```
                   ########
             ##################
         ######            ######
      #####
    #####  ####  ####      ##      #####   ####  ####  ####  ####  ####   #####
  #####    ##    ##      ####    ##   ##   ##  ###     ##    ####  ##   ##   ##
 #####    ########     ##  ##   ##        #####       ##    ## ## ##   ##
#####    ##    ##    ########  ##   ##   ##  ###     ##    ##  ####   ##   ##
#####  ####  ####  ####  ####  #####   ####  ####  ####  ####  ####   ######
#####                                                                    ##
 ######            ######        Issue #8
   ##################            Aug. '94
       ########
```

Editor's Notes:
by Craig Taylor (duck@pembvax1.pembroke.edu)

     Woe be to Commodore,
     The marketer's have finally killed it,
     With a little bit of spending here,
     And not much over there,
     You know that Commodore has finally died.

     And that's the way life should be,
     The Commodore fanatics cried.
     We'll probably be better off they yell,
     Let Commodore go to hell.
     So the question is who'll purchase Commodore?

Yes, for those of you who are unaware Commodore has declared bankruptcy. There are numerous rumours abound over whose interested in what divisions of Commodore and such - there's still no definate word on the net. Several factors can be blamed for Commodore's demise: Commodore _never_ was successful in marketing products. The engineers would often turn out miracle machines that the marketing department (I wonder if there even was one) promoted badly, if at all. What has put the nail in the coffin for Commodore was the lack of financial capital to keep the company in operation. A lot of this news has been discussed on GEnie, the newsgroup Comp.Sys.Cbm, and various magazines as well so I won't elaborate any further.

Speaking of magazines, Creative Micro Designs has started a magazine for Commodore 8-bit's called, "Commodore World". The magazine is very well done. There are other magazines that also deserve mention: DieHard, the Underground and many others. Commodore may be bankrupt but the Commodore will still live forever.

Speaking of living forever, Commodore Hacking is looking for articles on any subject dealing with any aspect of technical programming or hardware on the Commodore. If you've got an article already written, or an idea for one _please_ feel free to e-mail me via duck@pembvax1.pembroke.edu. Many thanks to the authors whose works make up this and previous issues.

==

Please note that this issue and prior ones are available via anonymous FTP from ccnga.uwaterloo.ca (among others) under pub/cbm/hacking.mag and via a mailserver which documentation can be obtained by sending mail to "duck@pembvax1.pembroke.edu" with a subject line of "mailserver" and the lines of "help" and "catalog" in the body of the message.

==

==

In This Issue:

Commodore Trivia Corner

This section of C=Hacking contains questions that test your knwoledge of tricks and little known-information for the Commodore computers. Each issue they'll be answers to the previous issues questions and new questions. How much do you know?

RS232 Converter

This article, with a minimum of parts, details how to make your own RS-232
converter.

Programming the Commodore RAM Expansion Units (REUs)

The REC chip is a DMA (direct memory access) chip that allows for the
Commodore 64 and 128 to use the Ram Expansion Units. This article examines
how to access the chip in your own ML programs.

A Different Perspective: Three-Dimensional Graphics on the C64

In this article, co-written by Stephen Judd and George Taylor, is
presented all the basic graphics tools and mathematical theory behind
3d graphics. The basic tools are using a charset to make graphics,
plotting a point, drawing a line, clearing the graphics, and double
buffering. The 3d tools are defining a 3d object, rotation of the
object in 3d space, and perspective viewing on a 2d screen.
Programs are presented in basic 2.0, basic 7.0, and assembly which
show a rotating cube outline.

Design of a 'Real' Operating System for the 128: Part I

Written Craig Bruce this article examines a 'real' operating system for the
Commdore 128. It focuses on the OS being Multi-tasking, Distributed and based
on a MicroKernal. Why? As he states, "Because I'm designing it, and that's
what interests me. The ease-of-construction thing is important too. Another
important question is 'can it be done?'. The answer is 'yes.' And it will
be done, whenever I get around to it (one of these lifetimes)."

==

Commodore Trivia Corner

by Jim Brain (brain@mail.msen.com)

It is time for another dose of trivia! As some of you may know, The
Commodore Trivia Editions are posted every month to the USENET newsgroups
comp.sys.cbm, alt.folklore.computers, and comp.sys.amiga.advocacy. This
article is a compilation Trivia Editions 2-8, with only the questions
appearing for Edition 8. These are part of a Trivia Contest in which
anyone may enter. If you wish to participate in the newest
Trivia contest (Which is on Trivia 8 as I write this), please send your
answers to me at 'brain@mail.msen.com'.

The following article contains the answers to the January edition of trivia
($00A - $01F), the questions and answers for Febrary ($020 - $02F), March
($030 - $03F), April ($040 - $04F), May ($050 - $05F), June ($060 - $06F),
and the questions for the July edition ($070 - $07F). Enjoy them!

Here are the answers to the Commodore trivia questions for January, 1994.

Q $00A) What was the Code-Name of the Amiga while in Development?

A $00A) Lorraine. Amiga was the company name. When Commodore bought the
company, they scrapped the model name and used the old company name.

Q $00B) What is Lord British's Real Name (The creator of the Ultima
Series)?

A $00B) Richard Garriott. Scott Statton has met him and says that he is son
of astronaut Owen Garriott.

Q $00C) What is the POKE location and value that will fry an early model
PET?

A $00C) 59458. It is in the (Versatile Interface Adapter, 6522)
No, I won't tell you what to poke into it, but I will tell you
that it is not the only way to fry a PET. here is a description from
none other than Jim Butterfield

"The poke shopwn above is correct. Its intention was to speed up early
model PETs by masking the RETRACE line (by switching it to output)...

however, Commodore subsequently REDESIGNED the interface in such a way
that making the VIA pin an output caused (now) two outputs to fight
each other ... result, VIA and/or video circuitry burnt out.

LATER (Days of "fat 40" and 80-column PETs), the new CRT controller
chip could be fiddled with POKES so that it generated scan rates

```
        completely out of the capacity of the CRT deflection circuits.
        Result: burnt out deflection circuitry ... and that was no YOKE!"

        Richard Bradley says that 59595 is the second poke that Jim is
        referring to.

        I also have in on word from Ethan Dicks that 59409 is another
        infamous poke, but I wouldn't try any of these!

Q $00D) On the Plus 4 and C-16, the VIC chip was replaced with the TED
        chip.  What does TED stand for?

A $00D) TED = Text Editing Device.  It did not have as many capabilities
        as the VIC II.

Q $00E) Commodore Produced a daisy-wheel letter quality printer in North
        America (maybe elsewhere) for the Commodore Serial Line.  Name it.

A $00E) The Commodore DPS 1101.  The CBM 6400 was another earlier attempt
        at a daisy-wheel printer, but it had an IEEE-488 interface.

Q $00F) What is the version of DOS in the 1541?

A $00F) 2.6

Q $010) What is the Version of BASIC in the Plus 4 and the C-16?

A $010) 3.5.

Q $011) What are the nicknames of the original three custom Amiga chips?

A $011) Daphne/Denise, Agnes/Agnus, and Paula/Portia, or Huey, Duey, and Louie.
        Denise, Agnes, and Paula were the American names, but the the others
        crept in from somwhere.  the ducks were always a joke, but caught on
        as alternate names.

Q $012) Commodore produced a 64 in a PET case.  What is its name and model
        number?

A $012) The Educator 64.  It was model number CBM 4064, and it was also called
        the PET64.  Note that this version of the 64 was the second attempt.
        Commodore first tried to sell the "Educator 64" to schools in the
        regular 64 case, but administrators and teachers disliked the "homey"
        look.  Thus, it was squeezed into a PET case and sold better, although
        I don't think it was ever a killer seller.

Q $013) Commodore sold a 1 megabyte floppy disk drive in a 1541 case.
        Give the model number.

A $013) The Commodore SFD 1001.  It was actually half of an CBM 8250 LP
        with a slightly revised ROM.

Q $014) What does GCR stand for?

A $014) Group Code Recording.

Q $015) Commodore produced a drive to accompany the Plus 4 introduction that
        was designed specifically for the Plus/4.  Give the model number.

A $015) the CBM 1551 was the new, high-performance drive that was designed
        specifically for the Commodore Plus/4 and C-16.  The 1542 was
        actually just a repackaged 1541 in a grey case that was made available
        for people who didn't want to spend the extra money for the 1551.  The
        extra cost resulted from the 1551 sporting a new, parallel transfer
        method that increased transfer rates 400%.

Q $016) What does SID stand for?

A $016) SID = Sound Interface Device

Q $017) What does the acronym KERNAL stand for?

A $017) KERNAL = Keyboard Entry Read, Network, And Link.  This is most likely
        another "words after the letters" acronym, along the lines of the
        PET acronym.

Q $018) What version of DOS does the 1571 have?

A $018) 3.0

Q $019) What other two Commdore Disk Drives share the same DOS version
```

number as the 1571?

A $019) I got more than I bargained for on this question, since there are four drives which have the same DOS version that I feel are adequate responses to this question.

The CBM D9060 and D9090, although I doubt the code is the same. The D series were hard drives.

The 8280 Dual 8" Floppy Drive.

The 1570, which was a single sided version of the 1571 in a 1541 case painted to match the 128. The ROM is slightly different, enough to make it unrecognizable as either a 1541 or a 1571 in some cases.

The 1571II and the 1571D, which is the drive in the C128D, also have this DOS revision, but that stands to reason, since they are in the 1571 line.

Q $01A) How many files will the 1571 hold?

A $01A) 144 in both modes. I am surprised Commodore didn't add a track or put another directory on the back.

Q $01B) How many files will the 1541 hold?

A $01B) 144.

Q $01C) What did Commodore make right before entering the computer market?

A $01C) Calculators. They also made office equipment, watches, adding machines, and thermostats, hence the name "Commodore Business Machines".

Q $01D) Commodore introduced an ill-fated 4 color plotter. Give the model number.

A $01D) the Commodore 1520. It used 4 inch wide paper and could use 4 colors.

Q $01E) Some formats of CP/M write disks using the MFM format. What does MFM stand for?

A $01E) MFM = Modified Frequency Modulation

Q $01F) On the Commdore 128, the user manual left three commands undocumented. One works, and the other gives a not-implemented error. Name the commands and what each one does or does not do.

A $01F) RREG reads the internal registers after a SYS command.
OFF gives an unimplemented command error.
QUIT does too.

Here are the answers to Commodore Trivia Edition #3 for February, 1994.

Q $020) What does the letters IEEE in IEEE-488 stand for?

A $020) Institute of Electrical and Electronics Engineers.

Q $021) What was the logo of Batteries Included?

A $021) It was a the face and hands of a man with glasses inside a circle. Early renditions of him were in black and white, while later ones had him with blond hair a a red shirt. Some views had him actually typing on the 64/VIC with one finger, but most just showed him, not the keyboard.

Q $022) The Commodore VIC-20, 64, and 128 computers emulate in software a very important integrated circuit. What is its number, and why is it important?

A $022) The 6551 UART IC. It is used for RS-232 communications.

Q $023) Commodore watches play a beautiful song for the alarm. What is the song's title?

A $023) Fleur-de-lis. The "Godfather" theme.

Q $024) The C2N style Commodore tape decks are impressive in handling errors. How many times is a single program stored onto tape?

A $024) Twice, second copy is placed right after the first. That means, even if you get a load error on load, you might be able to just run the program anyway, as a load puts the first copy in memory, and verifies it against the second copy.

Q $025) What is a jiffy?

A $025) A jiffy is 1/60th of a second. It is the same on PAL and NTSC Commodore computers.

Q $026) What is the screen resolution of the Commodore VIC-20?

A $026) On the VIC-I IC, the text and graphics screens are definable within limits. Therefore, there are a number of answers that are correct:

The default screen has (and the answers I was looking for):

```
Text:      22H x 23V = 506 characters
Graphics: 176H x 184V = 32384 pixels
```

However, on experimentation with a NTSC VIC-I (6560), I found that it could support a resolution of:

```
Text:      24H x 29V = 696 characters
Graphics: 192H x 232V = 44544 pixels
```

Your mileage may vary, but these numbers remove all border area. (I am not sure if you can use all the pixels, since the VIC-I only allows 32768 to be used. You might be able to flip the graphics page in the middle of the screen, but I leave that as an exercise.)

The VIC-I also supports a virtual screen, which can be "panned" so that the physical screen becomes a "window" into the virtual screen. The maximum "scrollable" virtual screen on NTSC is:

```
Text:      28H x 32V? = 896 characters
Graphics: 224H x 256V? = 57344 pixels
```

The VIC supports more resolution than 32V, but you can never see it since you can't scroll it into view, so the point is moot.

So, if I didn't thoroughly confuse you, email me and I will make sure I do!

Q $027) Why is the VIC-20 named the VC-20 in Germany?

A $027) Because 'V" is pronounced 'F" in Germany, and the resulting pronunciation was a naughty word.

Commodore put one over on many people. The VIC-20 was designed in the states and given that name due to the IC that did the graphics. When the marketing started, CBM found out the name was no good in Germany, so they quickly renamed it VC-20. The after-the-fact Volks-Computer conjured up images of the Volkswagon car (VW), which was popular at the time for its dependability and price. The rest is history...

Q $028) Why was early Commodore equipment built into such heavy enclosures?

A $028) Simple. Commodore made office furniture, which includes desks and filing cabinets. They simply used the facilities and parts on hand. The fact that, at the time the PET came out, people equated physical stability of a machine as an indication of its worth, served only to reinforce the decision. Also, the system had to hold up the built-in monitor.

Most people think it is due to FCC regulations. FCC regulations had not been determined at the time the PET came out, although the engineers did know that the CRT produced many electrical hazards which could be alleviated with a shielded metal case. Commodore has always been a "cheap" company, so the fact that they could get good shielding in-house at almost no cost proved to be the overriding factor. It might interest some to note that, even with the metal case, early PETs had foil inside as a secondary shield. The reason has to do with the keyboard being mostly plastic, as the shield fit directly underneath, but the reason for it remains a mystery to me.

Q $029) What two BASIC 2.0 commands might still work if mispelled?

A $029) The answers I was looking for are END and STOP, although someone correctly pointed out that GO TO can be construed as a mispelling. Also, print#, get#, and input# might work if the '#' was omitted and the program was getting data to screen or keyboard.

Although the following aren't really the result of mispelled commands, I put them in, since you could stretch the definition of mispelled to include them.

LET would work if it was left out, since LET was an optional keyword. Commands of the form <keyword> <number or variable> would work if letters were tacked onto the end. (example: RUNDY., prg has a valid line 0, and DY = 0). Finally, LOAD"jim",8,1garbage would work due to the way LOAD absolute worked, but that is a stretch!

Q $02A) What does CIA stand for? (not the U.S. CIA!)

A $02A) CIA = Complex Interface Adapter. The german Magazine 64'er calls it a Control Interface Adapter, but that is not its official name.

Q $02B) (hard one) What is the key VIC capability that makes full-screen hires graphics possible on the _VIC-20_?

A $02B) A lot of people answered redefinable characters, but that alone does not provide FULL-SCREEN graphics. 256 8*8 cells gives you a little over 1/2 of the screen in graphics, but the VIC has the ability to make each character cell be 8*16, which gives enough pixels to map the entire default screen.

Q $02C) How many cassette ports does the CBM 8032 computer have?

A $02C) Two. One on back, one on side near the back.

Q $02D) What 5 bytes must appear in every Commodore 64 autostart cartrdge and what location in memory must they be placed at?

A $02D) CBM80 at $8004. The letters must have bit 7 set. So, the actual PETSCII codes are 195, 194, 205, 056, 048.
                $c3, $c2, $cd, $30, $30 in HEX

Q $02E) What is the correct Commodore technical term for "Sprites"?

A $02E) MOBs, or Movable Object Blocks.

Q $02F) (Three parter, all parts must be correct) "Push-wrap-crash" is a nickname for a condition that can lock up an old-style C=64.
What causes it?
How can it be avoided (besides not doing it)?
What is the only way out once it has occured (besides rebooting)?

A $02F) Wow, I got so many responses to this! This question actually dealt with a typical user, but people sent in descriptions of what the code does and how to patch it. So, there are two sets of answers to this:

User Answer:

1) If you put the cursor at the bottom of the screen and type 82 characters (not 81) and then trying to delete back to the 78th one.
2) Any of the following will work:

   Do not use the following colors for the cursor: red, blue, yellow, light red, dark grey, light blue, light gray.

   Some people devised a IRQ wedge that will recover from the lockup.

   Have the following lines as the first lines of a program:
   10 open 15,8,15 20 input#15,a$.
3) There are actually two ways to recover. They are:

   If you have a reset button installed on the 64, reset the machine, then load and run an unnew program. (I accepted this, but I figured most people would assume this much)

   If you have a tape drive installed, press either Shift-3 or move a joystick installed in Port 1 in the UP direction. Then, respond to the directions on the screen "PRESS PLAY ON TAPE". Next, press RUN-STOP to stop the tape load.

What really happens: (I can't prove this)

1) The user types the line of text and the scroll code is invoked. The first two lines become linked as one logical line, and the third line is treated as a new line.

   The user deletes the 82nd and the 81st character and then hits delete while in the first column of the third line. Since the delete will put the cursor back up into the second line, which is linked with the first, the KERNAL gets confused and thinks the second line is at the bottom of the screen. Remember, the "cursor" is actually constructed by a combinations of using reverse characters and changing the color RAM nybble for that screen location. Thus, when the cursor gets "erased" from the first column of the last line, the KERNAL thinks the color nyble for it is at $DC00, which is 40 bytes off from the actual position. $DC00 is actually Port A for CIA #1, which is where the kernal writes the column of the keyboard it wishes to scan. Because the KERNAl is messed up, it puts the color nybble for where it thinks the cursor was into this location. (That is why there is a connection between cursor color and this bug.

   Now, the system integrity has been compromised, but it does not show yet. The user proceeds to delete the 80th character. As the user deletes the 79th character, the bad value in $DC00 goes to work and fools the KERNAl into thinking SHIFT/RUN-STOP has been pressed. It also pretty much disables the keyboard.

2) Since the Color RAM is what the KERNAl gets confused about, the solution was to not use certain bit patterns of colors:

RED	0010
CYAN	0011
BLUE	0110
YELLOW	0111
LIGHT RED	1010
DARK GRAY	1011
LIGHT BLUE	1110
LIGT GRAY	1111

   OK Colors:

BLACK	0000
WHITE	0001
PURPLE	0100
GREEN	0101
ORANGE	1000
BROWN	1001
MEDIUM GRAY	1100
LIGHT GREEN	1101

   All of the BAD colors have bit 1 set. I have no idea what the significance of that is.

3) You needed to get out of the tape load code, but you only had so many keys that were still active. So, if you followed the directions on the screen, you could break out. Since the tape load code uses CIA #1 for its operations, it would take over the IC and then restore it to a correct state when either the load was stopped or the load completed. Now, that is amazing!

   (Someone is free to check up on me concerning this, since I do not have a Rev 1 ROM to try out. If someone has one, I would like to have a copy of it on disk or in email. And if someone has the information on this bug from either the May 1984 Gazette p108, or from the COMPUTE! Toolkit Kernal VIC20/64, I would like a copy.)

Here are the answers to Commodore Trivia Edition #4 for February, 1994.

Q $030) On a Commodore 64, what is the amount of RAM available for BASIC programs to reside in?

A $030) Some people over-answered this question. The correct answer is 38911 bytes, which is what the BASIC screen says. Now, it is true that BASIC can use $C000-$CFFF, and some zero pages is easily used by BASIC, but it is non-trivial to get BASIC to use these areas. The math comes out to: $0801 (2048) to $9FFF (40959) - 1 (0 in location 2048). Please note that this is not the maximum size of a standard BASIC program, even if it does not use variables, since

BASIC steals 3 bytes at the end of the program to determine the end.

Q $031) Name one Commodore computer (pre-Amiga) that used two general purpose microprocessors?

A $031) There are two (or more) answers to this question. The obvious answer is the Commodore 128, but the Commodore SuperPET (SP9000) had two, also. There was also an optional card to add another processor to the B-series. Note that some Commodore peripherals also had two (or more) microprocessors, but that is another question.

Q $032) What are they?

A $032) Commodore 128: 8502(6510 clone) and Z80. SuperPET: 6502 and 6809. B-series: 6509 and 8088.

Q $033) Who was the Chief Executive Officer of CBM when the Commodore VIC-20 (VC-20) was introduced?

A $033) According to my sources, it is none other than Jack Tramiel. While some claim Irving Gould as the man-in-charge since he had controlling interest at the time, the CEO was Jack. Whether he was in charge or not is left up to the reader.

Q $034) the Commodore 64 and 128 (among others) have a TOD feature. What does TOD stand for?

A $034) TOD = Time Of Day. The 6526 Complex Interface Adapter is the holder of the TOD clock, which can be used in lieu of the system jiffy system clock to time things, as it does not suffer from interruptions to service I/O and screen. Note that the standard kernal uses the system clock for TI and TI$, not the TOD clock.

Q $035) What location in the Commodore 64 Kernal holds the version number?

A $035) $ff80 (65408).

Q $036) The first computer Commdore sold was the KIM-1. How much RAM was available on the KIM-1?

A $036) 1.125K or 1024+128 = 1152 bytes.

Q $037) Who designed the architecture for the 6502 integrated circuit?

A $037) Chuck Peddle

Q $038) What was the original name of the company that produced the 6502?

A $038) MOS Technologies

Q $039) What did the name stand for?

A $039) MOS = Metal Oxide Semiconductor, which has three major families: NMOS: Negative MOS, PMOS: Positive MOS, and CMOS: Complementary MOS. MOS Technologies produced mainly NMOS ICs, hence the use of NMOS technology for the 6502 and 6510.

Q $03A) Commodore acquired the company and renamed it to...?

A $03A) CSG = Commodore Semiconductor Group. The renaming was not instantaneous, happening a number of months(years) after the acquisition.

Q $03B) The Commodore VIC-20 graphics were powered by the VIC-I (6560) integrated circuit. Was the chip designed for the computer, or was the computer designed for the chip?

A $03B) The VIC-I 6560-61, was designed 2 years prior to the design of the VIC-20 computer. It was designed to be built into video games, but no one wanted to use it, so Commodore made their own system around it to recoup losses.

Q $03C) The VIC-20 had a Video Interface Chip (VIC) inside it, yet that was not what the 'VIC' in the model name expanded to. What did it expand to?

A $03C) VIC-20 = Video Interface Computer-20. The 20 was a rounding down of the amount of memory in the VIC: ~22K. Michael Tomczyk, who got stuck with the job of deciding on the name, did the rounding.

Q $03D) The most widely known disk drive for Commodore computers is the 1541.

how much RAM does the 1541 have?

A $03D) 2048 bytes, or 2kB RAM. It is mapped at $0000-$07FF.

Q $03E) On every Commodore disk, the drive stores a copy of the BAM. What does BAM stand for?

A $03E) BAM = Block Allocation Map, or Block Availability Map. I am checking sources to figure out which one is the real McCoy.

