@(#)top:

```
                   ########
             ##################
         ######            ######
      #####
    #####  ####  ####      ##       #####   ####  ####  ####  ####  ####   #####
  #####    ##    ##      ####     ##   ##   ##  ###     ##    ####  ##   ##   ##
 #####    ########     ##  ##    ##        #####       ##    ## ## ##   ##
#####    ##    ##    ########   ##   ##   ##  ###     ##    ##  ####   ##   ##
#####  ####  ####  ####  ####   #####   ####  ####  ####  ####  ####   ######
#####                                                                     ##
 ######            ######            Issue #12
   ##################               Version 1.3
       ########                      March 1996
```

@(#)contents: Table of Contents

@(#)legal: Commodore Hacking Legal Notice

## @(#)info: Commodore Hacking Information

Commodore Hacking is published via the Internet 4 times yearly, and is
presented in both ISO-8859-1 and HTML versions. This and previous issues can
be found at the Commodore Hacking Home Page
(http://www.msen.com/~brain/chacking/), as well as via FTP
(ftp://ccnga.uwaterloo.ca/pub/cbm/hacking.mag/)

In addition, the Commodore Hacking mail server can be used to retrieve each
issue. To request a copy of an issue, please send the following electronic
mail message:

```
To: brain@mail.msen.com
Subject: MAILSERV
Body of Message:

help
catalog
send c=hacking12.txt
quit
```

To retrieve a PKZIP 1.01 archive of the individual articles in Commodore
Hacking, request the file c=hacking12.zip

To subscribe to the Commodore Hacking and receive new issues as
they are published, add the following command to you MAILSERV message
prior to the quit command:

```
subscribe c=hacking Firstname Lastname msglen
```

(msglen is largest size of email message in line you can receive. Each
line is roughly 50 characters, so 600 lines is about 30000 bytes. When
in doubt, choose 600)

example:

```
subscribe c=hacking Jim Brain 600
```

Although no fee is charged for this magazine, donations are gladly accepted
from corporate and individual concerns. All monies will be used to defray
any administrative costs, subscribe to publications for review, and
compensate the individual authors contributing to this issue.

Any persons wishing to author articles for inclusion in Commodore Hacking are
encouraged to view the submission guidelines on the WWW
(http://www.msen.com/~brain/pub/c-hacking-submit.txt) or via the MAILSERV
server (send c-hacking-submit.txt).

==

## @(#)rch: Reading C=Hacking

Starting with Issue 11 of Commodore Hacking, the new QuickFind indexing system is utilized to aid readers of the text version in navigating the magazine. At the top of each article or other important place in the magazine, a word prefixed with a special string is present. (See the title of this article for an example. Throughout the magazine, if an article is mentioned, it will be followed by a reference string. For example, if we mentioned this article, we would add (Reference: rch) after the name. By using your favorite editor's search function and searching for the string after the word "Reference:", prefixed by the magic prefix string, will move you directly to the article of choice. To merely skip to the next article in the magazine, search only for the magic prefix string.

Some handy indexing strings possibly not referenced anywhere are:

top	top of issue
bottom	bottom of issue
contents	table of contents
legal	legal notice

For those with access to a UNIX system, the command "what" can be run on the issue, which will result in all the article titles being printed.

A slightly different magic prefix string "@(A)" is used to delimit sub-topics or main heading in articles. The text after the magic string differs depending on article content. For the Input/Output column (Reference: io), the text after the magic prefix will either be "c" for comment, or "r" for response. In features and columns, a number after the prefix indicates the ordinal of that heading or sub-topic in the article. If a specific sub-topic is referenced elsewhere in the article, a sub-topic reference will be indicated. A reference to "@(A)r" would be written as "(SubRef: r)".

As time goes on, the role of this indexing system will be expanded and changed to ease navigation of the text version, but minimize the clutter added by these extra items.

==

## @(#)editor: The Hacking Editor

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

Speed and the Web. The owners are asking about it. The developers are looking into it. The market is readying itself for it. No, not the PC market, I'm referring to the Commodore 8-bit market. The same market usually referred to as "mature". The same market usually referred to with a condescending tone. Well, mature we might be, but isn't that considered a good thing? People are supposed to mature as they grow older. As such, they are revered and looked up to. What parallels can we draw here?

If you haven't anything about the planned introduction of the CMD SuperCPU 20 MHz accelerator cartridges for the C64 and C128, shame on you! You need to stay in touch more. For those who have, let's not overdo the hype. CMD isn't the first to produce such a cartridge, but they will be the first to introduce 20 MHz as a speed option. C128 users will rejoice as the first 128 mode accelerator ships from CMD. When this happens, performance approaching that of the venerable Intel 80386 will be as close as the on/off switch.

The explosion of interest in the Internet and the World Wide Web is changing the way people view computers. Until recently, it seemed that people thought only computer systems including a 32 or 64 bit CPU, multiple megabytes of RAM, gigabyte hard drives, infinite resolution monitors and million bit sound cards were worth owning. Commodore owners have felt the sting of ridicule as they continually take blow after blow for remaining loyal to a machine with much to offer. Well, be patient, because 1996 might be the year of the "network computer", a smaller comuter system that trades all the fancy features of bloated PCs for a smaller size, cost, and a connection to the Internet. Big names like IBM, Oracle and Apple are pushing this idea, which would bring to market systems with modest RAM, small drives, television displays, and small operating systems. Does this idea sound familiar? It should, as it describes many features of Commodore 8-bit systems. No, the CBM 8-bit still lacks a few items present in the IBM/Apple/Oracle designs, but the bulk of features are already available on your so called "obsolete" CBM machine. Don't gloat yet, as there's much to do, but if your friends tout the benefits of such a machine, gently remind them that you own of of the first and best, a Commodore 8-bit.

Enjoy YOUR magazine,

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

==

@(#)io: Input/Ouput

Obviously, Commodore Hacking depends on the comments and article submissions from the Commodore community to flourish. Everyone sees the articles, but let's not forget those comments. They are very helpful, and every attempt is made to address concerns in them. Address any comments, concerns, or suggestions to:

Commodore Hacking
602 N. Lemen
Fenton, MI 48430
brain@mail.msen.com (Internet)

@(A)c: Time Travellin'

From: Robin Harbron <rfharbro@flash.lakeheadu.ca>

Dear C=Hacking,
I was looking at the Commodore Hacking page (fantastic magazine) and noticed that the "Publishing Schedule for 95-96" has 1995 for all the dates, while I assume that the last 4,5,or 6 probably should be 1996.
Thanks for everything!

Robin

@(A)r:
Yep, we must have just returned from our time travel experiments when we wrote that in the WWW pages. Note that the magazine was correct, but the home was in error. Oh well.

@(A)c: Run (Down) the Software

From: sis319@educ.di.unito.it

Dear C=Hacking,
I really appreciate the work you are doing with the Commodore Hacking on-line magazine. I like the new look and the new features you added, such as newsfront and hacking the mags.

I would like to see on the magazine some reviews about the latest and more interesting PD and Shareware software (with a list of FTP sites where these are available) and hardware products.

Please note that Commodore Hacking is the only Commodore magazine I can easily find here in Italy, because all the Italian magazine there were are dead and all the foreign magazines that were distributed, such as "RUN", "Compute!'s Gazette", "64'er" are either dead or no longer distributed.

@(A)r:
We appreciate your comments about the new look of C=Hacking. As for the inclusion of reviews, we're looking into it. it's not that we don't want to do it, just that we need to schedule the reviews (Commodore Hacking shouldn't do all of them, as that creates bias), and determining what software is worthy of review. Look for some reviews in upcoming issues.

@(A)c: Separate But Equal

From: alan.jones@qcs.org (Alan Jones)

Dear C=Hacking,
I like your new version of C=Hacking. I like the idea of including relevant news and summaries of other magazines and disks. Size should not be a constraint, although you should publish early when it exceeds some critical size. Don't scrimp on source code listings and uuencoded files! There is no other publication for this sort of bulky technical stuff.
It would also be wonderfull if we could get an apropriate means for including diagrams or pictures, viewable by C64/128 users. I would REALLY like to have the C64/128 html viewer/printer that you mentioned. You may not know it but we came very close to having Al Angers Tower article submitted to C=Hacking in place of _Commodore World_, but C=Hacking could not really handle drawings and photos.

I have been separating C=Hacking into separate articles and files, archiving them and placing the archive(s) on a local BBS. This compacts the length and makes it easier to read and use. I try to make C=Hacking easy to

download and use locally, but I still want to keep it as whole and original
as possible.

@(A)r:
Alan, we're glad you approve of the new format. We're going to try to keep
the size so that it will always fit onto 2 1541 disk sides. C=Hacking
is still working on the HTML viewer, but it's taking a back seat to other
more pressing issues. We'll have it finished at some point, and start
distributing the magazine that way as well.   As for your separation, we
appreciate the work you've done to make C=Hacking easier to distribute.
With issue #12, we are offering an archive of all the article in separate
files. The archival method has not been chosen just yet, but look on the
C=Hacking MAILSERV server for the file.

Late news: check Commodore Hacking Information (Reference: info) for
more information of retrieving an archive of the individual articles.

@(A)c: Enquiring Minds Wanna Know!

From: Peter Hofman <HOFMAN%NLEV00@btmv56.se.bel.alcatel.be>

Dear C=Hacking
I would like to make a suggestion for your "Commodore Hacking E-Zine" page.
Maybe you could add a link to a page with some info about the next issue of
Commodore Hacking, so people know what will be in the next issue. The reason
why I make this suggestion is that I read the other issues, and I am very
curious, what will be published.

@(A)r:
Good suggestion. So good, in fact, that we implemented it. Mind you, we
can't completely predict the future, so the information in the link may
not exactly reflect the contents of the issue when it is published, but
we'll try to keep the two in sync.

@(A)c: Pulling It Out of the Closet

From: bloodbane@rlion.com (Jeffrey S Cotterman)

Dear C=Hacking,
Well, I was just writing to say I think you did a great job on C=
Hacking... I am throughly amazed by the support and the interest in
the Commodores. I have a Vic-20, C-64, C-128, and an Amiga 1000.
I have not used any of them in a long time, I have two Beamers that I
use more. However seeing all this stuff makes me want to turn them back
on. (Actually I use the 64 quite a bit for playing games, plus the
1702 monitor works great with a Super Nintendo!) I used to be quite
proficient at the 64, but it is slipping. I will try to get my butt
back in gear so maybe I can post an article or two.... Geesh, and just
last year I got rid of all my Run and Compute's Gazette magazines....
Oh well I will look through the cobwebs and see what I can come up
with.   Anyway, congrats on the mag, I think it's going great.

@(Ar:
We appreciate the thanks. And, if we can get one person to pull a C64
or other CBM machines out of the closet and turn it back on through what we
do, it is worth it.

@(A)c: C=Hacking Flunked Geography

From: Peter Karlsson <dat95pkn@idt.mdh.se>

Dear C=Hacking,
I saw your mention of Atta Bitar in Commodore Hacking.

German? Heheheh... :-)

Anyway, the English information page is available now, but not much will=20
be in English (sorry). It is a Swedish paper :)

From: Erik Paulsson <ep@algonet.se>

Dear C=Hacking,
I'm the editor of the Swedish mag Atta Bitar (8 bitter), so I thought I
should drop you a line.

I really like the "new" C= Hacking it's really great, keep up the good work!

One small comment regarding Atta Bitar, it's not in German, it's in swedish.
I just thought you should know...

@(A)r:

Picky, picky, picky. It's not like we would react that way if you said
Commodore Hacking came from CANADA. Wait, hold it.... I guess we would.
Correction made. Thanks for the update, and if we ever learn Swedish, we'll
try to read it again. (Anyone want to translate for us?)

==

## @(#)news: Newsfront

* Matthew Desmond, the author of Desterm, has recently resurfaced and
   states that he is once again working on something. Although Commodore
   Hacking discourages hourds from emailing him to ask about Desterm
   progress, Matt's email address is mdesmond@ionline.net, for those
   who wish to register Desterm or express their support.

* Speaking of email addresses, LOADSTAR will be changing theirs. As
   the online service GEnie has recently been purchased and new fares
   have been put in place, LOADSTAR finds its monthly bill rising too
   high for pleasure. As of March, 1996, the Internet address for
   LOADSTAR will be loadstar@softdisk.com.

* While we're on the subject of email addresses, CMD has expanded their
   set of Internet email address contacts in order to better support its
   online users. The following addresses are now valid:

Email Address	Usage
cmd.sales@the-spa.com	Questions relating to product prices, catalog requests, ordering onformation,
cmd.support@the-spa.com	Technical questions concerning CMD products.
cmd.cw@the-spa.com	Questions or comments relating to Commodore World magazine.
doug.cotton@the-spa.com	superceded the cmd-doug@genie.geis.com address previously used for all CMD inquires. Should be used items not applicable to the above addresses
cmd.cac@the-spa.com	Direct link to Charles A. Christianson, VP of Sales and Marketing. Again, shuld be used for items not applicable to above email addresses.

* We're not done yet, as COMMODORE CEE has recently moved its office and
   is now at:

      COMMODORE CEE
      5443 College Oak Drive #26
      Sacramento, CA 95841
      Jack Vanderwhite@cee-64.wmeonlin.sacbbx.com (Contact)
      ceejack@crl.com (Contact)
      Jack Vanderwhite, editor.
      Fidonet: 1:203/999
      (916) 339-3403 (Bulletin Board System)

* The Commodore Zone. No, it's not an alternate universe, but a magazine
   for the Commodore gamer and/or demo fan. Each issue's 40 pages is full
   of reviews, interviews with top programmers, and an exclusive comic
   strip done by Alf Yngve. Accompanying each issue is a 5.25" disk or
   tape containing game demos, demos, and full games. Free software is
   often included.

   More information can be obtained through:

      Commodore Zone
      34 Portland Road
      Droitwich
      Worcestershire
      WR9 7QW
      England

   Copies are available for UK3.00. Make checks payable to Binary
   Zone PD.

* Also in the magazine front, Computer Workshops, Inc. is planning a
   World Wide Web magazine to feature gaming. The blurb follows:

      CWI is working on a new Web magazine to feature the newest and
      hottest in c64/128 Gaming. But, before we can do that, we need

your help. Send us what you're working on, or, if you're a
programmer with something for review, send us that too! Also,
if you've got a product you'd like to advertise, we'd like to
hear that too (a la Yahoo).

Send it to either, or both:

spectre@deepthought.armory.com

Computer Workshops/Cameron Kaiser
ATTN: Commodore Gamer
3612 Birdie Drive
La Mesa, CA 91941-8044

(Please don't send binaries to the spectre@ address.)

Thanks for your support, and barring any unforseen difficulties,
Commodore Gamer should be ready to premiere in about two months.

Cameron Kaiser

* The December 10, 1995 edition of the Waco Tribune Herald put one of our own in the spotlight. The headline read "'Antique' PCs have loyal fans here, elsewhere." and was written by Sherry W. Evans, Tribune-Herald staff writer. The Commodore user taking the spotlight was Karen Allison, known on the FIDO network. Sharing the spotlight with Karen was Brad Jackson, of Commodore Country, who said that a C64 was raced against an Intel 386 using identical programs, and the 64 won. Allison claims in the article that "the challenge is finding creative ways to solve problems since Commies have had no ... support..." Allison indirectly mentioned GeoFAX, the GEOS Fax program, and a low cost Tax program she uses to pay the IRS every year. Allison, a diehard "Commie", explained in the article that "(People who use IBMs) all think my Commodore can't do much and is just a toy. But for a toy, this computer does pretty good."

* For those good with an assembler and the VIC/SID registers, Driven Magazine is sponsoring a 4K Demo competition. The deadline is July 1st 1996. Although the program must run on an NTSC 64, PAL programmers are encouraged to enter. The entries will be released as a group at the close of the contest, and entrants can re-use their entries. The complete rules follow:

4k Demo Contest Rules

- 1 file only (no secondary loading)
- max file size is $1000 bytes
- must be started with BASIC 'run'
- 1 demo per coder; multiple entries per group are allowed.
  Multiple coders can collaborate on a single demo, so long
  as there remains only 1 demo per coder.
- credits for all parts of an entry must be given at time of entry;
  if a particular credit is unknown, mark it as "unknown"
- demos will be evaluated on NTSC c64.

Anything not specifically denied above is allowed; e.g. packing + crunching, use or non-use of music or graphics, entries by PAL sceners, etc.

Deadline = July 1.

Entries need to go to coolhand:

coolhand@kaiwan.com

or postal mail to:

Bill Lueck
17411 Mayor Lane
Huntington Beach, California, 92647
USA

Evaluators will be selected shortly, but will include Coolhand and 2-3 other non-demomaking NTSC demo enthusiasts.

There will be categories for evaluation, but there will not be separate winners - the scores from the categories will be added together. The categories will be announced after the evaluation team is established, but they will include design, originality, technical difficulty, artistic impact, and best overall impact, etc (all tentative at this time).

```
*  Bo Zimmerman has put his Commodore 128 on the net.  No, he didn't
   log into some Internet service from his 128, he actually PUT it on
   the 'Net.  Running the BBS program called "Zelch", the machine can be
   accessed by telntting to 147.26.162.107 and giving "zelch" as the
   login.  Bo warns that the system is running off a single 1571 at 2400
   bps, so don't hog the system, OK.

   For the technical types, the 128 is connected to the serial port
   of a Linux PC hooked up the Internet.  Nonetheless, we're getting
   closer to the standalone CBM Internet server.

*  In the March 1996 issue of _Next Generation_ magazine, on pages 31
   and 32, NG published a very unflattering definition of the Commodore
   64 as part of their: "The Next Generation 1996 Lexicon A to Z:
   A definitive guide to gaming terminology".  Among other things, the
   definition's writer confused the Apple II with the Commodore 64 and
   stated that the C64 could not display lowercase, a common problem
   on early Apples.  The writer was biased in favor of the Apple II line,
   but evidently had never used a C64 or never owned an early Apple II.
   In either case, the fervor caused by the definition sparked an outrage
   in the USENET newsgroup comp.sys.cbm.  See UseNuggets (Reference: usenet)
   for the scoop.

   If you would like to write to _Next Generation_, even though the
   article claimed no comments would be heard on the subject, or to
   request a copy of the article, their address is:

   Editorial:

      Email:   ngonline@imagine-inc.com
      Fax:     (415) 696-1678
      Phone:   (415) 696-1688

   Subscriptions:

      Email:   ngsubs@aol.com
      Phone:   (415) 696-1661

   Post Office Mail:

      Next Generation
      Imagine Publishing, Inc.
      1350 Old Bayshore Highway, Suite 210
      Burlingame, CA 94010

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

## @(#): trick: RUN64: Moving to 64 Mode

by Doug Cotton (doug.cotton@the-spa.com)

Reprinted with permission. Copyright (c) 1996 Creative Micro Designs, Inc.

Various routines have been used over the years to allow programs to move from 128 mode to 64 mode without user intervention. With the advent of modified Kernal ROMs (JiffyDOS, RAMLink, and others) many of the methods that work on stock machines have either failed to do the job completely, and in some cases fail all together.

RUN64 is the answer to those users looking to worm their way into 64 mode without having to be concerned with the different Kernal ROMs. The program is presented here in two ways: as a BASIC program that will move to 64 mode and load the program you request, and as assembly language source for ML programmers.

BASIC Notes: The BASIC version uses the ML code produced by the assembly language source. This is found in the data statements beginning at line 660. When you run it, the program will ask for the file name, device number, and file load type (BASIC or ML). The first two parameters should be self-explanatory, but the load type may confuse you. If the file you're loading is itself a small loader (1, 2 or 3 blocks) then it will almost always be an ML program. Likewise, if you usually load the file with a ",8,1" at the end of the load statement, it's ML. If you're loading a larger file, or a file that you normally load with just a ",8", then use the BASIC option.

Also, if you remove the REM instructions from lines 150 through 180 the program becomes a dedicated loader. Just specify the file name and other options within those lines.

@(A): How The Routine Works

RUN64 performs its trick by masquerading as a cartridge. When started, the code copies the payload routines into $8000, with the special header that signifies a cartridge is present. It then resets the system. The system initializes and checks for a cartridge. When it finds the payload routines, it executes them just like it would any cartridge. The pseudo-cartridge routines then switch out BASIC, call the remainder of the KERNAL init routines, switch BASIC in, call some BASIC init routines, set the "load" and "run" lines on screen, dump some "returns" into the keyboard buffer, and finally jump into the BASIC interpreter.

@(A): Assembly Language Notes:

The source code is pretty well documented, and ML programmers should have little trouble figuring out what everything does. Take note of the Buddy Assembler .off pseudo-op used a few lines below the code label. This adjusts all fixed references within the code that follows it to execute properly at $8000.

The code uses some indirect vectors (ibv, ibr and ibm) to overcome not having an indirect jsr opcode, and switches out BASIC ROM temporarily since the KERNAL finishes intializing by indirectly jumping through the address at $a000. Since the target application hasn't been loaded yet, the code must put its own address at $a000 to regain control.

To use the routine, just set up a file name at filename, put a device number in $ba, set the load type in sa1flag, then execute the routine.

