```
        __  __             ______
       |  ||__|  ______   /  __  \   ______   __  __   ______    ____   __  __
   ____|  | __  /  __  \ |  |  |__|_/  __  \ |  ||  | /  __  \  /    \ |  ||  |
  /  __ /  ||  ||  |__|__||  | |____/  |  |  ||  ||  ||  |__|  ||  ||__||  ||  |
 |  |  ||  ||  `\____  \ |  | |____\  |  |  ||  ||  ||   _____||  |  W |  ||  |
 |  |__|  ||  ||  |__|  ||  |__|  ||  |__|  ||  ||  ||  |__|  ||  |  a |  ||  |
 `\_______||__|`\______/'`\______/'`\______/'`\____/'`\______/'| /'  D `\    /'
          _     _  _ __ ___ ____ ______________________________|/________|  |
           _     _  _ __ ___ ____ _________________________________________/'
```

The Journal of the Commodore Enthusiast

Issue 2 : October 1, 1996

PREAMBLE

We greet you to the second issue of disC=overy, the Journal of the Commodore Enthusiast. Our inspiration for launching this work derives from you, the ones who still hold our beloved 8-bit machines in high regard and respect. In honor of your committment to these classic platforms we have pledged ourselves to assemble this entire journal on modest C64 and C128 systems. It is our sincerest hope that you will find our efforts to be of interest and special joy. We thank you from the bottom of our hearts and look forward to forging a solid productive relationship with the C= 8-bit community.

- Mike Gordillo, Steven Judd, Ernest Stokes, and the authors of disC=overy.

ARTICLES OF OPERATION

Article 1 : Mission Statement

Our intent is to present useful information in order to enhance and preserve the knowledge base of the Commodore 8-bit domain, including, but not limited to, the Commodore 64 and Commodore 128 home computers. To this end, we shall require that every article contain what in our discretion should be a viable Commodore 8-bit hardware and/or software point of relevance. Likewise, each issue should include material that can both potentially enlighten the most saavy of users as well as the layman. We intend to complement and assist all others engaged in similar endeavours. We believe it is of paramount concern to stave off entropy as long as possible.

Article 2 : disC=overy Staff

The current staff of disC=overy, the Journal of the Commodore Enthusiast, is as follows:

Editor-in-Chief	: Mike Gordillo	(s0621126@dominic.barry.edu)
Associate/Tech Editor	: Steven Judd	(judd@nwu.edu)
Webmaster	: Ernest Stokes	(drray@eskimo.com)

disC=overy, issue 2 logo by 'WaD'

We invite any and all interested parties to join us as authors, panelists, and staff members.

Article 3 : General Procedures

- Submission Outline -

a. Articles may range in size from 1 kilobyte and up. Approximately 15 kilobytes of text is the preferred length, including any software present.

b. Sufficient technical content about Commodore 8-bit home computers, concerning software and/or hardware relevant to these systems, is a requirement. What constitutes a sufficient amount of 'technical content' is left to the discretion of the Editor-in-Chief and/or the review panel (see below).

- Staff Priorities -

The Editor-in-Chief shall supervise the organization of each issue in regards to grammatical and syntactical errors, flow of content, and overall layout of presentation. The Editor-in-Chief and Associate Editor shall form a review panel whose function it shall be to referee literary work which the Editor in-Chief has deemed to be of advanced technical merit. The Editor-in-Chief iand disC=overy, the Journal of the Commodore Enthusiast, shall retain copyright solely on the unique and particular presentation of its included body of literary work in its entirety. Authors shall retain all copyrights and responsibilities with regards to the content of their particular literary work. Authors shall be required to submit their works to the Editor-in-Chief approximately two weeks prior to publication.

Article 4 : Peer Review

To the best of our knowledge, disC=overy shall be the first Commodore 8-bit journal with a review panel dedicated to uphold the technical integrity and legitimacy of its content. The Editor-in-Chief and the Associate Editor shall be responsible for the formation of the panel. The appointed panelists shall have the option of anonymity if desired. The panel shall review works primarily for technical merit if the Editor-in-Chief and the Associate Editor deem it necessary. Authors may be asked to modify their works in accordance with the panel's recommendations. The Editor-in-Chief shall have final discretion regarding all such "refereed" articles.

Article 5 : Distribution

Although we welcome open distribution by non-commercial organizations, there are currently three "secure" distribution channels available to interested parties. This journal may be obtained by directly mailing the Editor-in-Chief or via the World Wide Web at http://www.eskimo.com/~drray/discovery.html and at FTP site : ftp.eskimo.com - directory /u/t/tpinfo/C64/Magazines/discovery Several versions of this journal may be available for your convenience, please check with the aforementioned sources.

Article 6 : Disclaimers

The Editor-in-Chief and disC=overy, the Journal of the Commodore Enthusiast, retain all copyrights regarding the presentation of its articles. Authors retain all copyrights on their specific articles in and of themselves, regarding the full legal responsibility concerning the originality of their works and its contents.

The Editor-in-Chief and disC=overy, the Journal of the Commodore Enthusiast, grants the reader an exclusive license to redistribute each issue in its entirety without modification or omission under the following additional stipulations:

- If distribution involves physical media and is part of a commercial, not-for-profit, or PD distribution, the maximum allowable monetary charge shall not exceed $4 1996 United States Dollars per issue unless more than one issue is distributed on a single media item (i.e., two or more issues on one disk), in which case maximum allowable charge shall not exceed $4 1996 United States Dollars per media item. All dollar values given assume shipping costs are -included- as part of the maximum allowable charge.

- If distribution involves non-physical media and is part of a commercial, not-for-profit, or PD distribution, the maximum allowable charge shall be limited to the actual cost of the distribution, whether said cost be in the form of telephony or other electronic means.

! - Software included within articles (as text) may be subject to separate distribution requirements as binary executables. Please check directly with authors regarding distribution of software in binary form.

It is understood that distribution denotes acceptance of the terms listed and that under no condition shall any particular party claim copyright or public domain status to disC=overy, the Journal of the Commodore Enthusiast, in its entirety.

The Editor-in-Chief and disC=overy, the Journal of the Commodore Enthusiast, reserve the right to modify any and all portions of the Preamble and the Articles of Operation.

```
::::::::::d:i:s:C=:o:v:e:r:y::::::::::::::::i:s:s:u:e::2::::::::::::::::::::
::::::::::::::::::::::T A B L E  O F  C O N T E N T S:::::::::::::::::::::::::
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

-Software Section-

-Hardware Section-

-Corrections-

```
::::::::::d:i:s:C=:o:v:e:r:y::::::::::::::::i:s:s:u:e::2::::::::::::::::::::
/S01::$d000::::::::::::::::::S O F T W A R E:::::::::::::::::::::::::::::::::
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

: Innovation in the 90s :

The Super Hi-Res Interlace Flexible Line Interpretation Technique

by Roland Toegel (Crossbow/Crest) and Count Zero/TRC*SCS

Prelude from Count Zero
--------+--+--+--------

First seen earlier this year, the new SHIFLI video technique that is described in this article is -the- paramount example of programming brilliance. The inventor of the technique, Roland Toegel (Crossbow of Crest), was helpful in providing the original SHIFLI documentation, which I translated from the German into the English text that follows for the exclusive use of the disC=overy journal.

So without further ado, let us now learn from Mr. Toegel how to achieve the award winning :

'Super Hires InterLace Flexible Line Interpretation' Graphics mode !

Please note that the technique is described primarily for Commodore computers based on the European/Australian PAL TV standard. NTSC-based Commodore machines require an extra 2 cycles per line. This may require the programmer to time out the routines by hand, but this should not be a major obstacle to overcome.

Count Zero

--

i.   Forward

As the inventor of the SHI-FLI mode, I am pleased to have the services of Count Zero and the disC=overy journal for the dissemination of my technique into the English language. I must add that the text below does require the reader to be already familiar to a high degree with VIC-II programming on the C64. I would suggest books such as 'Mapping the 64' and the programming texts found at ftp.funet.fi /pub/cbm ..etc., for a solid base of instruction. Also, some terms used in this document (e.g., mix-color) are meant to be uniquely descriptive and hence, will not be found in any 'standard' programming text. The terminology is a result of the strain that occurs when new methodology meets old semantics. However, the experienced programmer should find the words to be self-evident in the context which they are used.

--

1.   Introduction to Super Hires

Super-Hires Interlace FLI : The absolute successor of the Super-Hires-Modes. Just like normal Super-Hires, the width is 96 Hires Pixels, which is equal to 12 Characters or 4 Sprites next to each other. For visual enjoyment this area is centered though using Char-Position 15 to 26 (included) on the screen. The Y-Axis is 167 pixels high having nearly 21 Character lines or 8 Sprites and to have as much flexibility as possible on choosing colors or pixels, 2 sprites are overlayed, 4 times next to each other (8 sprites on the rasterline) over the bitmap graphics. Due to the fact that all 8 sprites are used for a 96 * 21 pixel-wide area, a small multiplexer is needed to juggle all 8 sprites for 8 times (every 21 rasterlines and with individual patterns). We thereby win 2 colors plus the 2 normal colors of the Hires-Bitmap Mode in an 8*8 pixel block. Please note that the 2 additional colors are the same throughout the whole picture.

2.   Super Hires with FLI !?!

FLI is for most coders still quite hard. Sprites over FLI for most quite impossible. Maybe one or two sprites, but 8 !?! next to each other and still FLI in each rasterline! Hard to believe, eh?

First of all you need to know how to do FLI and what it does and also what effect sprites have on it.

2.1   Normal FLI

On each eighth rasterline (the Badlines, the first rasterline of each charline) the VIC stops the processor for 40-43 cycles to read the new Characters and colors of the video and color-ram. This is the case when the bits 0-2 of the registers $D011 and $D012 are the same. Now if you change on each rasterline the bits 4-7 of $D018, which holds the length of the video-ram (handling the colors on bitmap graphics) and set the bits 0-2 of $D011 to get a badline on each rasterline, you will get new colors on each rasterline in the bitmap graphics. The only condition to be followed is that on each rasterline 23 cycles are used whereby for the *used cycles - 22* char of the textline the next 3 chars the byte $FF (light grey) is read from the video ram and the FLI effect starts after that (the FLI bug). In the multicolor mode for the color-ram the next byte in the program after writing to $D011 is chosen as the color. As we are using the Hires Mode here, this is irrelevant.

> NOTE: The last 3 sentences were pretty hard to translate and I advise you
>   to read other articles about FLI aswell, if you want to know more
>   about Multicolor FLI. (CZ)

2.2   Sprites over FLI

So what does a sprite do over FLI? Pretty simple, as it just eats up some cycles. All 8 sprites use 19 cycles per rasterline, meaning in case we code our FLI routine without loops, we just need 2 LDA, STA commands (for $D018 and $D011), using 12 cycles per rasterline. All together with the 8 used sprites that makes 31 cycles. Thus, the 'light grey' FLI-Bug occurs on char-positions 9,10 and 11 and from char 12+, the FLI effect comes up. This doesn't matter much to us, as the Super Hires Picture starts at Char-Pos 15. We therefore get 3 cycles per rasterline for other commands.

3.   Mulitplexing over FLI

Now we rather have to increase the Y-Coordinates of the sprites by 21 pixels each 21 rasterlines and give them new patterns. As we use 8 different video-rams on FLI for the colors and the sprite-pointers are always at the end of the video-ram, we are supposed to write 8 * 8 values for the patterns plus 8 values for the Y-Coordinates, resulting in 72 different addresses. Thats far too much for a single rasterline and adding the FLI routine will bust the limits. Therefore we have to do a little trick.

3.1   Changing the sprite-pointers

The trick is not to change anything at all! On the other hand we don't want the patterns to look the same everywhere. Luckily, the height of a sprite (21 pixels) is not capable of being divided by the height of a textline (8) and the smallest mutual multiple is 168 (meaning 21 * 8). As we are writing (due to the FLI) a new value to $D018 on each rasterline and we use 8 video rams, we can abuse this and have different sprite-pointers on every video-ram. The handling of where the graphics for the sprite-patterns are located becomes a little bit confusing, but it doesn't eat up any rastertime as we don't have to change the pointers.

3.1.1   Table to illustrate the sprite pattern-handling

Spriteline	Video-Ram	Screenline (NOT equal to rasterline!)
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	1	9
10	2	10
11	3	11
12	4	12
13	5	13
14	6	14
15	7	15
16	8	16
17	1	17
18	2	18
19	3	19
20	4	20
21	5	21
----------------	----------------	----
1	6	22
2	7	23
3	8	24
4	1	25
.	.	.
.	.	.
.	.	.
20	1	41
21	2	42
----------------	----------------	----
1	3	43
2	4	44
3	5	45
.	.	.
.	.	.
.	.	.

3.1.2   Example

Small example for Sprite 0 under the following conditions:

```
Used 8 Video-Rams              : $4000 - $5FFF
content of the sprite-pointer: $43F8  80 00 00 00 00 00 00 00
                               $47F8  81 00 00 00 00 00 00 00
                               $4BF8  82 00 00 00 00 00 00 00
```

```
$4FF8  83 00 00 00 00 00 00 00
$53F8  84 00 00 00 00 00 00 00
$57F8  85 00 00 00 00 00 00 00
$5BF8  86 00 00 00 00 00 00 00
$5FF8  87 00 00 00 00 00 00 00
```

Thus the Sprite-Patterns are in memory from $6000-$61FF.

Therefore the pattern-handling looks like this:

Screenline	Memorylocation
1	$6000-$6002
2	$6043-$6045
3	$6086-$6088
4	$60C9-$60CB
5	$610C-$610E
6	$614F-$6151
7	$6192-$6194
8	$61D5-$61D7
9	$6018-$601A
10	$605B-$605D
11	$609E-$60A0
12	$60E1-$60E3
13	$6124-$6126
14	$6167-$6169
15	$61AA-$61AC
16	$61ED-$61EF
17	$6030-$6032
18	$6073-$6075
19	$60B6-$60B8
20	$60F9-$60FB
21	$613C-$613E
22	$6180-$6182
23	$61C3-$61C5
.	.
.	.
.	.

3.1.3 Remark to the display-routine of the editor

As the changing of the video-ram on the editor-routine happens inside of the textscreen (but the spritepointers are read inside the sideborder), the change takes effect one rasterline later. This means that whenever the colors for the bitmap of color ram 2 are read, the sprite pointers or video ram 1 are still active. This is the reason why the editor uses only 167 screenlines (instead of 168) and why the first textline of the first video-ram and the first rasterline of the bitmap stays empty.

3.2 Changing the Sprite Y-Values

As the changing of the sprite pointers more or less happens by itself, we just have to make sure the correct Y-Value comes into the game. These are still 8 values, but they don't have to be set in one rasterline and we got 21 rasterlines to set them. As we have just 3 cycles left on each rasterline on the FLI routine described and that wouldn't be enough for a simple LDA : STA, we have to change the FLI routine a little bit.

3.2.1 Load new sprite Y-Value

As the height of all sprites is the same, we just have to do a single LDX #$VALUE. Therefore, 2 of the 3 free cycles are used and we cannot do anything else with the last free cycle.

```
LDX #$VALUE
LDA #$08
STA $D018
LDA #$38
STA $D011
```

3.2.2 Load next but one $D011 value

As a STA $SPRITE0Y needs 4 cycles, we cannot include it in the next rasterline, but we can already load the next $D011 value into the Y-Register. The free cycle stays unused.

```
LDY #$3A
```

```
LDA #$18
STA $D018
LDA #$39
STA $D011
```

3.2.3   Write new Sprite Y-Value

As we already did the loading of the $D011 value for the next line, we now have 5 cycles left and therefore enough time for a STA $SPRITE0Y.

```
STX $SPRITE0Y
LDA #$28
STA $D018
STY $D011
```

Now plot 3.2.2 and 3.2.3 have to be repeated for the remaining 7 sprites with changed values for $D011 and $D018. So there is now 1 rasterline for loading the new sprite Y-value and 8 * 2 rasterlines to write the new sprite Y-Value. We now have 17 rasterlines and the 4 remaining ones just need an additional NOP so that all rasterlines use the same amount of cycles.

3.2.4   Remark to the Display-routine of the editor

For simplification of the routine, which generates the FLI routine, in the remaining 4 rasterlines an LDX #$VALUE was used instead of an NOP.

4.   Memory-allocation

Now video-rams, the bitmap, and the sprites have to be placed reasonable in a VIC-Bank. As the banks from $0000 - $3FFF and $8000 - $BFFF are useless for graphics due to the overlay of the Char-rom we choose the back from $4000 - $7FFF for now.

4.1   Video-rams

The 8 video-rams need $2000 Bytes. They are located from $4000 - $5FFF.

4.2   Bitmap

The bitmap needs $1F40 Bytes. It's located at $6000 - $7F3F.

4.3   Sprites

As we need 2 sprites overlayed (four times next to each other and 8 times below each other), we need 2 * 4 * 8 sprites, meaning 64 overall. We need $1000 bytes for the sprites. We check what the video-rams and the bitmaps already allocate and recognize that only $7F40 - $7FFF, enough memory for 3 sprites, is left open. How do we rectify this situation?

As the video-rams and the bitmap just need a small part for displaying the picture, the sprites can be put into the spare parts of the video-rams and the bitmap. They have to be masked by choosing the right color in the video-rams.

A textline of a bitmap covers $140 bytes. Our Super Hires cutout just needs $60 bytes though and is centered. Thus the first and the last $70 bytes of a textline of the bitmap is free. As a sprite needs $40 bytes, we can put 2 sprites in each textline of the bitmap (one to the left and one to the right). Due to the height of the picture (21 textlines), this results in space for 42 sprites. From textline 22 on (in memory from $7A40) we can use the whole textline for sprites, resulting in 5 sprites per line. Continuing this until textline 24 (included), we have space for 15 additional sprites. So overall we already have 57 sprites and just 7 are missing now. These we could place in the remaining free area of the bitmap ($7E00-$7FFF), but that's not very efficient as we have some space left in the video-rams.

The Textline of a video-ram contains $28 bytes. The Super Hires cutout just needs the middle $0C bytes. As the sprites we put to the left and to the right of the picture are supposed to be invisible, we need to set a background-color in the video-ram (in our case, the color light-grey $FF). So we don't have enough spare room for the sprites to the left and the right of the picture in the video-ram.

If we finish the FLI Routine from textline 22 on and keep the video-ram on until the end of the screen (filling the this area ($4370-$43E8) with the backgroundcolor $FF to hide the sprites) we can use the remaining 7 video-rams from textline 22 (from $4770, $4B70, $4F70, $5370, $5770, $5B70, $5F70) for one sprite each. Now we have placed all 64 sprites and the allocation of the sprite pointers looks like this:

```
$43F8  80 84 85 89 8A 8E 8F 93
$47F8  94 98 99 9D 9E A2 A3 A7
$4BF8  A8 AC AD B1 B2 B6 B7 BB
$4FF8  BC C0 C1 C5 C6 CA CB CF
$53F8  D0 D4 D5 D9 DA DE DF E3
$57F8  E4 E8 E9 EA EB EC ED EE
$5BF8  EF F0 F1 F2 F3 F4 F5 F6
$5FF8  F7 1E 2E 3E 4E 5E 6E 7E
```

The pointers from $80 to $E4 are the 2 sprites which are left and right next to the picture in the bitmap.

The pointers from $E8 to $F7 are the sprites from textline 22 to 24 below the picture in the bitmap.

The pointers from $1E to $7E are the sprites from textline 22 to 24 below the picture in the video-rams.

5.   Interlace

Until now we had the normal Super Hires FLI mode, supplying the basics for interlace. For the interlace mode we need 2 pictures of this kind switching, displayed 25 times per second (PAL). As such a picture fits into the VIC-Bank from $4000 - $7FFF and we have another VIC-Bank ($C000-$FFFF) with the same assumptions we can easily place the 2nd picture there.

We had in the Super Hires FLI mode (on a 8 * 1 pixel-area) the choice between 4 colors (2 sprite-colors, being the same for the whole picture + 2 FLI colors). Now, in the interlace mode, we have the choice between 16 mix-colors, meaning the combined 4 colors from picture one and two. When using interlace, mix-colors are created except for the case when the same colors are used for both pictures on the same 8 * 1 pixel-area (Check the following example) :

	Pic2 ->	Sprite1:$E	Sprite2:$0	FLI1:$6	FLI2:$9
Pic1	Sprite1:$1	$1E	$10	$16	$19
[	Sprite2:$3	$3E	$30	$36	$39
V	FLI1 :$E	$EE	$E0	$E6	$E9
	FLI2 :$6	$6E	$60	$66	$69

This results in the following 16 mixcolors:

1. White-Lightblue
2. Cyan-Lightblue
3. Lightblue-Lightblue (pure Lightblue)
4. Blue-Lightblue
5. White-Black
6. Cyan-Black
7. Lightblue-Black
8. Blue-Black
9. White-Blue
10. Cyan-Blue
11. Lightblue-Blue
12. Blue-Blue (pure Blue)
13. White-Brown
14. Cyan-Brown
15. Lightblue-Brown
16. Blue-Brown

When choosing the colors you should take care that the brightness-values of the 2 mix-colors are about the same and that they do not differ by more than 2 brightness steps, as things otherwise start to flicker too much.
(e.g. Black-White flickers a lot).

Here is a table with brightness-values from light to dark.
(Colors on the same line have the same brightness)

$1	White
$7, $D	Yellow, Lightgreen
$3, $F	Cyan, Lightgrey
$5, $A	Green, Lightred
$C, $E	Grey, Lightblue

```
$4, $8 : Lilac (Purple), Orange
$2, $B : Red ,   Darkgrey
$6, $9 : Blue,   Brown
$0     : Black
```

6.   Additional Graphics (Not handled by the editor)

We found out that on the left or right of the picture in the bitmap, $70 bytes was left for spritedata. We used just $40 bytes of that space, meaning we still have $30 bytes (6 Chars or 48 Pixels) left to both sides of the picture.

6.1   Left to the picture

Our Super Hires Picture starts at position 15. The spare $30 bytes are from position 9 to 14. As we use 14 cycles in our FLI routine and the 8 sprites use 19 cycles per rasterline, the light-grey FLI Bug now uses the chars 11, 12, 14. Thus meaning we could use char 14 for Hires FLI. On chars 9 and 10 we could just use 2 different colors (respectively 4 mix-colors for interlace) on the height of 21 textlines in the bitmap, as the FLI effect starts from Char 14 and before that no new data (colors in this case) are read from the video-ram. The colors are in the first video-ram in memory from $4008+$4009 respectively $C008+$c009.

6.2   Right to the picture

Our Super Hires Picture lasts until char-position 26. The spare $30 bytes in the bitmap are from position 27 - 32. Here we could use all 6 chars for Hires FLI (or Interlace Hires FLI).

7.   Memory-allocation of a picture startable with RUN

The included SHIFLI picture, once unpacked, can be easily modified for your own use, as follows :

```
$0801-$080C Basic Startline
$080D-$0860 Routine for copying the Graphic-data to the correct memory area
$0861-$095B Routine which is setting the I/O registers and creates the
            display-routine (from $085F-$10FB)
$095C-$475B Data of the 1. Picture (to be copied to $4000)
$475C-$855B Data of the 2. Picture (to be copied to $C000)
```

8.   Conclusion

That's it ... for further information check the editor code and other sources, most likely available at ftp sites such as ftp.funet.fi /pub/cbm ..etc., etc. Included is the uuencoded version of the X96 Graphics C-64 Contest Winner by Deekay/Crest. It was NTSC/PAL-fixed by the people of the American MegaGroup, Style.

--
For questions or comments concerning this article :
Roland Toegel is available at   : toegelrd@trick.informatik.uni-stuttgart.de
Count Zero/TRC*SCS is available at : count0@mail.netwave.de

```
begin 644 x96-winner
M 0@+",8'GC(P-3D H@!XA@&]'0B=^P":Z-#V3  !UJY9W$&]W4&=\ >]_4&=
M$ CHT/&@ (3WA/@@@0$F]] "D HF^""! ;#P(*H!J0"%]H7YA?H@@0$J\ DP
M!2"! ;#SYO9I H7WH@,@@0$F]LK0^*;VO=X!JO *(($!)ODF^LK0]J7V*0?P
M"N;YT +F^L;V$-X8I?QE^87YI?UE^H7Z(+T!\(X&^] D2*  L?ZD_M "QO_&
M_L#OT ZD_\ 'T BI-X4!6$P-"#@JA?MH8#BE_N7WA?F%_J7_Z0"%^N7XA?^D
M]_ 2I?PXY?>%_+ "QOVQ^9'\B-#YQ/CP!L;ZQO@0[6 # 0$! 0 ! 00! 0$!
M @(!^<8\QC[L=?)@( <(>.X:T.X-W*DRC1)$M96MC10#HA!X"@.I<3[0G3;0
M..D8RLK@"-#QACRB7X8[Z([_#Z(4J=B ?B^P"8HI ?8I#6GCC5EM;_#0"JD!
M;?@I#XWQ ;W;N? 0D3O(R='0]I@893N0 N8\A3NMKMT6C>#<[! 8:0$I!PDX
MC>40K6%?:1 I?XW;X /G$(WT$,H0I*W6$6D5R=OO JDSC=D0R4C0D(X5T(X@
MT%CP_J($ZLK0_"!?"*U<72D"20*- -VJH .]Z* G)]"]Z4.9*]"($/'N&=!,
M@>JB2'@MH* ]9H2I/(U[C@_0J1B-&-",$00W-VKAG5VE=FCF=G1Y)"(+SZNK
MUL3K;<ID:YE4*/G!PO-.26V9.JNTB;#?LG*B] 7U>_W;",_7W"IOIY]>)ND-
M0Y2.%=Z'5QTJO-&WH:0AA&E&_8J%'<UG'NJGJ<);R^IWGOO]6(?I"-/.*M=
M:<E[$+;]MA6>''?5>)*H_V<+@$"<PMM#V%;AK:%6FL.[M8DQ_K6N\#S22BV+
M2BI,S9A\6X7WG>XS<3+5&.'IFZ \OU=X:]]D>A;O/&_RSLFT+7H9RG.KA7=;
M9N-=>&).MHE3K=6%D_& W[7BW7=X'<4[-=?*4B6<3,_AQJ[*R=2:PK0[>,'8
M^]S)ZG14X:T[ *#4Z98X> /@1U\9Q1 *Q@%7?ED:X:#LT?!3%IX4(!G$K%=X
M$&  6)NF+P PRZ7P_ERK#]S=1Q=F>_+\LDD3=W=ISRN\[;52 W>WK59ZK/#4
M8BO/VYT7=06*+V]WYPK/FZUL$-[O:;M3 ^&PU92>O#VB=L8G[^W;G89-38Q9
M+;PMFR*FILQL9;3PS;\\QU: VFO2/.%]KO#TS8+'%\BSU978^T2KLNE"*Y.%
```

```
MIP9:7;6[M6S2_=K"VSX%/#?TDZD@ L"@3J)KLU9^&)F$"B (]VD-=68;YP-D
M!@ CZC'H"PA\&!H,SK/"\U/P7N&]7;OS$[VT07E_HM4&3FH3[2R</%EX:J_5
M!EX,?-HL/#79RJ.$]VMHZS2)=IHMO(W;G6NS]PP7WI&#&2V\SR\NO-Q\LU:8
MYD_=;Y7#PX#YI;<Z=!F%IQ[) R)S8+(BSVVU%BGO?JM,JP^W<HN%]V'(*T/>
MN5UW"F\#"!7L$#@U&C7UH0&^ W4#V<\0SE5..SN$W1TYC ^[R%T!LZU7O@?M
MD$ZEA\CE9=3.$.RDF*7"^T.C=]SNME=X:[=5K?#>NMT=E?F3I[>S+M!JDX7G
MC5";*CR_;]+HY+GG3%;0SF]1GALHO/6^W55)FHYJX6F6Z=9>X;F54[7P-(-2
MP+I%]K1M=QXN/",QW^[,F]1HX7G>[BZW\C[3=0OO5BO=9N%]!^*D[5;V1C4+
M[\$<<RNE[$Y*(&!%-YW>0&!F;@TSJ<\53<V8 4 @HTM[_#. <XF^!5T6HYNA
M+4X<QCL;? (9L\+S@ZW69.&I-<KS,Y%HM6W3&V1EL94;.5EI%2EE3:I-NU/]
M'\TUZ]0:VH5-U@IO-6UW:K?P/F&\POL#=U<UK199P&VUW7WD9*;59E!9P-^=
MK]V]]<(7?#-WN]L$VGVX\#Y,><;?</+&9E:OY<+SP,8<6JW=PO.6TMDTO3I6
MD9X'K80$&'#5".>NO(!B+)%FQ&DB#AEV/C>K'("KQ"&(E:_'!@[,\V@ ^,[%
MQ#Q:"V]/1!Q4M?#V:-B%;]*R*,9SGC@5L\[K)F^(NG"R9DIJMSL-RIQUQ#-<
M>!QNE^F8W>TV*<^S)NT4WG_4="'O>H7W6[:[O6?2).39,+R3IC+)GDF3@AB>
M,;QKP[>[RB\Q/,/*EMONWGKAK7V[^\DVD\F7F0F^S>2!R)FV V%XVNRX?.&I
M<"667](Y;9&5R$)5F@$8( -0#!BQ1SU?51,62F 'E7NG"$3,<:TZ<TR6>C!8
MA/V@S&X#)"#E%S[VPM/;(I?GN0.&MR;#SB2D%Y[>M;N.X;).CXF0=KV2T;K-
MA=S6<"[D#AC>P;:Y7(C$0DX-W9V>#3G-9UVT\/XO.T-.S_7<ID-NTVCWI>8J
MI!F>MR]ATKE-1]#P?!?.T/!.A]S/Q68.IL?%S_1B>'Y7ZTI+9]-PAK>!:+C<
M9; A]P9@P,UIM'1@%FW=V9#[3LC=U;J4X:E\QWFAG6]3SCN_HNJH@MT%-Q,(
ML\X5A_!8@\#=?V (8_J%F$=1V:GT"(=#.'6]6)H1WJ[F'3/T7<7\@C@"FH#]
M #" -]9_',V*PYD@_&*,< "& )% > C@@ F__\RUT'+311H.0KU@A[QLFN&A
M';D-HAVA )\,S@1I^C>3<Y+*[U]$:@\% 0L#C0#W_9F)\H*% !5P!?"]^/70
MW0MOG"_M0V@&%CHR93\*%79'_'[,P?A:.^C\J[-WH1;9%F!J.1&+R/!#4,S#
MB12PNA)X>%A'1S-F?@0PF_^KL!K#\RCZ5Z7$2T7,NIYERGP&?Q9T,KP PI#X
MTZ\!/#QX5/J[LDM'\,$O3CX@0#NM'PT(9X26CGP4,Q@*&+,-#.\' R,?*R\M
M)RDH$7Z-R]OSLYN8[?6:_IE PP+E'K#_]_CZ9J>_F9&[*_#4W+0D%!2(7BN)
M2]->TVBG#,^\'L/; *]7#"H5P21.L?!?J_Q"5:PG\']5ORK4QBGK3^ NP^N7
M+06C^S"H+\P8/Z.Z0!A!OX%9I4^WPNR0,V+V\829#C%& &'6KZ&)-J>ED$XO
M1GMHJ#^ .(X0J68)*LN@<#ND #A$U2L/S 9C*S*]H.M@Z=)4H60Z(@!@I"PM
MW4\3L DLMKL.HCL#8'7^WMS<_WO[=_?W]^<C(T,#AZ>7]QLC8CIL2%!7]N%!
M23X3/[Q^^ZZ#0]TKX ?^[K^S0P<!P,;ROL_7GXC(V,1&7#82"NK$Q,+ X>7I
M65WUWM_NG#[G7 ???U7(.\E/9D8TE.#[F5F637(_!\<$\%0T?A['' !KU @X
M83J5("2X7V;R"MD/=HI8\W(,3* *3RM0KF#$[ASDM93A$EX9V0^:MH@B8:P
MU^8W*WLL03$]BTTD D #=(@@!TQA/M8'RP"# T/TD,CLKQ P-C^YP*&82*6]
M\27V@$R $21L_K+<&L28*S<0YV) =(\PP_\O^QLSS>QG9VU-75E;<^UK&SMK
M^]Q\CM70_?OSXPH_XA'_O_A_::N_G)ZW0\'F9O;RNIK:SD2WKO3?V#Q6-^=&
M7@SAU/\Y.1D9&!CX!3)I> 0]WQ<S/ NB%_;-?W\\Z]'PXA_XZSJ__K*S!+.
M)K.#V*'/RY>="U615S4]TYZ(6Y?MPZXW)?U.)/[BHI!!D<?_$ZLS/&W5"'!:
M%*\L>&>J@'T(2CM328(SPS-&XN$4L&8N5-W@NNL6$;?_$()6NY_=@OA _>A9
MCJ<!MM,!^@$_R0;T]-S>Z^_IJ?+VY.+^R],<U6,_/CL_O[W[UH%5:[ZP ?_O
MO\L%"S[[_H*$B)"(B<IO3V\G1W_3R^,O+SL[EY>7H[X&0%Z7P&#AB"=0WJ\^
M38#8<9X5]9";G]T%=K9;EC*\'QX<#&^MXK)K<%5 .!.FH1XEM8">9IIR:A Q
MYB?S(=;S /B5+V4:+5RT(?I70E89WM8@B&:3"KP,/XS3'L+\ UV'@)X!8P-_
M&"!I";(: E<0WF;)L$C($J#R@V6$"T(1\@KC#.\1,3$Q(V/\<8</#VJYT='3
MT=U38<?MS=B?,39J^WL[VWN#8S/]C1/[^[J_N__CT__]__CN+;GY]?O^E>OS
MX[>S&_F,;%:=^XOSN\NJ[AS5\6KA?34(*["[C.K"J]OL,-,& __70EK^K&YV
M"9PRO+5VJ0,(6@X0-/4H)'6Q ;X!5='I9H@ RD'I#'[F(-T-EL7I 5V9P^G_
M[8A[0=W>0!\!JP8GFI1>0,B/2EDHZ5L[H0<:7OTQ3RFYL $@&.Q=:28)CS1I
M)L%6PZ%5 LL\Q8QA8V)"]<?GH0\"H )* ] 2M[V_;&MXL*%F1__?G[N_/!\'
M4.T-ZF'>4W#O\BQG]KC=$^O'KAVY^.B''S]"?\1FXO_Y.1U]?-WU[>G'A@V%
M\$ %0)<'"TC&1D+8&V-6PP<> P<<\#L_:!4JW&5\4L/3FC$?"%:=>&$ZC5+W
MX+RI^@$BMT2@?^J9SS$T7M4/%&/6BD @'*6]@&.[^L5^ C[E(?X(S%2H$.AS
M"30!ZX F\YX/?D4G-TAGK;K@"E8^9TSJ53V7V>9EU>(SXT!@3OX<R\A 4."G
M)A,LD"U7P[,(?D'PA$#+]@8;%0->!K"IL7ET^AP6H]_0QZR/B&._[?<_ 1OL
M3>A*2HRCQ7=W>>)BHAHHO]^RJ%*3E]IX^ 5*BBIWM_ ]#94-&BZ; GB ^AJ>
M:,#Y;)]-P]M,ZT5?JU>O7Z_!9SYM;_S^YDP"\QJ>-</'/S]_^O4@.6J(,S\+
M+]'L^J9 )F<&[F<@1KY,3<M\SQ)I/IBL U<.8B>#B'9MK##*PH$7.+WOZLS,
MSIL9AQ/CPU0[R#T>S3. +L.#E?B TR K.+TPBHA=?%IYT"$?PGHTNY#!W*KT
MV(L&A)D)':4<#AP,'M;:VK^_[\ [6_<,)@<3@82 W_T_;X\'NO/&D15>ZS^T
MTL3> M?&W!Y\+@4J&"7B7P#O/&S\_( _?"L;.G3L]G[^>+@P%]W6\#!X,H@A
M\U"PU57XG2.L2-Z)]ID X;DW/Q^J%1?_^JQ?W.[J$H]4@ 5@F>9# !HS$Y^9
MG!!9^7!<_9>1OI#%!_ <,])?9=9@I7HP=BD%GM2JME6XLTPO<W8W#4^X:YU5
M6 @?B @\I5Q)(=L/X ,_D#AY$"K2&U^-PQ#.S-=CX5<K #?=&EY@!<YJ>*8N
M!OK"PL*Z]B,-(\<&_/LF[(8-&54,&Q@(X<PG:$&-ZVEX;=J'>;T"!$CX<.!@
MXUO#*XP"K*4'Y@#I"Q8OWXNP(/[W"!-9CVIK4#.1K!6 X"OK2%>^0T-#/D&M
MK1^S$LJ#I?^TAH!0EPW5,H\%63YA?F?R"E!7IKHL/%>L2>HL/*P)&#_A>1./
MO(/< :*!D ',21V=9/4=#':EZ0<^ BB9UA!#^(01/<SN1,,[6B7%[%(48G:
M<&:D-@@K["].I8"Q,[VMNS(#*KU>)J#@!I^&=S;'SCJQ_,;'Q^=C8V%A#LPZ
M8PL+"0+S]U$3)('"XUC<K9@07:N!#VQ@Y.8$%8K[+ H&AD("OC]8G6,P(!QI
MP?AL3@#_J!$?(2.['.@- /!@@J) M<<-'W!PUN\FT-"00&/CX^?&QH;>WX89
MGAJ=,;_8>R J94#J_V#SWG=&7]4'D1D29L>?9RA2O_*BB*'.C1,LB%5Z4!@!
M F:-^_Q6[5= ;1+'!^"!4 X!(:"C!!1PDI$1.+E8 (9 @B3*.XZ9_@R%$0^C
M_\@R$;*3'$ (?((A6")A9 (9YYY C437O<A4*K.*!SN>054P\-"3>^/%!"TM
M)A8><5+Y_ L'X\.#/.'QSN_XKO:Z\L$']^@WZ0]'ZS^7MG3_V."BPX#6K]=O
MXL\'X/RN3L8\AX]S]R]3/GJ_/]#@A^/S-R"TO&1@>(*,C P/"QB2?>ES#<\!
M>W^O5RH5 "F>&U5>D:H%_NP$@A3KCB3-+MI'FB)=4V249,RBC3F]8S,+G5X*
M\^R#V94YO=.&$ EWYK_4CLPK_4!_\G=EU.PYQT"QCQC [-^H@'LZ/Z=GI^EL
MGYRQ(/@A$##YH++,X+H.'T$7H#&$UKX3/C(G?[F>9_Z.!YH2<=E6+U[Y 8N!
MQ\3P\6'^H6)@8L!R E=FW0W/?O=<<"Y:< ^OWZO ^?[O]ZTCTY XH0N+.PT#
```

```
M\?&_X_M@P/=86(0%:@-?^A<?>>?$-S8\!K7?\.J&I\#\.<^LAIEG6!YB2K ?
MBN%@JLAF26& 8QEU$K881=OQ#2^&_HH9(- .]J]L&)T!4<'BBU("(*P;.584
M*N L$Y #=\,#R#L+D*P8^?#QKPX6?^SY5O(?J7]EB^H06N%-S!B>C!MFQ9QX
MAR[Q=@. @9B8F)G9. _A?T\X\/T?\_T>92]>/-[[($9\E="9_H\O?V\_+[W^
M?;S%>QL V;XT/:-?K/\-S[B<4 5K%R_XWW_X\UJO/[P:BT T_?SG_O(YG^D_
MP &!&1D9F9M9(P('!@:D 0[ELSEK*0L</P+^7H\[X/#@E9=V)-=/OP<_ 2NT
M\C@A_O/)FP,( N.<&/&- #BYI]$5%2L@ =E(+&U0A16$C0X(EA;(70<S9Q=G
M!>B">2&,E^>]9RT8P[+9HR=)*";X/"P;>P469L;T3HR>Y4/  J9/6A9])-6\
MH)@DLD]0S<WL[.SD9*2L];^/PW ,27G&NM_O_0_#__M]O9_'BX,]E6"_-'\&
M H$I7S?<%0.C RCP@SU0X,*A9GOMRUM0:ILAX(/Y"9YC7?OWO_##-:_]\<,.
M,)*SLS<W-R<F)33ZO^$,6:XJ#'JAY\M7^#]_0%=7_0%F&R<<GDD\U%<*U 60
M R#5X?/50>&)62@0$GZ!OF87.@'1* (#6+LA!:RM]@R@9![2X?#.Q[%TEX[\
MHK.6L<S! ",!^I9&I*" D @4 2!^[)^'"TD!O@N\!$"5X^P53.0F>1* K/4&
MDF14,I,S +:V*D*"1 B42$A(Z"@H*&SCNK,\7R_?+]"I#@?9F+?O+<J__H*!
MP/#^]'C77W\7"D>7_OT:O^J@/HY0-?\!S1L_&S5PX)L9[?=G34,\^O3[]!(2
M$A<4%!0V;6U40D$B$"FOW>KP'FSXN=;5@ 4J5[W_WSX *N4V_OC@H#.:@ (X
M5VFYRM\?U0C,PQA\0#\J>06?18,9KO#2T^"K1,L@+AHQ 8M-KO;8-KL!BP[/
M$!H<(U]ZFOS9Y4!O,1&8F. :WKBWY:D;1!V.!&;7[4SM8 %Q$F%8<\!*P&Y-
M9J6"%_ YQ-JZ]/#Q]_?V]N;FRLW,C0[/5PKZT/@<#@;43AT?#@\#!K_]ZZP;
MQEV9O-E"10?W2UY9UZO7_U?YI[CD?KCX</#3=GKJ4%\+'SAP8%.S\=X== ^/
M[^]O;V=GC#H:GH-Z@T& (K@JO0BHS?35#/9GGMG+.KLSM@P=7KN5$0L'9F59
MRZ7.[L&(@:X.SP-8')E9$&S[*((V71H9=7?H\.Y"$-'ZU9#,$[[,!]:% &8
M&^!L]G-F9P5W"14-%0F C0;%#@2JE1P=?GC\SGYB\/KHU]/A15#]H1X6I:G$
M3O^^?S\T;CC[2"!7+#BX<6"@T*C0A>T?^WUFYFP.[_(H%3L5T(##3Z-4E& $
M75/$H%3EKNL;VE7V0"CW930'&9@=>UCAXP.\'>O0@ '8NR?9@.L_N[PN,W9V
M S"S?UT/Z$$O(VU&$LCC#W9AB['7B&CPRH\$##)CFP/ RUO798 2=J_+6-+R
MI..@P E,%[\[!B=G(S,S,S$S$6H-; L/!Q^/!H8&@M9"CY\9-C5;9V^7]JBS
MV/CTY0*J/%Z@;\" ^1O>%P\7'668;*S:YO;I[V!A++>B +-000+H]!7X\>3F
MQ,S,S(S7VAN>)9!?_#MGJ*M1_FZ39>! EB<&AP*H)=7'U?J9C<3\S1 5 *)!
M /SEO C$[$(_UY^JU^2HD :0V4O%91)FA7F$\TD4@5='O JB2)2%9#$!H"Y3
M0D4:HZ",H#.GMN@UA@C:9 54!,"9H:U8LE+5%IC9 0625K^E9[=D$G,%$3,3
M$Z6K&8^/AX>#A?$=@(+%"H+@PG#.O/#\"/CA^'EF7Q,SYX8607BEW\4:EL \
M93]_:D>A_W]^\3,=#L]S/0\_-)<!#D$!EP$#00/Y^8'YOG*(S,BZB)A8":DL
MY@65A@I5KW],6\>J7S-?NF4@S_4>OO6J58"E 0!0J+SO$729!!%GS@(&7$@X
M!?AE7A=N $SFE'G?DS@!IF4?H\+L,55PF 4U I 7>XXZG)S9H]^>FGQU+<!S
M/)Y%60TY!3',6$&.+FB"!Y=AYMD^ B)EF'D ^&)VUG3(H#=W>G"$%WE>G&V)
M8XS %W@Q X</0>+ XM/J]?^M9]C51.YG4_B=9>!@8.# 8.CX)W4U/!_K#&\M
MTIS_,#!#:0-'RU>O@"V!@X.!@9F9D80Z,6 TO#RJ)(U84P\ 9@%QI1L%RL&J
M0\HV<0)V3)U5IF=X,EHKLBBT" '#_,YF! Q.ZK'JF*GL2>D#.-O[*^^#"$,>
M#<^&MC@C* IK,?;))@PSTY,#!P&S_\K3*GRDHI@S7N^=E .2 #DY7,QCLSX!
M>?5&84[!\N)U"Y3<&C.ZCC"#WQROG0_91/8P,! ?@*G3HT;2E>P%$P+2IE3.
M\#0W-MMKRFN&N+ \>2E/HM?<8%V%Y977SB9G3%L41K(4@^!LP>",AG=/U$>\
M"("@H!BV#F[9@8".=3$X=/U) )!T@1L@$""(:2)S#4!"@.D:JBXQ-;\#) J
M$+09)9:(JLRXA -Y[D/..@Z25;O9&PDU1E[DB"@L3XRX (*9&\F(6<LKQ)BO
N!@%42FWF%I$$EP$TL/R/'O2CN6[[4S:>B<)8BB[RUK7\ S4G D(%] <U/3#
>Z:8T/R$2Q<G]'.W&3'\6HX8^6-YES: &[ZNUT/47(KF("KH8*X98:C 2MV%
D6PVO+9D"L,D R"\BAS,.485 0JBP+-QY.SALFQX$E<<P<"#]P!G;I!M>(D
.YUHS0%0"$>TR]O_N+DYEL,Q5E<(+C\-3(\&J#6\!$@X/$T/JR,^F=O,"^@0
+RLX  X /X$9F1[=W>3-AF<F8HR8&'%I9;: OLPC 'W83':Z.%P^8H28!CW=
G1!2 Q7I_["JY C, A$L/1"*9OZ EP'CH$)/=P2P=TX,3+/'DERP=P/,",:9
QP&W[LN@O  #O5*O*[45=Y;$P F'\P,SR)D!L5,PQ Z[0;;0$A@  RBAK]6U
2=840(AX<AD (@!>VA9J1 HY QAQT."((W64M0QDMC N7=8P3L+X;=8U$L [
P $PEC='/#HK8$0 *->1TP7Z# $$+F_#,;.6 4,(GYBV@H8L!2TSITL;W@U5
-C-U=]?)@ UFMF0((Q  9_I,7<+H4SH*+B4WJ%BL' "8 *[ ?=Z <U5CU0#
0+LA^#&<N&[=#<VO'NZ@  FWTO7O<LJ.@/A""&"<9IX;GC:=J[Z=R^1BFKG-
59\JE1'F1TNF6^K8D=(G8CS-M*KJLNA 2>TB[&V4MGZP$^TQ2VD3)[.\:@L8
*]!LU4Q)DPC+E#2<RT8=S%.Z<<I2J;YJ"Z^TP*H<L'BDE#:^:L7-,V 7RL75
4<D,)9GLQ[M6V3-9G_365QTRF5M>]2V3?:1D:966(52>R1>T&^9=3-NH_S &
Z$>+.X2%B8J.CY.M7UJY4=/'6OUY5ZNO9$QSY/0PHU?=D_EAT\9/MD9-:@-E
AD*AAVB3S3:9CK8*!(L/YTM]T\63%4[=C6W=M&42:/7:;+%5:BKQNF_ZOI8>
OE,>IVNM/($RK9PRK*X K(&R?,M/YB9C-4X'89#&6GG&R<2I;_)@JQLG\P$+
G#;/JXA'\')SK;1^LC?72A.8&?- *TNG0"L[)\V3S& K["X=<SIJ4O@0;C)X
 _  @]<6R!CX!\B4+=1PV3JT[L8>V3=SEN+6B9HPQ1@0S#&7$%<KE)B9G9ZB
HZ>E9JWV&,),G*X"K-E*[67LJ\3LKG.H6UXK/<CT=NV)DT''8*LWF>=!BPG3
XLERJ$PKOR,S37I.@DV? ,OST#.]H5V6,J;G3>$5X8@49L(B'-8'61S9;-64
2:.RR0(>O<N.**C+<&5E[*Y)*_BYR,C.QB_4:]H(DJUN0C:O7<Y*ZR!YE4^Q
5L\7;CXW#W85M@9;J3%(Q:59D]95F3/I[;VFBQ=*/",8-'LGG0%!H$3IX#Y(
$*I+7A^@*#+IR<L,PP^^8S;= PC.5&\'#&88&+:A/@"HK*VQLK:WN^%VK=X/
W^QEQ.E *E 73IMQS+8ZULM<J]RW7F&73WW39WBU0@%>+Z^?LTEOM5)/;6OE
%R*UE;?O<&\KM6?2PNG[I\S"FB:[,*K1OHPZ]4"U9;QIS8SZ%!TRMKT:HZ0>
SK#&Z9BMOGQ\C,5I\Z^.YW$9P<?!<)XGFC*7Q4L;KOH*%VMP==G)GF"T&(X@
K@)YQFD+"SMRTE@KK[=RP^V0Z*05;(A2V)=-JN&;/^P9+VNZRQ= 'X$; 0@A
@"#L0!@$@#U^S[,*SU[L@'X"F0?-]Q_=V'YC$);_!3^#N 27]B9A2TZ\C<7&
RLO/E[U;;<%5O)6F;G4@($!:!:,B<J; R7JKC3ZA5EMNI49+7A 5U*-Y:<ND
C5.OSU6W:<VX3TXROU8:<:6-D_=;K;'?RW#:.B_M_/+0R;&2DT<AB=>FH]BG
FFY%-$D+P=$-+%7BM%9-=DY:,92K?*XS2)A:I?S%(J[ OD./N_!<T1UZ-*S*
R2/(Y2D(83A4X/0&7+TE7HV36KFU49.EQJ;IJ*S]LM%PO0$QGZQ$*QLGS\I.
3GB<RZ#P98BT#6'$(\^&,8IALSN]Y0IC@);GR(3F,>QS82#]*)<L  &E8)5V
 PE#D #Z 98\ -#4U=G:WM_CL]6ZE=^"H)/62>.D'%>[,@V\E1XWI5O]<9*=
D]J362=-N!RRZ;3VG1S,%>U+#8;3DW7:8E,)M:73QXX=2@NNL;2_E9=.W@5E
ZJ15.IRD<3OIM(E5]DFIE;&3J58=4F;&PVZU'>!5%<#<4N:MM$0LY\A\L6]\
MR&6^><IIK;9Z*Z?'QS5C4[756SO%6VD-50K5$91;>9NQ=7"Z&P:U^@XB" &/
M8H2'# B >$W/[P6"M-;OTT(9"^9*Y% #>7F^064IT@_  2$4;)];! X> (7M
M)?R6/@#DZ.GJZ^SM[J-/)TU$&'&2D_>)B4@4NX8Y:2Y5T+1\-<<TG3E9)F9.
M]J !+U-GU(*FW+M*UF#@=%H&Q"%A27&=I-3,-T[OY399H2Y$=Q_RFE3]'7.Z
M$*)QE:9629PR IHT&19'C5/"3UADF.D6Q,*,N$FSI ,X;?*&##=I WF3M<HW
M+[5&GAS8;1T4&,O<'#+>9*P"XL7(3"Y6%]S(Q49_4^?IF+04*0/+D%2F'^((
M-VD$81R9VN25QZ3)!DP&:W++R#&]R<@NDP7&M&E;KEHFW4 C,/" $,( U&40
M, .@'L].:,M?@("@7#\A"S?6@] 0W+6(9Y7;P7/@.$R>4S51<($EH98_ ._P
M\?+S]/7VUM_!_QGX+,' $!!2&C"1'T@0$!='-!#ZD&GO_7+#'"4<I ]:!AV$
M 5[!P<$,TA9:9![(#+%?",P,#T)"H#,TW 93P2DH&WHI&B'-; F[,,TA)57O
M4A!&D#@"C+#DT.\985PR"!(D)&+>)-#%'>UBDYJQ3]!"&L3<BI@01M]+1=Q4
M0: 0T='1'=PV,>\&4HC)?/R. 1VI?,G!(D_D,BS'EQU!0=%Q<<_@8T^C+Q0?
MRK0=2A?*R2+O[>WOTB#-8,[&9L? 8,W-#]\=WP\OPQO-H$0NSQ<7.(+-U0]/
M8!E7%S!#D, (5@7D&(0QFD-QT>,$P _M"QH=[8,;PG,7'.PD+E2T#QT=EA;V
M#,"!RP_,%J,E'@SFB!%8U\(G+$J5<8P'?ZR ) -Z ;5'Q.SAX)PLOZ,5HI8&
M)-6'C 5RCKCBD*K[EHB^E(B)]92/CXR@EX&3H(V!A(6@@ID#U0J/DY."CY>@
M_8:@@Y*%DY2@CXZ@LK*NL;"NN;6#U\7W'BX^3EYN?DE-'G9[=_]<KQR&& +J
MP'SR3L%S@9([@?6(AQR0@6W_<MV \'ZV&[0:$8;>EA>FU<Y\0M(AP4^K)\AQ
M*@RE': .K3-_&/?!7(#.>!G:%S(0%JQQ(T,'YG__3V,X96 5^!7P?-JPD'$#
M.\$J!I^ O/BM-U .?ID/)2$W2XBHW9SG7SRX\%[,"_)*53LV("S].GE;3@_G
MA0@!5CO? G#,^6W6^O!F'RP!!G,QFL&=WN-.>%;04 &(A-&:#VQLF:XT?Q(R
M72FH:@'G$%WTS3P<_ BN90@PH#Q!S C#_'^57 @H!X2E+R@*'B,("CB6.%-G
M0\\Z1FH#IAXG+"0H+1!\B$-S,Q,1F "&ZP1JDQ %2GHDI)B0D!$C55C,-&04
MM C P7.-0L?GY2!A)RO)_%0S+I;9 YR_?[]7J@0L;RQ3AL,J*C%E],;/:@E@
M9?RPG?ZJ")I 112SA_#N$#CU.D <85$W5P,* ,*!SDFAGGU+0^P). QGX8%\
M ;L!/%@MV'MAB!)^ASX#JZP_/2(+U!1*(.!#Z97* DED#C4 5B?VTN)!$(Z3
MAVC19DEK!4$>P*0H.I"I*N.C/<60 !5C*)Z!$C@4_MW+T+C[1F?W[SB-'Y6=
MEQ\18V(R;DQ84IC87'P_.B>\#PO6+/4%(DJ5Z+J[@9Q)<OQ\?GY^_GZ(QD9,
M=C(:RBW516;XJ;GI^.V%LJ(>J?<G"D5"NW^9@6IT%?Y,2P<_EG9F#QQ% /Q0
M #X>#.N7 1Z(E2A!<#5P84B>;  *^FH@P&S,EH6$M@0#H5X,+'EC@H FB5K/
M91 W^1 D;0@5I69:/>7# ^*R-#@UZW&A1F2I!- S1I-@-VR+0):!!8<53X T
M$:P0V$/MXV.']:>*>$NSH24LLU<YW=B<4->X4##W-;KP@+('!Q\O\DGC^&1D
M) X=;! 1,7@TH1-9X+A^N%?Y\?/# AX^?M.?]Y7K/[Z<G)C,KA\F)B0V.L@(
MB(S@^/38V*BM'^_$KV'&J[L;.1T9'0K6WR"SMK' X.R??M4ZNP^K<L)_?)4+
MK"K1V@'&QH=#$,(L;X@BGIFGUTX&K"($$X >+06&W1IVEJG:;L4MZK8 2 :5
M=')P[SDKM46^NBL9AMB&/-FZN?G& '[R?@46X W@(K'AQ( Y12W<HQ0.WV5
M0%X)T@(1OT$9S'*.]S"8SOR"2VN?-7 P<"!@*+O] @8'LNN-A3<6.B(C8T%V
M2<+6U$X\/SX_O'C[[J@7 _CON^43!0/A?L]O[_^LK<XL1&.O&8)NDD-K94N5
M&=*8X9B9M> .# X$V:<+YK3P\/C\Z!VJ@0IJ]QY)C.)@9?@<_/^0!&)7_U#-
M%JQ]8$ 8>J XMCT?4$"?\?F"F!E=%C'\_8DJELTX'X4P!MSXJ"& 4':Z8VQ0
MA-E5!&#8M@57&!^$' + (?6/#GFQ"=L!?QAXZ1<I'+!P\%D48X(,0$#I"J"R
M%E8#; 7&><\*;* 0$! Q,3:W(H>'.8<'#V^ )G_9T5%;47/$C(C613$CXWD[
M&QN)P<'@_/D1$R,B/)N\OA&1F)APZ*RXF)$V[F-C;",Q$1F)C,3&!P>+8YO*
M@NI4>Y_@X>7&%N<-C(QE%D1S@1<ORDZO^I2% V5 JE7\ -][51;P#UBARNBR
M8%^Y4/7R"/_/&#_GNH 80;A-WP$?7Y0![$D[ND$M+#\#F0HQ3ORH WH"J>AK
M$(+]B$XSQY,&"#^#&6UQ0=:A*\$URZ#R1M,)&6194[*E@,0,CH9CF%ANIP*I
MB#7RT+,B 6!,D-ZP*1',#0 I0Z!5>H%"0D)#0]J AFC@XJ(!T/JCN[.SB0:A
MLL?#;YN?O+R_\'A_R'[OU38>:EY0:OW_U:JYL!.K1ZX=N]__P-_B?\*_X\/V
M6WT]/>V-S<7-UJL!A4T6!T<V"T ^\4+P@)8,,H   P\^^*07G2O\_+SGC?QR
M#QT+#B4'P,$4!W8("!P$": FYP\%* 438F.) F3E8N18#OS2B@ )(.A@1$QP
M4DTP[LJB-F! (PJ88 (<%("I".JBUQ#+T]0@ @0D[%/@0 +@#!C('5'?@3^!
M/6]UV@"FSD\#PEG%2KN*ENS:=)V $8 "4 $\]?TX[BD5H2W$AB@#Q(2$A-+0
M&Q43%0,7H;"!L(A0>')/+X_1U\3GYR$Q'T'#/ "01E6,"TR.JTUW;_CP;[ 0
MO Q-GKH2-ZQ6PH3!RXLC9^>%#8$-$0H>CDL+6&A(:$#HZ&YC(2,A(2$ [P%V
M%:_W?PHT>PI95\4BK> K72E\F"[!P\\_?]M^_^J/-65$" 4LCXBU!=8FF!4>
M@@'($]8@_1DSV\OL3)(:K/"" <B0U9@)0"8H!Z*4RS!B^1S5N#D0P$ IIJFT
M[QC & #JB/T((,";@!__Y5@@8LV@,12959T/,3Y;@B$W]3&.WI8G--'>?.L,
ME#8&$Z4.' X>#/I>7#\OK_O_76H'@.IE"8""B7Y_7X,#G:-(A>P%S+L!0L3+
M)_;/S#X\?"DJ$23Y_Y.X6N3LI1<<>Q\Z>GS4M-Z>-96PN' X<'@P^13F4"BO
M%PNHOX3J]>65L)9 @&%_@RMA*W:!"M(4_?KPJ&A78 ,1[#?559R $G55I:'A
M35''V/.$?XS0 '@4$_ <7V'U5 7TU%<',+G"RI AP $.7S5;5IRSY0Q6F+:J
MK$@/B A$/DY"<R#%BQ#/(I:0*,7J80XVZQ+9GQD^6$&  X2;9C$[ \_WY]&$
M-,*&PH)"'Q??G0H/!@S\VZ;,1HT;$H0*_%3!S$>$0+*ML%CBSD$&_-><AXQX
M\-GQ()U<,P(%"PH7-_<4U)$C1M^KLJ!^^Q$;$X' 0?C&O8= DX'P(;*-0V%#
M04)L"/KJDU>K8Q&*O=7JPD>N'NJJJ( I+0&JD(FOJAJ+] 5EJ@K7 *LXJ#]0
M%NT %Y08]RT4+Q&_ 5?6X\.@ =E,TAYG%A_=""C6*P /? =H;>.O[(1[<<2.
M!Q@^ Q1MZQS)^)QJ<HQ*(**XX,, +6"65057M8*. 05E0[3\S0IX"9Z#1]DB
ME!E3I9C/EC1FVT;&0F%!82$-#0<*"QOZ=037UU* 2/O4_DB*II\\] $?9"#D
MY;.[Y%G[#@K1NP8#?%\T4#$P8&'*[(_>_60!_]2"'Z&_@SSM/]'C4N*:'5N5
M%PT:M0G0T)53 $)B8T*"@H:$G_4#;8N.M!-HM,/Q@ I556R#4WV#S54,L.S.
M!HE88FAG4W$"@D#9&:SXPPCALH$U6!W%*)@#(YX])I@#]$] RAE]RY"!$5:P
M4HC%A@)<<;;I,9'D3=\!C ""+$C\,2!3\:0(AQ 3V?M"4BG &;! ":+>?%X"
M@!R/P)BEERE30%4P@+-M2LXPL 9PGPHAU$L0LUP0&)PY##86UE='8E'L!0-1
M@^,\Y?"NW_VUYKKBT0_R4.6WGM??[S?H+?+4PV/4;]]_;_+/#^#?RJ_BS#RG
M#W7[GZDZ?KIWH,"*P>"LGC!L:,:YA'CH",P(E.'F@0AJGP>T,,A_OU\J%6D/
MX*I5G$4'H$ %DSCNOF\_* 12TAM6"( &M%2GD:V' I: )0@(Z?LP$'LAB"I6
M(3OW,"F+81A9N0?8VOB (8WP3((X@3(.LQGYH)C>^,"^@QO,?;P],ZP'%(9@
M !G _& (8+C5&K3 7C"L@XD_^#G.] BP]7></_CX8VAI-_F[ 6#PK>!^A8G%
MAH[HLK=R3"+ 0CH KRA_FOSDDQ+_)???U^_6_^_ -Z8.4 *F!T X.0=['N&_
M=]!@[\R ! <U!S_O%Q]_[XWK-T%U&JA. :Y*H-*@?N#X9'\R1RJ ^#GWM0$'
M+Y5&OOI43'HZ80&,)QMF6PP[Y ?A*OTX8I41K& VAQ/AF0%)TUA8N0-P4,Z@
```

```
MT5I.X/6T!4KK@88XA]BR'F0"5# /:6"Q #@%HQP!/),@6#CW38X"> !3F4];
M#3;YRAD1B[9 EC.GPU=1F(B9B7H?YWI_?C_^ #A[13_Y4:+F^$&7^/+Z_/[^
M%G]?/]\Z^WTLFL4XJU7 OW^^6=)!#$B2N$YU)QZ?+XN>_A\5(US^18-'%9?W
M7OYF_/^*IZ01&1&9D?8'08NN#@P,'D!Q \"$5?SXZB7PX.! C[:@#P_RFBB%
MU?;X03(X[JDDV@6?F!][&81_ [X#E2>%@ ',3(0@F?^7HS!-[ 5Z Z@#:H^$
M_,(O<YP #A"E?BI)D?VHE*D!RZ#S-*Y(*&*P<J60S.4.?6%5_LH(00J^&);#
MH. D@9XFADA>^P7)C<ADS&1D9-BUUI_'>0Y)_(12ZO45=^/[_< -?M^>@)<H
M# \GGW6$%4,4N;PJAT"%4($?&ZK4)0M\A>"[ *$"#S]RGGZ#=0$_(4K&;UK?
MO\<;7W_YXYYPDI.Q$R8S)B9Q?4$,! 39):U"**W^H(Y" =:3!PD0\:A0@/VT
MBJ 2 !4 D H0W^5#7(E;V=,I$ )B#&TA%!<8D9H</)<A*820*.["H'1P0K'.
M*].NJD@>% .+6;5@4 EE-\!^C" $SDX!P. $E <P1JF;AD(8X,(\*2_V4(C#
M3O^N+B)0UE'%)"C $C*?!)2@I*8F0H($"'[@2$#(0"@H2/,%" <GGR?0" 8#
M 1 6*$(5@L6"P'"NJ/V_DWI%JFDNT/+Z_KSYE*!(@(,''S<;$&"5VUH(:!3^
M9T7D^>0"$@(3 A04$B5E9$)!(! CKNM,#SX0U@I_!P&@ 04*5Z]$[ZA5OLB1
M_/ <?\$F#Q\?!)=X'PZ(DE3T^^H_2H73= -8*49Z']8-6"]B]@$0 8;6@:L2
M@A @M+"4!413-B22LI8197G$< #<UGSL'4!U8_#JE*OA\+%/9&\, *R$/SV;
MP6P]A0(R". #E<'X5NH;(-8Q,=W@/Q!T^H(7V#J>G./W8/'G]N;TDW?(B(^3
M,P$CI%G@? YH-0TR$1L,#@L$UFO5?W^_?GU1#>O_JO^OZ*L'^UVJ7R'A^W^O
M5J[8['ZP3(C8,'!0(.C0F@<^</CV$Q'QDG=3!H_G;V<O^?-$</@FCG*H('",
M#8"IN!*\AY]C*J?D0:%W=2S/,@ETG%D57MA9 1Q#;P8#X<=Z<U63!O;81'/P
M4NQ' PK(:BX"P(@!@MOQE$"-KB^0A^P!!B.4Y9[D P^0V FK!@W,>](G%^</
M!MVXCC< "V5/,VIF3@^P9F(\#%@) @8&!@<,"@T4&?H\_+!X\/!Q]=!?H*"@
M4&_M?$ L#Q=L^^]\;WGN]N[WI,9R8#!0L"B8?(>S8&=_9B9R@F9&V#_8B!GU
MJ @CF#[T9&" (D!'_P>JY*,J1S ^ SL&9\]: *Y '5T4 P@/A]48\HW68U57
M-7+J"L?93#&[L'RZMDP*_ >[2C\+J.A!.7E< +*(/%D;6LBP= 5'Z$(B>;*/
M+*)03J9 8:R,[ $E9)QGZL9,,KM)=2K@23(^=C4 8V-C(WPS0&H!!PG\U  <
M!DI!@!DT(5ES9[?G_/S\>+EXN/G@<6TA0 #9!  '"W<>^W?  WM]>CV8+(2:
MSN;MY^@!#@5%  6 QL;&Q,+FS#"" 0I5^O[X5T#W__CP 2JP"0I5JH8EJ"U
MH$#H_=ET/[)!#C@)@)(#(//Z!$SMI;Y?:2@@ ].NAA1OCS <YD)5PE59!@S@
M"0B $/Q*RLD1<_BH!MI7]AQD2L1M "<!F!Q]9@(6J@@,EIX .< !+EI&!4(9
M+GC@ 200!R1>\,!@,:Z0K+H@@G*A?//:EC( ,S$QRF/1#P,' P,#@X&&@(:]
MV1PGSYS@JB\Y*XVXV?D;L42*!LX:,<A40,# '_\?O?\'!P=^/S_@X>'AP?.Y
M_1:57P!A 6% VEP!4HF *GS,C(PZ;384%2L5Z@55KWO_:0&+_Y(U]/ZR__VH
M((# Y/#DAW6<$!--' '36@/>13MJ1$)'!Z)%-/(DQG4-XJ9("*0N0V,H9@N)
MLJQ@]I%\+"IC?@1*V-"*F3! 9Y]/4U;[()GA^%I>&,$. !M #^RIK)L&2I_3
M;6 &&=[\(&QQ-B^<P%_DYX07]):4?G<92IB8B(B(8SN#W)*7H,&@T/YE_U-A
M"203F^@34"!@8!#0,."X2BT/'P\/'A[!P8&!X!XT ,:09^8G X#H!0N6"(&8
MF9D9&1$1$98B^$>C6 H'/_*O$!<&FHC!A4- $Y,78B,.J38P*COC!/#*0R<E
M& 'RJS.C,PQ$<4-WD*:J0,NI.N"S!8,( \*@?8)X(S]2&-9."F-C;Y]HZ:>'
M 1ANBA:UD+('DQ(&S$\'50 O:_CFKP"#&P(,$0WF0(1(_WF:'X>9&'K8;3"#
M/@+8LE,=)!_:F6/ !A _ YCQ:&:&#/H#.0)J3R_H.1XRXA+,R:)2B&36%3-G
M,NNB59V?#+#,4; CD&"4)B$!99CATSL8QY ">-H62Z"4)8UB41> /-(NTNT
MH"$LX&P(.@J.@T'@.>IAXTJ9 $D .$ @3-[.4% @K,@$ PIY.X(S @,D $7U
M0+G"0C@8/ &@$##%>SL53@%P@+2?"/T#7 Z&&I:EYAJ$&*B3\,@3!0$Z-PR$
M@D&XGQ,& ?H>)3 S2$ZE "436@#VMH!  CA%U_'ZI !-.30""83V':4P&:91
MMZ;L&,P90:;&!5*-.UB(-_A[CI@MN #.@GR$,,!W< TW?6R3NWHYW!A!C-B7
M@0'&V0";M\Y!1GH(Y5R5P/C<0T  S* _*4TF%E/"#$E(GZR( LG@I!R R 41
M/!"(8. ,8%!"!TP)"W5&-];#"'11[@'8%!\/RDVN\)4% ODH0D!MA8>(" !T
M*#  P".K_D,>"H *L0L'N@A( @SL$!S3Y@ V!W=)BQKL$3AQW@\;70$^ W,#
M"""'^ANZ9C"/"Q-T8.YP <8)] ;$!!(/>V^W97F)H)P<0'P.IMD(72?P%,P*3
M9@!@P,;#2$1CK@!1P$^G!T>% *I[R(4$)C.NF,9!$,!DMB:E$-F5 !3%D 6%
M_&< 0B+@TJ,'>D*" 6 #\M4"@"03B.+A]2&!C=^  X+ 4MC59I$.,4P!X!_$
M&2F8  A(91.>0(*$AP+ACP!-52+SOV#14X"DP <#P 3$D"6&6H\$P! !#1^!
MP#A"2OP@\15$)> !>  8%-*:AL'E0L&6!T)SB7$#!L#$ , "DM  3-&*G_ '
M'B4<(4\P(.B C&C #0((L@ F!P^$"*5T@PGC* %_H+P #43"$+I  )@W_%W6
M]((X"_<RD/B$$( W! $ UJ$08,; _'J,4 $'_IE P A@@!N[^ C&?@B  !W!
M_ ##ZQN&E@8 EJ2]((('_J0"-(-"-(="I :( XSO88-G VH#;!X/Q2(>0!""(
M006"B00)!KT BIT !\K0Y2#D__#[K38(3! (EKR]E@W<7Q04!>/SR3J6@W\0
M;)$4%<2O%P')2Q .43$P,"6"YJ59+DID)$$(#P7.H4$$<@(4L S[@HDOH9"&
MX@(14#HR,+B3$P$<1&]J0D!<TY68#*B($IP0T3$6&40)AY<*=#FV).<XDQ%Y
M321N30Q>#P\,IR_#1'!)0HR,#6/%48AI QJ)9",/L049)0LC1UP@"@]R*ND#
M"V B""X2+G0["0=\#Q@I2( R#@<@(O0@,B(L$GHV-" 71WH((P,Z"#$D%P4!
M"V,: C4("08,"2T%! G64Q*6"%P.!,#SX"0@LA$)#07A]0586HTY 0X+>C4-
M%1,N 9 H VDT#A0@&G072Z P#! )A@\4$AF@H4 @$Q ! P4" 4(M!@\2( P/
M!_ [#PXO#P8&(2XNK(U8( @%$@4G$R 4#R !# P@&0\5(! #+0<5&1,AE@0N
ME@ ";B"6&+OX"PC&!YXR,#8QUFM79P"-8BZ-(="-A@+CR>6B)[VZ9,,$J0^G
MD]C)//()F"@]V43ZHE*@>(0!N5@(F?\$E+VYKUB9 /]!/O?.&:[.20B1=^Y,
M"P63C(6$AX6(KJ")B>C(L7K EM 23LD T >0#JAHJB!.!8WH!^Y Z@PURN";
MB-#OS'30S[PG$7HMO"@%E"[ [W(!AJTSK^Y/!= #[E %Q@'JZNKF 6 QLAL8
M+3DV.B $!04+ 1DO Q(%1*HK !@@(" .+Q Z$Q09# 66'P"I&\#J7] @J.5@
M>*DT41N@A*+]($3A!WT@1@BB +UA")W<0>C0]ZE,C0T(J3>% 5BB'[V,A)TP
;_>S&]TP1EX0\ACX_NH0> USE@*(^L3N1/<C0

end
```

```
/S02::$d000:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Scrollies exposed :

A look at simple scroll-text routines

by XmikeX,

with contributions from Asger Alstrup

The first 'effect' I tried to accomplish on the C-64 was a "simple" 1 by 1 (one character wide, one character long :) scroll-text mover. At first, even this task was a chore. With the help of a few friends I was able to wring it out (thank you Paul G-S) and eventually come up with the source as listed below. However, there was more room for improvement. At the time, Mr. Asger Alstrup graciously volunteered the suggestions following the original code. I would like to thank him for his insights in this matter.

--

