Sorry that this is later than I had hoped to get it out, but here it is -- Expect another one soon in a month or two depending on submissions... Praise, Comments, (It sucks, I loved it, etc) are welcome -> duck@pembvax1.pembroke.edu. Many thanks to Craig Bruce for his article on line drawing / dot plotting with the 80 column screen on the C=128.

==========

Hacking / This Magazine
by Craig Taylor
duck@pembvax1.pembroke.edu

Def:
Hacker - Noun - A talented amateur user of computers.
Source - Webster's New World Dictionary

Correction:
Hacker - Noun - A talented user of computers.

There, now that we got that out of the way, let's see how some people interpret the word hacker. In the 1980's newspapers, magazines, movies - everywhere ya looked people were using the term "Hacker" to denote a person who maliciously tried to destroy / bore ill intent towards another computer system. This was the result of the misunderstanding of the actual definition of "Hacker" (under my correction above).

This magazine will not tell people how to "phreak", how to hack voice mailboxes and other illegal activities. However, it will attempt to reveal some of the "mystique" behind some of the new techniques and abilities found in the Commodore 64 and Commodore 128 that are just now being revealed.
In the event that an article is submitted and there is a question about it's ability to be applied towards illegal activites, the article will be carried with a warning that the intent is not towards that activity. Hopefully, these will never come along. :-)

The Commodore 64 came out in late 1982 (I believe) and was known to only support 16 colors, 320 x 200 resolution graphics of 2 colors, 160x200 resolution graphics of 4 colors. Since then people have pushed the Commodore 64 to its limits with apparent resolution of 320 x 200 with a resolution of 4 colors and even higher... more than 8 sprites on the screen... fast high-quality digitized sounds....

The Commodore 128 came out as an "upgrade" from the Commodore 64 and with it's unique memory management scheme and the Z80a chip still on there people are still finding out unique and interesting ways to explore the C=128. One of the most interesting has been that of the seperate video display chip which makes it possible to dispaly 640x200 resolution graphics quickly and easily.

ATTENTION

This magazine is going to be a sourcebook of many people - If you know anything about something, please feel free to submit it. Just mail the article to the following :

duck@pembvax1.pembroke.edu
and a subject of "ARTICLE - " and then the article name.

The source code for all programs mentioned within articles will be provided as well as any executables uuencoded sent out seprately. [Ed. Note - In this issue, the source is not sent seperately due to only one article with files]

In addition, the magazine will go out when there are enough articles collected. Also, I'm currently in college - so - it will also be dependant on few tests etc being around the release period.

In this issue:

Hacking - Definition Of	duck@pembvax1.pembroke.edu
Learning ML - Part 1	duck@pembvax1.pembroke.edu
6502 Known/Unknown Opcodes	compilation of several
Dot Plotting & Bitmapping	csbruce@ccnga.uwaterloo.ca
the 8563 Screen.	

** All articles and files (C) 1992 by their respective authors.

==========

Beginning ML - Part One
(C) 1992 by Craig Taylor

The best way to learn machine language is to actually code routines that you don't think will work, hope that they work, and then figure out why they don't work. (If they do work, you try to figure out why you didn't think they'd work). Ie: THE BEST WAY TO LEARN ANY PROGRAMMING LANGUAGE IS TO PROGRAM IN THAT LANGUAGE. And Machine Language is a programming language.

Now, let's get a few terms and definitions out of the way:

Machine Language - Instructions that the computer understands at a primitive level and executes accordingly.

Assembly Language - Instructions more understandable to humans than pure Machine Language that makes life easier.

	Assembly:	Machine:
Example:	lda #$00	$A9 $00

Huh? you might be saying at the moment. Turns out that LDA stands for, or is a mnemonic (computer people always come up with these big long words -- you'll see mnemonic's often when dealing with machine language) for the following:

```
"LOAD register A with the following value"
 ^  ^          ^
```

Cool 'eh? Yeah, but there's somebody grumbling now about why not make it LOADA etc.. Hey, that's life. (GRIN).

Oh, more definitions:

Register - A location inside the CPU that can be manipulated directly without having to access memory.