```
   100 rem run64.bas (c) 1996 creative micro designs, inc.
   110 :
   120 print "{CLEAR/HOME}run64"
   130 print
   140 :
   150 rem f$="filename" : rem filename
   160 rem dv=peek(186)  : rem device number (8, 9, 10, etc.)
   170 rem l$="a"        : rem load type (a=basic, b=ml [,1])
   180 rem goto 310
   190 :
   200 input "filename";f$
   210 input "{2 SPACES}device";dv$ : if dv$="" then 230
   220 poke 186,val(dv$)
   230 dv = peek(186)
   240 print
   250 print "select a or b"
   260 print "{2 SPACES}a.
   load";chr$(34);f$;chr$(34);",";right$(str$(dv),len(str$(dv))-1)
   270 print "{2 SPACES}b.
   load";chr$(34);f$;chr$(34);",";right$(str$(dv),len(str$(dv))-1);",1"
   280 get l$ : if l$<>"a" and l$<>"b" then goto 280
   290 print
   300 print l$;" selected"
   310 print
   320 print "going to 64 mode!"
   330 :
   340 : rem poke in main ml
   350 :
   360 i = 6144
   370 read d
   380 if d = -1 then 450
   390 poke i,d
   400 i = i + 1
   410 goto 370
   420 :
   430 : rem poke in filename
   440 :
   450 for i = 0 to len(f$)-1
   460 : poke 6356+i, asc(mid$(f$,i+1,1))
   470 next i
   480 poke 6356+i,0
   490 :
   500 : rem poke in device number
   510 :
   520 if dv$="" then 570
   530 poke 186,val(dv$)
   540 :
   550 : rem check load type
   560 :
   570 poke 6324,0
   580 if l$="b" then poke 6324,1
```

```
590 :
600 : rem sys to ml
610 :
620 sys6144
630 :
640 : rem ml data
650 :
660 data 32,115,239,160,0,185,22,24
670 data 153,0,128,200,208,247,165,186
680 data 141,157,128,76,77,255,9,128
690 data 9,128,195,194,205,56,48,169
700 data 0,141,4,128,120,169,0,141
710 data 22,208,32,132,255,32,135,255
720 data 169,230,133,1,169,43,141,0
730 data 160,169,128,141,1,160,76,248
740 data 252,169,231,133,1,32,148,128
750 data 32,151,128,32,154,128,162,0
760 data 189,159,128,240,6,32,210,255
770 data 232,208,245,162,0,189,190,128
780 data 240,6,32,210,255,232,208,245
790 data 162,0,189,180,128,240,6,32
800 data 210,255,232,208,245,173,158,128
810 data 240,10,169,44,32,210,255,169
820 data 49,32,210,255,169,145,32,210
830 data 255,32,210,255,173,157,128,133
840 data 186,162,0,189,185,128,240,6
850 data 157,119,2,232,208,245,173,158
860 data 128,208,2,169,4,133,198,76
870 data 157,227,108,149,227,108,152,227
880 data 108,155,227,0,0,17,17,68
890 data 86,61,80,69,69,75,40,49
900 data 56,54,41,58,76,79,65,68
910 data 34,0,34,44,68,86,0,13
920 data 82,213,13,0,70,73,76,69
930 data 78,65,77,69,0,-1
```

```
; RUN64.SRC
; Doug Cotton & Mark Fellows
; (c) 1996 Creative Micro Designs, Inc.
;
        .org    $1800
        .obj    run64.obj

run64   jsr     $ef73   ; go slow
;
        ldy     #0      ; copy cartridge
-       lda     code,y  ; code to $8000
        sta     $8000,y
        iny
        bne     -
;
        lda     $ba     ; get device number
        sta     dvtemp  ; and store it
;
        jmp     $ff4d   ; go 64
;
code    .byt    $09,$80 ; cold start
        .byt    $09,$80 ; warm start
        .byt    $c3,$c2,$cd,$38,$30     ; cbm80
;
        .off    $8009   ; offset code
;
        lda     #$00    ; disable
        sta     $8004   ; cartridge code
        sei             ; disable interrupts
;
        lda     #$00    ; zero out
        sta     $d016   ; VIC control Register
;
        jsr     $ff84   ; initialize I/O
        jsr     $ff87   ; initialize RAM
        lda     #$e6    ; switch in RAM
        sta     $01     ; at $A000
        lda     #<reenter       ; set up return vector
        sta     $a000   ; at $A000 to bypass
        lda     #>reenter       ; BASIC statup during
        sta     $a001   ; initialization
;
        jmp     $fcf8   ; let Kernal finish up
```

```
;
reenter lda     #$e7    ; back from Kernal, set
        sta     $01     ; $A000 back to ROM
;
        jsr     ibv     ; initialize vectors
        jsr     ibr     ; initialize RAM
        jsr     ibm     ; initialize memory
;
        ldx     #$00    ; output screen text
-       lda     part1,x ; to form LOAD statement
        beq     +
        jsr     $ffd2
        inx
        bne     -
;
+       ldx     #$00    ; print filename to be
-       lda     filename,x       ; loaded at end of
        beq     +       ; LOAD statement
        jsr     $ffd2
        inx
        bne     -

;
+       ldx     #$00    ; print device
-       lda     part2,x ; variable at end
        beq     +       ; of LOAD statement
        jsr     $ffd2
        inx
        bne     -
;
+       lda     sa1flag ; check secondary
        beq     +       ; address flag for load
        lda     #','    ; type, and print a
        jsr     $ffd2   ; comma and a 1 at end
        lda     #'1'    ; of LOAD statement if
        jsr     $ffd2   ; load type is ML
;
+       lda     #$91    ; print two CRSR up
        jsr     $ffd2
        jsr     $ffd2
;
        lda     dvtemp  ; get device number
        sta     $ba     ; and store
;
+       ldx     #$00    ; put [RETURN]rU[RETURN]
-       lda     keydata,x        ; into keyboard buffer
        beq     +
        sta     $0277,x
        inx
        bne     -
;
+       lda     sa1flag ; get load type and
        bne     +       ; branch if it is ML (1)
        lda     #$04    ; if not ML, change .A
+       sta     $c6     ; store kybd buffer NDX
;
        jmp     $e39d   ; enter BASIC
;
ibv     jmp     ($e395) ; initialize vectors
ibr     jmp     ($e398) ; initialize RAM
ibm     jmp     ($e39b) ; initialize memory
;
dvtemp  .byt    $00     ; device number temp
sa1flag .byt    $00     ; load type (1=ML,
                        ; 0=BASIC)
;
part1   .byt    $11,$11 ; 2 CRSR up
        .byt    'dv=peek(186):load'
        .byt    $22     ; quote
        .byt $00
;
part2   .byt    $22     ; quote
        .byt    ',dv'
        .byt $00
;
keydata .byt    $0d     ; [RETURN]
        .byt    'rU'    ; shortcut for RUN
        .byt    $0d     ; [RETURN]
        .byt    $00
;
filename        .byt    'filename'      ; name of file to load
```

```
        .byt $00        ; 00 byte must follow filename!
;
        .end
```

===

@(#)mags: Hacking the Mags

Not everything good and/or technical comes from Commodore Hacking, which is as it should be. (I still think we have the most, though...) Thus, let's spotlight some good and/or technical reading from the other Commodore publications.

If you know of a magazine that you would like to see summarized here, let C=Hacking know about it. These summaries are only limited by Commodore Hacking's inability to purchase subscriptions to all the Commodore publications available. We are very grateful to those publications that send complimentary copies of their publications for review.

@(A): COMMODORE CEE
At press time, Issue #5 was in the works, so we'll detail the contents next time. However, see Newsfront (Reference: news) for address changes for COMMODORE CEE.

@(A): Commodore 128/64 Power User Newsletter (CPU)
A while back, Gosser Games, Ltd., Inc. sent us a sample issue of this publication, which is published exclusively with a Commodore 128 machine, much like the defunct dieHard. For those just getting into the BBS arena, the "Cyberspace Cowboy", R.J. Smulkowski, previously writing this article in dieHard, has moved his column to CPU. The content is light, but useful, and a godsend for new users. Reviews of GeoFAX and "Radio Controlled Flight Simulator" also grace the pages. Printed on regular bond at 7" by 9", the 16 page publication is small but full of potential.

@(A): Commodore World
If you remember last time we spoke of Commodore World, we asked the rhetorical question: What's up with those funky graphics? We didn't expect an answer, but editor Doug Cotton called to explain the curious eye-catchers. He also mentioned that asst. editor Jenifer Esile was having trouble creating them now that we made fun of them. We're sorry Jenifer. We want you to continue, since they save us the cost of commercial inkblot cards for our self-psycho-analysis sessions here at Hacking Headquarters. (just joshing, we can be so mean sometimes). Speaking of Jenifer, we're not sure when she started, but the last few issues seem more colorful. Sure, content is great, but packaging is everything. We've even caught PC-centric folks perusing our copy. Kudos to CMD for that effect.

Obviously, Commodore World iisn't for all, but the content is consistent. Issues 11 and 12 are no exception. In issue 11, Doug tackles high level serial bus routines and includes ML source, Gaelyne Moranec shares some useful WWW pages, and Jim Butterfield explains the nasty NULL character and its implications. Of special interest in this issue is the two page spread on changing device numbers on the pesky 1541 drives. The article is worthy of archival for reference. CMD also takes time to note that the SuperCPU cartridge will contain a 65C816 8/16 bit CPU, not the earlier mentioned 65C02 8-bit CPU.

Issue 12 should be subtitled the "SuperCPU" issue. We think its great, but it's definitely not subtle. Doug Cotton and Mark Fellows preview the unit while Jim Brain details the CPU inside it. CMD ntes that the 10 MHz version has been scrapped, but the 128 version has been added, dealying introduction until April for the 64 version. C=H was hoping to review a prototype unit this issue, but we'll do it next time. Jason Compton and Katherine Nelson describe HTML, the markup language for World Wide Web pages, and Jim Butterfield explains using KERNAL devices 0 (keyboard) and 3 (screen). For those wanting to run a Bulletin Board System, Max Cottrell describes how to ensure success. Of special interest in this issue is a photo of the prototype accelerator. We won't even hint of our opinions on this round of funky graphics....

@(A): Driven
Driven #11 waxes somewhat philosophic about the demo scene in 1995. The tone expresses a tinge of disappointment with the hope that 1996 will be a better year for demos. This issue also ushers in Driven's first crack at covering the PAL scene. As part of the 1995 year-end review, a complete list of releases is given. In the news section, Charles Christianson's blurb on the CMD SuperCPU is reprinted, and King Fisher

of Triad discusses the origins of the demo scene in "Cyberpunk".

If you've ever wondered what goes on inside the mind of a demo "scene" programmer, Driven #12 will fill you in. Interviews with Phantom of the group FOE and Zyron of F4CG are included, both telling it as it is. For those wanting to set up or design a BBS system Mitron takes a look at CNET DS2 and details some general guidelines on how the networking code works. Of special note is a review of this issue's Polygonamy sample code (Reference: polygon).

@(A): LOADSTAR
Issue 139 starts off with the announcement that LOADSTAR is taking over the dieHard Spinner disk subscriptions, as reported in C=H#11. File Base 64 from John Serafino will be useful for anyone organizing their disk collection. Fender Tucker claims it is better than DCMR, the supposed standard. Jeff Jones cooks up the "Ultimate INPUT" for people wanting the perfect BASIC INPUT routine. The claims are substantial and Jeff delivers. The included LOADSTAR LETTER #29 contains another article in the Internet series.

As we started Issue 140, we noticed something was different. We couldn't place it at first, but then Jeff alerted us to the change. LOADSTAR now has highlighted words in the text, and the color scheme can be changed and saved. Nice for the eyes. In addition, LS#140 can mark up text using highlights, bold, and underline on printers that support such features. Bob Markland presents a ML module that provides better random numbers, and Fender Tucker challenges programmers ML programmers to write a routine that searches an in memory dictionary for a word. Speed is the key. C=H gets some space, as Issue 11 is reprinted in the 3.5" version. Of particular note to programmers is Don Forsythe's "Hidden Clocks" article that describes in detail the CIA TOD clocks and their bugs, err "features".

It's funny, but the LOADSTAR LETTER #40 that comes with LS #140 is subtitled "Special Commodore Hacking Issue". We were expecting C=H articles, but that shows just how egotistical we can be. Jeff Jones filled the issue with rumors of new products, handy tips, and information about CMD's SuperCPU. Of particular interest is the information about Craig Bruce modifying his Swiftlink to do 115,200 bps.

Right before we went to press, issue 141 showed up in the mailbox. #141 starts off with the changes of operation since LOADSTAR publication was taken over by J & F Publishing. The first is their new address:

LOADSTAR
606 Common Street
Shreveport, LA 71101

Also, they say checks should now be made out to LOADSTAR, not Softdisk.

For all you TUI (Text User Interface) folks, Jeff Jones goes over how to create "buttons" that depress on screen when activated. Source code is provided as well, which is rare for LOADSTAR. Of particular interest to us was Terry Flynn's "Virtual Optics" slideshow. Hard to describe, it displays impossible constructions and 3D illusions. Even C=H gets some space, as issues 3 and 4 are available on the 3.5" disk version. Jim Brain supplies article 4 in the Internet series on LOADSTAR LETTER #31, included with the issue. Of special note is LOADSTAR's new Internet address, given in the LL as loadstar@softdisk.com. See Newsfront (Reference: news) for more information.

@(A): LOADSTAR 128
We loaded up LS128 #30 for a look-see. Dave's Term, the 128 Telecommunications Program presented in the last 4 issues, seems to be one focus of this issue. Don Graham supplies a keyboard overlay and macros for the terminal program, while David Jensen includes a spell checker. In the issue as well is ZED, the 128 editor of editors from Craig Bruce.

@(A): Vision
At press time, Issue #8 was in the works, so we'll detail the contents next time.

Other magazines not covered in this rundown include:

* _The Underground_
* _Gatekeeper_
* _Commodore Network_
* _64'er_
* _Atta Bitar_ (_8 bitter_)
*   Commodore Zone

* _Commodore Gazette_

In addition, others exist that C=Hacking is simply not aware of. As soon
as we can snag a copy of any of these, or get the foreign language ones
in English :-), we will give you the scoop on them.

==

## @(#): Polygonamy: A Study in 3 Dimensions

by Stephen L. Judd (sjudd@nwu.edu)

We've been making some pretty small potatoes for a while now, so
the time has come for a little more ambition and challenge. I decided to
think up a real challenge, containing problems that I had no idea how to
solve, and see what I could come up with.

I set out to create a 3D virtual world for the C64, e.g. a space
populated with various three-dimensional objects, which I could wander around
in. I wanted it to be full-screen 320x200 hires bitmapped. Furthermore, I
wanted the objects to be solid, and since there are only two colors I
wanted to be able to put patterns on the faces. I also wanted it to
translate nicely to the 128's VDC chip, in 2MHz mode. Finally, naturally, I
wanted the program to be fast. This was the framework in which I placed
myself, and a few other ideas presented themselves along the way. The
outcome of all of this is Polygonamy.

Just a brief history of this project: I have wanted to do a 3D
world for a very long time, and have been thinking about it for some
time in the back of my head; my imagination was probably first fired
the first time I played Elite. I wrote down the necessary equations one
afternoon last summer, for a high school student I was teaching, and
the equations are very simple. I took a break to get some work of
measureable value accomplished, but in October I began work on the graphics
algorithm. I worked steadily on this for two months, and in December I
finally began to code the graphics. In mid-January, I got them to work.
Adding the rest took a few weekends more. I have about 128 pages of notes,
analytical calculations, and BASIC test programs in my C64 notebook (which
is, I think, a nice number of pages to have :). My original plans were
to place five objects in the world, but time and memory constraints whittled
that down to three. One of my disks self-destructed about the day I was
ready to finish, so I had to reconstruct a bunch of tables, but other than
that I finally managed to finish it up, albeit with a few rough edges ;).

Although the concepts from previous articles are used as a solid
foundation, the code is almost 100% from scratch -- I only borrowed a
tiny piece of code from an earlier program, and even that I modified
somewhat.

One caveat before we begin: I am primarily interested in the
challenge of designing the algorithms, which means I like to come up
with my own solutions. Thus, you may find more efficient methods
in a graphics book or perhaps in someone else's code; I have examined
neither, so I have no idea what the relative merit of my algorithms
may be. These are simply my solutions to challenges placed before me.
And if you know of a better way to do things, please feel free to email
me!

Furthermore, I consider the code a test of the theory. Some of my
assumptions work and some do not, and these will be considered at the end
of this article.

Finally, I am not including the source code. For one thing, it is big.
Like, _HUGE_, man. I had to split it up when I ran out of editor memory
on my 128 (which, incidentally, forced me to figure out Merlin 128's very
cool and very powerful linker feature). I will include numerous code
fragments in assembly and BASIC which demonstrate all the important
concepts.

By the way, if you are interested in measuring frame rates, you can
use the first object. Every full 360 degree revolution is 128 frames.
So time how long it takes to complete a full rev (or maybe several), and
divide that number into 128, to get an idea of frames per second.
For a rundown of frame rates for stock and SuperCPU operation, see
"Underneath the Hood of the SuperCPU" (Reference: cmdcpu) found
elsewhere in thi issue.

Some brief acknowledgements: This project would not have happened without
the extremely powerful macro and linking capabilities of the Merlin 128
assembler, by Glen Bredon. It would have been _really_ tough without
JiffyDOS and my FD-2000, from CMD. I used my Action Replay extensively
for debugging, and without the excellent documentation for the 64, such as

the PRG and Mapping the 64, this would have been a nightmare. Finally, I must acknowledge my friend George Taylor; a few days before I was all finished I explained some routines to him, and he made a great suggestion which made my fast fill routine blaze.

Okay, WAY too much talk. There are a ton of issues involved with this project so let's just wade in hip-deep and deal with them as they come.

@(A): The Equations

First some relaxing abstraction. In previous articles we have discussed how to project an object in three dimensions through the origin into a plane. We have also discussed rotations in three dimensions. In principle, then, we have all the mathematics we need to do a 3D world.

But we should be thoughtful. Let's say we're standing in the world and turn to the right; we can either rotate ourselves, and change the direction we are looking, or we can rotate the world around us, so that we are always looking 'forward'. This may bother you on physical grounds, but the two are mathematically equivalent. Given the way we have derived our projection routines, it should be clear that we want to rotate the objects in the world around us.

(Or, to put it another way, we are at the center of the world, and the world revolves around us.)

We have another issue: how do we know when an object is visible or not? How do we know when we've bumped into an object (or blown it out of the sky :)? Moreover, if we have ten objects, and each object has six points, it would be a real drag to have to rotate all sixty points, especially if none of the objects were even visible.

It should be clear that we really want to define every object relative to some center of the object. So we keep track of the center of each object, and rotate and translate the centers, and only calculate the full object if it is visible. We of course want to define the object relative to this center.

What happens to this center when we translate or rotate the world?

Let's simplify our model a little bit and only deal with rotations about one axis, e.g. we are driving a tank and can move forwards and backwards, and turn left or right. The generalization to other axes is very straightforward, but this way we can think in two dimensions instead of three.

First we need to agree on a coordinate system. For my system I let the x-axis go _up_ a page of paper, the y-axis comes up out of the page, and the z-axis goes from left to right. Thus, I am standing on the paper in the x-z plane, at the origin, with the y-axis extending upwards from me. If you still don't understand my orientation, draw a little picture.

I am going to choose my orientation so that I am always looking straight down the x-axis, e.g. in the direction that x is increasing. Thus, if I walk forwards or backwards, this corresponds to decreasing or increasing the x-component of the center coordinate:

```
   let C=(cx,cy,cz)
   move forwards => cx=cx-1
   move backwards=> cx=cx+1
```

So far so good. As always, draw a picture if you can't visualize it. That takes care of translations, what about rotations?

We certainly know how to rotate points about the origin. In particular, if we have a point with coordinates (x1,z1) and rotate it clockwise by an angle s, we get the new point as follows:

```
   (x1,z1) -> (x1*cos(s)+z1*sin(s), z1*cos(s)-x1*sin(s))
```

So that's easy enough. The problem is that we have this big object sitting around this point, and need to figure out what to do with it!

Consider the following: let's say we have a line out some distance from the origin,

```
                          X
                          X
            |             X
 z-axis ----O------------c----     c=center
```

```
            |             X
            |             X
          Origin          X <---- line
```

and we rotate it by some amount theta about the origin:

```
                      X
                      XX
                       Xc      c=rotated center
            |           XX
 z-axis  ---O------------XX-
            |             X
                          XX
```

You can see (from my incredible ASCII artwork) that the line is now at an
angle with respect to the origin.

Imagine that we draw a line from the origin to the center of the point,
in the first picture (or get a pencil and paper and actually do it), so
that we have the letter "T" laying on its side. Now we rotate this "T"
by some angle theta, so that the top of the "T" -- our line -- has now
been rotated.

The stem of the "T" meets the top of the "T" at the center point c. Drop
a line from the rotated center straight down to the z-axis, and call this
line l2. Since the T is at a right angle, and we have rotated it by an
angle theta, the angle between our line and the z-axis is 90-theta. But
this means that the angle between our line (the top of the "T") and the
line l2 is just theta.

Thus, if we rotate the center by an amount theta, all we have to do
is rotate the object by an amount theta *about the center*, and our
perspectives will all be correct. How is that for simple? It should
be clear now that this works no matter where the "center" of the object
is chosen. Thus, our center is not some physical center of the object,
but rather the center of rotation of the object.

Since this is true of rotations about any axis, we now know how to
generalize to higher dimensions.

Note further that we can now implement local rotations -- that is, let's
say the object is a tank, and this tank turns independently of whether
or not we turn. Piece of cake.

You can also see that the rotations are cumulative. If we turn to the
left, and then turn left again, we can simply apply two rotations to the
points of the object. In fact, if we turn left, move forwards, and then
move left again, we still apply just two rotations to the points of the
object; the center, however, has moved.

This is quite important, as it allows us to measure an object's relative
rotation to us, whether or not it is visible. Remember that we only
want to rotate the points that define an object when the object is
visible. We never actually change the points which define an object.
Instead, we track how much the object needs to rotate, and rotate the
original points by that amount.

The center of the object will change with each rotation and translation.
We never change how we define an object about this center, though. We
simply apply a rotation to the original points when appropriate. The
object centers must be kept track of because they can undergo translation
as well as rotation.

To summarize then: we define each object relative to a center of
rotation. The center determines where the object is located in the
world, and allows us to operate on centers when we need to rotate or
translate the world. It also lets us perform local operations on an
object, so that we could, for instance, have a spinning cube located
inside our world. If an object's center is visible then we can consider
the object to be visible, and plot it.

Whoops -- what does it mean to be 'visible'? Well, think about yourself.
You can see things when they are in front of you. Or, to be more
precise, you have a field of vision. Perhaps a decent model of this
would be a cone. I think a better model, certainly one which is
easier to deal with computationally, is the intersection of two planes:
a pyramid. Anything which lies within this pyramid we consider visible,
and anything which lies outside we consider not visible.

Two-dimensionally, we draw a little wedge extending from the origin.
Anything within the wedge we count as visible, and anything outside

of it we count as not visible. The sides of this wedge are two lines,
with equal but opposite slope (i.e. slope=+/-m).