```
!Simple 1*1 Scrolly - (C) 1994 XmikeX

org $3000 - a fake op...tells the assembler where we want the code to start.
            change this as you see fit, but keep it away from your text data.
         lda #$00  -
         sta $fd   - This is just so that we can do neato ind. index
         lda #$32  - addressing later.. ie., we are setting up the
         sta $fe   - spot where scrolly text will be pulled from !

mainloop ldx #$c7  ; Prepare the scrolly register for a "hard right"
bitloop  stx $d016 ; shove the "hard right" value into scrolly register
         txa       ; Preserve X by shoving onto A
         pha       ; and shove A into the stack (stack = temp storage)

             ;(we need to clear X because we are going to use it in a delay
             ;loop)
             ;(if we don't introduce a delay, the text will scroll by
             ; WAY TOO FAST to READ ! !   Incredible, this 1 Mhz bugger
             ; machine of ours is, eh?) - delays not set in stone, play
             ; with them :)  higher, lower, etc...

         ldx #$05     ; just a delay loop - ldx counted down by dex
waitloop ldy #$ff     ; a delay loop within a delay loop
wl2      dey          ; decrement y in our delay looop
         bne wl2      ; keep counting y down until y = 0
         dex          ; decrement X in our top delay loop
         bne waitloop ; repeat the y loop until x = 0
         pla          ; grab X from the stack and shove it onto A
         tax          ; copy A into X (X = #$c7 from mainloop)
         dex          ; ok, now we countdown X
         cpx #$bf     ; coarse scroll anyone?
         bne bitloop  ; back to bit pushing
         ldx #$00     ; clear X
rt       lda $7c1,x   ; scroll the last screen line one char to the left
         sta $7c0,x   ; we are shuffling one char into another
         inx          ; increment X
         cpx #$28     ; hex $28 = decimal 40 = width of VIC-II screen
         bne rt       ; still in coarse scroll?

         inc $fd      - here is where the indexing pays off, with
         bne ty       - these little lines all we gotta do is stick
         inc $fe      - our scrolly-text data beginning (in this case)
ty       ldy #$00     - at location $3200
         lda ($fd),y  - :))))
         sta $7e7     - shove the data into rightmost corner
         jmp mainloop - let it scroooollll, baby! (end of loop)
```

$3200 is 12800 in decimal I believe. If I wanted to "on the fly" put some character data there, I would just poke it in.

```
poke 12800,65 = a   at $3200
poke 12801,66 = b   at $3201

etc
        etc
                etc
```

I like the code as it is, but I am looking for an even smoother method. Any suggestions?

[...]

Hello XmikeX,

In regards to your scroll-text routine :

```
>              ;(if we don't introduce a delay, the text will scroll by
>              ; WAY TOO FAST to READ ! !   Incredible, this 1 Mhz bugger
>              ; machine of ours is, eh?) - delays not set in stone, play
>              ; with them :)  higher, lower, etc...
>
>          LDX #$05     ; just a delay loop - ldx counted down by dex
>waitloop LDY #$FF     ; a delay loop within a delay loop
>wl2      DEY          ; decrement y in our delay looop
>          BNE wl2      ; keep counting y down until y = 0
>          DEX          ; decrement x in our top delay loop
>          BNE waitloop ; repeat the y loop until x = 0
```

[even more snipped here]

In the following, I'll describe *very* basic features like the concepts of rasters and frames. This is probably only interesting for newcomers in the assembly-domain.

If you are looking for a smoother scroll, you would use what is commonly known as "rasters".

Your tv-set displays the image on the screen using an electron-beam which moves from the top, left corner down towards the lower, right corner in horizontal lines. If my memory serves my right, PAL (mostly european) screens have 312 lines, while NTSC (in the USA) screens have 262 lines. We count these lines starting from 0 (263 total), and call them "raster-lines", or just "rasters".

The Commodore 64 has a few registers in the VIC (the chip that generates the video-image) which reflect where the electron-beam is. We have one register at adress $D012 which contains the first 8 bits of the raster, and another at adress $D011, where bit 7 describes the 9th bit of the rastervalue.

You can safely regard the rasterline as a Y-coordinate on the screen that starts from above and works its way down.

In this way, you can use this formula to obtain the rastervalue:

```
  raster= peek(53266)+ (peek(53265) and 128) * 2;
```

Note, that this isn't of much use in a basic-program because basic simply is too slow, and when the calculation is done, the rasterline has changed so much that the value of "raster" is more random than useful.

Well, the value in $D012 goes from 0 to 255 while $D011 bit 7 is 0, refrecting that the rasterline goes from 0-255. Then bit 7 in $D011 is set to 1, and $D012 goes from 0 to 55 on a PAL-system, refrecting that the rasterline is 256-311. This is what is know as a "frame", and the cycle starts over with $D012 being 0, and bit 7 in $D011 also being 0.

You can exploit this register to get a smooth scroll in several ways. I'll describe the most primitive of these techniques in the following.

A smooth scroll is achieved, if the text is moved every frame, not more, not less. In other words, the delay loop in the scroll should be adjusted so that the entire scroll-loop is performed exactly once per frame.

Okay, that's fine, but how do I achieve this?

You simply monitor the rasterline. A simple way of doing this is to use a bit of code like this, which waits until the rasterline is at position 0 exactly.

```
wait    lda $d012       ;Wait until the lower bits of the rastervalue
        cmp #0          ;is 0.
        bne wait

        lda $d011       ;We also need to check if the high bit of
        and #$80        ;the rastervalue is 0, in order to distinguish
        bne wait        ;raster 0 from raster 256.
```

Another, slighty optimized version, could be:

```
        lda #$ff
wait    cmp $d012
        bne wait
```

which waits until the rasterline is $ff. This one exploits the fact that the rasterline never exceeds 311, so $d011 bit 7 will always be 0, when $d012 is >55.

You should try to play around with such wait-loops, and with a little effort you should be able to do a smooth scroll.

You can also try to use $d020 to set the border-color at different rasterlines, like:

```
wait1   lda $d012       ;Wait for rasterline $30
        cmp #$30
        bne wait1
        lda $d011
        bmi wait1

        lda #1          ;Set border-color to white
        sta $d020

wait2   lda $d012       ;Wait for rasterline $108
        cmp #$08
        bne wait2

        lda #0          ;Set border-color to black
        sta $d020
        jmp wait1       ;And keep looping
```

Notice that the second wait-loop exploits that the next time $d012 will be $08 after rasterline $30, will be at rasterline $108, which makes a check on $d011 unnecessary.

This little program will produce a white stripe in the border of the screen, but the area where the color changes will probably "flicker" a bit, i.e. move randomly around in a small area.

This is because we only check when the rasterLINE is right. Ideally, we would also want to check for the right "raster-coloumn" to be right, so that the cut could be at our specific (x,y) point, but unfortunately the c64 doesn't directly provide such a rastercolumn register, so removing the flicker can be tricky business.

Of course there's much more to it than this. For instance, the VIC provides a facility to automatically announce to the program when a certain rasterline arives, so that we needn't check the rastervalue ourselves in a loop. This technique is known as raster-interrupts, but I'll leave that subject to another time.

Until then, welcome to the world of rasters, and happy hacking!

Asger Alstrup
--
For questions and comments, 'XmikeX' may be reached through the Editor-in-Chief
of disC=overy.

```
/S03::$d000::::::::::::::::::::::::::::::::::::::::::::::::::::::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

```
                           The Raster StarterKit,
  -+-+-+-+-+-+  -+-+-+-+-+-+-+-+   +-+-+-+-+-+-+-+-+-+  -+-+-+-+-+-
                      A classic 'slice' of VIC-II code

                       by XmikeX and Dokken/Electron
```

My early experiences with VIC-II programming on the C64 led me to the conclusion that being able to manipulate the VIC-II on a rasterline level is essential to success as a "coder". Over the years, 64 coders have moved far beyond this simple dogma, but it is recognized that all have to start somewhere. The following is an example of one of my early attempts. I still use this bit of code to form the core of many of my programs. I would like at this time to thank Dokken/Electron for his part in providing the base code that follows. Please note that for the purposes of proper instruction to those who might be more familiar with BASIC than ML, the source below has all values in decimal (same as with BASIC peek/poke). However, please note that raster programming requires quick action and is best left to ML code.

```
*=49152      ; origin at 49152 or whatever free memory location desired

sei
lda #127
sta 56333
lda #18      ; the first 3 STA's just tell the 64 to
sta 53265    ; do IRQ's.  Well, the 53265 hi-byte value
lda #1       ; says to do a IRQ at rasters 0-255, but
sta 53274    ; whatever
lda #100
sta 53266    ; the IRQ'll happen at raster line 100
lda #<main
sta 788      ; low byte of IRQ routine
lda #>main
sta 789      ; high byte of IRQ routine
cli

jkjk
jmp jkjk     ; this just happily jumps back to itself
             ; whenever we're not in an IRQ

main
rol 53273    ; or: lda #1:sta 53273  tell the 64 an IRQ
             ; happened or something goofy like that
ldx #1
jsr rast
ldx #0
jsr rast

lda #126     ; this sets up a raster IRQ at scan line
sta 53266    ; 126 for the 'rain' routine
lda #<rain
sta 788
lda #>rain
sta 789

jmp $ea81    ; ok. you can use either $ea81 or $ea31
             ; $ea31 with an RTS where the 'jmp jkjk'
             ; is allows the 64 to process everything
             ; normally.  ie.  you can run a basic
             ; proggy and have an ML interrupt doing
             ; something on top of it like maybe playing
             ; a happy tune or something.
             ; $ea81 gives you total control with no
             ; overhead.  ($ea31 scans the keyboard
             ; takes a decent chunk of time).  So with
             ; $ea81, if you want to read the keyboard,
             ; you get to do it explicitly.


rain
rol 53273

ldx #6       ; rasterline is now set at color : blue!
jsr rast
ldx #0
jsr rast

lda #100     ; this sets up a raster IRQ at scan line
sta 53266    ; 100 for the 'main' routine
lda #<main
sta 788
lda #>main
sta 789

jmp $ea81


rast      ; this routine just makes a raster bar out of
lda 53266 ; the value in the x register.  Note that this
rast2     ; only works for 7 of 8 scan lines cuz the
cmp 53266 ; 64 needs to chunk away at graphix
beq rast2
stx 53280
stx 53281
rts
```

Try and expand this routine to include many many rasters or color bars, or anything you dream up. It would be prudent to point out at this time that a text such as "Mapping the 64" is extremely useful for delving into VIC-II chip registers. "Mapping the 64" in particular, gives a wonderful description of VIC-II register function.

Goodbye.
--
For more information or general commentary on this article, XmikeX may be reached through the Editor-in-Chief of disC=overy.

/S04::$d000:::
::

Heaven in the net, an unedited excerpt of IRC on #c-64

by Mike Gordillo

As a self-proclaimed "demo freak", the following transcript (largely unedited) represents one of the most interesting discussions concerning C-64 that I have ever witnessed on IRC (Internet Relay Chat). I present this to the reader in the hopes of encouraging further participation and patronage of IRC channel #c-64. We begin with a discussion on raster programming by "Fungus", "_dW", "hld", "Sorex of WOW", and "XmikeX".
--

<[Fungus]> I'd like to know just how to do that. Stable Raster that is. I've tried double interrupts and stuff and every now and then I'd get like this 1-2 cycle jitter I couldn't get rid of.

<_dW> My raster is completely stable.. or so I think.

<Sorex\WOW> fungus: use nops or something to time it out

<[Fungus]> Sorex : I did and still couldn't get it out. The next routine's timing kept changing the original timing.

<_dW> Fungus: it works like this: set up a raster int on line 'n', then in the irq code change the $0314-$0315 vector to point to a different routine and set a raster to strike on line n+1.. execute NOPs 'till it strikes. When the second raster hits, you'll be at most 1 cycle off, which can then be corrected by code like LDA # n+1 : CMP $d012 : BEQ sync (sync: rest of the code)

<[Fungus]> I 've tried that too, and banking out The Kernal etc... Using fffe & ffff. I still got goofy jitter. It would be totally stable for like 5 second then screw 1-2 cycles to the left exactly 3 times. Then would be stable for another 5 secs. CIA's are Disabled as well as NMI's: NTSC.

<_dW> Fungus: you must have left out something... what method did you use?

<[Fungus]> Tried it both ways.

<_dW> double ints and?

<[Fungus]> Didn't matter, double Interupts VIA 0314 0315 Double Int VIA FFFE FFFF too, and yes, all in NTSC R-8 VIC chip so 65 cycles.

<Sorex\WOW> fun:i have a source here from Graham/Oxyron for that 4*4 display routine, i uses multiple IRQ's to get the damn thing steady timed and nops

<_dW> Fungus: send me mail at agonzalez@nlaredo.globalpc.net I'll send you source code for a double int 0314/15 raster sync. Sorex: 4x4 display?

<Sorex\WOW> dw:you know... that mode everyone uses to get fullscreen effect like rotzoomers/plasmas/doomstuff/tunnels/...

<_dW> Sorex: the one with the plasma in dawnfall/oxyron?

<Sorex\WOW> dw:yeah, and the fire in 'The Masque' ....

<[Fungus]> You guys wanna idea? Have any of you tried playing with $d012 like decrementing it 1 cycle after it changes?

< dW> Fungus: that could cause an interrupt, if they're enabled and $d019 lsb

is clear.

<XmikeX> If I could interject a simple question -> Does the VIC-20 have a $d012 equivalent?

<hld> XmikeX - the vic20 has no $d012 equivalent - so you cant tell what raster you are on.

<[Fungus]> Hehehe On the contrary. I code stuff on the Vic-20 too. You can just poll the raster register and SEI and never clear it. Makes it preety easy. It's just that the Vic-20 treats 2 raster lines as one. I made a cool looking 16 color ROL scroller on it once. (Basically, its $d012 equivalent does not generate IRQ's like $d012 does on the C-64).
I prefer to just poll $9004 ($d012) Easy to do. Then you can do all sorts of twisted stuff. The VIC is a different Animal. Really neat registers. Remove the borders with out an INT even!
I'd like to see people get interested in the Vic-20. It's capable of some cool stuff. You can make the screen scroll in any direction really Fast. The Entire screnn too -No borders. Full Charsets and big color Memory. Full Screen High res too, no borders.=]

[...]

<_dW> hld: you did real raytracing in your 4k entry?

<hld> dw - yup.. My 4k entry distorted a checker board. The code was about $4f0 bytes (and 20KB of tables ;)

<_dW> Are you using lots of rom routines? Any division?

<hld> dw - It uses ROM routines to create the 20KB tables, but it is pure integer assembly when running - with very small error rate.
I have no division at all in this tracer.. only +,*,srqt
Remember, pre-calcing all the sqrt's takes quite a while and it is all pre-calced in floating point, but i do some nifty weird things so that it only needs to accumulate integers to do the trace.
Basically, it does one frame in about 4 seconds where a frame is defined as 256x64 monochrome.

[...]

<XmikeX> Wave, how is your plotter routine coming along?

<Waveform> Right now, its "doing" 16*16 character rotation. Going to optimize it a bit and see how far out I can go and still have the routine update reasonably fast, but my line vector is under 4k.

<XmikeX> Any drags so far?

<Waveform> The main thing that slows down the vector routine is the clearing of the next buffer. Man, that EATS time... but a character or bitmap rotation doesn't necessarily need to be cleared first, and even if it does, the amount of zeros to stuff into memory is significantly less.

<_dW> Wave: are you doing a complete clear? or a redraw-clear?

<Waveform> Dw: complete zap of memory...but I toyed with re-draw clear idea.

<MrX_TLO> Hmm, nothing really faster than inlined STA :(

<Waveform> Mr. X : I got a loop like this:

```
lda #$00
ldy #$00
sta $2000,y
sta $2080,y
sta $2100,y
... <etc> ...
iny
bpl <loop>
```

Each STA line will zap 128 bytes in that loop... I may unroll it further, but I don't know.

<MrX_TLO> How about :

```
LDX #0
LDA #times/4
Sorry, LDY #0
STY ....,X
BNE
```

```
       DEX
```

<MrX_TLO> If you INX, you need a CMP or you waste time going over your area.

<Waveform> ah... you are DEXing from times/4

<MrX_TLO> 1 = 256, 2 = 512, 3=768, 4=1024; none of which is the 800 bytes you mentioned.

<Waveform> 2k of zeros; 2k=$800

<MrX_TLO> You should still kill it by pages!

<Waveform> It does kill by pages!

<MrX_TLO> ?? It wasn't clear from the code. How many STA's inside loop?

<Waveform> several

<MrX_TLO> Define several? It should be like 8 STA to kill 2K : base+0, base+256, etc up to base+2K-255

<Waveform> Here is my 'clear' code, Mr. X :

```
       sta $2000,y
       sta $2080,y
       sta $2100,y
       sta $2180,y
       sta $2200,y
       sta $2280,y
       sta $2300,y
       sta $2380,y
       sta $2400,y
       sta $2480,y
       sta $2500,y
       sta $2580,y
       sta $2600,y
       sta $2680,y
       sta $2700,y
       sta $2780,y
       iny
       bpl <loop>
       rts
```

I cover the whole area in one loop, except I kill 128 bytes with each STA.

<MrX_TLO> Wave: But that's way past the point of diminishing returns.... And why pick 128 bytes?

<Waveform> Because it runs faster than 256 bytes!

<MrX_TLO> So you got, what, 16 STA in your actual code?

<Waveform> if each STA wipes 256 bytes, then the loop runs 256 times. Which means that all the loop overhead (agreed - not too much, but there) is multiplied by 256. If you loop 128 times, I would think it should reduce the overhead by half.

<MrX_TLO> Have you actually done the math?

<Waveform> I should time it tonight to be sure.

<MrX_TLO> effective cycles = loop + overhead / total cycles <-- will give an average number for overhead cost.

<Waveform> Well ok, but heck if I'm going to 1:1 unroll it though! 2000 STAs, no way...even if I write a program to generate them.

<MrX_TLO> 4K for your clear routine! hehe :) But 1:1 may lead you past the point of diminishing returns...

<Waveform> Well, I used 128 because there is only a INY (or DEY depending on what you think when you code) and one branch. (BPL or BMI respectively). BTW, 256 has the same functionality... Both will set a processor flag. Hmmm, actually all numbers will with a DEY.

<MrX_TLO> In general, I DEX BNE, just considering it a good habit, always trying by pages.....

<Waveform> But doesn't it stand to reason that a 1:1 is the fastest since it

```
       has no overhead? So if thats true, if you only looped twice, you'd
       cut out half the STA's, but increase the overhead by a factor or 2.
       The overhead is small though (couple cycles for dey, couple for the
       branch.)

<MrX_TLO> Yeah, the overhead is small.  Your first STA is the "best", each
       additional one after that gives less and less improvement in speed and
       just takes more space....

<Waveform> And if you need to zap like 2k of memory, there ain't a Bxx
       instruction in 6510 thats going to get you back to the beginning of
       that STA loop. =)  After all, you can only have 40 or 41 STA's to use
       a Bxx instruction.

<MrX_TLO> Time to unroll your loop, Waveform :).

<Waveform> Then I'd get 6144 bytes of code to clear 2048 bytes of buffer. :(
       I can't believe I am thinking of making a 6144 byte clear routine and
       actually, it would have to be 12288 bytes, since I double buffer.
       Three bytes per STA... 2048 bytes to clear... (2048 * 3 = 6144) * two
       buffers doubles that to 12288.

<Waveform> Hmmm...  Loop overhead... Let us see here...
       Each time you gotta INY/DEY and CMP and/or Bxx (branch) or JMP
       at the end of the loop, with the number of times through the loop
       multiplying the overhead.  The less times through the loop, the less
       overhead.

<MrX_TLO> Yes, and DEX  BNE = 2 + 3 = 5  (assuming your smart enought to line
       it up in one page! otherwise 6 cycles)

<Waveform> Yeah, if I completely unroll it, I'll have NO loop overhead
       to deal with at all...  I will certainly have to give this idea some
       further thought.

[...]

<XmikeX> So, did you take MrX's advice?

<Waveform> Yes, I unrolled that clear loop, but I'm not sure how much savings
       I was able to obtain.  I mean I know how much in cycles, but I don't
       see much of a real difference it made to the overall performance of
       the code.

<XmikeX> Well, there should be a difference.

<Waveform> Yeah, but I'm not seeing THAT much of a diff... hmmm... but it was a
       while inbetween running each version. I should run them one right after
       another and see. So far, to tell the truth, the effect seems neglegible.

<XmikeX> What are your cycle times now?

<Waveform> 10752 for old code - 8192 for new code = 2560 cycles.
       Hmmm, if I am doing my math right, then it is actually 2.5 ms.
       Let's see, NTSC C64 draws a full screen every 1/60, and if one
       raster line is 1/263 of 1/60 sec in NTSC, it takes approx. 0.0000636
       seconds to draw one raster line, and there are 65 cycles per line.
       This means one cycle is more or less .0000009 seconds which
       is approx. NTSC 1Mhz rate.  My calculator lacks the required
       precision here, but enough to give credence to the clock rate. :)
       So if we forget about 'bad lines' (VIC-II DMA), I am saving
       around 2.5 ms per iteration. My code rotates an object around
       its center completely in 8-bit increments, hence the routine is
       called 256 times.

<XmikeX> So 256 * 2.5 ms = 640 ms per 1 complete rotation  (.640 sec! :)

<Waveform> So I lose 12k of RAM for .640 sec..... NOT!  I will rip that code
       out and instead work on speeding up my line drawing code.

<XmikeX> Sigh...plotting...truth tables... blah :)

<Waveform> Hehehe, 0  OR 1 = 1
                  0 AND 1 = 0
                  0 EOR 0 = 0
                  0 EOR 1 = 1
                  1 EOR 1 = 0
       Etc., Etc., Etc...and remember, anytime you got that N in front
       (NAND, NOR, NOT, etc) you just invert the truth table.

<Elwix> Hey gang, wassup?
```

<Waveform> Wix, right now I am looking at a balance of neatness in programming vs speed of execution (plotter)

<Elwix> the age old conflict... speed vs smoothness, how many cycles?

<Waveform> Mine is 70 as it stands.

<Elwix> wave - that's your theta,r plotter right?

<Waveform> Yeah, and the plotter doesn't care where the coords come from, Wix.

<Elwix> yeah, I know, a general point plotter... but where it plots points would change the code a bit, no? the plotter takes r,theta and turns the right pixel on in the chargrid right?

<Waveform> I coded 3 general purpose routines:

```
       1> point rotater (basically, change all the thetas)
       2> Resolve r,theta into x,y
       3> plot x,y on charmap
```

<Elwix> ahhh... ok... I thought you had said that there was no step 2 (conversion of r/theta into x/y) ??? the speed of the steps 1-2-3 are combined at 76 cycles?

<Waveform> no no no, I don't have inital x,y... no need to convert x,y into polar just to rotate it and bring back.

<Elwix> well I udnerstand you start from polar, but I was imagining you had a direct r/theta to bitmap plotter... that's all

<Waveform> yeah... well I have to resolve into X,Y somewhere, Wix. =)

<Elwix> ok, so your step 1-2-3 combined is 76 cycles?

<Waveform> But the point of the routine is that I don't start in X,Y... Which makes the Rotation part very fast... just inc/dec the thetas.

<Waveform> And, no... just the plot into the char map is 70 cycles (as it stands, I think I can get it down to below 60 though)

<Elwix> but, yes, you do convert back to x/y to plot. That's your step 2 right?

<Waveform> Wix, yeah.

<Elwix> Wave well, you could do better on the x/y plotter. But if what you have works & looks good... who cares? At the most I was able to achieve 35 cycles onto a full screen bitmap coming from X & Y, with an X high bit. Think for a moment... :)

<Waveform> Oh wait! I've just had a brainstorm. Be right back, getting a pen.

<Waveform> DUDE!!!!! Woah!!!!! Just a sec... let me recount this.

<Waveform> I -think- I have a 26 cycle plotter now.

<Elwix> 26 should be possible in a right-made chargrid, but not on a bitmap I think...

<Waveform> Perhaps, but I will definately explore this! In fact, once I'm done, I think I'll put this all down in writing. :)

[...end of transcript...]

```
/S05::$d000:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

```
       A demo of 'disC=overy',      =
                                  -+*+-
                                    =       The exploration of rotation :)

                          by John Kaiser (Waveform/MLM)
```

Shoooom! Greetings from the Wave! =)

In this article I am going to talk about a relatively simple two-dimensional vector routine. Before I dive in, however, I'd like to give some credit to Steven Judd, who through one of his C= Hacking articles showed me the way to the core of a fast line drawing routine as well as an amazing method of quick multiplication. I'd also like to give thanks to those 'hard-core coding animals' on IRC channel #c-64 for their most-nifty insight. Before I forget, to all those who I have been taunting with my (I think, I hope..) 'super fast pixel plotter', you are finally able to see the source for it. Its so extremely simple, you'll probably kick yourself for not coming up with it yourself.

[Ed. Note : According to S. Judd, the 'quick multiplication' method which John is referring to was originally formulated by George Taylor.]

For all of those expecting a full blown demo in this article, I am afraid I have to disappoint you. The "real world" has been taking a heavy toll on me during the last month so the amount of time I was able to spend working on this article was extremely limited. What you will see, however, is a fully functional rotating vector routine, in a relatively basic form. There are tons of uses for the routines involved, and infact, my next demo uses quite a few variations on the code you are about to see.

It is interesting to note that I did not set out to code a rotating vector routine. =) I have another project I'm working on that involves some of the same routines. While working I thought, "Hey, I could probably rotate a two dimensional shape pretty easy with this."

And Voila! You get something like the source code that follows.

Preface To The Code

The trick to this code is referencing the endpoints of a line by a radius and theta, as opposed to regular X and Y.

The advantage to referencing the endpoints - or even just individual pixels - like this is that to rotate a point, you merely add or subtract from the theta value. This process ends up being extremely fast, especially when compared to taking an X,Y coordinate and sending it through a whole slew of mathematics just to move it around. Observe a quick comparison of the steps needed to rotate a number of endpoints about their geometric center:

```
   Old Method:
   -----------
   1. Get an X,Y endpoint
   2. Convert to radius and theta (or some other convenient rotation friendly
      reference)
   3. Apply the rotation
   4. Convert back to X,Y
   5. Store result
   6. Loop back until all endpoints are finished

   Nifty Method:
   -------------
   1. Get a radius,theta endpoint
   2. Apply the rotation
   3. Store result
   4. Loop back until all endpoints are finished
```

What is missing is the slow, and sometimes bulky conversion routines. Even if you have a fast conversion routine, this method still is faster because you have less steps from beginning to end. You have less code to execute per endpoint.

One other trick to this particular code is that my circles have 256 units to them, while most peoples circles have 360 degrees. Rather than have a boring name like "units" I decided to use the name "byts". So, by way of comparison, a normal circle has 360 degrees, while my circles have 256 byts.

Why use a different unit of measurement? The thinking man will quickly realize that in our nifty 8bit machines, the largest value a single byte - register, or memory location - can hold is 255. By using the byt unit of measurement, we can make an entire circle with just one byte, and not have to worry about fiddling with high bits. This way of thinking saves many many processor cycles, and many many hairs on a coder's head. =)

Finally, before I slam you with the source code (which, by the way, is in TurboAssembler format) there is one more trick to this particular code. Sometimes the only way to get real speed out of a Commodore is to reduce the code necessary to perform calculations. How do the great coders do that? The keyword(s) are LOOKUP TABLES.

This code uses tables for the plot routine and for the multiplication routine. I wrote basic programs to create the tables and will do my best to explain how they work, near the end of this article. As to the tables themselves, I'll explain how the code uses them when I get to the relevant section of the source code.

Wow, what a preface! Now, on to bigger and better things: The documented and commented Source Code!

The Source Code - In TurboAssembler Format

Okay, here it is. I'll be commenting on the code throughout the remainder of the article in hopes to further clarify what the code is doing.

First, no code is complete without a little self glorification before the origin and equates are setup...

```
                    ;---------------------------------------
                    ;  A Demo of disC=overy - By: Waveform
                    ;
                    ;  (c) 1996 for disC=overy Magazine
                    ;"The Journal of the Commodore Enthusiast"
                    ;---------------------------------------
```

The afore mentioned origin and equates...