Q $03F) Now, for those into 6502 machine language. What instruction was not available on the first 6502 chips?

A $03F) ROR (ROtate Right) was not available until after June, 1976. However, all Commodore VICs and C64s should have this instruction. Some people gave instructions that are found on the 65c02, designed by Western Design Center, and licensed to many companies. However, the 65c02 itself occurs in two flavors, and neither are used in any stock Commodore product I know of.

Here are the answers to Commodore Trivia Edition #5 for April, 1994.

Q $040) The company that produces The Big Blue Reader, a program that allows reading and writing of IBM formatted disk in 1571s and 1581s, is called SOGWAP. What does SOGWAP stand for?

A $040) Son Of God With All Power. They also market the Bible on diskettes.

Q $041) What version of DOS does the Commodore 8280 8 inch dual drive have?

A $041) The 8280 has version 3.0. Many have not ever seen this IEEE-488 compatible drive used on some PETs. It has the same DOS version that is in the D90XX hard drives, and could read 250kB and 500kB IBM formatted disks, as well as some CP/M formats. Note that although this version number is used on the 1570/71 disk drives, the code is different.

Q $042) What was the color of the original Commodore 64 case?

A $042) Some early versions of the Commodore 64 were housed in VIC-20 color cases, so off-white is the correct answer.

Q $043) On an unexpanded Commodore 64, how does one read the RAM locations $00 and $01?

A $043) Well, you cannot do so with the CPU directly, since it resolves these locations into internal addresses. However, the VIC II can see these addresses as external memory. So, just make one spritexs with the first bit in the sprite set, and move it over the first two bytes, pretending they are part of a bitmap. By checking the sprite-to-background collision register, you can tell if the bit in the byte is set. Email me for a more complete description.

Sven Goldt and Marko Makela get credit for this answer and the next.

Q $044) On an unexpanded Commodore 64, how does one write the same locations?

A $044) It seems the 6510 generates a valid R/W signal any time it does an internal read or write. This is to be expected, since the 6510 internal registers were grafted onto a 6502 core processor. Howevere, the address lines are also valid during any internal read or write, since failure to do so may write the data on the data bus to some invalid address. The data on the bus, however, comes not from the CPU, but from residual effects of the data last read of written by the VIC chip. Thus, by programming the VIC chip to read data from some known location, and by placing relevant data in that location, a write to location $00 or $01 will place the data from that last read VIC location into $00 or $01. This is usually accomplished by placing the data to be written out into location $3fff, which the VIC fetches during the time the border is being displayed. By triggering a routine when the raster hits the bottom border, you can copy location $3fff to $00 or $01.

Q $045) What is 'CB2 Sound', and on what computers was it popular?

A $045) This is the sound made by sending square out of the 6522 IC on some Commodore computers. It is called 'CB2', since that is the name of the pin on the 6522 that outputs the waveform. I won't go into a complete description, except to say that most models of the PET had the capability, and most PET owners used it as the ONLY sound source, since the PETs did not have a sound chip. Although the VIC did have some sound capabilities, by that time Commodore had realized its widespread use and included some information on it in the Commodore VIC-20 Programmer's Reference Guide. For more info, reach for your nearest VIC PRG and look at page 232.

Q $046) in question $021, the Batteries Included logo description was asked for. Now, what is the name of the man in the logo?

A $046) "Herbie" Jim Butterfield supplied me with this one.

Q $047) Why was the Commodore VIC-20 produced with so many 1K chips in it? (Hint: it had little to do with the cost of SRAM at the time)

A $047) Jack (Tramiel) decreed that Commodore had a surplus of 1K chips, so he didn't care how much memory it had, as long as the designers used 1K SRAMs.

Q $048) What does ADSR stand for?

A $048) ADSR = Attack, Decay, Sustain, Release. These are the four values specified to define a SID waveform envelope.

Q $049) In question $035, it was learned that the Commodore 64 kernal revision number is stored at $ff80 (65408). Now, what is the number stored there for:

a) The first revision?
b) The PET64 (4064)?

A $049) a) 170. (Yep, this was prior to 0!)
b) 100. (The PET 64 uses this value to adjust the startup logo accordingly.)

Q $04A) Who was the mastermind behind the original Commodore Kernal?

A $04A) John Feagan. He had intended it to provide upward compatibility for future computer systems. Unfortunately, the kernal was modified enough with each new computer system, that the idea of compatibility never really surfaced. Still, it was a nice try.

Q $04B) Who designed the first VIC prototype?

A $04B) There are two answers to this question. At the time, the VIC had no name and was called the MicroPET or No Name Computer. Jack Tramiel wanted to show some prototypes of the VIC at the 1980 Comsumer Electronics Show (CES). The funny thing is, he got not one prototype, but TWO. Bob Yannes, working against time, had hacked together a minimal working prototype using spare PET/CBM parts. Another prototype, brought to the show by Bill Seiler and John Feagans, had been put together after some preliminary discussions with Yannes.

Q $04C) How many pins does a Commodore 1525 printhead have in it?

A $04C) Trick Question. The two 1525 printers I have show that the 1525 printhead has but one pin. The seven dots are created by a revolving 7 sided star-wheel for the platen, which presses the paper against the printhead in the seven different dot locations.

Q $04D) Why does mentioning a PET computer in France make people chuckle?

A $04D) PET means "FART" there.

Q $04E) What interface IC is used to drive the IEEE-488 bus in a PET computer?

A $04E) A 6520. It is appropriately called a PIA (Peripheral Interface Adapter).

Q $04F) What was the primary reason Commodore went to a serial bus with the introduction of the VIC-20?

A $04F) Jim Butterfield supplied me with this one:

As you know, the first Commodore computers used the IEEE bus to connect to peripherals such as disk and printer. I understand that these were available only from one source: Belden cables. A couple of years into Commodore's computer career, Belden went out of stock on such cables (military contract? who knows?). In any case, Commodore were in quite a fix: they made computers and disk

```
        drives, but couldn't hook 'em together! So Tramiel issued the
        order:  "On our next computer, get off that bus.  Make it a cable
        anyone can manufacture".  And so, starting with the VIC-20 the
        serial bus was born.  It was intended to be just as fast as the
        IEEE-488 it replaced.

        And it would have been, except dor one small glitch.  But that is
        another trivia question.


Here are the answers to Commodore Trivia Edition #6 for May, 1994.


Q $050) The Commodore 1551 Disk Drive is a parallel device.  How did it
        connect to the Commodore Plus/4 and C16?

A $050) The Commodore 1551 connected via the expansion port.  Therefore, it
        was a parallel device, and could work at much faster speeds.

Q $051) How many could you attach?

A $051) Two, The second drive cable attached to the back of the first cable.

Q $052) What were the addresses they used? (Not device numbers)

A $052) The two drives were mapped into the Address space at $fec0 and $fef0
        of the Plus/4 or C-16.  The 6523 Triple Interface Adaptor chip is
        mapped in at these locations and has 8 registers each.

Q $053) What is the maximum number of sound octaves the VIC-20 sound generator
        can reach?

A $053) This has two equally valid answers. On the Vic-20, each sound
        generator has a range of 3 octaves.  However, all the sound generators
        together can range 5 octaves, since each sound generator is staggered
        one octave apart.

Q $054) Who wrote the reference guide that was distributed with almost every
        PET computer sold?

A $054) The infamous Adam Osborne, of Osborne I fame.

Q $055) The box that the C64 comes in has some propaganda on the side
        describing the unit.  In the specifications section, it claims how
        many sprites can be on screen at one time?

A $055) I neglected to note that the Commodore 64 packing box has underwent
        many changes.  However, for quite a while, CBM used a blue box with
        many views of the 64, and a specification list on on side of the box.
        On that spec list, it claims that the the 64 can have "256
        independently controlled objects, 8 on one line."  Why is this
        important?  It gives us a clue that the VIC-II designers figured people
        would and could use the interrrupts on the VIC-II to change sprite
        pointers.

Q $056) The Commodore Plus/4 computer contained the first integrated software
        package to be placed in a personal computer.  What was the name of the
        software package?

A $056) The package was called "3+1".

Q $057) What popular computer software did the software package parody?

A $057) Lotus 1-2-3.

Q $058) One familiar Commodore portable computer was called the SX-64.
        What did SX really stand for?

A $058) Depending on whom you believe, the SX stands for two things.  If you
        choose to believe Jack Tramiel, the SX stands for "sex", since Jack
        has been quoted as saying, "Business is like sex, You have to be
        involved".  This is a plausible answer, as Jack usually picked the
        names of the computers.  However, if you don't buy that, here is the
        marketing version.  SX stands for Single Drive Executive, as the
        portable 64 was called the Executive 64.  There was to have been a DX
        model, which would have had two drives.  You decide.

Q $059) Who (what person) invented the Sound Interface Device (SID) chip?

A $059) Bob Yannes, who also worked on one of the VIC prototypes, developed
        this chip.
```

Q $05A) The ill-fated UltiMax (later called the MAX Machine) contained a
number of Commodore 64 features. However, it did not share the 64's
feature of 64kB RAM. How much RAM did the MAX have?

A $05A) A whopping 2 kilobytes. If you plugged in the BASIC cartridge,
memory dropped to .5 kilobyte or 512 bytes. No wonder CBM scrapped
this one.

Q $05B) What famous person was featured in U.S. television advertising for
the VIC-20?

A $05B) William Shatner. Yes, Captain James T. Kirk himself did the ads.
He was not, however, in uniform, since CBM did not have rights to
Star Trek of any sort.

Q $05C) What company designed the first VICModem?

A $05C) Anchor Automation. Sometimes called the "Most Inexpensive Modem",
the VICModem was designed to be sold for under $100 when most were
$400 or more. The secret to the cost containment was the ability to
use what we soetimes think of as a disadvantage of the User Port to
the modem's advantage. The TTL level RS-232 signals did not need to
be buffered before driving the modem, and the +5 volt power available
through the User Port just was not available through normal RS-232
lines. Not having the already TTL level signals would have meant
extra components that would have increased case size and cost, and not
having the on-board power would have meant a power connector and power
supply would need to be bundled. Being one of those people who used
the first VICModem, I can tell you it was worth the hassle.

Q $05D) Everyone has seen or heard of BYTE Magazine. Known for technical
articles in the 80's, and coverage of PC products in the 90's, BYTE
was founded by Wayne Green. What Commodore computer magazine did
Wayne Green later publish?

A $05D) RUN Magazine. As of right now, CMD has purchased the rights to RUN.

Q $05E) (Three part question) What are the official names of the colors
used on the VIC-20:

a) case?
b) regular typewriter keys?
c) function keys?

A $05E) a) ivory.
b) chocolate brown.
c) mustard.

Q $05F) Commodore is set up as a ___________ chartered company. Name
the missing country.

A $05F) Bahamas. Doing so gave CBM a great tax break. With the tax rate in
the Bahamas as low as 1%, more money could be kept from the
governments.

Here are the answers to Commodore Trivia Edition #7 for May, 1994.

Q $060) When you turn on stock Commodore 16, how many bytes free does it
report?

A $060) According to the initial power-up indication on the monitor, a stock
Commodore 16 has 12277 bytes free for BASIC program use. A number od
people have calculated 12287 bytes, so the power-on message may be in
error. I guess it is time to dig out the C-16 and power it up.

Q $061) How many does a stock Plus/4 report?

A $061) According to its initial power-up message, the Plus/4 has 60671
bytes free.

Q $062) What was the VIC-20's subtitle?

A $062) "The Friendly Computer"

Q $063) What personality announced the birth of the Commodore 64 in
Christmas advertisements?

A $063) Though not well-known outside of the US, Henry Morgan introduced the new Commodore 64 computer system in the US. In other countries, the answers differ, as countries like Finland had the Statue of Liberty announce the C64 birth.

Q $064) What was the name of the monitor program included in the Plus/4?

A $064) TEDMon. TED, as you know, stood for Text Editing Device.

Q $065) How many sectors per track are there for tracks 1-17 on a 1541?

A $065) 21.

Q $066) There are two programs running in the Commodore single-6502 drives (1541,1571,1541 II,1581). What is the interpreter program called?

A $066) The interpreter program is called the Interface Processor (IP). It handles the dispatching of all commands sent to the drive, as well as corrdinating the flow of traffic between the disk and the computer.

Q $067) How do you do a hard reset on a Plus/4 ?

A $067) First, we need to define hard-reset. A reset differs from a power-cycle, since the latter does not retain the RAM contents. In this case, the answer is analogous to the RUN/STOP-RESTORE combination found on the 64 and VIC-20. Hold down RUN/STOP and CTRL and press the recessed reset button on the side of the computer. I believe this works for the C-16 as well.

Q $068) Where did the name "Commodore" come from?

A $068) Rumor has it that Jack Tramiel always wante to use a naughtical term, but most had been already used. However, one day he watched a moving company van pass by on the street with the name he decided to use as soon as he saw it: Commodore.

Q $069) Chuck Peddle, designer of the 6502, left Commodore twice. Where did he go first?

A $069) He went to Apple Computer. He stayed with them briefly, but it seems that Apple and Chuck got along even worse than Commodore and Chuck.

Q $06A) Where did he eventually go when he left for good?

A $06A) First, he went off to start a company called Sirius, which died almost before it started due to a lawsuit over the name. Then, he and some former Commodore designers came up with the "Victor" computer, which did modestly, but never took off.

Q $06B) What does the Kernal routine at $FFD2 do in terms of function and what parameters get passed and returned?

A $06B) The KERNAL routine at $FFD2 on all Commodore 8 bit machines outputs the PETSCII character code contained in the .A register to the current output device. The carry flag indicates the presence of an error on return.

Q $06C) What Commodore drive has a hidden message?

A $06C) The 1581 has a couple such hidden messages. In the idle loop of the IP, the text says "am i lazy???...no just wanted to save a few ms...". Also, in the same loop, the following can be found: "this is lazy!!!". Lastly, the credits in the 1581 roms are: "Software david siracusa. hardware greg berliNZDedicatedto my wife lisA". (Note: the N in berliN and the A in lisA is typical of how strings are stored in the 1581, last byte has bit 7 set. The Z after berliN appears to have been a typo, but I can't say for sure. I have a program that displays these. (Email me for info.)

The 1571 has the ROM authors' names hidden at the beginning of the ROM, but I don't have a 1571 to scan for them.

Q $06D) What computer was the first to have a hidden message?

A $06D) The PET 2001. Some said the 128 has a hidden message, but it wasn't the first.

Q $06E) What was it and how did you get it to come up?

A $06E) By typing:

```
    wait 6502,x  (where x was a number between 1 and 255)
    the computer printed Microsoft! x times on the screen.
```

```
Q $06F) What does NTSC stand for?

A $06F) Truthfully, NTSC can stand for different things.  In regards to the
        television standard for the US, the expansion is National Television
        Standard Code.  However, the body that formed the standard is also
        called NTSC: National Television System Committee.


Commodore Trivia Edition #8


Q $070) On a PET series computer, what visual power-on indication will tell
        the user whether the computer has Revision 2 or Revision 3 ROMs?

Q $071) The IEEE-488 interface is sometimes called the GPIB interface.
        What does GPIB stand for?

Q $072) Commodore manufactured at least two hard drives with IEEE-488
        interfaces.  Can you name them?

Q $073) Why didn't buyers like the original PET-64?

Q $074) On a PET Revision 2 ROM, what was the largest single array size that
        BASIC could handle?

Q $075) On the stock 1541, data is transmitted one bit at a time.  How many
        bits are transferred at a time on the Commodore 1551 disk drive?

Q $076) On all Commodore floppy disk drives, how fast does the disk spin?

Q $077) Upon first reading the Commodore 1541 Error channel after turning
        on the disk drive, what error number and text is returned?

Q $078) What error number and text is returned on a 1551?

Q $079) Commodore printers are normally assigned to device #4, but they
        can be also used as device #?

Q $07A) What microprocessor is used in the Commodore 1551 disk drive?

Q $07B) When the VIC-20 was designed, the serial port throughput was roughly
        equivalent to the throughput of the IEEE-488 bus?  Why isn't it
        very fast in production VICs?

Q $07C) On Commodore computers, how much RAM is set aside as a tape buffer?

Q $07D) On Commodore computers, most every peripheral has a device number.
        What is the device number of the screen?

Q $07E) What is the device number of the keyboard?

Q $07F) Commodore computers use 2's-complement notation to represent integers.
        What is the 2's-complement hex representation of the signle byte -1?

Some are easy, some are hard, try your hand at:

      Commodore Trivia Edition #8!

Jim Brain
brain@mail.msen.com
2306 B Hartland Road
Hartland, MI  48353
(810) 737-7300 x8528

============================================================================
RS232 Converter
by Walter Wickersham (shadow@connected.com)

[Editor's note: I'm wary of there being no voltage translation but am including
it because I _do_ think you can get away with it... However, because this
magazine is free you get what you pay for... ]

Here's a modem interface schematic for the C=64/128, with it, and around
$5.00, you can use almost any hayes compat. external modem.  To the best of
my knowedge, the 64 has a maximum baud rate (through the user port) of 2400,
and the 128's is 9600.

I DO NOT know who the original author of this is, but i re-wrote it in my
own words, hoping it will help someone. I CLAIM NO RIGHTS TO THIS ARTICLE.
```

```
PARTS LIST:
-------------
7404 Hex Inverter IC ($0.99 at Radio Shack)
Wires, solder, etc.
Commodore User port connector (I used one off a old 1650)

Here It is:
C64/128 USER PORT          RS232 ADAPTER               RS232C

A & N -----------------------GROUND---------------------- 1 & 7
B & C ---------------------2-7404
                             7404-1---------------------- 3
M -------------------------3-7404
                             7404-4---------------------- 2
H-------------------------------------------------------- 8
E-------------------------------------------------------- 20
K ------------------------------------------------------- 5
L ------------------------------------------------------- 6

Pin #2n the user port MUST be connected to pin 14 of the 7404.
Pins A&N (ground) MUST be connected to pin 7 of the 7404.

For those of you who don't have a pinout of the user port, here, have one.
         (TOP)
 1-2-3-4-5-6-7-8-9-10-11-12
 --------------------------
 A-B-C-D-E-F-G-H-I-J--K--L-
        (BOTTOM)

THIS DOES WORK, that's why i'm modeming at 2400. :->, but i sometimes
recieve line noise, so any upgrades to this would be appreciated (i know
it's not my phone line).

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

Programming the Commodore RAM Expansion Units (REUs)

by Richard Hable (Richard.Hable@jk.uni-linz.ac.at)

The following article, initially written for a mailing list, describes
the Commodore REUs and explanes how to program them.

Contents:

1) _External RAM Access With REUs_

The REUs provide additional RAM for the C64/128. Three types of REUs have
been produced by Commodore. These are the 1700, 1764 and 1750 with 128, 256
and 512 KBytes built in RAM. However, they can be extended up to several
MBytes.

The external memory can not be directly addressed by the C64 with its 16 bit
address space--it has to be transferred from and to the main memory of the
C64. For that purpose, there is a built in RAM Expansion Controller (REC)
which transfers memory between the C64 and the REU using Direct Memory Access
(DMA). It can also be used for other purposes.

2) _RAM Expansion Controller (REC) Registers_

The REC is programmed by accessing its registers. When a REU is connected
through the expansion port, these registers appear memory mapped in the
I/O-area between $DF00 and $DF0A. They can be read and written to like VIC-
and SID-registers.

```
$DF00: STATUS REGISTER
    Various information can be obtained (read only).

  Bit 7:     INTERRUPT PENDING  (1 = interrupt waiting to be served)
               unnecessary
  Bit 6:     END OF BLOCK  (1 = transfer complete)
```

```
                unnecessary
  Bit 5:     FAULT  (1 = block verify error)
               set if a difference between C64 and REU memory areas
               was found during a compare command
  Bit 4:     SIZE  (1 = 256 KB)
               seems to indicate the size of the RAM-chips;
               set on 1764 and 1750, clear on 1700.
  Bits 3..0: VERSION
               contains 0 on my REU.
```

$DF01: COMMAND REGISTER
By writing to this register, RAM transfer or comparision can be executed.

```
  Bit 7:     EXECUTE  (1 = transfer per current configuration)
               must be set to execute a command
  Bit 6:     reserved  (normally 0)
  Bit 5:     LOAD  (1 = enable autoload option)
               With autoload enabled, the address and length registers (see
               below) will be unchanged after a command execution.
               Otherwise, the address registers will be counted up to the
               address of the last accessed byte of a DMA + 1
               and the length register will be changed (normally to 1).
  Bit 4:     FF00
               If this bit is set, command execution starts immediately
               after setting the command register.
               Otherwise, command execution is delayed until write access to
               memory position $FF00.
  Bits 3..2: reserved  (normally 0)
  Bits 1..0: TRANSFER TYPE
               00 = transfer C64 -> REU
               01 = transfer REU -> C64
               10 = swap C64 <-> REU
               11 = compare C64 - REU
```

$DF02..$DF03: C64 BASE ADDRESS
16-bit C64 base address in low/high order

$DF04..$DF06: REU BASE ADDRESS
This is a three byte address, consisting of a low and high byte and an expansion bank number.
Normally, only bits 2..0 of the expansion bank are valid (for a maximum of 512 KByte), the other bits are always set.

$DF07..$DF08: TRANSFER LENGTH
This is a 16 bit value containing the number of bytes to transfer or compare.
The value 0 stands for 64 KBytes.
If the transfer length plus the C64 base address exceeds 64K, the C64 address will overflow and cause C64 memory from 0 on to be accessed.
If the transfer length plus the REU base address exceeds 512K, the REU address will overflow and cause REU memory from 0 on to be accessed.

$DF09: INTERRUPT MASK REGISTER
unnecessary

```
  Bit 7:     INTERRUPT ENABLE  (1 = interrupt enabled)
  Bit 6:     END OF BLOCK MASK  (1 = interrupt on end)
  Bit 5:     VERIFY ERROR  (1 = interrupt on verify error)
  Bits 4..0: unused (normally all set)
```

$DF0A: ADDRESS CONTROL REGISTER
With this register, address counting during DMA can be controlled.
If a base address is fixed, the same byte is used repeatedly.