```
 \ Visible /
  \  View /  Outside of visual area
   \Area /
    \   /
     \ /
      * <--- Me
```

Probably lines at some angle are more reasonable than others. But I'm a
simple guy, and the two simplest lines I can draw are at 45 degree angles
from the axis, so their slope is +/-1. Thus, any points which lie
between the lines x+z=0 and x-z=0 are visible. If the center of an
object is within this area, we will consider the object visible. That
is, if cx+cz>0 and cx-cz>0 the object is visible.

One last thing: if we are too close to the object, we either want to
bump into it (i.e. not move) or else not display it. So we also need to
check if cx<x0 for some x0.

We are now in a position to write some simple code. I wrote the
following in evil Microsoft QBasic, but BASIC7.0 on the 128 would work
just as well, although you need to change the variables (I didn't have my
128 handy, otherwise I would have written this on the 128):

@(A): Polygon Prototype Code

```
   SCREEN 1 '320x200

   delta= 3     'Rotations will be in 3 degree increments
   rad= 3.1415926/180
   cdel= COS(rad*delta)
   sdel= SIN(rad*delta)
   theta=0
   d=-135
   x0= 60       'Bumped into the object?
   REM z0 y0 would be 160,100 to place the object in the center
   REM of the screen
   y0= 170  'I want the bottom of screen to be ground
   z0= 160

   REM Set up the object
   REM Tetrahedron: 0,sqrt(3),0  0,0,1  0,0,-1  2,0,0
   DIM obx(4), oby(4), obz(4)

   obx(1)= 0
   oby(1)= 50*SQR(3)
   obz(1)= 0
   obx(2)= 0
   oby(2)= 0
   obz(2)= 50
   obx(3)= 0
   oby(3)= 0
   obz(3)= -50
   obx(4)= 100
   oby(4)= 0
   obz(4)= 0

   cx= 100
   cy= -10
   cz= 0

   REM Get input
   main:

   DO
     a$=INKEY$
   LOOP UNTIL a$<>""

   IF a$="[" THEN cx=cx-20
   IF a$="/" THEN cx=cx+20
   IF a$=";" THEN GOSUB rotl
   IF a$="'" THEN GOSUB rotr

   IF cx<x0 THEN CLS: GOTO main:
   IF cx<cz OR cx+cz<0 THEN CLS: GOTO main:

   ctheta= COS(rad*theta)
   stheta= SIN(rad*theta)
   p1x= cx + ctheta*obx(1) + stheta*obz(1)   'Rotate and add to center
   p1y= cy + oby(1)
   p1z= cz + ctheta*obz(1) - stheta*obx(1)
   p1y= y0 + d*p1y/p1x           'Project and add offset
   p1z= z0 + d*p1z/p1x
   [... similar for p2x,p2y,p2z,...,p4x,p4y,p4z]

   CLS
   LINE (p1z, p1y)-(p2z, p2y)
   [... lines between p2-p3, p3-p1, p1-p4, p4-p2, p4-p3]

   GOTO main:    'Main loop

   REM rotate left
   rotl:
       theta= theta + delta
       blah= cdel*cx + sdel*cz
       cz= -sdel*cx + cdel*cz
       cx= blah
   RETURN

   rotr:
       theta= theta-delta
       blah= cdel*cx - sdel*cz
       cz= sdel*cx + cdel*cz
       cx= blah
   RETURN
```

(You may note that cx=cx+20 is used for a translation, instead of cx=cx+1.
This will be detailed later).

So much for the easy part.

@(A): Filling

If there is one thing that the previous programs have taught us, it is
that graphics are slow. At least, they are far and above the major thing
slowing down the program, and deserve the most attention and thought for
improvement. Moreover, because there is lots of looping involved, the
elimination of just a few instructions can translate to thousands of
cycles saved.

We have examined several fill routines up to now, but neither of them is
up to the task of Polygonamy. The cookie-cutter method is OK, but doesn't
allow multiple objects, and certainly doesn't allow pattern fills. Using
an EOR-buffer is just plain slow and inefficient and a big drag. So it's
time to rethink this problem.

Recall that on the 64 the bitmap screen is divided into 8x8 cells, which
are arranged horizontally in rows. It's a pretty kooky way of doing
things, but we shall overcome.

First of all it should be clear that we want to fill from left to right
(as opposed from top to bottom). We can then fill a byte at a time,
instead of dinking in little pixels at a time.

Previously we used a custom character set to plot into. One of the major
reasons for doing so was to use Y as the Y-coordinate, so that storing a
particular point was as simple as STA COLUMN,Y. We can still use this
idea, but only within each row. That is, if we let Y=0..7, we can address
each individual pixel-row within each 8x8 block row with an STA ADDRESS,Y.

For real speed, we are going to want an unrolled fill routine. That is,
we don't want to mess around with loop counters and updating pointers and
such. Since there are 25 rows on the screen (25 times 8 = 200 pixels
high) we are probably going to need 25 separate fill routines.

I constructed my fill routine as follows:

```
            STA COL1,Y
            DEX
            BEQ :DONE
            STA COL2,Y
            DEX
            BEQ :DONE
            STA COL3,Y
            ... etc.
   :DONE    RTS
```

Thus X would be my counter into the number of columns to fill, A can contain our pattern to fill with, and Y can range from 0..7 to index the individual rows within the block. The first thing to notice is that each STA/DEX/BEQ code chunk is six bytes. So, all we need to do is calculate which row to start filling at, multiply by six, and add that number to the start of the fill routine. The idea is then to jump into the correct place in the fill, and let it fill the right number of columns, stored in X.

There is a little problem though -- what we're talking about doing is an indirect JSR, and there is no such thing. But it's easy enough to fake, because we can use an indirect JMP. So a call to the fill routine would look like the following:

```
         ...
         JSR FILL
         ...
FILL     JMP (ADDRESS)
```

where ADDRESS simply points to the correct entry point in the fill routine.

Moreover, you may also note that 40 columns times 6 bytes/column is 240 bytes, so that each little fill routine handily fits in a page. Thus, moving between rows in the bitmap corresponds to a simple decrement or increment of the high byte of the ADDRESS pointer.

This was the state of things when, days before I was to be all done with Polygonamy, I mentioned it to my friend George Taylor, who suggested the following modification: instead of using X to count how many columns to fill, just make the fill routine:

```
         STA COL1,Y
         STA COL2,Y
         STA COL3,Y
         ...
```

Then, insert an RTS into the right place in the routine. Thus, we calculate which column to stop filling at, multiply by three, and stick an RTS in the fill routine at that point. To fix it up we stick an STA ..,Y back on top of the RTS.

I don't think you're going to make a fill routine faster than that :).

Moreover, note that each fill routine takes up just 120 bytes, so we can now fit two fill routines in each page. I did not do this, but it is easy to do, and instantly frees up 25 pages.

@(A): Filled Polygons

I mean, hey, this _is_ "Polygonamy", so let's talk polygons, and lots of them.

Clearly all that is needed to draw an object are the left and right endpoints of the object, since everything in-between will be filled.

An observation to make is that if you take a slice out of a convex polygon, the slice will intersect the polygon at exactly two points. Another, more important, observation is to note that the highest and lowest point of a polygon will always be at a vertex. Finally, it is important to note that any vertex of a polygon has exactly two lines extending out of it, one to the left, and one to the right.

Consider a piece of a polygon:

```
  \         /
   \       /
    \     /
     \   /
      \ /
       *  <--- Vertex v0
```

where the vertex v0 is the lowest point of the polygon. All that needs to be done is to move upwards (DEY), compute the left and right points of the polygon at that point, and then fill between the two (JSR FILL).

The idea then is to start at the bottom (or the top) and to steadily move upwards while _simultaneously_ calculating the endpoints of the left and right lines, and filling in-between. But we need the equations of the left and right lines to do this.

Now it's time for another observation. Let's say we have a polygon with n vertices v1, v2, ..., vn, and furthermore that as we move between these points we move around the polygon counter-clockwise. Thus v3 is to the right of v2, v1 is to the left of v2, v4 is to the right of v3, etc. For example:

```
      v1____v3
        \  /
         \/
         v2
```

What happens if we rotate this polygon?

```
      v2____v1
        \  /
         \/
         v3
```

The vertices have changed position, but *their order has not*. v3 is still to the right of v2, and v1 is still to the left.

Now we have a real plan. We simply define the polygon as a list of points v1 v2 v3 ... vn. We then figure out which one is lowest, i.e. has the smallest (or greatest) y-coordinate, call this vertex vm (vmax). The endpoints of the left and right lines are vm-1,vm and vm,vm+1. So move along those lines until the next vertex is reached. At that point, recompute the appropriate line, and keep moving upwards until the top of the polygon is reached.

Perhaps an example would be helpful:

```
        v1
        |\
        | \ v3
        | /
        |/
        v2
```

v2 is the minimum. The left line has endpoints (v1,v2) and the right line has endpoints (v2,v3). We steadily move along the left and right lines as we creep upwards. At some point we hit v3, and at this point we compute a new equation for the right line, this time with endpoints (v3,v1). Now we continue to creep upwards and move along the left and right lines, until we hit v1, at which point we are finished.

It is important to keep in mind that the order of the points never changes. We don't need to do anything complicated like sorting the points; we only need to find the lowest point, and branch left and right from there, keeping in mind that the points are cyclic (i.e. v1 is to the right of vn).

It is now time to start thinking about code. One aspect of the fill routine we haven't considered is the clear. In the past the entire draw buffer was cleared and then the new stuff was drawn into it. But this seems like a bit of a waste; it seems wasteful to clear a bunch of memory that is just going to be overwritten again. So, as long as we can do it efficiently, it might be smart to combine the clear and fill routines.

Here is how Polygonamy does it: If a line needs to be cleared, then it is cleared up to the edges of the object, but the part that is going to be filled is ignored. (It isn't clear if this provides any substantial efficiency gains, though).

To see the status of a particular line, a table is used, containing a value for each Y-coordinate. If the entry is 255 then the line is clear, if it's 0 then the line has old junk in it, and if it's 1 then the line has new junk in it. Thus we only clear the line if its entry in the fill table is a zero.

So a fill routine might flow like the following:

```
   let Y count from 7..0

   If we are at the left endpoint then recalculate the left line.
   If we are at the right endpoint the recalculate the right line.
   Update xleft & xright
   If line needs to be cleared then clear line.
   If the starting fill column is different than the previous fill
       column then update the pointers, etc.
   Plot the left and right endpoints (since the fill routine only plots
       eight bits at a time)
   Fill the in-between parts
   Update Y
   If Y passes through zero then update fill entry point, set Y=7 etc.
   Keep going until we reach the top
```

The next thing to figure out is how to calculate the left and right lines. We do have the old line routine, which we could use to advance to the left and right endpoints, but clearly this isn't too efficient.

The question is: if the y-coordinate increases by one, then how much does the x-coordinate increase by? The equation of a line is:

```
   (y-y0) = m*(x-x0)  m=slope
```

or

```
   change in y = m*change in x
```

So, if the change in y is 1, then then the change in x is 1/m. All we need to do then is calculate the inverse-slope=dx/dy, where dx=x2-x1 and dy=y2-y1, and add this to the x-coordinate with each step in y.

Isn't this a fraction? Sure, big deal. The fraction can be written as dx/dy = N + Rem/dy, where N is an integer, and Rem is the remainder, which is always less than dy. So to calculate x=x+dx/dy:

```
   x= x+N
   xrem= xrem+Rem
   If xrem>=dy then x=x+1:xrem=xrem-dy
```

As usual, we want to start xrem at dy/2, which has the effect of rounding numbers up.

```
   10 REM LINE ROUTINE TAKE TWO SLJ 11/24/95
   12 REM ACTUALLY IT'S A FILL RTY NOW
   15 GRAPHIC 1,1
   20 X0=160:Y0=100
   30 X1=5:Y1=-50:X2=7:Y2=11:XL=X1:YL=Y1:Y=Y1
   35 X3=50:Y3=Y1:X4=X3+100:Y4=Y2:XR=X3:YR=Y3
   40 D1=Y2-Y1+1:DX=X2-X1:LI=INT(DX/D1):LR=DX-LI*D1
   45 TL=INT(D1/2)
   46 D2=Y4-Y3+1:DX=X4-X3:RI=INT(DX/D2):RR=DX-RI*D2
   48 TR=INT(D2/2)
   50 DRAW1, X0+X1,Y0-Y1 TO X0+X2,Y0-Y2, X0+X3,Y0-Y3 TO X0+X4,Y0-Y4
   60 REM MAIN LOOP
   70 XR=XR+RI:TR=TR+RR:IF TR>=D2 THEN TR=TR-D2:XR=XR+1
   75 DRAW1, X0+XL,Y0-Y TO X0+XR,Y0-Y
   80 XL=XL+LI:TL=TL+LR:IF TL>=D1 THEN TL=TL-D1:XL=XL+1
   90 REM DRAW1, X0+XL,Y0-Y TO X0+XR,T0-T
   100 Y=Y+1:IF Y<=Y2 THEN 60
```

In this program (x1,y1)-(x2,y2) is the left line, and (x3,y3)-(x4,y4) is the right line. The first thing to note is that in lines 40 and 46, Y=y2-y1+1. This issue was discussed in the very first C=Hacking 3D-graphics article. The problem is that although the line will be anatomically correct with DY=y2-y1, it will look silly. The easiest way to see this is to consider y2-y1=1, e.g. say we draw a line between (0,10) to (50,11). Ideally this line will consist of two line segments, one from (0,10) to (25,10) and the other from (26,11) to (50,11). But ifi we use DY=1 we will have one line segment from (0,10) to (50,10), and a single point at (50,11).

Adding one to DY is just a simple cheat. Most of the time the lines will look just fine, but lines which have a slope near one will come out a bit wrong. The other, accurate solution, which was used in the first article, is more complicated to implement in this routine. Adding one to DY will also have a useful benefit which we shall shortly see.

In line 50 above the boundaries of our object are drawn in, to check the accuracy of the algorithm.

In lines 70-80 the right point is updated, then the thing is filled, then the left point is updated. This is because both lines are moving to the right, e.g. they both have positive slope. Think about how the line segments will be drawn; in general, we want to draw from the left end of the left line segment to the right end of the right line segment. (Sometimes this will look a little off where the two lines meet).

Since the left and right lines can each have either positive or negative slopes, there are four possibilities: Plus-Plus, Plus-Minus, Minus-Minus, and Minus-Plus.

```
   Plus-Plus:  Update right, fill, then update left
   Plus-Minus: Fill, update left, update right
   Minus-Minus:Update left, fill, update right
   Minus-Plus: Update left, update right, fill
```

If this is still confusing, try out the above program with various left and right line segments, and these things will jump right out.

Now we need to think about implementing this in assembly. Since this is being done in hires 320x200, the x-coordinate requires two bytes, and the y-coordinate requires one. We also need another byte to store the remainder portion of the x-coordinate.

The most glaring question is the calculation of dx/dy: somehow we need a fast way of exactly dividing an eight bit number dy into a nine bit number dx. Recall that we always add one to dy, so that dy actually ranges from 2 to 200. Since the maximum value of dx is 320 or so, the largest value of dx/dy that we can have is 320/2 = 160. In other words, both the integer and remainder part of dx/dy will fit in a byte. Simply adding one to dy makes life pretty easy at this end.

One very fast method of doing division is of course by using logarithms. But they have a problem with accuracy. One the other hand, one thing we know how to do very quickly is multiplication.

This then is the plan: use logarithmic division to get an estimate for N, the integer part. Then calculate N*dy, compare with dx, and adjust the integer part accordingly.

A quick reminder of how logarithms can be used for division:

```
   log(a/b) = log(a) - log(b)
   exp(log(x)) = x
```

thus

```
   a/b = exp(log(a)-log(b)).
```

How do we take the log of a 9-bit number? We don't. Instead, we construct a table of f(x)=log(2*x), and use, not x, but x/2, as a parameter. Remember that the logarithms merely give an estimate to the integer part.

Moreover, if the tables are constructed carefully we can insure that the estimate for N is either exact or too small. Thus we only need to check for undershoots, which simplifies the calculation considerably. In particular, the tables were constructed as follows:

```
   10 DIM L1%(160), L2%(200), EX%(255): C=255/LOG(160)
   20 FOR I=1 TO 160
   30 L1%(I)=INT(C*LOG(I))
   40 NEXT
   50 FOR I=2 TO 200
   60 L2%(I)=INT(C*LOG(I/2)+0.5)
   70 NEXT
   80 FOR I=0 TO 255
   90 EX%(I)=INT(EXP(I/C))
   95 IF(I=129)OR(I=148)OR(I=153)OR(I=81)OR(I=98)THEN EX%(I)= EX%(I)-1
   100 NEXT
   110 L2%(3)=L2%(3)+1
```

The constant C is needed obviously to improve accuracy (log(160) simply isn't a very large number). Note that I divided the arguments of the logarithms in half; instead of calculating 2*dx/dy I calculate dx/(dy/2), which is of course the same thing. This was done to make C work out. By 'fixing' the tables in this manner, exactly 3927 calculations will undershoot, which works out to about 6% of all possible calculations we may perform.

The actual division routine works out pretty slick in assembly:

```
   DIVXY    MAC      ;Macro to compute 2*X/Y
            LDA LOG1,X  ;This is the division part
            SEC
            SBC LOG2,Y
            BCS CONT
            LDX #00     ;dx/dy < 1
            LDA ]1      ;LDA dx, since dx is exactly the remainder
            BCC L2
   CONT
            TAX
            LDA EXP,X
            TAX      ;X is now integer estimate
            STA MULT1
            EOR #$FF
            ADC #00     ;Carry is guaranteed set
            STA MULT2
            LDA ]1      ;ldxlo or rdxlo (i.e. low byte of dx)
            ADC (MULT2),Y
            SEC
            SBC (MULT1),Y  ;Calculate remainder
   L2       CMP ]2      ;ldy or rdy (i.e. ]2 = dy)
            BCC DONE
   L1       INX      ;Remainder is too large, so up int estimate
            SBC ]2      ;and subtract dy
            CMP ]2      ;Repeat until remainder<dy
            BCS L1
   DONE     <<<      ;Now X contains integer, A remainder
```

Do you see how it works? First the initial guess N is calculated. If log(x) - log(y/2) is negative then dx/dy is less that one, so the remainder is simply dx and the integer part is zero. Otherwise, R= dx - N*dy is calculated. Since N always undershoots, dx-N*dy will always be positive, so the high byte of dx isn't needed. This quantity R is the remainder, so if it is larger than dy simply increase the integer estimate and subtract dy from R, and repeat if necessary.

The end result then is a 9-bit/8-bit divide which takes 52 cycles in the best case. Pretty neat, huh? And quite adequate for our purposes.

Wait just a minute there, bub... what about when dy=0? Consider what dy=0 means: it means that two vertices lie along the same line. That in turn means that the next vertex can be immediately skipped to. That is, simply move on to the next point in the list, be it to the right or to the left, if dy=0.

Well, ah reckon that that just about completes the polygon fill routine. To summarize: start at the bottom (top, whatever) of the polygon. Calculate the "slopes" of the right and left lines from that point. Update the coordinates, fill the in-between parts, and plot the end-sections. Then update Y and keep going. If another vertex is hit, then recalculate the corresponding line.

Alert people may have noticed that this algorithm translates very nicely to the 128's VDC chip.

I should probably briefly mention the pattern fills. I use Y as an index to a pattern table, so it was very natural to use 8x8 character patterns. With different indexing of course more complicated patterns can be used. Moreover, it dawned on me that animated patterns were just as easy as normal ones, so I tried to think up a few interesting animated patterns (there are two in Polygonamy, each pattern is eight frames).

So that's the graphics part, more or less.

We ain't even CLOSE to being done yet.

@(A): 3D Code

Now it's FINALLY time to start writing the master program to control the 3D world. Luckily we have the BASIC program from waaaay up above to work from.

First is to decide how angles will be measured. The smart thing to do is to let the angle variable vary between 0..127 or 0..255; that is, to measure angles in units of 2*pi/128 (or 2*pi/256). The reason this is smart is because the angle is now periodic, wrapping around 256. Angles can be added together without checking for overflow, etc. (257=1, 258=2, 259=3, etc.). Note that in previous programs I did a very dumb thing and let the angle variable vary from 0..119, so angles were measured in three degree increments, and I had to place all sorts of checks into the code. Polygonamy uses angle increments of delta=2*pi/128.

Next there is the issue of cx=cx+20 instead of cx=cx+1. The problem is that if cx=cx+1 is used it takes forever to move around in the world. Moreover, the objects get really small at around cx=5000. What this means is that in the assembly version we can use a single byte for cx, and just treat each unit of cx as 20 "real world" units. That is, in the assembly program, we will keep track of cx/20 instead of cx.

Sort-of.

Consider the rotation which takes place when we turn left or right: the world is rotated through an angle delta=2*pi/128, so the calculation is:

```
   blah= cdel*cx + sdel*cz.
   cz= -sdel*cx + cdel*cz
```

The problem is that sin(2*pi/128)=0.049 and cos(delta)=0.9988, which means that, in practice, cdel*cx=cx. Equally bad is that sdel*cz is very small when cz starts to get small (e.g. 10*sdel = 0.49). The result of this is that objects close to the origin (e.g. us) will not be rotated at all!

Thus the centers need to be calculated more accurately. In particular, a second byte is needed to store the 'decimal' part of the center. To be precise, this second byte will contain the decimal part of the center times 256. This way we can add and subtract remainders and any over- or underflows will then affect the integer parts cx,cy,cz.

Very quickly we should decide how to represent remainders of negative numbers. A number like -1.5 can be represented as -1 - 0.5, but it can just as well be represented as -2.0 + 0.5. By using the second method remainders are always positive, and that's the smart way to do things (if nothing else it lets the remainder be a fraction of 256, instead of a fraction of 128). It's also the way any computer will round: type INT(-1.5) and see what happens.

further question arises about how to represent the centers, specifically, how do we represent an object which is behind us, e.g. has a negative value for cx. The normal way to represent negative numbers is of course to use 2's complement notation, but this has some disadvantages. One of them is multiplication: recall that in an earlier code some really fancy footwork needed to be done just to be able to multiply numbers between -64..64, and we certainly want the centers to range over more numbers than that. This gets worse if we decide to use more bits to represent the centers, as we must do if a larger world is constructed.

Moreover, Polygonamy is an excuse for testing new and different ideas and investigating their strengths and limitations, so why not try something different. As I look through my notes I'm not really sure what motivated this choice, but how about the following: let's add 128 to all of our numbers.(I think this is called excess-128 notation). -2 will be represented as 126, -1 will be represented as 127, six will be represented as 134, etc.

Shifting between the excess numbers and 'real' numbers is as simple as EOR #128. Recall that to multiply two numbers, let f(x)=x^2/4, so that a*b = f(a+b) - f(a-b). In the new system:

```
   xo = 128+x
   yo = 128+y
```

which means:

```
   xo+yo = 256 + (x+y)
   256+xo-yo = 256 + (x-y).
```

The 256 added above can be thought of as the carry bit. What this means is that all that is needed is to construct a single function,

```
   f(x) = (x-256)^2
```

where x=-255..255. We can now very quickly multiply signed numbers in the range -128..128, and with just a single (albeit 512 byte) table, using essentially the same multiplication procedure as before.

Now the downside of this method: adding and subtracting excess-128 numbers, and in particular checking for overflow.