```
                              *= $0820

                    points     = $033c   ;number of points
                    angleinc   = $033d   ;angle to increment
                    curshape   = $033e   ;current shape
                    theta      = $0340   ;thetas
                    radiu      = $0350   ;radius
                    itheta     = $0360   ;tab of init thetas
                    iradiu     = $0370   ;        init radius

                    scrloc     = 1196    ;location of top
                                         ; left corner of
                                         ; char matrix on
                                         ; screen

                    msin       = $3000   ;table of sines
                    mcos       = $3100   ;          cosines
                    msqr       = $3200   ;          squares
                    lobyte     = $3300   ;lo bytes
                    hibyte0    = $3380   ;hi bytes ($2000+)
                    hibyte1    = $3400   ;         ($2800+)
                    bitmask    = $3480   ;pixel bit mask
```

Okay: "points" is the number of endpoints for the current displayed shape.

"angleinc" is the amount to rotate the shape, in byts.

"curshape" is the number of the current shape. This demonstration has three predefined shapes. Feel free to experiment with the ones I included or add your own!

"theta" is a table of thetas. This code is written such that a shape can have up to 16 endpoints defined. To change this "limitation" simply give yourself more bytes in-between these tables.

"radiu" is a table of radii. Both "theta" and "radiu" change as the program executes. At any point in time, these two tables hold the current values of all the endpoints for the current shape.

"itheta" is a table of the initial values of the shapes theta coordinate component.

"iradiu" is a table of the initial values of the shapes radius coordinate component. Both "itheta" and "iradiu" tables are static and are not changed by the program. It is convenient to have these two tables for when you need to alter the coordinates of a endpoint relative to its initial position.

"scrloc" is used by the routine that places the character matrix on the screen. 1196 places it in the center of the screen.

"msin" is the base address of the table of sines. Each value represents the sine of the address, relative to the base. So that msin+0 = sine of 0 byts, msin+64 = sine of 64 byts, etc.

"mcos" is the base address of the table of cosines. Works the same way as the sine table.

"msqr" is the base address for the table of squares. Works similar to the sine and cosine table, except that this table is a table of squares of both positive and negative numbers. More on this later.

"lobyte" is the base address of a table of low bytes used by the plot routine as the low byte of the address where the pixel to be plotted lives.

"hibyte0" is the base address of a table of high bytes used by the plot routine as the high byte in buffer #0 where the pixel to be plotted lives.

"hibyte1" high byte of the address in buffer #1 where the pixel to be plotted lives.

"bitmask" is the base address of a table of bit masks applied to the byte where the pixel to be plotted lives.

```
                    ;---------------------------------------
                    jsr demoinit   ;init for demo
                    ;---------------------------------------
```

That's just a JSR to the demo initialization routine...

```
                    main       lda #%11111110 ;scan matrix
                               sta $dc00      ;
                               lda $dc01      ;

                               cmp #%11101111 ;f1
                               beq keyf1
                               cmp #%11011111 ;f3
                               beq keyf3
                               cmp #%10111111 ;f5
                               beq keyf5
                               cmp #%11110111 ;f7
                               beq keyf7
                               cmp #%11111101 ;return
                               beq keyreturn

                               lda #%01111111 ;scan matrix
                               sta $dc00      ;
                               lda $dc01      ;

                               cmp #%11101111 ;space
                               beq keyspace
                               cmp #%01111111 ;return
                               beq keystop
```

What the above does is scan the keyboard matrix to see if the user has pressed one of the keys our program is looking for. Later on you'll see that we disabled the CIA interrupts that cause the computer to scan the keyboard, so we needed a way to tell if one of our "do-something" keys was pressed. We "scan" twice since the various keys we are looking for are on different rows of the keyboard matrix.

```
                    reentry    jsr rotate     ;rotate shape
                               jsr drawshape  ;draw shape

                               jmp main       ;loop
                    ;---------------------------------------
```

The little code snippet above is where the JSRs to the real workhorses are. As you can see this demo is pretty simple. =) We rotate the current shape, and then we draw the shape. Then we go back to the beginning and see if the user wants to do something.

```
                    keyf1      jmp rotup
                    keyf3      jmp rotdn
                    keyf5      jmp expup
                    keyf7      jmp expdn
                    keyreturn  jmp newshape
                    keyspace   jmp stoprot
                    keystop    jmp stopdemo
                    ;---------------------------------------
```

Above is a little jump table to the various little routines that do what needs to be done when a key is pressed...

```
                    rotup      inc angleinc   ;increase
                               jmp reentry
                    ;---------------------------------------
                    rotdn      dec angleinc   ;decrease
                               jmp reentry
                    ;---------------------------------------
```

The above two "routines" change the value by which the shape rotates per run-though of the main loop. As you can see, referencing the endpoints as radius and theta takes alot of the work out of rotating a shape about its center! Each time the user presses the F1 key, the angle that the shape rotates per run-though increases. Holding down the F1 key will cause the shape to spin up quite rapidly. Indeed, if you continue to hold down the F1 key, the speed of the rotation will appear to increase to the point where it begins slowing down again.

Obviously, F1 increases the angle of rotation, and F3 will decrease it.

```
expup      ldy #$00
expup1     lda radiu,y
           clc
           adc #$01       ;add one
           cmp #63        ;at maximum?
           bcc expup2
           lda #63
expup2     sta radiu,y    ;store
           iny
           cpy points     ;do all points
           bne expup1

           jmp reentry    ;return
;----------------------------------------
expdn      ldy #$00
expdn1     lda radiu,y
           sec
           sbc #$01       ;subtract one
           cmp #2         ;at minimum?
           bcs expdn2
           lda #2
expdn2     sta radiu,y    ;store
           iny
           cpy points     ;do all points
           bne expdn1

           jmp reentry    ;return
;----------------------------------------
```

What the above two routines do is expand or shrink the shape. These two routines work well because of the way our shapes are defined. The shapes that come with this demo are all equidistant from the objects center. So, it is easy to change the shapes size by altering all of the shapes radius coordinate components. Obviously, more detailed shapes that have endpoints that are at varying radii from the center will need improved routines if the shape is to maintain its proportions through the expansion and shrinking.

Notice the error checking, also. Bad things happen when are radii get too large or too small for our line drawing routine to handle properly.

```
newshape   inc curshape
           lda curshape
           cmp #3         ;last shape?
           bne newshape1  ;nope
           lda #$00       ;select first
           sta curshape   ; shape

newshape1  cmp #2         ;8 point star?
           bne newshape2
           jsr initobj20  ;init shape
           jmp reentry
newshape2  cmp #1         ;2 triangles?
           bne newshape3
           jsr initobj10  ;init shape
           jmp reentry
newshape3  jsr initobj00  ;1 triangle

           jmp reentry
;----------------------------------------
```

The above routine handles the users request to change the displayed shape. There are better ways to handle this, but since this demo only has three shapes built in to it, the quick and dirty way suffices without being extremely bulky.

```
stoprot    lda #$00       ;stop all
           sta angleinc   ; rotation
           jmp reentry    ;
;----------------------------------------
```

The above little bit of code simply stops the rotation of the current shape. It merely stores a zero into "angleinc" which controls the amount of rotation applied to the shape.

```
                    stopdemo   lda #$81       ;reset cia
                               sta $dc0d      ; interrupts

                               jmp $fe66      ;exit via
                                              ; kernal warm
                                              ; start
                    ;---------------------------------------
```

The above code resets the CIA to its default setting so that when our demo exits back out to basic, the user is able to type something. =) It then exits via the Kernal warm start vector to reset other important things like the VIC chip. =)

```
                    rotate     ldy #$00
                    rotate1    lda theta,y    ;get theta
                               clc
                               adc angleinc   ;add amount
                               sta theta,y    ;store theta
                               iny
                               cpy points     ;do all points
                               bne rotate1

                               rts
                    ;---------------------------------------
```

Okay, there it is. The code that actually rotates the shape. It takes the current value of the theta coordinate for each endpoint and adds the value of "angleinc" to it. Extremely simple and quite fast too. =) You can speed it up a bit by unrolling this loop, but for this demo, there aren't enough endpoints to work through to get that much of a savings.

```
                    drawshape  ldy #$00       ;first point

                               lda theta,y    ;pass theta &
                               sta gxytheta   ; radius to
                               lda radiu,y    ; getxy for
                               sta gxyradius  ; conversion
                               jsr getxy      ;

                               lda xpos       ;convert polar
                               clc            ; to
                               adc #64        ; Cartesian
                               sta x1         ;
                               lda ypos       ;
                               clc            ;
                               adc #64        ;
                               sta y1         ;
```

The above code starts off the shape drawing routine. It gets the first endpoint prior to entering the loop below. The loop below joins endpoint to endpoint with a line.

You will notice that we do actually convert from the radius,theta system to the X,Y system, finally. This is necessary because I haven't yet written a routine that will draw lines only from radius and theta. =) Also notice that it adds 64 to both the X and Y coordinate. This is to center the shape in our character matrix. The routine that converts from radius,theta to X,Y returns numbers ranging from -63 to +63, so adding 64 also normalizes our coordinates to make them easier to plot in our character matrix. (It makes the range of possible values equal to 0 to +127)

```
                               ldy #$01       ;second point
                    ds2        lda theta,y
                               sta gxytheta
                               lda radiu,y
                               sta gxyradius
                               jsr getxy
                               lda xpos
                               clc
                               adc #64
                               sta x2
                               lda ypos
                               clc
                               adc #64
                               sta y2
                               sty dsy        ;preserve .y
                               jsr drawline   ;draw a line
                               ldy dsy        ;restore .y
                               lda x2         ;make endpoint
                               sta x1         ; of this line
                               lda y2         ; start point
                               sta y1         ; of next line

                               iny            ;do all points
                               cpy points     ;
                               bne ds2        ;
```

That's the routine that puts the shape on the screen. You may of noticed that each shape has one extra endpoint. For example, the triangle has four endpoints. This is to allow the shape drawing routine a place to end the last line. We of course, want to make the last endpoint equal to the first endpoint. This causes the routine to complete the shape.

```
                               lda $d018      ;show our work
                               eor #%00000010 ;
                               sta $d018      ;
```

Once we have finished drawing the shape in the buffer, we tell the VIC to display the buffer. This technique is known as double buffering. We display one buffer, while we do all our work in the other. When we are done with the work, we display that buffer, and do all our work in the first.

A nifty trick in situations like these is to use the VIC to our advantage. Since we are using a character matrix to do our drawing in, we make both of our buffers in sequential character matrix slots in memory. Then to swap the displayed buffers, we merely toggle a bit in the VIC register that tells the VIC which matrix to display. =)

A little confused? Well hopefully this will clear things up, at least a little. We start our program with the VIC pointing at buffer#0 which is at $2000. Bits 3,2,and 1 of $d018 control which address the VIC finds the character matrix. Here is a nifty table showing the addresses which correlate to those bits in $d018:

$d018	points to	
%xxxx000x	$0000	
%xxxx001x	$0800	
%xxxx010x	$1000	
%xxxx011x	$1800	
%xxxx100x	$2000	-> This is our Buffer#0
%xxxx101x	$2800	-> This is our Buffer#1
%xxxx110x	$3000	
%xxxx111x	$3800	

You can see how we can make the VIC do all the work of displaying the right buffer. All we have to do is toggle bit 1 of $d018 to make the VIC switch between the character matrix at $2000 and the one at $2800.

```
                               lda $d018      ;clear the
                               and #%00000010 ; next buffer
                               beq ds4        ; for drawing
                               jsr blank0     ; in
                               jmp ds5        ;
                    ds4        jsr blank1     ;
```

This code checks which buffer is currently being displayed and the JSRs to the routine to clear the OTHER one, in preparation for drawing in.

```
                    ds5        rts

                    dsy        .byte $00
                    ;----------------------------------------
```

Finish up... "dsy" is a temporary storage location.

```
                    getxy      sty gxyy       ;preserve .y
                               sta gxya       ;         .a

                               ldy gxytheta   ;
                               lda mcos,y     ;a
                               sec            ;
                               sbc gxyradius  ;-b
                               tay            ;
                               lda msqr,y     ;f(a-b)
                               sta gxytemp    ;store result
```

```
                                ldy gxytheta   ;
                                lda mcos,y     ;a
                                clc            ;
                                adc gxyradius  ;+b
                                tay            ;
                                lda msqr,y     ;f(a+b)
                                sec            ;
                                sbc gxytemp    ;-f(a-b)
                                sta xpos       ;=x coordinate
```

Ah, now you get to see the fast multiplication in action. This routine is based heavily on the routine outlined by Steven Judd in C=Hacking. He has already documented fairly well how the table of squares work, so I'll not re-invent the wheel by explaining precisely how it works. What I will do is step you through the steps of what this code does.

Okay, know that you can arrive at A*B with a function like:

```
   f(A+B) - f(A-B) when f(x) = (x^2)/4.
```

I created a table of squares such that the offset from the beginning of the table was the (x) in the f(x) above. For example, location $3300 + 0 = 0, since obviously (0^2)/4 is still 0. By the same token, $3300 + 9 = 20, since (9^2)/4 = 20.25.

In our case, the radius component is the (A) and the (cos (theta)) component is the (B). Those who didn't sleep through trigonometry class will recall that when converting from radius,theta into X,Y, your X coordinate is R * cos(theta). Hence, our A*B routine.

Those that have read ahead, will note that I made some adjustments to the tables to the effect that the sin and cos tables are multiplied by 64, and that our table of squares is really the result of a function where f(x) = (x^2)/4*64. The reason for this is to maintain some level of accuracy in our tables. If you don't understand why this was done, I'll explain it further when I detail the basic programs that build our tables.

```
                                ldy gxytheta   ;
                                lda msin,y     ;a
                                sec            ;
                                sbc gxyradius  ;-b
                                tay            ;
                                lda msqr,y     ;f(a-b)
                                sta gxytemp    ;store result

                                ldy gxytheta   ;
                                lda msin,y     ;a
                                clc            ;
                                adc gxyradius  ;+b
                                tay            ;
                                lda msqr,y     ;f(a+b)
                                sec            ;
                                sbc gxytemp    ;-f(a-b)
                                sta ypos       ;=y coordinate

                                ldy gxyy       ;restore .y
                                lda gxya       ;restore .a

                                rts
                                ;-----------------------------

                      gxyradius  .byte $00
                      gxytheta   .byte $00

                      xpos       .byte $00
                      ypos       .byte $00

                      gxyy       .byte $00
                      gxya       .byte $00
                      gxytemp    .byte $00
                      ;---------------------------------------
```

The code above does exactly the same thing the last piece of code did, except this time we were calculating the Y coordinate, and of course, to be trigonomically correct, our multiplication performs this equation:

```
   Y = R * sin(theta).
```

The labels at the end are for temporary storage of values during the multiplication routine.

```
                    plot       pha            ;preserve .a

                               lda $d018      ;plot in
                               and #%00000010 ; correct
                               bne plot2      ; buffer

                    plot1      lda lobyte,x   ;lo byte
                               sta $02        ;
                               lda hibyte1,x  ;hi byte
                               sta $03        ;
                               lda ($02),y    ;
                               ora bitmask,x  ;turn pixel on
                               sta ($02),y    ;

                               pla            ;restore .a
                               rts
                    ;---------------------------------------
                    plot2      lda lobyte,x   ;lo byte
                               sta $02        ;
                               lda hibyte0,x  ;hi byte
                               sta $03        ;
                               lda ($02),y    ;
                               ora bitmask,x  ;turn pixel on
                               sta ($02),y    ;

                               pla            ;restore .a
                               rts
                    ;---------------------------------------
```

Shoooom! There it is! If you blinked you may of missed it. The plot routine is quite small, and its even smaller if your program doesn't need to decide which buffer it needs to plot in.

In detail, first it checks which buffer we are working in and branches to the appropriate plot routine.

How it works: the X register holds our X coordinate, and the Y register holds our Y coordinate. Both our X and Y registers hold a value between 0 and 127. Knowing that, we can quickly and easily use some cleverly thought out planning ahead to make plotting a pixel super fast and very clean.

Things we planned for ahead of time: We drew our character grid on the screen such that the addresses that define the character data are laid out to our advantage. Then we made a couple tables with data that makes it extremely simple to look up the address of the byte where we want to plot a pixel.

Sound a little vague? I'll explain in greater detail when we get to the basic programs that set up the tables and also the little routine that lays out our character grid.

First, it looks up an address based on our X coordinate. Each byte has 8 pixels so, our table looks like this:

```
   $3300 $00 $00 $00 $00 $00 $00 $00 $00
   $3308 $80 $80 $80 $80 $80 $80 $80 $80
   $3310 $00 $00 $00 $00 $00 $00 $00 $00
   $3318 $80 $80 $80 $80 $80 $80 $80 $80
     ...
```

Notice that the values change every eight locations? Each byte has eight pixels. For example, imagine that the X coordinate is 0. We get the first (0th) byte from our low byte table and see its a zero. Its easy to see that we would always be plotting in the first (0th) byte until X is greater than 7 (i.e.: 8) in which case the value in our low byte table is $80. $80 is 128 bytes over from our first column of bytes, and holds the byte that has pixels 8-15 in it. If you are still confused, I will explain this further when we get to the routine that builds our character grid on the screen.

Now that we have the low byte, we need a high byte. Again we look into a cleverly planned table using X as our index. The high byte table looks alot like the low byte table except that we change values every sixteen bytes. Take a look:

```
   $3380 $20 $20 $20 $20 $20 $20 $20 $20
   $3388 $20 $20 $20 $20 $20 $20 $20 $20
   $3390 $21 #21 $21 $21 $21 $21 $21 $21
   $3398 $21 #21 $21 $21 $21 $21 $21 $21
     ...
```

This is because each column only has 128 pixels in it. If the X coordinate is 0-7, then the byte that holds our pixel is at $2000. When the X coordinate is 8-15, the byte is at $2080. Only when X is 16-23 does our high byte change from $20 to, in this example, $21, since the bye we are looking for would be $2100.

Now, we got an address based on the value of X. But what about Y? We aren't always going to be plotting in the first top row of pixels! Since we were clever in how we laid out our character grid, Y becomes an offset from our address. We figured our how far to go across in bytes from a table, but we don't need a table to figure out how far down to go. We simply address the byte using indexed addressing. =) So, for example if our Y coordinate was 5, the instruction LDA ($02),Y would get the 5th byte down from the byte we arrived at earlier.

Nifty eh?

After we get the byte, we need to turn on a pixel. This is where the bitmask table comes in. Our bitmask table looks like this:

```
   $3480 $80 $40 $20 $10 $08 $04 $02 $01
   $3488 $80 $40 $20 $10 $08 $04 $02 $01
   $3490 $80 $40 $20 $10 $08 $04 $02 $01
   $3498 $80 $40 $20 $10 $08 $04 $02 $01
     ...
```

It should be obvious how this table works. This table allows us to arrive at the right pixel to turn on (via ORA) when we plot. Notice that it repeats every eight bytes, and notice that there are eight pixels in a byte. =)

```
                    drawline   sec            ;get dx
                               lda x1         ;
                               sbc x2         ;
                               sta dx         ;

                               sec            ;get dy
                               lda y1         ;
                               sbc y2         ;
                               sta dy         ;

                               clc            ;
                               lda dx         ;
                               bpl drawline2  ;handle -dx
                               eor #%11111111 ;make dx
                               adc #%00000001 ; positive
                               sta dx         ;

                               lda dy         ;
                               bpl drawline1  ;handle -dy
                               eor #%11111111 ;make dy
                               adc #%00000001 ; positive
                               sta dy         ;

                               lda dx
                               cmp dy
                               bcs dl00
                               jmp dl10

                    drawline1  lda dx
                               cmp dy
                               bcs dl20
                               jmp dl30

                    drawline2  lda dy
                               bpl drawline3
                               eor #%11111111
                               adc #%00000001
                               sta dy

                               lda dx
                               cmp dy
                               bcs drawline4   ;dl40
                               jmp dl50

                    drawline3  lda dx
                               cmp dy
                               bcs drawline5   ;dl50
                               jmp dl70
                    el         rts
                    drawline4  jmp dl40
                    drawline5  jmp dl60
                    ;---------------------------------------
```

The above code determines the slope of the line to be drawn. This is the part of my code that I *know* in my gut I can improve on, but as of yet have been unable to. There are eight separate little routines below that draw a line. Each one is slightly different, they are written to handle each of the eight possible types of line slopes that can be encountered when drawing a line between two points.

The values DX and DY are calculated such that DX=X2-X1, and DY=Y2-Y1.

```
                    dl00       lda #$00         ;0 - dx
                               sbc dx           ;

                               ldx x1           ;plot first
                               ldy y1           ; pixel
                               jsr plot         ;

                    dl00a      clc              ;step in x
                               inx              ; positive
                               adc dy           ; until time
                               bcc dl00b        ; to take a
                               iny              ; positive
                               sbc dx           ; step in y
                    dl00b      jsr plot         ;
                               cpx x2           ;
                               bne dl00a        ;

                               rts
                               ;----------------------------
```

All eight of these routines are basically the same with only two real differences:

1) The value which is being stepped through until it's time to step the
   other value.
2) The direction of the stepping

Here is a step-by-step description of what the above does:

1. Subtract DX (change in X) from zero
2. Plot the first pixel
3. Step upwards in values of X, until it is time to take a step in Y
4. Plot the pixel
5. Loop back until we've reached the second endpoint

Each of the following routines works exactly the same way.

```
                    dl10       lda #$00
                               sbc dy

                               ldx x1
                               ldy y1
                               jsr plot

                    dl10a      clc              ;step +y
                               iny              ; until need
                               adc dx           ; to +x
                               bcc dl10b        ;
                               inx              ;
                               sbc dy           ;
                    dl10b      jsr plot         ;
                               cpy y2           ;
                               bne dl10a        ;

                               rts
                              ;----------------------------
                    dl20       lda #$00
                               sbc dx

                               ldx x1
                               ldy y1
                               jsr plot

                    dl20a      clc              ;step +x
                               inx              ; until need
                               adc dy           ; to -y
                               bcc dl20b        ;
                               dey              ;
```

```
           sbc dx          ;
dl20b      jsr plot        ;
           cpx x2          ;
           bne dl20a       ;

           rts
           ;-----------------------------
dl30       lda #$00
           sbc dy

           ldx x1
           ldy y1
           jsr plot

dl30a      clc             ;step -y
           dey             ; until need
           adc dx          ; to +x
           bcc dl30b       ;
           inx             ;
           sbc dy          ;
dl30b      jsr plot        ;
           cpy y2          ;
           bne dl30a       ;

           rts
           ;-----------------------------
dl40       lda #$00
           sbc dx

           ldx x1
           ldy y1
           jsr plot

dl40a      clc             ;step -x
           dex             ; until need
           adc dy          ; to +y
           bcc dl40b       ;
           iny             ;
           sbc dx          ;
dl40b      jsr plot        ;
           cpx x2          ;
           bne dl40a       ;

           rts
           ;-----------------------------
dl50       lda #$00
           sbc dy

           ldx x1
           ldy y1
           jsr plot

dl50a      clc             ;step +y
           iny             ; until need
           adc dx          ; to -x
           bcc dl50b       ;
           dex             ;
           sbc dy          ;
dl50b      jsr plot        ;
           cpy y2          ;
           bne dl50a       ;

           rts
           ;-----------------------------
dl60       lda #$00
           sbc dx

           ldx x1
           ldy y1
           jsr plot

dl60a      clc             ;step -x
           dex             ; until need
           adc dy          ; to -y
           bcc dl60b       ;
           dey             ;
           sbc dx          ;
dl60b      jsr plot        ;
           cpx x2          ;
           bne dl60a       ;
```

```
                               rts
                               ;-----------------------------
                    dl70       lda #$00
                               sbc dy

                               ldx x1
                               ldy y1
                               jsr plot

                    dl70a      clc             ;step -y
                               dey             ; until need
                               adc dx          ; to -x
                               bcc dl70b       ;
                               dex             ;
                               sbc dy          ;
                    dl70b      jsr plot        ;
                               cpy y2          ;
                               bne dl70a       ;

                               rts
                    ;----------------------------------------
                    x1         .byte $00
                    y1         .byte $00
                    x2         .byte $00
                    y2         .byte $00
                    dx         .byte $00
                    dy         .byte $00
                    ;----------------------------------------
```

The values above are temporary holding places for the values used and generated by the drawline routines.

```
                    demoinit   lda #$06        ;dark blue
                               sta $d020       ; background
                               sta $d021       ; and border

                               lda #147        ;clear screen
                               jsr $ffd2       ;

                               lda #$7f        ;disable cia
                               sta $dc0d       ; time irqs

                               jsr blank0      ;clear both
                               jsr blank1      ; buffers

                               lda #$00
                               sta char

                               ldy #$00        ;dirty way to
                    demoinit4  lda #<scrloc    ; draw a
                               sta $fa         ; character
                               lda #>scrloc    ; matrix on
                               sta $fb         ; the screen
                               ldx #$00        ;
                    demoinit5  lda char        ; the top
                               sta ($fa),y     ; left corner
                               inc char        ; is the
                               lda $fa         ; value in
                               clc             ; scrloc
                               adc #40         ;
                               sta $fa         ;
                               bcc demoinit6   ;
                               inc $fb         ;
                    demoinit6  inx             ;
                               cpx #16         ;
                               bne demoinit5   ;
                               iny             ;
                               cpy #16         ;
                               bne demoinit4   ;

                               ldy #$00        ;fill color
                               lda #$01        ; memory with
                    demoinit10 sta $d800,y     ; white
                               sta $d900,y     ;
                               sta $da00,y     ;
                               sta $db00,y     ;
                               iny             ;
                               bne demoinit10  ;

                               lda $d018       ;point VIC to
                               and #%11110000  ; first
                               ora #%00001000  ; buffer
                               sta $d018       ;

                               jsr initobj00   ;1st shape

                               lda #$00        ;init
                               sta angleinc    ; variables
                               sta curshape    ;

                               rts

                    char       .byte $00
                    ;---------------------------------------
```

The above initialization routine sets everything up for the demo. In order of appearance, here is an explanation of what each part of this initialization does.

1. Turn the border and background dark blue.
2. Clear the screen
3. Disable the CIA timer IRQs
4. Clear both of the drawing buffers
5. Draw our character grid on the screen
6. Fill Color RAM with WHITE
7. Point the VIC to the first buffer (Buffer#0)
8. Initialize the first shape
9. Initialize the "angleinc" and "curshape" variables.

Drawing the character grid on the screen could possibly of been done a little better. But as it says in the comments, its quick and dirty, but it does the job. =)

Using our brains a bit, we draw the character grid on the screen in such a way that starting with zero (the @ sign) we place sequential characters on the screen in columns, 16 down, by 16 across. This enables us to use a very nifty and fast plotter as described above. Laying out our character grid in this manner, gives us 128 sequential bytes in the first column, 128 sequential bytes in the second, and so on.

You can make character grids of any size (up to 16*16) in this manner very easily. A smaller grid frees up characters to be used for other things, like say a niftycool border or other graphics to be used along side of the character grid where your vectors are being drawn. You just have to adjust your low byte and high byte tables accordingly.

```
                    blank0     ldy #$00
                               lda #$00
                    blank0a    sta $2000,y
                               sta $2080,y
                               sta $2100,y
                               sta $2180,y
                               sta $2200,y
                               sta $2280,y
                               sta $2300,y
                               sta $2380,y
                               sta $2400,y
                               sta $2480,y
                               sta $2500,y
                               sta $2580,y
                               sta $2600,y
                               sta $2680,y
                               sta $2700,y
                               sta $2780,y
                               iny
                               bpl blank0a
                               rts
                    ;---------------------------------------
```

Clears the first buffer (buffer#0). Unfortunately, these clear routines are real cycle hogs. There just aren't many fast ways to zero out 4k of memory.

```
                    blank1     ldy #$00
                               lda #$00
                    blank1a    sta $2800,y
                               sta $2880,y
                               sta $2900,y
                               sta $2980,y
                               sta $2a00,y
                               sta $2a80,y
                               sta $2b00,y
```

```
                     sta $2b80,y
                     sta $2c00,y
                     sta $2c80,y
                     sta $2d00,y
                     sta $2d80,y
                     sta $2e00,y
                     sta $2e80,y
                     sta $2f00,y
                     sta $2f80,y
                     iny
                     bpl blank1a
                     rts
           ;------------------------------------
```

The above clears the second buffer (buffer#1)...

```
           initobj00  lda #$04
                      sta points

                      ldy #$00
           initobj01  lda theta00,y
                      sta itheta,y
                      sta theta,y
                      lda radiu00,y
                      sta iradiu,y
                      sta radiu,y
                      iny
                      cpy points
                      bne initobj01
                      rts
           ;------------------------------------
           initobj10  lda #$05
                      sta points

                      ldy #$00
           initobj11  lda theta10,y
                      sta itheta,y
                      sta theta,y
                      lda radiu10,y
                      sta iradiu,y
                      sta radiu,y
                      iny
                      cpy points
                      bne initobj11
                      rts
           ;------------------------------------
           initobj20  lda #$09
                      sta points

                      ldy #$00
           initobj21  lda theta20,y
                      sta itheta,y
                      sta theta,y
                      lda radiu20,y
                      sta iradiu,y
                      sta radiu,y
                      iny
                      cpy points
                      bne initobj21
                      rts
           ;------------------------------------
           theta00    .byte 0,86,172,0
           radiu00    .byte 40,40,40,40

           theta10    .byte 0,64,192,128,0
           radiu10    .byte 40,40,40,40,40

           theta20    .byte 0,96,192,32,128
                      .byte 224,64,160,0
           radiu20    .byte 40,40,40,40,40
                      .byte 40,40,40,40
```

The above short routines are the three that define the three shapes I included in this demo. The initialization for a shape is very simple. You store the number of endpoints in the "points" variable, and then copy the information from the appropriate table into the "theta" and "radiu" tables. I also copy the values into the "itheta" and "iradiu" tables in case there is a routine that needs to reference the initial values rather than the current values.

Also note, that as mentioned before, the last end point is the same as the first endpoint. This closes off the shape by drawing a line back to the first endpoint.

Well, there you have it! Documented source code for a demonstration of a different way to play with two-dimensional vectors. What follows are the basic programs used to create the data tables used by the above program, and of course, the uuencoded programs themselves.

The BASIC Programs

The first program is the program which builds the sine, cosine and squares tables.

```
1 rem -+-   make sin/cos/sqr 2.0   -+-
2 rem -+-                          -+-
3 rem -+-  by: waveform            -+-
4 rem -+- for: disC=overy magazine -+-
5 rem -+-  on: 09-14-96            -+-

10 rem ::: make sin and cos tables :::
12 ba=12288:bh=int(ba/256):bl=ba-bh*256
```

"ba" is the starting address for the data created by this program.

```
14 forby=0to255
16 de=by*1.407:ra=de*(pi/180)
18 s=int((sin(ra)*64)+.5)
20 ifs<0thens=255-abs(s)+1
22 c=int((cos(ra)*64)+.5)
24 ifc<0thenc=255-abs(c)+1
26 poke ba + by,s:poke ba + 256 + by,c
28 next
```

The above loop makes both the sine table and the cosine table. Also please note that in place of the word "pi" in line 16 above, you should instead use Commodore Basic's pi symbol.

Step-by-step, the loop does this:

1. Convert from byts to degrees.
2. Convert from degrees to radians. This is necessary since Commodore's Basic trigonomic functions work with radians.
3. Calculate the sine of the angle.
4. If the sine is less than 0 (i.e.. negative) adjust the number.
5. Calculate the cosine of the angle
6. If the cosine is negative, adjust the number.
7. Store both the sine and cosine in their tables.
8. Loop back until all 256 byts have been calculated and stored in a table.

When we calculate the sine or cosine of an angle, it becomes quickly evident that except when the result is 0 or 1, the result is always a decimal number. Our Commodores don't have a quick and easy way to store a decimal number, certainly not in one byte.

Rather than mess with program code to deal with decimal numbers, we can resolve this issue rather quickly by some planning ahead here when we create our tables.

If we take a decimal number, for instance .707107 (the sine of 45 degrees, or 32 byts) and multiply that number by a constant, in our case 64, we can arrive at a number which is much easier for our computers to store: 45.25. The fractional part of the number is chopped off when we store the value to memory, but what remains is a value that we can work with.

Incidentally, we add the .5 to the value to reduce the rounding errors caused when the fractional portion of the number is chopped off.

What is this about "adjusting" the number, anyway? Well, we know how to store a decimal number now, but what about a number that's negative?

One of the nifty things about the math instructions on a Commodore is that they work the same for both unsigned and signed arithmetic. But to store a negative number you have to use twos compliment. To arrive at the twos compliment (negative) number, you flip all of the bits, and add one. Commodore basic doesn't have an EOR instruction, so we do the next best thing: subtract the number from 255. Subtracting a number from 255 has the same effect as EORing a number by %11111111, which would, of course, flip all of the bits. We then add one, and POOF, the twos complement of a number.

The niftiness about twos compliment and signed values in assembly, is that it gives us a way to represent negative numbers. When you ADD or SBC with these numbers, the instructions work just as they did before, but the perform the task you'd expect by using a negative number.

Steven Judd talks a bit about this in the same articles for C=Hacking that he wrote detailing how the table of squares works for fast multiplication. Also, you can read more about this in virtually any book or reference work that talks about the 65xx line of microprocessors.

Having mentioned that, how do we arrive at the right answer later on if one of our values has been multiplied by a factor of 64? Read on...

```
                    50 rem ::: make sqr table :::
                    52 forby=0to127
                    54 sq=(by*by)/(4*64)
                    56 poke ba + 512 +by,sq:next
                    60 forby=0to127
                    62 sq=(by*by)/(4*64)
                    64 poke ba + 512 +255-by,sq:next
```

The above loop creates a table of squares. I did a little planning ahead as well, and realized that the table need only be 128 bytes long. Recall our niftycool formula for fast multiplication: f(a+b) - f(a-b) = a*b.

When we do the first part of the formula (f(a+b)) you can see from how our program works that the radius is never larger than 64, and at the most, the value from our sine table will be is 64. Remember from trig class that sines and cosines range from 0 to 1. =) Since we multiplied our sine and cosine values by 64, the largest value we could ever get is 64.

Well, you say, but since the values you pull from your sine and cosine table can be negative, what happens if the result of A+B or A-B is negative? The result of a squaring will always be positive, so we don't have to store any numbers using twos compliment, but we will end up with situations where the sum or the difference of A and B will be negative. We resolve this by building a second table of squares above the first, and we build it downward in memory.

Hopefully, this example will clear up any confusion:

If A (our radius) is 20, and the sine of our angle is -40, the sum of A and B is -20. The value in the register is 236, which means -20 in two complement. Since we built another table above the first 128 byte table of squares, we are free and clear. Our foresight put the correct value for f(-20) in that location.

Now, there is one more issue to deal with. We multiplied all of our sines and cosines by a factor of 64. Somewhere we have to divide by 64 to keep the equation equal (and to not freak out my 7th grade algebra instructor) so we take a look at the function we have for f(x):

```
   f(x)=(x^2)/4
```

We realize we can do that *divide-by-64* thing here, to arrive at:

```
   f(x)=(x^2)/4/64
```

Or, written another way:

```
   f(x)=(x^2)/4*64 which is also f(x)=(x^2)/128
```

I left it as 4*64 in the program for clarity sake.

```
                    100 rem ::: save to disk :::
                    102 open1,8,15,"s0:sin/cos/sqr":close1
                    104 open2,8,2,"sin/cos/sqr,p,w"
                    106 print#2,chr$(bl)chr$(bh);
                    108 fort=0to767
                    110 print#2,chr$(peek(ba+t));
                    112 next
                    114 close2
```

The above simply saves the table we created to disk, with a loading address of $3000. =) Just what the demo needs! =)

Now, on to the second program...

The second program is the program to build the low byte, high byte and bitmask tables.

```
                    1 rem -+- make base/mask table 2.0 -+-
                    2 rem -+-                          -+-
                    3 rem -+-  by: waveform            -+-
                    4 rem -+- for: disC=overy magazine -+-
                    5 rem -+-  on: 09-14-96            -+-

                    10 ba = 13056 : cb=8192
```

"ba" is the starting address for the data created by this program.
"cb" is the base address for the character matrix used by our demo.

```
                    12 bh=int(ba/256):bl=ba-bh*256
                    100 c=0:fort=0 to 15
                    102 forq=0to7
                    104 b1=cb+(t*128)
                    106 hb=int(b1/256):lb=b1-hb*256
                    108 poke ba + c,lb:poke ba + 128 + c,hb
                    110 c=c+1:nextq,t
```

The above loop creates the low byte table as well as the high byte table for the first buffer (buffer#0)

```
                    120 c=0:fort=0 to 15
                    122 forq=0to7
                    124 b1=cb+2048+(t*128)
                    126 hb=int(b1/256):lb=b1-hb*256
                    128 poke ba +256 +c,hb
                    130 c=c+1:nextq,t
```

The above loop creates the high byte table for the second buffer (buffer#1)

```
                    150 fort=0 to 15
                    152 forq=0to7
                    154 poke ba + 384+(t*8)+q,2 (7-q)
                    156 nextq,t
```

The above loop creates the bitmask table.

```
                    200 open1,8,15,"s0:base/mask":close1
                    202 open2,8,2,"base/mask,p,w"
                    204 print#2,chr$(bl)chr$(bh);
                    206 fort=0to511
                    208 print#2,chr$(peek(ba+t));
                    210 next
                    212 close2
```

Here we have a small routine to save the tables created by this program to disk, with a loading address of $3300.

UUEncoded Files

Below you will find the uuencoded files detailed in this article. I included TurboAssmembler source, Object code (sys 2080 to run), both tables needed by the program to run, as well as the basic programs used to create them.

I will make all of these available in a zip package on my web site:
http://marie.az.com/~waveform

How To Get The Demo To Run

There wasn't enough time to write a loader for this demo before the deadline, and even though my gracious host allowed me the opportunity to add one, it is really more trouble than its worth.

So for those of you explorers who want to know how things are really done, here's a quick and simple method of launching this demo:

```
   Step One:   load "sin_cos_sqr",8,1
   Step Two:   load "base_mask",8,1
   Step Three: load "disc-demo.obj",8,1
   Step Four:  sys2080
```

[Ed. Note : To prevent possible compatibility problems with unix file-handling, the UUencoded files in this article will extract with filenames as seen in the loading sequence above. If you generate new tables with the basic table generators, then the new tables will be named to their original designations (i.e., sin/cos/sqr instead of sin_cos_sqr). Please remember to change filenames in your loading procedure to match.]

Demo-Controls

F1 : increments angle (apparent speed)
F3 : decrements angle
F5 : expands shape
F7 : shrinks shape

<RETURN>   : changes shape
<SPACE>   : resets to default size and angle
<RUN/STOP> : quits the demo

--
For more information or general commentary on this article, Mr. John Kaiser
(Waveform/MLM) may be reached at waveform@az.com

Addenum by S. Judd, Technical Editor
------+------^-------^------+-------

John shrewdly omitted a few "tricks" in order for the code to flow more clearly as a learning experience. Once the process is well-understood, a bit of sneaky optimization can be undertaken, as follows :

[...]

```
>                     ;---------------------------------------
>                     expdn      ldy #$00
>                     expdn1     lda radiu,y
>                                sec
>                                sbc #$01       ;subtract one
>                                cmp #2         ;at minimum?
>                                bcs expdn2
>                                lda #2
>                     expdn2     sta radiu,y    ;store
>                                iny
>                                cpy points     ;do all points
>                                bne expdn1
```

This can be written more efficiently, as follows :

```
expdn   LDX POINTS      ;Start at top
:L1     LDA RADIU,X
        CMP #3
        BCC :SKIP
        DEC RADIU,X
:SKIP   DEX
        BPL :L1         ;Only allows 128 points though
```

It is always better to start Y large and count downwards when you can. His loop takes 4+2+3+2+3/(2+2)+4+2+3 = 23/24 cycles per iteration, while the rewrite takes 4+2+3/(2+7)+2 = 11/17 cycles per iteration, and a few less bytes too. Here it is not such a huge deal -- 6 cycles savings per loop for the one usually used -- but this kind of trick can sometimes lead to immense savings.

[...]

```
>                     rotate     ldy #$00
```

In 'sneaky mode' we could start at Y points and go downwards here as well.

```
>                     rotate1    lda theta,y    ;get theta
>                                clc
>                                adc angleinc   ;add amount
>                                sta theta,y    ;store theta
>                                iny
>                                cpy points     ;do all points
>                                bne rotate1
>                                rts
```

[...]

```
>                     ;---------------------------------------
>                     plot2      lda lobyte,x   ;lo byte
>                                sta $02        ;
>                                lda hibyte0,x  ;hi byte
>                                sta $03        ;
>
>                                lda ($02),y    ;
>                                ora bitmask,x  ;turn pixel on
>                                sta ($02),y    ;
```

```
>
>                                pla            ;restore .a
>                                rts
>                     ;---------------------------------------
```

Please note :

1- Using a JSR plot each time adds 12 cycles (JSR + RTS)

2- Most of the time you plot within the same byte.
That is, reloading $02/$03 each time is redundant by a
factor of 8 at the very least. (The x-coordinate is
not going to change by more than 1 at each iteration!)

Every cycle saved in a line drawing routine produces huge dividends. If you have an object with just three lines in it, and each line has 100 points in it, you suddenly start saving thousands of cycles, (i.e., using a JSR adds 3600 cycles immediately). Those 14 cycles in loading the coordinate each time translates to many extra thousand cycles too. In raster time, we're talking well over half the screen here!

In general, if it's in a loop, you can't optimize it enough :).

Again, though, doing things this way makes things much clearer (which makes life easier on the programmer too).
--