```
  Bit 7:     C64 ADDRESS CONTROL  (1 = fix C64 address)
  Bit 6:     REU ADDRESS CONTROL  (1 = fix REU address)
  Bits 5..0: unused (normally all set)
```

In order to access the REU registers in assembly language, it is convenient to define labels something like this:

```
  status   = $DF00
  command  = $DF01
  c64base  = $DF02
  reubase  = $DF04
  translen = $DF07
  irqmask  = $DF09
  control  = $DF0A
```

3) _How To Recognize The REU_

Normally, the addresses between $DF02 and $DF05 are unused, values stored there get lost. Therefore, if e.g. the values 1,2,3,4 are written to $DF02..$DF05 and do not stay there, no REU can be connected. However, if the values are there, it could be caused by another kind of module connected that also uses these addresses.

Another problem is the recognition of the number of RAM banks (64 KByte units) installed. The SIZE bit only tells if there are at least 2 (1700) or 4 (1764, 1750) banks installed. By trying to access and verify bytes in as many RAM banks as possible, the real size can be determined. This can be seen in the source to "Dynamic memory allocation for the 128" in Commodore Hacking Issue 2.

In any way, the user of a program should be able to choose, if and which REU banks are to be used.

4) _Simple RAM Transfer_

Very little options of the REU are necessary for the main purposes of RAM expanding. Just set the base addresses, transfer length, and then the command register.

The following code transfers one KByte containing the screen memory ($0400..$07FF) to address 0 in the REU:

```
  lda #0
  sta control ; to make sure both addresses are counted up
  lda #<$0400
  sta c64base
  lda #>$0400
  sta c64base + 1
  lda #0
  sta reubase
  sta reubase + 1
  sta reubase + 2
  lda #<$0400
  sta translen
  lda #>$0400
  sta translen + 1
  lda #%10010000;  c64 -> REU with immediate execution
  sta command
```

In order to transfer the memory back to the C64, replace "lda #%10010000" by "lda #%10010001".

I think, this subset of 17xx functions would be enough for a reasonable RAM expansion. However, if full compatibility with 17xx REUs is desired, also the more complicated functions have to be implemented.

5) _Additional Features_

Swapping Memory

With the swap-command, memory between 17xx and C64 can be exchanged. The programming is the same as in simple RAM transfer.

Comparing Memory

No RAM is transferred. Instead, the number of bytes specified in the transfer length register is compared. If there are differences, the FAULT bit of the status register is set. In order to get valid information, this bit has to be cleared before comparing. This is possible by reading the status register.

Using All C64 Memory

Normally, C64 memory is accessed in the memory configuration selected during writing to the command register. In order to be able to write to the command register, the I/O-area has to be active. If RAM between $D000 and $DFFF or character ROM shall be used, it is possible to delay the execution of the command by using a command byte with bit 4 ("FF00") cleared. The command will then be executed when an arbitrary value is written to address $FF00.

Example:

```
  < Set base addresses and transfer length >
  lda #%10000000 ; transfer C64 RAM -> REU delayed
  sta command
  sei
  lda $01
  and #$30
  sta $01 ; switch on 64 KByte RAM
  lda $FF00 ; do not change the contents of $FF00
  sta $FF00 ; execute DMA
  lda $01
  ora #$37
  sta $01 ; switch on normal configuration
  cli
```

6) _Transfer Speed_

During DMA the CPU is halted--the memory access cycles normally available for the CPU are now used to access one byte each cycle. Therefore, with screen and sprites switched off, in every clock cycle (985248 per second on PAL machines) one byte is transferred. If screen is on or sprites are enabled, transfer is a bit slower, as the VIC sometimes accesses RAM exclusively. Comparing memory areas is as fast as transfering. (Comparison is stopped once the first difference is found.) Swapping memory is only half as fast, because two C64 memory accesses per byte (read & write) are necessary.

7) _Interrupts_

By setting certain bits in the interrupt mask register, IRQs at the end of a DMA can be selected. However, as the CPU is halted during DMA, a transfer or comparision will always be finished after the store instruction into the command register or $FF00. Therefore, there is no need to check for an "END OF BLOCK" (bit 6 of status register) or to enable an interrupt.

8) _Executing Code In Expanded Memory_

Code in expanded memory has to be copied into C64 memory before execution. This is a disadvantage against bank switching systems. However, bank switching can be simulated by the SWAP command. This is done e.g. in RAMDOS. There, only 256 bytes of C64 memory are occupied, the 8 KByte RAM disk driver is swapped in whenever needed. Too much swapping is one reason for RAMDOS to be relatively slow at sequential file access.

9) _Other Useful Applications Of The REU_

The REC is not only useful for RAM transfer and comparison.

One other application (used in GEOS) is copying C64 RAM areas by first transferring them to the REU and then transferring them back into the desired position in C64 memory. Due to the fast DMA, this is about 5 times faster than copying memory with machine language instructions.

Interesting things can be done by fixing base addresses: By fixing the REU base address, large C64 areas can be fast filled with a single byte value. It is also possible to find the end of an area containing equal bytes very fast, e.g. for data compression.

Fixing the C64 base address is interesting if it points to an I/O-port. Then, data can be written out faster than normally possible. It would be possible to use real bitmap graphics in the upper and lower screen border by changing the "magic byte" (byte with the highest address accessable by the VIC) in every clock cycle. Therefore, of course, the vertical border would have to be switched off.

Generally the REC could be used as graphics accelerator, e.g. to copy bitmap areas or other display data fast into the VIC-addressable 16 KByte area.

10) _Comparision Of Bank Switching and DMA_

When comparing bank switching and DMA for memory expansion, I think, DMA is the more comfortable method to program. It is also faster in most cases. The disadvantage of code execution not possible in external memory can be minimized by always copying only the necessary parts into C64 memory. Executing the code will then take much more time than copying it into C64 memory.

==

A Different Perspective: Three-Dimensional Graphics on the C64

by Stephen Judd (judd@merle.acns.nwu.edu) and
   George Taylor (yurik@io.org)

Introduction

We've all seen them: neat-looking three-dimensional graphics tumbling around on a computer. But how is it done? In particular, how would you do it on a Commodore-64? Nowadays the typical answer to the first question is "Just use the functions in 3dgrphcs.lib" (or "Beats me."). The answer to the second is either "Well an elite coder like me can't let secrets like that out" or else "What, you mean people still use those things?"

So this is a little article which attempts to take some of the mystery out of three dimensional graphics. Most of the mathematics involved are very simple, and the geometric concepts are very intuitive. Coding it up on a C-64 is more of a challenge, especially when you want to make it fast, but even then it's not too tough. George and I wrote the code in about a week (and talked about it for about a week before that). Perhaps you will appreciate this aspect more if you know that I haven't written 6510 code since 1988, and until the last two days George had no computer on which to test his ideas (and on the last day it died)!

The goal of this article is that by the time you reach the end of it you will be able to do your own cool-looking 3d graphics on a C64. Some of you may find it patronizing at times, but I hope that it is at a level that everyone can enjoy and learn something from. And feel free to write to us!

The first part explains some of the fundamental theoretical concepts involved. Mostly what is required is some geometric imagination, although you need to know a little trigonometry, as well as how to multiply two matrices together.

The second part deals with implementing the algorithms on a computer; since this is C=Hacking, it is a good assumption that the implementation is on the C-64! Most of the code is designed for speed, and lots of it is also designed so that it can be called from BASIC!

Finally, an example program which uses all of the techniques presented here is included, including source. The program rotates a cube in three dimensions.

By itself the code is not the fastest in the world; what is important here are the concepts. With a little fiddling, and maybe some loop unrolling, you can get these routines going quite fast; for instance, a 26 cycle line drawing routine is not hard at all using more sophisticated versions of these algorithms. This time around the code is designed for clarity over quality.

There are lots and lots of little details that are not specifically covered by this article. But if you understand all of the concepts here it shouldn't be too hard to figure out the problem when something goes wrong.

This material is the result of a week's worth of discussions on comp.sys.cbm between George, myself, and several other people. So a big thank you to everyone who helped us to knock these ideas out, and we hope you find this to be a useful reference!

Incidentally, the ideas and techniques in this article aren't just for drawing neat pictures; for example, a good application is the stabilization of an orbiting satellite. The mathematical ideas behind linear transformations are important in, for instance, the study of dynamical systems (which leads to Chaos and all sorts of advanced mathematical subjects).

But it also makes you look really cool in front of your friends.

First Things First

Before we begin, you are going to have to get a few ideas into your head. First and foremost is the coordinate system we will be using: a right-handed coordinate system. In our system, the x-axis is going to come out towards you, the y-axis is going to go to your right, and the z-axis is going to go "up".

Second, you need to know a little math. You need to know about polar coordinates, and you need to know how to multiply two matrices together. The ideas are all geometric, but the computations are all (of course) mathematical.

Now, let us start thinking about a cube!

Let's first center our cube at the origin. Not only does this make it easy to visualize, but to make our cube do things (like rotate) the way we want it to we are going to have to require this. A cube has eight corners, all connected to each other in a particular way.

There's no reason to make things complicated already, so let's put the corners of the cube at x=+/-1, y=+/-1, and z=+/-1. This gives us eight points to work with: P1=[1 1 1] P2=[1 -1 1] P3=[-1 -1 1] etc.

Minimalists may disagree, but a cube all by itself isn't all that exciting. So how do we do stuff to it? For that matter, what kinds of stuff can we do to it?

Rotations in the Plane

One of the cool things to do with a three-dimensional object is of course to rotate it. To understand how to do this, we need to first look at rotations in the plane. A little later on, this article is going to assume you know how to multiply two matrices together.

Before starting, we need to know some important trig formulas (of course, everyone knows important formulas like these, but let me just remind you of them):

$$\cos(A+B) = \cos(A)\cos(B) - \sin(A)\sin(B)$$
$$\sin(A+B) = \cos(A)\sin(B) + \sin(A)\cos(B)$$

Let us take a look at rotations in the plane; that is, in two dimensions. Think of the typical x-y axis. Let's say that we have a point at [x1 y1] and want to rotate it by an angle B, about the origin, so that we end up at the rotated coordinate [x2 y2]. What are x2 and y2? The easiest way to find them is to use polar coordinates.

We can write the point [x1 y1] as the point (r,t), where r is the distance from the origin to the point, and t is the angle from the x-axis, measured counter-clockwise. Therefore, x1 = r*cos(t) and y1=r*sin(t). If we then rotate this vector by an amount B,

$$\begin{aligned} x2 &= r\cos(t+B) \\ &= r(\cos(t)\cos(B) - \sin(t)\sin(B)) \\ &= x1\cos(B) - y1\sin(B). \end{aligned}$$

Similarly,

$$y2 = r\sin(t+B) = x1\sin(B) + y1\cos(B).$$

In matrix form, this can be written as

$$\begin{bmatrix} x2 \\ y2 \end{bmatrix} = \begin{bmatrix} \cos(B) & -\sin(B) \\ \sin(B) & \cos(B) \end{bmatrix} \begin{bmatrix} x1 \\ y1 \end{bmatrix}$$

How do we extend this to three dimensions? Easy. The key thing to realize here is that, in three dimensions, the above rotations are really rotations about the z-axis. At any point along the z-axis we could take a thin slice of the three-dimensional space (so that our slice is parallel to the x-y axis) and pretend that we are really in two-dimensional space. Therefore, to rotate a point about the z-axis the x- and y-equations are the same as above, and the z-coordinate stays fixed. In matrix form this is

$$\begin{bmatrix} x2 \\ y2 \\ z2 \end{bmatrix} = \begin{bmatrix} \cos(B) & -\sin(B) & 0 \\ \sin(B) & \cos(B) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ y1 \\ z1 \end{bmatrix}$$

Similarly, it is easy to see that

$$\begin{bmatrix} x2 \\ y2 \\ z2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(B) & -\sin(B) \\ 0 & \sin(B) & \cos(B) \end{bmatrix} \begin{bmatrix} x1 \\ y1 \\ z1 \end{bmatrix}$$

is a rotation about the x-axis, and that

$$\begin{bmatrix} x2 \\ y2 \\ z2 \end{bmatrix} = \begin{bmatrix} \cos(B) & 0 & \sin(B) \\ 0 & 1 & 0 \\ -\sin(B) & 0 & \cos(B) \end{bmatrix} \begin{bmatrix} x1 \\ y1 \\ z1 \end{bmatrix}$$

is a rotation about the y-axis. You may have noticed that the signs of sin(B) have been reversed; this is because in our right-handed coordinate system the z-x plane is "backwards": in the z-x plane x increases to the left, while z increases "up".

You may be wondering why we write this all in matrix form. The above matrix equations are called linear transformations of the vector [x1 y1 z1]. There are lots of deep mathematical concepts sitting right behind what looks to be an easy way of writing several equations. Entire books are written on the subject, and that is as good a reason as any for me not to go into detail.

But writing things in this way also offers us several _computational_ advantages. Rotations aren't the only linear transformations; let's say that I want to rotate a point about the x-axis, shear it in the y-direction, reflect it through the line theta=pi/5, and rotate it through the z-axis. You could have one subroutine which did the rotation, and one that did the shear, etc. But by writing it in matrix form, the entire process is simply a series of matrix multiplications.

If you think about it you might realize that it really is the same thing no matter which way you do it, but there is a fundamental difference in the viewpoint of each method: one views it as a series of unrelated mathematical operations each with it's own individual function, while the other method views it as a series of matrix multiplications so that it's basically the same thing, over and over.

What this means for you is that if you want to rotate a point around the x-axis, the y-axis, and the z-axis, you can take the matrix for each transformation and multiply them all together, and then apply this one big matrix to the point. One thing to be very aware of: in general, matrix multiplication is not commutative. That is, if X is a rotation matrix about the x-axis and Y is a rotation about the y-axis, it will almost never be true that XY = YX. What this means geometrically is that if you take a point, rotate it around the x-axis by an amount A, then rotate it around the y-axis by an amount B, you will usually end up at a different point than if you first rotate it around the y-axis.

If you are interested in learning more about rotations and their uses, a good place to start is almost any book on mechanics, for instance "Classical Mechanics" by Goldstein. If you want to learn more about linear transformations you can find it in any decent book on linear algebra, as well as in a lot of physics texts. There is a good introduction in Margenau and Murphy "The Mathematics of Physics and Chemistry", and there is a semi-OK book on linear algebra by Goldberg.

Now we know the geometric and mathematical principles behind rotations in three dimensions. But we want to visualize this on a computer, on a two-dimensional screen: we need some way of taking a point in three-dimensions and bringing it down to two dimensions, but in a way that fools us into thinking that it really is three-dimensional.

What we need are projections.

Projections

Now, we could just do a simple projection and set the z-coordinate equal to zero, but in doing so we have eliminated some of the information, and it won't look very three-dimensional to our eyes. So we need to think of a better method.

Sit back in your chair and imagine for a minute or two. Imagine the three coordinate axes. Now imagine that there is a pinhole camera, with it's pinhole lens at the origin, and it's film at the plane at z=1 parallel to the x-y plane. Now we are going to take a snapshot of something.

Maybe a little picture would help:

```
                  |
                  |
                 /|
           lens / |film
          -----*--|------------ z-axis
              /   |
             /    |
            /    z=1
   object :-)                          (then again, maybe it won't!)
```

        What does this object look like on the film?

Let's say one of the points of this something is [x y z]. Where does this point come out on the film? Since the lens is at the origin, we want to draw the line from [x y z] through the origin (since that's where our lens is) and find the point [x1 y1 1] where it hits the film. The parametric equation of this line is

```
                t * [x y z]
```

so that we want to find the intersection of this line and the film:

```
                t * [x y z] = [x1 y1 1].
```

The z-coordinate tells us that t*z=1, or t=1/z. If we then substitute this in the above equation, we find that

```
                x1 = x/z        y1 = y/z
```

If, instead of placing the film at z=1 we place it at z=d, we get

```
                x1 = d*x/z      y1 = d*y/z
```

These then are the projection equations. Geometrically you can see that by changing d all you will do is to "magnify" the object on the film. Anyone who has watched an eclipse with a little pinhole camera has seen this.

By the way, if you stare at the above picture for a while, you may realize that, in that geometric model, the object gets turned upside-down on the film.

Now that we have a physical model of the equations that have been thrown around, let's look at what we've been doing.

Consider a cube centered at the origin. Already there is a problem above if z=0. What if one side of the cube has part of it's face below the x-y plane (negative z) and part above the x-y plane? If you draw another picture and trace rays through the origin, you'll see one part of the face at one end of the film (negative z, say), and the other part way the heck out at the other end! And the two parts don't touch, either!

So we need to be careful. In the geometric picture above, we assumed the object was a fair distance away from the lens. Currently we have our lens at the center of our cube, so something needs to move! Since rotations are defined about the origin we can't just redefine the cube so that the vertices are out at, say, z=2 and z=3. So what we need to do is to move the camera away from the cube. Or, if you want to think of it another way, we need to move the cube away from the camera before we take a picture of it.

In this case the translation needs to be done in the z-direction. The new projection equations are then

```
                x1 = d*x/(z-z0) y1 = d*y/(z-z0)
```

Where z0 is a translation amount that at the very least makes sure that z-z0 < 0.

Now not only have we eliminated possible problems with dividing by zero, but the mathematics now match the physical model.

Some of you might want to think about the less-physical situation of putting the object _behind_ the film, i.e. to the right of the film in the above picture.

As usual, there are some deeper mathematics lurking behind these equations, called projective geometry. Walter Taylor has written a book with a fine introduction to the subject (at least, I think the book was published; my copy is an old preprint).

Implementation

Now that we've got the theory under our belt, we need to think about implementing it on the computer. As a concession to all the programmers who immediately skipped to this section, most of the discussion will be at a reasonably high level.

One thing you need to understand is 8-bit signed and unsigned numbers. Here is a quick review: an 8-bit unsigned number ranges from 0..255. An 8-bit signed number ranges from -128..127 and is written in two's-complement form. In an 8-bit two's-complement number bits zero through six work like they usually do, but the seventh (high) bit represents the sign of the number in a special way. To find the 8-bit two's-complement of a number subtract it from 256. Example: what is -21 in two's complement notation? It is 256-21 = 235 = $EB. What is the complement of -21? It is 256-235 = 21 -- like magic. Another way to think about it is like a tape counter: 2 is $02, 1 is $01, 0 is $00, -1 is $FF, -2 is $FE, etc. And what is 24-21 in two's complement? It is: 24 + -21 = $EE + $EB = $0103. Throw away the carry (subtract 256) and we come out with... $03!

Onwards!

First, we need to decide what language to use. You and I both know the answer here: BASIC! Or maybe not. We need speed here, and speed on a Commodore 64 is spelled a-s-s-e-m-b-l-y.

Next, we need to decide what kind of math we want to use, signed or unsigned. Since the cosines and sines are going to generate negative and positive numbers in funny ways, we definitely want to use signed numbers. The alternative is to have lots of code and overhead to handle all the cases, and if we put it in two's-complement form the computer does most of the work for us.

How big should the numbers be? Since we are going for speed here, the obvious choice is 8-bits. But this restricts us to numbers between -128..127, is that OK? The size of our grid is 0..127 x 0..127, so this is perfect! But it does mean that we need to be very careful. For instance, consider the expression (a+b)/2. What happens if a=b=64? These are two numbers within our range of numbers, and the expression evaluates to 64, which is also in our range, BUT: if you evaluate the above in two's complement form, you will find different answers depending on how you evaluate it (i.e. (a+b)/2 will not give the same answer as a/2 + b/2, which will give the correct answer).

Now we've got another problem: sine and cosine range between negative one and one. To represent these floating point numbers as 8-bit signed integers the idea will be to multiply all floating point numbers by a fixed amount. That is, instead of dealing with the number 0.2, we use the number 64*0.2 = 12.8 = 13, and divide the end result by 64. As usual, we are trading accuracy for speed, although it will turn out to make little difference here.

Why did I pick 64? Obviously we want to pick some factor of two to make the division at the end simple (just an LSR). 128 is too big. 32 doesn't give us much accuracy. We also have to consider problems in expression evaluation (see the above example of (a+b)/2), but as we shall see 64 will work out nicely.

Now that we have accomplished the difficult task of decision making, we now need to move on to the simple task of implementation, starting with rotations.

Implementation: Rotations

We've got some more heavy-duty decision making ahead of us. We could implement this is several ways. We could apply each rotation individually, that is, we could rotate around the z-axis, then use these rotated points and rotate them around the y-axis, etc.

Well, yes, that would work, but... each rotation is nine multiplications. Each multiplication involves a lot of work, plus we have to shift the result by our fixed amount each time. We would not only be using huge amounts of time, but we would lose a lot of accuracy in the process. Computationally speaking, this is called a "bad idea".

Once again, mathematics saves the day: here is where we get the payoff for writing the equations as an algebraic system (a matrix). If X is the transformation around the x-axis, Y the transformation around the y-axis, and Z the transformation around the z-axis, then this is the equation to transform a vector v by rotating the point first around the z-axis, then the y-axis, then the x-axis:

        XYZv = v'

where v' is the new point after all the rotation transformations. (You might call it a conflagration of rotation transformations). Now the magic of linear algebra begins to work: operations are associative, which is a fancy way of saying that (AB)C = A(BC); For us this means that I can multiply all three matrices X Y and Z together to get a single new matrix M:

        M = XYZ
        Mv= v'

"But," you may say, "we have to do the same number of multiplications to get M as we do to apply each rotation separately! How is this supposed to help?" This is how it is supposed to help:

1) We now have a single matrix which describes ALL the rotations. For a single point we haven't gained much, but if we have a lot of points (and a cube has eight), transforming every point is now a single matrix multiplication. In other words, if we have a lot of points to transform we get a HUGE savings computationally.

2) We can take advantage of trigonometric identities and in so doing make the computation of M very simple.

Computationally speaking, this is known as a "good idea".

To multiply the three rotation matrices together, we need to take advantage of a few trigonometric properties. We need the two identites mentioned earlier:

```
        sin(a+b) = sin(a)cos(b) + cos(a)sin(b)
        cos(a+b) = cos(a)cos(b) - sin(a)sin(b)
```

We will also use the fact that cosine is even and sine is odd, that is

```
        cos(-a) = cos(a)
        sin(-a) = -sin(a)
```

Using the above identities it is easy to see that

```
        sin(a)sin(b) = (cos(a-b) - cos(a+b))/2
        cos(a)cos(b) = (cos(a+b) + cos(a-b))/2
        sin(a)cos(b) = (sin(a+b) + sin(a-b))/2
```

We are going to rotate first around the z-axis by an amount sz, then the y-axis by an amount sy, then the x-axis by an amount sx. Why rotate in that order? Why not.