```
   xo+yo = 256 + (x+y)
   if x+y >= 128 then we have overflow
   if x+y < -128 then we have underflow
```

which implies:

```
   xo+yo >= 256+128  implies overflow
   xo+yo < 128   implies underflow
```

with similar results for subtraction. Note also that after every addition or subtraction 128 needs to be either added or subtracted from

the result, which either way corresponds to an EOR #$80. So it's a
little more work to add numbers in this system. (Of course, adding
normal numbers to excess-128 numbers is no problem, so INC and DEC work
fine).

Back to rotations. The most obvious thing to do is to create two tables:

```
   f(x) = (x-128)*cos(delta) + 128
   g(x) = (x-128)*sin(delta) + 128
```

but remember that the remainders are also needed:

```
   fr(x) = 256*(remainder((x-128)*cos(delta)))
   gr(x) = 256*(remainder((x-128)*sin(delta)))
```

Since remainders are always positive none of this excess-128 junk is
needed. Note that we could also let f(x) and g(x) be 2's-complement
tables, then convert from two's-complement into excess-128 after
performing additions etc. The conversion is, what do you know, EOR #$80.
This is the smarter thing to do, and an even smarter thing to do is to
let the cosine table (f(x) above) to be in excess-128 format, and the
sine table g(x) in 2's complement. This way the numbers can be added
as normal, and no conversion need take place:

```
* Compare to BaSiC subroutine rotl: above

   ROTL
            INC THETA
            LDY #NUMOBJS   ;Y indexes the object
            DEY

   :LOOP    LDX CX,Y ;center coordinate
            LDA CDEL,X  ;CDEL = f(x) above
            LDX CZ,Y
            CLC
            ADC SDEL,X
            STA TEMP ;t1 = ci+si
            LDA CXREM,Y
            CLC
            ADC SDELREM,Y  ;Add remainders
            BCC :CONT1
            INC TEMP
            CLC
   :CONT1   LDX CX,Y
            ADC CDELREM,X
            BCC :CONT2
            INC TEMP
   :CONT2   STA CXREM,Y

            LDX CZ,Y
            LDA CDEL,X
            LDX CX,Y
            SEC
            SBC SDEL,X
            STA TEMP2   ;t2=cz-si
            LDA CZREM,Y
            SEC
            SBC SDELREM,X
            BCS :CONT3
            DEC TEMP2
   :CONT3   LDX CZ,Y
            CLC
            ADC CDELREM,X
            BCC :CONT4
            INC TEMP2
   :CONT4   STA CZREM,Y
            LDA TEMP2
            STA CZ,Y
            LDA TEMP
            STA CX,Y
            DEY
            BPL :LOOP
```

Well, that takes care of two lines of BASIC code :). As it turns out,
using a single byte for the remainder does a pretty good job of holding
the number. Rotating by 360 degrees one way, then rotating back again,
produces a center which is within a few decimal places of the starting
value.

Next up: projections. The projection calcuation is:

```
   Proj(P) = d*(P+C)/(px+cx)
```

where P=(px,py,pz) and C=(cx,cy,cz). In terms of the implementation, we want to calculate:

```
   d*((P-128) + s*(C-128)) / ((px-128) + s*(cx-128))
```

where s=20, to translate C into the 'real world'. To calculate this, consider the following function:

```
   g(x) = r*d / (s*((px-128)/s + cx-128)) + 128
```

where r is some scaling factor. The projection calculation then becomes:

```
   1/r*( (g-128)*(P-128) + s*(g-128)*(C-128) )
```

Thus we need some more tables, one of 1/(4r) * (256-x)^2, the other of s/(4r) * (256-x)^2, to do the multiplication. Furthermore a table of (x-128)/s would be pretty handy, and finally we need a table of g(x) = r*d/(s*(x-128)) + 128.

The general outline of a program would be:

```
   Get keypress
   1- If translate, then update all cx's (just some INCs and DECs)
   2- If rotate left or right, then rotate world
   3- Update angles for global & local rotation matrix (e.g. theta)
   4- Figure out which objects to display/construct a list
   5- Call each object in turn
   6- Update bitmap: clear out remaining garbage and swap bitmaps
```

Numbers 1 and 2 are done. In number three, by global matrix I mean the object rotation that results from us turning left or right. By local rotation I mean rotations independent of whether or not we turn. The local rotation allows e.g. the octahedron to spin around in Polygonamy.

Figuring out which objects to display is easy: just check to make sure it lies within the viewing cone/pyramid, that we are not too close, etc. If an object is to be displayed, it needs to be placed in a list. I constructed the list to make sure that objects which are farther away are drawn first; that way objects can overlap one another correctly. This was done via a simple insertion sort -- i.e. bump up objects in the list until the right spot is reached to insert the object.

We have most of the tools to deal with #5. Handling an object consists of rotating and projecting it, then displaying it. Rotation is the same as it has always been, albeit now involving sixteen bits, and projection is described above. Then each polygon needs to be drawn, by sticking the points of the polygon into the polygon list, setting up the fill pattern and the pointer to the minimum Y-value, and calling the polygon fill routine. Of course, if the face is hidden then we certainly don't want to plot it.

The minimum y-value can be found very easily while inserting the points into the point list -- just keep track of ymax and compare to each point as it is inserted.

We have discussed several methods of calculating hidden faces -- cross-product, rotated normal, parallel faces -- each of which involves looking at a vector normal to the face, and either projecting it or taking the dot product with a vector going to the origin. What a big pain in the butt, especially since values can be sixteen-bits, etc.

Did you ever stop to wonder about what happens to all the previous polygon-fill calculations if the point-list is entered in reverse order? Quite simply, left -> right and right -> left. And what happens when a face is invisible? The polygon is turned away from our eye. The points in the polygon, which go counter-clockwise around the polygon, will go clockwise when the polygon is turned around. (I should point out that at least in my code the points on the polygon are actually done in clockwise-order, since projection reverses the points).

So, we have hidden-faces already built-in to the polygon plot routine! In essence, we simply don't plot any polygon which the routine will freak out on. We can of course be systematic about this; within the plot routine:

```
   - Calculate the left and right lines.
   - Take a trial step along the left and right lines
   - If xleft < xright then we are OK, otherwise punt.
```

In principle we only need to do this on the first calculation, and use some properties of the lines to make things easier (for instance, if the left line is moving left and the right line is moving right, and they emanate from the same point, we know the polygon is visible). Unfortunately, nasty situations can arise, for instance when the left and right slopes have the same integer parts. So a check needs to be placed within the fill code to make sure the left point doesn't get ahead of the right point. This is unfortunate, as every cycle counts in the fill code, but luckily there is (was) a natural place to put in a quick check.

All that is left then is #6: run through the fill table, and clear any lines that still have old junk in them. Since I used two bitmaps as a double-buffer, all that is left is to swap the bitmaps, and do it all again.

Et voila.

@(A): Analysis and Conclusions

As you can see, the program is not without its flaws. The biggest one, I think, deals with the projection. Recall that I calculate px/s, where s=20, and add it to cx. My feeling was that px was going to be very small compared with cx, and so not modify the projection by much. But either this is a bad assumption, or the rotations are all screwed up, because certain rotations look a bit goofy. For instance, when you walk up close to the octahedron it starts to get jumpy, or wobbly. I note further that when you are far away from an object it looks much better, so that might be a way to fudge around the problem (e.g. make the value of d in the projection much larger).

Speaking of rotations, the 'funky shake' which used to plague the old programs has now been fixed. For instance, a rotation in the y-direction would work well but at some point it would appear to start rotating backwards, then start going the right way again. The problem was due to an overflow in the calculation of the rotation matrix, in a term that looked like(sin(t1) + sin(t2) - sin(t3) - sin(t4))/4, and the solution is to split such terms into two, e.g. (sin(t1)+sin(t2))/4 - (sin(t3)+sin(t4))/4.

Speaking further of rotations, I find the current system of rotation and projection unsatisfying, in particular too slow. Notice how much the program slows down when all three objects are visible; some 40 points are being rotated, both locally and globally, at this point. It is possible to reduce the number of matrix multiplications from 9 to 8 (and even lower, with lots of extra overhead), but I find this unsatisfying. A better method is needed...

There is another bug somewhere in the global rotations which sometimes causes the objects to wander around -- occasionally I can get the ship or the octahedron to move close to the pyramid. Also, when you are really close to an object and turn, you might notice the curious effect of the object rotating by small amounts, and then jumping position by a large amount. This is due to the 'units of 20' that are used in the program; the remainder part of (cx,cy,cz) needs to be used here, and then the display will be smooth as well.

Of course, if multicolor mode was used many of the calculations would be much simpler, since the screen x-coordinate would only require one byte instead of two.

The program should be made more efficient memory-wise of course. Shoving the fill routines for each buffer together would help out, and a system for rotating points out of a list, similar to that used in the last 3D program, would greatly streamline things (although it would be a tad slower).

There is still a minor bug or two in the fill routine, which causes little blue chunks to be taken off the ends of some polygons, but I didn't feel like tracking it down.

Note that although Polygonamy only lets you run around in a plane, running around in a full three dimensions is quite simple to add. And, although there are only three objects in the world, it is all set up to deal with a lot more. In summary, I see no major problems standing in the way of doing reasonably fast 3D graphics on the 64.

The object file for this article is available in "Hacking the Code" (Reference: code, SubRef: polycode) found elsewhere in this issue.

==

@(#)usenet: UseNuggets

COMP.SYS.CBM: The breeding ground of programmers and users alike. Let's
see what topics are showing up this month:

@(A): What is the HECK is BCD?
   As most ML programmers know, the 65XX CPU line has a arithmetic mode
   called "decimal mode", and is used to manipulate Binary Coded Decimal
   numbers (BCD). BCD numbers treat each nybble as a decimal digit.
   Possibel values for a byte than ar $00 to $99. Some fool asked on the
   group what earthly use BCD has on the 65XX CPU. Well, among other things,
   Willem-Jan Monsuwe shared this tidbit:

      I recall someone asking what the use of BCD (Binary Coded
      Decimal) was. I have here a 99-byte program that uses it to
      print out a number stored in the memory in decimal, with a
      maximum of more than 10^500 digits, Within 5 seconds ;).
      What's the use ?? Well, you can impress your friends by
      calculating the answer to the chessboard-problem (2^64
      - 1 or 0xFFFFFFFFFFFFFFFF) within 0.06 of a second. Oh,
      and the maximum is pretty easy to overcome, with a slight
      code change, if anyone needs numbers greater than, oh,
      509 digits.. ;)

```
      *         = $1000

      SNUM      = $1100
      BUFF      = $1200

      PRINT     = $FFD2

      SNUMPTR  = $FB
      SNUMBF   = $FC
      BUFFEND  = $FD

                  LDA #0
                  TAX
      CLRBUFF     STA BUFF,X
                  INX
                  BNE CLRBUFF
                  SED
                  STA BUFFPTR
                  LDY #210
                  STY SNUMPTR
      BYTELOOP    LDA SNUM,Y
                  STA SNUMBF
                  LDY #8
      BITLOOP     ASL SNUMBF
                  LDX #0
      ADDLOOP     LDA BUFF,X
                  ADC BUFF,X
                  STA BUFF,X
                  INX
                  BCS ADDLOOP
                  CPX BUFFEND
                  BCC ADDLOOP
                  STX BUFFEND
                  DEY
                  BNE BITLOOP
                  DEC SNUMPTR
                  LDY SNUMPTR
                  CPY #$FF
                  BNE BYTELOOP
                  CLD
                  LDA #13
                  JSR PRINT
                  DEX
                  LDA BUFF,X
                  AND #$F0
                  BEQ LOWNYB
      PRINTLOOP   LDA BUFF,X
                  LSR
                  LSR
                  LSR
                  LSR
                  CLC
                  ADC #48
                  JSR PRINT
      LOWNYB      LDA BUFF,X
                  AND #$0F
```

```
               CLC
               ADC #48
               JSR PRINT
               DEX
               CPX #$FF
               BNE PRINTLOOP
```

@(A): Commodore's Can't Compute! (or can they?)
   OK, try the following on your beloved 128:

```
      print 23.13 - 22.87   hit RETURN
```

      Do you get .260000005?

   The resuling thread after this question was posed started to lean in the direction of attacking the arithmetic units of BASIC in the Commodore 8-bit machines. Then an eloquent post from Alan Jones (alan.jones@qcs.org) started to set the record straight. We can't express it any better than Alan:

> Recently, the C64/128 floating point arithmetic has been maligned here. The C64/128 has good floating point math. It uses 5 byte reals with a 4 byte (32 bit) mantissa. There are no bugs in the basic FP arithmetic. The reals ARE limited in range and precision. They are more useful than compters using 32 bit reals, but not up to IEEE standard arithmetic. IEEE FP arithmetic (double and extended precision...) would be much slower than our existing FP routines. Of course it might be possible to interface a hardware FPU to the new Super64/128CPU (65816).
>
> The other C64/128 FP routines, such as SIN, EXP, and functions that use them are not accurate to full 32 bit FP precision. When used with care, they are often accurate enough for engineering work.
>
> The most annoying inaccuracy may be the conversion between binary FP and decimal for I/O. BASIC only prints 9 decimal digits of a FP number, but our binary FP number has about 9.6 decimal digits of precision. What you see is not what you have! Of course there are some simple tricks that you can use to print the FP number with more decimal precision, and you could do I/O using HEX notation. If you save intermediate results for later use, make sure you write the FP values as binary rather than ASCII (converted to decimal).
>
> If you do accounting type stuff with dollars and cents, using binary FP with its limited precision and rounding can be anoying. If your results are off one penny, all of your work will be suspect. Our 6502 family of CPUs also has decimal arithmetic. It can do decimal arithmetic exactly, although you may have to program it yourself. I think the Paperclip word Processor will do simple calculations with up to 40 decimal digits of precision.
>
> If you are using 64+ bit FP you can compute some things in a fast and sloppy manner. Some programs that work OK on an IBM PC or workstation need more careful attention when coded for a C64/128.
>
> Some numbers can not be represented exactly in binary FP formats of any precision. If you want to calculate:
>
> ```
>   a:=(1/3)*(1/5)*(1/7)*(1/11)
> ```
>
> You should code it as:
>
> ```
>   a:=1/(3*5*7*11)
> ```
>
> Aside from being faster, it is more accurate.
>
> There are many tips for preserving numerical accuracy in computations. There are often interesting tradoffs between computation speed, memory usage, and accuracy and stability. There are even some C64/128 specific tips. (e.g. we usually store a real value in 5 bytes of memory but push it onto a stack as 6 bytes when we want to use it.)
>
> This is not intended to be a Commodore FP tutorial. It is reminder that the C64/128 can be used for "heavy math", and there are no bugs in the Commodore +, -, *, /, Floating Point arithmetic routines. It uses 32 binary bit mantisa FP reals with proper rounding.
> Simple examples can always be contrived to demonstrate a perceived FP bug by computer illiterates(?).

   Alan got his dig in at the end there. That post, and others like it, pretty much squelched the arithmetic discussion. But, as is usually the case, we all learned a neat trick along the way. Peter Karlsson shared his easy way of determining whether his programs are running on a C64 or C128 by issueing the following statement:

```
C=64+64*INT(.1+.9)
```

Since the 64 and 128 differ ever so slightly in their arithmetic routines, the above line gives 64 on a C64 and 128 on a C128.

@(A): We need another OS!

It all started when Benjamin Moos posted a message in the newsgroup mentioning that he had been off the net for a while but was returning and wondered whether anyone would want him to finish work on a C++ based comiler for the GEOS 2.0 environment. Of course, everyone was for that, but Moos continued on, asking if there was any interest in an alternate OS for the 64 or 128. Moos mentioned that he had been also working on Common Graphic OPerating Environment (CGOE), and was thinking about finishing the project, which would provide a C= graphics character based graphic windowing system that would allow all 64 programs to run in the 128 80 column screen in 40 column windows.

Well, that brought out some friendly debate, to state the obvious. Part of the group posted words of encouragement, noting that we need to support those programming for the environment. The other half of the camp echoed the words of Patrick Leung, who expressed concern that there are many programmers in the arena that are doing the same thing separately. He encouraged programmers to consolidate features and code bases to arrive at robust full-featured programs instead of fragile bare-bones applications that single programmers can't support. ACE, Craig Bruce's UNIX-like OS detailed in earlier C=Hacking issues, was brought up by some, who asked that programmers heed Leung's advice and build modules for the already supported ACE environment.

Perhaps J. Shell has the best idea, as he is planning to set up an interactive WWW site to allows programmers to work with him to build COMMIX System II (CX2). The site will allow programmers to bring new ideas to the table and have them rapidly incorported into the design. We'll see if Mr. Shell can deliver on this neat idea.

Going full circle, Benjamin Moos reponded to some of the posts, saying that the OS work was going to be placed on the shelf for now, as many had expressed interest in the C++ like compiler. However, he did say that work would begin again at a later date, but no decision was made as to how he will proceed.

@(A): The "More Power" Swiftlink

Ever striving to squeeze the most performance out of his C128 system, Craig Bruce modified his Swiftlink and lived to tell about it in the newsgroup.. Basically, after researching the data sheets for the 6551 ACIA IC used in the SL, Craig noted that Dr. Evil Labs (the original creators of the SL) had used a double speed crystal to up the 19,200 bps maximum in the ACIA to 38,400 bps. The IC claims that any baud rate up to 125,000bps can be achieved with the IC, given the correct crystal frequency. Well, another feature of the 6551 is to use the crystal frequency/16 as the bps rate, which is 230,400 bps or the stock crystal. Too fast for the IC. However, by replacing the crystal (a 3.6864 MHz unit) with a 1.8432 MHz unit, the 1/16 speed becomes 115,200. That speed, less than 125,000 bps, is the standard top frequency for IBM UARTs and is supported by most newer modems. Craig verified that his 2MHz 128 can keep up with the extra data that his modofoed SL allows him to receive, but not always. he claims that every once in a while, the systm gets choked up and crashes, so he is working on solutions. Understandably, one will need very tight terminal program code to keep up with this speed, but it will fit nicely with the SuperCPU.

As with all things, there is a downside in that 19,200 becomes the next lower bps rate. 38,400 is gone forever. Craig speculated that perhaps a switch could be installed, but wasn't sure of the effects.

@(A): The Eternal Problem

Although this didn't receive much discussion, C=Hacking feels many users can relate. How many have ever went into the local CompUSA of local computer store and asked to look at modems, printers, or SCSI drives, only to hear the dreaded laugh and chide that you should "buy a REAL computer", or watch the quizzical look of the sales person as they exclaim "You can't hook that up to a Commodore!" We particularly enjoyed the ending to the lament that appeared in the newsgroup:

> When someone keeps an old car around, babies it, works on it, adds to it, drives it around in style, no one says "Look at

this dummy driving an out-dated gas guzzler that can't even do
80, and gets atrocious gas mileage. The frame is archaic.
The windows aren't electric. Why doesn't he upgrade?". Nah..
they are 'enthusiasts of classic automobiles'.

Well, ... we are "enthusiasts of a classic computer".

==

## @(#)fido: FIDO's Nuggets

The CBM and CBM-128 FIDONet echoes. The place where Commodore users
unite. Let's what things they discussed over the past month or two:

@(A): UNZIP 2 or not UNZIP 2?

For a while now, Commodore users have been able to uncompress
archives created with the popular PKZIP 1.01 compression program
by PKWare or one of its clones. Well, PKWare upped the ante and
upgraded the PKZIP product to version 2, and that left a bunch of
Commodore users compressed! The easy solution is to ask all
archive creators to not use version 2 of the ZIP product, but that
presents a problem. Most of the FIDONet crowd reads their mail
offline via popular programs like QWKIE 3.1 on the 64 or QWKRR128
4.3 on the 128. The programs work by retrieving a COMPRESSED
packet of news and mail from a BBS. Well, it turns out that BBS
systems have migrated over to the new version of ZIP, and some
refuse to offer ZIP version 1 as an optional compression method
for retrieval packets. So, the FIDONet crowd, including David
Schmoll and others, have been working on or searching for a way
to bring PKZIP 2 functionality to the 64. Some thought it was a
done deal when a FIDNetter contacted Info-Zip, the authors of a
free clone of PKZIP 2 by the same name. They were told the source
code was available. The catch, it is written in C, and so far,
no compilation on the 64 or 128 has been successful.

@(A): QWKIE v3.1 FREE!

Many C64 users have delighted over the use of QWKIE v3.1 to read
offline news and mail. However, many had been unable to register
the product. The mystery was solved as of late a letter by the
author was read that stated that he was ceasing support for the
product and had placed it into the public domain. As well,
interested programmers could contact him about source code. So,
QWKIE FREE, a patched version of the program that is marked as
registered, was uplodade to the many CBM BBS systems for users to
enjoy.

@(A): That Darn Internet!

As of late, many FIDONet regulars have been diappointed in the
trafiic flow on the CBM echoes. They blame the growing
popularity of the Internet as one reason the amount of messages
has dwindled. Almost immediately, reasons why FIDONet is still
useful started popping up in the echoes. Many claim that the
Internet and FIDONet are complementary for the Commodore user, and
that both resources are needed. Others, however, stressed that
FIDONet is still the most useful. While C=Hacking isn't going
to cast a vote here, we do hope that interest in the echoes stays
high, as some only have access to FIDONet, and Commodore support
should be on every network.

@(A): Let's Randomize

Some soul on the echo was looking for a way to generate a random
number from 2 to 350. Well, always eager to help, many FIDONetters
came to the rescue, with varying degrees of complexity.

The first post, by Ken Waugh, included the text from one of
Rick Kepharts WWW Site pages that explained, in two BASIC lines or
less, how to create a set of 255 nonrepeating random numbers.

Then, ever the guru, George Hug, of 2400 bps on a 64 fame, described
a method to find random numbers based on "linear maximal length
shift registers", complete with 3 part article on the method. Wow!
needless to say, the method looks promising, but was probably more
than what the original author was looking for. Nonetheless, the
treatise looks worthy of inclusion in an upcoming C=Hacking issue.

@(A): Catch the Wave!

By now, most know that Maurice Randall, the author of GeoFAX, has been
working on a GEOS telecommunications program that will operate at the
14,400 bps or better mark. It's been discussed in both USENET and
FIDONet before, but Gaelyne Moranec reopened the discussion with a
statement that World Wide Web page viewing support might possibly

be incoporated and under test. Mr. Randall was hoping to add such
support at some time, but it was unclear when. It looks like sooner
rather than later.

@(A): Who's First?
Rod Gasson posed an interesting question on FIDONet a while back. He
asked which CPU was in control of the 128 when it is first powere up.
The abvious answer of "the 8502" was given many times over, but Rod
finally noted that it is, in fact, the Z-80 in the system that gains
control of the system first. Herman Yan supplied the relevant page
from the C128 Programmer's Reference Guide that explains the
reasons. If you want to know more, check out page 576 in the manual.

@(A): Desterm Confusion
Many in the 128 arena use a telecommunications program called Desterm,
by Matt Desmond. At present, there are two versions of the shareware
application out, 2.00 that works on all drives except the RAMLink, and
2.01 that works with RAMLink, but has bugs not present in 2.00. So,
which to use? That questions gets asked in verious forms in the echoes
repeatedly. That, coupled with the inability to find Mr. Desmond for
a while, the supposed hand-over of the code to Steve Cuthbert, and the
recent emergence of Matt (Reference: news) added to the confusion. The
current thinking is that Matt will be working on a new release of
Desterm that will include CTS/RTS support (the present version only
supports XON/XOFF flow control) and some bug fixes. Somehow, the
rumor that Matt will add Z-modem capabilities keeps pooping up, but
Matt has denied any such work. He merely doesn;t see a need, since
add in modules can be created to do this.

So, that gives you a glimpse into the world of FIDO, the wonder dog of
networks. C=Hacking laments that their own FIDO feed has been
experiencing problems as of late, so we too may have missed some juicy
tidbits. We'll catch them later on, though.

Here, boy....

==

@(#)cmdcpu: Underneath the Hood of the SuperCPU
by Jim Brain

Does your mind go blank when you hear about the SuperCPU? With all the
mention of it in magazines and newsletters, are you left wondering how
much of the discussion is hype and how much is true? Are you worried
that this latest attempt is just another design destined for failure
like the others before it? Well, if so, then you're not alone. With
the reputation accelerator cartridges and their manufacturers have
acquired over the years, you are wise to be concerned. Judge for
yourself, as we peer under the hood of the Creative Micro Designs
SuperCPU accelerator cartridges.

Note: The information contained in this article has been gleaned from
talks with CMD, Mr. Charlie Christianson's post to comp.sys.cbm,
responses to USENET posts by Mr. Doug Cotton, and information from Commodore
World Issue #12. While general information is not likely to change,
some details discussed in this article may differ slightly from those
incorporated in the final product.

@(A): What's An Accelerator?

Did you know a Commodore 64 CPU executes things at 1 MHz? A tiny clock
inside the 64 ticks off 1 million "cycles" per second, and instructs
the CPU to move forward one cycle at a time. The CPU, in turn,
either executes an internal operation, reads from memory, or writes
to memory during that cycle. These operations are concatenated to
form funtions, which is the smallest piece of work a programmer can
ask the CPU to perform. These function are called instruction, and
take an average of 3 cycles each to perform. So, the typical C64
CPU does 333,333 things a second. The C128 fares a bit better, as it
can run twice as fast when in "fast" mode. In either case, there is
an upper bound on the amount of useful work each CPU can do in a
amount of time.

An accelerator increases that amount of work done by substituting a
faster CPU and clock speed for the 1 MHz 64 CPU. The ratio of
increase should be as easy to determine as dividing the new clock
frequency by 1 MHz for a 64. If this were true, an accelerator that
runs at 4 Mhz would execute things at 4 times the speed of a stock 64.
Sadly, this is not true, since not all parts of the system can be sped
up to the higher frequency. So, the accelerator runs at full speed while
it utilizes ICs designed for the faster clock speed, and slows down when

it must "talk" with ICs like the SID and VIC-II in the 64, which run only at the slow 1 MHz clock speed.

Most accelerators are produced as large cartridges that plug into the expansion port of the computer system. Some require special wires be attached to internal components, while others do not.

@(A) The New Kid on the Block

In mid 1995, Creative Micro Designs, after having evaluated the FLASH 8 accelerator from Europe with only mild success, noted that there might possibly be a market for a speedy accelerator that would run GEOS and other useful applications in the USA. After surveying the readership of Commodore World, the Internet, and FIDONet, CMD decided that interest in such a unit was forthcoming. Shortly thereafter, the SuperCPU announcement was made.

As development work ensued, progress reports and preliminary information about the product surfaced from CMD. The first items involved the processor choice, which was originally the 65C02S but is now its bigger brother, the 16 bit 65C816S. Another piece of information involved the case, which is an enclosure 6" wide by 2" deep by 3" wide. This enclosire contains a circuit board protruding from the front of the unit that will plug into the Commodore 64 or 128 expansion port. In back, a complementary card edge connector is provided to pass signals through the cartridge. This will allow users to attach other expansion port cartidges to the system. On top sit three switches, described below.

The first switch enables or disables the SuperCPU unit. The second switch enables or disables JiffyDOS, which is built into the unit. The third switch determines the speed of the unit. This third switch has three positions. The first position forces the accelerator to operate at 1 MHz speed (the same speed as the stock C64). The second position allows the programmer to change the speed via a register in the SuperCPU memory map. The third position locks the SuperCPU into 20 MHz mode, regardless of register settings.

The use of the CMD SuperCPU will be straightforward. Simply plug the unit into the expansion port, set the appropriate switches on the top of the unit, and powering on the unit.

@(A) Technical Details

The basic system utilizes a WDC W65C816S 16 bit microprocessor running at 20 MHz. This CPU can not only fully emulate a CMOS 6502, it can be switched into "native" mode which allows access to 16 bit registers and 16 megabytes of RAM without bank switching, DMA, or paging.

Attached to the CPU is a bank of 64 kilobytes of Read Only Memory (ROM) and 128 bilobytes of high speed static RAM (SRAM). The extra RAM above 64 kB is used to "mirror" the contents of the slower ROM. See below for details.

A number of features designed to maximize the performance of the SuperCPU are being developed into the unit. Since the late 1980's ROM speeds have not been able to keep pace with CPU clock frequencies. With the CMD accelerator moving into the frequency range of newer PC systems, this becomes a problem for the SuperCPU as well. The Commodore typically stores its KERNAL and BASIC code in ROMS, and the SuperCPU will need to read that code. The easiest solution is to read the stock ROMs in the computer, but those ICs can only be accessed at 1 MHz (they are part of that set of older ICs that cannot be utilized at 20 MHz). So, the next option is to copy that code into faster ROMs and instal those ROMs int the cartridge. Well, as stated earlier, ROMs of sufficient speed are very expensive and not widely available. So, the third option, which is the one CMD will use, is to copy the KERNAL and BASIC at startup to RAM and write protect the RAM area, making it look like ROM. Fast static RAM (SRAM) is available to meet the 20 MHz clock requirements, and is not terribly expensive, as most new PC systems use the same memory for similar uses. This technique is called ROM shadowing and has been utilized for a few years in the IBM PC community.

The heart of the unit is the Altera Complex Programmable Logic Device (CPLD). Analogous to electonic "glue", this single chip can replace ten or hundreds of discrete ICs in circuits. This unit is responsible for decoding the complex series of signals presented in the expansion port, handling DMA requests to an REU unit, emulating the specialize I/O port found at locations $00 and $01 on the 6510 CPU, and handling the synchronization of the SuperCPU memory and C64 memory.

One item that has plagued accelerator designers for years and minimized the widespread acceptance of accelerators invoves this RAM sync operation the Altera CPLD handles. In areas of the stock C64 memory map where only RAM is present, like $0002 - $40959, the synchronization of memory can be handled very easily. However, when dealing with areas like $d000, where RAM AND IO can be present, the situation becomes more complex. The SuperCPU overcomes this problem as well, which is important since many video applications use the RAM under IO at $d000 for graphics or text.

As the VIC-II IC in the C64 and C128 requires that screen information be present in on-board memory, memory "mirriring" is necessary. However, CMD has introduced two new technologies, called WriteSmart(tm) and CacheWrite(tm) to reduce the slowdown associated with mirroring the SuperCPU SRAM and the slower on-board DRAM. According to documentation, WriteSmart allows the programmer to decide which portions of memory need mirroring. The four selections include "BASIC", where only text and color memory are mirrored, "GEOS", where GEOS foreground bitmap and color memory are mirrored, "ALL", where all 64 kB of RAM is mirrored, and "NONE", where the SuperCPU does not attempt to syncronize memory contents between the two RAM areas.

The other technology, called CacheWrite(tm), minimizes the effect of this mirroring. When storing a value into SuperCPU RAM in a range of RAM that requires mirroring, the value is stored not only in SuperCPU RAM, but also into a special cache memory location. The SuperCPU is allowed to continue processing, while the system waits for the on board DRAM to acknowledge readiness to store a value. When successive stores to mirror ranges are done, the system must slow down, but can still operate at about 4 MHz. This speed is achieved because the SuperCPU need not wait for the value to be successfully stored before it attempts to fetch the next opcode and operand. Since opcodes that write value to memory avarage 4 cycles to complete, the SuperCPU can effectively do 4 cycles worth of processing in 1 period of the 1 MHz clock. Note that this slowdown does not occur if the cache is not full when a store instruction is executed.

@(A) Features

Being a CMD product, the CMD SuperCPU comes with JiffyDOS, CMD's flagship speed enhancement routines, installed. However, JiffyDOS can be switched out for those applications that fail to run with this serial bus enhancement functionality.

The unit also features compatibility with RAMLink, CMD's RAM drive unit. As the RAMLink fucntions by sharing the CPU with the computer system and runs a special set of instructions called RL-DOS, the SuperCPU contains its own version of RL-DOS optimized to take advantage of the speed and extra features available in the 65C816S. Preliminary information suggests that RAMLink data retrieval, typicially much slower the REU data retrieval, will now operate at speeds approaching that of the REU. In addition, the on-baord RL-DOS will handle usage of the special parallel CMD HD drive cable available with the RAMLink.

For those with expansion in mind, CMD has incorporated a special expansion port internal to the unit. The port, called the "Rocket Socket", will allow access to the complete signal set from the W65C816S CPU and possibly other support ICs. This will allow developers to produce peripheral cards for the unit containing hardware that will run at 20 MHz (The cartridge port will still be limited to slow speed).

@(A): Myths About the Unit

In the early phases of development, CMD hinted that possibly extra RAM installed in the unit could be used as a fast RAM disk, a la RAMLink. However, the inability to battery back up that RAM area, coupled with the small increase in speed gained form doing so and the lengthy development time needed to realize this feature, has prompted CMD to abandon this idea for the time being. Later in the development cycle, such an idea might resurface, but the feature is most likely never to be implemented.

Also, early information about the units noted that two speed options would be available, but low support for the slower 10 MHz model prompted CMD to discontinue development on that version. As of now, there is only one speed option available: 20 MHz.

When CMD first announced the unit to the public, it was to include the Western Design Center W65C02S microprocessor. However, in late 1995/early 1996, CMD opted to switch from that CPU to its bigger brother, the W65C816 16 bit CPU, owing to small increase in per item cost, more flexibility, and more expansion options.

Although the speed of the CPU in the SuperCPU unit is running at 20 MHz, that does not imply all operations will occur twenty times faster. Some operations, like reads from I/O ICs, derial bus operation, and mirroring of video memory, require the CPU to slow down temporarily. This will reduce the effective speed to about 17-18 MHz.

@(A): Compatibility Issues

All legal 6502/6510/8502 opcodes are supported in the accelerator. Undocumented or "illegal" opcodes are not supported and will fail.

Although not a compatibility issue, some applications that rely on the CPU running at a certain speed to correctly time events will most likely fail or operate too quickly to be useful. Event or interrupt driven code should operate correctly.

The SuperCPU 64 model will operate correctly with any C64 or C64C model of computer system, as well as with any C128 or C128D in 64 mode. However, CMD has recently announced a 128 native version of the cartridge.

@(A): Super128CPU

In early 1996, CMD announced that interest was compelling and that would begin development on a 128 version of the SuperCPU. As a result of this announcement, the ship date was moved from Februarty to April as CMD validated the SuperCPU design so that it could be used to manufacture both the SuperCPU 64 and SuperCPU 128. Both units will operate at a maximum of 20 MHz, and will most likely be packaged in the same enclosure. The SuperCPU 128 will operate in both 64 mode and native 128 mode. It will not enhance CP/M mode on the C128. CMD announced that the availability of this unit would be Auguest or September ot 1996. As far as cost is concerned, a current estimate falls at $300.00, and advance orders are being taken with a security deposit of US$50.00 needed to place an advance order.

As this announcement was made, some confusion has resulted in the naming scheme. Previously called the SuperCPU or SuperCPU 64/20 (64 model at 20 MHz), the new models are referred to as alternately:

128 model	64 model
Super128CPU	Super64CPU
SuperCPU 128/20	SuperCPU 64/20

@(A) Prototype Testing and Benchmarks

As no developer unit have shipped as of this date, CMD has the sole unit availabel for be testing and benchmarks. CMD's prototype unit consists of a handwired unit on perfboard. At first, CMD was hesitant that the prototype would actually run at 20 MHz, since such designs are not "clean" and can suffer from eignal degradation, signal skew, and crosstalk, which inhibits operation at higher frequencies. So, with that in mind, early tests were done at 4 MHz. CMD reported in late Fenbruary 1996 that the prototype had been ramped up to 20 MHz and was operating correctly. In fact, the unit appears to run faster than it can, illustrated by the following example:

CMD tested the following program at 1 MHz on a Commodore 64

```
10 TI$="000000"
20 FORI=1TO10000:NEXT
30 PRINTTI
```

The result from this test was 660. After enabling the unit, the test was rerun and the result printed out again: 31.

Quick calculations by the CMD personnel verified that the unit was executing this program 21.29 times the normal speed. However, that is impossible, as the CPU is only clocked 20 times the nortmal speed.

The supposed impossibility is explained if you delve deeper into the timing of the 64. As many know, the VIC-II "steals" cycles from the CPU in order to refresh the VIC-II video screen. Extra cycles are "stolen" for sprites. With the SuopercPU disabled, the above code runs at 1 MHz minus the amount of time the VIC-II "steals" from the CPU. With the SuperCPU enabled, the VIC-II does not "steal" cycles from the unit, as the accelerator uses it own private memory area for operation. The VIC, meanwhile, uses the on-board C64 memory.

CMD notes that games that use timers or are event driven function correctly, but hotse that count processor cycles or utilize spin-wait loops run so quickly as to be virtually unusable.

Of partiular note to Commodore Hacking readers is the test done with the object code for the Polygonamy (Reference: polygon) article elsewhere in this issue. On a stock 64, the program renderes approximately 12-13 frames per second. With the SuperCPU enabled, the frame rate jumped to 128 fps. CMD notes that further gains might be realized if the code was modified to cooperate more fully with the CupserCPU memory scheme.

As for Ram Expansion Unit compatibility, CMD responds that the issues have been tackled and that DMA operation is available on the SuperCPU unit. In adiition, CMD notes that the CPU need not be running at 1 MHz to initiate a DMA transfer.

As stated from the beginning, the 64 model of the SuperCPU accelerator wil work on the Commodore 128 in 64 mode, and test have confirmed that the prototype 64 model does indeed frunction correctly any the C128 and C128D.

@(A): Conclusion

While it is too early to determine the success of the CMD SuperCPU product, the company has a reputation for delivering stable products packed with features. While no accelerator can guarantee 100% compatibility with all Commodore software, the CMD offering should provide the best compatibility options thus far, due to its solutions to RAM synchronization problems that have plagued accelerator designers for years. The fact that CMD also owns the marketing rights to the GEOS family of software products and manufacturers a wide variety of successful mass media storage devices bodes well for compatibility with those applications and peripherals.

@(A): For More Information

TO find out more about the CMD SuperCPU family of accelerators, contact CMD at the following address of via email:

```
Creative Micro Designs, Inc.
P.O. Box 646
East Longmeadow,  MA  01028-0646
(413) 525-0023  (Information)
(800) 638-3263  (Ordering only)
cmd.sales@the-spa.com  (Internet Contact for Sales)
```

Advance orders are being taken for all units, and the cost to place an advance order is $50.00.

For programmers, CMD is planning to make available a Developer's Package, which will help those wanting to exploit the potential of the new unit to achieve success. A W65C816S assembler supporting all the new opcodes and addressing modes will be provided, as will documentation pertaining to the unit, the CPU, and its capabilities.

===

@(#)surf: Hack Surfing

For those who can access that great expanse of area called the World Wide Web, here is some new places to visit that are of interest to the Commodore community. In early 1994, when the US Commodore WWW Site started, the number of sites online that catered to Commodore numbered in the 10's. Now, the number is in the 100's. What a change.

If you know of a site that is not listed here, please feel free to send it to the magazine. The following links have been gleaned from those recently changed or added to the US Commodore WWW Site Links page (http://www.msen.com/~brain/cbmlinks/).

To encourage these sites to strive to continually enhance their creations, and because we like to gripe :-), we'll point out an improvements that could be made at each site.

@(A): Companies

o http://www.armory.com/~spectre/cwi.html
   Computer Workshops Incorporated. CWI shows off their newest software offerings on this well-crafted WWW site. The darkbackground provides for visual effects, and the content is good as well. At the time we "hit" the page, CWI was working on a new game called Nether for the 64/128. From the information, it looks like a 3D action adventure. CWI offers

both CBM and MS-DOS titles. Some are shareware, while others are commercial. C=Hacking gripe: We don't mind the MS-DOS information, but the diehard CBM user should be able to skip it. As of now, it's all on the same page.

o http://www.msen.com/~brain/guest/Gaelyne_Moranec/qwkrr/
   QWKRR128, by Rod Gasson. Gaelyne Moranec, a supporter of QWKRR, presents this site for new and advanced users. The site is devoted to QWKRR128, a QWK-based offline mail reading program for BBS and Internet use, and Browser, a utility for reading large files on the 64/128. The site is clean and simple, with no fancy graphics, but lots of meaty information. Links include the QWKRR128 user's manual, the actual product's binaries, and helper applications needed to use QWKRR128. C=H gripe: It's hard to tell what all I need to read Internet email via QWKRR128.

o http://www.msen.com/~brain/guest/rms/
   RMS Computer Systems. RMS offers up its line of services from this site, including software distribution, parts and accessories, and consulting/ training. RMS can even design your WWW pages. The pages are colorful and clean, using either Microsoft Explorer or Netscape Navigator extension depending on the browser chosen off the home page. RMS offers the C-Net BBS software for sale and present information on the 64 and 128 versions of the program. C=H gripe: The home page offers a choice of using Netscape of Microsoft Explorer. What about the Lynx text mode browser? Which do they pick?

@(A): Publications

o http://www.the-spa.com/cmd/cwhome.html
   Commodore World. CMD's publications is presented at this site, with select articles, and information for potential writers and subscribers is detailed. The site is laid out well and provides for easy reading. Of course, we're not sure it does justice to the magazine, but that's true of LOADSTAR's home page as well. C=H gripe: the site needs updating, as the change dates are 9-95. When they do update it, we hope they'll remove that annoying "blink" tag!

@(A): User's Groups

o http://www.ccn.cs.dal.ca/Technology/CUGNS/CBM.html
   Cnada Commodore Users Group of Nova Scotia. The site makes use of color and grpahics to provide links to a number of Commodore content sites on the Internet. It links up with other user groups on the 'Net, and provides a public download area for software retrieval. C=H gripe: We still think this is a user's group, but no meetings, minutes, newsletters, or times and dates were mentioned. Whare are they?

o http://www.fastlane.net/homepages/msessums/64.html
   Metro C-64/128 User's Group. Meeting dates, times, agendas, and some general information are provided on this page. You can also learn about this groups parent organization, the Metroplex Commodore Computer Club. C=H gripe: some newsletters from past meetings and a bit more about the group would be nice.

o http://www.inna.net/mpcug/mpcug.html
   The Middle Peninsula Computer Users Group. Go here to find out just WHY the groups is named this way. Meeting times, dates, places, past newsletter articles, and background information is provided. The site has a sprinkle of color and graphcis to break up the text. C=H gripe: It looks like the group is multi-platform, but no mention is made of what Commodore 8-bit owners will find at meetings. Maybe we missed it.

@(A): Demo Groups

o http://rphc1.physik.uni-regensburg.de/~pem03049/eqx/
   The EQUINOXE WWW Site. This demo group puts on a good show, with content and color on their WWW site. Here is where you can find the announcement on the upcoming Shout! #2 magazine release. The list of links is implressive as well. C=H gripe: The front page use of large fonts sizes is a bit overdone.

o http://flash.lakeheadu.ca/~jgvotour/index.html
   The OMNI/Revenge WWW Site. Color and content are mixed well on this site as well. A bit of history about Revenge is given, links to the demos to download is present, and information about upcoming releases is detailed. C=H gripe: We'd like to know more about the person behind the well-done page.

@(A): Miscellaneous

o http://vanbc.wimsey.com/~danf/cbm/languages.html

Dan Fandrich's Commodore Languages List. Extensive doesn't really
   describe this page, which provides information on assemblers, compilers,
   cross-compilers, and interpreters for many different programming
   langauges supported by the Commodore 8-bit. Rare items like language
   support for the 264 series and the SuperPET are described as well.
   C=H gripe: the page is HUGE. Any chance of a breakdown into spearate
   files?

o http://rrnet.com/~bfrandse/viccarts.html
   The Commodore VIC-20 Cartridge List. Cartridges from many different
   software companies are detailed, and both games and utilities are listed.
   As with the Programming Language Page, this list is extensive. It notes
   in the opening credits the trasnsitions the list has made to arrive at
   this current form. C=H gripe: same as for the langauge list. This thing
   is LARGE, and might benefit from a more heirachial listing treatment.

o http://fox.nstn.ca/~ptiwana/john/webpage1.html
   John Elliot's WWW Site. This page explores the use of Commodore computers
   and other "orphan" machines to better education by improving the student/
   computer ratio. The information presented in this site is heartwarming,
   as it shows practical uses to dispel the myth that 8-bits are truly
   useless. C=H gripe: Not really a gripe, but we sure would like to see more
   of these reall world examples.

o http://www.ksk.sala.se/~sp93rob/dungeon/
   The Alternate Reality WWW Site. For those wanting to relive the best of
   this game for the 64, visit this site. Everything from tips to tricks,
   stories to confidential material, and screenshots are available at this
   site. C=H gripe: The color scheme is a bit rough on the eyes, but it
   does look neat.

o http://www.lysator.liu.se/tolkien-games/c64.html
   Fredrik Ekman's Tolkien Games WWW Site. The name says it all. If
   you've ever played a Tolkien game, here is where they are listed and
   examined. Fredrik give the history of each game, the solutions if there are
   any, and describes the game itself. C=H gripe: We left impressed with the
   information but wondering why someone would go to this effort. Tell us,
   Fredrik.

o http://ubmail.ubalt.edu/~telliott/commodore.html
   Todd Elliott's Commodore 64/128 WWW Site. Todd provides some commentary
   and links to hardware hacks and ML tips. Of particular interst is his
   introduction to the Commodore C64 and C128 computers, which explains
   some of the history behind the machines. Our favorite passage in this
   page details his experisnces with Radio Shack.... C=H gripe: We'd like
   to know how Todd Elliott fits into the Commodore 8-bit arena.

@(A): Change of Address

o CMD recently moved to http://www.the-spa.com/cmd/
   CMD heard our issue #11 gripe, as the home page now has links directly
   to the SuperCPU information.

o LOADSTAR has moved to http://www.softdisk.com/comp/loadstar/

o Marc-Jano Knopp's CBM WWW Site is at:
   http://www.student.informatik.th-darmstadt.de/~supermjk/c64.html

o The US Commodore WWW Links Site has moved to:
   http://www.msen.com/~brain/cbmlinks/

==

## @(#)trivia: Commodore Trivia
by Jim Brain (brain@mail.msen.com)

@(A): Introduction

I had the good fortune of receiving some fine back issue of magazine and
old books from a friend in Michigan (thanks Gaelyne), so I got busy reading
and gleaning. The result is a new crop of trivia questions guaranteed to
rack your brain and have you reachin' for those numerous Commodore
publications. Go ahead, I won't mind.

As some may know, these questions are part of a contest held each month on
the Internet, in which the winner receives a donated prize. I encourage
those who can received the newest editions of trivia to enter the contest.

This article contains the questions and answers for trivia editions #23-26,
with questions for the current contest, #27.

If you wish, you can subscribe to the trivia mailing list and receive the
newest editions of the trivia via Internet email. To add your name to the
list, please mail a message:

```
To: brain@mail.msen.com
Subject: MAILSERV
Body:
subscribe trivia Firstname Lastname
help
quit
```

@(#): Trivia Questions

A publication describing BASIC on the Commodore makes the claim that
BASIC variables are limited to 5 characters, with the first two being
significant. The example to prove this point in the book is given as:

```
ABCDE=5   works, while
ABCDEF=6  does not.
```

The following questions refer to this claim:

Q $160) What is wrong with the above statement?

A $160) Variables can indeed be longer than 5 characters.

Q $161) What causes the variable ABCDEF to fail?

A $161) The variable name fails becase the BASIC keyword "DEF" in it.

Q $162) How long can variable names really be?

Extra Credit: Who was the book publisher?

A $162) As long as the maximum command line length. Theoretically, using
automated code generation, you can get a variable name that is
just shy of 255 characters in length.

Oh, and Abacus wrote the offending book.

The Commodore LCD Computer system, much like the Commodore 65,
was a product that never reached the market. Do you remember this
pint-size CBM machine?

Q $163) How many keys were on the CLCD keyboard?

A $163) 72 keys, including 8 function keys and 4 separate cursor keys.

Q $164) What does LCD in the Commodore LCD stand for?

A $164) Liquid Crystal Display.

Q $165) Was an internal modem to be includes?

A $165) Yep, A 300 bps auto dial/auto answer modem.

Q $166) Like the Plus/4 the CLCD unit had integrated software. What programs
were included?

A $166) As referenced in $158, there are 8 integrated programs:

Word Processor
File Manager
Spreadsheet
Address Book
Scheduler
Calculator
Memo Pad
Telecommunications Package

Q $167) How many batteries of what type did the CLCD use for power?

A $167) 4 AA alkaline batteries.

Q $168) Approximately how much did the CLCD unit weigh?

A $168) 5 pounds.

Q $169) What version of BASIC was to be included with the CLCD computer?

A $169) 3.6. It contained all of Basic 3.5 plus a few extras.

Q $16A) The CLCD unit contained a port that could be used with a Hewlett-Packard device. What did the device do?

A $16A) An HP bar code reader.

Q $16B) What microprocessor did the CLCD unit utilize?

A $16B) The 65C102 CPU. This CPU was built using the 65C02 core from Western Design Center, who licenses the popular 65C816S CPU as well. CBM licensed this chip at little or no cost as a result of a lawsuit settlement between WDC and CBM over 6502 architecture patent infringements.

Q $16C) In addition to the usual inclusion of standard Commodore ports, what two industry standard ports were included on the CLCD?

A $16C) Centronics Parallel (printer) port, and an EIA-232 (RS-232C) port.

Q $16D) How much RAM did the CLCD computer include?

A $16D) 32kB of battery backed RAM.

Q $16E) How many pixels are on the LCD screen on the CLCD machine?

A $16E) 480 x 128 or 61440 pixels

Q $16F) How much ROM did the CLCD computer contain?

A $16F) 96kB of ROM, which held the OS and the integrated programs.

Q $170) What text is displayed on the screen of a Commodore 128 upon bootup?

A $170) The following text is centered on either the 40 or 80 column screen:

```
COMMODORE BASIC V7.0 122365 BYTES FREE
  (C)1985 COMMODORE ELECTRONICS, LTD.
        (C)1977 MICROSOFT CORP.
          ALL RIGHTS RESERVED
```

Q $171) How many bytes free does a Commodore 128 have on powerup?

A $171) As shown above in Q $170, 122365 bytes.

Q $172) On the Commodore B-128 series, the bell beeps at the right margin. What column is the default right margin on the B-128?

A $172) Column 70.

Q $173) When a Commodore C64 is hooked up to a 1541 and an MPS 801 printer, everything is powered up and connected correctly, and the floppy won't load. What is wrong?

A $173) The printer is offline. Put the printer on-line, and the floppy will operate correctly.

Q $174) How do you access the "hidden message" in the C128DCR?

A $174) One brute force way:

While in the machine language monitor, type:

```
  m f63f5 f640b
```

Q $175) Some of you may remember the Commodore Magic Voice cartridge. If so, how many words was in the base unit's vocabulary?

A $175) 235

Q $176) Who write the 3+1 software bundled with the Commodore Plus/4 in ROM.

A $176) Tri Micro wrote the code, and created a version for the C64. It turns out that the 3+1 software included with the Commodore Plus/4 was originally designed to be but one of the many choices for bundled software with the 264. When the focus changed, 3+1 became the only software bundled, and some assumed Commodore had written it. (Ref. RUN April 1985:43)

Q $177) The BASIC extension "Simon's BASIC" was created by whom?

A $177) David Simons (Ref: Commodore Power/Play April/May 1985:56-7)

Q $178) Simons' BASIC was influenced a lot by what other computer
        manufacturer's BASIC?

A $178) Hewlett Packard. (Commodore Power/Play April/May 1985:56)

Q $179) How many commands does Simons' BASIC add to the Commodore 64?

A $179) 114. (P/P Apr/May 1985:57)

Q $17A) In the United Kingdom, there was an extension to Simons' BASIC
        developed by David. Among other things, what major complaint
        about the original BASIC extension does it address?

A $17A) Renumbering GOTOs and GOSUBs when renumbering a program.

Q $17B) In the Commodore Plus/4 File Manager, there exists two bugs,
        which show up if you have over a certain number of records. What
        is this magic number?

A $17B) When merging over 255 records in the Word Processor, a printout might
        stop early int the file and continually reprint a single record, or
        entering one record might trash another record. (RUN April 1985:43)

Q $17C) Commodore Semiconductor Group (CSG) manufactured an 8500 IC.
        What common IC number is this IC functionally equivalent to?

A $17C) The 6502. The change in number owes more to a change in
        manufacturing process than anything else.

Q $17D) How many BASIC commands were included in BASIC 3.5, not
        including the monitor commands?

A $17D) 80. (RUN November 1984:37)

Q $17E) On the Commodore VIC-20, 64, and C16 keyboards, what row and
        column pins on the keyboard connector does the letter D
        correspond to?

A $17E) Row 2 Column 2. (RUN July 1984:109)

Q $17F) What is special about the keys in Row 4 of the hardware keyboard
        matrix?

A $17F) Column 2-4 spell out CBM. (RUN July 84:109)

Q $180) Most people know what CPU is in a Commodore disk drive, but what
        CPU powers the venerable CBM 1525 printer?

A $180) You had better sit down.... The 1525 is powered by an Intel 8039
        8-bit microcontroller. Actually, this isn't so hard to believe,
        since Commodore didn't actually develop the printer, but used a
        Seikosha GP-100 printer mechanism for the unit, and most likely
        contracted Seikosha to develop the firmware.

Q $181) What is the maximum number of characters per line on a CBM 1520?

A $181) 80. 22 columns per inch times 3.63... inches of usable paper width.

Q $182) Commodore rarely manufactured its own printer mechanisms. Who's
        mechanism did Commodore use in the DPS 1101?

A $182) The Juki 6100 printer mechanism.

Q $183) What is unique about the DPS 1101 printer?

A $183) It is daisy-wheel, but Commodore made other daisy-wheel printers. what
        makes it unique is that it is the only such serial daisy-wheel made
        for the Commodore line.

Q $184) Which was the first Commodore modem with DTMF dialling capabilities?

A $184) The first to offer some kind of DTMF support was the Commodore 1660
        modem. The modem itself didn't provide any DTMF support, but included
        a cable to allow the SID to output to the phone line. Thus, with the
        SID's ability to reproduce DTMF tones, the modem could tone dial.
        Note that this was only possible on the C64, which has a SID. The
        first mode to INCORPORATE DTMF into the modem itself was the 1670.

Q $185) Which was the last Commodore 8-bit peripheral drive developed?

A $185) By develop, we are referring to actually produced models. With that definition, the 1581 holds this title. For models not actually produced, The prototype 1590-D-1 3.5" 1.44 MB model owned by Jack Vander White probably was the last under development.

Q $186) What is the maximum size of RAM available for use for program storage on an expanded VIC-20

A $186) If you discount the screen area (512 bytes) and Color RAM (512 bytes), up to 28159 bytes can used for BASIC programs and variables (original 3583 bytes and 3 banks of 8192 bytes each), and up to 40448 bytes can be used for ML programs. (0-32767 minus 512 bytes for screen and 40960-49151).

Q $187) One of the most poular magazines for computers in the 1980's was COMPUTE! What Commodore content magazine did it give birth to?

A $187) COMPUTE!'s Gazette.

Q $188) In a strange twist of irony, COMPUTE! was itself descended from a Commodore content magazine. Which one?

A $188) The PET Gazette. The PET Gazette was started in April 1978 by Len Lindsey. For the first year, the magazine was sent out for free to at times 4000 people. In August of 1979, Small Systems Services, headed by Robert Lock, purchased the magazine from Len and changed the name to COMPUTE. The focus changed from PETs to all computer systems at that time. The first issue of COMPUTE. appeared in the Fall of 1979. It seems the relationship between Len Lindsay and Robert Lock was less than ideal, but I refer readers to INFO #15, page 8 for the scoop.

Q $189) COMPUTE! underwent a name change very shortly after introduction. What subtle change was made to the name?

A $189) COMPUTE. changed to COMPUTE! Notice the change?

Q $18A) How were LOADSTAR and Commodore Microcomputing-Power/Play once connected?

A $18A) In the mid 1980's, LOADSTAR distributed the type in programs for both magazines in the disk magazine.

Q $18B) What is the fastest Commodore ever clocked a 6502 or derivative CPU in a machine?

A $18B) The CSG65CE02 CPU, clocked at up to 3.54 MHz in the Commodore 65 (64DX) prototype.

Q $18C) Name one byte that yields the same character when printed and poked to a Commodore screen.

A $18C) Any byte between 32 and 63 will produce identical results.

Q $18D) Quick, which chr$ value flips to uppercase/lowercase mode?

A $18D) chr$(14)

Q $18E) Quicker, which chr$ value flips it back to uppercase/graphics?

A $18E) chr$(142)

Q $18F) How do you get INPUT to not display a question mark?

A $18F) open 1,0:input#0,a$

Q $190) In reference to Commodore, what does TOI stand for?

A $190) The Other Intellect. Evidently, it was the computer the CBM engineers were working on before the VIC-20 project. The name sounds like it was dreamed up after the fact. In either case, this machine might have been the "Color PET" mention in _The Home Computer Wars_ that Chuck Peddle was designing before company shifted to the VIC architecture.

Q $191) Name two values that, when poked to the screen, will yield the identical character appearance.

A $191) 32 and 96 or 160 and 224. Space and reverse space.
103 and 106 or 101 and 116. Left and right lines.

Q $192) What chr$ codes lock out and re enable the shift/commodore keyboard
flip from uppercase to lowercase on the VIC-20?

A $192) chr$(8) and chr$(9), respectively.

Q $193) What chr$ codes lock out and re enable the shift/commodore keyboard
flip from uppercase to lowercase on the C64?

A $193) chr$(8) and chr$(9), respectively.

Q $194) What chr$ codes lock out and re enable the shift/commodore keyboard
flip from uppercase to lowercase on the C128?

A $194) chr$(11) and chr$(12), respectively, while in 128 mode.

Q $195) On CBM machines prior to the VIC-20, what chr$ code outputs the
same character as chr$(44), the comma.

A $195) 108.

Q $196) Is the character described in $195 of any use?

A $196) To put commas in strings read via INPUT. Remember, INPUT treats
a comma (chr$(44)) as a delimiter between input fields, but chr$(108)
does not produce the same effect, so you could replace 44 with 108 in
data written to disk, and read it in with INPUT.

Q $197) The speed of Commmodore BASIC increased dramatically after the first
OS upgrade in 1979. Why?

A $197) Jim Butterfield supplies us the answer:

"The original PET 2001 suffered from the same kind of "screen
sparkle" that was later seen in the early Commodore 64. So
the original code would write to screen memory only during
the "refresh" period; that really slowed down the speed of
output to the screen. By the time the first revised PET came
out, the screen sparkle was solved, and characters were
delivered to the screen with no wait. (The new operating
system also did a massive relocation of system variables,
and used zero page very heavily, to the dismay of home
programmers. When asked about this, Commodore pointed
proudly at the "new, higher speed". But in fact it was
the screen reorganization that caused 95% of the
improvement)."
--Jim

Related to this question is $00C, which implies that the
"sparkle" problem was fixed in the original PETs, so some people
increased the performance of the original PET by setting the RETRACE
line mentioned above to an output, which fooled the system into
thinking the video was ALWAYS in RETRACE mode.

Q $198) COMAL, a programming language available for Commodore computers, was
created by whom?

A $198) Borge Christensen and Benedict Lofstedt, although Borge is given
the most credit.

Q $199) At the 1980 COMDEX, Commodore PETs proved instrumental during a
crisis. What happened?

A $199) The following is excerpted from _The Whole PET Catalog_, page 21:

"PET PROVEN USEFUL" During the 1980 MGM Grand fire in Las
Vegas, Commodore moved its entire COMDEX '80 booth dowstairs
to help track rooms, guests, etc. According to _InfoWorld_,
7 PETs with OZZ data-bases (predecessor to SILICON OFFICE)
were used for two straight days. Local police agreed they
could not have kept of the guests as well as the PETs did.
Also, untrained operators quickly learned the system. In the
crisis, PET was both powerful and useable.

Q $19A) Who designed the PET/CBM 8032 computer?

A $19A) Bill Seiler, the able assistant to Chuck Peddle, designed the unit.

Q $19B) What was the "cursor gone out to lunch" bug in the first PETs?

A $19B) No answer available yet (I can't find my notes!)

Q $19C) On a PET/CBM (early models), what will "POKE 14,1" do?

A $19C) If done immediately prior to an INPUT, the poke will suppress the
question mark prompt.

Q $19D) What version of BASIC would not utilize disk drives?

A $19D) BASIC 1.0

Q $19E) Who is Lyman Duggan and why is he important?

A $19E) He is one of the founding fathers of the Toronto PET User's Group
(TPUG), along with Jim Butterfield.

Q $19F) Jim Butterfield notes to me that he received plenty of help in
creating the first PET memory map (Q $0D8) from the Sphinx group,
who published critical information in their early newsletters. How
did Commodore influence the name of the group?

A $19F) The name "Sphinx" was chosen because of the way early PETs resembled
the Great Sphinx, the Lion with the head of a pharoah.

Q $1A0) Commodore produced an assembler for the 128 called HCD65. What
does HCD stand for?

Q $1A1) Who wrote most of RAM DOS?

Q $1A2) What is the name of the first C64 disk copy program? (hint: it
sported a "gas gauge".)

Q $1A3) What was the case color of the original Commodore 64s?

Q $1A4) There are at least two ways to enter 64 mode from 128 mode on a C128:
go 64 and sys 65357. They produce the same result (64 mode), but
they differ in at least one noticable way. How?

Q $1A5) What CPU powers the B-128 computer system?

Q $1A6) What type of drive mechanisms are in the D series hard drives from
Commodore?

Q $1A7) Commodore produced a 16kB RAM expander for the Commodore VIC-20.
What is its model number?

Q $1A8) Commodore produced at least one disk drive with an optical track
one sensor. Which drive?

Q $1A9) The Commodore PET series used the IEEE bus to communicate with
peripherals. Each peripheral had a unique ID. What range of IDs
are supported by the PET?

Q $1AA) Many people have developed Commodore software with the PAL assembler.
What does PAL stand for?

Q $1AB) Many people remember Compute's Gazette. This magazine is best known
for the word processor program it shared with thousands of
subscribers. Name the program?

Q $1AC) In some 6502 assemblers, the opcode "bge" is available. It stands
for "branch if greater than or equal to". What more common opcode
is this opcode referring to?

Q $1AD) If I wanted to do a "blt" (branch if result less than), what 6502
opcode would i use?

Q $1AE) Each Commodore peripheral has a device number, which is associated
with a type of device. 8-15 implied disk drive, 4-5 implies
printer. These have remained constant from the PET to the C128.
However, one peripheral in the PET was phased out and its device
number was reused. What device number was reused?

Q $1AF) What is the maximum amount of general purpose RAM can one utilize
in a stock C64? (I need an exact number here)

==

@(#)gfx: Talking to TED: The MOS 7360/8360 Text Display ICs
by Levente Harsfalvi (TLC@MSZI.PMMF.HU)

@(A): Introduction

This information file is based on my old books, descriptions, and especially my experiences while I was coding. That's no mistake. The Plus/4 series was not very famous in the world, but they were very poular in mideast Europe. In fact, there were even demo groups for the machine. I learned some of this information while writing demos for the machine in demo groups, while other things were gleaned from personal work on the machine. These computers did indeed play an important part in Commodore computer history.

I started my first code development on a Plus/4 in late 1986. After I saw a HomeLab 3 (made in Hungary, U880 - GDR made Z80 compatible proc, B/W, 16K), I started writing demos and other software for the Plus/4 machine I owned. It actually wasn't that strange to see demo groups sprout up for all kinds of machines, including the Plus/4. All over, there were groups and individuals, writing software while trying to keep the flame lit for each machine. In fact, I know people currently working in groups writing for the Plus/4 in Hungary, Germany, and as far away as Alaska.

@(A): Overview

Let's discuss the TExt Editor (TED) IC and its environment. This DIL-48 IC was designed specifically for the 264 series of machines, which initially included the CV364 and the 264, evolving into the Plus/4, C16, and C116 machines. Unlike the CIA or ACIA or other machines, this IC isn't well suited to any other system.

The TED contains all functions done by several chips in former Commodore computers. The TED is a complete video-interface and composite video signal generator, sound generator, keyboard input latch, timer, clock generator, memory manager and DRAM refresher in a single IC. It can address the full memory map of the 264 series machines, and it generates the RAS', CAS', and MUX signals for the DRAM memory used in that series. For ROM, it generates the chip select (CS) lines, depending on the state of the internal registers. So, in addition to all the above duties, the TED IC is a simplistic MMU as well.

@(A): Video Information

We see the TED chip shine as it does its primary job, displaying graphics. Its abilities mostly parallel those of the uniquitous VIC-II video IC in the C64. It has the following modes:

* 40x25 screen (characters)
* enhanced color mode
* multicolor mode
* 320x200 Hi-Res Graphics
* 160x200 Multicolor Graphics

Of course, there are differences. TED does not contain sprite support.

To offset this omission, the TED chip can select 8 intensities for each of the 16 supported colors, giving 121 colors (the 8 shades of black are all black). Other features include a hardware cursor, hardware text blinking, and hardware inverse character support. Character sets, screen and color memory, and graphics bitplanes can be addressed directly, without additional logic as found on the C64. In fact, even RAM/ROM selection requires change of a single bit.

Character modes need $800 bytes of RAM for screen and color memory. The first $400 bytes act as color memory (the memory permanently located at $d800 on the C64), with the lower 4 bits containing color codes, exactly as found on the 64. Bits 4-6 denote the intensity level of the color, while the high bit select flashing/no-flashing attributes. The other $400 bytes contain the screen codes for the displayed characters. If hardware character inversion is selected, the lower 7 bits hold the screen code and the high bit selects inversion for the character. If character inversion is not selected, all 8 bits denote the screen code. Extended Color Mode (ECM) and Multi Color Mode (MCM) modes work exactly as described on the 64. While these two modes are in effect, inversion and blinking are disabled.

Things get a bit more complex in graphics mode (pun unintentional). In graphcis mode, the bitplane occupies $2000 bytes and is handled just like a VIC-II biplane. The colors are handled differently. $800 bytes are needed for color memory, which is laid out in $400 bytes of intensity memory and $400 bytes of color memory. An "off" bit in the bitplane uses the lowest nybble of the appropriate color memory location as the color and retreieves the intensity from bits 4-6 of the appropriate intensity memory location. For an "on" bit, the color is taken from the high nybble of the appropriate color memory location, while the intensity is taken from bits

0-2 of the intensity memory location. Bits 3 and 7 in intensity memory are unused.

In multicolor mode, differences abound. The 64's VIC-II enabled one to utilize 3 different colors in each 8x8 cell and a single background. The TED simply cannot accomplish this due to the lack of adequate color memory. So, TED allows only 2 varying colors per 8x8 cell. Those colors are chosen from the palette of 121. The remaining 2 colors are chosen for the entire screen, again from the 121 color palette. The mapping is as follows:

```
   00   background color
   01   same as "off" color in hires mode
   10   same as "on" color in hires mode
   11   another "background" color
```

The TED IC is able to generate both PAL and NTSC compatible signals from a single IC. Only the crystal need be changed to go from one standard to the other. In PAL mode, there are 312 lines hown, while NTSC only has 262 lines of display. The line synchronization is the same in either PAL or NTSC mode. It's always 57 clock cycles per rasterline. The TED divides the supplied crystal frequency by 20 for PAL display and by 16 for NTSC.

For the serious video programmer, raster interrupts are implemented as on the VIC-II. However, the 0 line of the register corresponds to the first line of the character screen area, not the top of the border. In addition, the current raster line can be read from TED registers. you can modify the counter as well. Doing so will most likely affect the screen display. As a bonus, the horizontal location of the raster can be read and modified in the same way. Unfortunately, these registers provide the basis for most effects, as the TED can't handle sprites.

@(A): Running The Show

As earlier mentioned, the TED IC does more than produce graphics. One of its tasks involves generating the clock signal for the 7501/8501 microprocessor. The clock is not constant, as it switches from from 885 kHz and twice that speed, 1.773 Mhz. The speed depends on TED's current task. It generates the slower clock signal when refreshing DRAM or fetching data for the video screen. Otherwise, the high clock signal is generated. The user can disable fast clock generation via a register. The end result is a machine that operates at approximately 1 MHz, as the CPU runs in slow mode while the screen is displayed, and operates in fast mode when the TED starts drawing the top and bottom borders.

@(A): Sound Advice

As far as a sound device is concerned, the TED doesn't stack up to the SID in the 64. Just 2 squarewave generators, of which the second can be switched to generate white-noise, are available for sound generation. Volume control is available in 8 levels.

To play samples, the TED can switch the sound generators to constant level outputs. D/A is then done by changing the volume register setting. Each generator can generate frequencies from 100Hz to 23kHz.

@(A): Other features

The timers available in the TED appear to be nothing more than 16 bit decrementing timers. They are always clocked with the slow clock. The first timer reloads its starting value when it reaches 0, the other 2 are free-running.

Since it already does almost everything else, it's not unusual to notice the TED handles the keyboard matrix. A simple 8-bit imput latch handles keyboard interfacing.

As noted above, a single bit in the register space will page ROM or RAM into the upper 32kB of the address map. Since the TED knows what is paged in at all times, it knows what to output to access the memory locations in this area.

@(A): Conclusion

Well, that about wraps up the TED IC. All that is left is a map of the registers. Assume all registers are read/write unless noted otherwise. If you have questions, I cna be reached at the Internet address listed above or at:

Levente Harsfalvi
7200 Dombovar

```
Gorkij 33.
Hungary

By the way, catch FLI ED. V1.0; Its info file may contain some more about
TED's screen-handling. It may be retrieved as
ftp://ftp.funet.fi/pub/cbm/plus4/tlc/cns.lzh