```
begin 644 disc-demo.sou
M.RTM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+0T[("#!
M(,1%34\@3T8@Q$E30SU/5D5262`M(,)9.B#7059%1D]230T[#3L@("A#*2`Q
M.3DV($9/4B#$25-#/4]615)9(,U!1T%:24Y%#3LBU$A%(,I/55).04P@1D]2
M(,-/34U/1$]212#%3E1(55-)05-44R(-.RTM+2TM+2TM+2TM+2TM+2TM+2TM
M+2TM+2TM+2TM+2TM+2TM+0T@("`@("`@("`@("H]("0P.#(P#0U03TE.
M5%,@("`@(#T@)#`S,T,@("`[3E5-0D52($]&(%!/24Y44PU!3D=,14E.0R`@
M(#T@)#`S,T0@("`[04Y'3$4@5$\@24Y#4D5-14Y4#4-54E-(05!%("`@/2`D
M,#,S12`@(#M#55)214Y4(%-(05!%#51(151!("`@("`@/2`D,#,T,"`@(#M4
M2$5405,,-4D%$254@("`@("`]("0P,S4P("`@.U)!1$E54PU)5$A%5$$@("`@
M(#T@)#`S-C`@("`[5$%"($]&($E.250@5$A%5$%3#4E2041)52`@("`@/2`D
M,#,W,"`@(#L@("`@("`@24Y)5"!2041)55,-#5-#4DQ/0R`@("`@/2`Q,3DV
M("`@(#M,3T-!5$E/3B!/1B!43U`-("`@("`@("`@("`@("`@("`@("`@.R!,
M1494($-/4DY%4B!/1@T@("`@("`@("`@("`@("`@("`@("`@("`[($-(05(@34%4
M4DE8($].#2`@("`@("`@("`@("`@("`@("`@(#L@4T-2145.#0U-4TE.("`@
M("`@(#T@)#,P,#`@("`[5$%"3$4@3T8@4TE.15,,-34-/4R`@("`@("`]("0S
M,3`P("`@.R`@("`@("`@($-/4TE.15,,-35-14B`@("`@("`]("0S,C`P("`@
M.R`@("`@("`@(%-154%215,,-3$]"651%("`@("`]("0S,S`P("`@.TQ/($)9
M5$53#4A)0EE413`@("`@/2`D,#,X,"`@(#M(22!"651%4R`H)#(P,#`K*0U(
M24)95$4Q("`@(#T@)#,T,"`@("`[("`@("`@("`@*"0R.#`P*RD-0DE434%3
M2R`@("`](#0S-#@P("`@.U!!)6$5,($))5"!-05-+#3LM+2TM+2TM+2TM+2TM
M+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2T-("`@("`@("`@("`@("!*4U(@1$5-
M3TE.250@("`[24Y)5"!&3U)(@1$5-3PT@("`@("`@("`@("`@(#LM+2TM+2TM
M+2TM+2TM+2TM+2TM+2TM#4U!24X@("`@("`@3$1!("",E,3$Q,3$Q,3`@
M.U-#04X@34%44DE8#2`@("`@("`@("`@4U1!("1$0S`P("`@("`@.PT@("`@
M("`@("`@($Q$02`D1$,P,2`@("`@(#L-#2`@("`@("`@("`@0TU0(",E,3$Q
M,#$Q,3$@.T8Q#2`@("`@("`@("`@0D51($M%648Q#2`@("`@("`@("`@0TU0
M(",E,3$P,3$Q,3$@.T8S#2`@("`@("`@("`@0D51($M%648S#2`@("`@("`@
M("`@0TU0(",E,3`Q,3$Q,3$@.T8U#2`@("`@("`@("`@0D51($M%648U#2`@
M("`@("`@("`@0TU0(",E,3$Q,3`Q,3$@.T8W#2`@("`@("`@("`@0D51($M%
M648W#2`@("`@("`@("`@0TU0(",E,3$Q,3$Q,#$@.U)%5%523@T@("`@("`@
M("`@($)%42!+15E215154DX-#2`@("`@("`@("`@3$1!("",E,#$Q,3$Q,3$@
M.U-#04X@34%44DE8#2`@("`@("`@("`@4U1!("1$0S`P("`@("`@.PT@("`@
M("`@("`@($Q$02`D1$,P,2`@("`@(#L-#2`@("`@("`@("`@0TU0(",E,3$Q
M,#$Q,3$@.U-004-%#2`@("`@("`@("`@0D51($M%65-004-%#2`@("`@("`@
M("`@0TU0(",E,#$Q,3$Q,3$@.U)%5%523@T@("`@("`@("`@($)%42!+15E3
M5$]0#0U2145.5%)9("`@($I34B!23U1!5$4@("`@(#M23U1!5$4@4$]):4Y44PT-
M("`@("`@("`@("!*4U(@1$E35$%.0T4-("`@("`@("`@("!*4U(@4$Q/5"`@
M("`@("!*35`@34%)3@T[+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM
M+2TM+2TM+2TM+2TM+2TM#4M%648Q("`@("`@3$1!("0P,#)"4@0W0T5-4U0@@
M("`@("`@("`@0D1$1((#1%#1%#1%#1%#1%#1%#1%#1%#1%#1%#1%#1%#1%#1%#1%#1%
M04-%#2`@("`@("`@("`@3$1!(",P,#`P("`@("`[4T-2145.#2`@("`@("`@
M("`@4U1!("1$0S`P("`@("`@.PT-4D543%5%4E1E,4%2"`@("`@("`@("`@
M0TU0(",E,3`Q,3$Q,3$@.T8U#2`@("`@("`@("`@0D51($M%648U#2`@("`@
M("`@("!)3ED-("`@("`@("`@("!#4%D@4$])3E13("`@("`[1$\@04Q,(%!/
M24Y44PT@("`@("`@("`@($).12!%6%!54#$-#2`@("`@("`@("`@2DU0(%)%
M14Y44ED@("`@.U)%5%523@T[+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM
M+2TM+2TM+2TM+2TM#4584$1.("`@("`@3$19(",D,#`-15A01$XQ("`@("!,
M1$$@4D%$254L60T@("`@("`@("`@(%-%0PT@("`@("`@("`@(%-"0R`C)#`Q
M("`@("`@(#M354)44D%#5"!/3D4-("`@("`@("`@("!#35`@(S(@("`@("`@
M("`[050@34E.24U533\-("`@("`@("`@("!"0U,@15A01$XR#2`@("`@("`@
M("`@3$1!(",R#4584$1.,B`@("`@4U1!(%)!1$E5+%D@("`@.U-43U)%#2`@
M("`@("`@("`@24Y9#2`@("`@("`@("`@0U!9(%!/24Y44R`@("`@.T1/($%,
M3"!03TE.5%,-("`@("`@("`@("!"3D4@15A01$XQ#0T@("`@("`@("`@($I-
M4"!2145.5%)9("`@(#M215154DX-.RTM+2TM+2TM+2TM+2TM+2TM+2TM+2TM
M+2TM+2TM+2TM+2TM+2TM+0U.15=32$%012`@($E.0R!#55)32$%010T@("`@
M("`@("`@($Q$02!#55)32$%010T@("`@("`@("`@($--4"`C,R`@("`@("`@
M(#M,05-4(%-(05!%/PT@("`@("`@("`@($).12!.15=32$%013$@(#M.3U!%
M#2`@("`@("`@("`@3$1!(",D,#`@("`@("`@.U-%3$5#5"!&25)35`T@("`@
M("`@("`@(%-402!#55)32$%012`@(#L@4TA!!4$4-#4Y%5U-(05!%,2`@0TU0
M(",R("`@("`@("`@.S@@4$])3E0@4U1!!4C\-("`@("`@("`@("!"3D4@3D57
M4TA!!4$4R#2`@("`@("`@("`@2E-2($E.251/0DHR,"`@.TE.250@4TA!!4$4-
M("`@("`@("`@("!*35`@4D5%3E1260U.15=32$%013(@($--4"`C,2`@("`@
M("`@(#LR(%1224%.1TQ%4S\-("`@("`@("`@("!"3D4@3D574TA!!4$4S#2`@
M("`@("`@("`@2E-2($E.251/0DHQ,"`@.TE.250@4TA!!4$4-("`@("`@("`@
M("!*35`@4D5%3E1260U.15=32$%013,@($I34B!)3DE43T)*,#`@(#LQ(%12
M24%.1TQ%#0T@("`@("`@("`@($I-4"!2145.5%)9#3LM+2TM+2TM+2TM+2TM
M+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2T-4U1/4%)/5"`@("!,1$$@(R0P
M,"`@("`@("`[4U1/4"!!3$$P-("`@("`@("`@("!!35$$@04Y'3$5)3D,@("`[
M(%)/5$%424].#2`@("`@("`@("`@2DU0(%)%14Y44ED@("`@.PT[+2TM+2TM
M+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM#5-43U!$14U/("`@
M3$1!(",D,$$@("`@("`@.U)%4T54(%)/5$%424].#2`@("`@("`@("`@(%-%0(D$,P
M1$$@1$5324Y%(",D,#`@("`@("`@("`@(%-%0(%,D,("`@("`@("`@("!3$4-%5(
M1$$@4D%$254L60T@("`@("`@("`@(%-%0(%-%0(%-%0(D,P
M1$$@4D%$254L60T@("`@("`@("`@(%-%0(%-%0(%-%0(D,P
M041#($=865)!1$E54R`@.RM"#2`@("`@("`@("`@5$%9("`@("`@("`@("`@
M.PT@("`@("`@("`@($Q$02!-4U%2+%D@("`@(#M&*$$K0BD-("`@("`@("`@
M("!314,@("`@("`@("`@("`[#2`@("`@("`@("`@4T)#($=8651%35`@("`@
M.RU&*$$M0BD-("`@("`@("`@("!35$$@6%!/4R`@("`@("`[/5@@0T]/4D1)
M3D%410T-("`@("`@("`@("!,1%D@1UA95$A%5$$@("`[#2`@("`@("`@("`@
M3$1!($U324XL62`@("`@.T$-("`@("`@("`@("!314,@("`@("`@("`@("`[
M#2`@("`@("`@("`@4T)#($=865)!1$E54R`@.RU"#2`@("`@("`@("`@5$%9
M("`@("`@("`@("`@.PT@("`@("`@("`@($Q$02!-4U%2+%D@("`@(#M&*$$M
M0BD-("`@("`@("`@("!35$$@1UA95$5-4"`@("`[4U1/4D4@4D5354Q4#0T@
M("`@("`@("`@($Q$62!'6%E42$5402`@(#L-("`@("`@("`@("!,1$$@35-)
M3BQ9("`@("`[00T@("`@("`@("`@($-,0R`@("`@("`@("`@(#L-("`@("`@
M("`@("!!1$,@1UA94D%$2553("`[*T(-("`@("`@("`@("!405D@("`@("`@
M("`@("`[#2`@("`@("`@3$1!($U345(L62`@("`@.T8H02M"*0T@("`@
M("`@("`@(%-%0R`@("`@("`@("`@(#L-("`@("`@("`@("!30D,@1UA95$5-
M4"`@("`[+48H02U"*0T@("`@("`@("`@(%-402!94$]3("`@("`@(#L]62!#
M3T]21$E.051%#0T@("`@("`@("`@($Q$62!'6%E9("`@("`@(#M215-43U)%
M("Y9#2`@("`@("`@("`@3$1!($=864$@("`@("`@.U)%4U1/4D4@+D$-#2`@
M("`@("`@("`@4E13#2`@("`@("`@("`@.RTM+2TM+2TM+2TM+2TM+2TM+2TM
M+2TM+2TM+2T-1UA94D%$2553("`N0EE412`D,#`-1UA95$A%5$$@("`N0EE4
M12`D,#`-#5A03U,@("`@("`@+D)95$4@)#`P#5E03U,@("`@("`@+D)95$4@
M)#`P#0U'6%E9("`@("`@("Y"651%("0P,`U'6%E!("`@("`@("Y"651%("0P
M,`U'6%E414U0("`@("Y"651%("0P,`T[+2TM+2TM+2TM+2TM+2TM+2TM+2TM
M+2TM+2TM+2TM+2TM+2TM+2TM#5!,3U0@("`@("`@4$A!("`@("`@("`@("`@
M.U!215-%4E9%("Y!#0T@("`@("`@("`@($Q$02`D1#`Q."`@("`@(#M03$]4
M($E.#2`@("`@("`@("`@04Y$("`E,#`P,#`P,3`@.R!#3U)214-4#2`@("`@
M24P@3D5%1`T@("`@("`@("`@($%$0R!$6"`@("`@("`@("`[(%1/("M8#2`@
M("`@("`@("`@0D-#($1,,3!"("`@("`@(#L-("`@("`@("`@("!)3E@@("`@
M("`@("`@("`@.PT@("`@("`@("`@(%-"0R!$62`@("`@("`@("`[#41,,3!"
M("`@("`@2E-2(%!,3U0@("`@("`@(#L-("`@("`@("`@("!#4%D@63(@("`@
M("`@("`@.PT@("`@("`@("`@($).12!$3#$P02`@("`@("`[#0T@("`@("`@
M("`@(%)44PT@("`@("`@("`@(#LM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM
M+2TM#41,,C`@("`@("`@3$1!(",D,#`-("`@("`@("`@("!30D,@1%@-#2`@
M("`@("`@("`@3$18(%@Q#2`@("`@("`@3$19(%DQ#2`@("`@("`@("`@("`@
M2E-2(%!,3U0-#41,,C!!("`@("`@0TQ#("`@("`@("`@("`@(#M35$50("M8
M#2`@("`@("`@("`@24Y8("`@("`@("`@("`@(#L@54Y424P@3D5%1`T@("`@
M("`@("`@($%$0R!$62`@("`@("`@("`[(%1/("U9#2`@("`@("`@("`@0D-#
M($1,,C!"("`@("`@(#L-("`@("`@("`@("!$15D@("`@("`@("`@("`@.PT@
M("`@("`@("`@(%-"0R!$6"`@("`@("`@("`[#41,,C!"("`@("`@2E-2(%!,
M3U0@("`@("`@(#L-("`@("`@("`@("!#4%@@6#(@("`@("`@("`@.PT@("`@
M("`@("`@($).12!$3#(P02`@("`@("`[#0T@("`@("`@("`@(%)44PT@("`@
M("`@("`@(#LM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM#41,,S`@("`@("`@
M("`@3$1!(",D,#`-("`@("`@("`@("!30D,@1%D-#2`@("`@("`@("`@3$18
M(%@Q#2`@("`@("`@("`@3$19(%DQ#2`@("`@("`@("`@2E-2(%!,3U0-#41,
M,S!!("`@("`@0TQ#("`@("`@("`@("`@(#M35$50("U9#2`@("`@("`@("`@
M1$59("`@("`@("`@("`@(#L@54Y424P@3D5%1`T@("`@("`@("`@($%$0R!$
M6"`@("`@("`@("`[(%1/("M8#2`@("`@("`@("`@0D-#($1,,S!"("`@("`@
M(#L-("`@("`@("`@("!)3E@@("`@("`@("`@("`@.PT@("`@("`@("`@(%-"
M0R!$62`@("`@("`@("`[#41,,S!"("`@("`@2E-2(%!,3U0@("`@("`@(#L-
M("`@("`@("`@("!#4%D@63(@("`@("`@("`@.PT@("`@("`@("`@($).12!$
M3#,P02`@("`@("`[#0T@("`@("`@("`@(%)44PT@("`@("`@("`@(#LM+2TM
M+2TM+2TM+2TM+2TM+2TM+2TM#41,-#`@("`@("`@3$1!(",D,#`-("`@("`@
M("`@("`@("!30D,@1%@-#2`@("`@("`@("`@3$18(%@Q#2`@("`@("`@("`@
M("`@3$19(%DQ#2`@("`@("`@("`@2E-2(%!,3U0-#41,-#!!("`@("`@0TQ#
M("`@("`@("`@("`@(#M35$50("U8#2`@("`@("`@("`@1$58("`@("`@("`@
M("`@(#L@54Y424P@3D5%1`T@("`@("`@("`@($%$0R!$62`@("`@("`@("`[
M(%1/("M9#2`@("`@("`@("`@0D-#($1,-#!"("`@("`@(#L-("`@("`@("`@
M("!)3ED@("`@("`@("`@("`@.PT@("`@("`@("`@(%-"0R!$6"`@("`@("`@
M("`[#41,-#!"("`@("`@2E-2(%!,3U0@("`@("`@(#L-("`@("`@("`@("!#
M4%@@6#(@("`@("`@("`@.PT@("`@("`@("`@($).12!$3#0P02`@("`@("`[
M#0T@("`@("`@("`@(%)44PT@("`@("`@("`@(#LM+2TM+2TM+2TM+2TM+2TM
M+2TM+2TM+2TM#41,-3`@("`@("`@3$1!(",D,#`-("`@("`@("`@("!3
M0D,@1%D-#2`@("`@("`@("`@3$18(%@Q#2`@("`@("`@("`@3$19(%DQ#2`@
M("`@("`@2E-2(%!,3U0-#41,-3!!("`@("`@0TQ#("`@("`@("`@("`@("`@
M(#M35$50("M9#2`@("`@("`@("`@24Y9("`@("`@("`@("`@(#L@54Y424P@
M3D5%1`T@("`@("`@("`@($%$0R!$6"`@("`@("`@("`[(%1/("U8#2`@("`@
M("`@("`@0D-#($1,-3!"("`@("`@(#L-("`@("`@("`@("!$15@@("`@("`@
M("`@("`@.PT@("`@("`@("`@(%-"0R!$62`@("`@("`@("`[#41,-3!"("`@
M("`@2E-2(%!,3U0@("`@("`@(#L-("`@("`@("`@("!#4%D@63(@("`@("`@
M("`@.PT@("`@("`@("`@($).12!$3#4P02`@("`@("`[#0T@("`@("`@("`@
M(%)44PT@("`@("`@("`@(#LM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM#41,
M-C`@("`@("`@3$1!(",D,#`-("`@("`@("`@("!30D,@1%@-#2`@("`@("`@
M("`@3$18(%@Q#2`@("`@("`@("`@3$19(%DQ#2`@("`@("`@2E-2(%!,3U0-
M#41,-C!!("`@("`@0TQ#("`@("`@("`@("`@(#M35$50("U8#2`@("`@("`@
M("`@1$58("`@("`@("`@("`@(#L@54Y424P@3D5%1`T@("`@("`@("`@($%$
M0R!$62`@("`@("`@("`[(%1/("U9#2`@("`@("`@("`@0D-#($1,-C!"("`@
M("`@(#L-("`@("`@("`@("!)3ED@("`@("`@("`@("`@.PT@("`@("`@("`@
M(%-"0R!$6"`@("`@("`@("`[#41,-C!"("`@("`@2E-2(%!,3U0@("`@("`@
M(#L-("`@("`@("`@("!#4%@@6#(@("`@("`@("`@.PT@("`@("`@("`@($).
M12!$3#8P02`@("`@("`[#0T@("`@("`@("`@(%)44PT@("`@("`@("`@(#LM
M+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM#41,-S`@("`@("`@3$1!(",D,#`-
M("`@("`@("`@("!30D,@1%D-#2`@("`@("`@("`@3$18(%@Q#2`@("`@("`@
M("`@3$19(%DQ#2`@("`@("`@2E-2(%!,3U0-#41,-S!!("`@("`@0TQ#("`@
M("`@("`@("`@(#M35$50("M9#2`@("`@("`@("`@24Y9("`@("`@("`@("`@
M(#L@54Y424P@3D5%1`T@("`@("`@("`@($%$0R!$6"`@("`@("`@("`[(%1/
M("M8#2`@("`@("`@("`@0D-#($1,-S!"("`@("`@(#L-("`@("`@("`@("!)
M3E@@("`@("`@("`@("`@.PT@("`@("`@("`@(%-"0R!$62`@("`@("`@("`[
M#41,-S!"("`@("`@2E-2(%!,3U0@("`@("`@(#L-("`@("`@("`@("!#4%D@
M63(@("`@("`@("`@.PT@("`@("`@("`@($).12!$3#<P02`@("`@("`[#0T@
M("`@("`@("`@(%)44PT@("`@("`@("`@(%)44PT@("`@("`@("`@(#LM+2TM
M+2TM+2TM+2TM+2TM+2TM+2TM#41,-C`@("`@("`@3$1!(",D,#`-("`@("`@
M("`@("!30D,@1%@-#2`@("`@("`@("`@3$18(%@Q#2`@("`@("`@("`@3$19
M(%DQ#2`@("`@("`@2E-2(%!,3U0-#41,-C!!("`@("`@0TQ#("`@("`@("`@
M("`@(#M35$50("U8#2`@("`@("`@("`@1$58("`@("`@("`@("`@(#L@54Y4
M24P@3D5%1`T@("`@("`@("`@($%$0R!$62`@("`@("`@("`[(%1/("U9#2`@
M("`@("`@("`@0D-#($1,-C!"("`@("`@(#L-("`@("`@("`@("!)3ED@("`@
M("`@("`@("`@.PT@("`@("`@("`@(%-"0R!$6"`@("`@("`@("`[#41,-C!"
M("`@("`@2E-2(%!,3U0@("`@("`@(#L-("`@("`@("`@("!#4%@@6#(@("`@
M("`@("`@.PT@("`@("`@("`@($).12!$3#8P02`@("`@("`[#0T@("`@("`@
M("`@(%)44PT@("`@("`@("`@(#LM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM
M#41,-S`@("`@("`@3$1!(",D,#`-("`@("`@("`@("!30D,@1%D-#2`@("`@
M("`@("`@3$18(%@Q#2`@("`@("`@("`@3$19(%DQ#2`@("`@("`@2E-2(%!,
M3U0-#41,-S!!("`@("`@0TQ#("`@("`@("`@("`@(#M35$50("M9#2`@("`@
M("`@("`@24Y9("`@("`@("`@("`@(#L@54Y424P@3D5%1`T@("`@("`@("`@
M($%$0R!$6"`@("`@("`@("`[(%1/("M8#2`@("`@("`@("`@0D-#($1,-S!"
M("`@("`@(#L-("`@("`@("`@("!)3E@@("`@("`@("`@("`@.PT@("`@("`@
M("`@(%-"0R!$62`@("`@("`@("`[#41,-S!"("`@("`@2E-2(%!,3U0@("`@
M("`@(#L-("`@("`@("`@("!#4%D@63(@("`@("`@("`@.PT@("`@("`@("`@
M($).12!$3#<P02`@("`@("`[#0T@("`@("`@("`@(%)44PT@("`@("`@("`@
M4T-2145.#2`@("`@("`@("`@3$18(",D,#`@("`@("`@(#L-1$5-3TE.250U
M("!,1$$@0TA!4B`@("`@("`@.R!42$4@5$]0#2`@("`@("`@("`@4U1!("@D
M1D$I+%D@("`@(#L@3$5&5"!#3U).15(-("`@("`@("`@("!)3D,@0TA!4B`@
M("`@("`@.R!)4R!42$4-("`@("`@("`@("!,1$$@)$9!("`@("`@("`@.R!6
M04Q512!)3@T@("`@("`@("`@($-,0R`@("`@("`@("`@("`[(%-#4DQ/0PT@
M("`@("`@("`@($%$0R`C-#`@("`@("`@("`[#2`@("`@("`@("`@4U1!("1&
M02`@("`@("`@(#L-("`@("`@("`@("!"0T,@1$5-3TE.250V("`@.PT@("`@
M("`@("`@($E.0R`D1D(@("`@("`@("`[#41%34]])3DE4-B`@24Y8("`@("`@
M("`@("`@(#L-("`@("`@("`@("!#4%@@(S$V("`@("`@("`@.PT@("`@("`@
M("`@($).12!$14U/24Y)5#4@("`[#2`@("`@("`@("`@24Y9("`@("`@("`@
M("`@(#L-("`@("`@("`@("!#4%D@(S$V("`@("`@("`@.PT@("`@("`@("`@
M($).12!$14U/24Y)5#0@("`[#0T@("`@("`@("`@($Q$62`C)#`P("`@("`@
M("`[1DE,3"!#3TQ/4@T@("`@("`@("`@($Q$02`C)#`Q("`@("`@("`[($U%
M34]]262!7251(#41%34]])3DE4,3`@4U1!("1$.#`P+%D@("`@(#L@5TA)5$4-
M("`@("`@("`@("!35$$@)$0Y+#`L62`@("`@.PT@("`@("`@("`@("`@(%-402`D
M1$$P,"Q9("`@("`[#2`@("`@("`@("`@4U1!("1$0C`P+%D@("`@(#L-("`@
M("`@("`@("!)3ED@("`@("`@("`@("`@.PT@("`@("`@("`@($).12!$14U/
M24Y)5#$P("`[#0T@("`@("`@("`@($Q$02`D1#`Q."`@("`@("`@("`[4$])3E0@
MULG#(%1/#2`@("`@("`@("`@04Y$(",E,3$Q,3`P+#`@(#L@1DE24U0-("`@
M("`@("`@("!/4D$@(R4P,#`P,3`P,"`@.R!"549&15(-("`@("`@("`@("!3
M5$$@)$0P,3@@("`@("`@.PT-("`@("`@("`@("!*4U(@24Y)5$]]"2C`P("`@
M.S%35"!32$%010T-("`@("`@("`@("!,1$$@(R0P,"`@("`@("`@.TE.250-
M("`@("`@("`@("!35$$@04Y'3$5)3D,@("`@.R!605))04),15,-("`@("`@
M("`@("!35$$@0U524TA!4$4@("`@.PT-("`@("`@("`@("!25%,-#4-(05(@
M("`@("`@+D)95$4@)#`P#3LM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM
M+2TM+2TM+2TM+2T-0DQ!3DLP("`@("!,1%D@(R0P,`T@("`@("`@("`@($Q$
M02`C)#`P#4),04Y+,$$@("`@4U1!("0R,#`P+#`P+%D-("`@("`@("`@("!35$$@
M)#(P.#`L60T@("`@("`@("`@(%-402`D,C$P,"Q9#2`@("`@("`@("`@4U1!
M("0R,3@P+%D-("`@("`@("`@("!35$$@)#(R,#`L60T@("`@("`@("`@(%-4
M02`D,C(X,"Q9#2`@("`@("`@("`@4U1!("0R,S`P+%D-("`@("`@("`@("!3
M5$$@)#(S.#`L60T@("`@("`@("`@(%-402`D,C0P,"Q9#2`@("`@("`@("`@
M4U1!("0R-#@P+%D-("`@("`@("`@("!35$$@)#(U,#`L60T@("`@("`@("`@
M(%-402`D,C4X,"Q9#2`@("`@("`@("`@4U1!("0R-C`P+%D-("`@("`@("`@
M("!35$$@)#(V.#`L60T@("`@("`@("`@("!)3ED-("`@("`@("`@("!"4$P@
M0DQ!3DLP00T@("`@("`@("`@(%)44PT-.RTM+2TM+2TM+2TM+2TM+2TM+2TM
M+2TM+2TM+2TM+2TM+2T-0DQ!3DLQ("`@("!,1%D@(R0P,`T@("`@("`@("`@
M("!,1$$@(R0P,`T@("`@("`@("`@(%-402`D,C`P,"Q9#2`@("`@("`@("`@
M("`@4U1!("0R,#@P+%D-("`@("`@("`@("!35$$@)#(Q,#`L60T@("`@("`@("`@
M("`@(%-402`D,C$X,"Q9#2`@("`@("`@("`@4U1!("0R,C`P+%D-("`@("`@
M("`@("!35$$@)#(R.#`L60T@("`@("`@("`@(%-402`D,C,P,"Q9#2`@("`@
M("`@("`@4U1!("0R,S@P+%D-("`@("`@("`@("!35$$@)#(T,#`L60T@("`@
M("`@("`@(%-402`D,C0X,"Q9#2`@("`@("`@("`@4U1!("0R-3`P+%D-("`@
M("`@("`@("!35$$@)#(U.#`L60T@("`@("`@("`@(%-402`D,C8P,"Q9#2`@
M("`@("`@("`@4U1!("0R-C@P+%D-("`@("`@("`@("!)3ED-("`@("`@("`@("!"
M4%D@04$]])3E13#2`@("`@("`@("`@04DY%($E.4$E/0DHQ,"`@("`@("`@("`@
M(%)44PT[+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM
M#4E.251/0DHR,"`@3$1!("(",D,#D-("`@("`@("`@("!35$$@04$]])3E13#0T@
M("`@("`@("`@($Q$62`C)#`P#4E.251/0DHR,2`@3$1!(%1(151!+$,`L60T@
M("`@("`@("`@(%-402!)5$%%5$$L60T@("`@("`@("`@(%-402!42$5402Q9
M#2`@("`@("`@("`@3$1!(%1(151!+$,`L60T@("`@("`@("`@(%-402!)4D%$%$
M254L60T@("`@("`@("`@(%-402!2041)52Q9#2`@("`@("`@("`@24Y9#2`@
M("`@("`@("`@0U!9(%/24Y44T@("`@("`@("`@(%)44PT-.RTM+2TM+2TM+2TM
M("`@("`@("`@("!25%,-.RTM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM+2TM
M+2TM+2TM+0U42$5403`P("`@("Y"651%(#`L.#8L,3<R+#`-4D%$254P
M,"`@("`N0EE412`T,"PT,"PT,"PT,`T-5$A%5$$Q,"`@("`N0EE412`P+#8T
M+#$Y,BPQ,C@L`U2041)53$P("`@("Y"651%(#0P+#0P+#0P+#0P+#0P#0U4
M2$5403(P("`@("Y"651%(#`L.38L,3DR+#,R+#$R.`T@("`@("`@("`@("Y"
M651%(#(R-"PV-"PQ-C`L`U2041)53(P("`@("Y"651%(#0P+#0P+#0P+#0P
C+#0P#2`@("`@("`@("`@+D)95$4@-#`L-#`L-#`L-#`-#1HP
`

end

begin 644 disc-demo.obj
M(`@@A`NI_HT`W*T!W,GO\"G)W_`HR;_P)\GW\";)_?`EJ7^-`-RM`=S)[_`<
MR7 P&R#F""#Y"$PC"$QM"$QS"$QY"$R3"$RM"$S6"$S>".X]`TQ/",X]`TQ/
```

```
M"*``N5`#&&D!R3^0`JD_F5`#R,P\`]#K3$\(H`"Y4`,XZ0')`K`"J0*94`/(
MS#P#T.M,3PCN/@.M/@/)`]`%J0"-/@/)`M`&(*0,3$\(R0'0!B"$#$Q/""!D
M#$Q/"*D`C3T#3$\(J8&-#=Q,9OZ@`+E``QAM/0.90`/(S#P#T/!@H`"Y0`.-
MS@FY4`.-S0D@=`FMSPD8:4"-?@NMT`D8:4"-?PN@`;E``XW.";E0`XW-"2!T
M":W/"1AI0(V`"ZW0"1AI0(V!"XQS"2`""JQS":V`"XU^"ZV!"XU_"\C,/`/0
MQ*T8T$D"C1C0K1C0*0+P!B#T"TQR"2`L#&``C-$)C=()K,X)N0`Q..W-":BY
M`#*-TPFLS@FY`#$8;<T)J+D`,CCMTPF-SPFLS@FY`#`X[<T)J+D`,HW3":S.
M";D`,!AMS0FHN0`R..W3"8W0":S1":W2"6``````````2*T8T"D"T!.]`#.%
MK7\+[8$+C8,+&*V""Q`I2?]I`8V""ZV#"Q`22?]I`8V#"ZV""\V#"[`W3)`*
MK8(+S8,+L'!,U`JM@PL0$DG_:0&-@PNM@@O-@PNP#TP8"ZV""\V#"[``'3%P+
M8$SV"DPZ"ZD`[8(+KGX+K'\+(-0)&.AM@PN0!,CM@@L@U`GL@`O0[6"I`.V#
M"ZY^"ZQ_"R#4"1C(;8(+D`3H[8,+(-0)S($+T.U@J0#M@@NN?@NL?PL@U`D8
MZ&V#"Y`$B.V""R#4">R`"]#M8*D`[8(+KGX+K'\+(-0)&(AM@@N0!.CM@PL@
MU`G,@0OO[6"I`.V""ZY^"ZQ_"R#4"1C*;8,+D`3([8(+(-0)[(`+T.U@J0#M
M@PNN?@NL?PL@U`D8R&V""RNV#"R#4"<R!"]#M8*D`[8(+KGX+K'\+(-0)
M&,IM@PN0!(CM@@L@U`GL@`O0[6"I`.V#"ZY^"ZQ_"R#4"1B(;8(+D`3*[8,+
M(-0)S($+T.U@````````J0:-(-"-(="IDR#2_ZE_C0W<(/0+("P,J0"-\PN@
M`*FLA?JI!(7[H@"M\PN1^N[S"Z7Z&&DHA?J0`N;[Z.`0T.C(P!#0V:``J0&9
M`-B9`-F9`-J9`-O(T/&M&-`I\`D(C1C0(&0,J0"-/0.-/@/@-@`*``J0"9``"9
M@""9`"&9@"&9`"*9@"*9`".9@".9`"29@"29`"69@"69`":9@":9`">9@"?(
M$,U@H`"I`)D`*)F`*)D`*9F`*9D`*IF`*ID`*YF`*YD`+)F`+)D`+9F`+9D`
M+IF`+ID`+YF`+\@0S6"I!(T\`Z``N<0,F6`#F4`#N<@,F7`#F5`#R,P\`]#H
M8*D%C3P#H`"YS`R98`.90`.YT0R9<`.94`/(S#P#T.A@J0F-/`.@`+G6#)E@
M`YE``[G?#)EP`YE0`\C,/`/0Z&``5JP`*"@H*``!`P(``*"@H*"@`8,`@@.!`
MH``H*"@H*"@H*"@@:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:
4&AH:&AH:&AH:&AH:&AH:&AH:
`

end

begin 644 m.sin_cos_sqr
M`0@H"`$`CR`M*RT@("!-04M%(%-)3B]#3U,O4U%2(#(N,"`@("TK+0!/"`(`
MCR`M*RT@("`@("`@("`@("`@("`@("`@("`@("TK+0!V"`,`CR`M*RT@
M($)9.B!7059%1D]232`@("`@("`@("`@("TK+0"=""`0`CR`M*RT@1D]2.B!$
M25-#/4]615)9($U!1T%:24Y%("TK+0#$"`4`CR`M*RT@($]..B2TQ-"TY
M-B`@("`@("`@("`@("TK+0#J"`H`CR`Z.CH@34%+12!324X@04Y$($-/4R!4
M04),15,@.CHZ`!$)#`!"0;(Q,C(X.#I"2+*U*$)!K3(U-BDZ0DRR0D&K0DBL
M,C4V`!\)#@"!0EFR,*0R-34`/0D0`$1%LD)9K#$N-"#W.E)!LD1%K"C_K3$X
M,"D`5`D2`%.RM2@HORA20202FL-C0IJBXU*0!J"10`BU.S,*=3LC(U-:NV*%,I
MJC$`@0D6`$.RM2@HOBA20202FL-C0IJBXU*0"7"1@`BT.S,*=#LC(U-:NV*$,I
MJC$`N0D:`)<@0D$@JB!"62Q3.I<@0D$@JB`R-38@JB!"62Q#`+\)'`"""`-P)
M,@"/(#HZ.B!!-04M%(%-14B!404),15,@.CHZ`&0)"(%"6;(PI#$R-P"C8`
M4U&R*$)9K$)9*:TH-"*PV-"D`&`HX`)<@0D$@JB!"62Q3,@*(@JD)9+%-1
M`(%"6;(PI#$R-P\"CX`4U&R*$)9K$)9*:TH-"*PV-"D`6`I``)<@0D$@JB!"U
M,3(@JC(U-:343J343J343,%B!0DFR,*0R-34`/R"P)"0(!1)4+%-1
M9@"?,2PX+#$U+#-1)`3]3*U-"*U`14B(ZH#$`L`IH`)\R0D%B4TE.+
MT+-/4R$SU(L4"Q7(@##"FH`#&(`@8(0`H*"@H*"@H*"@H*"@H*"@H*"@H*"@
M`.,*;@B+BS'*@`@T.2@J5"DI`P#`@`&@*"@H*"@H*"@:&AH:&AH:&AH:&AH:
#&AH:
`

end

begin 644 m.base_mask
M`0@H"`$`CR`M*RT@34%+12!"05-%+%+TU!4TL@5$%"3$4@,BXP("TK+0!/"`(`
MCR`M*RT@("`@("`@("`@("`@("`@("`@("`@("TK+0!V"`,`CR`M*RT@
M($)9.B!7059%1D]232`@("`@("`@("`@("TK+0"=""`0`CR`M*RT@1D]2.B!$
M25-#/4]615)9($U!1T%:24Y%("TK+0#=""`H`0D$@LB`Q,S`U`-0@Z(`P`
M`$)(LK4H0D&M,C4V*3H`M`$(R`R*PR-38`!$)'`$$E(-B`F,.&@3@'`3P($$R$*&$(`P,-
M.$D%$%"LD(QLE$0`0K(`PK2`0R`KR-%C,$R$5@`@-TR`@"J$@8R0`4R`@0@I"%0
M&`!F`I%$XP`)+('!HD92'`P+0`'`5``0S"DI1D#1"S0P$TQ&`)+("S0P$TQ&`)+
M-$`4>@`B@S!`CE$`@W0`,$P`H8@`0DP`C`%1$XPK$T"A`!`0J@X`!%`A`!`0J@X
M`!%`@D(`A`!`0J@X`!%`@D(`A`!`0J@X`!%`@D(`A`!`0J@X`!%`@D(`A`!`0J@X
```

```
M/3T\/#L[.CDX.#<V-30S,C$P+RXM+"LJ*2<F)20B(2`>'1L:&!<6%!,1$`X,
M"PD(!@4#`@#^_?OZ^/?U\_+P[^WLZNGGYN7CXN#?WMS;VMG7UM74T]+1S\[-
MS<S+RLG(R,?&Q<7$Q,/#PL+"P<'!P,#`P,#`P,#!P<'"PL+#P\3$Q<;&
MQ\C(R<K+S,W.S]#1TM/4U=;8V=K;W=[?X>+CY>;HZ>OL[N_Q\O3U]_CZ^_W_
M``(#!08("@L-#A`1$Q06%QD:'!T>("$B)"4F*"DJ*RPM+C`Q,C,T-#4V-S@Y
M.3H[.SP\/3T^/CX_/S]`0$!`0``````````````````````!`0$!`0$!`@("
M`@(#`P,#!`0$!`4%!04&!@8'!P<("`D)"0H*"@L+#`P-#0X.#P\0$!$1$A(3
M$Q04%146%Q<8&1D:&AL<'!T>'A\@(2$B(R0D)28G)R@I*BLK+"TN+S`Q,3(S
M-#4V-S@Y.CL\/3X_/SX]/#LZ.3@W-C4T,S(Q,3`O+BTL*RLJ*2@G)R8E)"0C
M(B$A(!\>'AT<'!L:&AD9&!<7%A45%!03$Q(2$1$0$`\/#@X-#0P,"PL*"@H)
M"0D("`<'!P8&!@4%!04$!`0$`P,#`P("`@("`0$!`0$!`0``````````````
M```````:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:
M&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:
I&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:
`

end

begin 644 base_mask
M`#,``````````("`@("`@("```````````"`@("`@("`@```````````@("`
M@("`@(```````````("`@("`@("```````````"`@("`@("`@```````````
M@("`@("`@(```````````("`@("`@("```````````"`@("`@("`@"`@("`@
M("`@("`@("`A(2$A(2$A(2$A(2$A(2$A(B(B(B(B(B(B(B(B(B(B(B(B(B,C
M(R,C(R,C(R,C(R,C(R,D)"0D)"0D)"0D)"0D)"0D)24E)24E)24E)24E)24E
M)28F)B8F)B8F)B8F)B8G)R<G)R<G)R<G)R<G)R<G*"@H*"@H*"@H*"@H
M*"@H*"DI*2DI*2DI*2DI*2DJ*BHJ*BHJ*BHJ*BHJ*BHJ*RLK*RLK*RLK
M*RLK*RLK*RPL+"PL+"PL+"PL+"PM+2TM+2TM+2TM+2TM+2TM+BXN+BXN
M+BXN+BXN+BXN+B\O+R\O+R\O+R\O+R\O+R^`0"`0"`0"`8!`(!`(!`(!@$`@
M$`@$`@&`0"`0"`0"`8!`(!`(!`(!@$`@$`@$`@&`0"`0"`0"`8!`(!`(!`(!
M@$`@$`@$`@&`0"`0"`0"`8!`(!`(!`(!@$`@$`@$`@&`0"`0"`0"`8!`(!`(
M!`(!@$`@$`@$`@&`0"`0"`0"`1H:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:
M&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:
M&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:&AH:
*&AH:&AH:&AH:&AH:
`

end
```

```
/S06::$d400:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

SID Primer: The Working Man's Guide to SID

by

Stephen L. Judd
(sjudd@nwu.edu)

The Sound Interface Device (SID): it is one of the signature chips in the C64, but to this day a number of people still do not have a good understanding of SID -- the meaning of its registers, the idea behind how it generates sounds, etc. (I myself was in this category not so long ago). This article exists to rectify this situation once and for all with a general overview of the chip which everyone should be able to comprehend. I have also included a program for experimenting with SID, which allows the user to change the various registers and see (not to mention hear!) the effect.

To use the program, just load it and run. Use the cursor keys to move between the various settings, and use the +/- keys to change those settings. To change certain settings, like frequency, by large amounts, you can press shift +/-. Large jumps will work on any setting that takes up more than one byte. Press 1, 2, or 3 to switch between the settings for voices 1, 2, and (guess!). Certain settings, such as the waveforms, are simple toggles, and the space bar is used to toggle these on and off. Finally, * will toggle the output of voice 3 on and off, although you need to press it very fast, as I only included it for the sake of completeness and did not debounce it or anything (all keys repeat).
The upper left corner displays all of the (relevant) SID registers, so when you change a particular setting you will see the corresponding change in the appropriate SID registers. Since SID is located at 54272 ($D400), these values reflect the contents of SID+register number.
The two registers in the lower right corner of the SID box show the current values of the voice 3 oscillator and envelope generator. If you are feeling impatient, set the sustain to 15, select the triangle waveform, use shift+ to get a reasonably large frequency, and set the gate bit, which turns the sound on. Viola! Violin! SID!

SID is pretty straightforward. There are a total of three voices. There are also three filters which the output of a voice may be run through. SID is also used to read the paddles (i.e. potentiometer settings).

All three voices have a number of features in common. There are four possible waveforms which may be selected: triangle, sawtooth, pulse, and noise. The waveform determines the basic "sound" of the sound. Multiple waveforms may be selected on a voice, which results in a sort of mashing together of the waveforms, although noise should never be selected at the same time as the others.

The frequency of each voice runs from about 0 Hz to around 4000 Hz, with 65536 steps in-between. Higher frequencies may be generated via ring modulation. According to "Mapping the 64", the exact frequency may be calculated as :

FREQUENCY = (REGISTER VALUE * CLOCK)/16777216 Hz

where CLOCK=1022730 for NTSC systems and CLOCK=985250 for PAL systems.

One of the more important features of a SID voice is the ADSR envelope. The ADSR Envelope is quite easy to understand. Here is how to create an envelope using a stereo: first turn the volume all the way to zero (my volume goes to 11, so it takes me a while). Now start turning it up to some level, say 5. As soon as it hits 5, start going back down again, until it hits 2. Let it sit at 2 until you get tired of sitting, and then turn it back down to zero.

ADSR stands of course for Attack Decay Sustain Release. The first part, turning it up to 5, is the attack phase. Changing the attack changes how quickly the volume goes to its maximum. The second phase, turning down to 2, is the decay phase. Again, changing the decay changes the rate at which the volume decays from the maximum volume to the sustain volume, which in this case is 2. Changing the sustain value changes this sustain level. The sound will remain at this volume until the release phase is initiated. Changing the release value changes the rate at which the sound will decay to zero.

To start the attack phase, simply set the gate bit. To start the release phase, clear the gate bit. The program may be used to investigate the envelope, via the voice 3 envelope generator output register: select voice 3, and set the ADSR values to, say attack 14, decay 10, sustain 10, and release 8. Set the gate bit, and the register will first increase, then decrese, then sit still, until the gate bit is cleared, at which point the sound will decay towards zero.

There are two more important features for each voice: ring modulation and synchronization. Ring modulation produces nonharmonic overtones, i.e. gives it a bell-like or gong-like sound. Synchronization combines two waveforms in a special way, and tends to amplify higher frequencies contained in the waveform.

These two features modulate the voice with the one "underneath" it; that is, Voice 1 is modulated by voice 3, voice 2 by voice 1, etc. Ring modulation can only be applied to the triangle waveform; that is, if ring modulation is selected for voice 1, then voice 1 must have the triangle waveform selected. Using the program, start some note playing with voice 1, and select ring modulation. Then go to voice 3 and select a waveform and a frequency -- the ADSR settings may all be 0. There are also three filters available, which may be combined. These are not separate filters for each voice, but rather one set of filters which voices may all run through. Any sound generated by SID will contain a number of frequencies. As you might expect, the filters will filter out frequencies in a special way.

To use the filters, the cutoff frequency must first be selected. All frequency components above or below this cutoff frequency will be reduced in volume -- the further away these frequencies are, the more they will be attenuated. The low pass filters will let all frequencies below the cutoff through, and attenuate frequencies higher than the cutoff. The high pass filter does the opposite. The bandpass filter attenuates frequencies on both sides of the cutoff. When low and high filters are selected simultaneously, the result is called a notch filter.
As an example, let's say we had a sawtooth wave playing at 100 Hz. This wave contains a number of higher harmonics, in particular harmonics at 200 Hz, 300 Hz, 400 Hz, 500 Hz, and so on. If the low pass filter were selected, and the cutoff frequency was set at 380 Hz, the 200 Hz and 300 Hz frequencies would pass right through, but the 400 Hz 500 Hz etc. frequencies would be attenuated, the 500 Hz harmonic being decreased more than the 400 Hz harmonic.

Resonance is a special feature which boosts frequencies near the cutoff frequency. This creates a somewhat sharper filtering effect. The downside of the filters is that they vary quite a bit between different SID chips, so filtered sounds on one machine may sound quite different than the identical settings on another machine. The game Beach-Head even allowed the user to change the filter settings, to try to compensate for this. According to "Mapping the 64" the exact cutoff frequency is :

$$\text{FREQUENCY} = (\text{REGISTER VALUE} * 5.8) + 30 \text{ Hz}$$

Note that the cutoff frequency is only 11 bits wide, i.e. has values from 0 to 2047.

One other setting is bit 7 of location 54286 ($D418), which disconnects the output of voice 3. This lets voice 3 be used, in particular the envelope output register, without having to listen to it. The waveforms are the "shape" of the sound. Sound is how two sensors on the side of your head interperet pressure variations in the air. Speakers convert changes in voltage into pressure. The waveform generators are what control this voltage. Once again, the program may be used to get a feel for how the waveforms look, by using voice 3 and a very low frequency setting (like 1), and seeing the change in the waveform output register.

The first waveform is the Triangle wave. This is SID's closest approximation to a pure sine wave. It starts at some value, increases up to its maximum value, then decreases down to its minimum value, and so on. Mathematically, this wave may be expressed as :

$$\sin(x) - \sin(3x)/9 + \sin(5x)/25 - \sin(7x)/49 + \ldots$$

so a triangle wave with fundamental frequency 100 Hz contains frequencies of 300 Hz, 500 Hz, 700 Hz, and so on. Note that the amplitude of each harmonic decreases as the square of the frequency.

The second waveform is the Sawtooth. This waveform increases up to its maximum, like the Triangle, but once it gets there it suddenly drops down to its minimum, so it is like half of a triangle. This wave may be expressed as :

$$\sin(x) + \sin(2x)/2 + \sin(3x)/3 + \sin(4x)/4 + \ldots$$

As you can see it has a much higher harmonic content than the triangle wave -- more harmonics are present, and their amplitudes decrease less rapidly than the triangle wave (e.g. compare sin(3x)/3 with sin(3x)/9).

Next up is the Pulse, or Square, waveform. The pulse waveform is either high or low. With SID you can set how much of the time is spent high and how much is spent low. The ratio of the time the signal is high to the time of a complete cycle is called the duty cycle. A duty cycle of 1:2 is a special case called a square wave :

$$\sin(x) + \sin(3x)/3 + \sin(5x)/5 + \ldots$$

The neat thing about pulse waves is that as the duty cycle is changed the harmonic content varies widely -- try using the program to change the pulse width and it will be obvious. Pulse waves can have a very irregular energy distribution among the various harmonics; compare with a sawtooth, where the harmonics decrease smoothly.

The final waveform is the Noise waveform. Noise is simply a randomly generated waveform; that is, random values are output through the waveform generator according to the frequency setting. Obviously the result can't be written down as a harmonic expression like the above waveforms, but the frequency spectrum as a whole may be described. White Noise contains all frequencies in equal proportion. SID generates what is known as Blue Noise: a minimum frequency is set and all frequencies above this minimum are generated with equal probability; thus SID noise is biased towards higher frequencies.

A few words about ring modulation and synchronization: ring modulation is a multiplication of two signals. To see what happens, consider multiplying two sine waves together :

$$\sin(f1) * \sin(f2) = \sin(f1-f2+\pi/2)/2 - \sin(f1+f2+\pi/2)/2$$

where a handy trig identity is used. The important thing to notice is that two new frequencies are generated, f1+f2 and f1-f2, with smaller amplitude and different phase. Try ring modulating a wave at one frequency with another whose frequency differs by 1 or 2, using the program. The result is two waves with fundamental frequencies slightly off from one another, which generates beats. In short, though, using ring modulation creates a whole slew of new sum and difference harmonics, which gives a bell-like sound. It should also be clear that frequencies higher than the maximum SID frequency setting may be generated in this way.

SID doesn't work quite this way, though. That is, it does indeed generate all the sum and difference frequencies, but there is no waveform multiplication going on. See the Yannes interview, elsewhere in this issue, for more information.

Synchronization synchronizes one voice to another. That is, if voice 1 is synchronized to voice 3, the voice 1 waveform will "start over" according to the frequency of voice 3. Adding a discontinuity like this has the effect of generating higher harmonics, and can change the pitch as well. Imagine a triangle waveform, counting downwards, when suddenly the waveform is reset to zero and starts counting up again. The triangle is now starting to resemble a sawtooth, which, as was pointed out earlier, contains more high frequencies with larger amplitudes.

That then is a fairly complete summary of SID. For more detail on the inner workings of SID see the interview with Bob Yannes, designer of the SID chip, elsewhere in this issue. For information on programming SID, see just about any book on the Commodore 64. Otherwise, have some fun playing with SID!

```
begin 600 wave3.run
M`0@`"+49GC(P-S$@4TQ*(#DO,34O.38`J0"%_:(7G0#`RA#ZJ0^-&,"I0(7Z
MJ8"-B@*-(-"I!HTAT*F3(-+_3'8)$Y(1$1$1$1$1$1F4%45$%#2R`Z!3`P
M(""95T%6149/4DTZ#41%0T%9("`Z!3`P("`@5%))(%-!5R!054P@3D])#9E3
M55-404E..@4P,"`@F5!53%-%(%=)1%1(.@4@("`P#9E214Q%05-%.@4P,"`@
MF49215$Z!2`@("`P#1$@1T%412!224Y'#2!364Y#(%1%4U0-$9E&24Q415(Z
M!2!6,2!6,B!6,PT@3$]7(%!!4U,-($)!3D0@4$%34PT@2$E'2"!005-3#1&9
M4D533TY!3D-%.@4P,`V90U543T9&.@4P,#`P$Y(%(%8Q("`@5C(@("!6,R`@
M($A)(&(3$1$1$1$1$%8V-C8V-C8V-3241C8V-C8V-C8VN1G6*1G6*1G6*1
MG6*1G6*1G6*1G6(`J0B%_ZE!A?Z@`+'^\`H@TO_(T/;F_]#RH!*I`YDH
MV)E0V)EXV)F@V)G(V)GPV)D8V8@0Z*(`J0"%_JD$A?^@`!BE_FDHA?Z0`N;_
MBB"8#\B]`,"=`-0@CP_(BAAI!R"8#\B]!\"=!]0@CP_(BAAI#B"8#\B]#L"=
M#M0@CP_(X`2P$8H8:14@F`_(O17`G174((\/X`70#*D+()@/R*T;U""/#^`&
MT`RI#""8#\BM'-0@CP_HX`?0AJD$A?^E_<D`T`*I`<D'T`*I!LD.T`*I"X7^
MH`$@>P^@`:7+R3C0"2"%#ZD`A?WP),D[T`D@A0^I!X7]T!?)"-`'((4/J0Z%
M_<DQT`BM&,!)@(T8P*F0A?ZI!87_H`BF_;T%P$I*2DJ%^R"F#Z7ZR4#0%Z`&
M(!@/"@H*"H7[IOV]!<`I#P7[G07`J;B%_JD%A?^@"+T%P"D/A?LE!
MT!.@!B`8#X7[IOV]!<`I\`7[G07`J>"%_JD%A?^@"+T&P$I*2DJ%^R"F#Z7Z
MR4+0%Z`&(!@/"@H*"H7[IOV]!L`I#P7[G0;`J<2%_JD&A?^@"+T&P$I*2DJ%
MI@^E^LE#T!.@!B`8#X7[IOV]!L`I\`7[G0;`J<2%_JD%A?^E_1AI!*JI$(7[
M/0#`\`*I,1AI(*``D?ZE^LE$T`6@`B!)#ZG(A?ZI!87_I?VH`(6@`B!)#ZG(A?ZI
M,1AI(*``D?ZE^LE%T`6@`B!)#ZG,A?ZI!H7_J4"%^ST`P/`"J3$8:2"@`)'^I?K)2]`%H`,@20^IT(7^J0:%_[T&P$I*2DJ%^R"F#Z7Z
MR4?0!:`"($D/J>R%_JD%A?^@!*T`P$H7_I/0`^I!X7]T!?)"-`'((4/J0Z%
MI?K)3@0!:`"($D/J>R%_JD$A?^@"+T%P$J*2DJ%^R"F#Z7Z
M_JD&A?^I`"%_JD&A?^I`(7_J4"%^ST`P/`"J3$8:2"@`)'^I?K)2]`%H`,@20^IT(7^J0:%_[T&P$I*2DJ%
```

/S07::$d400:::

::

Progenitor of the SID :

An interview with Bob Yannes

by Andreas Varga

This is an interview with the creator of the SID chip, namely Bob Yannes, who after working for Commodore, co-founded the well-known synthesizer company Ensoniq. In the latter part of the interview he gives us a very detailed description of the SID's technology.

The interview was done via e-mail in August 1996 by Andreas Varga. Some questions came from Linus Walleij.

[A note from your friendly technical editor: There is a little addendum at the end of this interview, with a few comments to clarify or expand upon some of the statements in the interview. These comments are referenced in the interview by a number in square brackets, like this: [3]

These comments are due in large part to a series of very helpful conversations with Andreas Boose, Marko Makela, Michael Schwendt, and Andreas Varga. Special thanks to Andreas Boose for explaining how a phase accumulating oscillator works :).
-TechEd]

--

> Did you foresee that people would actually treat your little
> VLSI-chip like an instrument?

Actually, I was an electronic music hobbyist before I started working for MOS Technology (one of Commodore's chip divisions at the time) and before I knew anything at all about VLSI chip design. One of the reasons I was hired was my knowledge of music synthesis was deemed valuable for future MOS/Commodore products. When I designed the SID chip, I was attempting to create a single-chip synthesizer voice which hopefully would find it's way into polyphonic/polytimbral synthesizers.

> Are you aware of the existence of programs like SIDPLAY,
> PlaySID,... which emulate the SID chip up to the smallest click ?

I only recently became aware of them (through your website). I'm afraid I haven't thought much about SID in the last 15 years...I am constantly amazed and gratified at the number of people who have been positively affected by the SID chip and the Commodore 64 (which I also designed) and who continue to do productive things with them despite their "obsolescence".

> Have you heard the tunes by Rob Hubbard, Martin Galway, Tim
> Follin, Jeroen Tel, and all the other composers ?

I'm afraid not, are recordings available in the US?

> Did you believe this was possible to do with your chip?

Since I haven't heard them I'm not sure what we are talking about, however, I did design the SID chip with enough resolution to produce high-quality music. I was never able to refine the Signal-to-noise ratio to the level I wanted, though.

> How much of the architecture in the SID inspired you when
> working with the Ensoniq synthesizers?

The SID chip was my first attempt at a phase-accumulating oscillator, which is the heart of all wavetable synthesis systems. Due to time constraints, the oscillators in SID were not multiplexed, therefore they took up a lot of chip area, constraining the number of voices I could fit on a chip. All ENSONIQ sound chips use a multiplexed oscillator which allows us to produce at least 32 voices per chip. Aside from that, little else of SID is to be found in our designs, which more closely resemble the Mountain Computer sound card for the Apple II (the basis of the Alpha Syntauri system). The DOC I chip (used in the Mirage and ESQ-1) was modeled on this sound card. Our current designs, which include waveform interpolation, digital filters and digital effects are new designs that aren't really based on anything other than our imaginations.

> How big impact do you think the SID had on the synthesizer
> industry?

Well, I don't think it had much impact on the synthesizer industry. I remember once at Commodore that Sequential Circuits was interested in buying the chip, but nothing ever came of it. My intention in designing the chip (since MOS Technology was a merchant semiconductor house at the time and sold chips to the outside world) was to be able to sell the SID chip to synthesizer manufacturers. SID chip production was completely consumed by the Commodore 64 and by the time chips were readily available, I had left Commodore and never had the opportunity to improve the fidelity of the chip.

> What would you have changed in the SIDs design, if you had a
> bigger budget from Commodore ?

The issue wasn't budget, it was development time and chip size constraints. The design/prototype/debug/production schedule of the SID chip, VIC II chip and Commodore 64 were incredibly tight (some would say impossibly tight)--we did things faster than Commodore had ever done before and were never able to repeat after! If I had had more time, I would have developed a proper MOS op-amp which would have eliminated the signal leakage which occurred when the volume of the voice was supposed to be zero. This lead to poor signal-to-noise ratio, although it could be dealt with by stopping the oscillator. It would also have greatly improved the filter, particularly in achieving high resonance. I originally planned to have an exponential look-up table to provide a direct translation for the equal-tempered scale, but it took up too much silicon and it was easy enough to do in software anyway.

> The SID is very complex for its time. Why didn't you settle
> with an easier design ?

I thought the sound chips on the market (including those in the Atari computers) were primitive and obviously had been designed by people who knew nothing about music. As I said previously, I was attempting to create a synthesizer chip which could be used in professional synthesizers.

> Do you still own a C64 (or another SID-equipped computer) ?

Sure, I have a couple of them (including the portable), but I honestly haven't turned them on in years.

> Did Commodore ever plan to build an improved successor to the SID?

I don't know. After I left I don't think there was anyone there who knew enough about music synthesis to do much more than improve the yield of the SID chip. I would have liked to have improved the SID chip before we had to release to production, but I doubt it would have made any difference to the success of the Commodore 64.

> Can you give us a short overview of the SIDs internal architecture ?

It's pretty brute-force, I didn't have time to be elegant. Each "voice" consisted of an Oscillator, a Waveform Generator, a Waveform Selector, a Waveform D/A converter, a Multiplying D/A converter for amplitude control and an Envelope Generator for modulation. The analog output of each voice could be sent through a Multimode Analog Filter or bypass the filter and a final Multiplying D/A converter provided overall manual volume control.

As I recall, the Oscillator is a 24-bit phase-accumulating design of which the lower 16-bits are programmable for pitch control. [1] The output of the accumulator goes directly to a D/A converter through a waveform selector. Normally, the output of a phase-accumulating oscillator would be used as an address into memory which contained a wavetable, but SID had to be entirely self-contained and there was no room at all for a wavetable on the chip.

The Sawtooth waveform was created by sending the upper 12-bits of the accumulator to the 12-bit Waveform D/A.

The Triangle waveform was created by using the MSB of the accumulator to invert the remaining upper 11 accumulator bits using EXOR gates. These 11 bits were then left-shifted (throwing away the MSB) and sent to the Waveform D/A (so the resolution of the triangle waveform was half that of the sawtooth, but the amplitude and frequency were the same).