```
        M = XYZ
```

If you multiply everything out (and I encourage you to do so, not only for practice, but also as a double-check of my work), and use the above trig identities, the result is:

```
            [A B C]
        M = [D E F]
            [G H I]
```

Where

```
        A = (cos(t1)+cos(t2))/2
        B = (sin(t1)-sin(t2))/2
        C = sin(sy)
        D = (sin(t3)-sin(t4))/2 + (cos(t6)-cos(t5)+cos(t8)-cos(t7))/4
        E = (cos(t3)+cos(t4))/2 + (sin(t5)-sin(t6)-sin(t7)-sin(t8))/4
        F = (sin(t9)-sin(t10))/2
        G = (cos(t4)-cos(t3))/2 + (sin(t6)-sin(t5)-sin(t8)-sin(t7))/4
        H = (sin(t3)+sin(t4))/2 + (cos(t6)-cos(t5)+cos(t7)-cos(t8))/4
        I = (cos(t9)+cos(t10))/2
```

with

```
        t1 = sy-sz
        t2 = sy+sz
        t3 = sx+sz
        t4 = sx-sz
        t5 = sx+sy+sz = sx+t2
        t6 = sx-sy+sz = sx-t1
        t7 = sx+sy-sz = sx+t1
        t8 = sy+sz-sx = t2-sx
        t9 = sy-sx
        t10= sy+sx
```

How is this supposed to be the "simplified" version? If you look closely, there are no multiplies. We can calculate the entire rotation matrix M in about the same time as it would take to do two multiplications. This also means that the associated problem with multiplications, loss of accuracy, is now gone.

Here is also where we need to be extremely careful. The first entry in the matrix M is the example I gave earlier about evaluating signed numbers. How do we overcome this?

Easy! Notice in the matrix M that, apart from element C, every term is a sine or a cosine divided by two. This is the only part of the program which uses sines and cosines, so why not use the offset floating-point values divided by two? This will make more sense in a minute.

The question arises: the above is all well and good, but how do we take the sine of a number and make it fast? The answer of course is to use a table. We used a BASIC routine to calculate the table for us (and to store the numbers in two's-complement form). Calculate the sine and cosine of every angle you want ahead of time, and then just look up the number.

The tables contain the values of sine and cosine multiplied by 64 (our floating-point offset) and then divided by 2. Since the value is already divided by two, the above calculation becomes at the same time faster and safer: faster because I don't have to keep dividing by two, and safer because I don't have to worry so much about overflow. (It can still happen, but it won't if you're careful).

Here is an example of how to calculate elements A and B above:

```
        LDA sy
        SEC
        SBC sz
        ...
        STA t1          ;t1=sy-sz
        LDA sy
        CLC
        ADC sz
        ...
        STA t2          ;t2=sy+sz
        ...
        LDX t1
        LDA COS,t1      ;COS is a table of cosines*offset/2
        LDX t2
        CLC
        ADC COS,t2
        STA A           ;A=(cos(t1)+cos(t2))/2
        LDX t1
        LDA SIN,t1
        LDX t2
        SEC
        SBC SIN,t2
        STA B           ;B=(sin(t1)-sin(t2))/2
        ...             ;Result is offset by a certain amount
```

Note that the elements D E G and H involve a division by four, which means that the code does need to perform a division by two during the calculation of those elements.

That's all there is to calculating the rotation matrix. Next we have to actually rotate the points. We have another decision to make: do we take the rotated object and rotate it by a little amount, or do we take the original object and rotate it by a big amount? Because of the way we have set things up, the answer is clear: we want to increment the angle at each step, and rotate the original object by this large angle (besides, geometrically you can see that it will look much nicer this way).

        For a cube this is easy. The points are P1=[1 1 1] P2=[1 -1 1] P3=[-1 -1 1] P4=[-1 1 1] P5=[1 1 -1] P6=[1 -1 -1] P7=[-1 -1 -1] P8=[-1 1 -1]. This means that the rotations are just a series of additions and/or subtractions of A,B,C,...,I! The code implements this in a funny way, partly to make these procedures easy to see, but mostly to make debugging the code much easier. It is much faster to do each rotation separately, i.e.

```
:P1     LDA A
        ADC B
        ADC C
        STA P1.X
        ...

:P2     LDA A
        SBC B
        ADC C
        STA P2.X
        ...
:P3     LDA C
        SBC A
        SBC B
        STA P3.X
```

You get the idea. Of course, the code needs to remember that it is dealing with signed numbers, and to watch carry flags carefully (something the above fragment does not do).

Still worried about overflow? If you think about it geometrically, you will see that the maximum value any part of a rotated coordinate can have is sqrt(3). Since we have offset our numbers by 64, this means that, for instance, the maximum possible value for A+B+C is 64*sqrt(3) which is about 111 -- in range of a signed 8-bit number with a little cushion for additions. In other words, we ought to be safe from overflow.

So far we have managed to rotate all the coordinates -- a complicated series of matrix operations involving trigonometric functions -- by just using a bunch of additions and a bunch of table lookups! Not too bad! Now we just need to project the point.

Implementation: Projections

Recall that the projection equation is

```
                        x' = d*x/(z-z0)
                        y' = d*y/(z-z0)
```

It looks as if we have gone from a bunch of sneaky additions to multiplications and divisions! Yuck.

Well, wait a minute, maybe we can do something. How about using a table for 1/(z-z0), and then just use a multiply? Oh yeah, that's a really small number. As long as we're using a table, why not incorporate the d into it? Come to think of it, if the number weren't multiplied by the offset 64 it would be a pretty reasonable number!

So, what we want to do is to construct a table of numbers such that when the program calls

```
        LDX z
        LDA table,z
```

it gets the absolute (i.e. non-offset) value A=d/(z-z0). What if we want to change d? You could put a little piece of code into your program which multiplies by a number less than one, and let d represent the maximum value for d which makes the code work. But for the moment we won't bother with that -- one thing at a time!

Since z is a signed number, we ought to add 128 to it to convert it into an index. Does this have any meaning in two's-complement arithmetic? Yup. We also need to remember that floats are offset by 64, and that the highest value a signed number can have is 127.

Here is how the table is generated:

```
        10 bz=whatever
        20 d=45:z0=3:z=-128:dz=1
        30 for i=0 to 255
        40 q%=64*d/(64*z0-z):if q%>127 then q%=127
        50 poke bz+i,q%:z=z+dz
        60 next
```

Note that the offset chosen forces q% to always be positive. This fact can be made use of in the multiplication routine (but isn't in the source code).

You may have noticed that z0-z is used, and not z-z0 like in the projection equation. If you put on your geometric thinking cap for a moment, you will realize that the way the projection equations were set up causes the image to become inverted. To uninvert it, we need to multiply by negative one. So we just add that step into the table.

But we still need to do a multiplication!

Fast Signed 8-bit Multiply

A binary number looks like the following:

```
        P = 1*128 + 0*64 + 0*32 + 1*16 + 1*8 + 0*4 + 0*2 + 0*1
```

Therefore, if we want to multiply P by another number, 13 say, we find that

```
        13*P = 13*128 + 0*64 + 0*32 + 13*16 + ...
```

that is to say, if there is a one in bit position N, then the new number will have 13*2^N in it. So, to multiply two numbers we find out what bit positions are high, and then add the other number*2^N to the result. This doesn't seem too fast. Here is a trick: we can write 2^N as 256/2^(8-N).

So, let's say we want to multiply the number P by the number R. If P has a high bit in position N, we can start out with 256*R, and bit-shift it to the right 8-N times. Why in the world would we do this? Because we can _pipeline_ the process in a way somewhat similar to the way a Cray supercomputer multiplies two vectors together -- yes, I'm comparing your C-64 to a Cray! Watch:

```
*
* 8-bit multiply -> 16-bit result
*
* ACC*AUX -> [AUX,EXT]  lo,hi

MULT    LDA #$00
        LDY #$09
]LOOP   LSR
        ROR ACC
        BCC MULT2
        CLC
        ADC AUX
MULT2   DEY
        BNE ]LOOP
        STA EXT
```

Pretty slick. Now we need to modify it for signed numbers. All we need to do is check to see if the result is positive or negative. If it's positive, we check one number (they are either both positive or both negative), and if it's negative we fix them both to be positive, and use the above process. If the result is going to be negative, we need to find the negative number, make it positive, multiply the two numbers together, and make the final result negative (take the two's-complement of the result).

See the source code for an implementation of this.

Note that we could do a divide in a similar fashion, except shifting left instead of right. Since we don't need a divide routine for our calculations we don't need to worry about this.

Now we have all the tools we need to implement the mathematics. There is still one part of the program left: drawing the thing!

Drawing a line

The geometric idea is: given an initial point [x1 y1], we want to draw a line to the point [x2 y2]! Now we want to do this on a computer by taking one step at a time, from point to point. The idea is to make it fast, and since we're on a C64 there aren't any MUL or DIV instructions.

To do this, we first need to find out which is larger:

```
        dx = |x2-x1|
        dy = |y2-y1|
```

where | | denotes absolute value. Let's assume that it is dx, and that the variable x is going to run from x1 to x2. Therefore, we want to increase x by one at each step, and we want to increase y by some fractional amount (If dy were larger we would want to take big steps in the y-direction). But we don't want to calculate this fractional number. We do, however, want to take a certain amount of steps in the x-direction before taking a step in the y-direction.

If we take k steps in x before taking a step in y, then we want to chose k such that

```
        dx/k = dy
```

which gives

```
        k = dx/dy
```

where dx and dy are as above, the total number of steps to be taken in the x- and y-directions respectively. What is dx/dy? We don't care. Instead, every time we step in x, we need to increase a counter by the amount dy. As soon as this counter is larger than dx, we have successfully divided dy into dx, and so simply reset the counter (in a special way, so that we keep any remainder from the division) and take a step in y.

Of course, if dy were larger than dx, the idea would be the same, but now k = dy/dx. k is never smaller than one.

In the code fragment which follows it is assumed that x2>x1, y2>y1, and dx>dy. Obviously, then, any self-respecting line drawing routine needs to handle all of these cases. One way is to have eight different routines, one for each case. Another way (the way used by the program), is to force x2>x1, so that there are only four cases to deal with. For the plotting routine which we use, this turns out to be necessary. If you think about it, you can come up with some more clever ways to deal with this.

Note that you also need to figure out what column the first point is in: this algorithm knows how to walk forwards, but it doesn't know where it should start.

The code is next to some similar BASIC7.0 code to make it easier to understand.

The code can be sped up in a lot of ways. For one thing it could be made self-modifying. All variables could be stored in zero page. In fact, the entire routine could be stored in zero page! Also, with a little change in the logic (and a subsequent change in the plotting routine) you can eliminate the branching instruction. For the sake of clarity we don't do that here; maybe in another paper ;-).

Also note that the largest value x can take on in this routine is 255. For the way we are going to plot things, this won't matter. But a more general routine needs a way to overcome this. One way would be to draw two separate lines.

```
10 REM All of the above comments ;-)
20 REM Input x1,x2,y1,y2
30 GRAPHIC1,1:DRAW1,x1,y1:DRAW1,x2,y2
31 :REM above is a double-check          ;Drawin' a line
39 REM Set up variables                  ;v1.3 SLJ 7/2/94
40 DX = X2-X1                            LDA     $(X2)   ;X2 in zero page
                                         SBC     $(X1)
                                         STA     DX      ;For speed, store
50 DY = Y2-Y1                            LDA     $(Y2)   ;directly into code
                                         SBC     $(Y1)   ;below
                                         STY     DY
60 X=X1:Y=Y1                             LDX     $(X1)   ;Plotting coordinates
                                         LDY     $(Y1)   ;in X and Y
64 REM A counts steps in x
65 REM Below you might want to
66 REM change to A=1 or A=DY
67 REM Otherwise the line always
68 REM takes only one step in y
69 REM before the last point (x=x2-1)
70 A=256-DX:REM A=0                      LDA     #00     ;Saves us a CMP
                                         SEC
                                         SBC     DX
80 DRAW1,X,Y                             PPLOT           ;Mystery plotter
90 REM Main routine                      CLC
100 X=X+1                           LOOP INX             ;Step in x
110 A=A+DY                               ADC     DY      ;Add DY
120 IF A>=256 THEN Y=Y+1:A=A-DX          BCC     NOPE    ;Time to step in y?
121 REM IF A>=DX THEN...                 INY             ;Step in y
                                         SBC     DX      ;Reset counter
130 DRAW1,X,Y                       NOPE PPLOT           ;Plot the point
140 IF X<>X2 THEN GOTO 100               CPX     X2      ;At the endpoint yet?
                                         BNE     LOOP
150 PRINT"All done!":REM Yay!

                                         Cycle count:
                                               LOOP: 2 3 2 2 3 3 3 = 18
                                                     (worst case)
                                               + dx PPLOTs (one for each point)
```

The point here is that it's fast. If you use self-modifying code, you can get this down to 15 cycles per point. If you are clever, you can get it down to 13 cycles per point, excluding plot, worst case. Not too bad! We won't be clever right now, but maybe you'll get to see it later...

Note also that this could easily be used in a BASIC program; even a BASIC2.0 program. (If you would like the DATA statements to do this just drop us a line, er... contact us).

Now, this routine works fine, but for drawing a line on a computer it doesn't always look great. For instance, what happens if we draw a point from 1,1 to 11,3? k=dx/dy=5, so se will take five steps in x and then a step in y, then five more steps and... a step in y at the very last point! So our line doesn't look so good -- we have a little square edge at the endpoint.

One way to fix this is to trick the computer into thinking it needs to take an extra step in y by letting k=dx/(dy+1), and being careful in keeping track of our counter. The big problem with this method is that it produces the square end-pixels when dx and dy are nearly the same (slope ~= 1).

A better way to fix this is to initialize the counter not to 0 (in our case, 256-dx), but instead to DX/2 (256-DX/2 in our case). This has the effect of splitting one of the line segments between the two endpoints, and looks good for all slopes. This is what the program does. In fact, as far as I can tell, this is what BASIC7.0 does too!

There is still a part of our routine missing, however...

Plotting a point

In the line routine presented earlier, the nebulous statement PPLOT was written. Now we come to plotting a point in all its gory detail.

For this project, speed is the name of the game, and for speed we don't want to use normal bitmapped graphics. Instead, we want to use character graphics. The advantages of using a custom character set are:

- Less memory
- Speed of plotting
- Double buffering
- Convenient organization

The first advantage, less memory, should be clear. A custom character set takes up 2k. A bitmap, on the other hand, takes up 8k.

For the second advantage, it is much faster to poke a character into screen memory than it is to calculate and plot all 64 bits in a character. This way, VIC does all the hard work for us. Also, if we are clever, we can exploit several aspects of our cleverness to make plotting a single point much easier.

Character graphics also give us a very simple means of double buffering: we can just plot into two different character sets and tell VIC-II to move between them. No raster interrupts here! If the two character sets were at $3000 and $3800, here is how to switch between them:

```
        LDA VMCSB        ;VMCSB=$D018
        EOR #%00000010  ;Flip the bit
        STA VMCSB
```

True, clearing the buffer each time is a bit slow, but for our purposes it will do just fine.

The last is less obvious. A normal hires bitmap is organized like the following:

```
        00  08 ...
        01  09
        02  0A
        ... ...
        07  0F
```

where the number represents the offset of the byte. This is fine for some things, but calculating the position of a pixel is tricky. With a character map, we can represent our data any way we want. In particular, we can organize our bitmap to look like the following:

```
        00  80  ...
        01  81
        02  82
        ... ...
        7D  FD
        7E  FE
        7F  FF  ...
```

Or, in graphic form

```
        @P... etc.
        AQ
        BR
        CS
        ..
        O<-  (the back-arrow)
```

What we have done is, instead of putting characters side-by-side like a hires bitmap does, we put them on top of each other. The above represents a 16x16 character array, which is a 128x128 pixel array. Now the y-coordinate is a direct index into the row we are in. That is, base+Y = memory location of point.

This brings us to the primary disadvantage of using a character set: our pictures are pretty small. TANSSAAFL.

Now we could just go merrily plotting into our character bitmap, but as usual a little thought can yield some impressive return. The first thing to notice is that the maximum value for y is 127; the only thing that sets the high bit is the x-coordinate, and then only when it crosses a column (just look at the above memory map if you don't see it).

Therefore, if we could keep track of the bit position of x, we could tell when x crossed a column, and just add 128 to the base address. Not only that, but we also know to increase the high byte of the pointer by one when we have crossed two columns.

The logic is as follows:

- Find the bit pattern for a given x (for speed, use a table)
- If it is 10000000 then we have jumped a column
- If the column we are in doesn't have the high bit set in the low byte of the pointer to the base of the column, then set the high bit (add 128)
- Otherwise, set the high bit to zero (add 128), and increase the high byte of the column pointer (step into the next page).

Here is (more or less) the code:

In BASIC:
```
        2000 rem bp(x) contains bit position for x
        2010 if int(x/8) = x/8 then base=base+128
        2020 poke base+y, (peek(base+y) or bp(x))
```

In assembly:
```
        LDA     BITP,X  4       ;Load the bit pattern from a table
        BPL     CONT    3  2    ;Still in the same column?
        EOR     $LO        3    ;If not, add 128 to the low byte
        STA     $LO        3
        BMI     CONT       3  2 ;If the high bit is set, stay in the same page
        INC     $HI           5 ;Otherwise point to the next page
        LDA     #$128         2 ;We still need the bit pattern for x!
   CONT ORA     ($LO),Y 5
        STA     ($LO),Y 6       ;Plot the point
                        --------
            Cycle count: 18 26 32
```

Therefore, it takes 18 cycles to plot a point, 26 cycles to jump a column, and 32 cycles to jump a page. Over 16 points, this averages 19.375 cycles.

When combined with the earlier line drawing routine, this gives an average time of 38 cycles or so (with a best time of 34 cycles); six of those cycles are for PHA and PLA, since the line drawing routine uses A for other things.

Like most of the code, you can improve on this method if you think about it a little. Most of the time is spent checking the special cases, so how can you avoid them? Maybe if we do another article, we'll show you our solution(s).

Now, this method has a few subtleties about it. First, what happens if the first point to be plotted is x=0, or x=8? The above routine will increase the base pointer right off the bat. This case needs to be taken care of.

Second, the above assumes that you always take a step in x. What happens if we are taking a big step in y? Let's say that we take ten steps in y for every step in x. What will the above plotter do if x takes a step across a column, and then doesn't change for a while? Look to the source code for one solution to this problem.

So that's all there is to it!

Post Script

That's all there is to it. Well, OK, there are a few details we left out,

but you can figure them out on your own. You can always look to the source
code to see how we overcame the same problem. The program is set up in a
way that you can experiment around with the projection parameters d and z0,
to see what changing them does to the math.

What's next? In the future you will undoubtably see lots of things from
George and myself, both the written word and the coded byte. Maybe we will
see something from you as well?

Da Code

```
********************************
*..............................*
*.Stephen.Judd.................*
*.George.Taylor................*
*.Started:.7/11/94.............*
*.Finished:.7/19/94............*
*..............................*
*.Well,.if.all.goes.well.this..*
*.program.will.rotate.a.cube...*
*..............................*
*.This.program.is.intended.to..*
*.accompany.the.article.in.....*
*.C=Hacking,.July.94.issue.....*
*.For.details.on.this.program,.*
*.read.the.article!............*
*..............................*
*.Write.to.us!.................*
*..............................*
*.un(bee)mo....................*
*..............................*
*.vi...........................*
*.n(in)g.......................*
*.are(th.......................*
*.e)you(o......................*
*.nly).........................*
*..............................*
*.asl(rose)eep.................*
*.............e.e.cummings....*
*..............................*
*.P.S..This.was.written.using..*
*......Merlin.128...With.a.....*
*......little.modification.it..*
*......will.work.fine.with.....*
*......Merlin.64...If.you......*
*......don't.have.either.......*
*......well,.we.all.have.our...*
*......little.faults...........*
********************************

 ORG $1000

*.Constants

BUFF1 EQU $3000 ;First.character.set
BUFF2 EQU $3800 ;Second.character.set
BUFFER EQU $A3 ;Presumably.the.tape.won't.be.running
X1 EQU $FB ;Points.for.drawing.a.line
Y1 EQU $FC ;These.zero.page.addresses
X2 EQU $FD ;don't.conflict.with.BASIC
Y2 EQU $FE
DX EQU $F9
DY EQU $FA
TEMP1 EQU $FB ;Of.course,.could.conflict.with.x1
TEMP2 EQU $FC ;Temporary.variables
ACC EQU $FB ;These.four.variables.are.used
AUX EQU $FC ;by.the.multiplication.routine
EXT EQU $FD
REM EQU $FE
ZTEMP EQU $02 ;Used.for.buffer.swap...Don't.touch.

ANGMAX EQU 120 ;There.are.2*pi/angmax.angles
OFFSET EQU 6 ;Float.offset:.x=xactual*2^offset

*.VIC

VMCSB EQU $D018
BKGND EQU $D020
BORDER EQU $D021
SSTART EQU 1344 ;row.9.in.screen.memory.at.1024
```

```
*.Kernal

CHROUT EQU $FFD2
GETIN EQU $FFE4

***.Macros

MOVE MAC
 LDA ]1
 STA ]2
 <<<

GETKEY MAC  ;Wait.for.a.keypress
WAIT JSR GETIN
 CMP #00
 BEQ WAIT
 <<<

DEBUG MAC  ;Print.a.character
. DO.0  ;Don't.assemble

 LDA #]1
 JSR CHROUT
 >>> GETKEY ;And.wait.to.continue
 CMP #'s' ;My.secrect.switch.key
 BNE L1
 JSR CLEANUP
 JMP DONE
L1 CMP #'x' ;My.secret.abort.key
 BNE DONE
 JMP CLEANUP
 FIN
DONE <<<

DEBUGA MAC
 DO.0
 LDA ]1
 STA 1024
 FIN
DONEA <<<

SETBUF MAC  ;Put.buffers.where.they.can.be.hurt
 LDA #00
 STA BUFFER
 LDA ZTEMP ;ztemp.contains.the.high.byte.here
 STA BUFFER+1
 <<<

*-------------------------------

 LDA #$00
 STA BKGND
 STA BORDER
 LDA VMCSB
 AND #%00001111 ;Screen.memory.to.1024
 ORA #%00010000
 STA VMCSB

 LDY #00
 LDA #<TTEXT
 STA TEMP1
 LDA #>TTEXT
 STA TEMP2
 JMP TITLE
TTEXT HEX 9305111111 ;clear.screen,.white,.crsr.dn
 TXT '..............cube3d',0d,0d
 TXT '................by',0d
 HEX 9F ;cyan
 TXT '....stephen.judd'
 HEX 99
 TXT '....george.taylor',0d,0d
 HEX 9B
 TXT '..check.out.the.july.94.issue.of',0d
 HEX 96
 TXT '..c=hacking'
 HEX 9B
 TXT '.for.more.details!',0d
 HEX 0D1D1D9E12
 TXT 'f1/f2',92
 TXT '.-.inc/dec.x-rotation',0d
 HEX 1D1D12
 TXT 'f3/f4',92
 TXT '.-.inc/dec.y-rotation',0d
 HEX 1D1D12
 TXT 'f5/f6',92
 TXT '.-.inc/dec.z-rotation',0d
 HEX 1D1D12
 TXT 'f7',92
 TXT '.resets',0d
 TXT '..press.q.to.quit',0d
 HEX 0D05
 TXT '......press.any.key.to.begin',0d
 HEX 00
TITLE LDA (TEMP1),Y
 BEQ :CONT
 JSR CHROUT
 INY
 BNE TITLE
 INC TEMP2
 JMP TITLE
:CONT >>> GETKEY

****.Set.up.tables(?)

*.Tables.are.currently.set.up.in.BASIC
*.and.by.the.assembler.