@(A): Register Map

Register      Description
--------      -----------
$ff00- $ff01: Counter #01. It always starts to decrement from the last
              written value into it.
$ff02- $ff03: Counter #02. It runs freely from $ffff.
$ff04- $ff05: Counter #03. Same as above.
$ff06       : Mostly the same as VIC's $d011.
              Bit 0,1,2 : Vertical smooth-scrolling
              Bit 3     : 24/25 rows screen
              Bit 4     : Blank screen
              Bit 5     : Bitplane mode
              Bit 6     : Enhanced color mode
              Bit 7     : TED's internal test, it should be 0.
$ff07       : Most similar VIC-reg is $d016.
              Bit 0,1,2 : Horizontal smooth-scrolling
              Bit 3     : 40/38 columns screen
              Bit 4     : Multicolor mode
              Bit 5     : TED stop. If set, the TED stops it's counters and
                          screen-generating, only single clock and refresh
                          cycles remain.
              Bit 6     : PAL/NTSC. 0:PAL, 1:NTSC
              Bit 7     : Disable reverse mode. If 0, we got 128 characters
                          and higmost bit tells if the character should
                          appear in inverse. If set, no inverse mode but
                          256 characters.
$ff08       : Keyboard input latch. Giving a strobe - writing to the register,
              the latch stores the values of the input-lines. Then, we
              can read them from this register.
$ff09       : Interrupt request register. When a counter sends want to send
              an IRQ, it's bit will appear as a 0; then, if the IRQ was
              caused then highmost bit is set.
              Bit 0     : Unused
              Bit 1     : Raster-counter
              Bit 2     : Lightpen. Not implemented.
              Bit 3     : Counter #1
              Bit 4     : Counter #2
              Bit 5     : Unused
              Bit 6     : Counter #3
              Bit 7     : Interrupt occured. This bit is set when an IRQ
                          was enabled and therefore, the IRQ was sent to the
                          processor. Physically, this is the negated level of
                          the TED's IRQ output. The IRQ should be deleted
                          with writing the register-value back after
                          accepting an interrupt.
$ff0a       : Interrupt mask register. These bits could be used to disable and
              enable interrupt-sources. When a place is set to 1, that will
              be able to cause an interrupt to the processor. If not, the sign
              of the interrupt request will only be appear in the above
              register.
              Bit 0     : 9th bit of $ff0b (see there)
              Bit 1     : Raster-counter
              Bit 2     : Lightpen. Not implemented.
              Bit 3     : Counter #1
              Bit 4     : Counter #2
              Bit 5     : Unused
              Bit 6     : Counter #3
              Bit 7     : Unused
$ff0b       : Raster interrupt register. Same as $d012 when writing; it stores
              the position of occuring raster interrupt. Higmost bit is in
              $ff0a's 0. bit.
$ff0c,$ff0d : Hardware-cursor position (10 bits). Lower bits: $ff0d, higher
              2 bits in $ff0c's 0. and 1. places. Beyond 1000 the cursor is
              not seeable.
$ff0e       : This reg is the first sound-source's frq-value's lowmost 8 bit.
              More 2 bits are in $ff10's 0. and 1. places.
$ff0f       : 2nd. source, lowmost 8 bits. More 2 bits in $ff12, 0. and 1.
              places.
              The soundregister-value can be calculated as
                reg=1024-(111860.781/frq[Hz]) (NTSC)
                reg=1024-(111840.45 /frq[Hz]) (PAL)
$ff10       : 1st. sound-source, higmost 2 bits. 2-7 bits are unused.
$ff11       : Sound control register.
```

```
                Bit 0-3   : Volume. Maximum value is 8.
                Bit 4     : Sound #1 on/off.
                Bit 5     : Sound #2 squarewave on/off.
                Bit 6     : Sound #2 noise on/off. If You set both, the square
                            will sound.
                Bit 7     : D/A mode. See above for more.
$ff12         : Bit 0,1   : 2nd sound-source, highmost bits.
                Bit 2     : Character generator in ROM or RAM. When set, TED
                            will enable ROM when trying to get data from the
                            charactergenerator to build screen. Else, it will
                            give out control-signals to the DRAM's.
                Bit 3,4,5 : These bits tell, where to find bitplane in the
                            memory when using bitplane-mode. TED assumes them
                            as A15,A14 and A13 bits. So, the bitplanes can be
                            switched as 8K pages, anywhere in the 64K.
                Bit 6-7   : Unused.
$ff13           Bit 0     : A sign to having control about memory paging. This
                            bit always sets to 1 when ROM is active over $8000.
                            Else, it will be 0. READ ONLY.
                Bit 1     : Force single clock mode. Then, TED will disable to
                            generate twiee clock.
                Bit 2-7   : Charactergenerator. Bit 7 corresponds to A15, 6 to
                            A14 and so on. This value shows and sets the start
                            of the charactergenerator. It can be paged as $400
                            bytes. Use with addition of $ff12-2.bit.
$ff14           Bit 0-2   : Unused
                Bit 3-7   : Start of the video-ram. Bit 7 also corresponds to
                            the A15 line as above. So, video-ram is mappable
                            as $800 bytes - 2K. The above $ff12-2.bit doesn't
                            affect this, but the actual RAM/ROM mapping (see at
                            $ff3e/$ff3f and $ff13/0) does.
$ff15         : Background. Lower bits contain color-code, higher 3 luminance
                and higmost is ignored.
$ff16         : Color-reg 1
$ff17         : Color-reg 2
$ff18         : Color reg 3. This and the above are used in ECM and MCM modes.
$ff19         : Border. All color registers use codes as described in $ff15.
$ff1a         : Bit 0-1   : Higmost bits of the next $ff1b
                Bit 2-7   : Unused
$ff1b         : Actual character-position. Higmost bits in the above register.
                TED counts the characters that it had fetched and put out to
                the screen. The number is increasing by 40 after every
                characterline (8 rasterline).
$ff1c         : Bit 0     : Higmost bit of $ff1d
                Bit 1-7   : Unused
$ff1d         : Actual position of vertical scanning. Higmost
                bit is in $ff1c. Read/Writeable!
$ff1e         : Actual position of horizontal scanning. R/W!. Lowmost bit is
                unused. It contains the TED's internal counter's highmost 8
                bits. So, it increases 4 with every character. When writing,
                it seems to put the value to a functionally different register
                (writing back a reading value in right time affects the screen).
$ff1f         : Bit 0,1,2 : Actual vertical scanning-line in a character-row.
                            R/W!.
                Bit 3-6   : Flashing counter. It's value increases with every
                            frame, and TED fits it's flashing feature to this
                            register's reaching to 15.
                Bit 7     : Unused
$ff3e         : Switching to ROM. A writing statement to this address will
                cause to turn on the ROM between $8000-$ffff. It's an other
                matter, which one; this time, only sure thing that it'll give
                CS signals instead of RAS', CAS' and MUX.
                See $ff13/0 and $ff14
$ff3f         : Switching to RAM. The opposite of the above.
```

==

@(#)error: ? DS, DS$: rem The Error Channel

We are not aware of any errors with issue 11, save the changes to some WWW addresses as noted in Hack Surfing (Reference: surf).

==

@(#)next: The Next Hack

"... and that's not all you get." Well, it is for this issue, but here's what Commodore Hacking is cooking in its TV informercial cookware for issue #13:

o CMD has announced that SuperCPU development units should be made available shortly, so C=Hacking will scrutinize it and detail the registers of interest as soon as it shows up.

o Exploiting the 65C816S. We're holding this article over to next issue to allow testing of the examples with the CMD SuperCPU. This article will detail the new opcodes available to programmers, show how to detect CPU clock speed on any C64, accelerated or not, and discuss pitfalls in code migration.

o Let's get HTMLized! It's about time the Commodore 8-bit learned to do HTML. There's nothing that says this popular WWW markup language can't do used to create nice disk magazines and newsletters on the CBM system, so C=Hacking begins a 4 part series on the language and how to render HTML pages on a Commodore machine.

o And, of course, C=Hacking's regular goodies.

So, go ahead, buy that box of disks, and label one now for Commodore Hacking Issue #13.

===

@(#)editor: Hacking the Code

For articles in Commodore Hacking that include binary files as part of their article, these binaries files are made available in this section as encoded text files. The format used for encoding is called UUCode, which is a standard widely used on the Internet to transmit binary files using only printable ASCII characters. To that end, each of these files must be decoded with a suitable decoding program before they can executed. Typical examples inlucde UUXFER for the 64, uudecode on the ACE OS for the 64 and 128, and uudecode on most UNIX OS machines. Some encoders can decode multiple files, while others will require the user to manually split this section into individual pieces prior to decoding.

WARNING: The UUCode format trasnlates files from binary to ASCII, not PETSCII. Therefore, either decode this section before downloading this section to a PETSCII mode computer system, or download this section without translation to PETSCII. Some decoder programs can handle PETSCII converted UUCode files, but the practice is not recommended because conversion is typically done in a telecommunications program and cannot be guaranteed to be accurate.

@(A)polycode: Binary for Polygonamy

```
begin 600 polygonamy
M`0@0",0'GC(P-C8@5C4N,````'BB_YJ]-PB=^`#*T/>.$=#N,-"&`:G-H&F%
M+80NA:Z$KZ`Y3``!`*@%D3_,:;TL"9WH!^C0]^X"`>X%`8C0[J("()T!\"S)
M!I`2*0&H()H!:0:0"*H@G0&%^1#CA8NE_#CEBX7\A8RE_>D`A?V%C2"&`:;Y
M\`3&^1#%()T!\`D@F@&B`H:+D!OH()T!\`GH()T!()H!:0%I`X6+Z""=`<@@
MF@%E_H6,I8UE_X6-.""&`?",I(NE_N6+A?ZP`L;_L8R1_HC0^6"^Z@&I`(6-
MI/OP#`;Z*B:-QOO*$/*H8(6.L?R%^JD(A?NECJ3\T`+&_<;\P.?0W*3]P`?0
MUJDWA0'.,-"I&XT1T%BI`(T`""!@IB".IDRNIP,'"PL("@">,C`V,>-_>*E_
MS+&I-(4RC0R@(KFJ:9G=_XC`_]#USB#JE0CL'0@XC>8!HO.:R&"""B$!F+X+
M*=@8@$I*4MK(T.KN/PL.0@CN2<B`.0C./`@0V:(.XP<"U)T#<"&=!.`&")T%
MU*F`G0;4BCCI!ZH0XJ`/J0`*D!>B_XX.U(X/U(T3;GSPCA34HH&.$J!XCXP8
MU*`"J?^@.-R9`=V(T/?`$+`,N3#@>0#<N4`;W;D`#9D`T,C`+]#E(%`-K3#`
M>L.M,0.%R*D@H@:-8)F-_M_@#-ZFH`!,C@![69I,!,Y)FHV#"$@@+)!H(,J.
M)("I"B\UR@(T`U0#=`.4I,5U*PC$'X88AB"Q"L8<A0"E$)$*B!#SQA[*E7?G
M3`X"?P@(@0$(4[(M_Y5"_P1J8`1X)0`DR@J0&2(#,,84!$BMW:/F8:3.4V)@
M%*J`F!#P+,KM\`6D!(6$H`O)__#C4`+T\!0&WY`$)M_&ET;_D.$@2(V1W,C0
M^.;=T.AR[*T2T,F`T/DR#``$W`P`K0C=)N_JZL8!(8"I_XU@`*(*3!$`J7N-
M$=#F`820A(0``H0"`!#%62@0,$.H`F#___S]""8!`P$`,S@Y,3%JW#`P,'X`
MQ@8`+P,`BP%`"@!$````O;U*`.J9/_]]F0X"B!#ZC!G0ZH8!:S,`:$`(",``P
M`(!&X>FG>::<X\FZ)TX,YT(&L:@R`%,!T`H<.(8@)&P#&3J,#+!28*)"D!-J
MO1!,!`(K/*@8>J1P%!=<`"$Q>`EA"Q#Q`HH0`$`,`$P`$(+Q[NY&`$``X>$H
M`"4`"$%H$1$CBQX>C0"0X"`$!9LWUJ`5R``5</`L"`[^]BD`"!*3_)2%F(B3
M*P=L/1*,QP`_!@`PGQ&L``'\0$(-Y$/NBF%^E,T0>D30P'N$%,`XAI357'I$V
M,![=XQ%C0*4WC01,.(PH@HT%<@BQI3F-!MRE.HQB@HT'F@)"W*U`"2T@`]VE
M,,(T-W&`%GC,R-S;/`#@`+C%*+D^B.+W)!94"Y9V1`LH0\^8!J9`L$=`P^\T2
MT-#VJ0@2X@Q`W7'=H#U(:*4`J0&B"(T.W(X/W*D`HG=,X`$%./'Y,`B!"`4"
M;P44/A:`#KX$"B=+$`=(ZP"JL9&S(J$X`"`/B@1Q@!F!>@IVHP4&`$"CL[V2
M-87@&0`!`S"2H`[%H#BHD)FD!2$FC__P!04`"J/F>M`"YGNM!@+).K`*!18C
M3F/W@Z1DZAJGY*>&KMBL3$BR`#'J`[!'_DKSD?(.\E#R,_-7\<KQ[?8^\2_S
M9OZE].WUR2#P[T6-!S`XZ=!@@$_'4EA`+/-D57\"!``*@`!O`!&!)H`""0!8
MIG+V/`,D(PUO8&`(GX,0:*`!%A'$"`L(P=$)()3`!P``)Q@-P'``'A$41A0('
MA@(&A\#;@>LLIVN`3#)7KR`0#PP9!P\.`0T9`D<3M`OL(%0%$`@%#B`,+B`*
M%00$`04TG3(8(#(O-B\Y-@+`(3,#%!,@-CHX(#HI`A<T"P'H(`$8OO`,RU<%
M!@<("0H+#`T.#U"91A.2&Q=)<B0;T9$@J%`06"C@D2&82(SAA8+@(P$+%55E
MA8JP(U^)C@0L9I2H4!"4;@@I<Y2@4'=R0GM\?7YA"8&"@X2%AH>(B8J+C(V.
MCY"1DI.4E9:7F)F:FYR=GI^@H:*CI*6FIZBIJJNLK:ZOL+&RL[2UMK>XN;J[
```

```
MO+V^O\#!PL/$Q<;'R,G*R\S-SL_0T=+3U-76U]C9VMO<W=[?X.'BX^3EYN?H
MZ>KK[.WN[_#Q\O/T]?;W^/GZ^_S]Y>X1!P;Y=Q!TEA44.DL5RIT-%87.DA4%
M!P7((B`>'!D7%1,0#@P*"`8#`?_]^_GW]?,]XMH1Z.;DXN#>W!36CM&A;$?'
MQ</!P+Z\NKBW$+D[KZZLJJBGI:.BH)Z=8NZ.E).1D(Z,BXF(AH6#@J`S=Q1W
M=G1S<7!OA;UC9F5D8F%@7EU<6XB\'4M.34P+CT`ZOO`=1R"/8#P"\@B\+1"W
M(2!H,+KC"?86B,<1P*,)YA&,QQ'*HP@?``80*((!"`<*B$@+`$@(`4!P`D@%
M`T"B6T@2'DTHCR/L6C`>2S"/%A8CC<<?'Y/8(R)ACR5D/8(J*[0>04B/(WQ'
M%]XC3$U.2X];7%U>8&%B9&5F);)[#(&B<W1V=T+W*(*#A8:(B8N,CI"1DY0Z
MQT>=GJ"BHZ6GJ*JLKJ^0>3ZWN+J\OL#!P\7'0IZ/T:#(Q]S>X.+DYNA0F#Z"
M_7W^?O]_P$#!@@*#`X0$Q47&1P>("!>$`$"\)`"`O$``@^;`/[\^OCV]/+P
M[NSJZ>?EX^'?W=O:V-;4TM#/S<O)R,;$PL&_O;NZN+:UL[&PKJRKJ:>FI*.A
MGYZ<FYF8EI63DI"/C8R*B8>&A(.!@']]?'IY>'9U=')Q<&YM;&II:&=E9&-B
M8%]>75M:65A75E134E%03TY-2TI)2$=&141#0D%`/SX]/#LZ.3@W-K[PP30S
M,C$P+\&Q+2PK*EDH.O0E)!15(HX_!0$5$%1P4(%"!7'A4,%9C$5$!6&B@O`4
M%A6$(*`G)FH))<``'<"$$BF@`#@L0@4P,`$P(`4P&`N.``PD$3`0%!@8&&%ZG
MH)R";PH*BJ@I#`P-#0X.#P\0$!$1$A(3$Q04%146%Q<8&!D:&AL<'!T>'A\@
M("$B(R,D)28F)R@I*2HK+"TN+B\P,3H[/#T^/T!!0D-$149'(ITC@@0D-$149'
M2$E*2TU.3U!14E-45E=865I;75Y?8&)C9&3_F&EJZ&-P<7)T=79X>7I\?7^`
M@8.$AH>)BHR-CY"2DY66F)F;G)Z?H:.DIJ>IJZRNL+&SM;:XNKN]O\'"Q,;(
MR<O-S]#2U-;8VMO=W^'CY>?IZNSN\/+T]OCZ_/ZM@%,F^<V@=$@<\<6:;T09
M[L2:<$8<\LF@=TXE_=6LA%TU#N:_F7)+)?_9LXUH0AWXU*^+9D(?^]>TD6Y+
M*`;DPJ!^7#L:^=BWEW96-A;WU[B9>EL]'@#BQ*:);$\R%?C<OZ.';%`U&O_D
MR:^4>F!&+1/ZX<BPEW]F3C<?!_#9PJN5?FA2/"81_.;1O:B4?VM70S`<"?;C
MT;ZLFHAV9%-!,!\/_N[>SKZ2KIZ/@'%B5$4W*1L-`/+EV,N_LJ::CH)V:V!5
M2C\T*B`6#`+Y\.;=U<S$N[.KI)R5CH>`>7-M9F!;55!*14$\-S,O*R<C(!T:
M%Q01#PT+"0<&!`/_ID?_5298F>`]`0(#!`8'"0L-#Q%2QAH=(",G*R\S-SQ!
M14I055M@9FUS>8"'CI6<I*NSN\3,U=WF\/D"#!8@*C0_2E5@:W:"CIJFLK_+
M14I055M@9FUS>8"'CI6<I*NSN\3,U=WF\/D"#!8@*C0_2E5@:W:"CIJFLK_+
MV.7R``T;*3=%5&)Q@(^>KK[.WN[^#Q\P05-D=HB:K+[1X_8)'#!#5VM_E*B]
MT>;\$28\4FA^E:O"V?`''S=.9G^7L,CA^A,M1F!ZE*_)Y/\:-5!LAZ._W/@5
M,D]LB:;$X@`>/5MZF;C7]Q8V5G:7M]CY&CM<?J#"Y`8H2VZ1M-?['T)FBZ_4
M^!U":(VSV?\E2W*9O^8.-5V$K-7])4YWH,GR'$9PFL3N&41OFL7Q'$ATH,WY
M)E.`GWZB!!;Z`^ZZ$>XA1D0P@"NJ`*<@)`$$&X`F5@!2H`/'@%@)T.8!D`@!J`
M`S5$%S8`%Q8!%Q(%"%9`"A7@(%)@(%"%PH&%PH$@(D%*%PD*D@M@"`P`7"`TR
M`#@D#F(`$!Q0(&`@*A)@&)A(%P%%A@%&FQ(%9@$0`%\9PG%LA*A(`!H%.!4
M&1(%)%PH3&&@@8`W&P@-(`?@.8B`"T(!@`L_J%P6%`PA.#(!LK@*^8!%(%J-
M$A!T3@O$&(!K/)%`(@#'0%P$!D8%(&HIA(BJ+!4<`:4%<0*?H#"6)!!P.$P
M-$!M#%!#@$9'(`@O@`@"0$!$@-/P)6"0D;&DPDB/4!(K%&PB@PDB$`(0X@2!
M$WK/GG($4!%I$5T`R#(`"$5'RD,T%-3XC$0L9)!`<(G)JDJA@<8@4%IL%#"$
M"C$*0A)Y4PX%<T$3&P'3.R$0A"GEL/2)FC:@#*;PB#'1!K-`D9%`<07`JPB%
M```
```