The Pulse waveform was created by sending the upper 12-bits of the accumulator to a 12-bit digital comparator. The output of the comparator was either a one or a zero. This single output was then sent to all 12 bits of the Waveform D/A.

The Noise waveform was created using a 23-bit pseudo-random sequence generator (i.e., a shift register with specific outputs fed back to the input through combinatorial logic). [2] The shift register was clocked by one of the intermediate bits of the accumulator to keep the frequency content of the noise waveform relatively the same as the pitched waveforms. The upper 12-bits of the shift register were sent to the Waveform D/A.

Since all of the waveforms were just digital bits, the Waveform Selector consisted of multiplexers that selected which waveform bits would be sent to the Waveform D/A. The multiplexers were single transistors and did not provide a "lock-out", allowing combinations of the waveforms to be selected. The combination was actually a logical ANDing of the bits of each waveform, which produced unpredictable results, so I didn't encourage this, especially since it could lock up the pseudo-random sequence generator by filling it with zeroes. [3] [Actually, the result isn't a logical ANDing at all. -TechEd]

The output of the Waveform D/A (which was an analog voltage at this point) was fed into the reference input of an 8-bit multiplying D/A, creating a DCA (digitally-controlled-amplifier). The digital control word which modulated the amplitude of the waveform came from the Envelope Generator.

The Envelope Generator was simply an 8-bit up/down counter which, when triggered by the Gate bit, counted from 0 to 255 at the Attack rate, from 255 down to the programmed Sustain value at the Decay rate, remained at the Sustain value until the Gate bit was cleared then counted down from the Sustain value to 0 at the Release rate.

A programmable frequency divider was used to set the various rates (unfortunately I don't remember how many bits the divider was, either 12 or 16 bits). A small look-up table translated the 16 register-programmable values to the appropriate number to load into the frequency divider. Depending on what state the Envelope Generator was in (i.e. ADS or R), the appropriate register would be selected and that number would be translated and loaded into the divider. Obviously it would have been better to have individual bit control of the divider which would have provided great resolution for each rate, however I did not have enough silicon area for a lot of register bits. Using this approach, I was able to cram a wide range of rates into 4 bits, allowing the ADSR to be defined in two bytes instead of eight. The actual numbers in the look-up table were arrived at subjectively by setting up typical patches on a Sequential Circuits Pro-1 and measuring the envelope times by ear (which is why the available rates seem strange)!

In order to more closely model the exponential decay of sounds, another look-up table on the output of the Envelope Generator would sequentially divide the clock to the Envelope Generator by two at specific counts in the Decay and Release cycles. This created a piece-wise linear approximation of an exponential. I was particularly happy how well this worked considering the simplicity of the circuitry. The Attack, however, was linear, but this sounded fine.

A digital comparator was used for the Sustain function. The upper four bits of the Up/Down counter were compared to the programmed Sustain value and would stop the clock to the Envelope Generator when the counter counted down to the Sustain value. This created 16 linearly spaced sustain levels without havingto go through a look-up table translation between the 4-bit register value and the 8-bit Envelope Generator output. It also meant that sustain levels were adjustable in steps of 16. Again, more register bits would have provided higher resolution.

When the Gate bit was cleared, the clock would again be enabled, allowing the counter to count down to zero. Like an analog envelope generator, the SID Envelope Generator would track the Sustain level if it was changed to a lower value during the Sustain portion of the envelope, however, it would not count UP if the Sustain level were set higher.

The 8-bit output of the Envelope Generator was then sent to the Multiplying D/A converter to modulate the amplitude of the selected Oscillator Waveform (to be technically accurate, actually the waveform was modulating the output of the Envelope Generator, but the result is the same).

Hard Sync was accomplished by clearing the accumulator of an Oscillator based on the accumulator MSB of the previous oscillator. [4]

Ring Modulation was accomplished by substituting the accumulator MSB of an oscillator in the EXOR function of the triangle waveform generator with the accumulator MSB of the previous oscillator. [5] That is why the triangle waveform must be selected to use Ring Modulation.

The Filter was a classic multi-mode (state variable) VCF design. There was no way to create a variable transconductance amplifier in our NMOS process, so I simply used FETs as voltage-controlled resistors to control the cutoff frequency. An 11-bit D/A converter generates the control voltage for the FETs (it's actually a 12-bit D/A, but the LSB had no audible affect so I disconnected it!).

Filter resonance was controlled by a 4-bit weighted resistor ladder. Each bit would turn on one of the weighted resistors and allow a portion of the output to feed back to the input. The state-variable design provided simultaneous low-pass, band-pass and high-pass outputs. Analog switches selected which combination of outputs were sent to the final amplifier (a notch filter was created by enabling both the high and low-pass outputs simultaneously).

The filter is the worst part of SID because I could not create high-gain op-amps in NMOS, which were essential to a resonant filter. In addition, the resistance of the FETs varied considerably with processing, so different lots of SID chips had different cutoff frequency characteristics. I knew it wouldn't work very well, but it was better than nothing and I didn't have time to make it better.

Analog switches were also used to either route an Oscillator output through or around the filter to the final amplifier. The final amp was a 4-bit multiplying D/A converter which allowed the volume of the output signal to be controlled. By stopping an Oscillator, it was possible to apply a DC voltage to this D/A. Audio could then be created by having the microprocessor write the Final Volume register in real-time. Game programs often used this method to synthesize speech or play "sampled" sounds.

An external audio input could also be mixed in at the final amp or processed through the filter.

The Modulation registers were probably never used since they could easily be simulated in software without having to give up a voice. For novice programmers they provided a way to create vibrato or filter sweeps without having to write much code (just read the value from the modulation register and write it back to the frequency register). These registers just give microprocessor access to the upper 8 bits of the instantaneous value of the waveform and envelope of Voice 3. Since you probably wouldn't want to hear the modulation source in the audio output, an analog switch was provided to turn off the audio output of Voice 3.

> Any other interesting tidbits or anecdotes ?

The funniest thing I remember was getting in a whole bunch of C-64 video games which had been written in Japan. The Japanese are so obsessed with technical specifications that they had written their code according to a SID spec. sheet (which I had written before SID prototypes even existed). Needless to say, the specs were not accurate. Rather than correct the obvious errors in their code, they produced games with out of tune sounds and filter settings that produced only quiet, muffled sound at the output. As far as they were concerned, it didn't matter that their code sounded all wrong, they had written their code correctly according to the spec. and that was all that mattered!

--
For questions or comments, Mr. Andreas Varga may be reached through his SID Homepage on the World Wide Web at - http://stud1.tuwien.ac.at/~e9426444/

Addendum by S. Judd, Technical Editor

------+-----+---------+-----+-------

References / suggested reading:

"Design case history: the Commodore 64", Tekal S. Perry and Paul Wallich,
IEEE Specturm, March 1985

"Programming the Commodore 64", Raeto Colin West, Compute Publications.

"Mapping the Commodore 128", Otis Cowper, Compute Publications.

The SID Homepage, maintained by Andreas Varga:
http://stud1.tuwien.ac.at/~e9426444/

"SID Primer: The Working Man's Guide to SID"
by Stephen L. Judd, disC=overy issue 2 (Yep, this issue!)

"Mapping the Commodore 64", Sheldon Leemon, Compute Publications.

"Commodore 64/128 Programmer's Reference Guide", CBM.

The first article makes for very interesting reading and should be available at most public libraries. The second and third references have more detailed and accurate explanations of SID and the theory behind its general features, as well as actual implementation. The SID homepage has lots of technical and general information. The disC=overy article attempts to provide a general overview of the chip. The last two are included as good references for information on programming SID, especially since they are easy to acquire.

The program included with the disC=overy article (wave3) is useful for investigating the SID. By selecting voice 3, and using a low freq. (e.g. a frequency setting of 1 or 2 say) the output of the waveform generator may be seen visually. Ring Modulation, Synchronization, and multiple waveform selection may all be investigated in this manner.

Notes:

[1] In the words of Andreas Boose:

> "The phase accumulating oscillator is just a 24-bit accumulator which is increased by the 16-bit value of the frequency register every phi2 cycle. And like Bob said, the upper 12 bits of this accumulator are sent to the waveform generators.
>
> Note that although he uses 12 bit, the resulting *resolution* of this signal is only 12 bit on lower frequencies, if the frequency register is smaller than 4096. On higher frequencies the resolution drops down to nearly 8-bit when the frequency register is at its max value (65535)."

It should now be clear why all of SID's waveforms are linear in nature (i.e. composed of straight lines): instead of using this counter as an index into a wave table, only the counter itself is used in generating the waveforms (and counting up is an awfully linear process). On the other hand, this raises the intriguing possibility of modulating the waveform via rapid changes in the frequency register.

[2] Asger Alstrup Nielsen has done a good deal of research into SID's random number generator for generating the noise waveform. I am told that the algorithm was implemented by Michael Schwendt in SIDplay and the result was not too accurate. For more information, visit the SID home page, in the references above.

[3] From the IEEE article referenced above:

The precise capabilities of the sound chip are not clear even today, largely because of incorrect specifications having been written when the chip was first designed. "The spec. sheet got distributed and copied and rewritten by various people until it made practically no sense anymore," said Yannes. An example of the faulty documentation is the claim that the chip can logically AND several waveforms. ... "There is no interlock to make sure that if one bit is on, the others are off," Yannes said. "That would have taken too much silicon." So if more than one waveform is selected, the internal nodes of the output multiplexer are discharged, and what emerges is the minimum of amplitudes."

The meaning of that last statement is unclear, as the result is certainly not the minimum of the waveform values either. A very simple way to see what the result looks like is to use the "wave3" program in the other disC=overy article with voice 3. To test the logical ANDing hypothesis, "freeze" the voice 3 pulse waveform output at $FF by selecting pulse and a nonzero pulse width. Select a very low note frequency, 1 or 2 say. When the waveform output becomes $FF set the pulse width to zero: the pulse output is now stuck at $FF. Select another waveform, such as sawtooth, and watch (and listen to) the result! (To hear the result, make sure the sustain level is set (to 15 say) and that the gate bit is set as well). About the only thing that can be said about multiple waveforms is that they are periodic with the expected frequency.

In short, nobody really knows what the result is when multiple waveforms are selected!

[4] It should be clear that synchronization generates a new waveform at a fundamental frequency related to the preceding voice. Moreover, resetting the accumulator to zero will in general introduce a discontinuity into the waveform. Discontinuities will always amplify the presence of high frequencies, as can be seen by taking the Fourier transform of a discontinuous function. Compare for instance the triangle waveform, whose mode amplitudes decay as 1/k^2, and the sawtooth waveform, whose mode amplitudes decay as 1/k, where k is the wave number. High frequencies are needed to make sharp transitions.

Try synchronizing a triangle waveform -- if the synchronization frequencies are very different, the result will unsurprisingly sound like a sawtooth. The higher frequencies are easy to hear; sometimes even the fundamental pitch of the note will change significantly.

[5] Ring Modulation. Most books, not to mention the other SID article elsewhere in this issue of disC=overy, will explain ring modulation as a multiplication of two waveforms, which generates sum and difference frequencies. Simply put, if one signal is cos(w1*t) and another signal is cos(w2*t), then the ring modulated output is:

```
cos(w1*t) * cos(w2*t) = 1/2 (cos((w1+w2)*t) + cos((w1-w2)*t) )
```

Voila! Two new waves, with frequencies which are the sum and difference of the two frequencies. Note that if the two frequencies differ only slightly, very noticable beats will result.

SID works a little differently. What it does not do is to multiply any waveforms. What it does do is generate a new waveform, somewhat related to the original waveform, which contains all the sum and difference frequencies.

Recall that the triangle waveform may be written essentially as:

```
        f(x) = { x,      MSB=0  i.e. 0 < x <= max/2
               { max-x,  MSB=1  i.e. max/2 < x < max
```

where max is the maximum value of the accumulator, 2^24. f(x) then counts upwards on one half of the cycle and downwards on the other half.

Ring modulation uses the MSB of the preceeding voice to evaluate the function above. The normal triangle waveform is continuous, since when x=max/2 the two pieces of the function have the same value. If MSB is suddenly changed, however, the function, which was couting upwards, might suddenly change values significantly and start counting downwards. It is pretty simple to sketch out the result on a piece of paper. Consider the following:

V3: At some frequency a little higher than V1 (waveform doesn't matter)
V1: A triangle wave at some base frequency, modulated by V3.

Modulated output of V1:

```
------------*-------*[---------------------------------- maximum value
            [*     * [
            [ *   *  [
       *    [  *  [  [
      * [   [   * [  [      *   etc.
```

```
     *  [   [    *   [     *
    *   [   [        [    *
   *    *   [        [   *
  *      *  [        [  *
 *        * [        [ *
*----------*[--------[*-------------------------------- minimum value
        ^   ^     ^  ^
        [   [     [  [
        [   [     [  Curve hits FF and wraps around to 00
        [   [     [
        [   [     V3 completes a cycle; MSB is cleared; V1 wave counts up again
        [   [
        [   Curve hits FF and wraps around to 00; MSB is high so result
        [   is 255-value (i.e. counting downwards)
        [
        V3 hits midway point in cycle; MSB is now high; V1 goes from t
        to 255-t, goes downwards.
```

Well it's not really 0..255, but the meaning should be clear. :)

There are now two important features: first, a number of discontinuities have been suddenly introduced, which tends to amplify higher frequencies. The second is that the frequency of the resulting waveform is quite different than that of the pure triangle. What is the frequency? To put it another way, what is the period: how long must one wait before the waveform repeats itself?

In terms of SID, let both of the accumulators start at zero, and let one count up faster than the other. The period of a given voice is the amount of time it takes for its counter to return to zero. At some point both counters will return to zero, and of course the modulated wave will then repeat. This then gives the period of the modulated waveform.

Let one wave have period T1 (frequency w1) and the other have period T3 (and frequency w3). When m*T1 = n*T3, for some integers m and n, the wave will repeat. That is, perhaps after ten repetitions of wave 1 and three repetitions of wave 3 the waves will both be at zero again. The resulting wave will then have frequencies w=(p*w1 + q*w3) for integer p,q. To see this, consider a wave like:

```
        f(t) = e^i 2*pi (p*w1 + q*w3) t,  where w1=1/T1, w3=1/T3

        let t = T = m*T1 = n*T3, then

        f(T) = e^i 2*pi (p*m + q*n) = f(2*pi) = f(0)
```

Thus, such a wave has period T (although not necessarily least period T), for *any* p,q integer. This then implies that the resulting waveform contains all of the sum and difference frequencies, e.g. w1-w3, w1+w3, 2*w1+w3, etc.

Q.E.D.

In summary, SID's ring modulation produces a new waveform containing sum and difference frequencies of the modulating waveforms, with a strong high frequency content due to the discontinuities in the resulting modulated waveform.

Note that by using ring modulation (not to mention sync) frequencies much much higher than than the upper frequency limit (~4000 Hz) of the frequency register may be generated.

/S08::0100h::
:::

--- Z80, The basics of it ---

by Raver/Phantasy

Introduction

This article is a brief overview of the Z80 CPU processor. Made by the people at the American company ZILOG. It was and still is a very popular CPU. Used in many machines, from the early Cromenco's to the later Kaypro's and C128's, in 1996 it is a prime workhorse in many microcontroller operations.

The Z80 first saw action in a Commodore computer with the introduction of the CP/M cartridge for the C64. Later, CBM introduced the C128 home computer, which has a built in Z80. Both Z80 implementations are accurately regarded as half-done, limiting the processor in many ways. Focusing on the shortcomings however, would be the topic of another article. Instead, I will concentrate here on the Z80 itself rather than on how it interacts with any given specific Commodore system or operating system protocol. I hope this will be of sufficient value for further exploration on your particular system configuration.

The Z80 Registers.

AF - can be only used as two 8bit registers, but push on stack and change to alternative always both. "A" is 8bit accumulator, very useful register. Most of operations can be done with "A". "F" is flag register. There are "C", "Z", "P", "V" and "S" flags.

BC - 16bit register, can be used as two 8bit registers "B" and "C". It's usually used as "back counter".

DE - 16bit register, can be used as two 8bit registers "D" and "E". It's usually used as "destination index", as register, who points to address of destination.

HL - 16bit register, can be used as two 8bit registers "H" and "L". It's very special register, there are lots of commands available only for this register. It's sometimes called "16bit accumulator".

IX - 16bit register. It can be used as two 8bit registers "XL" and "XH" too, but it's undocumented feature. That register have one very useful addressing mode, but arn't widely used, cause it's slow (14-23 cycles).

IY - the same as "IX". The difference is two 8bit registers - "YL" and "YH" for this one.

SP - 16bit register. "Stack pointer", points to memory location, used as start of stack. Stack can be pointed to every memory location (#0000 - #FFFF).

I - 8bit register, which points to interrupt.

R - 8bit register for memory regenerating.

PC - 16bit register, "program counter", what points to address, where are next executing command.

! Z80 has two independent sets of registers: "AF", "BC", "DE", "HL". They can be changed by EXX registers "BC", "DE", "HL" to "BC'", "DE'", "HL'" and by EX AF, AF' register "AF" to "AF'". Values of other register set will be saved. Processor don't know, with which one of two sets it works now.

Commands: What moves numbers in registers and memory?

LD A,#04 - moves 4 into register "A"

LD HL,#4000 - moves 16384 into register "HL"

LD (#C000),A - moves value of register "A" into memory location 49152

LD (HL),B - moves value of register "B" into memory location pointed by "HL"

LD A,(HL) - moves value, which is held in memory location pointed by "HL", into register "A"

LD (IY+#03),A - moves value of register "A" into memory location, pointed by register "IY" +3. This addressing mode is only for "IX" and "IY".

Commands: What moves blocks of bytes?

LDIR - very cool command, moves block of bytes pointed by "HL" to memory location pointed by "DE". The lenght of block is in "BC".

Example:

```
                 LD HL,SOURCE
                 LD DE,DESTINATION
                 LD BC,LENGHT
                 LDIR
```

It's fully automatic, but you can do it manualy by replacing LDIR with a lots of LDI - it's much faster! There is another block command LDDR, the difference is: LDIR moves blocks from start to end, LDDR moves them from end to start.

Also, there are commands for searching block of bytes in memory and commands to put in port block of bytes and take from port block of bytes.

The Stack.

Stack is very useful on Z80. It's pointed by register "SP" and can be located anywhere you want, because "SP" is a 16bit reg! (Note : In the 65xx series CPU, the SP (Stack Pointer) is merely an 8bit offset of page 1 ($01xx). This is why the 65xx stack is limited to page 1).

Stack is used for commands like CALL, it pushes on stack return address, which is later used by an RET. The stack can be used as a very fast block-byte mover and buffer refresher, for example.

Commands for stack using are PUSH and POP. PUSH puts 16bit value of registers on stack and POP pops 16bit value from stack. Remember, the stack can be of any 16bit (0-65535) size and placed anywhere.

Interrupts.

Z80 has two interrupt modes - NMI and INT. NMI is a non-maskable interrupt and INT is maskable. NMI defines the main hardware interrupts and generally, they can not be used by coder. Fortunately, INT can be used by coder and register I holds value of interrupt vector. There is IM0, IM1 and IM2 mode of INT. Rather use have IM2 mode. Command to return from intrrupt routine on Z80 is RETI.

Port control.

The commands to control port are: OUT and IN. OUT writes value to port, but IN reads value from it. All controling of music processors, disk controller and RAM pages swithching is done by OUT. But some, for example to check, if key is pressed, you must use command IN. There can be following combinations of IN:

```
 LD A,high value of port address
 IN A,(low value of port address)
```

or

```
 LD BC,full 16bit port address
 IN A,(C)
```

and for OUT:

```
 LD A,value to send to port
 LD C,high value of port address
 OUT (low value of port address),A
```

or

```
 LD A,value to send to port
 LD BC,full 16bit port adress
 OUT (C),A
```

There is more to the Z80 of course (Flags, etc.), but this should get you started. Worthy texts on Z80 ML include works by Rodnay Zaks or Kathe Spracklen (co-author of Sargon chess). These and a bit of hard gumption will yield great rewards with Z80, as with any human endeavour persued at length. :)

Raver
--
For questions or comments on this article, Raver may be reached at the following internet address : raver@10mb.com

```
::::::::::d:i:s:C=:o:v:e:r:y:::::::::::::::::i:s:s:u:e::2::::::::::::::::::
\H01::::::::::::::::::::::::::H:A:R:D:W:A:R:E::::::::::::::::::::::::::::::::
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

Charger up!

The VIC-20 to ATARI 2600 'Ram Cartridge' Programmer

by Ravid Noam (nravid@newton.bgu.ac.il)

A while back, I was searching the internet and found several hundred ROM images for the Atari 2600 video computer system. I also found a couple of ways to transfer the images to the 2600, but none used a Commodore computer as a host. Seeing that my own VIC-20 was equipped with sufficient hardware to become a host, I set about the task of devising a VIC-20 --> Atari 2600 programming interface. This article includes the instructions on how to build it. Be warned that most of the wiring is in schematics (.gif format--two of the .uue's at the end of this article) so you do need to have the ability to read and carry out a schematic. In this text, I will attempt to outline some of the more salient points required to build the interface and thereby ease the process. As with all hardware projects, you undertake this at your own risk. Neither myself nor the staff of disC=overy is responsible for any use or misuse of the information presented in this article.
--

What you need to own:

- Atari 2600 video game.

- Commodore Vic20 plus ram expansion of 8k or more. If you don't have 8k expansion, read the notes about the basic program included in this document.

- Commodore 1540/1541/1571 disk drive, or other model compatible with the Vic20. A tape drive can be used instead, but you have to change the device numb. from 8 to 1 in the basic program running on the Vic. The "8" appears three times in the program.

What you have to build:

RAM CARTRIDGE

It's like a regular 2600 cartridge, but with the rom chip replaced by a 32k cmos ram chip, plus backup battery and some additional parts. The ram cart is plugged directly to the atari like a real cartridge. To load games to the ram, the cart has to be plugged to the main circuit. The ram cart can hold up to 16 games of 2k or 8 games of 4k length.

<Note : Schematic for the ram cartridge is in CART.GIF>

Components description:

- U1 : ...62..256... - the name isn't important. Has to be a 32k by 8bit static ram memory device, with access time lower than 200 nano-seconds and low-power data retention ability, i.e. ability to remember the data when the operating voltage is falling to 1.5V or lower. Most CMOS chips have these capabilities. It has to come in the standard 28 pin dual-in-line package.

32k static ram pinout diagram:

```
            ________
    A14  -|1     28|- Vcc           Vcc - operating voltage +5V,
    A12  -|2     27|- R/W            or 1.5V in sdandby mode.
    A7   -|3     26|- A13                _
    A6   -|4     25|- A8            R/W - read/write select
    A5   -|5     24|- A9             0 - write , 1- read
    A4   -|6     23|- A11           __
    A3   -|7     22|- OE            OE - Output Enable, active low
    A2   -|8     21|- A10           __
    A1   -|9     20|- CS            CS - Chip Select, active low
    A0   -|10    19|- D7
    D0   -|11    18|- D6            A0..A14 - adrress lines
    D1   -|12    17|- D5
    D2   -|13    16|- D4            D0..D7 - data lines
  ground -|14    15|- D3
            --------                ground - common ground
```

- U2 : CMOS 4011 - quad two-input nand gate, 14 pin IC.
  In this circuit and in the main circuit, it can be replaced by a CMOS hex inverter or by NPN transistor + 2 resistors -- implementation for

each inverter. I used 4011 because I have many of this kind.

- Q1 : 2N2222. Can be replaced by any silicon NPN small signal transistor with these ratings : Vce(max) > 10V , Ic(max) > 30mA , Hfe(min) > 100 .

- D1 : 1N4001. Can be replaced by any similar rectifier diode.

- D2 : 1N4148 , or similar.

- C1 : 100 micro-farad , 10V, electrolithic capacitor.

- R1 : 22 kilo-ohm resistor .

- R2 : 10 kilo-ohm resistor .

- R3 : 500 kilo-ohm resistor .

- R4 : 50 kilo-ohm resistor .

All resistors are small power resistors. 0.125 Watt type should be big enough.

- BAT1 : 3.6V Nickel-Cadmium rechargable battery. The same as the CMOS-RAM backup battery inside the IBM-PCs. It can be soldered into a printed circuit.

- S1...S5 : two-position switch.

- PCB edge connector : The ram cartridge has to be plugged to the Atari like a regular cartridge, so it must end with double-side printed circuit edge connector, with 24 contacts, 12 on each side, and with the same size as of a regular cartridge edge connector.

  The Connector can be an old atari cartridge as the PCB connector. Just remove the rom chip and connect the old cart to the rest of the ram cartridge circuit by wires. Using one or two flat-cables instead of 24 seperate wires will give a nicer look.

  Another option is that the connector and the rest of the cartridge can be on the same printed circuit. I know There are EPROM cartridges made for the Atari2600, where instead of the rom they have socket for eprom chip, plus the 4011 chip. The whole ram-cartridge can be built on this platform, saving a lot of wiring work.

- >> Important notes << -

1. When using one of these options, remember to disconnect the ground (contact 13) from the signal ground (contact 12), as the latter one is the read/write control.

2. It is strongly advised that the ram cart is made in a way obtaining only one possible way to plug in the cartridge, Otherwise, every time you plug it to the atary or to the main circuit, you have to be carefull not to plug it the wrong way, i.e. upside down - this can cause permanent damage.

Edge-connector diagram :
(looking at the edge connectors of an atari cartridge, when the cartridge's big label is up)

```
D3   D4   D5   D6   D7   A11  A10  A12  A9   A8  +5V   SGND   up side
 1    2    3    4    5     6    7   8    9   10   11   12
--------------------------------------------------------------- printed circuit
13   14   15   16   17   18   19   20   21   22   23   24
GND  D2   D1   D0   A0   A1   A2   A3   A4   A5   A6   A7     down side
```

Functional description:
(you don't have to read this if the schematic is clear to you)

- CS inverter: the nand gate in the lower-right corner - invert the active-high chip select signal from the atari to the standard low-active signal used on memory chips.

- chip select protection: R3,R4,Q1 - lock the ram when the power is down, ie. when Atari is off. R1 and the CS inverter also contribute to this task.

- R/W inverter: the upper nand gate. Like with the chip select, converts active-high to active-low.

- back-up section : the battery BAT1 feeds the ram and the 4011 when the external 5V is down. R2 recharges the battery when power is up, and pass on the stand-by current from the battery when power is down. D2 and C1 improves the power supply in transition times.

- s3,s4 and s5 : select one of the eight 4k blocks avilable.

- s1: 4k-mode / 2k-mode selection.

- s2: in 2k-mode - select which one of the 2k-blocks is used.

Notes:

Build the cart in a way you can hold it in your hands without touching the metal parts. Shorting something with your fingers can lead to data lost. If the circuit seems too complicated, there are some ways to simplify it:

- The rechargable battery. Can be replaced by a cheaper non-rechargable battery, and the resistor R2 has to be removed, to prevent charging. Remember that the 4011 chip must receive minimum 3V operating voltage, and plus the voltage drop over D2, we get minimum voltage of 3.6 that the battery must supply. Battery that consists of three 1.5V (=4.5) cells is applicable, but isn't a common thing. In theory, a 3.0V or even 1.5V battery can be used, since most of the cmos ram chips can retain their data under 1.5, 1.0 or even 0.7V stand-by operating voltage. If the problem is with the 4011, it can be replaced. Each one of the two gates used in this circuit can be replaced by a NPN transistor and two resistors, and this combination should function under 1.0V or even 0.8V .

- The chip select protection circuit: Q1, R3 and R4. Before the 3.6V battery, there was 2.4V battery. After some months, the cartridge became forgetful. Then I added these parts, but the problem didn't get solved. Only when the battery was replaced by the 3.6V one, the ram cart returned to function properly. It's probably the 4011 which decided to demand the 3.0V. Now, R3,R4 and Q1 seem useless, but I have decided not to remove them, because the data retention issue is too sensitive from being economized.
  If you want to check this, remove them and see if the ram cart still function. The CS pin of the ram has then to be connected directly to the output of the nand gate.

- s3,s4 and s5 : If you use standard pin-compatible 8k ram chip instead of the 32k ram, you don't need s3 and s4. But then the cartridge can hold only two 4k atari games or four 2k games. s5 can be replaced by shorting A12 of the ram to ground or to +5V, but then the amount of data that can be stored on the chip is divided by two.

- s1,s2 : After selecting the 4k bank by s3..s5, s1 is used to select between two options: one 4k game or two 2k games. In 2k mode, s2 selects between the two 2k banks. s1 and s2 can be omitted if you load the 2k games as 4k games- you load a 4k file which it's last 2k half is a copy of the first 2k half. In this case, the A11 pin of the ram should be connected directly to the A11 incoming signal from the atari.

THE MAIN CIRCUIT

The task of the main circuit is to load games into the ram cartridge. Also, it can read to save the content of a real cartridge. The circuit is connected to the Vic20's user port.

<Note : Schematic for the main circuit is in MAIN.GIF>

[Ed. Note : Assuming the VIC-20's user port is similar to the C64's user port, then my guess is that this portion of the project could be adapted to the C-64 with some minor hardware and software revisions. Any takers?]

Components description:

- U1,U2,U3 : CMOS 4094 - shift register, 16 pin IC.

- U4   : TTL 74LS165/74HCT165/74LSHCT165 - shift register, 16 pin IC.

- U5   : CMOS 4011 - quad two-input nand gate, 14 pin IC.

- Q2,Q4   : 2N2222. Can be replaced by any silicon NPN small signal transistor with these ratings :
  Vce(max) > 10V , Ic(max) > 30mA , Hfe(min) > 100 .

- Q1   : 2N2907. Can be replaced by any silicon PNP small signal transistor with the ratings of Q2 and Q4.

- Q3   : 2N2905. Can be replaced by any silicon PNP transistor, with

these ratings :
Vce(max) > 10V , Ic(max) > 300mA , Hfe(min) > 50 ,
metal package (TO-3,TO-5 etc).

- LED : any common small (2mm size) visible LED. The LED and its resistor R9 are optional.

- R1,R2,R3,R6,R8 : 22 kilo-ohm resistor .

- R4,R7 : 1 kilo-ohm resistor.

- R5 : 220 kilo-ohm resistor.

- R9 : 5 kilo-ohm resistor.

All resistors are small power resistors. 0.125 Watt type should be big enough.

- Vic20 User Port interface : The user port in the Vic20 is a double-side printed circuit edge connector, with 24 contacts, 12 on each side. The interface is a female connector in the same size which can be plagged on it, and contacted to the main board by 8 wires.
  Be careful not to plug the interface to the Vic20 the wrong way, i.e. replacing the top side with the bottom side. Such a mistake can damage the VIA-6522 on your Vic20 or some parts in the external circuit. Write "top" on the top side of the female connector or, if you have time, you can shape two little plastic parts and insert them in the female connector, against the two a-symetric slots in the user-port edge connector. This will prevent you from doing such a dangerous mistake.

- Atari2600 cartridge interface : exactly like the one in the Atari - a female PCB connector with 24 contacts, 12 on each side. like in the case of the VIC's user port, it is recomended to ensure that there is only one way to plug in the ram cartridge or the regular atari2600 cartridge.
  The easiest way to obtain this interface is to take it out of old Atari video game cart. Then the interface can include its plastic housing which eliminates the possibility to insert a cartridge the wrong way. You can use the entire case of the atari cart. to hold the main circuit inside.

Functional description:
(you don't have to read this if the schematic is clear to you)

- U1, U2 and U3 : three 8 bit serial to parallel shift registers, which spreads the serial signals from CB1 and CB2 to 20 bits word - 8 bit data out + 12 bit adrress.

- U4 : converts the parallel data byte read from the cartridge to serial data sent to CB2 under control of CB1.

- chip-select inverter - the lower nand gate inverts active-low chip select signal from PB0 to active-high one needed for the Atari cart. Although PB0 is software programmable and can produce active-low signal, the default position of it (i.e. after turning on the Vic20 or hitting RUN/STOP+RESTORE) is high. Also this gate isolates the vic from the plugged card - you can never know what you plug in, so this method of operation is prefered.

- Q4, R5 and R6 : Q4 and R5 enables U4 to send data to the serial line CB2 when PB7 is in read mode. Otherwise CB2 is used to send data to the 4094s. R6 is pull-up resistor for data coming from U4.

- read/write inverter: the upper nand gate - similar to the chip select inverter (default position,isolation).

- R7, R8: this voltage divider protects the nand gate from overload when PB7 is in write mode and a real cart is plugged. R8 is also pull-down resistor for the read/write in the ram cart.

- power control: Q1, Q2, Q3, R1, R2, R3 and R4 amplifies the 0.2mA control current from PB5 to the current consumed by the main circuit and the cartridge.

Notes:

- The LED isn't necessary, but it helps you know when the main circuit is turned off - when it's on, you are not allowed to plug in or out the cartridge.

- The ground sign on the schematic doesn't mean physical grounding. It means that everything ended with it is connected with each other. It can be really grounded, but that depends on your Vic20's power supply.

- The 4011 can be replaced. See the notes about the 4011 in the ram cartridge section of this document.

- The power control can, theoriticaly, be reduced to R1,R2 and Q3 only. The base of Q3 is then connected to the hook up of R1 and R2, instead of the base of Q1. But the current amplification of Q3 has to be high. We don't want to overload PB5 and, Q3 has to be in its saturation area in order to output more than 4.8V, under load currents of 50mA, 100mA and more.
  If we don't chage R1, then I(PB5) = 0.2mA. The user port of the Vic20 is permitted to supply up to 100mA. Thus Hfe(min)= 100/0.2 = 500. Too high. The nominal active current of the main board and the ram cart is less than 50mA. Now we get Hfe(min)= 250. Still can't be obtained from every 2N2905-type transistor, according to the manofacturer data. But you can try it, it will probably work. A darlington transistor isn't suitable because it will give 4.3V output, not enough for U4 and the memory chip on the cart.
  One transistor can be saved if PB5 turns into active-high instead of active-low. But then the circuit is automaticaly turned on when the VIA6522 in the Vic is being reseted.

The software part:

The program which controls the ram-cart loading and the cartridge saving is a 3.5k Basic program, running on the Vic20. It uses 0.5k machine language program.

The program files are:
- ATARI14/C the basic program.
- ATARI MCP the machine code program.

[Ed. Note : The uuencoded versions of these files will extract with these filenames. If this is a problem for your host machine, please edit the filename field in the uuencoded versions at the end of this article. Please keep in mind that you should rename them (as seen above) on your VIC-20's drive. Such action will prevent potential load-chaining problems.]

The basic program is written for using a disk drive. If you have only tape, change the "8" device number in the basic program to 1. It appears three times: When loading games, when saving cartridge content and when loading the machine code program.
Every time the program is runned, a checksum is made over the m.code program in memory, and only if the checksum fails, the m.code program is reloaded. Something i forgot - after reloading, the m.c. program isn't being checked- so it's downloading from the PC has to be done carefully. To check it, load first the machine code program with ",8,1" or ",1,1", type "NEW", and only then load and run the basic program. If the m.c. program isn't reloaded, then you know it had been downloaded properly.
If the ram expansion on the vic is non volatile, the mc. program will remain in memory when the Vic is off. In the case of 16k+ ram expansion, the basic start can be raised to $4000 and then the basic program becomes non volatile too. In this case you have to change the memory settings in the program's first lines.

The current memory configuration:

```
  $1000  4096 - $11FF  4095  : screen bytes
  $1200  4608 - $2DFF 11775  : basic program + variables
  $2E00 11776 - $2FFF 12287  : machine language code
  $3000 12288 - $3FFF 16383  : 4k game buffer
```

It's possible to cancel the 4k buffer by transferring data between the disk and the cartridge byte by byte, and thus to compress the entire thing to an unexpanded Vic20. But It's not simple and limits the program's ability to grow up.

Operating instructions:

SAVING-CARTRIDGE CONTENT

1. The main circuit has to be plugged to the Vic20's user port, before the Vic is turned on.

2. Run the basic program, if it is not already running.

3. Plug in the Atari2600 game cartridge.

4. From the main menu, select "copy 2k" / "copy 4k" with or without check. Before selecting, check that the cartridge is plugged in. DON'T plug it in or out when the vic is reading it. It is avdised to connect a LED (+ its resistor) to the main circuit, to see when it is powered on. Only when

the power is off, the cartridge can be inserted or removed.

5. If you selected copy with check, the saved file will be reloaded & compared with the cartridge content.

6. The cartridge content is now saved in a file. The format of the file is 2 bytes - low and high byte of the start adrress of the buffer, followed by the content itself - 2048 or 4096 bytes.

7. Now you will be asked for continue. If yes, the procedure repeats itself in the same copy-type selection.

Notes:

- 2k games can be saved as 4k games, by choosing "copy 4k" on the menu. The 4k file following the start adrress will be a double image of a 2k game.

- The saving proccess can be done with no cart plugged in without damaging anything. The saved file will be full of 255's ($FF).

- You can hit RUN/STOP + RESTORE anytime and the main circuit will be turned off.

LOADING GAMES TO THE RAM CARTRIDGE

1. The main circuit has to be plugged to the Vic20's user port, before the Vic is turned on.

2. Run the basic program, if it's not already running.

3. Insert the ram cartridge, if it isn't already plugged in. DON'T plug it in or out when the vic is writing to reading it. It is avdised to connect a LED (+ its resistor) to the main circuit, to see when it is powered on. Only when the power is off, the cartridge can be inserted or removed.

4. Select the desired switch combination on the ram-cart. Check that it is not the combination of an already loaded game, which you don't want to be overwritten.
 If loading 4k game,move s1 to 4k mode. For a 2k game, move s1 to 2k mode and select (with s2) one of the 2k banks. s3, s4 and s5 selects between 4k banks, and gives 8 combinations.

5. From the main menu, select "load game". Type the name of the game and it will be loaded. No disk browser at this stage.

Notes:

- When loading 2k games which have been saved as 4k games, move s1 to 4k mode.

- The loading proccess can be done with no cart plugged in without damaging anything.

- You can hit RUN/STOP + RESTORE anytime and the main circuit will be turned off.

```
begin 644 atari14/c
M 1(<$@$ CR!3059%(D P.D%405)),30O0R(L.  O$@( ES,V.#<Y+#@ZF2(%
MDR( -1(% (\ 6!(+ (\@*BH@051!4DD@0T%34T545$4@34%.04=%4B J*@!>
M$@P CP"%$A0 ES4R+#0V.I<U-BPT-CJ<.H\@-TL@4D%-($9/4B!"05-)0P"0
M$C( C2 U,#8P ,@2/ "+($*R,2"G()DB15)23U(Z($-!3B=4($9)3D0@34%#
M2$E.12!#3T1%(%)/551)3D4B.I  U1)& $U#LC$Q-S<V .T24 ">($U#JC$U
M.H\@4$]715(@3T9& /,27P"/ /X2O@")(#(W,#  #Q-> 8\@4D5!1"!"3$]#
M2P N$W(!ER X,C0L3$\ZER X,C4L2$DZCR!33U520T4 2A.& 4A)LK4H5*TR
M-38I.DQ/LE2K,C4VK$A) &D3D &7(#@R-BQ,3SJ7(#@R-RQ(23J/(%1!4D=%
M5 "'$YH!2$FRM2A,3JTR-38I.DQ/LDQ.JS(U-JQ(20"F$Z0!ER X,C L3$\Z
MER X,C$L2$DZCR!,14Y'5$@ R!.N 8\@0T]062!&4D]-($-!4U-%5$4@5$\@
M345-3U)9 -,3N &>($U#JC8 V1/" 8X ZQ,F H\@5U))5$4@0DQ/0TL "10Z
M DA)LK4H042M,C4V*3I,3[)!1*LR-3:L2$D *!1$ I<@.#(V+$Q/.I<@.#(W
M+$A).H\@4T]54D-% $043@)(2;*U*%2M,C4V*3I,3[)4JS(U-JQ(20!C%%@"
MER X,C0L3$\ZER X,C4L2$DZCR!405)'150 @11B DA)LK4H3$ZM,C4V*3I,
M3[),3JLR-3:L2$D H!1L I<@.#(P+$Q/.I<@.#(Q+$A).H\@3$5.1U1( ,,4
M=@*/($-/4%D@1E)/32!-14U/4ED@5$\@4T4N(%!/4E0 SA2  IX@34.J,P#N
M%(H"F2 BG2(@3$X@(D)95$53($-/35!,151%1"XB /04E *.  @53 2/($=%
M5"!&24Q%($Y!344 (156!(L@3DTDLB(BIT(DLB(B.HDQ,3,P $D58 1")++(
M*"*=G9V=G9V=G9V=G9V=G9V=G9TB+,,H3DTD*:HR*0!H%6H$F2)&24Q%($Y!
M344@(" B3DTD0B0[.H5.320 >15T!(M.322R(B*G,3$Q, "L%7X$3++#*$Y-
M)"DZBTRQ,3:GF2)-05@N(#$V($Q%5%1%4E,B.DY-)+(B(CJ),3$Q, "R%8@$
MC@#)%9P$F2)/550@3T8@4D%.1T61(CJ. . 5L 2/(%)%4T54($1)4TL@1%))
M5D4  16Z!)\R+#@L,34ZA#(L0BQ")"Q!+$$ZH#(ZC@ 7%CP%CR!3150@1DE,
M12!005)!345415)3 "(61@6!0K(QI$P /!90!9<T,#DUJD(LQBC**$Y-)"Q"
M+#$I*0!"%EH%@@!<%F0%ES<X,"Q,.I<W.#$L,#J7-S@R+#$V '<6;@6>-C4T
M-CDZCR!3150@1DE,12!.04U% ) 6;P67-S@P+#$ZES<X,2PX.I<W.#(L, "R
M%G %GC8U-#8V.H\@4T54($9)3$4C+$1%5B,L0T]-32, N!9S!8X PQ9X!8\@
M3$]!1 #.%H(%C2 Q,S0P .D6R 67-S@P+# ZES<X,2Q,3SJ7-S@R+$A)   7
MT@6>-C4T.3,ZCR!,3T%$($9)3$4 &A?4!45/1K+"*#<X,2FJ,C4VK,(H-S@R
M*0 @%]<%C@ K%]P%CR!3059% #87X067,30T+#  4A?F!8TQ,S0P.H\@4T54
M($9)3$4@4$%204TN &P7\ 67-S@L3$\ZES<Y+$A).I<W.# L-S@ C!?Z!4A)
MLK4H14]&K3(U-BDZ3$^R14]&JS(U-JQ(20"@%P0&ES<X,2Q,3SJ7-S@R+$A)
M +<7#@:>-C4T.38ZCR!3059%($9)3$4 O1<8!HX VA= !H\@5T%)5$E.1R!&
M3U(@2T59(%-44D]+10#Q%TH&F2(1'1TM2$E4($%.62!+15DB  485 :A020Z
MBT$DLB(BIS$V,C  "QA>!HX &1A\!H\@34534T5'10!;&($&F2(124Y315)4
M(%1(12 24D%-DB!#05-3151%04Y$(%-44DE+12!!3ED@2T59(" @(%=(14X@
M4D5!1%DN(@!E&(8&B3$V,C  B1BB!H\@("!>7EX@($5.1"!/1B!354)23U1)
M3D53("!>7EX CQBC!H\ M1@T"(\@3$]!1"!!5$%262!'04U%(%1/(%)!32!#
M05-3151410#$&#D(F2*3(CJ-,38V, #.&#X(C3$Q,#  X1A#")DB3$]!1$E.
M1R B3DTD /T82 A,3[(P.DA)LC0X.HTQ-# P.H\@3$]!1  '&4T(C3(T,#
M%QE2"(M%4K(P(*<R,38P # 95PB9(E1262!!1T%)3B _("A9+TXI(@!&&5P(
MH2!!)#J+022R(EDBIS(Q,C  5QEA"(M!)+(B3B*G,C<P, !A&68(B3(Q-#
MBQEP"$%$LC$R,C@X.E2R,#I,3K)%3T:K042J,3J--34P.H\@5U))5$4 FQEZ
M"(TQ-C P.HDR-S P + 98 F/($524D]2($A!3D1,,24Y' +X990E%4K+"*#<X
M,"D [AEO"8M%4K(UIYDB15)23U(M(CJ9(D1%5DE#12!.3U0@4%)%4T5.5"(Z
MB3(T.3  !1IT"8TQ,C P.H\@4D53150@1$E32P =&GX)BT(DLB)/2R*G15*R
M,,#J),C0Y,  U&H@)F2(115)23U(M(CJ90B0Z15*R0@ [&KH)C@!3&HP*CR J
M*BH@34%)3B!-3U5+5("HJ*@!Q&I8*F2*3("HJ($%-405)9($-!4U-%5%1%("HJ
M(@"'&I@*F2(1$1T2,9(@0T]]062 R2R( IQJ9"IDB$1(@0T]]062 R2R!7
M251((%-44DE+12!#3U)5(%-)7!L3U0@,2!/4B R2R!&3$]04%D@1$E32R!7
M3U!!9(#1+(%=)5$%#0D$@4$%34T5.5"$@*#$R-5ET(T$Z,"@T55)%(%-54RD
M"IDB$1(=49(@455)5"(  (!N>"IDB$1$@("!93U52(#$E,24K($%!4"!#5$U%
M3U(I.3)%4D1%1R!)4" W"@0,1E4@2T1!1"TS3U(V*%EL.C9C3R@R2U)3(/ P
M0@"@0#$+9B!$&$T@8M%5$(!3U!53%-315)44T0I,#X3.$D!(3)%4%)A.4$@
M10@C1!.3U)9C@H-4$-%+51,03%6,2E!4$P+11%26;E,.A&3!B3,P-S%-;*3U
M,T4!&-%-D,#4$-0@#-5%2:"%U,&(R;3%12$P3U1,"$13$%2,$E5(&\M+%,4,5
M,T!,0C8(5!&1U1,41!4$X-&25R\%1"%D-%3A9#)))C0X,0)%05!&1RQ.$2(X
M-T5,10$4Q%3,E@#4",E,!5X-UQT-DQ$$3!"H-T,T&3A9"*$P3-3U)CM",$0"
M,,-$-5!%5%8R/1)-3,4$!AT-4$UE%5A)3$53'$,9D&(@!@!G3!F)0E!B!K(@
M,0Q:+J3-A%-U7#1,/1$!$N&F$T=(0D$#$)@$Q14$%<M1%E-5%$P,$A)$SXD
```