TABLES

****.Clear.screen.and.set.up."bitmap"

SETUP LDA #147
 JSR CHROUT
 LDA #<SSTART
 ADC #12 ;The.goal.is.to.center.the.graphics
 STA TEMP1 ;Column.12
 LDA #>SSTART ;Row.9
 STA TEMP1+1 ;SSTART.points.to.row.9
 LDA #00
 LDY #00
 LDX #00 ;x.will.count.16.rows.for.us
 CLC

:LOOP STA (TEMP1),Y
 INY
 ADC #16
 BCC :LOOP
 CLC
 LDA TEMP1
 ADC #40 ;Need.to.add.40.to.the.base.pointer
 STA TEMP1 ;To.jump.to.the.next.row
 LDA TEMP1+1
 ADC #00 ;Take.care.of.carries
 STA TEMP1+1
 LDY #00
 INX
 TXA  ;X.is.also.an.index.into.the.character.number
 CPX #16
 BNE :LOOP ;Need.to.do.it.16.times

 >>> DEBUG,'2'
****.Set.up.buffers

 LDA #<BUFF1
 STA BUFFER
 LDA #>BUFF1
 STA BUFFER+1
 STA ZTEMP ;ztemp.will.make.life.simple.for.us
 LDA VMCSB
 AND #%11110001 ;Start.here.so.that.swap.buffers.will.work.right
 ORA #%00001110
 STA VMCSB


****.Set.up.initial.values

INIT LDA #00
 STA DSX
 STA DSY
 STA DSZ
 STA SX
 STA SY
 STA SZ

 >>> DEBUG,'4'

*-------------------------------
*.Main.loop

****.Get.keypress

MAIN
KPRESS JSR GETIN
 CMP #133 ;F1?
 BNE :F2
 LDA DSX
 CMP #ANGMAX/2 ;No.more.than.pi
 BEQ :CONT
 INC DSX ;otherwise.increase.x-rotation
 JMP :CONT
:F2 CMP #137 ;F2?
 BNE :F3
 LDA DSX
 BEQ :CONT
 DEC DSX
 JMP :CONT
:F3 CMP #134
 BNE :F4
 LDA DSY
 CMP #ANGMAX/2
 BEQ :CONT
 INC DSY ;Increase.y-rotation
 JMP :CONT
:F4 CMP #138
 BNE :F5
 LDA DSY
 BEQ :CONT
 DEC DSY
 JMP :CONT
:F5 CMP #135
 BNE :F6
 LDA DSZ
 CMP #ANGMAX/2
 BEQ :CONT
 INC DSZ ;z-rotation
 JMP :CONT
:F6 CMP #139
 BNE :F7
 LDA DSZ
 BEQ :CONT
 DEC DSZ
 JMP :CONT
:F7 CMP #136
 BNE :Q
 JMP INIT
:Q CMP #'q' ;q.quits
 BNE :CONT
 JMP CLEANUP

:CONT
*.>>>.DEBUG,'5'

****.Update.angles

UPDATE CLC
 LDA SX
 ADC DSX
 CMP #ANGMAX ;Are.we.>=.maximum.angle?
 BCC :CONT1
 SBC #ANGMAX :If so, reset
:CONT1 STA SX
 CLC
 LDA SY
 ADC DSY
 CMP #ANGMAX
 BCC :CONT2
 SBC #ANGMAX ;Same.deal
:CONT2 STA SY
 CLC
 LDA SZ
 ADC DSZ
 CMP #ANGMAX
 BCC :CONT3
 SBC #ANGMAX
:CONT3 STA SZ


****.Rotate.coordinates

ROTATE

***.First,.calculate.t1,t2,...,t10

**.Two.macros.to.simplify.our.life
ADDA MAC  ;Add.two.angles.together
 CLC
 LDA ]1
 ADC ]2
 CMP #ANGMAX ;Is.the.sum.>.2*pi?
 BCC DONE
 SBC #ANGMAX ;If.so,.subtract.2*pi
DONE <<<

SUBA MAC  ;Subtract.two.angles
 SEC
 LDA ]1
 SBC ]2
 BCS DONE
 ADC #ANGMAX ;Oops,.we.need.to.add.2*pi
DONE <<<

**.Now.calculate.t1,t2,etc.

 >>> SUBA,SY;SZ
 STA T1 ;t1=sy-sz
 >>> ADDA,SY;SZ
 STA T2 ;t2=sy+sz
 >>> ADDA,SX;SZ
 STA T3 ;t3=sx+sz
 >>> SUBA,SX;SZ
 STA T4 ;t4=sx-sz
 >>> ADDA,SX;T2
 STA T5 ;t5=sx+t2
 >>> SUBA,SX;T1
 STA T6 ;t6=sx-t1
 >>> ADDA,SX;T1
 STA T7 ;t7=sx+t1
 >>> SUBA,T2;SX
 STA T8 ;t8=t2-sx
 >>> SUBA,SY;SX
 STA T9 ;t9=sy-sx
 >>> ADDA,SX;SY
 STA T10 ;t10=sx+sy

*.Et.voila!

***.Next,.calculate.A,B,C,...,I

**.Another.useful.little.macro
DIV2 MAC  ;Divide.a.signed.number.by.2
;It.is.assumed.that.the.number
 BPL POS ;is.in.the.accumulator
 CLC
 EOR #$FF ;We.need.to.un-negative.the.number
 ADC #01 ;by.taking.it's.complement
 LSR  ;divide.by.two
 CLC
 EOR #$FF
 ADC #01 ;Make.it.negative.again
 JMP DONEDIV
POS LSR  ;Number.is.positive
DONEDIV <<<

MUL2 MAC  ;Multiply.a.signed.number.by.2
 BPL POSM
 CLC
 EOR #$FF
 ADC #$01
 ASL
 CLC
 EOR #$FF
 ADC #$01
 JMP DONEMUL
POSM ASL
DONEMUL <<<

**.Note.that.we.are.currently.making.a.minor.leap
**.of.faith.that.no.overflows.will.occur.

:CALCA CLC
 LDX T1
 LDA COS,X
 LDX T2
 ADC COS,X
 STA A11 ;A=(cos(t1)+cos(t2))/2
:CALCB LDX T1
 LDA SIN,X
 SEC
 LDX T2
 SBC SIN,X
 STA B12 ;B=(sin(t1)-sin(t2))/2
:CALCC LDX SY
 LDA SIN,X
 >>> MUL2
 STA C13 ;C=sin(sy)
:CALCD SEC
 LDX T8
 LDA COS,X
 LDX T7
 SBC COS,X
 SEC
 LDX T5
 SBC COS,X
 CLC
 LDX T6
 ADC COS,X ;Di=(cos(t8)-cos(t7)+cos(t6)-cos(t5))/2
 >>> DIV2
 CLC
 LDX T3
 ADC SIN,X
 SEC
 LDX T4
 SBC SIN,X
 STA D21 ;D=(sin(t3)-sin(t4)+Di)/2
:CALCE SEC
 LDX T5
 LDA SIN,X
 LDX T6
 SBC SIN,X
 SEC
 LDX T7
 SBC SIN,X
 SEC
 LDX T8
 SBC SIN,X ;Ei=(sin(t5)-sin(t6)-sin(t7)-sin(t8))/2
 >>> DIV2
 CLC
 LDX T3
 ADC COS,X
 CLC
 LDX T4
 ADC COS,X
 STA E22 ;E=(cos(t3)+cos(t4)+Ei)/2
:CALCF LDX T9
 LDA SIN,X
 SEC
 LDX T10
 SBC SIN,X
 STA F23 ;F=(sin(t9)-sin(t10))/2
:CALCG LDX T6
 LDA SIN,X
 SEC
 LDX T8
 SBC SIN,X
 SEC
 LDX T7
 SBC SIN,X
 SEC
 LDX T5
 SBC SIN,X ;Gi=(sin(t6)-sin(t8)-sin(t7)-sin(t5))/2
 >>> DIV2
 CLC
 LDX T4
 ADC COS,X
 SEC
 LDX T3
 SBC COS,X
 STA G31 ;G=(cos(t4)-cos(t3)+Gi)/2
 >>> DEBUGA,G31
 >>> DEBUG,'g'
:CALCH CLC
 LDX T6
 LDA COS,X
 LDX T7
 ADC COS,X
 SEC
 LDX T5
 SBC COS,X
 SEC
 LDX T8
 SBC COS,X ;Hi=(cos(t6)+cos(t7)-cos(t5)-cos(t8))/2
 >>> DIV2
 CLC
 LDX T3
 ADC SIN,X
 CLC
 LDX T4
 ADC SIN,X
 STA H32 ;H=(sin(t3)+sin(t4)+Hi)/2
:WHEW CLC
 LDX T9
 LDA COS,X
 LDX T10
 ADC COS,X
 STA I33 ;I=(cos(t9)+cos(t10))/2

**.It's.all.downhill.from.here.

**.Rotate,.project,.and.store.the.points
DOWNHILL LDA A11 ;This.is.getting.to.be.a.real.mess
 STA TA
 LDA B12 ;The.reason.this.is.done
 STA TB ;is.to.make.the.code.a.little
 LDA C13 ;easier.to.read.(and.debug!)
 STA TC
 LDA D21 ;These.are.all.temporary.locations
 STA TD ;Used.by.the.projection.subroutine.
 LDA E22
 STA TE ;Otherwise,.there.would.be.eight
 LDA F23 ;long.routines.here.
 STA TF
 LDA G31 ;But.it.would.be.significantly.faster
 STA TG
 LDA H32
 STA TH
 LDA I33
 STA TI

*.A.neat.macro
NEG MAC  ;Change.the.sign.of.a.two's.complement
 CLC
 LDA ]1 ;number.
 EOR #$FF
 ADC #$01
 <<<

*.P1=[1.1.1]
 JSR PROJECT ;Unroll.this.whole.thing
 LDX TX1 ;(sorry.about.these.two.lines)
 LDY TY1 ;(see.PROJECT.for.reason.why)
 STX P1X ;For.a.pretty.big.speed.increase!
 STY P1Y
*.P2=[1.-1.1]
 >>> NEG,B12 ;Change.these.elements
 STA TB
 >>> NEG,E22 ;Since.y.is.now.-1
 STA TE
 >>> NEG,H32
 STA TH
 JSR PROJECT
 LDX TX1
 LDY TY1
 STX P2X
 STY P2Y
```

```
*.P3=[-1.-1.1]
 >>> NEG,A11
 STA TA
 >>> NEG,D21
 STA TD
 >>> NEG,G31
 STA TG
 JSR PROJECT
 LDX TX1
 LDY TY1
 STX P3X
 STY P3Y
*.P4=[-1.1.1]
 LDA B12
 STA TB
 LDA E22
 STA TE
 LDA H32
 STA TH
 JSR PROJECT
 LDX TX1
 LDY TY1
 STX P4X
 STY P4Y
*.P8=[-1.1.-1]
 >>> NEG,C13
 STA TC
 >>> NEG,F23
 STA TF
 >>> NEG,I33
 STA TI
 JSR PROJECT
 LDX TX1
 LDY TY1
 STX P8X
 STY P8Y
*.P7=[-1.-1.-1]
 >>> NEG,B12
 STA TB
 >>> NEG,E22
 STA TE
 >>> NEG,H32
 STA TH
 JSR PROJECT
 LDX TX1
 LDY TY1
 STX P7X
 STY P7Y
*.P6=[1.-1.-1]
 LDA A11
 STA TA
 LDA D21
 STA TD
 LDA G31
 STA TG
 JSR PROJECT
 LDX TX1
 LDY TY1
 STX P6X
 STY P6Y
*.P5=[1.1.-1]
 LDA B12
 STA TB
 LDA E22
 STA TE
 LDA H32
 STA TH
 JSR PROJECT
 LDX TX1
 LDY TY1
 STX P5X
 STY P5Y

****.Clear.buffer

 >>> SETBUF
CLRBUF LDA #$00 ;Pretty.straightforward,
 LDX #$08 ;I.think
 LDY #$00
:LOOP STA (BUFFER),Y
 INY
 BNE :LOOP
 INC BUFFER+1
 DEX
 BNE :LOOP
 LDA BUFFER+1

****.Finally,.draw.the.lines.

 LDA P1X ;[1.1.1]
 STA TX1
 LDA P1Y
 STA TY1
 LDA P2X ;[1.-1.1]
 STA TX2
 LDA P2Y
 STA TY2
 JSR DRAW ;First.line

 LDA P3X ;[-1.-1.1]
 STA TX1
 LDA P3Y
 STA TY1
 JSR DRAW ;Second.line

 LDA P4X ;[-1.1.1]
 STA TX2
 LDA P4Y
 STA TY2
 JSR DRAW ;Third.line

 LDA P1X ;[1.1.1]
 STA TX1
 LDA P1Y
 STA TY1
 JSR DRAW ;Fourth.line...One.face.done.

 LDA P5X ;[1.1.-1]
 STA TX2
 LDA P5Y
 STA TY2
 JSR DRAW ;Five

 LDA P6X ;[1.-1.-1]
 STA TX1
 LDA P6Y
 STA TY1
 JSR DRAW ;Six

 LDA P2X ;[1.-1.1]
 STA TX2
 LDA P2Y
 STA TY2
 JSR DRAW ;Seven

 LDA P7X ;[-1.-1.-1]
 STA TX2
 LDA P7Y
 STA TY2
 JSR DRAW ;Eight

 LDA P3X ;[-1.-1.1]
 STA TX1
 LDA P3Y
 STA TY1
 JSR DRAW ;Nine

 LDA P8X ;[-1.1.-1]
 STA TX1
 LDA P8Y
 STA TY1
 JSR DRAW ;Ten

 LDA P4X ;[-1.1.1]
 STA TX2
 LDA P4Y
 STA TY2
 JSR DRAW ;Eleven

 LDA P5X ;[1.1.-1]
 STA TX2
 LDA P5Y
 STA TY2
 JSR DRAW ;Twelve!

****.Swap.buffers

SWAPBUF LDA VMCSB
 EOR #$02 ;Pretty.tricky,.eh?
 STA VMCSB
 LDA #$08
 EOR ZTEMP ;ztemp=high.byte.just.flips
 STA ZTEMP ;between.$30.and.$38

 JMP MAIN ;Around.and.around.we.go...


*-------------------------------
*.This.subroutine.calculates.the.projection.of.X.and.Y

PROJECT CLC
 LDA TG
 ADC TH
 CLC
 ADC TI ;This.is.rotated.Z
 CLC
 ADC #128 ;We.are.going.to.take.128+z
 TAX  ;Now.it.is.ready.for.indexing
 LDA ZDIV,X ;Table.of.-d/z
 STA AUX ;This.is.for.the.projection
 STA REM ;Multiply.can.clobber.AUX

 CLC
 LDA TA
 ADC TB
 CLC
 ADC TC
 STA ACC ;This.is.rotated.x
 JSR SMULT ;Signed.multiply.ACC*AUX/2^OFFSET
 CLC
 LDA ACC
:CONT1 ADC #64 ;Offset.the.coordinate
*.See.below.for.the.reason.why.this
*.next.instruction.is.commented.out
*.TAX..;Now.X.is.x!
 STA TX1
 CLC  ;Do.the.whole.thing.again.for.Y
 LDA REM
 STA AUX
 LDA TD
 ADC TE
 CLC
 ADC TF
 STA ACC ;This.is.rotated.y
 JSR SMULT ;Signed.multiply.ACC*AUX/2^OFFSET
 CLC
 LDA ACC
:CONT2 ADC #64 ;Offset.the.coordinate
*.For.some.completely.unknown.reason.to.me
*.the.instruction.below.doesn't.work...Somehow
*.the.RTS.is.modifying.X.and.Y???
*.TAY..;Store.in.Y
 STA TY1
 RTS  ;I.hope.to.heck.this.works.

*-------------------------------
*.SMULT:.8-bit.signed.(sort-of).multiply
*
*.ACC*AUX/2^OFFSET.->.[ACC,.EXT]..16-bit.result..lo,hi
*
*.Note.that.this.routine.divides.the.end.result.by.2^OFFSET

*.Yup,.another.macro.
DIVOFF MAC  ;Divide.by.the.float.offset
 LUP OFFSET ;Repeat.offset.times
 LSR  ;A.contains.high.byte
 ROR ACC ;ACC.is.low.byte
 --^
 <<<


SMULT CLC
 LDA ACC ;First,.is.the.result.positive.or.negative?
 EOR AUX
 BMI :NEG

 LDA ACC ;They.are.either.both.negative.or
 BPL :CONT1 ;both.positive
 EOR #$FF ;In.this.case,.make.them
 ADC #$01 ;both.positive!
 STA ACC
 >>> NEG,AUX ;Little.macro.used.earlier.
:CONT1 LDA #00 ;Multiply.the.two.numbers
 LDY #$09
]LOOP LSR  ;Read.the.article.for.details.
 ROR ACC
 BCC :MULT1 ;Or.figure.it.out.yourself!
 CLC
 ADC AUX
:MULT1 DEY
 BNE ]LOOP
 >>> DIVOFF ;Remove.this.line.for.a.general.multiply
 STA EXT
 RTS

:NEG LDA ACC ;One.of.the.two.is.negative
 BMI :CONT2
 >>> NEG,AUX ;Otherwise.it's.AUX
 JMP :CONT3
:CONT2 EOR #$FF ;Take.two's.complement
 ADC #$01
 STA ACC
:CONT3 LDA #00 ;Multiply
 LDY #$09
]LOOP2 LSR
 ROR ACC
 BCC :MULT2
 CLC
 ADC AUX
:MULT2 DEY
 BNE ]LOOP2
 >>> DIVOFF ;Again,.divide.by.the.offset
 STA EXT
 LDA ACC
 BPL :OK ;Something.is.really.wrong.if.this.is.negative.
 JSR CHOKE
:OK EOR #$FF ;Otherise,.everything.relevant.should
 ADC #$01 ;be.completely.in.the.low.byte.
 STA ACC
 RTS  ;I.hope...

*-------------------------------
*.General.questionable-value.error.procedure

CHOKE LDX #00
:LOOP LDA :CTEXT,X
 BEQ :DONE
 JSR CHROUT
 INX
 JMP :LOOP
:DONE RTS
:CTEXT HEX 0D ;CR
 TXT 'something.choked.:('
 HEX 0D00

*-------------------------------
*.Drawin'.a.line...A.fahn.lahn.

***.Some.useful.macros

PLOTPX MAC  ;plot.a.point.in.x
 PHA  ;Use.this.one.every.time
 LDA BITP,X ;X.is.increased
 BPL C1
 EOR BUFFER
 STA BUFFER
 BMI C2
 INC BUFFER+1
C2 LDA #%10000000
C1 ORA (BUFFER),Y
 STA (BUFFER),Y
 PLA  ;Need.to.save.A!
 <<<
```

```
PLOTPY MAC  ;Plot.a.point.in.y:.simpler.and.necessary!
 PHA  ;Use.this.one.when.you.just.increase.Y
 LDA BITP,X ;but.X.doesn't.change
 ORA (BUFFER),Y
 STA (BUFFER),Y
 PLA
 <<<

CINIT MAC  ;Macro.to.initialize.the.counter
 LDA ]1 ;dx.or.dy
 LSR
 EOR #$FF ;(Not.really.two's.complement)
 ADC #$01 ;A.=.256-dx/2.or.256-dy/2
 <<<  ;The.dx/2.makes.a.nicer.looking.line

XSTEP MAC  ;Macro.to.take.a.step.in.X
XLOOP INX
 ADC DY
 BCC L1
*.Do.we.use.INY.or.DEY.here?
 IF I,]1 ;If.the.first.character.is.an.'I'
 INY
 ELSE
 DEY
 FIN
 SBC DX
L1 >>> PLOTPX ;Always.take.a.step.in.X
 CPX X2
 BNE XLOOP
 <<<

YSTEP MAC  ;Same.thing,.but.for.Y
YLOOP IF I,]1
 INY
 ELSE
 DEY
 CLC  ;Very.important!
 FIN
 ADC DX
 BCC L2
 INX  ;Always.increase.X
 SBC DY
 >>> PLOTPX
 JMP L3
L2 >>> PLOTPY ;We.only.increased.Y
L3 CPY Y2
 BNE YLOOP
 <<<

****.Initial.line.setup

DRAW >>> MOVE,TX1;X1  ;Move.stuff.into.zero.page
 >>> MOVE,TX2;X2  ;Where.it.can.be.modified
 >>> MOVE,TY1;Y1
 >>> MOVE,TY2;Y2
 >>> SETBUF ;Now.we.can.clobber.the.buffer

 SEC  ;Make.sure.x1<x2
 LDA X2
 SBC X1
 BCS :CONT
 LDA Y2 ;If.not,.swap.P1.and.P2
 LDY Y1
 STA Y1
 STY Y2
 LDA X1
 LDY X2
 STY X1
 STA X2

 SBC X1 ;Now.A=dx
:CONT STA DX
 LDX X1 ;Put.x1.into.X,.now.we.can.trash.X1

COLUMN LDA X1 ;Find.the.first.column.for.X
 LSR  ;(This.can.be.made.much.faster!)
 LSR  ;There.are.x1/8.128.byte.blocks
 LSR  ;Which.means.x1/16.256.byte.blocks
 LSR
 BCC :EVEN ;With.a.possible.extra.128.byte.block
 LDY #$80 ;if.so,.set.the.high.bit
 STY BUFFER
 CLC
:EVEN ADC BUFFER+1 ;Add.in.the.number.of.256.byte.blocks
 STA BUFFER+1 ;And.store.it!

 SEC
 LDA Y2 ;Calculate.dy
 SBC Y1
 BCS :CONT2 ;Is.y2>y1?
 LDA Y1 ;Otherwise.dy=y1-y2
 SBC Y2
:CONT2 STA DY
 CMP DX ;Who's.bigger:.dy.or.dx?
 BCS STEPINY ;If.dy,.we.need.to.take.big.steps.in.y

STEPINX LDY Y1 ;X.is.already.set.to.x1
 LDA BITP,X ;Plot.the.first.point
 ORA (BUFFER),Y
 STA (BUFFER),Y
 >>> CINIT,DX ;Initialize.the.counter
 CPY Y2
 BCS XDECY ;Do.we.step.forwards.or.backwards.in.Y?

XINCY >>> XSTEP,INY
 RTS

XDECY >>> XSTEP,DEY
 RTS

STEPINY LDY Y1 ;Well,.a.little.repetition.never.hurt.anyone
 LDA BITP,X
 ORA (BUFFER),Y
 STA (BUFFER),Y
 >>> CINIT,DY
 CPY Y2
 BCS YDECY

YINCY >>> YSTEP,INY
 RTS

YDECY >>> YSTEP,DEY
 RTS


*-------------------------------
*.Clean.up

CLEANUP LDA VMCSB ;Switch.char.rom.back.in
 AND #%11110101 ;default
 STA VMCSB

 RTS  ;bye!

