"E5/`!`*51\`BI`(51J0H88*55
M*4#P"*G_A52I`!A@(,__II"&59`'C2"`(,S_8"0.$`RJO?:)\,?)#=`"A5&B
M_QA@I5K-2("E6^U)@)`!8*5:I%L@=(2$`@4"T`:E6J1;&&#F6M#<YEM,8X<@
M8X>0`6"%"80*I60%9?`2I62D9:8)AF2F"H9E()*$3,>'K2>`K"B`A0*$`Z`:
MI0F1`H5DR*4*D0*%9:G_H`^%"80*I62D92"2A*5DI&4@+8.0`6`8I5QE8H5<
MI5UE8X5=D`+F7AA@H`Z,`/\@J8:I`(5DA66%6X5<A5V%7JD"A5JI]J"<A6"$
M8:T^@`JHJ0"%4H13(+:&D`%@\`8@C(>0X&"M)X"L*("%`H0#H@"@'+5<D0+(
MZ.`#D/8@TX088*`.C`#_A@^JJ0`@J8:I]J"<A6"$8:D`H`2%4H13(+:&L!+P
M$*GVH)R%8(1A(#6&L`-,6(A@H`Z,`/^J(,;_D`RI`(TL@(TI@(TJ@&`@Y(@@
MY(@@Y(B-*8`@Y(B-*H"I`(TL@"#DB,DB\`C)0M#U(,S_8*``(.2(R2+P!IDM
M@,C0\XPL@"#DB,D@\/F-*X`@Y(C)`-#Y(,S_8"#/_[`%))!P`6!H:"#,_TR+
MB```````````%`D-`),````````````````("$B(R0E)B<H*2HK
M+"TN+S`Q,C,T-38W.#DZ.SP]/C]`P<+#Q,7&Q\C)RLO,S<[/T-'2T]35UM?8
MV=I;7%U>7\!!0D-$149'2$E*2TQ-3D]045)35%565UA96MO<W=[?``````````
M```````````````````````````````````````````````````````````````
M```````````````````````````````````````````````````````````````
M```````````````````````````````````````````````````````)````
M#0````````@```````````````"`A(B,D)28G*"DJ*RPM+B\P,3(S-#4V-S@Y
M.CL\/3X_0&%B8V1E9F=H:6IK;&UN;W!Q<G-T=79W>'EZ6UQ=7E\````````````
M```````````````````````````````````````````````````````````````
M```````````````````````````````````````````````````````````!@
M04)#1$5&1TA)2DM,34Y/4%%24U155E=865I[?'U^?P``````````````````
2```````````````````!^````
`
end
```

7. THE FUTURE

Future improvements to this program would include implementation of MS-DOS formatting, more file manipluation commands (such as Rename), re-writing the user-interface BASIC program in machine language, and making a file buffering facility for those people with only one disk drive. However, I don't intend to do much more to this program. My intentions are to put this functionality into a device driver for a new operating system (or at least, operating environment) for the C-128. Anyone else is free to improve this program.

==========

In the Next Issue:

TWO-KEY ROLLOVER

This article will examine how a two-key rollover mechanism would work for the keyboards of the C-128 and 64 and will present Kernal-wedge implementations for both machines. Webster's doesn't seem to know, so I'll tell you that this means that the machine will act sensibly if you are holding down one key and then press another without releasing the first. This will be useful to fast touch typers.

The Second Rob Hubbard Music Routine

In this article, the second Rob Hubbard music routine will be presented in the same way as the first. Future issues will hopefully examine various other music routines including various Martin Galway, Benn Daglish, Jeoren Tel, and Manaics of Noise routines. Note: Unfortunately the author completes university (and thus loses internet access) in August 1993.

DYCP - Horizontal Scrolling

DYCP - is a name for a horizontal scroller, where characters go smoothly

up and down during their voyage from right to left. One possibility is a scroll with only 8 characters - one character per sprite, but a real demo coder won't be satisfied with that.

Multi-Tasking on the C=128 - Part 2

This article will examine the actual code that makes up the multi-tasking kernal in detail and include some example programs explaining it use.

The 1351 Mouse Demystified

This article will explain how the 1351 mouse works as well as provide an easy to use interface in machine language for both basic and machine language programmers.

==========END===