```
M12!005)!345415)3 "(61@6!0K(QI$P /!90!9<T,#DUJD(LQBC**$Y-)"Q"
M+#$I*0!"%EH%@@!<%F0%ES<X,"Q,.I<W.#$L,#J7-S@R+#$V '<6;@6>-C4T
M-CDZCR!3150@1DE,12!.04U% ) 6;P67-S@P+#$ZES<X,2PX.I<W.#(L, "R
M%G %GC8U-#8V.H\@4T54($9)3$4C+$1%5B,L0T]-32, N!9S!8X PQ9X!8\@
M3$]!1 #.%H(%C2 Q,S0P .D6R 67-S@P+# ZES<X,2Q,3SJ7-S@R+$A)   7
MT@6>-C4T.3,ZCR!,3T%$($9)3$4 &A?4!45/1K+"*#<X,2FJ,C4VK,(H-S@R
M*0 @%]<%C@ K%]P%CR!3059% #87X067,30T+#  4A?F!8TQ,S0P.H\@4T54
M($9)3$4@4$%204TN &P7\ 67-S@L3$\ZES<Y+$A).I<W.# L-S@ C!?Z!4A)
MLK4H14]&K3(U-BDZ3$^R14]&JS(U-JQ(20"@%P0&ES<X,2Q,3SJ7-S@R+$A)
M +<7#@:>-C4T.38ZCR!3059%($9)3$4 O1<8!HX VA= !H\@5T%)5$E.1R!&
M3U(@2T59(%-44D]+10#Q%TH&F2(1'1TM2$E4($%.62!+15DB  485 :A020Z
MBT$DLB(BIS$V,C  "QA>!HX &1A\!H\@34534T5'10!;&($&F2(124Y315)4
M(%1(12 24D%-DB!#05-3151%04Y$(%-44DE+12!!3ED@2T59(" @(%=(14X@
M4D5!1%DN(@!E&(8&B3$V,C  B1BB!H\@("!>7EX@($5.1"!/1B!354)23U1)
M3D53("!>7EX CQBC!H\ M1@T"(\@3$]!1"!!5$%262!'04U%(%1/(%)!32!#
M05-3151410#$&#D(F2*3(CJ-,38V, #.&#X(C3$Q,#  X1A#")DB3$]!1$E.
M1R B3DTD /T82 A,3[(P.DA)LC0X.HTQ-# P.H\@3$]!1  '&4T(C3(T,#
M%QE2"(M%4K(P(*<R,38P # 95PB9(E1262!!1T%)3B _("A9+TXI(@!&&5P(
MH2!!)#J+022R(EDBIS(Q,C  5QEA"(M!)+(B3B*G,C<P, !A&68(B3(Q-#
MBQEP"$%$LC$R,C@X.E2R,#I,3K)%3T:K042J,3J--34P.H\@5U))5$4 FQEZ
M"(TQ-C P.HDR-S P + 98 F/($524D]2($A!3D1,,24Y' +X990E%4K+"*#<X
M,"D [AEO"8M%4K(UIYDB15)23U(M(CJ9(D1%5DE#12!.3U0@4%)%4T5.5"(Z
MB3(T.3  !1IT"8TQ,C P.H\@4D53150@1$E32P =&GX)BT(DLB)/2R*G15*R
M,"D [AEO"8M%4K(UIYDB15)23U(M(CJ9(D1%5DE#12!.3U0@4%)%4T5.5"(Z
M?3-S%WB+<C*,H6554SV,B&.M(X-0?;"><7I 2F=6JFN:6BJ<D\7&S,W.S\88
M -, 2@4 U]<:U ! U]1&UT;8TR_@"]\%#.F!!M3J).\UT]\*W.+DV.'=,@ #
MU"'H\7.7+Q"W!>4(E+-'#F%!=.<(9%O(C9^K7$?4=;#X#O^%N",#I;')5PVB
M/H<+!9!4X0\@)HH:[14(4^9@.V@X<^K<R> D-@X+"3*4UDWBP(3N!'U4:9&A
M4Z%0>\[LYU0A/XH#/T(-BG3AQZ53IQ%(P_5I.:89KW9C2A3ATWDE@RYS('>H
M@8_ZVA(-,;5G47]FU08&.>"IUDI8X:Z%X77@7)Z0(TOF@O>HNKUWXPINNR9Q
M ZR<Y6(XX#BP7G%MMVV..DV)9P\IZ99&J[* :J=HMX;LM[AJ71H_Z@$!)^ :
MY]SNM!8=)UCWWMR>6R-'W#PM%8<R;T[>SKU[K-J<,P/=G/"?PQO0N8T/K5EV
MYYFU>6>WFUR&<@!BOJKM9;P3]L__[=T5%GS_2&,;0<&5]%]_URVCTC_\9!/6
M:ES1 U(=T0686'KV15C:%YXQI,A_8GEGXHDGKE!12<BMR%!#_ZVC8(:#1850
M7\-5]*)<%9[S5G0S02C3;]-U.-&.6'TS'D%JW0?C!CJ:4]Q!6-G$&I-) >AC
MAC9%J=M+U0E%Q&LHEMD%)5 0\Y@7/ATU8W.OU874EW2&B(%1[M7IYI)0M0CG
MGSFT!N6?[+$W&"!V4J 0;\U5QJ<[KOE((J!ZUO@;FYK!]J&A@ICIZ9D-1I+'
MFJ.BR<JD,=*(X%OU^56H<S;*Q$>EK+E)7TH9<CK(24((ZNM^SL4G9G\<%'#9
MI1+,N"2'__)MP\(TA762*ZT;AD@=JZ(--^$HGW:[Q46):#=B!*6X8"H8WE!8
M#:<A\NK/B*JJJJE;U"XF':RMULMJ30K"4YFE'N+VJQX#EX%';Y-BP*"@65;E
M%\ 0TVK)64V)*9^8/WFK\1.V;,!)@^"V<RX4"=V7X\7M9B60FWF]>-F.$-C$
M;,FVRN?G<! JR*1ED@HB4 V.P@I3/"[MC!^^%O2V+$RI?D97SII6&'%+#$\,
M[5CS8D?44AMW'<L''W_]""RD>I',J&2?FDO8'>O@BQ-JIRWWW+@88G >HJ29
MPS"2-+%*(Y]4,K(62+S0MM>(IP!43U$T^;"!')UER8B>\. @VO^]8&1W'7]T
M>B<@Z,%0.0Z_;%Y,YWYTJAW:-Q%AN,AOBQIJY;OXS3K@=) ]N"Z:?PW([HES
M%T!/L $><REA]3J/1(O3S;OST,.M.ZBHZ_TUQT[L CSV;E]O@NN=;Q\\].*/
M'YE*69Y#D5!R-FS/< WC",DW*]9O__TDX:^__NGL[___^:,?_0!(P (:L"(#
M/.#]$DB2 ?;O'1!4X(H$*,$*YB.!#.R9^;K6C:.8HT_N6PO][#./NZ#C3JP;
MF<(LR,(#9K"%,(RA#&=(PQK:\(8'W&#7BG60XHB!',5QVON&F!2+J,.#H8J>
M&SQ7IC;HT &%>Z(4GS&K*6KL:.YPE@G_T:>S](&@+]UX&7YB$I//*3%NGBK?
MI]1HQ39^RXUIU CZ[$&4#M;$:(QZUD*LLB[5O?&,D\D<'($QR$*:C9!L-.0E
M&!'%7] $=G=;TR.SD$A,"-*-(5.D)D&UR?.YK05C@]NX8B>(T772>(/,Y"E7
M";=*LO(+H=.>Z4!!2LQMHELXS.7_^O?*7EK E;Y\0L7H-41B**AXB,A8V2H
M3,&M0Y,?"J8T5S=-[I5MCF[;S>&$&(O=I&Z0T>P6$5RG /DULIKH3&<%I 8I
M<=2C@_"4B49@%!0#)4<M5SF64(13(!%EYYR("Z>G%$%.$#AFDNI,J$(?<!7B
MF:=YL;$CI-8B_]$^TA$^%,W(^[@VD&U(I)S,<]\3!8HB*YB"CLM<J$JKB8$!
MJ.@GRQDBPJI1C47%96=0\F+Z+'3,^)6C)*JIVDBAV<R5&M6-#<VHSE0B$L9=
M!7U<C)D/O3C'IM+KIFLI#,*F2-*23O(.+(B)O;!XU+)*LS]\O,L9P@B'(!+D
M0!:R6 D;JL7U?=$D3[$*4E-YPE9UU:R #28* %I%(F9@#:#\YBL :;Z_GJAV
M:#M*'0$:V,I:,7</")T.D'4[LSK6.QXDC@'^@2,D1LBRJ$TM=XHJF<]V)[0@
MA53#/ C58:KVMKB]A!A8.QG7!C(I(K&76[,(W(84"6^Y3:Z9=,G<^O]%BYKC
M\^UOPWBA(PQ@4>_#;B"*0(Z8*/>[X(W ;-HHW>U4U"W+P>>*<B:0 :8TO/!%
M[<O F3C3RI28CGAD/(03W_Z"M[Q> [ GP:A,( )M<>?D8TS]R^#;>A>. H8,
M+8H!KQ7>Y!$NA6Z#-[S2"%\1FFXYA#<Y3&+/*M+#MF-39<"1C\4U-6PA+K&,
M36Q(%/?NO=_C1P#$.@^(>@,$>6N _&9,9(7:^%-'IJ4-V+@HFA#@N<N9YXNR
M>0[>%OG* 3ZQ,TGWA2D=S5D:.))AE7&"$>(8RVB&L)8%EY42=OFZ1.Q@&%Y8
M/_?Z,<UX[F22EPN)YLV1LSFF5P#L^[[<!4'_'GE.]"KW7*8]J\C/#\Y"T8B+
MOKI"-*ZW>(B5%<WIWJX9,<2T\;G"0-$(':S3J!;GWNCFH4T[XR$'Q<(,)&M8
M:)B6LPM.M:ZWTTA21/)]9-U@0V6;N5#&[<EN%BDDC/NSV5(AU_UP<B9<O6L&
M@_7.A\[,9:C-C+5HMW1W+D%:DPUJ9S6EJM%Z*BZZ6.UV[^1WTB.;G(<K;/S6
M#7ACHZRD:^#2F4X9A!O8=A(/Z>Z"N^&Y_6A!L$$Z3VX;(R3\""JW(BT]..B;
M"QY4TH7( &^Z\!<:PS.XP7O#*SP9+$R-+5Z<'6*J,6PZO5"%BP/O,<&5T7F&
MAQ'YKNW($L]%\T"8_QV?"L#8XZARC]O"[6*N30FY^3JC. [7N645QE3X,<[C
M"&RN+NWL[&%[RB4%P6;=A& 496IBUE*O-M=W,]Z>/I$12$S*1-C-06'NQ+;H
MRL(YHYYV/EMC>5>? CPEM[%>\ J;=CP05<[,C&V&.M8,?7I.H_"#BX-M>GU'
MYXSHU;PX''%XC,8)4%RAD6WD?*N>[-T/AHUZ;C:C7' .W*]AW-FUR3[SP<1L
MD ,?>BKVQ#S#\<2WO0/CW5]=W5;H?0DV=6^0,1-P-S ^[ED9Q<,I7ODYN2D1
M#\?W%&#:B8__*_9U$ _O6<^J;ZT&?[L__>Y4E8A&<?J?-8SDE<N<-Q^/#/]'
MN!SY=5S2WL^@3<"Q?0:U3BI75XS5?E,T>C?B,_!3/$V&.*NW&(B'7^-W!=<W
M)-S"30R3?#H14FOR4:0C4"IB ][51WBG@)LD4?8G4B-F/JO'>7%G7CU6,=QE
M=Z_V0I'64F2Q?'[A=72G@HUW.10'2(=01(%'1ORB=38T<_43:B]R+F"D#>74
M/O3G;#9R: _D/Q2T0"]H J@73LP7A#$3>-@T9-3$?JB&0G7S(9!E, BU*^IA
M51&G= 6U1J'B:_;R@*U0A R%?$?$AUQC@&.D/+%3'"&G60Z@>(6 ;(PG>!\4
M#,^T;A-P XY!<K%U.D*8"2CP$W541+813WU2#P+_("1R8$10B%)7Q2*%]U)E
M015A!((I"(E_@5X((0U5IV$K=#0:Q(&MYWJ%@&+Q,(RR(2G0DBT-$ #C 5/@
MH"07$8>TMXG6%(A%9"M-94<_Q']T877$]6^=0E.+-SXQ47XR]T. QCC8M7\[
M4XI4T5:$8$)0M RUF"4"$$4QI0H1=PPS$4Y ,2XBR MP>&$W%FYJ6' L!FPN
M0'+G8#*-A RV!Y# %F8P&!'P&'B*Q5 CU&;CD'X',DK?,7 6J6S;6 )HV 07
MY"7!5PSL)G;2MUWFIS32N E)%5Q?!!N,%$+1(PHZJ3G8I7&)XP-(%&2^)CM9
M D8JQX#T9XQ*1E8=%0%E_U2)2W4%Y)B0%O@L_=<Q&LE,OE)[,<E,RR@#0L**
M/043^$22,I*/,!!FR."'9J)PA>8^_-42E(5"<,84=:AXZ/ ]ER%'T#6#R?)_
M#H!PQ_ A\TB.T=1%QI84H60YB^B 5]B5>?>(8^>#+>@?[4AXT05Q4+B'E+ '
M.V"2\,.,8@@S$P!YX@4BVTB8P$4#)*<1U^!2=JDG->-LE@>9OQ1O9G->JEB&
M7%EXAKB:!^5TD(%%2A6.Q-.;M7:*JW,NA'F8-AAC6'6-BQ.=..EL@&F;*59.
MGFD&"5A^0&<E,<,_3 A#&>0S?J96DBDX/(:$(EES^5-S-T</=!:-O.!=N\@K
M3/^"&M^8E^PY;$\V?QF5'/"085,5;MC9-^Y31;4) 5QG7:59-[O5->-"6UA8
MD-OX4<-S7;"Y52"()L_I?1<@G$M$3.!#81Y#:X6H4W+G40TU$PI!AP?J3-E
M1[5E'@M:@"\&*0?3>0P"DI)1.)=X)_NHF!<H"=]749;S)NMTD=Q$!EH4">=Y
M"B/@1(M1&2>X>(*B3T_F$ZIHH08'4>S(GC?*@1T2=/QI+-*P#7%H)DPQ/!*)
M$D-"A=O1<;=W7]IECQ[9?Y9X30%GIFGSC,HFHKX0 I7W&2CDI3HWDW=Q&TST
M:CVFF&SI?D(0B"YFD7Z:$R&S!T%4+'WIHR%)HH?%>4S_RGAB%PC&)DA1-*8Q
MN@:#AI#5V3?*D%C]!ZEOF8 \825NJ(GFA1=)" =L1RJFR:!I\XM5X!C7UDH=
M 'X>XP*$A4JKF@7UH!K[>(Z4J8$/>HW)F2+RMI$.<:6O98M7E34!FBSO-9JW
MV:#:^'QQP#=Y5SW/ZBUQ1X8R92U9]F/MF#Y3H:J6@"/?QH)%]"CD*BKFJF0R
MZ#U2BJCO.E!RII@,RYMI"8.I.%Q%2C)WE"/(A U.2&=S9S]2-JNKQJ,^JEGZ
MFK!Z-ELY]P )!"'Q.9X*Q%[V.EN)>7>'VA0296X+5RY7R09\\:2N9RYEB'[D
M$H\(2[*/I2/J* 6(64KC$W(5__B6Q(JIL&6OE\@RP:A89.!U/1MVL3E<4W$V
M$$FT1P5;0+N(]?BGM@I:RZ,.8C6T?G$L+-9YE>:7=#%*GH"U(1EQ:AN:WJ!@
M"/(DO0BV"65:Q'*:=S.ROZ5%GWB>OV6@9\.+CDFX#6FW]A:U[,F>/7>)Y :X
M"968><HXA25%:%FJDLH$LA1GP-=(P6J( \L0^MD1=1B;53AQ *BY@*5*;L>V
ML@:)<V&XF7!K!,9Z+\N;IR:2+EBJN"A&=V5TE4F[WS6QK150X%JY$O6=K+:D
MQ%L:='5"5:5^N*BB<L>\\>6\^N<U9PB/5^N+&6:@Q%N"UVNHA58BM+EBSB$H
MN N^P?\COL,I@4,:*2P#?\L4J5CJL)\)G;/$<1O N_:;3OA[J_6E;-B[I0$D
MGBE+G-DJ4IG45"^:0F&#P D<3 M\=S_9+^<+5#7[.KPYE-JE?2/9/23@L@,I
M"QV<7!^L$S-L-F5$BF^UFY0  "$WDFZUNI>C!J5Q;:H4PZE5PSB!Q)3D"'%0
M..^W3"$A NL)Q,MIG*(S!.EIQ-.DQ 'H-4Y,KLQYFMCJL<M7J+>IQ4>,29-8
M8S@XJY);P7\X06A<8ES<#';)LLWU#< 'AH3KO9\:F00[Q_]52$4\12W9QRO\
MQL0; Q51R(*L7'5L2<KZ=OOFL(HL76KRR.$5R6=G5 )%:"O_++!!P<&:O%"<
M?%N.'+!N7,&.Y8ZD0\JEK,!J_)C!HZ^??%4"G&)#&<O-"[ZW?%]D;'ZTS,N6
M=<I8]LM^/+O$C&?&?&7(?,G+C&K-7&3/S,K1W&G33&35W+[7K&C9/&/;3,7=
M?,R^W,?0/,YY]LTR%LZYC,[.7,Z6;,WNG&;J3,?F+,\2" UWV'+#C,;U7+O(
M&<IDRLU>'-!>< =XT+D&'</_3'"$/#CL',P:X\K&@+-8G*Y9G, -S04*AL=:
M9RP+&M$"/=$W\IXX%T%;N#(\HG+J6\H;31F*-'S6N\H$'<*I+&L^D,GP6& O
M76T];3V+O$&^VL:)C,_D"\==X); _Q84,VI_X*C0FOS3W].S:@::\5S3]>5J
M4WE>3-5D4VPL3^O2AD1O@U22JES46+U#00UJ&K5_1G2& ='/,;D]JU&1&Y./
M'<NG<ZJNN1AO(G&#P#S2]>H,1]12Y1=:6Y.YL%QP%X>T.K5'>%@KQ10$A\P3
M= "8,<<B^-FD RW.=6?'_>F Z96@1IRJ!OACU2"8300,WC6@EQV/)'V<ZQ,^
M",/2YQQ=2>QFPU88=WQS8$NGN)E=^#K*\-HK8XL10[N=KN-!XR;13MW.@^VH
MK-L*Y%!^ "75X>6(1G-.7H?! S6I=Y4E$@<YH+4X3<TD0SU$>:UP]HC+SGW7
MT+#+90M%M/^&%EB#W9N\K7\I,^/U=562PN JTY:];;H)A"IP &,AN4!KVT;=
MP$.HT&BG$ ,:9XLM<CP78AXX/?3):^/2>4EWB]QA$?UFB]@Z)?/T@C'G"J!,
MU=%M"450.S%H6$ 7<R1+=4KG/![=A/B3BG^;OP;.W,P';EYVF.XMV)\M&6Q8
M3O@EL9J[=BUM6(J8SUAHN=G<06JZ4?;:A5M8GFCMV?#=99G5S\)%5;(EY250
MOPS\=P3(FY*3$FC..:&6UXFG$E,YIZ8%9M_I"HN9(W/"X]!]Y%HP(QU]"R9I
M("?.*/A]'1=LRI"63#M=7!S44%R$0GY[LI;M9K![3]P*=?A89;S_U^#0>PS
MN9H4G33+>I;X'3[;Z:FG!-$<1Z%L:@@6@8L(@UWP]N;9<W6'D>$J6>0L#NB5
MW$Y/<J/OD-+@J4,WW4M[I\H'%=8F<J5A2CZ22NA5($>?U=]8Z.4?EKMQQKX4
M.WM@G>@6Q^KK*DU/?'@_NQ;SYRWGZQJB:B%,88J3(7%XA*M]!:P;Z.?OO>U+
M''?';>IQ!E2BF\]M@^NBMYLP689=,HLHTA#L$*\O8-8Z072%IMU J"B+O.)_
M#.Q8$%IU#II&07426^&OIZ[)>+:@F\R(/!<&#]N_0_%BY^Q4!'D0OU/V68QX
MI_'*_.48ITSKAX-S5'[__G6^#J?>^$HT_WE/XJ?2.:[C-5^K)A_(5U T4#<D
M?LG=GO[I:=WBV6D\22Y,S<.SB5ZAG1&]7SC61D/GNWF5DZRMC^2D6T2/;WW&
MP;T)79M>'[6,3*\_5ZWMQ8V@?U!0IJ+NH]V)+<_(<P57L<962G[XEH2*J#@W
M<4'@%D'R!^V^&*.W&6@"+4G*=4@O&&KM+TR:>QK8O\[S')/"##-\TE42[\1B
M].#XNV)"YF:<0+C+)8NE)^&!2;7F_GU?-(L-'M'U?J1&7TD&:XM2ZB@SBW7S
M^F[DJ"],%]\JX0'PL'WLQ>W695],8W]+EV3 4E5"81_K;F9I0K2FE2CU8'XW
MS0-\)$CJG?WGT?_/Q]J7]RYHDED7[@\1W=0XKO$S%@A@NMS^,,I)*R-&Y$W+
M!HL ")IEGF9I$!X#L JF8FFEHA$YE02XC@8 H-70@'BNA<_G8.*>T(HS5< ,
M,LR7PE<B6J8*E5 8+9LW!1Z0+"B,/2+A[4ROXXAN-[BQY'CM@ QS/BT@3GHA
M0P&!@"5C6X]&'WPA&VTT34I;$'N,GI1/0X\@&%E\F%*0-SH&?Y^>&B6QE96R
M%V&ON72FMYQQ<F-.<[I1CT%]F\EI%:BF;61V*B*38%R%6W.E'*!!R9G$Q)TV
M04:.FDG#$8:;MI+@[ZP/Z>]GJ((.\_CKQ] 2XK/T/DUCU^V(!3%*!G;_\\3$
M0S4%>HP,L)5!3R=>%P.^6H;#',0A+IJAV.<%H<8HTB#8XY"O"+&6^%C26N1R
M6P,:(*I0RA?,8L]@0(,*'4JTJ-&@/O-H 5+0&$PTFDPR,A6I8HQ>]T0=&70N
MXTDHZ20]W;'"YIRQ2?C)$_<UQRP-*^_1JHFK;$" -VG!C!O!IZ K$CQ@H &W
M;ML4*4LL<E;0HP0:/<YYFZJ)J1*M0GT&F[MPH=?#*--"B8O$#*G)*S3  4W6
M,&&[$BC.'<@W6@1M-G,\F5:.HT%.&08<8EV, RD?+*;MZRPEED&.#!=X "RF
M=$Q(T.QA!$Y\W ,""D_,P+5W)*0'F[NWKHT7_]4P>R6)V7/L6K=<=5&-^1-A
MI)06SNJAET4!BT"GGWX4" !#,FS1(8(;6 CA6Q&K?(-79\A\$V -TAE $R K
MH24:9P4@LB$#RRP!X3%8M/(;=I=A0P:"NCB""*&K%.B?V$80^,)KW$'T22&
MG<A9'W"\V)EL-DA(PAL-EC%8)9B,$,]CG6G7E9!&3F!19*&95:0E)P@EG%!=
M,LA<'T/($$P;FR")7"LB.M@CG.3TTZ::3/ 'X#T4U;93*^!ADZ8\&TS$SW(+
MN/>4=67!56<]&[B)Z%6"B+,=>H?6Y-$(@*&0DEYTC93-55H=BHP<BQ**Z99+
M\$9D0"S,V*(30!@QAO]J640I64(3Q"K:I%])Y9D+(/U"AJ !SM/G%V-BB-JO
MG5YX;&@B3GI$/S+Y>EB&=/8)QJ?D%,0-L5"DV(URC7KQ[&1*<LFBESU$\F.G
M$6XJZSI!XNL/J?\*">Z(U;I$2&ZM@;.."''%VU8ZH[#JX@JN2*?GP,PQ6T8L
M]:X!X9PO5+D$-+M2$!E3:3B<5EC1-NN #+CP J<0PK5\$EYH>4NP:!A/ZV\7
M<.7T*82\ZES'MGN0H)-Z9]55&I/<> ?.,*,"#"D.^RS%K$6:'>7UUV 3I5G7
M>)RC8+DG0BUDTYP*[#-S^-X IKEQFM-">.%<RQ6Z+_UI0JBP2?V)#!5>>M__
MRQ-T/*,>@N'7,M]WU6(3$8P19&3.%MO\MK1P4WLHRZTLQ!]T:&OTPA$#--RE
MH++EHS%6N8S7]I_]7N?VK/C5:R/D#T,BG Y6!@&#0X[P+E"9F\SM^.P]=YXF
M9E"2W(:-O1J]RWD.6.3R?87O=KCFKY!@2\6!(SJ6S'B6+QT!@*7LCO$:+4AY
MS&P<0[J_W+6@$T_+-V_]84N[$IW891>\629O.1 %^1ZF%:Y!J6M%<: $)024
MONF#9 RJWZL0EI?<U,Y"=#M@I]Q3KG(4\%GP"\0OU,>=%;9$4UN:';[&U2-+
M)2]T!OE?&:#'PWNUQ0JR0<7K0L VF5SM>/=)4NC,_T41JKTBAVK"GSS<Y;?+
M-4YG4X@2#*.HH1E>JG(E4TM :/@!'WXK=K%1#YM,V(\AKN4R)G)<KII7L"65
M#D$#R4<*3;"TD9&13M@3%!F90@B(74XOJY%A=Z2QP)T11X=2'(?6^N 8O@D
M<";D7ZKJ%4E]B,0;UD#AF*I7QF<U,F"- 5 E._B-E7 !&5Q1E5)2E:FPV?*6
M0:D9!2"9"UYVTG;\H)X869B@/!DG%9C88^12XQ+VX016L"D<!JVRFEZ!3Q]E
M*<32,E 8(S .*! ,Q@&&$DX]?<Y#=@E1VDJ%R@#Y\I?7$6 8X&<.Y#A%'9A)
M"B[WR<^>D,V<<?K!#K*6$/]""A22H$+;)0F B69HRVF209T+Z,,T/C!.=FEZ
MZ(G>"4_$0;1[Q(I#')\4+&Z:M(IUK 07%%40B_#/50W!#D'=2#^DB3!3,2+F
MD SI#J;,C:,"D=L&@?E+H ;"J!TU7.!X5TVT-2BF/[",,L'!R1@HBEW+,"$$
M1 H2*RTT9D'07G%,ZD1V'*RD3705("W#BPT-0S_MX"!Q1(14.]2UDWKT*$U%
MHS]6O?1W!E#4*5W&@Q84:DT24TW5;+*&7YVF0'*-36'R8DA'@N*M6\()1;NT
MV&JQXGQ&NBM>TR@7WA'K1BW"VSF-PZV'=(1R%^MB"A!TOR)LTR"UV29.A[0F
```

```
MS\7_;1M3!8UHSS#<I*)QJ)KK8X^X]=MSS*^8[FP1H-[HC1=R3U^<,ZYVH[9,
M?-1VN^N$U[)0^9OT13:TWB"""X/!TKE2+@[#O($?YR0OGL'+M^#M:'%WF%\I
M*G=S&4P2%_"G QNM[)5V^8\:H4*.:]Q$23 A7V1(>DG[]G>[^R4M$8%U854%
MUCTJJZZ8KGG(,+PW&:I]9(\.6A'K .!,+Y90.14(3C? V U2R7"'$5BL+13@
M ,_<\?-$9Q6;0>^?,^ZGDH_R)6\(-CMJ%$L( ?6;US@+4SG>X']T+.0GW@P=
M 7!PES<$E,VBAX1O@=GG<(8J3> FR&I<D3:6\*%$R6!%_\&J\S'=0<2VCCF2
M7+X-'Q3TW3]_Y:< 1I%C\ Q/K9A# U<8@:1Q-UKI9NC*\PR)B$%IZ$X&.IZS
M&G"G0:/<"O+O0;BY:48GRA)<#4&8Y^VQ4J,BW9+>SA9^'G6G/FTX7.DZI2T<
M4'##5Q<$B;IT)#;=-'?:6\M:=E_,V^JOH_MEW*E65X^8T+3/, \2"JRGWRNQ
MA0TCOEB?A"IIV4I8&60M1ZK&1<S-[E:'_5L*BLK>4C"CALF#M?1P9$$90_"V
M[3KH74YB>L[C+"@,DKH/X-D8>WTBQ]8EHP1W@T<1QL\;-/2/@?-!WYP N7/B
MB(*&AOND (/SAV7*:8_3P4#';O];C)7&!7K'#B01[5#P5'WH/QS"7=C^B5%F
M612.&RX 2TZZTL%6HJ7'.#-(P:4#HP?U!N:3#14TCB1XO>.<^ %\/(\X:,K:
MF2O@!K]M>8$L#-1;:*2U;9!9<7K&':W![M@_\=XEQ'43#+%_3XB93N>9:9WN
M%+L<2'3S1FV* F.G.YZ"1[D30?#D=\ILO<T*Q?&L7:#G@U_"Z,#D^EQ1-%8L
M5=YO.&EL,7'3'^DZ4P& .SP4LGYQZW[4YHSX8)4<0TAY@H8%*W(T5"W'$DRC
M7-XYUXW=.UPEG?KCL&XYN7V"V*AM(6%*0]TF0)RT8MGO$"Z* OA^4OL&W"/P
M?888B.C_S3 ?9R_/0GO(-3:GC>O2\_@(#W\&(4C)HA2;W^5MY2O[AP7O1F",
MY6M*$'QS%TES\%TO%6V@5RK6DRK&(D4-<FJ2(WWWYAG@AFQ",V7KYWW)1FNY
M4D*-!D@"A2DQ%UX>5%_GM47.(W^T4!)\U@1PEG=>A%+D!@;VT'ZS!R:%="RE
M\#%;4"MNH#PB6",*\"&=]P#SLTK-533:QA=G(VZO8Q"8<SO<T"!(UPOB<#=]
M< 7_=U3@4SC:9WK(A8;/M2U8H TG0U*GD7!)""+PPFC!MCYC>#SB@BP#MH*K
M]AZN !,P&']HEQ;-%W+2(X<>=@LP06'.)WWS(%^L53P,LB*(_P!?BCB'@  $
M/V5[/E)IIP L?K(0$')Z2!19U9%3<C6($/@]-6AXJ50O)I*'UU,D>M1 BY4.
MKZ$Q=9*+PL,!"T*+LI=^ZT(;O@ P225  =A]276(ZJ=3K!B!2G4:. A(T!$>
MPLA?EW(AG]<9*R("8MA5.&<_OB<3%&)N\\)=FL@0ZF=F#' F^O1X\@AY0I<>
M>90%:B-K^2@7OU&-:">#,N@2[V9&UF 2V6@:MBAMG--8J_*+4G:!=.&#S7!L
M&%!HZ[@+3^=4P:)FUG*0H@),"@%%B79&R1-'HL!G=+0Y %F(5$9R?S8NM"1&
M B>*GB$, 4E4!P$W[GB1UR,)VO9^+_])#4MD@F16&?54&< 28G(XDW3G(YNA
M+'\D68 6;>AV7Q\0"PYQ8&5B*TWA,0B6(2'H<1CQ:I,E6V/F:EH :9UU&"$9
M6)F612@2/$ I4V?',\#C9B599F61>D/WC1I(DA?05+.P''YF6.7"!,D1>]*6
M1^4R$0R9:KSQ7#RIA%G8:<<F#(''-"G#31)R&9?&3AW2!)\8@?@X>2N3" K)
MF8[2)4Z0>IH7FF,Y(Q%S.B.3CODF4-/#AM=P,H:";)/)"-TSEUV6-/C#*+56
MA#?&=#U$-(1X0[QPB18%%+JD$@GH3Z88'4W F&J5:U78*)?"BU;F >I"(#@D
M71,Q$3!0@[__:1N9:7"CUIH@Z58H<Y><N!SMUDJH8@P5@QPL94\J:!@*%@,-
MHG(IN%%S 99T9SC=9@(G"2.B ".EX$V5LY[A\Q1A:3JVXY'BH5[0V9F^N9><
MH&<>@82<!H>MT(7$AW/RU9=NMY/"E1=5T!N;X6=Q]2J?]9&% GW34"B= &F$
M1Z$JE!C"*61^N(0E5I;85P28Z9['9U"9H%7\P)>4L*0[<$07VDZJM TRN%B$
M$9P>96O$9R,R('Y "IB&%B7[>&B;AP[HN)2^Q0O.R"FPJ)#S8)%FZC2Q0!'<
M*1Z-\)<<5J9FD*9#>CQCT3H=,9+]]3_1N(6SQGM+59/P5DIFJ4*@_TD+)&JA
M+DH7&JI(@,JG;(,Y\5"6;1H:?P!\9S4]=/5KBMH_K0B:"#B1!368E(94+5$G
M.H2+ _6@+*DPFVIH!8:,7T #B?2A=6! 7."&!*2=%-!>?S:G+MARC$I:<"FG
MMHE\'!47,'-:9')$2LJ@!V>'7T%?G:H;102L<M2'1;HQ6<(HZ>-UO<D,/32/
M\NHUPK%7BYJ@-;$T!,J9<E=]S(BHB,%M1!!WL1:'8%J:/4:FXVH!WH:3E%"1
MVW W1S6C-U06!Y*)Y7.=^"-V]YI\"62HF-53".%:H0%]VYJ&SZ>.G(I/]4.B
M8T1D"RLU0M6FI>D8&DL> V)L7>&L.MAAJ_]@JZPJC37!%A.9(SER'G:H8\1"
M Z&BGC&!F"4EAF9S,\89LSG9$X@6&)3H9LITB!@ 6;*:7J,ZMF(9M-$J@2 E
M:(('HBI; S11(5X:$_6'/;$AL?1"9%D5KC!KM0HI<([Z+TH3F$@T"(^FLQS2
MJ_!@?V>+KY.:H>3V@L05"5RSEJE0%>.7"M"JMU=JF9Y)210''($[(C?+M\!I
M8MR7G3YSDZVB"U0*74Z5BOO6;F^R2R$UAFR#;N77*OH'KI/)53894$8F"(A+
MNF 15N-X6?&'E8_[)'A4O+#3";S8<GA#=MX)&[2W5>EI09#X'28FE'@0C)/6
MA$!Z&K.@/R[[<?/_FK[JN[[L2W71\TWU2(_ORUX\-((.6VSWNU7X]B E:&3<
MXA!ZJ0XN*9I/%W4XV(ASH6;5^XG^N30.,<#CZ['#NC:L1+RKQG*YT58PPY2W
M,+N4 C%X8:?W<  P6Y<9N) DBXA/-2:Q1*XZ&+=)D!X>G*DB&"1BT%K2=T S
M:\'B%A6R\&C.PQ',N@VLIZZS<38#BX)?T($\FUWQP60T$WFLLRAB0T&--S8/
MY+XX9I- #+!S>"KG^%*@P\,C1)2XX,!4DIYS<V6C:XZSTL3'1'Z:)Z0;O&)*
M8[)/53)\D:[[.EW@ 1"_ZE%,,@T<48#ALK5 RBRI2A-6AK)D_$.V%7>'_QB#
MOO5)]GLX0DJ"J%B]/>L+/NIZJL A O(UJ4G$"7,?S_0B]@C!0%JKF$N]C^QA
M:+:2DT$U=<6_GID?WN.GJ$F,2U*U^=M!@ P["?1PI$ ZM=D4[A;+1\/,HT>V
M'9NYSSI[2,E;RB$B&M-^C(3#1:A6-5!6#8L\W; ,*4) ;!AOFPNHZ1P[]]FL
MXMRJ'CNHGFI/<=>/WL*_0R="0H0W9^=+--$/"O@+6(F#0'@L['(<3M(X>^;,
ME-%I4-F^$)V[[RRTTKRR'\E;S:9HT")7-!PI* (9''4PC77-'&:P.J 3IP-*
MHRBQS<G09"AD*0%D[DPQ6&.VC-NV. I%4I8(\?^*)JYLP%='>Q%JQ+^809CR
M7P0LH,%#0#^@+JNPSA'\GJHZMM&,?!9-S6E0"*ISA1LM.(/"I[O%:7L($6\Q
MM=R#DRH0HRZ]B5W6PF/6T3A-RQAK5,##*B^6!_,$98N99+GK1(#7@T U)RQ]
M7V>(E\?WI:+J'&O]TIRK:TIYU7*M1=R6/>VI6#KS09:ZIFV<%;-I"!]3FWH]
M'EXZ!1>R;(M=BVV]J]I5@9@+SQ7=N(P=A+_7R5[M*7XSR*VM2DITVLWLQ@R8
M1E#M/8+:ECZ&'Y5%R2.Y7U62E8@:6V&E?Y+:81>#;0Z*B0#5O[RMN&3+&C",
ML9VT>-C#<ZJQQV^ZJX'_=D^2;<[ ^Y#]%9D7%VD$J'[;00! 9E[970PZL-G=
M]1X\5\8W\:FA0V&Z2JT>;=4&CM. (%:9."7]6-R3'-QGH*^QXF">>YAJ,KQ1
MS<L7;#"_A'[!!".2)X13\$S6$ME,"@LRQ88%2QBHA9@<G(Q+9 AS5@X[BH .
M-A$83J$ S+M%E00]D./F$5#\@2<Y5+A0>L8V'<_3/-E+-#[GI1 LP.+O"LS&
MU0,<H7D--S^-PRO\"L?W_2_I"FBPA=&>U@6(5I\I. 2%0-JKJR6I:]ZZ<+Z:
M V.M$C&JP.,=;J/;F!OS<%5?GBX=<\G=X5!=H=]Q7A!#G7@5_K80H."+J^17
M_TUPZI0X4EE CPB*3TX68?OGU*S:YX2 $#Y;>K,P-2F_D>?:!^Y^DRW"KFL^
ML@?@=0?D?Q9'LV1P:>#E8/T.N *E^@&=!AI%"'J_CLZXJNO%^+VFG.[=/.PG
M W9VF.9VP"E6F^T?:UX%R]@0LFXV54$S4AZFN$ X*F/B8+H1P<SIH3Z<%JOJ
MFT:I_##!=M(H*7'M KK=@^NG N:P2_WHKXW@E%)EM,W;YTZDC@5VIIZ^C9=T
M6 QYR=>9V3Y=EJZE-K3G\ GIJ>[IVM@+#;]M ;]C$[+3N44E=O(:FX'9SBL7
M$!L<1VGL[Q 1QOO9]0.G" -G;H[<<\UCPYSLL.W,*O\.L%N6>VRJWS?H8%AU
MC 0."PPW[QX>DLRB#<?=TI+]"B=?RCB \=+-!HX0SNG6W^PWE()N'^N&<" @
M+A)S3PO6+27-#B273#YVO,0.YPP!  < OTF*\_S^R%1X39(&)WO41.<]I3GO
M3MRL+-W;@@#J0^*.I>Q>] #_Y9$(=JO["0=DZ+M.K%,IUJPU=5E\O6V?W+VT
M@'2_Y,R,UZSPFJOXK]&QPV'RM[^,=9M$;0N1O58R/VHOVMC1 GIV^/*LKAF_
MCE3?WBH.Y<$"U\=.YT<%(=HS1W:<&N=.<C.I>D2NJR 6JA1?[,I>#'C^^19O
MP3D$ SW()=TY%7BW?*-AE]7_Q>HGP5 "N1:Z K$/^"JX#_I1P)&[[WV]GZ@Z
M-^#^@/GS:&-.%X]((1D(,&)D8##*2:N]&+;,3(F/ X308@3.MH$L19(7S,UT
M;=]XKN^YS/_ H+ ",#T>"R1&T1E:D*-1T-0PE4I%R*/QTCB!*T-R-)9J2X5P
M-F(4$UN3;L8W%%B_^+S>2=_[_S\A2&EG<V-_'X5#*VLC7&X##@*-@#A61AM'
M;IDL A\$'R94 &$2<G%P,7H;=Y6NKWY]L+.T(C % (ES#HIYIW63@@L?,",%
M5FNU%ZTA!)JHP5NYK:EWOW"RJ7C4RMW>'-G??ZT+Y%-BY6Y4&-=XE!+<-:*\
M+<=N_UJ7.%VX47I=4<CTJ5*$S(0]=CSBB5O($&'#6MP4+KJ'L,V>+),H7M$1
ML,FSC1IO(,G(R1\+$D:BJ%S),I=+E05P\6LYTQC-ERUSLM3UL*=/;3_SI(,7
M,H@Q.R)D@I-$S^2"2!=*95CQB.0S+KA,A=N8$5\O=Q&.^=CJ@BL((]3DY+M'
M%IXH0D'C,FPKMZZDC)[,QAA#0&*@!<Z(-K$$PDPN1202H2PZ(0G@4$<ODK&U
M):](D 9;("T4V(+%&TCIVAW]131I&NO8>!$"ZF0_O\G^)![FDHW?910V'(ST
MB%0/S8_\2 4QG-T*:Y\]ICJNP;2UT]!=F8Z.(:(O+*KA7O_ -4"G]^\Y:\*,
MTAV\>9MF)K5Y"9+"\R1-IW.,D.DVJL'8WC!(1*UDCJ[4!0B6@ D)=H=]-"B1
MC&4(%5=::&6$L%F!@[TC23L5B%*;'?T(AQEV"/:"' 0'7:$+#/, -4-R!+;(
M@WPN>I"/)VL9Y16&$]ACQD4M1%'B?$W1(^$1&>&(&V.K['*91O")08>1H;T8
MXY0ZP$C=<ZK1PXH0'9*X(T*"Q**9E42\$"8(66GQ)8'3R8*AD2K60":5;-*9
MX&2&Y8DD$"$F,^<NP7G@X&]%*N+?EB'^U.9 0>:G9)5V1KJ4I.X5HD17GY0
MU0_]5?I$)SQZ!: Z4AI0G@+-'.G_(@G,Q>.FBG!^M22EM)I2JWMG A3?FCN\
M9BLXP26Z)("Q 7& '8X%V>%*,4HHV*?0-HK8I)#>>NN?I#7Y3*YB_-BKDR4$
MP]P;/HIY!GS.=-G#8878H]X9F=)7YZQER1H?M;]96RNVT+VF*U/JEDH/,>U1
M$,IJ)IT[<, )?3:;2H.29N8,K]H;J[0V\*NOHAN32M$ZY@ !  J!05%C'"_%
M=-[*X^&$'LO>S:0R+N4,.<$ $=LE#,;WZ8?QQ61JW/%#0A,HK T$1.(8LWLJ
ML:>-X,+G"<,<#$51%5U(C=VJEN+;\Y-Q_EKMT'86+3$'.<OY%5FSF<M+DSQ7
MO9%*^3T@_Q:H=<RWX;-E)8%+Q!7_'/;@7JM*=EUFCU8LWT:E=Q9^]4IQM-K3
M@MO8TQHE#3=&264BN0UW;(DYM3X<V)P$G2D4.-CV]IQOEE?XU<:$A].2N%W=
M*:;MYTY4$?(;@>IQ$"&3Z):V9YMH*2VK?+I3GRWRJH8S ]W=XRJCK,=-N)(.
MI@:[" 77_LKMB(,;YAJ3>^II7ZIO'6I3<*$8B-AZ(2OWBEY,SM]'!EJ>_*/U
M$ESKZ">0>%C'-B02G?A@03ZY#$)(W1+:OYAFG %RR5$R:MX6PM( +NQH&(2)
M'N-FE07'B1  Y;&)MP@HMNQM)6@ND%_XV,"3!<ZB@7$YGQ88,?^$$LYMACV:
M3,(8M+\(52$#WE,:B)A"CV"D"(E,*H75,L:+8F2( 4@X(L("&$ 7%HZ*OW*6
M%F0B,Q\JT(: P&%05(:UPW3H>(\ZA19= )@T+E$.=,L! 437-@@FHGN[>D&:
M_ ,:?SENBHEA&_9\]D*!24 [X /,9EK3H?2A\75DVXPE*0<P7I7I)2F$F2C-
M(Y[SR.1E[MO6]U"6K"IBPTJME!VW /@U1GZ1DSG21HIB \=+BLR7=O3*]F 0
MAB?.;UKHFHDEAW**([QD1K;RX1D4!!\UTE"+3K, )-GPIF'N@$%=(\DJ'5#*
M- $S8>=,V)G ":W.6&Z3M5081F18'6O_Z()8O#@(@(:"%@A:2F6\<$8O"Z,=
M622'>=!*"]XPR"A,JFF/!BW',:SF":JDLX<7]<>77%(.U1U"39<[YKU(Y(8L
M]L!J?R2G#+)IL/+<S3&CL$5,\E6Z8HX)';0TD6YLF5/\-4H-@B*I"I6'E*[
M,Z,L1*H>C*F?O!0*<@*30^I66!U35 \+$W(G:D"HI4V]C3@G%>H18B*5KI0+
MB ?KV]LN ;0JJ2L*@3'J$8 :R:\<-:/65.HXXVF,Z8V.8AA\Q^2DV90#QNX)
M>U1I88K*U1MHKFLA2.GCF@";\*F%#4S0G@5S4Y@=->.J&RS"3,QG*4< 4:\)
M0FT/O\2/<,6@_PH*P$1I MM!>J9VI&. #'N.-(\.;A%APAJ)#IMRQ)2][#L'
M",]*^&'.1B)-3>/! A0T4,-G#(-#+KOK1?.*QERQ2#]=$B/4<)L8" 3 /LRX
M5#-9:,#[[55))Q+26A:$N53!8TC40,%F-1N#VIS!@Y4KZ4BAMU_51LO 4<42
M.YAJO?&NMTFC!5T\\1E"B:3OK:"PPSW'5,,#K\9"17%NM5)",'S^ V[H*C""
M7;?BL#(1+T",62E'2>.8X:0F,UZO+;2;-\.%!5TO/.. />RS?!E3$)_S%V5J
EM06-]3))/SQ>XF+.MK],FI-5!!TQ2NZ4JC@JS;K:(<;8@+],F8D!/\X & _
!3<B<Q%V!A0 SFC'9!&Z[HC\&164';+G&Q#B&AZ% !-L@@XK&SF^W0HB[XA"
NSQF.9\9+@2/ V&8%K(%=1-U"#F*R.>&FJXZMA7F*D?@4I68F;M#0_4")QAA
AT@AKNZPR+L>_5_52$07)<EM&U5-K33XJ1>2.Q,&THH*EXS9FP<*'>K&& 4S
YP-]#0Z,D,/2YUM6&WE9HL;U=B>#@?)TP!+BR:<;\T8/O/EDW2 %GDTQ*B*!
=(KW,9TS/>G-JN**VI>KF1BMT&%$3AJ8O$;C$6W:S@7Y!JHY&)Y$#5+G)AHZ
I/Y+E16^W!/)7FH /)&&3G(<A>0J-QLB7K;_"%> Z/8@HV7I*2M)K_UDEK-C
3<3>BZ7:[.TN8C!3,C '7;4<%L]VI^872<<^?<U-\/U5>:IQYE7=# U,+\L8
.G)#IBOE90R<,>#6PCK9PC0&^T A3_".JH#E&41SOKSHT]+RO[\US4LK2X82
*9VDP]'(7;K/29O#MH\[[?(F]QT:#%I<D?=^Z+$[M8F0:6W.F DJ%,7\&ZU=
LBY*M*AHT?V+OP@SX@^Q-ZKUM.]:'QI_2H[P+=^DQCLY+G-)26H;LZ0#@LW@
T0'!3[WS]\V6YOO@(V0]OU+3 ^5UUF77_'>_%W^'9*_O!,( =,N/'<Q;H]L<
EY^EK.&V(8!\%E5U_\_0P3<UFO(J: N2AHY<]*6DL\/E\4/?L9V)&DD66M-1
K4__#OZQ1.@&]F,*I=6>M-=CK4$O\=1R=&0PW;(95A1@1< ;G!$*",)^E *!
UC(U]F5?VT$B.K)VM+:!S21%&3(")&-NG<5L?\,0OV,( IA[!!AO$1  OD$L
""AK\=5N>=%+$A@I-V@MTS="MI(L>U-ZPV(S[Z0$P[-*0U%B"E,-4P815<5,
P;5(W.<ZXX8? =A)\T9.($5O*^AR.5@K*5)[#M(%H-  XO9+@1):'$1V^;=E
0'%&&M@\ZH-V8/1M3->#;F%G'?%5F@1Q#E5\74@KOD)@NZ M6#1;<W=3E?]C
#LQP!<7 +BK&A"/T%@+A?%$H-E_(A]>@<HZT?L?G:A$W*4\W(. 66$>'$OLS
=M, A&>3@MUW;AKA'%NF'1KXAU-"BY)R<.YF5$_F/1B%?/$1);[X9YX'.VI!
>BU2>71X;_7SB&\PA;W8B;?7=R?B3ELA'J6&>MZ1.S6AC2K!C<?U':&$!M@P
;?W"@[C7BCEB9M\C'PPW>V/3B;:(@U(G(VWV!,JV@U<&8YJWCU@TC.:52HWG
CD$A7M86*RP5+O4H)^K2?)_'A="(6/[$1-LA /@8=B.&:($GCLBG$@]84JUQ
(F]X0]0'A*N# 2X80ZN'>F;W/O#XD.Y1 %=5!O?_4!SB54</DD"ME 8Y>7X\
!T7! CVRQC6GQ6(JZ&$2%XTQ-!DAB91.%H^14AR\V'2_Q0=;(U=5*80I $:L
X)2.A"AU:) EQ1.$M")^XXANTY(NB1N) FT"R4DZ-HK^@VV?T2E44@S;%S9P
XE145866P!]\.%YHF987)!VDJ(#;0Y21XC!.,B@C87,>=6S,6'UQ6"F&Y7U_
QY4 -SZ%"9=,V9D,@8_J@Q):J(3HZ 5DT#Y#Z5MJ$GG#$2+HAID! IN7Q%UO
69N':7Q!<8;0-8(C19"_(D4V0Y.+X7W:1FX\"4WV%CWX*)O1P9PV1)N;:9MS
F"TKEQ'NH@GP<7]BD%D0_WDYX09G4B!X!%2<^ $5>O8A<MF0+.><JZ:9AOF>
G<F>8D,,C,>(C.!<FA1AQ4)A,U0ZCN9?UZ<^%FEMZRF8SQA,G"F=ZE>.\]0=
^E4; 4=\IA!YCN<TPUF /X0/\;.=:8B%E $^N1  BV:9H&>@@UD)"JJ@MP4=
M:)%69(@Z\R8>'&=L.>$_["2"W/</C_."84)),^<T!^&;MPEE\ED[T F?*BJA
M+ INU'B>IY5S'RHDF:8AE964"N)9SS>$'9 5 7%=E5B@)GIE*!J=F[FBI$%7
M^^%9E38L1$$)LHB;301^PL1U+Y9Q)Z$.(/FEUU:DAW.D"5JF,P"9H\%@'=!/
M5/^D;2<2&PK&*)NA.[^H)2] D9+V-O%B-U$YI$T9IF**H"E:FC)W&D?32C70
M825A0N<8!ZUPDB0BDQR"BJ2V.>&)9$9W9B[)IUOGGG\*GRMBJZNQ%C9I;FLY
MD^%",&)4"B'WEPKF5_EP?I#S<"%6JYHJ4IQ*I@%F.+SZ?QEJ P=AK,)6APM5
M, I5B WV5Y($?] :K6PWID@*J,N@84;#./Q4>7-D6JUAD<=*J&[',^-"A9,9
MF2O&J^V'JYU:K7H7#=AXL#2&8\OU>BDH!<9XJDEUB?>U6[PU0\[JF0@&L!OC
MIP,[9/:VE+07%9M*8981,L<*11W!5.W5)^>J+_@ZE2;_UX3P.H=#P9"4F*L>
MZY#O1K 0BZD]2Y4= [(^VR(>R"0C61S,QX>#E6@ )HH=>S$(Q@F'YZG[Q8YV
MU98$ZG] @JM4P@T$=W2R\S&VM@,?D":*8;,VMZX\6VV/0#S*L5DGBT0Z<99R
M 4]HNZ!38G\N8UPZ<1M/-$?S1Y%^R4[S1ZV<>6T(V =@J9[(0U%T*Q=H>[=9
MNZ2BD&Q7%!5]<*E&JXH7^%59.91S^+1#JU=F0'$YBYBHBQOI$C!KQY[I<[&B
MV+5[LJ^7)JF%%8</1%FE4E[7H;:'NV?Z)I$?1CAQZU["BDX_X:N@83NT4K/C
MBE.6>*%Z9UM'(UID<%:<ZY:&_PNU2,5# [2X>KH,ZO*R[YB;7C"C-G:7!RJ[
MKA%!A<&;-W)?5>1,0)F*H-N,C(:U4(BSW(M4P2"':;N%0_M_K;N*./"UCXN#
M>!1->7>G)21\78.Q:(*P%%S!"XN^%IS!*R.C?#L6^TMDBZ@S?T"[2.15,%8U
MA""HO?A=\@A[QN EV,ES[DH*BG%?(!0-*IP!VQ0J^GA'/HQE/K4Q$W=8)_"*
M'TRBO^*UR$J8>["R,'MQ165RI@,@DEL!XN:7>R*TUY%K$)0%]K!-=B..WNO%
M4_2$'C-$&)G&FI'#T#N!$'=$UW.S;G8'8W:"SB@4.(0U1F=^(+6:C(AOI_,!
M/X="J__K)?XP#_(S*%7,Q,&[FY.0)FQ9!-= "EXJ1)!"OB(C:X*UR2PL<DM(
M)PI11R&<C.<X(N"J$?H%LM:PR-NA=G:J(X2D'F'\OST2P4Z@BW*FN7EK%A9&
M?4.<;8RA;'B+3J++C(^LP<B,C3]WM%])O"(;4&_<#7FX!UR@(<: %:JT-E74
M9GB!9*,I89HQ0&&GQ3T6Q0@4E1XHL19@LQOF!PIW@*L*SQE$(_&,(.K;M;X9
ME4ZZML:W$D_TS[\,"+ F&TG\-NX668/1JD*H&R 4H5-@![&U1,=[*Q?;FD11
MM%"%R:BJQ<5\NN$;(PT\JQ +OLK9#&IF#2]L!&78Q [@<0+_75V-<)JY0Y$R
M] "^%V](IEUL!5<>@ )YQ)%)\<UQ$1'P%,JM,%"A9HC0EWQ+[;GVIK%J0\L'
MZ7<D'< !J8?Z*Q+^>,LT )IWZ WCUJ.2\,ZCR[RN50&FNX:VI]33ZKO]ZZ\/
MH7)/%*E)\:E6C<0YNP^L?"?)W-=^/4H<!S<$ VR*LW=I([V7>Z)MS;\^RZ=H
M>,P3O#7D9QE] )E5/8"=I7%FB4"^H!2?>5CQX)6Z&Y*6*V"@L"&.%I=K),SY
M=KGZY=-[V*M9+<!*W=&FX=C8X8!(*&P]JKAQK%D@)V6,2,\!*@'J> [U*@Y#
M3)[+?36S_1]2UICW0H9[S=6WD!62_YS3DI!AY1$!W:T5N_G</PNTC,W/M+U&
M=PK B5:_D/EXYFU![9 (H5!R:BT$._P-=LR#6CS.PD;=6BK"?7$,)1.D;0=I
M%]*J'FF=7+#7'+N]C1T@UCD,8=!A&Q1M'I:'EP>% ?&BB@5$K+S/WS NH8W6
MKS N/H0,GOU^/M' <U7+TQ4F\KT)@M 90KJ);F/;K&@7>2$-1-EZRY5C7D41
ME]U"4)DY)#R@WT+.WW2_L_!I!YE5=A84-!(<UB6"[E9BI%!"8QR1Y+W8'6VF
MS0E[ H9?3GW6)&YH0QY8,3U&DX!QYB432^F<2GY;G"!M\):T.AY(GDO+T"--
M 2,Y41>[!/_MUE2+U^534FJ*E9.C8!GN?/2<S=A9G8G3R;2PJ&FZ118HP5K]
MOL)+>#V!IB"^AC42RB&%Y.?]C#C>3E;^R4)YHAG.BZ?09H*@3QYMWW/>/UQ!
M#J6SP]:D"=D<Y41,G<#7RY'HFK\%XJ?NZH:+<R&EIK66=>3=Z%8,!Y$6SRE@
MIPG$U=2UE,P4;.<CR8I!A!U5OX.@)^+M6K/4SC!;%Y5I4Y7[+"0,9TUUVP+K
MX"N7ZSSQ"4Q0W403[>J)$UO*;K4,>\63P$-DT"(P"CS4;I/J?G"]%*Q0PV,;
M(\5QW% 4/O- .Q/NO6!^X_;N#$QEH5ICZL?H[YJV2AD'5^S16GK_\D&"OE3O
M]DR<(R1FD.]XMV/PXXAWA> "ANRP.]1_F1WITK3]D!7J:$XE:#?.!!=L7-:O
M\[29 3+W3MFH$^1T(D%ADY!J%1)%U2HE?BPL71:^P>%W&FGH ^.6 \N?ZP0_
M$@*^AR=[$WGL'A(6[R1XE#3^G8#)0 SB=SK#W+OE/=U)"5W]-]7-8O("["/+
MVI;_1NE"<'+A%NBS@T((+0'=7<A4OME <-SKP-M!M<T/[PK:1I? 2HBS]'[]
M$!L6.IV#'OB)X64B3MB&7_*H;G6TCE.KF\]7Y(P[_TA^H,\;,7WCPAR:Z.Y"
M\0:JBHA9"O2[&D8#XZ7[T78[/KQW-[FU_^^[B<K!<RMULU8VB(_6>YN2"8L>
MJX?!K+>W?FW^"NOCYN\CJ??7RX(R:P+5^%&AT<_ELPH0%!Z9#4[HRQM(" !@
MW/XPRDFK738S;+AN0J<4(4,8Y8="J2IY;ARWPLF4+>ODLE5BL$6( ^#UCLA)
MRL83-&V:H@N6K! ;UPUV&W%R.P4M"B.@5L])\S&K"IV&-/0C9(S*,_4&],$T
M[%=W'W000%L -E(8?X&,1WE*-RR/%&J-;)=?E5B%91T_?AV-HA^:,6R"#T))
M)8MHI8&MD("S.WH6L6@G81L*8P6](@JOHZ)_*3B1>#?',L.NF=!BH3K+9'!D
M(,3:E'>G,@2JH/\S)LF17FO;U+1O<[0KQ[BSDZC9G[5=*//I=W$:^BSQ(C@[
M@TE:P0J[M @8 "+@/E$#,WAKAH:.0XK[W$@8)"Y;.W>V(%STP2=;PI+,'JKD
MZ)&D+% C4ST\2)."%R&=Q  H\$NERH@7OM"[R=+4" :_A-'Y]\*G'9,>."I
M1HC<&1S"!@R!Q(RITW$?N^1@A:\<2"LSHTU;.Y%0SZ1A@-C[2@PH6FDN4MBM
MNB&G%[(]]LH($<#F()8"PA"@BC?95D8A#IP %ZI%*Z]T'?'[&%/FOIH&A4[H
M\R-$0L&9GSYKG+=R6(DK%@@[F>8GL-MEXF+H*<UTEEY#A)!8*T=1W-3_R).K
MW@::[29AT*-+%X8";N>6@9J8Q>X !F;/Q,>IFAN%^H/A9TDIT^S:4RA@> O%
MUCGMN"?9V8,)">=2N?]TJ)FBUD%<6>6%+AO8H!5)A;GS75GM\,". ^Q8@\0)
M9-C%0S@Y;53+ NA5N-I+26"X"W3$$;!+4KTMP 1^[^EUB"4KV,>='QP]^-^.
MRVG37%L4IO+#?F)T6(%&]<A!FED2NE. +AQ<9P*0ZHBC%SE R<=!0CJR5B58
M>L2E5PNR83C$B_2=. U^"ZX"7A%2!&-  "C.Q^.=C 0XQ8!J2:1E9776\8^4
MZABA74>?A-'A@W%J,A8M5WJICA0Y4==E=T?>_].:" -,]HL3'E!6QDY(I;*E
MB73LA.$8%=U0YVZ?_*8GGK1B^AF?H65JYZKN 7+II2(%VY$..82CB'@WH*<!
MBS90U@&O&^6!Y(T"9@JL11$LIFE+EGG4CR@TF!E>HRBH2&BMZ(I6%Z[.Z>H'
M;8?L0E4L#>8J:2. ;73:I<Z&1P&=B-08YY.TV;0MI$@XD]1T##.\\,( #+"3
M=!#/!MW#T54\0L,<9^PPBV6029T-!:=K,FRWVDL@A&*H201PNI)QFK(E\B)7
M Z=9!86+- ?FZGHY>R*O"*1XO+'$\/ELP9-7$1OM:X!,N S"K09I@)J1,)T8
M%L">S..LZJG<YP6W<?\2M(?1P%C<?![()R<O^AT,V\(&;[&5VZR2J.B-NIQ[
M[Q=^HWWH=N5\*\]FP;JM-I5>-VXK@.PR'G;)!LM[G)%(S/68%IB " 5C>]99
MUERRS=8=QQ4;'-QT&AM="GHZ7DLX2%WW4,)I9<.,N>.\/\Y<Y*YVS+K%/86<
MUXSOT=UT$(,DO>5^1K<NO )(5V(90S!N%:D$VBIT9,_K#<M7YM1,NUV3A<NM
M/GN/+W).OKW'[^\H/Z8G%K=Y'^M/%J;Q$U7;T^C0(;*7)%. @T4: $?$/L>!
M4=7A<U>;&'2J)[IRC"0EWA/?IJ F.,)U:UN&F\$Y9H$-:LGOA.JB'_!,V!3_
M4"3">Z_8V51\=Z&\72V H7#6J(B@*)@-YH8V%$0)L32_NK%LA"[YBU7P$C@L
ME"Q*/S,B"SSP*1R$,'QLV9H4:X?"KX!--0<IF#[@ PRF@6%H4@P)%T\G!;H!
MH$$EU%8G"N$W<I'("8M0Q0LKT <0F*\HH,!,L:S6-2KL#HBS>\?4UO*8-3H
M:W$J4!?E]T64M2N+*P@(F5" /2;V9Q90:.)'!L?!] S-DL,82 /PTS^;\$:!
```