*-------------------------------
*.Some.variables

TX1 DS 1
TY1 DS 1
TX2 DS 1
TY2 DS 1
P1X DS 1 ;These.are.temporary.storage
P1Y DS 1 ;Used.in.plotting.the.projection
P2X DS 1
P2Y DS 1 ;They.are.here.so.that.we
P3X DS 1 ;don't.have.to.recalculate.them.
P3Y DS 1
P4X DS 1 ;They.make.life.easy.
P4Y DS 1
P5X DS 1 ;Why.are.you.looking.at.me.like.that?
P5Y DS 1 ;Don't.you.trust.me?
P6X DS 1
P6Y DS 1 ;Having.another.child.wasn't.my.idea.
P7X DS 1
P7Y DS 1
P8X DS 1
P8Y DS 1
DSX DS 1 ;DSX.is.the.increment.for.rotating.around.x
DSY DS 1 ;Similar.for.DSY,.DSZ
DSZ DS 1
SX DS 1 ;These.are.the.actual.angles.in.x.y.and.z
SY DS 1
```

```
SZ DS 1
T1 DS 1 ;These.are.used.in.the.rotation
T2 DS 1
T3 DS 1 ;See.the.article.for.more.details
T4 DS 1
T5 DS 1
T6 DS 1
T7 DS 1
T8 DS 1
T9 DS 1
T10 DS 1
A11 DS 1 ;These.are.the.elements.of.the.rotation.matrix
B12 DS 1 ;XYZ
C13 DS 1
D21 DS 1 ;The.number.denotes.(row,column)
E22 DS 1
F23 DS 1
G31 DS 1
H32 DS 1
I33 DS 1
TA DS 1 ;These.are.temporary.locations
TB DS 1 ;for.use.by.the.projection.routine
TC DS 1
TD DS 1
TE DS 1
TF DS 1
TG DS 1
TH DS 1
TI DS 1

*-------------------------------
*.Set.up.bit.table

 DS ^ ;Clear.to.end.of.page
   ;So.that.tables.start.on.a.page.boundary
BITP LUP 16 ;128.Entries.for.X
 DFB %10000000
 DFB %01000000
 DFB %00100000
 DFB %00010000
 DFB %00001000
 DFB %00000100
 DFB %00000010
 DFB %00000001
 --^
SIN   ;Table.of.sines,.120.bytes
COS EQU SIN+128 ;Table.of.cosines
   ;Both.of.these.trig.tables.are
   ;currently.set.up.from.BASIC
ZDIV EQU COS+128 ;Division.table
```

UUencoded Binaries

```
begin 666 runme3d
M 0@>" H BT&R,*=!LC$ZDR)#54)%,T0N3R(L."PY "X(% "3(DE.250S1"(L
M.    !H:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:
F&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:

end

begin 666 init3d
M 0@D" H CR!04D]'4D%-(%1/($E.251)04Q)6D4@0U5"13-$ $8(% "/(%-4
M15!(14X@2E5$1"P@1T5/4D=%(%1!64Q/4@!7"!D F2(%5T]22TE.1R([ ',(
M'@!"4[(V-3(X.D)#LC8V-38Z0EJR-C<X- "#""@ 0;(P.D1!LO^M-C  LP@R
M (%)LC"D,3(P.E,ELC,RK+\H02FJ+C4Z0R6R,S*LOBA!*:HN-3I!LD&J1$$
MO0@S )DB+B([ -0(-P"+(%,ELS @IR!3);(R-3:J4R4 ZP@X (L@0R6S,""G
M($,ELC(U-JI#)0 !"3H ET)3JDDL4R4ZET)#JDDL0R4 !PD[ (( (0D\ $2R
M.# Z6C"R,SI:LJLQ,C@Z1%JR,0 N"48 @4JR,*0R-34 . E' )DB(2([ $X)
M4 !1);(V-*Q$K2@V-*Q:,*M:*0!D"5$ BR!1);$Q,C<@IR!1);(Q,C< ? E2
M (L@426SJS$R-R"G(%$ELJLQ,C< DPE3 (L@426S,""G(%$ELC(U-JI1)0"@
M"54 ET):JDHL424 K0E: %JR6JI$6CJ" ,4)9 ">-# Y-CJ9(DY%050L($A5
M2#\B    &AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:
1&AH:&AH:&AH:&AH:&AH:&AH:

end

begin 666 listme3d
M 0@8" H CR!.3U1%4R!&3U(@0U5"13-$ !X("P"/ $4(# "/(%1(15)%($%2
M12!35$E,3"!!($9%5R!-24Y/4B!"54=3 &D(#0"/(%1(12!-3U1)3TX@25,,@
M2D522UD@1D]2(%--04Q, (T(#@"/($E.0U)%345.5%,Z(%!23T)!0DQ9(%)/
M54Y$3T9& ),(#P"/ +@($ "/($9%14P@1E)%%12!43R!-3T1)1ED@5$A%(%!!
M4DU3+@#>"!$ CR!*55-4(%%5250@5$A%(%!23T=204T@04Y$($Q)4U0N  ,)
M$@"/(%1262!#2$%.1TE.1R!$($%.1"!:," H64]5($-!3@ C"1, CR!%5D5.
M($U!2T4@6BU:,"!.14=!5$E612DN $8)% "/($)%($-!4D5&54P@04)/550@
M4D535$%25$E.1R$ ;0D5 (\@248@64]5(%=!3E0@5$\@4T5%(%E/55(@0T]-
M4%5415( D@D6 (\@1$E%($%.($%-05I)3D<@1$5!5$@@2E535"!465!% +@)
M%P"/("=254X@-C G+B!!3%=!65,@4E5.(%1(12!%3E1)4D4 QPD8 (\@4%)/
M1U)!32X S0D9 (\ \PD: (\@4T]-151)3453(%1(12!%6453($=%5"!#3TY&
M55-%1  8"AL CR!!0D]55"!42$4@4$524U!%0U1)5D4L($%.1"!42$4 .PH<
M (\@0U5"12!724Q,($Q/3TL@4D5!3$Q9(%=%25)$+@!?"AT CR!*55-4($),
M24Y+($]2(%1262!43R!&24Y$(%1(10"#"AX CR!224=(5"!015)34$5#5$E6
M12X@($5615(@4T5%3@"H"A\ CR!42$4@)T-205I9($-2051%)S\@(%-!344@
M241%02X N@H@ (\@4TQ*(#<O,3DO.30    :&AH:&AH:&AH:&AH:&AH:&AH:
M&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:
#&AH:

end

begin 666 cube3d.o
M !"I (T@T(TAT*T8T"D/"1"-&-"@ *D?A?NI$(7\3%41DP41$1$@(" @(" @
M(" @(" @("!#54)%%,T0-#2 @(" @(" @(" @(" @(" @0ED-GR @("!35$50
M2$5.($I51$29(" @($=%3U)'12!405E,3U(-#9L@($-(14-+($]55"!42$4@
M2E5,62 Y-"!)4U-512!/1@V6("!#/4A!0TM)3D>;($9/4B!-3U)%($1%5$%)
M3%,A#0T='9X21C$O1C*2("T@24Y#+T1%0R!8+5)/5$%424].#1T=$D8S+T8T
MDB M($E.0R]$14,@62U23U1!5$E/3@T='1)&-2]&-I(@+2!)3D,O1$5#(%HM
M4D]4051)3TX-'1T21C>2(%)%4T544T@(%!215-3(%$@5$\@455)5 T-!2 @
M(" @(%!215-3($%.62!+15D@5$\@0D5'24X- +'[\ L@TO_(T/;F_$Q5$2#D
M_\D \/FIDR#2_ZE :0R%^ZD%A?RI *  H@ 8D?O(:1"0^1BE^VDHA?NE_&D
MA?R@ .B*X!#0Y*D A:.I,(6DA0*M&- I\0D.C1C0J0"-SQB-T!B-T1B-TAB-
MTQB-U!@@Y/_)A= -K<\8R3SP8^[/&$PV$LF)T NMSQCP5,[/&$PV$LF&T VM
MT!C)//!#[M 83#82R8K0"ZW0&/ TSM 83#82R8?0#:W1&,D\\"/NT1A,-A+)
MB] +K=$8\!3.T1A,-A+)B- #3+$1R5'0 TRR&!BMTAAMSQC)>) "Z7B-TA@8
MK=,8;= 8R7B0 NEXC=,8&*W4&&W1&,EXD +I>(W4&#BMTQCMU!BP FEXC=48
M&*W3&&W4&,EXD +I>(W6&!BMTAAMU!C)>) "Z7B-UQ@XK=(8[=08L )I>(W8
M&!BMTAAMUAC)>) "Z7B-V1@XK=(8[=48L )I>(W:&!BMTAAMU1C)>) "Z7B-
MVQ@XK=88[=(8L )I>(W<&#BMTQCMTABP FEXC=T8&*W2&&W3&,EXD +I>(W>
M&!BNU1B] !JNUAA] !J-WQBNU1B]@!DXKM88_8 9C> 8KM,8O8 9$ X82?]I
M 0H82?]I 4PS$PJ-X1@XKMP8OO :KML8_0 :.*[9&/T &ABNVAA] !H0#AA)
M_VD!2AA)_VD!3&(32ABNUQA]@!DXKM@8_8 9C>(8.*[9&+V &:[:&/V &3BN
MVQC]@!DXKMP8_8 9$ X82?]I 4H82?]I 4R?$TH8KM<8?0 :&*[8&'T &HWC
M&*[=&+V &3BNWAC]@!F-Y!BNVAB]@!DXKMP8_8 9.*[;&/V &3BNV1C]@!D0
M#AA)_VD!2AA)_VD!3.P32ABNV!A] !HXKM<8_0 :C>48&*[:&+T &J[;&'T
M&CBNV1C] !HXKMP8_0 :$ X82?]I 4H82?]I 4PI%$H8KM<8?8 9&*[8&'V
M&8WF&!BNW1B] !JNWAA] !J-YQBMWQB-Z!BMX!B-Z1BMX1B-ZABMXAB-ZQBM
MXQB-[!BMY!B-[1BMY1B-[ABMYAB-[QBMYQB-\!@@G1:NNQBLO!B.OQB,P!@8
MK> 82?]I 8WI&!BMXQA)_VD!C>P8&*WF&$G_:0&-[Q@@G1:NNQBLO!B.P1B,
MPA@8K=\82?]I 8WH&!BMXAA)_VD!C>L8&*WE&$G_:0&-[A@@G1:NNQBLO!B.
MPQB,Q!BMX!B-Z1BMXQB-[!BMYAB-[Q@@G1:NNQBLO!B.Q1B,QA@8K>$82?]I
M 8WJ&!BMY!A)_VD!C>T8&*WG&$G_:0&-\!@@G1:NNQBLO!B.S1B,SA@8K> 8
M2?]I 8WI&!BMXQA)_VD!C>P8&*WF&$G_:0&-[Q@@G1:NNQBLO!B.RQB,S!BM
MWQB-Z!BMXAB-ZQBMY1B-[ABMX!B-Z1BMXQB-[!BMYAB-[Q@@G1:NNQBLO!B.
MR1B,RABMX!B-Z1BMY!B-[ABMYQB-[Q@@G1:NNQBLO!B.QQB,R!BI (6CI0*%I*D H@B@ )&CR-#[YJ3*T/:E
MI*V_&(V[&*W &(V\&*W!&(V]&*W"&(V^&""".%ZW#&(V[&*W$&(V\&"".%ZW%
M&(V]&*W&&(V^&"".%ZV_&(V[&*W &(V\&"".%ZW'&(V]&*W(&(V^&"".%ZW)
M&(V[&*W*&(V\&"".%ZW!&(V]&*W"&(V^&"".%ZW+&(V]&*W,&(V^&"".%ZW#
M&(V[&*W$&(V\&"".%ZW-&(V[&*W.&(V\&"".%ZW%&(V]&*W&&(V^&"".%ZW'
M&(V]&*W(&(V^&"".%ZT8T$D"C1C0J0A% H4"3,41&*WN&&WO&!AM\!@8:8"J
MO8 :A?R%_ABMZ!AMZ1@8;>H8A?L@Z!88I?MI0(V[&!BE_H7\K>L8;>P8&&WM
M&(7[(.@6&*7[:4"-O!A@&*7[1?PP-:7[$ U)_VD!A?L8I?Q)_VD!J0"@"4IF
M^Y #&&7\B-#U2F;[2F;[2F;[2F;[2F;[A?U@I?LP"ABE_$G_:0%,.!=)
M_VD!A?NI * )2F;[D ,89?R(T/5*9OM*9OM*9OM*9OM*9OM*9ON%_:7[$ ,@
M:1=)_VD!A?M@H@"]>!?P!R#2_^A,:Q=@#5-/34542$E.1R!$.1R!#+140@.B@-
M *V[&(7[K;T8A?VMO!B%_*V^&(7J0"%HZ4"A:E?ZD_(7\A/ZE 2I?ZD_(7\A^L7Y
ML%BD_+T &1&CD:.E^4I)_VD!Q/ZP(NAE^I #+&T Y&D:.E^4I)_VD!A?2@ XCE^4I)_VD!A?^H XCE^4I)_VD!A?2@ XCE^4I)_V
MH*"@H*"@H*"@*@TJH&=%3U)'1:!T05E,3U*@H*"@H*"@H*"@H*"@H*"@*@TJ
MH'-405)4140ZH#<O,3$O.32@H*"@H*"@H*"@H*"@*@TJH&9)3DE32$5$.J W
M+S$Y+SDTH*"@H*"@H*"@H*"@*@TJH*"@H*"@H*"@H*"@H*"@H*"@H*"@H*"@
MH*"@H*"@*@TJH'=%3$PLH$E&H$%,3*!'3T53H%=%3$R@5$A)4Z"@*@TJH%!2
M3T=204V@5TE,3*!23U1!5$6@0:!#54)%+J"@*@TJH*"@H*"@H*"@H*"@H*"@
MH*"@H*"@H*"@H*"@H*"@*@TJH'1(25.@4%)/1U)!3:!)4Z!)3E1%3D1%1*!4
M3Z"@*@TJH$%#0T]-4$%.6:!42$6@05)424-,1:!)3J"@H*"@*@TJH&,]:$%#
M2TE.1RR@:E5,6: Y-*!)4U-512Z@H*"@*@TJH&9/4J!$151!24Q3H$].H%1(
M25.@4%)/1U)!32R@*@TJH%)%042@5$A%H$%25$E#3$4AH*"@H*"@H*"@H*"@
M*@TJH*"@H*"@H*"@H*"@H*"@H*"@H*"@H*"@H*"@H*"@*@TJH'=2251%H%1/
MH%53(:"@H*"@H*"@H*"@H*"@H*"@*@TJH*"@H*"@H*"@H*"@H*"@H*"@H*"@
MH*"@H*"@H*"@*@TJH%5.*$)%12E-3Z"@H*"@H*"@H*"@H*"@H*"@H*"@*@TJ
MH*"@H*"@H*"@H*"@H*"@H*"@H*"@H*"@H*"@H*"@*@TJH%9)H*"@H*"@H*"@
MH*"@H*"@H*"@H*"@H*"@H*"@*@TJH$XH24XI1Z"@H*"@H*"@H*"@H*"@H*"@
MH*"@H*"@*@TJH$%212A42*"@H*"@H*"@H*"@H*"@H*"@H*"@H*"@*@TJH$4I
M64]5*$^@H*"@H*"@H*"@H*"@H*"@H*"@H*"@*@TJH$Y,62F@H*"@H*"@H*"@
MH*"@H*"@H*"@H*"@*@TJH*"@H*"@H*"@H*"@H*"@H*"@H*"@H*"@H*"@H*"@
MH*"@*@TJH$%33"A23U-%*45%4*"@H*"@H*"@H*"@H*"@H*"@*@TJH*"@H*"@
MH*"@H*"@H*!%+D4N0U5-34E.1U.@H*"@*@TJH*"@H*"@H*"@H*"@H*"@H*"@
MH*"@H*"@H*"@H*"@*@TJH' N<RZ@=$A)4Z!705.@5U))5%1%3J!54TE.1Z"@
M*@TJH*"@H*"@;4523$E.H#$R."Z@H'=)5$B@0:"@H*"@*@TJH*"@H*"@3$E4
M5$Q%H$U/1$E&24-!5$E/3J!)5*"@*@TJH*"@H*"@5TE,3*!73U)+H$9)3D6@
M5TE42*"@H*"@*@TJH*"@H*"@;4523$E.H#8T+J"@:4:@64]5H*"@H*"@*@TJ
MH*"@H*"@1$].)U2@2$%61:!%251(15(N+BZ@H*"@*@TJH*"@H*"@5T5,3"R@
M5T6@04Q,H$A!5D6@3U52H*"@*@TJH*"@H*"@3$E45$Q%H$9!54Q44RZ@H*"@
MH*"@H*"@*@TJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*@T-(&]R
M9R D,3 P, T-*J!C3TY35$%.5%,-#6)U9F8Q(&5Q=2 D,S P," [9DE24U2@
M0TA!4D%#5$52H%-%5 UB=69F,B!E<74@)#,X,# @.W-%0T].1*!#2$%204-4
M15*@4T54#6)U9F9E<B!E<74@)&$S(#MP4D5354U!0DQ9H%1(1:!405!%H%=/
M3B=4H$)%H%)53DY)3D<->#$@97%U("1F8B [<$])3E13H$9/4J!$4D%724Y'
MH$&@3$E.10UY,2!E<74@)&9C(#MT2$531:!:15)/H%!!1T6@041$4D534T53
M#7@R(&5Q=2 D9F0@.T1/3B=4H$-/3D9,24-4H%=)5$B@8F%S:6,->3(@97%U
M("1F90UD>"!E<74@)&8Y#61Y(&5Q=2 D9F$-=&5M<#$@97%U("1F8B [;T:@
M0T]54E-%+*!#3U5,1*!#3TY&3$E#5*!7251(H%@Q#71E;7 R(&5Q=2 D9F,@
M.W1%35!/4D%26:!605))04),15,-86-C(&5Q=2 D9F(@.W1(15-%H$9/55*@
M5D%224%"3$53H$%21:!54T5$#6%U>"!E<74@)&9C(#M"6:!42$6@355,5$E0
M3$E#051)3TZ@4D]55$E.10UE>'0@97%U("1F9 UR96T@97%U("1F90UZ=&5M
M<"!E<74@)# R(#MU4T5$H$9/4J!"549&15*@4U=!4H&1/3B=4H%1/54-(
M+@T-86YG;6%X(&5Q=2 Q,C @.W1(15)%H$%21: R*E!)+T%.1TU!6*!!3D=,
M15,-;V9F<V5T(&5Q=2 V(#MF3$]!5*!/1D93150ZH%@]6$%#5*!3"HR7D]&
M1E-%5 T-*J!V:6,-#79M8W-B(&5Q=2 D9# Q. UB:V=N9"!E<74@)&0P,C -
M8F]R9&5R(&5Q=2 D9# R,0US<W1A<G0@97%U(#$$S-#0@.U)/5Z YH$E.H%-#
M4D5%3J!-14U/4EE@052@%D%,3$M%W3&!%3V5 $+'8!'9T:*)I$@<4/)/
M9F0R#6=E=&EN(&5Q=2 D9F9E- T-*BHJH&U0!U)/4T-TR+6]V92!M86,-(&QD
M82!=,,0T@<W1A(%TR#2 \/#P-#6=E=&EN#&%*!C86,@*#MW04E4H$9/4J!!H$M%
M65!215-3*0T@8FYE(&=E=&EN#2!C-7 @(S<=A:70@:G-R(&%S<V5M8FQE2@D@T@/#P\
M0UD96)U9R!M86,@*#MP4D=E+'5*!!H$-*(05)!0U1%4B @*#MD3TXG
M5*!4U-%-)34*(&Q1%-4*A;@971Q;&4-9>5"9@87-S(&5M8FQE<B!G:79E3<W,NGB1G7S-B
M+E8E,2*!G04>0H9$)/1E-%/4W2%A#C:!G05(@/U&(4Q=>#)7Q#:4
M5<&5E39P1E43&X&12BDHG96I%-@F,7E@.UB.V0
M.NBCQG5&PI5'34:95=9X@9RU;-R5 S<4
M5F&+3
```