```
M4H\B!$A,4,`$6$Q@P`1H3'#`!'A,@,`$B$R0P`283*#`!*A,L,`$N$S`P`3(
M3-#`!-A,X,`$Z$SPP`3X,)18!`A,$,`$&$P@P`0H3##`!#A,0,`$2$Q0P`18
M3&#`!&A,<,`@>#&4>A0AB&`"D":88`*@)JA@`K`FN&`"P";(8`+0)MA@`N`F
MZ&`"\";X,:#$(@A@`A`F&&`"("8H8`(P)CA@`D`F2&`"4"988`)@)FA@`G`F
M>&`"@":(8`*0)IA@`J`FJ&`"L`:X,J'4HP@!R!/0,`'8$^`P`>@3\#`!^#(E
M%@$($Q`P`1@3(#`!*!,P,`$X$T`P`4@34#`!6!-@,`%H$W`P`7@3@#`!B!.0
M,`&8$Z`P`:@3L#`!N!/`,`'($]`P`=@3X#`!Z!/P,U!B,Z,>10@(F(`0"1B8
M@"`)*)B`,`DXF(!`"4B8@%`)6)B`8`EHF(!P"7B8@(`)B)B`D`F8F("@":B8
M@+`)N)B`P`G(F(#0"=B8@.`)Z)B`\`GX-"BQ"`B8@!`)&)B`(`DHF(`P03@U
M*/4H0DC`!%!,6,`$8$QHP`1P3'C`!(!,B,`$D$R8P`2@3*C`!+!,N,`$P$S(
M`303-C`!.!,Z,`$\$SX-4")10C`!!!,&,`$($PHP`0P3#C`!$!,2,`$4$Q8
MP`1@3&C`!'`,>#9"J4<1`H@FD&`"F":@8`*H)K!@`K@FP&`"R";08`+8)N!@
M`N@F\&`"^#9*+`(()A!@`A@F(&`"*"8P8`(X)D!@`D@F4&`"6"9@8`)H)G!@
M`G@F@&`"B":08`*8)J!@`J@FL&`0N#=*/8H0R#`!T!/8,`'@$^@P`?`3^#=0
M8A$(,`$0$Q@P`2`3*#`!,!,X,`%`$T@P`5`36#`!8!-H,`%P$W@P`8`3B#`!
MD!.8,`&@$Z@P`;`3N#`!P!/(,`'0$]@P`>`3Z#`!\#@E.#;J482`"`D0F(`8
M"2"8@"@),)B`.`E(!("5"8@%@)8)B`:`EPF(!X"8"8@(@)D)B`F`F@F("H
M";"8@+@)P)B`R`G0F(#8">"8@.@)\)B`^#D2BX`("1"8@!@)()B`*`DP&(0X
M.KH30$I(P`103%C`!&!,:,`$<$QXP`2`3(C`!)!,F,`$H$RHP`2P3+C`!,!,
MR,`$T$S8P`3@3.C`!/!,^#I`B44(P`003!C`!"!",*,`$,$PXP`1`3$C`!%!,
M6,`$8$QHP`1P#'@[0MT)@"6(8`*0)IA@`J`FJ&`"L":X8`+`)LA@`M`FV&`"
MX";H8`+P)O@[H,0B"&`"$"888`(@)BA@`C`F.&`"0"9(8`)0)EA@`F`F:&`"
M<"9X8`*`)HA@`I`FF&`"H":H8`*P!K@\H>Z$P!+(,`'0$]@P`>`3Z#`!\!/X
M/%!B$0@P`1`3&#`!(!,H,`$P$S@P`4`32#`!4!-@,`%@$V@P`7`3>#`!@!.(
M,`&0$Y@P`:`3J#`!L!.X,`'`$\@P`=`3V#`!X!/H,`'P/24]-GA`\(J`"`D0
MF(`8"2"8@"@),)B`.`E(F(!("5"8@%@)8)B`:`EPF(!X"8"8@(@)D)B`F`F@
MF("H";"8@+@)P)B`R`G0F(#8">"8@.@)\)B`^#Z9``P(F(`0"1B8@"`)*)B`
M,#^9.#\!B.&S(/\>+R#`R@"`8@"`"`!20,!`"<#E"&"K`F$F8F`"8R9E8`)F
M)F=@`F@F:F`":R9L8`)M)F]@`G`F<6`">B9[8`)\
M)GY@`G\F@&`"@2:#8`*$)H4!"(8@`1`-`$"&`2T`(`$`"``!``0"]`:D/X+&-
M(=#('B,5A?Q3C_U,38&3#AP1`0Z05T],64=S'DU9#04!$F`<#1\($F&9IT`E
M-"(4R(W$`$1)3;J)4TE:#B:!A"H*8W-OQE!&CX)LLFH\!T!!!5U32E5$1$!.
M5U4N1415SR.0`08@<T5%PX19`YYC/6A!0TLRK$Z0($99($16K4%)3%/F!Y4!
M!R!UFAD).I%`$1&=+Y$[QS01`H!624=!)6*0#9P!!2`^/CX@<"3;Q5\@T$M%
MD%@H3P$%14<YDB`\/#P-#9IAJ2XN+G-/4E+AE4%"3U5<`N(G0E5'4R$-=&$-
M&@(!!=Y-DZQ3(%"<`T].04Q)5%D@14Y(04Y#15)3`+'\\"X@TO\F#O8W<#$%
M!4]W%4$@4T5#4D4(#U1%6%0@34534T%'1303(%E/5R$@Y/_)`/#YI0$I_H4!
MJ0AT$T<8C1C0K<<.Q`$"@A`%*?P)`[DPK<D)((T1T,#:_(7^80\>:&!,!@"-
M05DP4*D$A?VI7('7@ZD6LB2,YOWFEBORB8'`2-#WJ1V1_)'^R-#YHL"&6NB&
M6*(*AFBB#(9JH@Z&;*(0AFZB$H9PD,@=*W*AEB`P`F$2"%*[DNA)M(T$%`(!
MR$R-`L`-@XTAW(*-(L`-1(U!W+R-0A9`=HT@%HTERP5G\3DV1'@@G_^ER\E`
M\"_)+LT*0MY)("#)-]`+H@/*_@`6RA#Z,!')+6(W1Z.%3)N"R3+"UP:&$0'F
M..8YI3H$%P.%.J6QK"6$%^QI?8:0C!Z\"%^*4YE"*L#%UL`#L%`4$H
MO!0$A3PI"(T_^0!"/<$,`#J%/FTB"CN%0*4\=@!"I?]HA3@#0#C)-`5$ICND
M//6112EY3$9AG<L*(,E'ID)@QD)(4,*?$-Z`22%X"`%L@E@D]!A(A0GH`L(0
M%.!`Y@<+?D200$FF0Z1$:HD1$$J0'B/(3:9`OQ,\`%=`6&$"",L2`0Q%"SZD
M/=)DRCCYL(202Z8_I$)!#:#0ID"D0;VS`8#*&!`!.&HXY?^F/:0^Q'-HRR07
M`,NU*4RB`-EA!`_)AI`KV>@`)AAY0!:0(,(*A?^$^8I(\`ZT!KD`%L7_D`58
M%T3RI/F4!VBJZ(@0R\HP)(:MM0?)`%?W`6B)9``!T`8@3(M,=83)`M`#(-"/
MIJW*$-REO]`_-/$;&*3L`!B%QD`I+EH!X<>B&,%PC%FB(X036Z+(O7`P'-[_
M6/`7AJ!XBDI*2AAIIX5<BBD'J.(9(7*FIO_*T-R"I22_J<:B>(X8T(T`W:D9
MA;Q,8X(B5^Z)<(B.(`DL*$5612*>`!<`4S`$1$0IQAT)@0?$0`4P`:$(24Q(
M&_$05$L@2;<0)%*#1S$F(4$R%1%2,2U,DT]25$%,LA=#$R!.3U0@2E535$E&
M6=+L2$4@5T%94R!/1B!G3T0@5$\@34%.(B`M+2"0X@!%+D-534U)3D=3YLM6
M4C`4PA<0"(U0&Q``E!=LA/`%!!KYHY``-$:(XL:LH`.(Z@5``@20,`!,_[F*
M./VB3`+&_[<<B$`"!)@0%L(`-@@')@J`"87YN4@8,0D8\_D8OHTU?0`7D`+F
M!`]0S/F90!9`AID`%H@0HV`.KP#(Q<8+,$#8B"ZTK+X`R25&",06>*+_BD@6
MT3CE0C6W"*;_DD[</%NX8M9(4B-B29%!2K'(H$&E2]AJ1$PL,BA-EAD3PZ5%
MI#M>&F#!`D:D/#/P0H5'I#V%:[%K2$0L0JAHI)9KP4F`@A04$*@^R0JP7/^J
MO0`5JJ0>5@HJ%@,/H$*D0'90,$BIH+:"_Y`#YD,5"X9":&4S0TKP!LVP6(B&
MJ&K)H)`(W#Y"J8D:0Z1!#B0D:29#:2D@9^,!9Q0/::"`1*0_BH5.2F^821QI
M`44IL6](L6VD_SCQ;87_:/%OI?]<=T$0"UP!#:D`\`D8::C)R)`"J<>%09$:
M/((893G6'4?NZ8XC6(`8M*,)(D@P0S%H1Q.T@H'&H!U-$$$"2%"0&)JF0"DY
M>Y'D1,"2$P`)3@0@!2>"0()G7?B3X#&2#)\S8WCG(VQ'BR$S:@QW<P2P6#%X
MQ(RA=HY@$T-XE1'#5M08!*A@I:Q(..4Z*`P7P7`$3>7!Q9`63;P9,8)-?*&-
MKB""Q90)$T-:-$1%'C^1.T?&N>!&5<BM+[#)B.&V1@PU2G"-&B7DU8AA+6H,
MW78$T`K"G'T068QB:(6L:%4``XWA7YI@%7A$D$`Q_$L33(!@%1`LAG]I@@D0
MK$)3BV`!8OB7)H@@J&7"Q)`R31"(U2E8"HT!)#,2(909"4'*C`2OAREBIR<B
M$$$H$8@FH!19C$$T:@RG5(0<(;S$B$&5QE!"PX27&#&HTB]8"(DF<C@IAE49
MI?@3DZ88?&;$L%<U!H<Y@@X0$&K$<$,UACHK0HX0@&C$$$PUAL,IIBG28Z48
M@++&$$%%8!D5@@.,X;ZJCS+F8KBOO*H\N!CVJPE00+;)<3'L5Z)3OFP,>K8$
MF@I02N;!Q=!<540Y?4%L:\AQ<G2"V-:0(=B"P(-B33E.04Y/D!(AX,!06$Y1
MD!-2Q-R6`!(4)%.6$U3D!%4!ISL$R3RIC#(0.AJQT%`;0D,<B4"]J3B:E)@D
M`@4E%"@CH$`!(V5>7@PGT0F<1"=P$IU`F@S!D@DC<2`4[4@Q),F0-L4+-RA>
M0#79,C.*%S1OO(`RX@4R%*=H,>A$L"8NB9H$+$5-@K`8OP"CJ4`E`414400B
M52@"$5.4TXHQG,0.<!([P$GL`"<A#V+0`0+MC!A.8@<XB1W@)':`DY`',>@`
M(44'(B@Z0"#1`2!*2/P8_HU3`HMB>(\DX#V2@/=(`DIC"'I8N+="OP0A*G:
MJ-@!HF('B+S'#O`>.\!![`!T,P;`%B1@RN-&.I*)A0E3L`%&Q
M`T3%#A`5.D`VT0$WB@XU4'2`-0)#()4<IDTA)0&WD03<1A)P&TG`;2`!&P4)
M'"A(0!I!`AI*_50PQ9-.*/<13Y3["$T*?@0I`-,^)88C*3&8QXB!.222Y0A$
MRG($!`4YC@2E5(TE.E6-17`/4XT%1U,Q$D`%I5&-)AU2C4;&I5"-!L;D4%C^
MC`:@V$8'HC5#&)F-_S`#3'6B*D@13(``B6(0D"H(A#B+04"J8`($2!2#@%1N
M)<?%("!5T`.#RGH,&4R`@0(?.882DJ&,Q(LRY00S.31!RH6&<GJ.3)LU6,XV
MD7,W)#6^*N(%*Q0O*:!X`2FE2R<H7D`H\0(FBA<F4+B().(%)10O(Z!X`2$T
MAI"EI;$DT"@MBE4$+L4J@BPR%4<VL8HWH%A%-5"L(C4H53$S%*L(-(I5!#+%
M*H(R4A5#P(LCY"Z"0W(700.YBR`$10,2%`1)8APA1`EF*""6$R'D!"(I$5]*
MHXLCY00D.25!2H19DN-B)CDG04XHD"6`I4R8*5A.*I`3*Z1$IR@G?LB$B2SE
M!"TY+JEBA3NIH84\J7R%/:T"07H^K2+@!C^M0A:%0"!IAJ5!A2^E0H4PI4.%
M,:4A1AP)(BA(0"`I,B<*$BA0D(`F@@0FE",DH2`!)04)(RA(0"/YD4(@<5*E
M*HV$A:4KC5ZE*8WI!46JI3"-(?X.,8U!G"^-`<--+[`"7*4MC2)W#BZ-0L:E
M+(T"QN0L+.NJH`*$^ZD#A:.%_R!VF6"D^UU;!310A?ZH2DI*&*:_T`YI0/0\
M&C`*0!MAT`QII\B!6H;!O@!;AL*%7(5FA6"I:;);F"D'A?I@".V:LL"Y+9R;
M0&+2D!9#T!?LG6#:)74!WP@/\F<8D2^\FQ,.HV%6@C]`*Q*"#I@H=A`(Y:FJ
MF.D`D/KPND.@8-`4C)>*\)2->D":HOJO+O2&1BRU6B"GJBKEJ)`6J(HGN)"0
M`<@XY;<"YH@P!B"LG$SQI&`8IJ=[YHHE9$6FM98&(X5EJ&JV$03&J&`*M(UF
M65*J"-W1;I4YH2+T6<)A*NX.1Y*%G\"0.^R<DXB,#UE2J@COD6QPCK2D-,<^
MHI+*:-$@,$/H#8$9P@(1>+817F!>@F4R+T'8!;;*MD09IB+K<"19^`D,N4.?
M,]'(C[G%#_V9DKF@!/T,GQ#^`SONYY]-#Y)H@)%HPWD,$L4\BK!`%!0'D`_H
M.H8`%MF0`QCFJ)$^QXP!/VX-4S*$G\B0.YJAQJ70>Z3\$_<%I*.(X,E9Z84)
MI83_?*0<!9Y2+VAXA.G2A^H4"#"#Z2ADZ9*'*96]HBDT@>DQ:0T/ZK$#ZN&
MJ<:ST'VD_9BJR,2CT`*@ZX/_$`H)_I@)!@#&\.>%LH6SA/V0)0"2<+1`'/D0
MAK`#3)"=2KT@XH$X0,:%MD^(:JJDLLAUPCCY`,.P!J(`.'Z0%*HXX,2JA5=)
M_VD`A5FEM'%9./%7ZX*0!^CELL6RL/F%N)A*Y]V&MZ6K?`N&<*2=!B"XQH$2
M6JNEI^6IA:<1Y@2HBJCKI!K%7_`0A5^!<8U`J#^\7LBQN]`F;R8300&X`!"$
M3[)?,(="`/$@4JA5(-BE94(]7[`;:0/0%*2GN<D<I+4Y)Q(+$4&P(*1!^K%Q
M(%6FIJ<@<<S"(%V17::UO0#-,7$18Y'@S+DXY;BP$V6R5-P`&!0#D`Z%N:6U
MY;>%M:JP)G.V&$Z1\`*4?0#%Q#4#Q67P'"V%PJP(@29,`HHI^!AEP24X"+9E
MPA.I`:3^D;N8\`G&_@+&^C`&JO9,2*;!<(C&8,9FQERI8,0RH`>$^J5=P0E`
M7:5>4UZE8\`2@&.E9*9DI<$XZ4"%P:7"Z0&%PDSQHZ1EJ9F16V!L6P"*!V4`
M;%\``JB]%+='$`@P`1`3&#`!(!,H,`$P$S@P`4`32#`!4!-8,`%@$V@P`7`3
M>#`!@!,00R$6`9@3H#`!J!.P,`&X$\`P`<@3T#`!V!/@,`'H$_`P`?A@)18!
M"!,0,`$8$R`P`2@3,#`(.&&E'D4(2)B`4`E8F(!@"6B8@'`)>)B`@`F(F("0
M"9B8@*`)J)B`L`FXF(#`"<B8@-`)V)B`X`GHF(#P"?AA*+$(")B`$`D8F(`@
M"2B8@#`).)B`0`E(F(!0"5B8@&`):)B`<$%X8BCU*$*(P`203)C`!*!,J,`$
ML$RXP`3`3,C`!-!,V,`$X$SHP`3P3/AB0(E%",`$$$P8P`0@3"C`!#!,.,`$
M0$Q(P`103%C`!&!,:,`$<$QXP`2`3(C`!)!,F,`$H$RHP`2P#+AC0JE'$0+(
M)M!@`M@FX&`"Z";P8`+X8THL`@@F$&`"&"8@8`(H)C!@`C@F0&`"2"908`)8
M)F!@`F@F<&`">":`8`*()I!@`I@FH&`"J":P8`*X)L!@`L@FT&`"V";@8`+H
M)O!DH,1D1CV*$`@P`1`3&#`!(!,H,`$P$S@P`4`32#`!4!-8,`%@$V@P`7`3
M>#`!@!.(,`&0$Y@P`:`3J#`!L!.X,`'`$\@P`=`3V#`!X!/H,`'P$_AE4&(1
M"#`!$!,8,`$@$R@P`3"#.&90ZE&$@$@)4)B`6`E@F(!H"7"8@'@)@)B`B`F0
MF("8":"8@*@)L)B`N`G`F(#("="8@-@)X)B`Z`GPF(#X9A*+@`@)`$D@
MF(`H"3"8@#@)0)B`2`E0F(!8"6"8@&@)<!B$>&=2CR($B$R0P`283*#`!*A,
ML`$N$S`P`3(3-#`!-A,X,`$Z$SY18!`A,$,`$J$RPP`3X9Y18!`A,$,`$@$S0P`283*#`!*A,
M0,`$2$Q0P`183&#`!&A,<,`$>$R`P`2(3)#`!)A,H,`$J$RPP""X:)1Z%"'(
M8`+0)MA@`N`FZ&`"\";X:*#$(@A@`A`F&&`"("8H8`(P)CA@`D`F2&`"4"98
M8`)@)FA@`G`F>&`"@":(8`*0)IA@`J`FJ&`"L":X8`+`)LA@`M`FV&`"X";H
M8`+P:4II;-2C"`$($Q`P`1@3(#`!*!,P,`$X$T@34#`!6!-@,`%H$W`P`7`3
M`7@3@#`!B!.0,`&8$Z`P`:@3L#`!N!/`,`'($]`P`=@3X#`!Z!/P,`'X:B46
M`0@3$#`!&!,@,`$H$S`P"#AKI1Y%"$B8@%`)6!B8@(`)B)B`\`DP;E8@
M```
```

```
M4&`"6"9@8`)H)G!@`G@F@&`"B":08`*8)J!@`J@FL&`"N";`8`+()M!@`M@F
MX&`"Z";P?:#$?28/`?&*@`@)$)B`&`D@F(`H"3"8@#@)0)B`2`E0F(!8"6"8
M@&@)<)B`>`F`F("("9"8@)@)H)B`J`FPF("X"<"8@,@)T)B`V`G@F(#H"?"8
M@/A^F0`,")B`$`D8F(`@"2B8@#!_F3A_`8AAGX`!@@2&"8P0E!F>)*HQN$#(
M4=ID[GD$D!RI-L12X7``D"&R1-9I_)`DN4[D>A&H0-AQ"J0^V700K$GFA"+!
M8`"@0>*$)LEL$+19_J1*\9A`Z)$ZY(XYY)`\Z99$\J%0`+!A$L1V*=R01/FN
M9!K1B$#XL6HDWIE4$,R)1@3"@4``P(%"!,:)3!#4F5XDZK%X0`C1FF0N^<20
M7"GVQ))A,`#0H7)$%NF\D&0Y#N2ZD6A`&/'*I'Y9-!#LR::$8D$@`.#!HH1F
M22P0]-F^I(IQ6$`H$?KDSKFDD'QI5D0R(1``\.'2Q+:IG)"$>6YD6E%(0#@Q
M*B0>&100#`D&!`+W_P`!`@0&"0P0%!D>)"HQ.$!(45ID;GF$D)RIML32X?``
M$"$R1%9I?)"DN<[D^A$H0%AQBJ2^V?00+$EFA*+!X``@06*$ILGL$#19?J3*
M\1A`:)&ZY`XY9)"\Z19$<J'0`#!ADL3V*5R0Q/DN9)K1"$!XL>HD7IG4$$R)
MQ@1"@<``0('"!$:)S!!4F=XD:K'X0(C1&F2N^420W"EVQ!)AL`!0H?)$END\
MD.0YCN0ZD>A`F/%*I/Y9M!!LR2:$XD&@`&#!(H3F2:P0=-D^I`IQV$"H$7KD
M3KDDD/QIUD2R(9``<.%2Q#:I')`$>>YDVE'(0+@QJB2>&900C`F&!((!@$?^
M`1/];R(W15!:86AN<WA\@(2(BXZN]@]9G9^AHZ6GJ:JLKJ^QLK2UMKC*>B*\
MOK_`0HW$Q<;'R,G*2)@DS<XBE`C2BDL$,!$$F@AN(MS<XHG?W^#@4-."!1!
MIN;GY^CHZ>GJZNOK[.SM[>WN[N_O\/#P\?'R\O+S\_3T]/6%X[)Y?]G9VMK:
MV]O;8H7<W=W=WM[>W]_?9(*PJB`$X@SCY.3D00;E3.;`".>,"!$,D"";0<#-
M(``080`("`$`#P`"(7IF.CHC`3H5`CH.`SH,!#H)!3K0FC("!SH&"#()8`@*
MY@LZ!0P-#0TZ!`X/#P\$!`1$1$2$A(3$Q04%!45%A87%Q@8&1D:(,CW*0CX
M5"`A(2(C(R0E)B8G*"DJ*RLL+2XO,#$R,S0U-C<X.3L\/3Y`U/%F"9AE3"C*
M45)45EA96UU?86-E9VEK;6]R='9Y>WZ`@X6(BXTOKB29G)^95P,D`083"3`!
M#!,/,`$2$Q4P`1@3&S`!'A,A,`$D$R<P`2H3+3`!,!,S,`$V$SDP`3P3/S`!
M0A-%,`%($TLP`4X343`!5!-7,`%:$UT!"&"0'5P`@@)J_`!.`$Y4=#450\6L
M='AA8"<^/CX@;75L=,5V<X-'3@5%A$!XH$A)1TB@0EE480Q&,RY9"")T99'H
M0D4(/41T-UIP;(/8<&]S>7)P3U-)(Q&`H$]2MJ`_E!]LY"(V:88D2)`P5$A%
M3I$G8605(S$V.),Y041$+*!!3D2@24:@<W1I;&R@3D5'051)5D4N+BXOCF-M
M<'`1,@,1!V)C8R`Z;VL-(&QD82`C,#`@.W-%5*!43Z!:10$M'A!5"/\`[N[N
M`+N[QP!`4%6[@0"`JA&J1"A@510`(`!5JA0`[MV[=Q`!@'>[W>X(`+M5[C2%
M?D$!`O_;3IF]P]3!K1@$YY#%(``,`(D@``S_D`$@&.<@!`!&&-O,``!^O5,"
M!O]^`0EKH@0$0``P]^OW[]?O"#8`,"("6/O5J]\`5RW?J]7]/`@`_]^OW_OU
M^_\!I'$ORZ&/U@%B'P+L!I[`J;Z]O+NZN+>UL[&OK:NIIJ2AGIN8EI.0C(F&
M@X!]>G=T<&UJ:&5B7UQ:5U5344]-2TE(1D5$0T(;\X$!!4!!04)#1$5&2$E+
M34]14U576EQ?$/^H:FUP='=Z?8"#AHF,D).6F)N>H:2FJ:NMK[&SM;>XNKMH
M?*B"H0H!`00:D.:!0,8@("`!"(!N``(#!08("0L,#@\0$A,4%1<8&1KNB!P=
M2H@!0"`?'Q\>'AT<&QL:&1@7%103$A`/#@P+"0@&!0,"`/[]^_KX]_7T\O'P
M[NWLZ^GHY^;GRB/DXRHA!@$'X.'AX>+BX^3EY>;GZ.WN\/'R]/7W^/K[
M_?Z0<0!\`P">`0#'`(`%`/]_/Q\/!P,!)#$`O@$`SP"`8P#``@"`P.#P^/P2
MV#8,F'^]U(`Z`X(&\`""#0/0`O?7V]O;W]_?X^/GY^?KZ^OO[^_S\_/W]_?[^
M_MC%1;2P`04-(W``"$30`"((YB$`B#``!(2`__#_`00`_P`"``!0!%2,N-S]&
M3%%66EYB96AL;G%T=GA[?7^!@X6&B(J+C8Z0D9.4E9>8F9J;G)Z?H*&BHZ2E
MIJ>GJ*FJJZRMK:ZOL+"QLK.SM+6UMK>WN+FYNKN[O+R]OKZ_O\#`P<'"P\/$
MQ,7%QL;'Q\?(R,G)RLK+R\S,S,W-SL[/S\_0T-'1T=+2T]/3U-34U=75UM;7
MU]?8V-C9V9#F`;``@(03))0PO3'JI$?^2O.1\@[R4/(S\U?QRO'M]C[Q+_-F
M_J7T[?4!L$6,`P"]`0$`O1H:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:
,&AH:&AH:&AH:&AH:
`

end
```

```
==========================================================================
@(#): bottom
```