```
MFX#;3DBP& 0:D2Q;F4RK$A,</TR,%QA*&MP20RHG#.!$9&2 Q$2Y@R+("QM7
M=.0D4P8YL9W2:0M"D2'Y.#MICA*$<WC,"# 8MGXI_Z-8,#.G0+3@,G$^RT[M
M<5=O2G*ADY@-!9*Y&5*<<)Q^48:.E^O4^MS5BS>X: =7G&;C*AD4:\:S/P!T
MEBUE(;6!VLXLYFF+7WKUG4YL[H<H@M9=.*<0WD2Q5^%K((W>)A_M*4N":^(%
M"8AT'_. KC6+\X=">\?0D8+FIA.P*=-HRA]:Z?&2LQB.,#%21 K\X6+SN!T4
M(-D78Y$G;.N:T5#I@[<BV -&8"6I-__T*+3MU&0]Y89#S9@/IR**%BU-:W9D
M%D"5ED:1B43+1UT@KKP20B-F=!$GJ%B&BY2@,')M0(.0F:'@A&!!>]M #_L2
MP=+M#0@,0<-B:Y1#Y=4IF/]G/5EBU^E0"<Z28TBK& 7]T*;1RB%G%M-(A7X!
M,>E-S[;3VQB7E '3A8&*L1&4)>7P,40YT( J_4 B7ANT!V,$J09-\V-WM'>U
M  3-M:'M1EH<:K]P;J=>='PGNJZ /"AR36?=#>I=V>>8([$BN0]<(LK&DEO4
MF79A%-R8;CW&$_U:+"O^I5Y]&T;;_Z)(@,[!;G;71LV#(/$?M\/&5L5+JS.%
MX<(I1'#F).=6_(V&GL,BIP8S5=SJ&#=]FC*48KDW8NE6+0<K$@:4@+.Q!8]W
MNY?<)A9INJ:EXFEU-E36E8AIOC9D]I!\_!93HNJK\C6T'6 [C!&,D2-D5#18
M"=7_S%(^1%U?IJHR!,BLC>^DX!6FEYZ;%(TW&8$FLXI!G10QC5=RP$R/[.$8
MQST2\L8'"ZOA8PGW(*5RB^$9-.HQ9'L=LW_*7-I2RH(5J@"?R=!WOL!6H5$#
M[AAN"3R\VPIO%WGL,PO;JU,4C\)]^<N&Z,RCZ*_AF B?*&2"=,+J&-P9H9&
M(%/CUD(**ZW5(R6&H-Q,/L0U<\W QO'OK"G(>@##G2%VE6]Q1L,R'CAVQ(DD
M"%@D6%+5,=G<0/8BW=%'#TDK(SI@-+A76LVU7-9O_2!9$&\MHE["C0N@<G%'
M,XB-@/E&Q7DYB;B-"M3HML\ ;<H62DA%CO==,V&W.?.Z_\G\:GQGFM,8 \)2
M-;(Y$)TC0Z:"3^",)<P,F=; %T^YRE?.<N&!:PY,8"LH#X8,)'>GSBPYM+HG
MKMT&XTS$ME,BH-:G(H4 $"FU?GA04J7<A/SB#^>R.<_]B@3>_))HM*5*1F45
MFT3XL.<XN_+4:Z7NE8%ED,DJU>&@6ZK@).5V4TH6,F95=%#L-GC9RY$,V#KP
M._6= 71Z5V4491][8LJCK)3-8UG%3(Z/2^)C5T[9H]&EE"@@YJO3! S,V;D4
M&J4G%91)P85(=6![\W;-XZKW>ID8!.KF\OK3=C<X3JG(H[7B15K;09OJCB&^
M]!V6>AOD1X>=.*@T#7^?>IKAQ/^J.5)6U9@K0L('"ZXM/=3V6 23"N)$RDN/
MC2#!WV;RMY$"29O#3SZ>^-_+2-G2\8?5R23:,(-8A6,P*YM7+["T=^Z?.M-.
MK^?G:'MB+\/'8OQF,XH16O[49-N'-/(5$J;75@*H LT2-2;V*,TE/KH4:">F
M1IUE")QU5,I7@*6T92]@&H@P-'56$&N$(3O3?&.V9Q%"=?=U,1LS?E\Q9<&P
M:;G%@P$68 5F@T+8<I\6'1V'8!RF@*7',FN'%CTA*G'D?0;!12+64F<%::Z@
M% S(6CCH$RJ&/(W'5U,#:#AB9UEV!.66/^;A69X';M(4$#25: 21"3[H:0TC
M4S'H!(/_1FI* &KCQGN3E$<RASA6!(&?-"\DV 800 +&<U(YI7X698!*4T+>
M(87$T7>(UV+RDV8S8Q[U8A.JT K\]Q4&E8")6#<5*"S1YF*X-H%N$D6&EBAI
M-W&[\(FM48B&N(@N)8>UP1H.UP8B<WGT\8 HQ J=DDS#IEZO(7LVMGMKEA*X
M2#42*(%= 2YTIA-+$"@CX']>,RL(%'%75Q5;QVZ)U65J-XH^H3BV-'IVHR_
MEG[<2&RNF$;$("KRU#C<%D%#0%7PEWY853HDA4/PM(2_QE!])&,_]R7) 4A3
M\%$&-8 AES'995+81P]_17Q=6&SN(1M[TT/&$U;[47E>90+'_\%\U0$8$V-;
MQ;%>'6@K'9*&R1%'0J8VPT@D2"0O8_0T\L@C8L<1,I>1*(186^!<M5!D%:8F
M(2(<:Y(3,]48F'$B8H(?(@@")40#174$9T,C*A)KE76*P@8(#%$L1G S5S $
MMG@7&P(=#'DRQL ;>E>1VT=MMD"40'E1(1E)<J%5,84U.G)4R-0A=S=$2'*5
M!4F0QR-0@B=>P^43*@(B#/=+!T1,&A<,E=)66GAFLP%P._D?\%,#.U&7H056
MEQ&)?D>9H!9,) -ZG&0S\P@>8D),2RD.+G@CE11ID,&!,:6).3A3=Z-Z:\(K
M8D0BID*1R<!J\:@-BT ;W0.:\E-TG/_0<-S!=K32/?-19>A5B)W9##-$7E]G
M#[>6>[VX0T3X@R<7<<&E'+UI2A^9FV0YA$"86^9P4ZF8+D_U"LS9.*!B.NA(
M?OK(0=_I83_D+2A!0N!428SHE0;(#/4V'%MSGYA2"&H29D"4"/F&0Q/R8)2
MD*A(C.-U? [:1>22@%=71COTH5P!/Z5&E/SI&=<(B''9FMII*XUBHJQ2"'^B
MA4CX3=6R/C2:4H8)EZBTD89@HTH'I"/"1L0#=$^&+^=5?L^!GQ"EF2]*%P,W
M0T9*?!Y88IX1%26)5%>J0KIRAA.PF-U0/%IPG-J HK@V&3RTA5CEA8=Q42?(
MH8I6%&3(!.C_4RF<)DZ*YSG/]*7+)A UAJ N&J" H#@"VJ/<U%P; @>(M&OI
M\(RG\Z.A^9D 2325LA"?B73"$8W--Y)@E3UD"JB6^!)J"AMH2B&(ZAK[66IN
MY@:M) *7PFAYUC5X2I&*^A/L!'+.6:.<XTY()&\+X2I%!YM[1*I@^@*J,H=I
MBG1A4DOG52M_D5RD%8>0ZA1"IXB/LX?KEI=9DR!$FA@2*J3<\3)Q)7S("A$0
MLIP(JIQ,BG9)F!EZP1.296)'4:%8\VN,Z8_^P'JZ"6Y$$B5;)03'F)[Z% %H
M-(VDF:Z]R(275CW]-9X4,QTV(XSI G+@F&J^I">T2B%BECFOE(L\_S=C*"5_
M<J(EFXH3)<&,_\JP>8*ET:EE58-BJRJM.11V4KD%YG>M+W>H_)H!.UN1>R5V
M3O8WT2.QF9:NK:JKK"I:<-.43[(EN2&QG[(*D?44A,DRRTJI[Q@@S+"T+AL%
MU1A:^0JV)$$$E-&8O:)X2K*PA'!\8(!RN9D>Z"2-1DJ=2 !'>!6VKM:R*#22
N>I3/<:>+8D.E,@ATR5_AS)(V<JW'X!J,NBX.Q*TT[11;-EU;C-3P"%U&W2*
<,$%VG)G$2$C()&JQ8B&I"6=DBMY7)LNFZ2DI3EL:-<J\"9 AG,"^34QZ%-0
JSMJ.E6)O8L<9BML_,0[C(%<+$"Y[+5]F_^SEI4(O)=80*0J.Q,[O,$[(Y<Y
2<> <DC;O?\%L?LEM[=Y:3JDN>/TO.(##!GHLNR8?80:O(' 2E<3N,O[ONC6
A/9+?OF814$0-]&7M3$+OU4BG@"\4VLI2:VKD$7I5_)KO0(<J?^H5$/*?B07
KP_,'O1K; <B9*-Z?K")EA;H("@S41=L8T-2A+&TML4Y1X\9,=E;PMV03F.F
'6$)3@/<H+EF$H#6!Y>9=#)A1S"<75-[9EH49IRP-;JD-BQ,!:8KP/^I:+?Q
A*3R=B34PG('!ETG6:%"E3I[50YPC"X4Q H8,>6R?:JB5 &C1[TZMP]PEF*L
?1G\LFM,I(^*#3?_REJQR;(*\0EPR)IOK+U78XJLB =*9:4996\H%<?8=\ &
;"?:0Z'82R:,I90UIFWF14-QJU:*_,=%HVV8L9P6$BQ"]GPKQLD:.7$0>J:=
))""U[^_1ZZ 5'M3,UP25(?>>\NXG,NZ++%+E(Q&%+GE4+V#;,KUA\I_4E.'
^S)<8,@-1,L@>"]P2\Q=Q,A3VD(F*,W8?'@KH;LRA0O1G,UP:;%\!L[DK'9N
;&Q^"Q[E#)> )8O2L\[P+*;Y\:*;#,^,B67V;,\#)\\7F<^+S"3^',\);&OK
(''U'-#OZK8(+<;M6\RON-"V5W,0G<U-K,#I/- 3K5 5G='LC*T<S;X?_QW2
(CW2)%W2)GW2*)W2*KW2+-W2+OW2,!W3,CW3-%W3-GW3.)W3.KW3/-W3/OW3
0!W40CW41%W41GW42)W42KW43-W43OW44!W54CW55%W55GW56)W56KW57-W5
7OW58!W68CW69%W69GW6:)W6:KW6;-W6;OW6<!W7<CW7=%W7=GW7>)W7>KW7
?-W7?OW7@!W8@CW8A%W8AGW8B)W8BKW8C-W8COW8D!W9DCW9E%W9EGW9F)W9
FKW9G-W9GOW9H!W:HCW:I%W:IGW:J)W:JKW:K-W:KOW:L!W;LCW;M%W;MGW;
N)W;NKW;O-W;OOW;P!W<PCW<Q%W<QGW<R)W<RO^]W,S=W,[]W- =W=(]W=1=
W=9]W=B=W=J]W=S=W=[]W> =WN(]WN1=WN9]WNB=WNJ]WNS=WN[]WO =W_(]
W_1=W_9]W_B=W_J]W_S=W_[]WP >X (^X 1>X 9^X B>X J^X S>X [^X! >
X1(^X11>X19^X1B>X1J^X1S>X1[^X2 >XB(^XB1>XB9^XBB>XBJ^XBS>XB[^
XC >XS(^XS1>XS9^XSB>XSJ^XSS>XS[^XT >Y$(^Y$1>Y$9^Y$B>Y$J^Y$S>
Y$[^Y% >Y5(^Y51>Y59^Y5B>Y5J^Y5S>Y5[^Y6 >YF(^YF1>YF9^YFB>YFJ^
MYFS>YF[^YG >YW(^YW2:7N=V?N=XGN=ZON=\WN=^_N> 'NB"/NB$7NB&?NB(
MGNB*ONB,WNB._NB0'NF2/NF47NF6?NF8GNF:ONF<WNF>_NF@'NJB/NJD7NJF
M?NJHGNJJONJLWNJN_NJP'NNR/NNT7NNV?NNXGNNZONN\WNN^_NO 'NS"/NS$
M7NS&?NS(GNS*ONS,WNS._NS0'NW2/NW47NW6?NW8[NP)   [
M
M
M
C

end


begin 644 cart.gif
M1TE&.#=A0 +0 J(   4#!=S<W//S\Q87%NGHZ0\/#_S^_"0B)"P     0 +0
M @ #_V@FROUJR>EHG%>2*@5OGX5!X>.$7)HQ+ %E\%?"L4QWYWO?=N2XO@N+
MI_O1&$#A4'<D>C+)W8C)S,VDTEZ0M-)>7,"2."8D3[?8M'K-;K>7U/1S5:Z[
M<F[X"X7'JM!T2DT&459F(Q4S2XEU9V9S@XT=5&.49!YZBTY!C&YH1E^2GD%)
MH#YZ"@ *I36M+'@?4;)[?7&CM[BY6(5;?89;87L[KZUTO*NB7!3$$<$8BH=(
M@E.0S\,S8;57=;"]3,?>G=;6E3E1"]@6VU7GR*>_NC   -46SJH%A!'Y2_Q;
M\ZH,T)O$X4XS 07R 8HPT%:\AQ#9< C03(XQ;Y9$:!0Q\?_0CG9QM/FI5N^/
MP84:_Z!,H9)&-D8;7-(R\8E0)QEZ[H"DN9 ESS?H/.J+2">@F7.J!!CM$7"(
M4H8"ASZ"8A1#&(52B6K=JH8!0(7S"@ 40$"L6 D !P!( @"KP+9K'P"4._<)
MP*1E[C9+:_=K*KCSW@)DJU"IVGD_ #,4&_B$"[T,S@HL .0NB[NJ[D VP!@Q
M0L4.&>IM '< NJ7(&DX [56L:0]MS?X-B\^O@\!_$V9>L+ES 6FV$+^]@#O1
MX[F3N 9L7#4JVE22&S,6#.CI<GT5D./F?'QQY,9<PXL/-+R\^;-*EP=,&!-\
MX;:W*5^(KI#^"_ #U?[=;WY[8 +_;;6'%V[2&066:AD4ET]92467E'#,J4<:
M?Z(=1U$R"J#7UG[I[1>,9#Z(=9M7#ZI'HFB72<@?@ 4$T&%Y3\&(H6@4U@4'
M@<U!=(=7,39HW0E&7:<40EXY]]LI40$H$(F!+;4<;%']R!F4XU5IY5\>@+@A
M?_]%2.-]ZMTS$(XG)MA094<6N("61WY68U,B<J;B:CFB=22=)&ZP98$G*B7?
M!.@Y8%]04N"(Y7  %O=6FS1R$.=_#NH#HH.W!2FAG\Z@UP:$3J)F9YE/K.-)
M6<Y=ITI21::2ZJF/#;DD!+N=!408B6:('&6,Y=,<92==Z6L\2!BZ9D/PE6=H
MF4%PBIRQ_RIN%VQSZ94IK)N'XF<IA^!]R=-VI#7K+6\J)H::E[7(D^:!U6JW
MK'>PFLALI8T6Y]^W$]3J!IF-_G,I5[<EV=2IQ 4\Y9*FKNF.:O0TI<.IB> #
M9"J\8?3KQ+=@-B"QF,WF+(C=PLLLEW,R- !2&0NV,<;+4OOQ<&&UQFW')I@U
MELG*"L<RCG%.-M=)-VG\[UUITCR<HYYB=F1G"FM:,K*<[J<NQ^M BM?0\H2[
M5;]O9><<Q&BE*M]<UXDF6]@53.>!?EQO:9YSHE+L=AD$=,?@8E\*.Z^90WA&
MYHMZ*PPPN! S"YN[=('_Y8J1#YWRC/RB^ZSB!YV;94&O(V@J7!/\41=@:ETS^
M+42QQ*$[M,WS?N?QQX9!)_D)XL#PJ%'VND/M/*P 2UX+A-YPTJP1SU%-.R'T
MVI,&FLSX]O%@O@GO9WJ.&5?6C:,U^/2'/G! 5=+2,_=Y&ۈ ];&<;99V[U$B]W
M&*V;YVOH?.%T53]"(<?:S8)IW64 HL=V!WGGL>8_7B)_\*"!YAJ%!\BAY"%.
M&8;Q5)<%3H0&"HXH0]N01\'LZ<-:%F21MG!CD/RQ[V63P5J 5O8?JD$(*X13
M'P'P'P'P
M*  )%Y"WAA"%H5T *0P!\"=QZ(QW^ 6/N JU4WG8UW1)-HHH!ER^=2%O-UL5
M24_0IY$5PSN^A'KM9Y(0_T:+*?9<>B@3J))]O!<36E=1/T&2)<EE\[..[JAH
MUE5?D0&/S@@."P=Z741]K>>4">:6QP5F%\B"199_/. I9695W5!_<'E@?7F$
M""D*FG MN68I&&-)N[%_$F>52IA\C B#?UE@D2ES84=,E==D?P(A.!8T&<)H
MN*8H)+B8RH&43^59@SB9^H6:E'D&0MD+4'E4_T(P/-(@"A   %,DK2*: E0E
MT1=F==B%JFE?P7D+')4=T(*(LQD5,Y@DX,80KS4AC;D5O>F%(S"<<6>=[#5X
MN]5,]X1.8<B=GH0$:8*+8 ,5*2AJB@F$TDF:1G4?IYEFV"E;5=F)7T!FBO9:
M[O_"-6&HCH.6(].YGM%I5$CQGF@6GUFP'N8  .AG&;EV5:)&+!V *G#$(%79
MDJ&IGEKQG\IDFL )GZK%0]"!'4*@*Z!#$( @/%JTCAI*%"N*G,9"H-LG>( 9
M!Z/75(RI4U68*@1UG'8XFL$!9PATH:V65;CQ-8]9#AF1#/W%E5RF>V697(!S
M)RC:.G 81@P1>&!IEVGE%)D1-L[ %D:3B[Y2)%?(EKYGGZX#I#<0AD5"GC93
M"]_T #>53E/%$W7)<$Z:'@X9%R@9%PMX,!DR&9DU+/5!+ DW%B,4(<]398J%
M GR$:W/Z'"NHG5>"EWBVD!917<.@<E* >3S"'0%EBT/_Q6I50 QPX  )=Z=X
M^A +HC X1&GD)F^$<)MN,34+F ^V^6W6P1OU=@!68J!UFA5*,GH46EU&\)(4
M,Y\"A*7$":*("*6[R376,3-:@T0ZZIPN$UR=!&Z! HQ=F0M> W925(:O\AO\
M,*@W!A@ R4/JV*Z_FECA-!;+N23[$R7<<I,GYRH1U!0Z"7RO8G VY!)@4YZQ
MV2_(.I);XX$>AXL08VYRQ*1;!GLCTIS<86MKD9A Z:YINJCFAJ"71FGSVHR[
MQ6VVTIFO=AH$DS:A-#%H*IK@YZPQ0';SPVCHB42O^J^A.##*^ #F.JS*>*TP
M]WP=9RBDDFM>11VA(DARDB(G_U ??1(VC)I8Y\=(FOEM"U!X.FLDWP@7N0)7
M,J.-?O6U-7=W7)LQG5%IRFEU(]H!3U<J#/-GQ/*<00N87X20H% GA&91/Y6E
M&DA:?@B=+[HF@J)WO:,S$/NS^K<1K$JD1FEF#O,P>8.>$YL#]+"?_G*;"7N(
M&8>=(8 :4NE\HM4<Y9FP;V:D<L&WQL>>+*JZ;?!G .DW4I(#(W-=F-NN"2.M
M>3BW?DNGRG"X?:N FPHNYRJKHT&S'<BZNQ>@]/@OGWH!(Y.8S4"J<"00UV.+
MN-B2 X&I[P:LNL9-*9I:W-M[CH).=J91Z)FLR)NZRLL&-8A-*T@'0&D36 2M
M"2)4)/\'K'I*)Z^Y6N%K@-3T/+&0)8S!4M3F*RV:O(F;>CBE0.NP$U4*5"8W
MG!^PG+%9DQ34O_4E9%RT K&FJKR9OD>YOFH@9'%Z5-&X4&+)4SJ@O8JWNZ2;
M-[X;PHD%+9V+L-[+9JKX3P>LO@G<:$S)#0&;=5(5J3&L9!)L9(;V'+P+NHJ%
MMX8P;;DB#D7\F$)J?%%CID%@H1>EA7DK%6O)G3_@P?!FG8EPBK@KQAC8Q#Y5
M$Q(#O"+,PU/YNZ$Q?PL\Q4=&QO-!IM V44R,HP(Q"Q:<5CLLPW'\@D48GS,G
M/8*DN:.%P;QQ((:U9" <HV]<D#MHH)189'9,A&:D)#-(:;K_-<B4W,-RK(-D
M?&6ID1R;;(/D%[EZ<ERBG)"5/,D;B,GIEK,;K%K<&W)NQL)N3,H "LPYZ(2(
M;&2JD*M>3%N.C&MD0:ZY1\M:&JVR_(1X_ ^!BCVKS,K' URC8:G//,O37,C#
M;(35W$.K0'&H>[Q;:H#VJ\S07,H82H\=VHQ@^U+V+!PV-K86(\@:5L_B2'/^
M[!M;*]!1Z)[@',VA-<K$?#7.:4-42KO\VL95S+)X2*G$209;97MN4[0.VKGO
M;,@)[9NZ><F,BI4^ 0M?\Y)+UP[ NF.90<?9&!LL9M&U-[%(TE^QC- C+0?E
M1<W\8HO'BM$-JW26^ X336R9VK+F__=+(G6JZ0RNIRM1XKS3XW?0Z=2_2.>/
M^KK$>'.A(-'29_JR0'B]G/I^-V%3(3W552W,)36@\PR%N,L[)OH#M6K"M #6
M,DH48? :.#9<_3JF^^ (/)73\)R>%0VC*76^;,Q0@5"W'/K+#:34J^D# /.G
MN[-32@ [PD35A@W'SVH1ID#%)AB"6E/:QA*Y#0U;N8BL%WS88MTOMTL=6 R%
M(E1LRT?8(,W9+L$Q6'UT?@?!72!%!@+*,/(RQ3(@Y+-Q6<"1YI)VS3LC_;06
M*!12NE!&L."N4+L(^!K/:ZW6GQV,[:G1XI=EJZ0:90';H'JQ N,"!V!OL_J;
MAX527$46A?\7N98&:]BAW*'@3OC0'L)@??TZ!@ @E"AD1ZSCS9U-R,C)AH^Q
M;VH*3X%,>G-G4L;P2@J[JS5+27\J;220&4#0FI& HND\-03>&J^\5'^]>DE!
MH2WIRV*X3]6)RW_<"![GW9XM0&/U:D%"5=3@XI(WD616P? !-/$K6&WB-Q.>
M7&C,#4[F*@N(EQQE1;' U*MKWSO[*G'K:ID<T=)\HWK(AO:ZLW*M3?WH!HQ,
M%3W4'I6#,+O3I<_Q,/J084;%#+,]%)Y;KQB.*B2Z/^@LTCGIB<E"' J#SV2]
M.B,(L%&K?VPXBF>R(-?EK/Z0S:OJ5!IY;$ADJXU8QJ2A*3(>553_ODK?D2T,
M2B!6&ZMITT^1VJ/?AY+,>;X>Z-0!:S _:)(UYH.BXDM.0DER(NB2[G!)'FW@
MHC98![]J:A6FJC,$N0O"&VCTA1R]11S5^!0%[G';G:F56=W$ 9*3HZ\SK<]I
ML:B?[>4.9E<N98BN6[A6V^L"]]MEW%;CJ*02C0:1>E?E^%+U_E@%SID4R]I+
M'8>"TW&!;HTD]=>T;NV.R=$)0[*(I>[KKM.%5<#1 (N-K3'F54 #UNP0+W&T
M:G@&@EKXV379+E:\K<K MN3MYGW'$2<],AIJJ\3^1W9$.1O.+/&<*)84>B;+
MW0R=$=HB2V1BY2,-"J\@,(_K^J!3@I*5_Q/=8'YY,R_EF<%JKC*[I#L?=<YY
M,&]4T-M +%4I4+F6WDJ_R_3U/K[TI9O(8(=:J:Z3+\F\=6[#08J[@$X9_4*S
MKRZW3>GP"]_6M#OU!/9O3RU+;[R7JC6).,!R2@?W8%GPM7BRQGOL4)#BPG?U
M/H>'T.:9H<D2GG)6IY#YE&YM-P'(#$_LK;:)3$<6#V*:28"I4S.&GPCHKJY0
M#FI_DI^XFB_S]8O"/63=]]%X.2QQ@!CZVG1N.1,7!& :%ZL=G)[X% [5BLW\
MEF2\J_]?+G;W\ S>3@ $NB(-R,JXLUWT%UUQNDY0P,\$?S:-SAGS5KH&@QK,
M@-XK<@^B^!'8LO^/]YBVH:62]VI_U(. FFJHA:-EF@@ )F9%JS2C@+TXZZT+
M_R#FA62)354#*0U+37 LSW1MWWBN[WSO_SR73RAC$&\"A9%R@D%H  ;P-9T>
MJ]CLQ'@I,)(+BC0:I4FW0:VMK&Z<4>'*6;)LM>_XO'ZOONK\86)(@5M29X8J
M-'16?&N-CV9VDH)@"RE,@6QN/8!]>A83!"P6(RA?+)60JJNLK3>=KS5C.Z0
M'@4CM: &3S-";W^N+<#">@0#LW9CH&!EHZA L%G2.\DN2@H$ @^6<)/%X.'B
M5=102&7$,'4"#$]23](2A.DUY7CVXSYO[O6A*=F!Z)EKA"]8DB]>*,#_"T.'
M"\!\$"-*]/;#SQE;&$,L0A,C28D '-,(*SC1#+]>A61$8??,0Z@ACTCBV+:+
M2L@%* F5W,F3E<QO'1$)]'9HTRF=LJ(5^\E3(#N;3)3(VRA(G4@^3&?P*R/*
MZ )BB'J*';N'J<6B-N!MR%7KIKHW0WTM)=OCG1,W;%#)P7 T(*>8=Q *4<(+
M&0J7:)[27<P8IM)(/,*& >O6*M)@(QMGD3!J3*K*%0&K^8)T,%Z*ES6K7HUZ
M$(RLKU%@\W)!=D)]'7V]P1C['FNEUVJO[6*"0RZVQ8TG7TZ\MIW.0']+9TTR
M&18+=;!1J4- GC[*1%^PDV<K#VRZ"'.AB>+=_XTAH3RXI:["X+86Q;N_RFF@
M?KI_ZCJ0=IH6H_#RC#KR%20@+^X1(H1\FP"0TS3__;&0-2VPT!\$<:GDU7VS
M=+C# !QB5XD(B4!5X8ICF95%=I*0,=DUN#&X22 0R"- 5P)^Y@F+,SQ%HP,L
M=%7@+TI)-IH$$T;#@F$I=#7,@$!6R9,]=JDXQ0-2,ND%&+/\!!]AJ&GBAGT_
M6ED#AS)LI$ I2F*&E6\.L"2(A+S0@9B:?$:$90SGT42>,QIB4X&" V42H0HD
M1D=AGY!M(LHLO0C1I W>B>B8IN?8^0(T-S8(Z:ASY0"7EC\\<>D\#'+*$02B
M9&0(F=W9<I%YI,ZX1?\I5 R)BJLMK)I%=\#>,-4*8'#X5):Y-KM*.?;5H<6E
M"]*7J"FY%7M5KBXD^"DI+,$&C;:0G??:+NCVYI"S[(I6Q&3B*9H4>):A)>P/
MDH&ZW8>@D>/LFY(T\^ZC_6Y)KAEA@3F)M^TV3&=N$^!"2F\#UPNQLB\%QN\,
MC89$VL'70GI07N5YU!QSRHV ',H=9,0R<0-8$#-&GFF7I\,XIQD#FW8*",C'
M U<;*A_'#:=+!B"'S"=-PL&A(XZ/QFFMN8EYV$UP9N:LM6.1HNFC#>W BP8_
M-LY75X2WV'+<RAJ8 1BI>B7[Z0H#W/'KT$!<)'0TBX"9#5N5;+/UX)L.#-+
MA'Z0/; VETF-Q1,KLZW!S$G+-:H1D+<0,W\O3.JVA05_AVJ2!]99%6?6$:YZ
M+!#C_87@25F5)7Q^W2LZ+Z]K=2-^N.9ZZE<+VYWB:*T1^*DR2OA=T^K,*PV\
MU%]WU 2#":J\$(.V!YB23F&#C?=USL:YQ%&50U5^(60:3T$I+J1S?O.CEI,<
MQ7AV\]JL5)N10>0@4*6SR#9!VB7RE[S05<-1C-A"@63W/?@Y,'\A&=+Q6M.^
MS[TH$_0H5O9&QZ?;]((V&1)>U=J0O_[TZGX9<HX#5T@2#@2%"7L26!PD=0#%
M/ (8E>@%T'IDL6W%KRJ66$'%_%6V^Q1O"L"HU#O:_S/#%;+P;$'231")I$(S
MC0LKO$E=$;W7.AQ L"=/X0Q.. ?#EY'@ &9LV<G2Z+(VSM"*3G2B_(ISJQC!
M05K?>%\D2!.M[5D&7L7ZHE@RX(:.H6-X?DEDP@:T2$7N[I%Z@U@C&PFKL070
M)MI 4QQ7QY2AK* VZ]!+0)2PP0M2I%J*^Q[06 >W&:Z#23:IG/^(YP"[!2=&
M&;G>)N'WIZ@04FQE.T02Y\4'M&1-;'$*0%L8 !(O^DZ"U^#"%A.XM\>ECX@#
M% .8%';-71(.@LR2I.X "0G)> =/-(G#+Z0@#Y383I [<>=-4L!$\%GP;4:L
MG\V@Z4U>_H&.#/HE_J1%!/\]8B9=,@05'CD%3[$T\E.\TH)\##JERN'Q>-6\
M9S^;99&-I;*!6BH2)1HJDFW Z$C969Z< !A+)HR/@Z$Q8&00^!B_:8FB&UW1
M$7 8 [:MH2@%E!),GS6@SFA'*DL B0V=R=)O$"85$(I:64 :!.<P0Q.@4FE.
M<_:S4.6.3#R-BCJ:(,-Q$$9#PS JC&(*P(2.-4NEQ-0CZHF%=#S533C=*H!,
M);T/\2Y$ 7.& 0QCUD^NA!F8H&(SV*11DNX$A$PC%%LDQYRTL5$YES4:C6P6
M5[WR25^/#$4=6\61Y'&C9R/U$T?"F@J0.18BRRC%;1Y *XVR3E73.J(1VP%+
M,AC_801Y]2QC.M%,4K0'K$.+E8Y&@9CQJ'9'0ZP/XZ#VEUQY 5FB/)2M;%N/
M9LHT0 <AX5D30BB9'>2BPF773FE -C,),R"4^9U94P)=274QD*1"%B[5^K"E
M6BM@CTJ!6Y^C KJFEUM19)4.]8661'*/JJX ;;82 ;T<O%8B B);,VS!6/#E
E4?$LX^,+BG6 SO,16!+E^DN\ 9-EBK">W *7_'@N/81L!$*NR T%HC#C [Q
NW_TZ@4;;&*KL;5UZ'('86K8&O^JXL*6TUA()N1D)_?A%'S))G<3-5U3#O4O
R0 Q\+KHFM#.3K[WE3$[>#AAOP*9>2A&&.H.187I.F>I_\&-<BO *:K4_ [-
2,0@?_*S/BM7Q+L0CDQX"<2;6XB12K_DHGO"J4/J3IK",^$I?"3SYT3[$Y_F
R&;]LA/6/;_XAO<T:(DZUX1NJN^C>=,MOKYBE^[<[Z62'C,YW>QG8+(*DA!^
#R+[O-$X!TEYW:B;H"T%#B@;F9:D%:<?MSR(2VR",\2A;&7;F-DULG&R -@<
@:-BX-B]I'NGXMU=,DJ[R:@[P47^<C_*E:$<+7J >5ZI3Z1L;K< >M8F T6E
8EG)S>"XW+$.55YRJ )8D&95/$5)7-I=M?H&V==3UJNQH;#J2D14A84!.22<
#:@\L+G-H<(M"4WQI3I)1:12_?^$IRU,)L&N#Y2RMN$QG*-+(L#2$KN AUJ8
H;(%@CS@LIG-+O*MIHTOSFTY6896"7+JNIZKP$NW*E]"N'+G*G:"AN8:M6-W
XUX9/4P%CAU@32--:=X1*@ 1)1ZM7?.(I6(_\GZ@]@ZA[6[[O3@0\G&-EN?J
/7K/-./^%%P0GG#W56!/!()UY,<N R':,5XM8:@WC#0,,;+A:V @#[;V)2U"
-P,1>1$N/O8!B9DY=ICA-"6/D006P6MQ5@BZN88ZZ[T>Y:N6MBZ?A%=.O,$8
@=/##EH*12K@OP$8';=Z.7:^HDFF&45EA0$4<%6?<#V\SO+%?'.JYL&&I_W:
<4$Y@6+_?M[2;WF@?*,-XQ)L7KDKOD_,%*JCOE0>GWT1\A<[=A?_IR+7-%J4
 &!)P#@)P7154A"2YWTD12W)9W74<VN[\S%$!C'0Q'<9 W:,=T "H&S+$".S
M"64EP,?2 M6]7XM5TL.4F8G9 <YTAW\(&)-9!ANY1F 8AT@YBDOT$XDUX!-
L2LX1F8F-R44002#,74J 5TVPQ#N0%AX9RYH%RQ,5D'PIU!&A(1;B%N! T2)
Y1JW87-B0"OIDT6=,S>3@%QB90C^ RZ^U#TYU4L8YWUYAR_(AW[C!W1+MPB$
$2MA)&N!,#,9 ES&E7D-0(A#5DO[M'Q'DUEHU&W@UG<F8%DN_P,'V95KB80Q
<($(E?9':,%FU8)\'C--Q09%D&!G>09Q;;!#\")8[6-^-.4&&A9,87,JO,=%
*((TLU1_$QAHLPA%_!=W+HA6@Z"'K4-I*7)R6W 2#M9C)G&".%,.I6:'6F0>
?U<"R($8 J8,*-1ONK:,%TA?HS$DQ_<MTZ<^&Y-_?7!.T/%_TQ-%4B<A*F0R
RX=ZVU5BZ;*/MQ !V^4RO]4206@E WED,B<.QQ=IT5)X,_%C-S2&^B<(+I8D
>J0898=O!<(.'0-3P/!YHG=)6IAZY^12W_(<HT4C+T5WTLA5=/%:08A8^]A$
VK0 RF8M1;%FX7<7$;1YZKB21<"0T?\@)5JV+Q\X!XJ2D#V5,)8BD@(C(]:V
#8A&4%]P )X3/?U4D&)WD*BF)_YW=U3"%L3#@$+F;@KAC:_&;_F3'4(D4IT8
&1.S316P MA607VS'2E (A)R "^ #!,32VNG'[IW8%A977.B"O6E71$@/@Y6
(V'PB<'%@2V0"X)S7B14:?"'7D04@L'R$*7(5VZ"B:>!) 18A>5Q(-F :WC!
3@RQ#;8V<Y_6(N[R+.,C1F?084/C*J^S2LAH==>#>R=2AS'U;L 87O6G=5QX
5O%(9@$HE*&Y(":55G%@4AYQ)];F.6DXDN/V>1K7DK')!ZMF  % 7G9Y"%P!
EE60$XSSF(G_ET-_*'XA\SZ8V7V\@PK$4G+@Z'J#M71,H&P9 2A!)S%2Z8?D
5HC^X Z\TBWN.3B#Z4.]\V3&616_ GGVPP.&%6[K<H< -V(FPP6WX'J795F4
"'BU$*(B2AP!-9G!F!+R!!EP\8D=^&\OJGRK)99-YU I:D^L(#1'LBLV)"9%
!9HQD9PCYI/JXIJ>25%PD8F1V549.&W<DT$R:E^JY&]FQGUDX9*.F(T9(3&Y
(&YO$DDI-&LW)6"R.&2;A8EN)WP7!XSQ,BVW9'8L)B_QQE'<2750F9.XHPZV
AB(WDZ3X(C.J$H)+L(!CU 8"&A4[HI?8<#XKL0>MV8[C,V*H4#=G_S6G_U*G
A?E:2U5S]0FAJ7FCH&@Z#@ Y5P"?:UHZ/IDI!VE1,0@P7ID$$FJID+*@M" :
G!EC7#=_32145/2GIP$[XY8[7:B3;Q0L_].@?5 BME(_5":KEWJE@&&5=AB+
]+)K%;$+8+E 1MI[9FE58H!__K*;,R4XN5AY!*@B9]!,-.JLX4"K^M8[]8.G
.7:K9@,L0A.*UCH:SXEY)P)!5[2%;3H-MM(=Y+%FGZ&([#JKF#I5HAI^0K(H
W3!P.(FA:]IK1VAMH-FD0!!5>3"1L68('')<;^>N"=N=5V*K>U839S4K[3&=
C'E, 22N?_%4XB$"?T>BRX&SR4%9Q4IH)?];H]!*=>U'I"B8>+$$0IJX.W[
AN?C$"J;#,MRK/= 4>T%I(+XL_Y!LDR5J2T0B;B@LV94'T"TL@'KD+QAG/98
'NM3KG*Q5E.BMFHJM.6(%U/A$.]GEE@KA$'+M0UK<51GHFO8GPU9%3GR/+,G
'C4Y#0A!;O_ A3C*L&K@I7;"65)!M'E;$EK+2@RKK>6TFLL%A>\!LY6ALDDW
#/-GF]!V(U%2@@:)K"]2)+B /!48:9>K4PM[D!FIL6&Y'U[Y@C*F&\SI@C4I
J0"KF_;#&SZ)I2)$!W&):0E:N[=[LD(+'9:[M;>88;:U#(1%G:>))X9"8W T
=XQJLH'FO4)4D^__M:W0N[>#A+(N:)@-!U+5^"[#-&S-RK9)T4XD*92BV[H/
PVBEXY%ED[GK*[68Z[Y:F)/].Y;@6+'/.U.7(+8 1K3*F[HP(++J5, L0L#S
QK>=L"#NL#<B,I!0"AALYTM& [9'H\(?P,)F]$9@JL%9&[UTPKF1TCC<59!R
6$2Q1Q\2DW54,CP$@+HZ,%$TUK"/LUH!\VBH*<._P<'.(T(V'%H=5L);2Q >
41]_ PI+)WB<<"$D=F_BLHY(!%A75D9P>K<4Z\32"YO3.XL05YN$B:LU\;!&
1V-2*8-F=WG V0\R&QGL1Q_8:B"=TT[(@#EL/!U0K&>XBT!;D8\W^5U0_Z8C
.<+%296"Y]EGO9L-%33'6ODCEJ>M$I3(>^7&'CQ.CNAO7@RJ@MP;Q%NVT+8/
YUBSG!.U_:>^F%$^CUI+*%J_I/S$-"S%-"4DE(@+&P1EZ*0Y<1E>G;Q;2\@D
 ,$-AY6O7H2_-]"I]T&,$TI=J_S+.['(S];(7T:[A+#+FFN-2FRN?YP&*<F]
N5I3EGN-2'2@T7DFYP+.WGS.;;RYD\!Z>,&$5TLQ?5*%8XBN\ZMO!N6Q@=9B
MT)0^:P9^"S0XAP%+FH_(2NDM9I?HZH=\[<_18<1*@RBV6B)6EJIG!S0#RT1
$8W2V#0)$H<JZ&4/*VU+PE$SV.LQKI+ ^90'P_]H"IT1GRG]S<%,?#^-J($;
3>'L")?#KTH:0&RY*>ML;H=T06GC#2XC5#,=U*S<K@B\?$C[&1[!L0LLT7Q"
M@_LE@[A :NX9K_ )U(-'6LF2/'S)TEHM#EFMO,_0 <!3*Y6&)C+-+GZK+G<\
M=C ';> 4UFXX:&==U^S;$WB=(N$6IU[!NTJ8T0/M.A2&G6V(3Q698VH #V.0
M2KK+V*K5V/!J%9)=NEK\AI8]T#CW2PLGI_AE0_"WU47L==*ZKJ0-:OLLSFI7
M<:71VDO#8TU4!G#"84:HU&1,1*V("6&2CAC 9+O=V]1=PS.A%BYFD?>4U<93
M'Z2!DIJ&RX4-L&DYEI?_((O<K=5W[;Y2W068_=< E'H#(@HD_8@O$XF9!6X?
M:M^@ (0*--WM:]K6_0?;F O%I2GIW<I+?,'6.COIK)15ZD@2OB":,$D1SHR;
M&5!LHM  ;M=#_2,.]\]QRD>,=,5J$LH\NJM *)K]AZ1DA*P1[(>ZW>'O*M1O
MC 129QE(;6'Y51MVQN A6+G;@5]]/ 7@2A]M-[DK\$'52^,]Z=CL?6QN48,R
M:>)+$TQ6$SB()7[#9\"Q]B:50K"'5)5.KM(?KCXA'J6CK,]-%1;R3=P^8,3F
MT<W_U(? UN1E_KA0?N/])A"HP+I)724<&K5/*[MK+-"F>NAK4L^VG>=<+>#"
M_\RMHNB-76Z?(L,-BV!^><UUXHW+N>QEXM'1%030C@YCD$[4'SMZ\"TRF8@2
MVEDQKH+))G@'5$9;$SK6I;YOIP[B.E!)4=L79<GCD*)?Z)@\-NLRKA?2W.9W
M+LP!,S-9G@J4N6YJNX[F,Q8)HE3C!!DQ-J$G#V<C%"?L_)?)C0XVT$6>,:M^
MT_[HILS/:?'=S+!@2"?LEPT=O64__8K<U 8 B$:T+$%1S*"K-R/?ZUYUU1WI
M+$H!^%ER[MTA"4X.D"PQV#)]7Y(5[=.TX<'K)KEB\5OP!F_C?$MSQ:'M0*(D
M\QD>E>Y,'R;M,]4-1Y':R.OQF5'M ;:F^(-SF)C:RO]]V19;J1F?E>#K[Q [
MDPR$YS+/Y@>,JQ_9F ^E\P8"[%9N)8*2=.C2%RBZUJ)EV+@2?#!]].P>X#&F
MK>D+]:[\\QU\Y>^8&.15]2I YRC0[TV[:+:T**^=G%Y/ODG/MTWJ/Y\DMAJ)
MX C&.=/#XK\F$@^(BGR6<C+RF^%X]W@_$205J<'T<4-'8/<FW%9"[$30*/OC
M,A^=L\LNB27]?P@Q[X[_9&=>5QIR"!YZ2>TPV >(]"6?[1U_3;W MK*.+[F?
M-RJ9"+A^^@0#]MX'S=X6\#197I9,D]MM786K,- ,VIR^[YXN[*S*!4IT[P8(
M_) K_)^<">INGVZ[ZFAO*SG_GE+]W($KE>A\MOH";S/[JOW;O^>%&4%;[A ^
M$UB8WU91EM<'O0:'3PX(0";6_C!"P0QP%^BGBY)@*(YD:9YHJJXL>[4P^L;B
M[%3-1^43AYTVFG 887R*EE]RB9,UFL2E$"=X,5ZZA8' @$:_X+!XW J2RV?S
MHEFI:  ?1>'= 'A!ZK,^0J!3=@5N6@4++W8.1R966G<L?5I3A1 7B7N6EYB9
M:)HI>41J;6P6;Y1U=%(DGIQA5W4W&U=7&Z,/C2*$MC2$40)S&H( <T\_N:O&
MQ\B=R:EI2%0*;J3!TH>+(ZK+4S.4 U)F.VTJ5<.0,3B'8ZW5Y[O9[N_P2O$^
M9'E0_Q5]I,/Z%L7R\Y$.C:OV@=JT@P83)CR@L*%"7QT0.IQ(BN&T<774E0/(
ML6,SC]AH?*NU!M*T+>6V ?$XY *#0!%=6B-)S@E*(N? X'M"")T=0RR#"HT2
M,EE1&/8VWI#01:6BH2TPVABGP*F^2BN7^K.YE41.+1E^2(-*5D37,SN0U&!Y
M=)-7FA)<HKI6=L6;2E:KU;1;$J<@G*9NP"Q$I6U=@&>'' E'$BO<N?,,\RW1
MA1$Y-X*V2CY<:Z<\F;,>,UO*2^X4S!BJ!./1P /GUY 2PS#RH!)C:#Q:@?P8
->4_NK!/+"Y7I>FDM+*Q\/H]>Q\]I6F#0XVN=#GURC<8-__!RY:W"=K@&#GU
MYFL+->C3LW8=2% .[O#98/843[$5TG2\VO:KK;_L(+=U8-"+K$P,$!7]9$%
1Q+W&0)+ 0-D4=U:?<FG5G$3IG#.+ 1TTT%3OJ"G'T@0%0 112@F=*)"Y[3Q
7H*7P%C"45R !<MBVR X8@A66($+?N%8.$D/0^BVW%)A];./D#MB@AX%CAV)
VX6!6-!.:)#%(^-H(X0"P "G< #@4TU&L. 28::HYAMSA+AF0VZN^"8I*PHC
%Q;OE0F0-4S.*-:7](7FDR0V;*F'H13*D.)>>.A9VX_HN,($4XSZ&1M@57ZQ
74^I&0 FHHZ& :JE20!515622I7_9H!C@'(AG1LY16JH>C$!E*2T$&=3I2SD
U&<M,TCES9"A:GE)+V QB*Q+Q>7H@(B+_"K&J'$A4>.EIHCX65;%:A A!ARR
YX&P?<E 710$Y E#/N!N$=8X&&59+#+4=IE!+!FATQ,%IU#%C[33>E<3<OAA
 %&\;=4+T'"U#;N&0"RT ["9O$YV;ZFK/1'FO.XHO-:MI>:0I':FP,*1Q[]1
$<&/#Q,K;[7%)M(%#J@6[(IGW 9<I %5ZB,75A-S_(DFV]3\ GLQ=2!GALN@
G"IQOF!0T 80X9HPQV]L^"T3A/3!KZXK,18)LE'R!2"\H@G=\;%U%#1'-*4$
&X>-LP3]_P7*+ZF\P*<;J!2>*[)RZ6BZ6W2SA->J&O*CA='N?'-+&9_*<[I8
;JPVO9G8-G<3/CZ[R$S&CM%'B1+YT0683U3@0># =:LQDJXPC'-BVH*1J8!V
P,OZY:LX;=]EHJ@JZ:"[B4$RW"8/V:/#LSKJ8XN%)"V-G"E2/^?TUSMD_057
NLR[,=3:?:=SR1;/=LF7]ILU\X+KB1'"SZ$TMUO33BPV2A<S://WO:-5\>\"
*<4"ND<:>/B.4HA('6NTD(C=\0AK%,A4L)0%J9DDQG?4DDF;'A&W3O'O&%O"
SDWN]YW'+8@]'[C=GCAQ'7*L;PU54U+SRE0%891L%D>[5:14@/]!HA@B+3_)
R/X^R(D0,LLE;0H 4K+E+@?]XG\@9*$"D<2FFY#"2.UK$@E'R"#",4)B3(-9
JP#C1%1%K60V)"+1A'"G7E2@&V6;#W[FA[^-V:TE^R'.E"CRM-852Q0C)& A
="""?'D9A:R_1GX3BJ$:!N>!L,&02-4BGCSL2917W$!D_;BC$&6KQ(@^0V'C4
E1Q'KDLF5WA$ 9_ R$:*B@9>1-8@MCB"F@%R%&FT)!OSJ!7LG&1*[/.C\RC0
#5DXIQ=]^] "<_;*3\ "3.H(S=?"Z$HP&$I66R(>+5"C2SQ*\0YZ:Q S:8@%
O_5Q&")4AH"LXQN9M*N;U6S4"1JQ/%G_D@0<Q1 >SZP&3V_2H!&TO P4'2A/
U[7F+S>,R?:JQZ;L3=*AVC.(# D:SS&6D(L&4&(PD&9'UM@BD:G2C3VIV;]_
DN:CY6)83<;T,HK-JYZ3,@W8$GB")VF*9W8[@@[L "G\5103DB'9?P8& 95*
P&N2^(M(3T8&[+1RE3=194N]YSXDON(YP=I%0!]HF=,\;@@'>HDAD#K1G\8(
!5,*Q7Q4MA6$$@^A3!V;I, H1&!&)UY8'6>97GA0U#Q1-315QE9[U37()155
"YK!(_II5EP-B!CIH0W+)I5/2D3-9WZ(JSGVP1ZEJ*(R6LTKF4(ELYO00ZHX
A<13X]?,(B76_X8_V< 19-I8G<WS%0097J%,0YXL*)$#86*L"^[&H,[2S48#
7 V&3$-1E\Z+K0G,U.)HFP*Z$I>D))!#VT)6U"G6UJ)HE41G\352*>03F0_]
4B4UN\O4CNQFA1V( #A(V>9254\=F&TP(+F(PCY2="H4R1:JY $5IE.X/PWJ
S=R9*KDLU9/XV"2">>A#.@1#A&--#YIR=57'"A._P_M+-7 H3>PJ(9V;G6HG
2 RXRCWQNS=5,5,0PK5.*36[2L+2?4-WW23Q)()+X%2Z!N)"YG#5H (XT'JP
>,FTQ2# HB*7?6$L'&#=EJ:6+8CJ8 ?5(B1I@BWK<M,B81+"*.D*;_]#'U!
!K_1T@H8KTL/1%5TD(5>KTUS=LA$W<AD*J\+KAJ*UQD)L],% )-2//VJ#P[]
#D1!5XXG[2H@IVSD'2$K;V?:A;J\.5@-R;@Y0:R*H"?,/R\8!I#G0J"3AY0/
F?YEM45TG%.)1-29UH;2'B[38AGL+YT1+,5GKH<72P5H/\>@9N&]'XJOQ4#*
%$IV.XP,NE*Q0U,+U&IZ;=+&.E ^:QK5''W6R;L2.PAC]X8-;<EG#QZMF>@M
,R+E,B K>C FO?W#"$:ZFMK.99QFPIHRE2;NMO=K[MXTT-,5N]9O/=/II,@9
9XU&EQN$ 5C=@DL6Z @'KC\-&]NLS+8F_@[_VHQWN)*@N."/E3/';WEM2MF;
1]@KW0%C8"@V!"(S2!26#M%9YEP?V5$OCX['F"5F<?C<FJL\.<K-4@L"QTOI
F+0&7I09<J#J)%?\R,!E.7=D?9>ZS-@))X77*==8B>W7_EEZ 44]*=%4YN4W
=4W&L%5M'GO5#:AUX[B&=_1@)DJ+\SS3&C)9=4-^P;>3W8C8U?X8'716I1)*
N[@W++:ML3<J5/B4Y\JG7 [C5;2>S(]*"2_ >Q;]YZTUZ?U 5^M5PY@-7"=4
G#>JL>1571P.>O3,:<Z+0P>INU#<EIOS@VJU*C!'"/UWP/TYA;UK -D>):KR
J5P9E5A?9$I:+&/G_WL +^:.)*%=X1X(O_9$>=W2ID5@*Y#9+EJ'?FA%@FTB
,Q[IO9 :_;11K3I41]:"P?V5(!,;07!_P^=5)@=PR^0RYU=J^98OIE5*9"<$
0]8I/U):QL=XV8$(G^(+?(-84R-!MH<62'. O= FJ;8V)G4I S<SG=$Z"R@=
M._!;D"9\&T5=V09_0],B$"-08D> P<$8C;-@;3=R;4=R[X,'EF-WCG-M03=:
M+Z@?ZK%F+^8L_C<9X&52.X=$+%8J:8:!8D0L:'-]0T1RZ24]G5920[%QHQ)Y
MMO86/$<@GP%8S2*$E+,>\;$K72485$<>_8)+Y1&% Q2&JJ$11.B%1S<^8/^1
M1E1G"6"T$ZU&=-(&%6IX>(P"<2!06IX&1DK&7?)U=J['57&$9]@"<U@20RO
M@N+E0DGV"#?'<7Z6!U7B!<Q&'%!G4H]0(_34,$K(*D_H I1T$M0F<\%B"W'@
M(/V0 1.(4\*89QI@$0U59]3 -\SX)B>")G!P $G *83A@]J&A#*W/6XRC=/3
MA9&V?K=W5I*X?#MV;#1Q@3.H%>9"&(J3!'+0?[^7?M_!)S2Q7EOU+W(V?5F0
M$]*58W#0.>=H5DFQ@H\!3ZD&<;%XD(>B(.K(6C1'8EU8#,"H&D%E3Q?3-]LU
M080G&XD'/#)T>EF$@)TQAX 3&%8U?;6%#?S&<P[_((/C!Q?08(,1EXY]YUP"
MUC;Z> LU@HRXLEJMEE088TXM*$DRR5:E5XO0$6<(]VY5\2!KH&0;0).&^#(M
MI&R9 $ZQYY3@(Y$[N8Y6R#VYD8#T  B?2 Y1XAA:%EC944B,H@".!P5Q=$OA
M!G"_1UUOYPQ9*2]A)WG'D&IU)V\ZR7&@,C[!11<[T8==\C\GYRI]8E-[ 7U1
M.9'O&$K@])1_B9F'L7M+=)A%D9=E682M$W9!2&:NB(=_UY,XV)GJM%>L(A13
MAAJ[A%BFUW5^)RT:!W*MR7M-!INQV8WF0YL366Q(06(Z5@,FDU\A@)47U7-(
M]WY6R'S"68!0.)M!@6L8_P(P]YB!T?F=+!>=J!6<'Q::KWF=U D;H$D_VZF.
M[+",#B4G=D8GZ74BX7AG0X>=[NF:ZKF>K]&>PR6:7O&=+=! L(>/E$(XE4%(
MJG5[XVF>OXF>UOF?$YH? EJ=[SF6=X438_)\=;!IT8<*LL4S4'9E)RBAJ ><
MZ6FA*YJ=Q;FA*L987^9>OQ-)^!)$&!*C+]J?+.JB%RH=&3IV:>B95_<T3R*B
M4#" %S.2NUA0_Q1M<WFC8PFDJWF8E\>+R"!5&5<[NCA $( +E""E3_J%)@ -
MHH %*@,TM6%M5AJDP3&DPRFC<@I$J=1M(3!;'"!BW#BG/0IIVC$6P0<V>OJF
M?_\:I]JII=\T$!:6A-X5AL 8&BZ)CB<9 J;(BLEB3&@VE3PSA8!2I4 JIWL@
MJC<H?E<ZG:KFEVSB!?]GF)6*!W]A)[-G1OCA8PY#JFJ$JQ%HJL8)JE<W7YT:
M<PT!)BO2@O;'JW#J4ASI,*]&D$56JX9ZJ.R9J.93IT=EDH;6&J930):8DZ\:
M%U&30K>H/S'U.I&"B+KZ0>GJFY%8I+Y*<L"'734$HBXWJ&AXGCQ2'#GA%,-&
MK@'H8^OZ/0&+JEE:K:^!G$-AF_@J3X525E:23!!K)<:$0]$*I3LRL!;+%@K+
M&1@;GO:4"X:Q/ VPB548D%/T:*<*FQW;HF7*$8)4%EW_T*_F 5D+JY<6D$+Q
M=F7O^I\K6Z$M:ZJL1Q8GRA)P(Z)D64O/^G2WH4!I@8M&RK/.0ZT=(1=&RQ*X
MT;._@P_..%!&5Q-G@I,BX!A8:U"RR:/%HY'BF+9P\HQJV[;20R<1>JIZHU5!
M&)ARA+#1.K8_BJPLT8I]>C=_>XHN!(Q,<VHS1;7D9QGW>!\52Y$P.K5BN9;Q
M$'MU\148V;5-)V%PT82."Z2]"6*(L4GSX"+."9'+@+=8VI?2R725V+D^:J5I
M4;4'"V:GFRU8JW$FPEN!*Q*[VUZV03.RFY03DD$5"P4!X+;(NR:_F+S,V[P&
M(8T) ;W.FV?+.[W)ZXQ*TU!N__B&L7)=C>M^(_(S]FH)5\01U9=9K,JMZMMZ
M]M:^ZZL45_N#/[F6"0JRWFNH"4I#P;L*WP8/]Y"X=>&E9>&D^3L@D7F_Q=N[
M9[!W"JQX'2%"= 26H[L%,ON9,CF*CVD$R^FZ ]JXK8JA?_F[\6K!B$I-'P6\
M:4>FK_N]9=O +.R&>FL"E"F8[8@^PXO +PRZ.8RE''NWK;BYI.%.=Y!!2[O#
MCVO$,AJ@S"F3!R&-@6!<*NBDCTF+6NG"2*RA5URP9>$)[R6Q,=MA=9-.*GRF
M5J)H5IS%*XS&*#BM(! B-Z:+,+FSJU0 &K6C;:C&[HK'?%L7GC!?+UL=!F$B
M5'!II_^H.E+XC7J<QXGLJDI< [I1"7OD95X[OJ2!5&;)AFBYR.VJR6O<R*EJ
M!X8#+[Y0(+"#(STIE-R&<:;+R:/*RK6;LHR<N37J"MV3 7[P.8OH:14W,@#K
MREKLRT=2N$*Z>@:#*D:22N]BPRHW-)%B3Y,*S*8$S;VQ%)AXM.Y:!1NX#4=#
M'WN*,4@CO6J[3RD1@M)L%.4L@3F;SD][,H#@(D: .LQ3-.8J*([:*SZ&8>?<
MR?G<M?!6?=(J%!(T"?O5,[.D2(0@IN>$K0 '![4,E?L<10_M" XKL1G+QV_<
M=-'#4:LQICU2P%AHP_,:T>8LTBQP($(8M#%< KCP><@!DE__F!DM%Z6*"X\D
M?:\UO=#<=K,OD])<4GJ3?("6(1&#%K<JP'8+>=.QAM1.X!KHM<&P[*VT"X9,
MFU2H&)&4K-2[BM44$@3T%S^==L:U^6(\:75:G0W,<L86:DLOH$3%-%3ZTZV2
MF]13)5GVJ5XV7=;@@]8N"L !V0:\I;-CYJO=RFBMC-<CK<YE'9@H=808;*Q7
M#5XAH6R[R]/Y#(F&#5SC6#I6Q%=Z[8(9+&;<2-GE[$;>9=B?EZT\@%K\@,D-
M(\&4:KAYN,H=?-F]8[VV?=NXG3W654!L6]=PLA@*O'$T3*FT+47%/21P&"RB
M"QQG6-B?YG&8<]S2G3E!7"<)L;="_R(^ZYR3VS?=QTTS?)#:1_W802MR->P6
MHLUEC^W=['U1'7JM?, XN9ER?AF;Z<W!FN*.[:W4R!.B$UNB)Y$<?8)JPRU&
M]WUB#!G7^_W0K7"3-Q,A$/.U8\?:)!!#@JI\PNT.^++@'*Z:R=4SQCQW#59X
MX.L5':2P%[3=^IP.=MOA-1V8L[PZ&85M@J7@G\%2<BG'V< OS9UBG>WB:H=S
M.<9W/_:'4-,)@Y991'K@A5"?TSBT0%[3GII<W)-[8IV9<R$9MBH/*:[C4?[E
MNL8::&H9A/V.YMA2=LD>=VJ:I,+D8/[F*5B9K)NJ6> 2Y8E R/8"G_+!!>7F
M</[GE\D?\?_F#^@%5T=#I43BOO8-Z(S.,7@)ETR#'K9<9.EV<F,.@4_=Z)IN
MON=MEX@M*0S:S&5\HV(5O1&QOZ"WZ:I.?.E;VA4CSZ;!4A<2.TF'?;OBYZN>
MZRS7ZF'D-S-1AW1V,#$-+FNBXKE^[$O(U>MVWCSQ?:C&>C+.9@&G,CL(3BSS
M;[B.[-)]VB5>#M>"5W8[$AV29I_'+O9Q#L3ZC"%B0RGZS]K^[LYDPQM]D24!
MB!87HM%C XF0=P\0(:@.U&KE&,V=[?"NU6/%8$G2$Q:&$J3]0WIA64"R1;89
M@.ZG;H_*F@6?\:^96$JT+SN*IK)52=UP, N"&8G+#A,]8SG=G,;_^$L:1AX:
M'_-$$0L(WR&D:9;U/G([EW1+6C<OHU'D$T0YUJ^41O R+^4_Q""V7&T:J3&A
M+*;:S+@"'Z)U([OJ@<P/#HEM]*U'W_5NV):[C>@*:FM3+ZA;05_9J!NV6O1>
MW_;-8:8,QYG,CO&/N6: H_8'9:[\Z?9\GZK68A)ETQ1P1=1F2ADU]$ZCGC5?
MXFY.;<U]__A1]>F7%@[#05N$7]%B?V .#-1C/]:0__GU_D2V;$RX%!BD?UG?
M0G'UC/DEP#!&U;]<#_I\WUES\*%"Q UH,E^**7N(B7 GMY5Y&ONR[_6BC%>'
M0-!^2*[#\O#O:KA/)4+07ZK#W_5;%C)\_TIY(;9G" WQ>_\6X:&ZJ-A"-3O]
M;7_+@PO&(-C-H$'D .IV_F'QXT_^1\_MR/"YD/OC\B_2!C+C[Q"["T.S"&"Z
MW/XPRDFKO3CKS;O_8"B.9$D)"F&2 >"^<"S/=&V[!0ZL?.__P*!P2"P:CSY4
M1FE@]I1.I'1*K5JOV.P%I;HZ!=$L:JPMF\_HM#I-5H2/3#![3:_;[_C\TAQG
MR]]Z@8*#A(6&AXB)BHN,C8Z/D)&2DY25EI>8F9J;G)V>GZ"AHJ.DI::GJ*FJ
MJZRMKJ^PL;*SM+6VM[BYNKN\O;Z_P,'"P\3%QL?(R<K+S,W.S]#1TM/4U=;7
MV-G:V]S=WM_@X?_BX^3EYN?HZ>KK[.WN[_#Q\O/T]?;W^/GZ^_S]_O\  PH<
M2+"@P8,($RI<R+"APX<0(TJ<2+&BQ8L8,VK<R+&CQX\@0XH<2;*DR9,H4ZI<
MR;*ERY<P8\J<2;.FS9LX<^K<R;.GSY] @PH=2K2HT:-(DRI=RK2ITZ=0HTJ=
M2K6JU:M8LVK=RK6KUZ]@PXH=2[:LV;-HTZI=R[:MV[=PX\J=2[>NW;MX\^K=
MR[>OW[^  PL>3+BPX<.($RM>S+BQX\>0(TN>3+FRY<N8,VO>S+FSY\^@0XL>
M3;JTZ=.H4ZM>S;JUZ]>P8\N>3;NV[=NX<^O>S;NW[]_ @PL?3KRX\>/NR),K
M7\Z\N?/GT*-+GTZ]NO7KV+-KW\Z]N_?OX,.+'T^^O/GSZ-.K7\^^O?OW\./+
MGT^_OOW[^//KW\^_O___  8HX( $%FC@@0@FJ.""##;HX(,01BCAA!16:.&%
M&&:HX88<=NCAAR"&*.*())9HXHXHDHIJCBBBRVZ.*+,,8HXXPTUFCCC3CFJ...
M//;HXX] !BGDD$06:>212":IY)),-NGDDU!&*>645%9IY9589JGEEEQVZ>67
```