```
MH$=205!(24-3#2!S=&$@=&5M<#$@.V-/3%5-3J Q,@T@;&1A(",^<W-T87)T
M(#MR3U>@.0T@<W1A('1E;7 Q*S$@.W-S=&%R=*!03TE.5%.@5$^@4D]7H#D-
M(&QD82 C,# -(&QD>2 C,# -(&QD>" C,# @.UB@5TE,3*!#3U5.5* Q-J!2
M3U=3H$9/4J!54PT@8VQC#0TZ;&]O<"!S=&$@*'1E;7 Q*2QY#2!I;GD-(&%D
M8R C,38-(&)C8R Z;&]O< T@8VQC#2!L9&$@=&5M<#$-(&%D8R C-# @.VY%
M142@5$^@041$H#0PH%1/H%1(1:!"05-%H%!/24Y415(-('-T82!T96UP,2 [
M=$^@2E5-4*!43Z!42$6@3D585*!23U<-(&QD82!T96UP,2LQ#2!A9&,@(S P
M(#MT04M%H$-!4D6@3T:@0T%24DE%4PT@<W1A('1E;7 Q*S$-(&QD>2 C,# -
M(&EN> T@='AA(" [>*!)4Z!!3%-/H$%.H$E.1$58H$E.5$^@5$A%H$-(05)!
M0U1%4J!.54U"15(-(&-P>" C,38-(&)N92 Z;&]O<" [;D5%1*!43Z!$3Z!)
M5* Q-J!424U%4PT-(#X^/B!D96)U9RPG,B<-*BHJ*J!S152@55"@0E5&1D52
M4PT-(&QD82 C/&)U9F8Q#2!S=&$@8G5F9F5R#2!L9&$@(SYB=69F,0T@<W1A
M(&)U9F9E<BLQ#2!S=&$@>G1E;7 @.UI414U0H%=)3$R@34%+1:!,249%H%-)
M35!,1:!&3U*@55,-(&QD82!V;6-S8@T@86YD(",E,3$Q,3 P,#$@.W-405)4
MH$A%4D6@4T^@5$A!5*!35T%0H$)51D9%4E.@5TE,3*!73U)+H%))1TA4#2!O
M<F$@(R4P,# P,3$Q, T@<W1A('9M8W-B#0T-*BHJ*J!S152@55"@24Y)5$E!
M3*!604Q515,-#6EN:70@;&1A(",P, T@<W1A(&1S> T@<W1A(&1S>0T@<W1A
M(&1S>@T@<W1A('-X#2!S=&$@<WD-('-T82!S>@T-(#X^/B!D96)U9RPG-"<-
M#2HM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM#2J@;4%)3J!,3T]0
M#0TJ*BHJH&=%5*!+15E04D534PT-;6%I;@UK<')E<W,@:G-R(&=E=&EN#2!C
M;7 @(S$S,R R [9C$_#2!B;F4@.F8R#2!L9&$@9'-X#2!C;7 @(V%N9VUA>"\R
M(#MN3Z!-3U)%%H%1(04Z@4$D-(&)E<2 Z8V]N= T@:6YC(&1S>" [3U1(15)7
M25-%H$E.0U)%%05-%H%@M4D]4051)3TX-(&IM<" Z8V]N= TZ9C(@8VUP(",Q
M,S<@.V8R/PT@8FYE(#IF,PT@;&1A(&1S> T@8F5Q(#IC;VYT#2!D96,@9'-X
M#2!J;7 @.F-O;G0-.F8S(&-M<" C,3,T#2!B;F4@.F8T#2!L9&$@9'-Y#2!C
M;7 @(V%N9VUA>"\R#2!B97$@.F-O;G0-(&EN8R!D<WD@.VE.0U)%%05-%H%DM
M4D]4051)3TX-(&IM<" Z8V]N= TZ9C0@8VUP(",Q,S@@.F8U/PT@8FYE(#IF-
M82!D<WD-(&)E<2 Z8V]N= T@9&5C(&1S>0T@:FUP(#IC;VYT#2!J;7 @.F-O
M;G0-.F8U(&-M<" C,3,U#2!B;F4@.F8V#2!L9&$@9'-Z#2!C;7 @(V%N9VUA
M>"\R#2!B97$@.F-O;G0-(&EN8R!D<WH@.VE.0U)%%05-%H%HM4D]4051)3TX-
M(&IM<" Z8V]N= TZ9C8@8VUP(",Q,SD@.F8W/PT@8FYE(#IF.!B;F4@.F8X
M#2!L9&$@9'-Z#2!B97$@.F-O;G0-(&1E8R!D<WH-(&IM<" Z8V]N= TZ9C<@
M8VUP(",Q-#$@.V8W/PT@8FYE(#IF.&%1%;G0-.F8W(&-M<" C,3,W#2!B;F4@
M.F8X#2!L9&$@9'-Y#2!C;7 @(V%N9VUA>"\R#2!B97$@.F-O;G0-(&EN8R!D
M<WD@.VE.0U)%%05-%H%DM4D]4051)3TX-(&IM<" Z8V]N= TZ9C@@8VUP(",Q
M-C(@.V8X/PT@8FYE(#IF.0T@;&1A(&1S>0T@8F5Q(#IC;VYT#2!D96,@9'-Y
M#2!J;7 @.F-O;G0-.F,Y(&-M<" C,3,V#2!B;F4@.F,Q, T@;&1A(&1S>@T@
M<V)C('-I;BQX#2!S96,-(&QD>"!T-PT@<V)C('-I;BQX#2!S96,-(&QD>"!T
M. T@<V)C('-I;BQX(#ME23TH4TE.*%0U*2U324XH5#8I+5-)3BA4-RDM4TE.
M*%0X*2DO,@T@/CX^(&1I=C(-(&-L8PT@;&1X('0S#2!A9&,@8V]S+'@-(&-L
M8PT@;&1X('0T#2!A9&,@8V]S+'@-('-T82!E,C(@.V4]*$-/4RA4,RDK0T]3
M*%0T*2ME22DO,@TZ8V%L8V8@;&1X('0Y#2!L9&$@<VEN+'@-('-E8PT@;&1X
M('0Q, T@<V)C('-I;BQX#2!S=&$@9C(S(#MF/2A324XH5#DI+5-)3BA4,3 I
M*2\R#3IC86QC9R!L9'@@=#8-(&QD82!S:6XL> T@<V5C#2!L9'@@=#@-('-B
M8R!S:6XL> T@<V5C#2!L9'@@=#<-('-B8R!S:6XL> T@<V5C#2!L9'@@=#4-
M('-B8R!S:6XL>" [9TD]*%-)3BA4-BDM4TE.*%0X*2U324XH5#<I+5-)3BA4
M-2DI+S(-(#X^/B!D:78R#2!C;&,-(&QD>"!T- T@861C(&-O<RQX#2!S96,-
M(&QD>"!T,PT@<V)C(&-O<RQX#2!S=&$@9S,Q(#MG/2A#3U,H5#0I+4-/4RA4
M,RDK9TDI+S(-(#X^/B!D96)U9V$L9S,Q,#2 ^/CX@9&5B=6<L)T<G#3IC86QC
M:"!C;&,-(&QD>"!T-@T@;&1A(&-O<RQX#2!L9'@@=#<-(&%D8R!C;W,L> T@
M<V5C#2!L9'@@=#4-('-B8R!C;W,L> T@<V5C#2!L9'@@=#@-('-B8R!C;W,L
M>" [:$D]*$-/4RA4-BDK0T]3*%0W*2U#3U,H5#4I+4-/4RA4."DI+S(-(#X^
M/B!D:78R#2!C;&,-(&QD>"!T,PT@861C('-I;BQX#2!C;&,-(&QD>"!T- T@
M861C('-I;BQX#2!S=&$@:#,R(#MH/2A324XH5#,I*U-)3BA4-"DK:$DI+S(-
M.G=H97<@8VQC#2!L9'@@=#D-(&QD82!C;W,L> T@;&1X('0Q, T@861C(&-O
M<RQX#2!S=&$@:3,S(#MI/2A#3U,H5#DI*T-/4RA4,3 I*2\R#0TJ*J!I5"=3
MH$%,3*!$3U=.,2$E,3*!&4D]-H$A%4D4N#0TJ*J!R3U1!5$4LH%!23TI%0U0L
MH$%.1*!35$]21:!42$6@4$])3E13#61O=VYH:6QL(&QD82!A,3$@.W1(25.@
M25.@1T545$E.1Z!43Z!"1:!!H%)%04R@34534PT@<W1A('1A#2!L9&$@8C$R
M(#MT2$6@4D5!4T].H%1(25.@25.@1$].10T@<W1A('1B(#M)4Z!43Z!-04M%
MH%1(1:!#3T1%H$&@3$E45$Q%#2!L9&$@8S$S(#M%05-)15*@5$$^@4D5!1* H
M04Y$H$1%0E5'(2D-('-T82!T8PT@;&1A(&0R,2 [=$A%4T6@05)%H$$,3*!4
M14U03U)!4EF@3$]#051)3TY3#2!S=&$@=&0@.W53142@0EF@5$A%H%!23TI%
M0U1)3TZ@4U5"4D]55$E.12X-(&QD82!E,C(-('-T82!T92 [;U1(15)725-%
M+*!42$521:!73U5,1*!"1:!%24=(5 T@;&1A(&8R,R [3$].1Z!23U5424Y%
M4Z!(15)%%+@T@<W1A('1F#2!L9&$@9S,Q(#MB552@252@5T]53$2@0D6@4TE'
M3DE&24-!3E1,6:!&05-415(-(-('-T82!T9PT@;&1A(&&@S,@T@<W1A('1H#2!L
M9&$@:3,S#2!S=&$@=&D-#2J@8:!.14]4H$U)0)/#6YE9R!M86,@(#MC2$%.
M1T6@5$A%H%-)1TZ@1D]2H$%,3*!-05-415)3H$)53U14G4Z!#3TU03$%4#2!C;&,-(&QD82!S=&$@2!!C;&,-
M,2 [3E5-0D52+@T@96]R(",D9F8-(&%D8R C)# Q#2 \/#P-#2J@<#$]6S&@
M,: Q70T@:G-R('!R;VIE8W0@.W5.4D],3*!42$E3H%=(3%0Q(24Y'#2!L
M9'@@='Q(#LH4T]24)/552@5$%4T5=(%=0(3D53*0T@;&1Y('1Y
M,2 [*%-%1:!P<F]J96-TH$9/4J!214%33TZ5TA9*0T@<W1X(' Q>" [9D]2
MH$&@4%)%5%19H$$))1Z!34%5$%1*!)3D-214%34$-!-('-T>2!P,7D-*J!P,CU;
M,: M,: Q70T@/CX^(&YE9RQB,3(@.V-(04Y'1:!!42$531:!%3T5$-14Y44PT@
M<W1A('1B#2 ^/CX@;F5G+&4R,B [<TE.0T6@6:!)4Z!.3U@@+3$$-('-T82!T
M90T@/CX^(&YE9RQH,S(-('-T82!T: T@:G-R('!R;VIE8W0-(&QD>"!T>#$-
M(&QD>2!T>3$$-('-T>"!P,G@-('-T>2!P,GD-*J!P,SU;,: M,: Q70T@/CX^
M/B!N96<L83$$Q#2!S=&$@=&$-(#X^/B!N96<L9#2!S=&$@=&0-(#X^/B!N
M96<L9S,Q#2!S=&$@=&<-(&IS<B!P<F]J96-T#2!L9'@@=&QD>"!T.2D=F
M#2!S=&$@;&1A(#X^/B!N96<L:3,S#2!S=&$@=&D-(&IS<B!P<F]J96-T#2!L
M9'@@=&0@.W-Q5)T6'@P/CX^(&YE9RQE,C(-('-T82!T92 ^/CX@;F5G+&8R
M,R!S=&$@=&8-(#X^/B!N96<L9S,Q#2!S=&$@=&<-(&IS<B!P<F]J96-T#2!L
M<F]J96-T#2!L9'@@=&QD>"!T>#$-(&QD>2!T>3$-('-T>"!P,WL-('-T>2!P
M=68@;&1A('9M8W-B#2!E;W(@(R0P,B [<%)%5%19H%1224-+62R@14@_#2!S
M=&$@=FUC<V(-(&QD82 C)# X#2!E;W(@>G1E;7 @.UI414U0/4A)1TB@0EE4
M1:!*55-4H$9,25!3#2!S=&$@>G1E;7 @.T)%5%=%14Z@)#,PH$%.1* D,S@-
M#2!J;7 @;6%I;B [85)/54Y$H$%.1*!!4D]53D2@5T6@1T\N+BX-#0TJ+2TM
M+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+0TJH'1(25.@4U5"4D]55$E.
M1:!#04Q#54Q!5$53H%1(1:!04D]*14-424].H$]&H'B@04Y$H'D-#7!R;VIE
M8W0@8VQC#2!L9&$@=&<-(&%D8R!T: T@8VQC#2!A9&,@=&D@.W1(25.@25.@
M4D]4051%1*!Z#2!C;&,-(&%D8R C,3(X(#MW1:!!4D6@1T])3D>@5$^@5$%+
M1: Q,C@K6@TJH#X^/J!D96)U9V$L875X#2J@/CX^H&1E8G5G+"=!)PT@=&%X
M(" [;D]7H$E4H$E3H%)%0419H$9/4J!)3D1%6$E.1PT@;&1A('ID:78L>" [
M=$%"3$6@3T:@+40O6@T@<W1A(&%U>" [=$A)4Z!)4Z!&3U*@5$A%H%!23TI%
M0U1)3TX-('-T82!R96T@.VU53%1)4$Q9H$-!3J!#3$]"0D52H&%U> T-(&-L
M8PT@;&1A('1A#2!A9&,@=&(-(&-L8PT@861C('1C#2!S=&$@86-C(#MT2$E3
MH$E3H%)/5$%4142@6 T@:G-R('-M=6QT(#MS24=.142@355,5$E03%F@86-C
M*F%U>"\R7F]F9G-E= T@8VQC#2!L9&$@86-C#3IC;VYT,2!A9&,@(S8T(#MO
M1D93152@5$A%H$-/3U)$24Y!5$4-*J!S146@0D5,3U>@1D]2H%1(1:!!214%3
M3TZ@5TA9H%1(25,-*J!.15A4H$E.4U1254-424].H$E3H$-/34U%3E1%1*!/
M550-*J!T87B@H#MN3U>@>*!)4Z!8(0T@<W1A('1X,0T@8VQC(" [9$^@5$A%
MH%=(3TQ%H%1(24Y'H$%'04E.H$9/4J!Y#2!L9&$@<F5M#2!S=&$@875X#2!L
M9&$@=&0-(&%D8R!T90T@8VQC#2!A9&,@=&8-('-T82!A8V,@.W1(25.@25.@
M4D]4051%1*!9#2!J<W(@<VUU;'0@.W-)1TY%1*!-54Q425!,6:!A8V,J875X
M+S)>;V9F<V5T#2!C;&,-(&QD82!A8V,-.F-O;G0R(&%D8R C-C0@.V]&1E-%
M5*!42$6@0T]]/4D1)3D%410TJH&9/4J!33TU%H$-/35!,151%3%3%F@54Y+3D]7
M3J!214%33TZ@5$^@344-*J!42$6@24Y35%)50U1)3TZ@0D5,3U>@1$^@4TXG
M5*!73U)++J"@<T]-14-/1$5.4VH%1(1:!!4D5!.@25.@.@34]24924Y'H'B@04Y$
MH'D_/S\-*J!T87F@H#MS5$]21:!Y#2!S=&$@=&Q#2!R=',@.@(#MIH$A/
M4$$6@5$^@2$5#2Z!42$E3H%=/4D%"#@T-+*BTM+2TM+2TM+2TM+2TM+2TM
M+2TM+2TM+2T-*J!S;75L=#J@."U"252@4TE'3D5$H"A33U)4+4]&*:!!-
M54Q425!,60TJ#2J@86-C*F%U>"\R7F]F9G-E=* M/J!;86-C+*!E>'1=H* Q
M-BU"252@4D5354Q4H*!,3RQ(20TJ#2J@;D]]41:!!42$%$4H%1(25.@4D]]55$E.
M1:!!$259)1$53H%1(1:!%3D2@4D5354Q4H$)9H#)>;V9F<V5T#0TJH'E54"R@
M04Y/5$A%A%4J!-04-23RX-9&EV;V9F(&UA8R @.V1)5DE$1$$1!!"6:!!42$6@1DQ/
M052@3T9&4T54H$$1!44Q4H$9!0U1/4B!A8V,J875X(#-R15!%051$30#6$$1DQ/
M052@3T9&4T54H$%$1U435&D>=N#@,3T9&4T54H$%$1E-%5*!M86-R;R!T>7!E
M2!T>7!E#@T-"C)R!S=&$!@;&$*#U3-51!-(&QD8D!#&QD82!A97@-('-T82!A975X
M<V]A8V,J875X+T1)5BD-#0TJH&YO=&4Z(&5#R!J,$58Y*P.(&$.<<&%R=&(-
MH$%#0T=R("@Z(&%C8VMN(&$@>'@%A@0R!T9]X!%4T]-1$%!%U32%1A-;M>)P1R4D]-
M```
```

```
M2T6@0:!35$50H$E.H'@-(&-P>"!X,@T@8FYE('AL;V]P#2 \/#P-#7ES=&5P
M(&UA8R @.W-!346@5$A)3D<LH$)55*!&3U*@>0UY;&]O<"!I9B!I+%TQ#2!I
M;GD-(&5L<V4-(&1E>0T@8VQC(" [=D526:!)35!/4E1!3E0A#2!F:6X-(&%D
M8R!D> T@8F-C(&PR#2!I;G@@(#MA3%=!65.@24Y#4D5!4T6@> T@<V)C(&1Y
M#2 ^/CX@<&QO='!X#2!J;7 @;#,-;#(@/CX^('!L;W1P>2 [=T6@3TY,6:!)
M3D-214%3142@>0UL,R!C<'D@>3(-(&)N92!Y;&]O< T@/#P\#0TJ*BHJH&E.
M251)04R@3$E.1:!315154 T-9')A=R ^/CX@;6]V92QT>#$[>#$@(#MM3U9%
MH%-4549&H$E.5$^@6D523Z!004=%#2 ^/CX@;6]V92QT>#([>#(@(#MW2$52
M1:!)5*!#04Z@0D6@34]$249)140-(#X^/B!M;W9E+'1Y,3MY,0T@/CX^(&UO
M=F4L='DR.WDR#2 ^/CX@<V5T8G5F(#MN3U>@5T6@0T%.H$-,3T)"15*@5$A%
MH$)51D9%4@T-('-E8R @.VU!2T6@4U521:!8,3Q8,@T@;&1A('@R#2!S8F,@
M>#$-(&)C<R Z8V]]N= T@;&1A('DR(#MI1J!.3U0LH%-705"@<#&@04Y$H' R
M#2!L9'D@>3$-('-T82!Y,0T@<W1Y('DR#2!L9&$@>#$-(&QD>2!X,@T@<W1Y
M('@Q#2!S=&$@>#(-#2!S8F,@>#$@.VY/5Z!A/418#3IC;VYT('-T82!D> T@
M;&1X('@Q(#MP552@6#&@24Y43Z!X+*!.3U>@5T6@0T%.H%1205-%-(H'@Q#0UC
M;VQU;6X@;&1A('@Q(#MF24Y$H%1(1:!&25)35*!#3TQ534Z@1D]2H'@-(&QS
M<B @.RAT2$E3H$-!3J!"1:!-041%H$U50TB@1D%35$52(2D-(&QS<B @.W1(
M15)%%H$%21:!8,2\XH#$R.*!"651%4RP@4$$),3T-+4T@;'-R(" [=TA)0TB@345!
M3E.@6#$O,3:@,C4VH$)95$6@0DQ/0TM3#2!L<W(-(&)C8R!Z979E.B [=TE4
M2*!!H%!/4U-)0DQ%H$-!4E29H$%3H$1!3D0@(R0X," [
M24:@4T\LH%-%5*!42$6@2$E'2*!"250-('-T>2!B=69F97(-(&-L8PTZ979E
M;B!A9&,L@8G5F9F5R*S$@.V%$1*!)3J!42$6@3E5-0D52H$]&H#(U-J!"651%
MH$),3T-+4PT@<W1A(&)U9F9E<BLQ(#MA3D2@4U1/4D6@250A#0T@<V5C(#(!L
M9&$@>3(@.V-!3$-53$%412!$3U1/3T@0D]9-($%4#0T@;&-E<C!!,0R#MI4Z!9
M,,CY9,3\-(&QD82!Y,2 [;U1(15)(4D5'2D4RB!B2$5.($E(('0N!S8F,@,5Q;'0T@
M;&ES=&-B=7IF97(K,0T@;&1A('AG9F5R*S$@<W1A(&)U9F9E<BLQ#0T@;&1A
M>3(@.VYE=F5R($9!5$%3*!1=6%N(&-T:6$@;&ES=$%R(&%T(&-H86YG92!O
```

```
M2T6@0:!35$50H$E.H'@-(&-P>"!X,@T@8FYE('AL;V]P#2 \/#P-#7ES=&5P
M(&UA8R @.W-!346@5$A)3D<LH$)55*!&3U*@>0UY;&]O<"!I9B!I+%TQ#2!I
M;GD-(&5L<V4-(&1E>0T@8VQC(" [=D526:!)35!/4E1!3E0A#2!F:6X-(&%D
M8R!D> T@8F-C(&PR#2!I;G@@(#MA3%=!65.@24Y#4D5!4T6@> T@<V)C(&1Y
M#2 ^/CX@<&QO='!X#2!J;7 @;#,-;#(@/CX^('!L;W1P>2 [=T6@3TY,6:!)
M3D-214%3142@>0UL,R!C<'D@>3(-(&)N92!Y;&]O< T@/#P\#0TJ*BHJH&E.
M251)04R@3$E.1:!315154 T-9')A=R ^/CX@;6]V92QT>#$[>#$@(#MM3U9%
MH%-4549&H$E.5$^@6D523Z!004=%#2 ^/CX@;6]V92QT>#([>#(@(#MW2$52
M1:!)5*!#04Z@0D6@34]$249)140-(#X^/B!M;W9E+'1Y,3MY,0T@/CX^(&UO
M=F4L='DR.WDR#2 ^/CX@<V5T8G5F(#MN3U>@5T6@0T%.H$-,3T)"15*@5$A%
MH$)51D9%4@T-('-E8R @.VU!2T6@4U521:!8,3Q8,@T@;&1A('@R#2!S8F,@
M>#$-(&)C<R Z8V]V]N= T@;&1A('DR(#MI1J!.3U0LH%-705"@<#&@04Y$H' R
```

by Craig Bruce <csbruce@ccnga.uwaterloo.ca>

0. PREFACE

I originally planned to write this entire article all in one go, but its size, complexity, and scope of required design decisions have forced me to split this article into two pieces. (Not to mention my own poor time management in writing this article). This part gives an introduction to what I am talking about and discusses, at an abstract level, how the system will work. The next part will dive into all of the nuts and bolts of the low-level design.

Also, this article may be a bit to weird for some people to grasp. Please bear with me. This article is as much a scratchpad for my rough ideas about the kind of system I want to build as it is an explanatory article. You may need a Master's degree in software systems to understand some of the things I talk about. This article makes references to the ACE operating system, which is available via anonymous FTP from "ccnga.uwaterloo.ca". ACE is a uni-tasking OS that has a Unix-like flavor. (Yeah, yeah, yeah, I'm still working on the next release...).

One more note about the article: it is written in the present tense ("is") rather than the future tense ("will"), since the present tense is easier to read and understand. The system, however, does not presently exist and the design may change in many ways if the system ever is made to exist.

1. INTRODUCTION

The full title of this article should be "Design of a Multitasking Distributed Microkernel Operating System for the Good Old '128". For purposes of discussion, we will call the new operating system "BOS". A "multitasking" operating system (OS) is one that is able to execute more than one "process" "concurrently". A Process is an instance of a running program. "Concurrently" means that the programs appear to be running at the same time, although in reality they are not, because there is only "one" processor in the 128 and it can only do one thing at a time.

A "distributed" OS is one that runs on a collection of independent computers that are connected by a network. Unlike a "network" OS, a distributed OS makes all of the independent computers look like ONE big computer. In general, a distributed system, as compared to a centralized one (like MS-DOS or Unix), gives you "a higher performance/price ratio ('more bang for the buck'), potentially increased reliability and availability because of partial failure modes (if protocols are implemented correctly), shared resources, incremental growth and online extensibility, and [a closer modelling of] the fact that some applications are inherently distributed." This is quoted from my Ph.D. thesis about distributed systems of powerful workstations. To us, a distributed system means increased modularity and ease of construction, sharing devices like disk drives and resources like memory between multiple computers, and the true parallelism of running different processes on multiple computers at the same time. Not to mention "coolness".

A "microkernel" OS is one that has the smallest kernel possible by pushing higher-level functionality (such as the file system) into the domain of user processes. The kernel ends up being small, fast, and easy to construct (relative to a monolithic kernel).

So why would we want our OS to have the features of Multitasking, Distributed, and Microkernel? Because I'm designing it, and that's what interests me. The ease-of-construction thing is important too. Another important question is "can it be done?". The answer is "yes." And it will be done, whenever I get around to it (one of these lifetimes).

2. GENERAL DESIGN OVERVIEW

There are a number of high-level design decisions that must be made before going into a detailed design. This section discusses these decisions.

2.1. SPECIAL C-128 FEATURES

The C-128 has a minumum set of special features that make it feasible to run a multitasking operating system, as opposed to earlier machines like the C-64. The simplest special feature that the C128 has is *enough memory*. The 64K of the C64 just isn't enough. The 128K of the C128 is just barely enough. Expanded internal memory makes the proposition even easier.

The C-128 also has relocatable zero-page and stack-page pointers. This feature are absolutely essential and you could not make an effective multitasking OS for any 6502 machine without it. I wonder if Commodore thought about this prospect when designing the MMU chip...

The last C-128 feature is *speed*. The C128 has a 2 MHz clock speed when used with the 80-column VDC display. This is enough speed, when harnessed properly, to make your applications zip along. For an example of speed that is not harnessed properly, see Microsloth Windoze. The VDC display is also very nice, too. Only the VDC display should be supported by a "real" OS, not the VIC display.

2.2. NETWORK

The OS should be designed to run on a system of between 1 and N C-128's, where N has a maximum of something like 8 or 16. We'll choose 16 for our software design. The theory is that the style of operating system that we are proposing makes the step between 1 and N C-128's a (relatively) easy one, so why not go for it. Also, if N were to become some number like 256 or 65536, then we could start kicking some serious ass performance-wise, for certain classes of computations. Also, I happen to own two C-128's and I have already constructed a parallel-port network (a Jedi's weapon!), so I might as well use it.

The required network connects the user ports of C-128's into a bus. I'm not completely sure how to connect more than two C-128's to this bus (I'd probably need some diodes or logic gates), so the initial version of this network hardware will have a maximum of two hosts. We will still be careful to make the software of the system easily reconfigurable for any number of hosts.

You will need two appropriate connectors and some 14-conductor ribbon cable to build the network. One of my connectors is a 44-conductor connector of the type used with the VIC-20 expansion port that I sawed in half and the cable is some old junk ribbon cable that was lying around that I removed some of the conductors from. Any old junk will do. You're probably best off if your cable is less than six feet long (2 metres). The network is wired up as follows:

```
C128-A  name/pin                                      pin/name  C128-B
         GND <A>+----------------------------------------+<A> GND
        FLAG <B>+----------------------------------------+<8> PC2 ***
         PB0 <C>+----------------------------------------+<C> PB0
         PB1 <D>+----------------------------------------+<D> PB1
         PB2 <E>+----------------------------------------+<E> PB2
         PB3 <F>+----------------------------------------+<F> PB3
         PB4 <H>+----------------------------------------+<H> PB4
         PB5 <J>+----------------------------------------+<J> PB5
         PB6 <K>+----------------------------------------+<K> PB6
         PB7 <L>+----------------------------------------+<L> PB7
         PA2 <M>+----------------------------------------+<M> PA2
         GND <N>+----------------------------------------+<N> GND
        CNT2 <6>+----------------------------------------+<6> CNT2
         SP2 <7>+----------------------------------------+<7> SP2
         PC2 <8>+----------------------------------------+<B> FLAG ***
```

Here is the Commodore 128 User Port when looking at the back of the unit:

```
                        111
               123456789012    top
               ------------
               ABCDEFHJKLMN    bottom
```

This gives a parallel bus that can operate at a peak of about 80 kiloBYTES/sec with a shift-register serial bus thrown in that can operate at a peak of about 21 kiloBYTES/sec. Both communication channels are uni-directional, so some media-access-control protocol will need to be provided by software. The price, in terms of hardware for using this network, is that you can't use a modem that plugs into the user port at the same time. Of course, any serious user will have a modem that plugs into a UART card anyway.

You can also write your own applications for this network, since programming it is quite easy; the hardware takes care of all of the handshaking. To blast 256 bytes over the network from C128-A to C128-B, you would:

```
C128-A: sender                              C128-B: receiver
==============                              ================
  lda #$FF   ;ddr-output                      lda #$00   ;ddr-input
  sta $DD03                                   sta $DD03
  ldy #0                                      ldy #0
- lda DATA,y ;get data                      - lda #$10   ;wait for data
  sta $DD01  ;send data                     - bit $DD0D
  lda #$10   ;wait for ack                    beq -
- bit $DD0D                                   lda $DD01  ;receive data/send ack
  beq -                                       sta DATA,y ;store data
  iny        ;next                              iny
  bne --                                        bne --
  rts                                           rts
```

These routines can even be tweaked a little more for higher performance. Programming the shift register is analogous to the above.

There is probably no need to do error checking on the data transmitted over the network since the cable should be about as reliable as any of the other cables hanging out the back of your computer (and none of them have error checking (except maybe your modem cable)).

2.3. PROCESSES

A process is a user program that is in an active state of execution. In uni-tasking operating systems like ACE or the Commodore Kernal, there is only one process in the entire system. In a multi-tasking system, there are, duh, multiple processes. Each process executes as an independently running program, in isolation, logically as if it were the only process in the system. Or, as if there were N 8502's available inside of the 128 and one of them were used to run each program you have loaded.

In reality, there is only 1 CPU in the 128 (well, that we are interested in using), so its time is divided up and given out in small chunks to execute so many instructions of each program before moving onto the next one. The act of changing from executing one program to executing another is called "context switching", and is a bit of a sticky business because there is only one set of processor registers, so these must be saved and restored every time we switch between processes. Effectively, a process' complete "state" must be restored and saved every time it is activated and deactivated (respectively). Since the 8502 has precious few internal registers, context switching can be done quite efficiently (unlike with some RISC processors). The maximum period of time between context switches is called the "quantum" time. In our system, the quantum is 1/60 of a second. It is more than just a coincidence that this period is the same as the keyboard-scanning period. Depending on priorities and ready processes, a new or the same old process may be selected for execution after the context switch of the 60-Hz interrupt.

Splitting the time of one processor among N processes may sound like we're simply making each one run N times slower, which may be unbearably slow, but that is not generally the case. One thing that a CPU spends a lot of its time doing is *waiting*. Executing instructions of a program requires the full attention of the CPU, but waiting requires absolutely no CPU attention. As an example, your speedy computer spends a lot of its time waiting for its slow-as-molasses-launching-into-orbit user to type a key. If we were to put the process that asks the OS for a keystroke into a state of suspended animation, then the CPU time that process would have consumed in a busy-waiting loop can be better spent on executing the other processes that are "ready" to execute. In practice, many processes spend a lot of their time waiting, so "multi-programming" is a big win.

There are a number of things other than keystrokes that processes may wait for in our envisioned system: modem characters, disk drive operations (if they are custom-programmed correctly), mouse & joystick movements, real-time delays, and interactions with other processes. The OS provides facilities for processes to communicate with one another when they cannot perform some operation in isolation (i.e., when they become lonely).

A process has the following things: a program loaded into the internal memory of the 128, its own zero page and processor stack page, and the global variables of its program. A process can also own "far" memory (below) and various other resources of servers throughout the distributed system. The process is the unit of ownership, as well as execution. Processes also have priorities that determine how much execution time they are to be given relative to other processes in the system.

Processes are allocated memory at the time of startup at a random location on some random bank of internal memory on the 128. The biggest challenge here is to relocate the user program to execute at the chosen address. The kernel interface is available to programs on all internal banks of memory.

2.4. APPLICATION PROGRAM INTERFACE

To take advantage of existing software, we would like our OS to provide an application-program interface (API) that is identical to that of the ACE-128/64 operating system. In fact, this is the *real* reason why ACE was developed -- as a stepping stone toward a real operating system. The ACE Programmer's Reference Guide, which describes the API, is available from "ccnga.uwaterloo.ca".

Some useful software already exists for ACE, and ACE has a well-definied interface and well-behaved programs. The ACE interface may need to evolve a little too. The ultimate goal would be to have the same API for both systems so you could run software with the more functional BOS if you have a C128 and 80-column monitor, or you could use the less functional ACE if you didn't have all this hardware.

The software wouldn't be "binary-identical" since the operating systems provide quite different program environments and requirements, but the two systems should be application-source-code compatible.

Because of the vast differences between a microkernel and a monolithic kernel, all of the ACE system calls would be redirected to user-library calls in BOS. This user library would then carry out the operations accessing whatever system services are needed.

2.5. MEMORY MANAGEMENT

The memory management of BOS is analogous to that of ACE. There are two different classes of memory: near and far. Near memory is on the same bank as a program and can be accessed directly by processor instructions. Far memory can only be accessed through the kernel by the special kernel calls Fetch and Stash and must be specially allocated to a process by the operating system. Note that near memory is considered a sub-class of far memory; the far-memory primitives can be used to access near memory.

Only the basic memory-accessing code is provided by the kernel; higher-level memory management, such as dynamic memory allocation and deallocation, is handled by the Memory Server (below).

Unlike ACE, BOS provides the fundamental concept of "distributed memory". The Fetch and Stash primitives can also access the memory of a remote machine in a completely user-transparent way. Thus, a far-memory pointer can be passed between processes on different machines, and the memory that the pointer refers to can be read and written with equal programming by both processes. This feature can be dangerous without a synchronization mechanism, so this memory sharing is intended to be used only with the communication mechanism.

There should not be an unacceptable overhead in accessing remote memory on the 128 (like how there would be with bigger computers) because far-memory fetching for local memory is quite expensive anyways (relative to near memory), so an application will optimize its far memory accessing, and the necessary interrupt handling on the remote machine can be done with very little latency because of the "responsiveness" of the 6502 processor design.

2.6. COMMUNICATION

In the type of system that is envisioned, processes are not strictly independent and competitive; many must cooperate and comunicate to get work done. To facilitiate this interprocess communication (IPC), a particular organization is chosen: the Remote Procedure Call (RPC) paradigm. RPC is a message-passing scheme that is used with the heavily hyped Client/Server system architecture model. It reflects the implicit operations that take place when you call a local procedure (a subroutine): the call, the entry, the processing, and the return. The kernel provides three primitives for RPC:

```
Send( processId, requestBuffer, reqLength, replyBuffer, maxRepLength ) : err;

Receive( ) : processId, requestBuffer, reqLength, replyBuffer, maxRepLength;

Reply( processId ) : err;
```

Send() is used to transmit a message to a remote process and get back a reply message. The sending process suspends its execution while it is waiting for remote process to execute its request. A message consists of an arbitrary sequence of bytes whose meaning is completely defined by the user. The message contents are stored in a buffer (hunk of memory) before sending, and a length is specified at the time of sending. A buffer to receive the reply message must also be allocated by the sender and specified at the time of sending. To save us from the overhead of copying message contents to and fro unnecessarily, only pointers to the buffers are passed around and the far memory primitives are used to access message contents. This also works across machine boundaries because of the distributed-memory mechanism described above.

Receive() is used to receive a message transmitted by a remote process to the current process. The receiver blocks until another process does a corresponding Send() operation, and then the request and reply buffer pointers and lengths are returned. The receiver is expected to fetch the contents of the request message, process the request, prepare the reply message in the far-memory reply buffer, and then execute the Reply() primitive. There are no restrictions on what the receiver can do between receiving a message from a process and issuing the corresponding reply message. So, it could, for example, receive and process messages from other processes until it gets what it needs, compute pi to 10_000 decimal places, and then reply to the process that sent a message to it a long time ago.

Reply() is used to re-awaken a process that sent a message that was Receive()d by the current process. The current process is expected to have set up the far-memory reply buffer in whatever way the sending process requires prior to issuing the Reply().

The expected usage of buffers is for the sender to use near memory for the request and reply buffers and access them as regular near memory to construct and interpret request and reply messages. The receiver will access the buffers as far memory (which they may very well be since processes are allowed to execute on different banks of internal memory and even on different machines), and may wish to fetch parts of messages into near memory for processing. The use of far pointers makes it so that data is copied only when necessary.

And that's it. You only have this RPC mechanism for communicating with other processes and for all I/O. Well, that's not entirely true; the RPC stuff is hidden behind the application program interface, which provides such facades as the Open and Read system calls, and a very-low level interrupt notification mechanism which a user process will not normally use.

2.7. SYSTEM SERVERS

Since all that user program has for IPC and I/O is the RPC mechanism, a number of system server processes must be set up to allow a user program to do anything useful. These special servers execute as if they were regular user programs but provide service that is normally implemented directly into the operating system kernel. There are a number of advantages and disadvantages to organizing a system in this way. A big advantage is that it is easier to build a modular system like this, and a big disadvantage is that you lose some performance to the overhead of the IPC mechanism.

A useful implication of using servers rather than having user processes execute inside of the kernel is mutual exclusion. Servers effectively serialize user requests. I.e., user requests are serviced in order, strictly one-at-a-time. This is important because some of variables that need to be manipulated in order to provide service must not be manipulated by multiple processes simultaneously or you may get inconsistent results. To provide mutually exclusive access to shared variables in a monolithic system, either ugly and problematic semaphores must be used, or more-restrictive, simpler mechanisms like allowing only one user process to enter the kernel.

2.7.1. PROCESS SERVER

This server is responsible for starting and terminating user processes. Because of the way that the procedure is organized, the process server is actually quite responsive dispite all of the work that must be done in order to start up and terminate a user process.

The server is highly integrated with the kernel, and it is able to do things that regular user processes cannot (like manipulate kernel data structures), but it still functions as an independent entity, as a regular user process. Its code is physically a part of the kernel for bootstrapping purposes, since it can hardly be used to start itself.

When you wish to run a new program, a request message is sent to the process server. This message includes the filename of the program to run, the arguments to the new program, environmental variables, and a synchronous/ asynchronous flag. If you want to run a sub-process synchronously, the process server does not reply to your request until the new process terminates. If you select asynchronous mode, the process server replies to your request as soon as the new process is created. Both of these modes are quite useful in Unix (although Unix has a more complicated mechanism for providing the service) (think "&" and no-"&" on command lines), so they are provided here.

The process server allocates and initializes the kernel data structures necessary for process management, and then starts the process running bootstrapping code in the kernel. Since this code is in the kernel, it is known to be trustworthy. The process then bootstraps itself by opening the program file, reading the memory requirements, allocating sufficient memory, reading in the program file, relocating the program for whatever memory address it happened to load in at (bank relocation is no problem) and far-calling the main routine (finally). The return is set up on the stack to kill the process.

Since the process bootstraps itself, the process server's involvement in the process creation procedure is minimal, and the process server is ready to process new requests with minimal delay (maximal responsiveness). This self-bootstrapping user process concept comes from my Master's Thesis. Another advantage of having a process server is that you can start a process running on any machine from any other machine in exactly the same way you would start a process on the local machine; we have achieved transparentness, Park.

The process server also takes care of process destruction (exit or kill) and provides other less-significant services, like reading and setting the current date and time. The mechanism by which process destruction is done is similar to the self-bootstrapping idea and is discussed, probably inappropriately, in the next section.

The server is located by having a well-known address. That is, the process id is a constant and hard-coded into clients. Well-known addresses are small integer values, for each machine (a machine-id is encoded into process ids), and these integers are indexes into a small look-up table with the actual addresses for well-known addresses, so the process ids aren't pinned but can be used as if they were pinned.

2.7.2. MEMORY SERVER

The memory server handles the dynamic allocation and deallocation of far memory. The client specifies in the request message the exact types of memory that it can use, and the server gets the memory, sets the ownership to the process, and returns a pointer. Deallocation of some of the memory owned by a process is handled easily.

There is also a call that deallocates all memory owned by a certain user process. This call is normally only called by the process server*, since the memory of the user program is be deallocated along with the rest of the process' memory. A record is kept internally for each process about what types and banks (later) of memory it has used so that bulk deallocation can be done efficiently when the process exits.

A client process can also ask that far memory be allocated on a remote machine. Remote memory is relatively slow to access, but it can be convenient when you need LOTS of memory for a process. The obvious way to get at this remote memory is to simply send a message directly to the remote memory server of the machine you want to allocate memory on, and this does indeed work, so this is what we will do. But, this doesn't record the fact that you have allocated memory on a far machine by itself, and we don't want to waste any effort in freeing all of the memory, both local and remote, that a process owns when it terminates; i.e., we don't want to send deallocation requests to all remote memory servers just to be sure.

There are a few alternatives for solving this problem, but I think this is a good place for a quick-and-dirty hack. Whenever a user process sends a message to a memory server (both local or remote, for whatever reason), through the memory servers' well-known addresses, the bit corresponding to the machine number (0-15) in a special 16-bit field of the sender's process control block is set. Then, when the process terminates, the termination procedure (next) peeks at this special field and sends free-all messages to all remote memory servers that the process in question has interacted with. This insures that all memory in the entire distributed system that is allocated to a process is tidied up when the process terminates. Like the process server, the memory server is integrated with the kernel.

MORE PROCESS TERMINATION

Come to think of it, I should talk more about process termination. The best idea would probably be for a user process to terminate itself, in the same way that it bootstraps itself. A termination message is sent by a client process that wants to kill someone to the process server. It is a valid situation for a process to commit suicide. The termination message includes the process id to be terminated and the exit code for the termination.

The process server then suspends the doomed process' execution and rigs the process' context so that the next thing it executes is the process shutdown code inside of the kernel. This shutdown code closes all of the files that the process has opened through the standard library calls (and other server-resources held), deallocates all memory held by the process, maybe does some other cleanup work, and then sends a special message to the process server to remove the process control block. The process server will only accept this special message from the process that is terminating after the first phase of the process shutdown has been completed, to insure a proper termination. The process control block is then deallocated and may be used again. The process server is the only process that is allowed to manipulate process control blocks.

Come to think of it, there is a slight problem with process initialization: getting a copy of the arguments and environmental variables for an asynchronously started new process. We don't want the sender to continue before the new process has had a chance to make a copy of the arguments and environment, so we will rig things so that it is the newly started process that sends the reply message back to the parent process. Another dirty hack.

2.7.3. FILE SERVERS

Each disk drive in the system has a special server that provides an interface for executing Open, Read, Write, Close, and a number of other common file operations. A big problem with distributed operating systems is resource reclamation for processes that die. There are a few ways to provide this, and each has implications about the overall design of a server.

One possibility is to have "stateless servers". In other words, each server does not keep track of, for example, which files a process has open or the current file positions. Each time a read request comes in, the server opens the file to be used, positions to the section of the file, performs the operation, and closes the file again. This sounds like a lot of work, but some intelligent caching makes it work efficiently. And if a user process dies without closing all of its files, it doesn't matter since the files will be closed anyway, logically at the completion of each operation. But, this approach doesn't really work well with Commodore-DOS, which we will be using for devices for which we don't have a custom device driver, so we won't use it.

Another possibility is to have "semi-state" or "acknowledgementless" servers (my own invention). Here, the server keeps track of, for example, which files are open but doesn't keep the file position. When a request comes in, the already-opened file is positioned according to the request and the file operation takes place. If a client dies unexpectedly, the open file control block (FCB) is left behind, but the FCB will be closed and reused after a certain period of time. If the client actually hasn't died, then the situation will be detected (through details not explained here) and the file will be reopened as if nothing has happened. Other contingencies like a dead process' name being reused are handled too. And the model works well with an unreliable communication service. But, again, this doesn't model the Commodore-DOS very well.

The final design considered is to have a registry of servers that that a process has resources currently allocated on be associated with each process. When a client makes an open request to the server (or some equivalent resource-grabbing operation), the server checks to see if the client is currently holding any other of the server's resources. If so, then the request is processes normally. If not, then the server (or some agent on the server's behalf) sends a message to the process server on the client's machine telling the process server to record the fact that the client is (or may be) holding some of the server's resources. The process server records the server's process id in the process control block of the client, and when the client terminates, it will send a standard "release all of the resources that I am (may be) holding on this server" to the server as part of the client's shutdown procedure. All of the client's open files will be closed, etc.

In this "registry" design, servers can be completely "stateful", e.g., they would contain both an open file entry and the file position information, and files would always open and close when we intuitively expect them to. It is assumed that the communication mechanism is reliable, which it is here. This mechanism *does* model Commodore-DOS well. In fact, this idea is so nice that I may redesign the memory allocation recovery mechanism to use this. There is a slight possibility of a "race condition" in this mechanism, but nothing bad can happen because of it. (This is just a note to myself: make it so that if a process is killed while it is receive- or reply-blocked, then ignore the reply from the server if the process id is reused; damn, there's still a potential problem; I'll have to figure it out later; also watch out for a distributed deadlock on the PCB list).

So, our server supports the regular file operations and implements them in pretty much the expected way, since it is a "stateful" server. The main loop of the server accepts a request, determines which type it is, extracts the arguments, calls the appropriate local procedure, prepares the reply message, replies, and goes back to the top of the loop. Each opened file is identified by a user process by a file control block number that has meaning inside of the server, as per usual. But, unlike with ACE, we need a special "Dup" operation for passing open files to children. Dup increments the "reference count" of a FCB, and the reference count is decremented every time a close operation takes place. A file will only be "really" closed when the reference count reaches zero. Our system will not implement any security at this time.

Because of the abstraction of sending formatted messages to a server, different types of disk drives (Commodore-DOS, custom-floppy, ramdisk) are all dealt with in exactly the same way. As one slight extension, we have to hack our devices (at least some of them) a little to be able to handle "symbolic links" in order to integrate well with the Prefix Server which is described next.

2.7.4. PREFIX SERVER

The prefix server idea is stolen from the computer science literature about a network operating system called "Sprite". The prefix server simply provides a pathname lookup service for the pathnames of different disk-file and device servers. This is needed to provide a single, global, unified pathname space on a system of multiple distributed file servers. It works a lot like the "mount table" in Unix. Its prefix table looks something like the following:

```
PREFIX      SERVER
------      ------
/           <1:ramdisk>
/dev/tty0   <1:console>
/fd1        <2:floppy1571>
```

BTW, BOS uses Unix-style filenames rather than the Creative-Micro-Designs-style filenames that ACE uses.

If an application is given an absolute pathname, it will consult the prefix server to resolve it to the process-id of an actual server. For example, the pathname "/fd1/bob/fred" would resolve to server "<2:floppy1571>", relative pathname "bob/fred". Pathname "/" would resolve to server "<1:ramdisk>", relative pathname "".

The user process would then contact the appropriate server with the relative pathname. A user process can assume that the prefix table will not change while the system is running, so some intelligent caching can be done. Also, directory tokens are given out for executing a "change directory" operation, and these server/token pairs can be used for quick relative pathname searches. A symbolic link mechanism is needed to insure that these relative searches always follow through correctly.

2.7.5. DEVICE SERVERS

Device servers are just another type of file server, except they control a specific device other than a regular disk device, and they are likely to support some custom operations and return error codes if some disk operations are attempted. The interface is identical to a file server for convenience.

2.7.6. CONSOLE SERVER

Just a specific device server. It handles window management and console calls, like WinClear, WinPut, GetKey, and ConWrite, that are used in ACE.

2.8. ASYNCHRONOUS EVENT HANDLING

As mentioned in the Process section above, there are many external events that a process may have to wait for, including: modem characters, disk drive operations (if they are custom-programmed correctly), mouse & joystick movements, and real-time delays. There will be an AwaitEvent() kernel primitive to allow a process to wait for one of these events to happen. Normally, the only processes that wait for these events will be device drivers. The kernel will also have to do some low-level processing for of some devices (like the modem and keyboard) to insure that things don't become unnecessarily inefficient.

3. KERNEL DESIGN

Next time.

4. SYSTEM SERVER DESIGN

Next time.

5. APPLICATION PROGRAM INTERFACE

Next time.

This is quite similar to the ACE-128/64 Programmer's Reference Guide, which is available via anonymous FTP from "ccnga.uwaterloo.ca" in file "/pub/cbm/os/ace/ace-r10-prg.doc". Release #10 of ACE was the most current at the time of writing this article.

6. CONCLUSION

Next time.

Implementation: someday, maybe.

===---END---===