```
M8(8IYIADEFGFF6BFJ>::;+;IYIMPQBGGG'36:>>=>.:IYYY\]NGGGX &*NB@
MA!9JZ*&()DI7 @ [
M
1

end
```

```
\H02::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

The 8 bit Modplay 128 Board, a three-diode addition

by Nate Dannenberg

Hello all, Nate Dannenberg here again, with an update to my original schematic and assembly instructions for the 8-bit DAC Circuit described in disC=overy issue #1.

This change will require three 1N914 diodes, and some solder.

What we will be doing here is rigging the DAC Circuit to trigger all four tracks simultaneously at the moment track 1 has been updated and clocked. The point of this is that, when used with my MODplayer software, all four tracks will output their samples at the same time, thus eliminating the switching and mixing noise that the circuit was producing before.

Those who used this device for only 1 track output will probably not hear any improvement, but multitrack players will hear a significant reduction in the "ringing" sound that accompanies low sample rates.

1) Remove the jumper or wire that connect pins 7 and 8 of the MAX505 together, and move any ground from pin 8 over to pin 7 of the MAX505. We want to take pin 8 completely out of the circuit.

2) Connect all three diode's ANODE leads to pin 8. The Anode is the "arrow" in the diode symbol, the end of the diode opposite the line on the diode's case.

3) Connect one each of the diodes' Cathode leads to pins 17, 18 and 19 of the MAX505, or to the corresponding pins on the User Port plug.

That's it!

Have fun and enjoy the wonders of 8 bit four track digital sound!
--
For questions or general commentary on this article, Mr. Dannenberg may be reached at the following addresses :

Internet   : Bowes1@cris.com or Bowes2@cris.com

Snail Mail : Digital Audio Concepts, Ltd
             c/o Nate Dannenberg
             9804 Northcliff Drive
             Dallas, TX   75218

Phone   : (214) 319-9879 data, or (214) 320-1386 voice

```
\H03::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

The Virtual PLUS/4 :

Upgrading your C16 to 64 Kilobytes!

by Martin Gierich

How many times have you heard from PLUS/4 fans, "Your C16 is only good as a PLUS/4 parts depot" ? How many times have you been offerred money for your C16 by these same TED-chip-seeking adepts? Disregard temptation because now your C16 can be turned into the pseudo-equivalent of a PLUS/4 with the ability to run a great deal of PLUS/4 software. You see, the C16 and PLUS/4 belong to the "264" series of Commodore machines. The C16 is actually a PLUS/4 with a few chips missing, notably the 6551 UART (and userport) and a full 64 KB of memory. In this article, we will be upgrading a C16 to a full 64 KB, allowing us to run many PLUS/4 or "generic 264" software (minus userport-terminal programs and some assorted misc. software).

:::Important:::

Before starting read this carefully and check out the schematics in your C16 manual. You should be familiar with soldering. I have reconstructed this, it is about 10 years since I have done this. I can give you no warranty, do it on your own risk ! Neither myself nor the staff of the disC=overy journal is responsible for any use or misuse of information presented in this article.

1. Buy two 64x4 bit dynamic RAM chips like "TMS 4464" or "41464". They should cost less than US$10 together.

2. Remove your two old "TMS 4416" RAM chips from your C16. They are labelled U5 and U6. You might destroy them, but be careful to not destroy something else ! I have used scissors to cut the pins and then I have desoldered the pins.

3. Now solder two 18 pin sockets in where the old RAM chips have been. Again do it carefully to avoid destruction ! Check out where pin 1 is. Then plug in the new RAM chips.

4. Check everything again, then switch on your C16. It should still show "12KB free".

5. Adress lines A0 to A13 are connected to the multiplexers U7 and U8. You need to connect A14 and A15 to them instead of +5V at U7 pin 2 and U8 pin 14 (but the +5V connections at pin 16 must be left). For example get them from the CPU (U2) pin 21 and 22. First scratch the +5V connections. U8 pin 14 is connceted on the lower side only and should be easy to disconnect. U7 pin 2 is more tricky. Disconnect the line between pin 2 and pin 16 at pin 16 on the upper side (U7). Pin 16 still gets +5V from the lower side.

Now you have two choices:

6a. To always have 64KB:
Connect U7 pin 2 with U2 pin 21 (A14).
Connect U8 pin 14 with U2 pin 22 (A15).

6b. To choose between 16KB and 64KB:
You need a double switch (or whatever this is called, it has 6 pins). Connect U7 pin 2 and U8 pin 14 to the two middle pins of the switch. Connect U2 pin 21 and 22 to the two upper pins of the switch. Connect +5V or Ground to the two lower pins of the switch. With that you can chosse where your 16KB area is located in your 64KB area. I have used FB13 to get Ground and FB14 to get +5V.

Keep the connections short !

7. Check everything again carefully. Then switch on your C16. It should show "60KB free" if the switch is in right position.

8. It is a good idea to replace the 7805 (labelled VR1) with a 78S05, because the 78S05 switches itself off, if it gets too hot.

Simple, eh? Enjoy !
--
For questions or general commentary on this article, Mr. Martin Gierich may be reached at the following internet address : uj3w@rz.uni-karlsruhe.de

\H04:::
:::

Continued Lt. Kernal Hard Drive Support

by Ron Fick

(rfick@nyx.net or Caped Crusader @ Batcave on CommNet)

Probably my best enjoyment from this great Commodore 8 bit hobby is helping fellow Commodore enthusiasts with my expertise on electronic hardware. I have been an electronics design engineer for 20 some years and I have an in all types of computers, but the Commodore has always been my favorite for hardware/software work ever since I bought my first C64.

One peripheral lacking on Commodore 64's and 128's was a means of mass data storage. Disk drives are slow and unreliable, especially for applications where constant 24 hour use is required. Commodore did make some hard drives for the Pet series but none were produced by Commodore for the newer 64's and 128's. It was time for some aftermarket designers to fill the void.

Several designs came to market, but the one that has stayed popular the longest is the Lt. Kernal hard drive system. A couple of fellows named Roy Southwick and Lloyd Sponenburg came up with the initial design of the Lt. Kernal. The story has it that Roy bought his son his first C64 and datasette and quickly found that system drudgingly slow for loading and saving programs. Even when he upgraded the system to use a 1541 disk drive, Roy was frustrated. You all know how you can take a coffee break while some programs load off a 41, which didn't impress Roy since he and Lloyd were software/hardware engineers on mini-computers at the time.

The Lt. Kernal hard drive system for the C64 was introduced in December of 1984 and it's main customers were businesses who found it could fill their needs and was a lot less expensive than PC's at the time. Since the Lt. Kernal is designed to use the 'parallel-oriented expansion bus' instead of the serial bus that disk drives use, it's transfer rate is many times faster than 'normal' 15xx serial transfer rates. For example, in 128 mode a Lt. Kernal drive can run up to 65 kilobytes per second. Eventually, a family run company in Kansas, Xetec Inc., provided additional design and manufacturing of the Lt. Kernals and they became quite popular to hobbyists running Commodore BBS's. It was an expensive accessory, retailing -then- at over $1000, but then hobbies are supposed to cost money, aren't they?

Here are some of the facinating features designed into the Lt. Kernal and its operating system:

- Fourty-two additional or enhanced system commands
- Automatic power-up execution of any application program
- Up to 7 files can be open for reading or writing simultaneously in addition to the command/error channel
- Built in backup & restore software using a speedy "Fastcopy" utility
- User can set screen & character colors
- CP/M software can be run on the Lt. Kernal system
- DOS allows Key File indexing which is not available on disk drives
- Can run copy protected software with limitations
- Ability to use up to 15 C64's or C128's on one drive simultaneously using the multiplexer which I also build and support

Xetec performed a great service to Commodore users by manufacturing several peripherals for Commodores: printer adapters and software, plus accessories for Atari's and Amigas and even the Macintosh. But eventually, the market for such accessories dwindled and Xetec decided to close it's doors in April of 1995. They had an auction in May of that year to sell off their inventory. Prior to the auction, Xetec contacted me since they knew I was in the habit of assisting Commodore users with my electronics expertise. I think Xetec genuinely cared for the future of the Lt. Kernal users, and since these precious unique parts would be forever lost to Commodore users by ending up at surplus electronics dealers, I was contacted. I emptied out my savings account to rescue these parts and even though it was tempting to set up a business of Lt. Kernal repair, I knew if Xetec wasn't making a profit at it, I wouldn't have much chance either.

My work with Lt. Kernals is strictly a hobby. I charge folks just my costs and eventually will be pleased if I break even on my investment in parts. Besides, if I made a business out of my hobby, I might just suffer burnout and that could ruin my hobby.

Word of mouth has provided me with all the work I can handle with the Lt. Kernal. But it is a great pleasure to provide continued support for these popular hard drives and I hope I'll have enough spare parts to continue to support them for a long time to come.

Currently, the CMD hard drive system is the only commercially-available Commodore hard drive system and it is better suited for the regular Commodore hobbyist since it is more software compatible to your normal Commodore programs than the Lt. Kernal system. Commodore bbs's are the main use nowadays for the Lt. Kernal since most Commodore bbs software was written originally with a Lt. Kernal in mind. The Lt. Kernal Multiplexer allows, for example, the Commodore sysop to do true multi-tasking with his bbs without tying up the bbs for Sysop functions and even provides the capability of running a multi-line bbs on the Commodore. CommNet is the largest network of Commodore bbs's in the world with around 50 Commodore boards in the USA and Canada all networked and a good deal of those bbs's are running on Lt. Kernal hard drives. C-Net 128, DS-2 and New

Image bbs networks are all linked via this network and other Commodore networks are encouraged to contact me to join this Commodore information highway. My bbs, the Batcave (303)/252-0735, has been operational for over 7 years and has around 700 active members with around 100 of those Commodore 8 bit users and guess what, it runs on a 105 meg Lt. Kernal! :) With Commodores getting cheaper every day at garage sales and support available for these great Commodore hard drive systems, there's no need to tie up a Pentium to run a bbs.

--
To our knowledge, Mr. Ron Fick is the best and last source of Xetec hardware in 1996. If you have any problems with your Xetec peripherals or would like more information on Ron's stock of Xetec products, do not hesitate to email him at rfick@nyx.net

\H05::
::

>> The Metal Shop ---
+
--- with SMS Mike Eglestone <<

SysOp of Diamondback BBS (305)258-5039

Senior Master Sergeant Mike Eglestone has been a devout C= Hardware guru for over ten years and is the Sysop of one of the most active BBS's on the Commnet Commodore BBS network. Mr. Eglestone has written for illustrious magazines in the Commodore 8 bit community such as dieHard magazine and the very distinguished Commodore World. We are pleased to have him entertain our questions pertaining to hardware concerns. As always, the editors, the staff of disC=overy, and Mr. Eglestone are NOT responsible for the use or misuse of any information presented in this article.
--

>> Dear SMS Mike,
>>
>> How do you run two or more SFD 1001 drives together with only one
>> interface? I have a Quicksilver 128 hooked up to my SFD and was wondering
>> how an additional SFD could be added to it? Could you tell me please?
>>
>> GF

GF,

Fairly simple, you daisy chain the drives with the appropriate cables. It works just exactly like the serial bus on the standard C= drives, but you do the hookup externally (cable to cable).

My cables had MALE and FEMALE plugs on the same end. The cables hooked into each other at each plug point. Male into drive, Female side out. Next Male pluged into the back of that same connection, and so on down the line.

If you don't have the cables with the double connectors, you can't make the parallel hookup. I don't know where you might find those cables anymore. They used to be quite common. I gave all my old ones away about six years ago.

You might want to check with Bo Fain at Centsible Systems. He runs across a wide variety of things that are not currently available.

(318) 687-4613
--

>> Dear SMS Mike,
>>
>> When I try to load up a program with my 1581, the power light starts to
>> flicker and then refuses to load. What is going on here?
>>
>> RB

RB,

If the power light only flickers when the drive is turning a disk, it sounds more like a loose connection inside the drive.

Just open up the case and check the drive power plug inside.

It's fairly simple. There are NO dangerous voltages present inside that drive. None!! 5 Volts and 12 Volts... That's IT!

The drive unit can be seperated from the mother board in a few minutes, but you don't even have to do that. Just check the drive plug. Run the drive with the top off and LOOK... It ain't no big deal and you can't hurt anything.

Just pull the two screws (on the bottom) that hold the top on, Turn it over and lift the back of the top, take the top off... The front may fall off too, but it just sits there on a couple of pins...

Don't be afraid to LOOK inside that drive. It's easy to put back together again.. Hell guy, I run two of them with NO tops at all.. Gotta smack one of them now and then (to get it turning); it's easy with the top off.
The plugs to the drive are obvious. Pull them out and put them back a few times.. Might just be dirty connections.

However, you may also have a dirty r/w head. As Mr. Ron Fick pointed out to me once, the power light can flicker after a file misread. The solution there would be to clean it lightly with drive-head cleaning solution available from drive cleaning kits at Radio Shack and most any computer store vendor.

--

>> Dear SMS Mike,
>>
>> I have an odd 1581 problem. My 1581 does not always recognize disk swaps
>> and sometimes falsely indicates that the write protect is set. There is a
>> small microswitch that detects the write enable and presumable the disk
>> swap as well. What does your SAMS guide say about this? It could also
>> be the circuitry. What is the part number if the micro switch and is it
>> generally available?
>>
>> RM

RM,

The write protect sensor is nothing more than a optical cell. It is either logic high or logic low depending upon the position of the write protect tab on the disk. No moving parts involved.

I would venture a guess that you have one of the old Newtronics drives in that unit. If this is the case, the only way I have ever been able to recover from a failed reset is to re-initialize the drive then try to re-inseart the disk again. You should HEAR that dust shield slide back out of the way of the read write heads, folowed by a very short motor run. That is the indexing cycle for track 40. Unlike the other drives, the 1581 uses a fixed track index sensor to pre-position the stepper motor. It's a real tricky system to align properly.

SAMS doesn't list the part numbers or the Manufacture Codes for any of the sensors on those old Newtronics Drives. There are three of them involved in the alignment process, and they have to be darn near perfect.

Not much help here, guy... Those drives are no longer manufactured!

Note : On one of my old drives, I just removed the write protect sensor and soldered the connections closed. It worked again, but you could not write protect a disk anymore. Sometimes you have to fudge a bit!

--

>> Dear SMS Mike,

>> I have this old Apple IIE that I would like to infuse with software. I don't
>> have any telecommunications software or hardware for it though. I do have
>> a C-64 with modem and 1541 drive. I hear the disk][Apple drives and the
>> 1541 use a similar encoding scheme based on GCR. Is there a way to r/w apple
>> disks on my C-64 & 1541, especially without hardware modification.
>>
>> OP

OP,

Yikes! Good question but as far as I know, I don't think there is an easy "software-only" answer here. I can think of a few options. I know that there was once an Apple II+ hardware emulator for the C-64. This 'Spartan' unit from Mimic systems allowed R/W to Apple disks, but I am not sure what role (if any!) a 1541 played in this arrangement. That being said, with so many C= software/hardware throughout the years, it may be conceivable that a disk][interface was available for the C-64. I do not know if such an interface exists, but the possibility might be worth investigating.

As far as the actual drives are concerned, the following are known as problems to be overcome for a potential 1541 - disk][reader :

- The 1541 rotates its disks at fixed speeds while its R/W head frequency can be changed in four steps (in series from 250 khz to 300 khz I believe). The disk][apparently has a variable speed drive engine which allows its R/W head to sit at one fixed frequency as it read/writes. As you can see, here we have two opposite paradigms regarding what remains fixed and what becomes variable to get the end-result of being able to read/write "GCR". Therefore, a 1541 could possibly read/write to a disk][format only in those areas on the disk where the paradigms cancel each other (i.e., where 1541 Static Rotation + Variable Freq. = disk][Variable Rotation + Static Freq.) The 1541 frequency resolution as mentioned earlier is in 4 steps, so it cannot match most of the changes incurred by the disk]['s variable rotation. My guess is that a hardware modification to the 1541 would have to be done.

- But that's not all, GCR does not equal GCR! The disk][and 1541 use a different GCR scheme! The 1541 uses a 4-5 encoding and the disk][uses an 8-9 encoding. On top of this, you then have to rewrite the whole file system on the C-64 end to match AppleDOS 3.xx whatever.. Not trivial.

- As an aside, some Apple sources I checked with tend to suggest that the disk][runs at a constant 300 RPM. This is contrary to my admittedly limited experience with Apple hardware, but I was given a canonical source to reference this point : BENEATH APPLE DOS. It is supposed to be much like the classic tome : INSIDE COMMODORE DOS. If you or anyone else wishes to attempt a 1541 to disk][project, these are the books to get acquainted with.
--

>> Dear SMS Mike,

>> I hear the SuperCPU 128 will require a daughterboard or clip lead to the
>> VDC chip. I don't want to open up my 128. What gives?
>>
>> PO

PO,

I have not heard this, but it would not surprise me. The VDC is poorly interfaced to main memory (for that matter, so is the Z80) in the C128. I can imagine a few reasons why additional hardware links above and beyond a simple cartridge plug-in might be required for Super-CPU & VDC operation. I won't speculate much on this forum, since I lack proper information about the SuperCPU paradigm and the VDC itself. Although, now that I think about it, I am reminded of the VDC article in disC=overy issue 1 (by S. Judd). I believe the goal of that article was to explore enough of the VDC internals to "time it out" so to speak. Now that we have gotten off-topic here :), I have never run into a way to make a stable raster on the VDC. Hmmmm, according to my sources, the VDC runs at the 16 Mhz dotclock while everything else on an NTSC C128 system runs at 14,318,181/14 Hz (on NTSC). I can't see a stable raster 100% under those conditions. Anyways, as per your original question, we will have to wait until CMD releases the SuperCPU 128 to find out -what- is required and -why-.
--

>> Dear SMS Mike,
>>
>> I bought a VIC-20 at a thrift shop the other day, and OF COURSE, the
>> thing came with no cabling or power supply. I managed to hack on a
>> power supply and had no problems using standard C= serial cables for
>> drive access, but how do I get a video/audio signal out of it? By
>> the way, the VIC-20 will let me blindly type in and load a directory,
>> so I'm pretty certain it works.
>>
>> FN

FN,

Here is the quickest way I know how to do it. Just remember that this is a VIC-20 Video Cable for use with TV/VCR/Monitors with VIDEO IN/AUDIO IN ports. Now looking at the back of your VIC-20, you will see the 5 DIN video port, as follows :

```
                               /--\_/--\
                              /         \
                             /  3     1  \
                            [  .       .  Ü
                            [    5   4    Ü
                             \  .  2  .  /
                              \    .    /
                               \_______/
```

Pin 1 : +5-6 V, max. 10 mAmps (WARNING : DO NOT CONNECT THIS PIN!)
Pin 2 : Ground
Pin 3 : Audio Out
Pin 4 : Video Low (Not Connected)
Pin 5 : Video High

You'll need a 5-pin DIN connector, two RCA jacks, and RCA cables.

Take the 5-pin DIN connector (remember that the pins will be reversed when you view the connector as it faces you) and connect the pins, as follows :

Pin 3 (Audio Out) to an RCA jack (label this one as audio out)
Pin 5 (Video Out) to a second RCA jack (label this one as video out)
Pin 2 (Ground) to the ground sleeve/prong of -both- RCA jacks.

Using RCA-style cables, connect these Audio/Video-Out jacks to their respective Audio/Video-In counterparts on a VCR or Monitor/TV. You should now be able to watch your VIC-20 in action. If everything is connected properly, you will see the powerup message with a CYAN border and WHITE background (NTSC VIC-20).

Your video difficulties should now be resolved. You also mentioned a problem in securing power supplies for the VIC-20. Although you resolved this issue, the traditional steps for those who might not be familiar with the VIC-20 is to swing your VIC to where the On/Off switch is located and find the power socket next to the switch. (It will be the socket closer to the big open port (for cartridges) at the back of your computer).

If you look at your VIC-20's power socket and notice that there are more than 2 prongs :) on it, you are in luck. A regular C-64 power supply will work with your VIC-20.

If you look at your VIC-20's power socket and it does have two prongs facing out, then you have to search! Find a power supply with at least 9 Volts AC delivered at 1 Ampere (with two prongs, each delivering power at 180 degrees out of phase from each other) and hook it up. If this is not an option, you can pull 9 Volts AC @ 1 Ampere from a C64 or flat C128 power supply. For example, the online 'zine C= Hacking, issue #6, has an article on building power supplies for the C-64. With two extra wires drawing off the 9 VAC and the proper two-prong nylon connector, the power supply described in the article becomes an excellent "old-style VIC-20" supply.

```
:::::::::::d:i:s:C=:o:v:e:r:y:::::::::::::::::i:s:s:u:e::2::::::::::::::::::::
$2bad:::::::::::::::::::::::::::E:R:R:A:T:A:::::::::::::::::::::::::::::::::::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

ERRATA : ISSUE 1

>> /S04 - "A complete dissection of Gfx-Zone"
>> $d000   by "XmikeX"

- In TBB's music player code, "lda #$7f : sta $dc0d" was described as enabling timer interrupts when in reality it disables timer interrupts.

- The author may be reached through the Editor-in-Chief of disC=overy.

>> /S05 - "A Beginner's Guide to the JCH Editor V2.53, NewPlayer V14.G0"
>> $d400   by Sean M. Pappalardo

- SHIFT and SPACE inserts the sustain marker in the blocks (+++), * NOT * SHIFT and RETURN, as written in the article.

- The author's internet address was omitted! Mr. Pappalardo is available for questions and comments at : pegasus@planet.earthcom.net

>> /S07 - "Some preliminary data on VDC timing"
>> $d600   by Stephen L. Judd

- At least one of the uuencoded files included in the original article was found to be corrupt. Both have been verified and included below. Note: 'vdc-explorer' is the basic portion, which loads (from drive 8) the ML object file (vdc-explorer.o).

```
begin 644 vdc-explorer
M 1PP' H F2 BFY/6Q,,@Q5A03$]215(@0ED@TU1%4$A%3B#,+B#*541$(CK^
M"R Q $P<% "+(,(HT2@B,3,P,"(I*;(Q,C @IR T, !C'!X _A$B5D1#+458
M4$Q/4D52+D\B '$<* !.LC$R,#I#LC  DQPR /XF.D:R,3J7(#4S,C8U+,(H
M-3,R-C4I(*\@,C,Y *(</ "/($9!4U0Z1CTR -X<1@"&(%0Q*$XI+%0R*$XI
M+%0S*$XI+%0T*$XI+%0U*$XI+%0V*$XI+%0W*$XI+%0Y*$XI+$%6*#@I #@=
M4 "9(I,B.E1%4U2RT2@B,31#.2(I.E(QLE1%4U2J,3I2,K)2,:HR.D%2LE(R
MJC(Z4C.R05*J,CI2-+)2,ZHR.D93LE(TJC(Z34&R1E.J,CI&1K)-0:HR 'P=
M6@!!5B@Q*;(P.D%6*#(ILC Z058H,RFR,#I!5B@T*;(P.D%6*#4ILC Z058H
M-BFR,#I!5B@W*;(P.D%6*#@ILC  BAUD ($@2;(Q(*0@3@":'6X GB#1*"(Q
M,S P(BD PQUX %0Q*$DILL(H5$535"DZ058H,2FR058H,2FJ5#$H22FK,3"M
M1@#Q'8( 5#(H22FRPBA2,2FJ,C4VK,(H4C&J,2DZ058H,BFR058H,BFJ5#(H
M22D 'QZ, %0S*$DILL(H4C(IJC(U-JS"*%(RJC$I.D%6*#,ILD%6*#,IJE0S
M*$DI $T>E@!4-"A)*;+"*$%2*:HR-3:LPBA!4JHQ*3I!5B@T*;)!5B@T*:I4
M-"A)*0![’J  5#4H22FRPBA2,RFJ,C4VK,(H4C.J,2DZ058H-2FR058H-2FJ
M5#4H22D J1ZJ %0V*$DILL(H4C0IJC(U-JS"*%(TJC$I.D%6*#8ILD%6*#8I
MJE0V*$DI -<>M !4-RA)*;+"*$93*:HR-3:LPBA&4ZHQ*3I!5B@W*;)!5B@W
M*:I4-RA)*0#P'KX 5#BR,C4VK,(H34$IJL(H34&J,2D 'A_( %0Y*$DILL(H
M1D8IJC(U-JS"*$9&JC$I.D%6*#@ILD%6*#@IJE0Y*$DI %T?T@"9("(3$1$1
M$2)).U0Q*$DI.U0R*$DI.U0S*$DI.U0T*$DI.U0U*$DI.U0V*$DI.U0W*$DI
M.U0Y*$DI &,?W "" )D?Y@!/2+)!5B@Q*:U..IDBDP5415-4,3H@3U9%4DA%
M040](D]((B!%6%!%0U1%1#TB,ZU& +4?\ "9(AQ415-4,CH1G9V=G9V=HZ.C
MHZ,B.P#,'_H 4U*R*$%6*#(IJT%6*#$I*:U. .<?! &9+")324Y'3$4@4D5'
M25-415(](E-2  ,@#@&9+")%6%!%0U1%1"!604Q513TB,3"M1@ -(!@!C2 V
M,#  *2 B 9DBGU1%4U0S.A&=G9V=G9VCHZ.CHR([ %<@+ &9+")324Y'3$4@
M4D5'25-415(@5%=)0T4](BA!5B@S*:M!5B@Q*2FM3@!S(#8!F2PB15A014-4
M140@5D%,544](E-2K#( ?2!  8T@-C P )D@2@&9(IQ415-4-#H1G9V=G9V=
MHZ.CHZ,B.P#"(%0!5%*R*$%6*#0IJT%6*#$I*:U..IDL(E1/5$%,(%1)344]
M(E12 /(@7@%44[)44JLH,S>L-:LQ*:U&.IDL(DU)3E53($Q/3U @3U9%4DA%
M040](E13 !0A: &9+")$258@,S<@0DE4+4Q/3U!3/2)44ZLS-S"M1@!#(7(!
MF2PB/3X@0DE4+4Q/3U @4D50151)5$E/3E,](BA44ZLS-S"M1BFL1JTW $TA
M? &-(#8P, !I(88!F2(>5$535#4Z$9V=G9V=G:.CHZ.C(CL DB&0 9DL(E17
M3R!214%$4R!214<@,3@](BA!5B@U*:M!5B@Q*2FM3@"R(9H!F2PB15A014-4
M140@5D%,544](E-2K#*J.*U& +PAI &-(#8P, #8(:X!F2(?5$535#8Z$9V=
MG9V=G:.CHZ.C(CL !R*X 4TQLBA!5B@V*:M!5B@Q*2FM3CJ9+")/3D4@345-
M3U)9($9%5$-(/2)-,0 [(L(!15:R4U*L,ZHQ,JU&.IDL(D584$5#5$5$(%9!
M3%5%/2)%5B(@1$E&1CTB33&K158 =B+, 9DL(E!23T(N($Y532!215!3(%1/
M(%=!250@1D]2(%9$0R!&151#2#TB*$TQJT56*:TH-ZU&*0" (M8!C2 V,#
MG"+@ 9DBGE1%4U0W.A&=G9V=G9VCHZ.CHR([ ,LBZ@%54K(H058H-RFK058H
M,2DIK4XZF2PB3$]/4"!-14U/4D4@1DE$,3#TB55( _2+T 553LE52JS.M1JLR
M-3:L*%-2JCFM1BDZF2PB15A44D$@5D1#(%=!2513/2)54P O(_X!F2PB059%
M4D%'12!"250M3$]/4"!215!%050]4"(U-JPWK48I #DC" *-
M(#8P, !5(Q("F2*!5$535#@Z$9V=G9V=G:.CHZ.C(CL A",< E92LBA!5B@X
M*:M!5B@Q*2FM3CJ9+")43U1!3"!"3$]#2R!&24Q,/2)64@#$(R8"F2PB/3X@
M05!04D]8(%9$0R!424U%(%9/4B R-38@0EE412!"3$]#2R!&24Q,/2)64JLR
M-*U&JS.L4U( RB,P CH X",Z HT@-C P P.HL@0[(Q(*<@-3DP /,C1 +^)3I#
MLC$Z1K(R.HDX,  ")$X"F2*?0EE%(2(Z@  9)%@"H2!!)#J+($$DLB(B(*<@
,-C P !\D8@*.

end

begin 644 vdc-explorer.o
M !-XJ?^-!-V-!=VI&: (C0[=ZNKJZNJ,#MVI_SCM!-V-R12B$JD9C0[=C@#6
M+ #6$/N,#MTXJ?_M!-V-RA2I_^T%W8W+%*D9C0[=C@#6+ #6$/N. -8L -80
M^XP.W3BI_^T$W8W,%*G_[07=C<T4HB2I&8T.W8X UBP UA#[RA#UC [=.*G_
M[03=C<X4J?_M!=V-SQ2B$JD9C0[=C@#6+ #6$/NM =R. -8L -80^ZT!W(P.
MW3BI_^T$W8W0%*G_[07=C=$4J1F-#MV. -8L -80^XT!UNB. -8L -80^XT!
MUJ(?C@#6+ #6$/N,#MTXJ?_M!-V-TA2I_^T%W8W3%*(2J0". -8L -80^XT!
MUHX UBP UA#[C0'6J1F@ *(?C0[=J6:. -8L -80^XT!ULC0\J (C [=.*G_
M[03=C=04J?_M!=V-U12B$JD C@#6+ #6$/N- =;HC@#6+ #6$/N- =:I&:(?
MC0[=J6:. -8L -80^XT!UJ(8J0". -8L -80^XT!UJ(>J?^. -8L -80^XT!
MUJ(2C@#6+ #6$/N,#MVM =:-UA3HC@#6+ #6$/NM =:-UQ0XJ?_M!-V-V!2I
:_^T%W8W9%%A@                      #6

end
```

>> /S09 - "Software analysis and reconstructive therapy, a historical view
   $dd00   on 'cracking'"
            by Pontus Berg

  - The author's internet address was omitted! Mr. Berg is available
    for questions and comments at : PBG@hk.mobitel.telia.se

--
Note : None of these errors are present in the 'WordPerfect for Windows
       5.2' version of disC=overy, issue 1.

```
::::::::::d:i:s:C=:o:v:e:r:y:::::::::::::::::i:s:s:u:e::2::::::::::::::::::::
\END:::::::::::::::::::::::::October 1, 1996:::::::::::::::::::::::::::::::::
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```