The "A" register is often called the accumalator which indicates its function: all math and logical manipulations are done to the "A" register (from hereon out it will be referred to as .A).

There are two other registers inside the 6502 processor, specifically .X and .Y. These registers help act as counters and indexes into memory (sorta like mem[x] in pascal but not quite...).

Now, let's add 3 and 5 and leave the result in the accumalator (.A).

```
               lda     #3              ; Here .A = 3 (anything w/ a ; is a
 ; comment and will be ignored by the assembler...
               clc             ; hu? - This clears the carry. The 6502
; does addition *everytime* with the carry ... so if we clear it it won't
; affect the result.
               adc     #5              ; Now, .A = .A + 5
```

and we're done. If the CLC confused you then consider that if you're adding a column of #'s:

```
      12 <--\__The 2 + 9 = 11, but we put the 1 down and set the carry to 1.
  +   89 <---/
      --
     101
```

Then we say 1 + 8 + carry , which in this case happens to = 1 and we get 10 and again we set the carry and write down 0. Then it's just the carry and we

write that down. If we didn't clear the carry we may have ended up with the value of 9 instead 8 if the carry had happened to be set.

  Aaagh, Math - Let's continue - The CLC mnemonic stands for "CLEAR CARRY" and the ADC stands for "ADD with CARRY". On many processors there is a ADD (without a carry) but unfortunately the 6502 processor inside the C=64 doesn't have it.

 So we've got:

```
    load reg A with the value 5          lda #5
    clear the carry                      clc
    add reg a and value 3                adc #3
```

 In Basic it's just:

```
    A = 5+3
```

  One statement... In Machine Language you've got to break everything down into smaller and smaller steps and quite often the ML listing will be far longer than the BASIC or PASCAL or C equivlent.

 Definitions:

   Assembler - Program takes source code in basic form or from a file and writes to memory or a file the resulting executable. Allows higher flexibility than a monitor (see below) due to use of labels etc and not having to keep track of each address within the program.

  Monitor - A program, resident in memory, invoked by a sys call from basic or by hitting the restore key that will let you disassemble, assemble and examine areas of memory and execute programs directly from the monitor. Useful for debugging programs and for writing short programs.

  Let's enter the following into a monitor (if you don't have one then contact duck@pembvax1.pembroke.edu and I'll send ya one):

```
128:                    c64:
>a 1300 lda #$93        >a c000 lda #$93
>a 1302 jsr $ffd2       >a c003 jsr $ffd2
>a 1305 rts             >a c005 rts
(exit monitor)          (exit monitor)
bank15:sys4864          sys 49152
```

  Wow! It cleared the screen. Neat 'eh? But see how much ya gotta break problems down? The first statement loads in $93 hex into the accumalator ($93 hex just happens to equal 147 which is also the Commodorscii code for clear screen. For a whole list just look in the back of the book that came with the computer). Then we jump to a system routine which Commodore so graciously supplied us with that prints the value of the character in .A to the screen. (jsr $ffd2) then we do a RTS (ReTurn from Subroutine) so that we will go back to basic and the Ready prompt when we are finished with the sys call.
  You C= 128 people may be wondering why you had to do a bank 15 and assemble the stuff at a different memory location. Turns out that the C128 memory map of where routines etc are at is much more complex than the C=64 and thus you have to tell basic which bank you wish to have all sys, peek, and poke calls to take place in. Also, $c000 as used on the C=64 is not an area that is free to use on the C128 in this manner.

  Assignment: Take a look @ the different commands as listed in 6502 Opcodes and try to understand what they do. Experiment with the jsr $ffd2 routine by using different values etc.

  Next Time: Printing out strings, and understanding 'Indexing'.

==========

6502 Opcodes and Quasi-Opcodes.

The following table lists all of the available opcodes on the 65xx line of micro-processors (such as the 6510 on the C=64 and the 8502 on the C=128)

Std	Mnemonic	Hex Value	Description	Addressing Mode	Bytes/Time
*	BRK	$00	Stack <- PC, PC <- ($fffe)	(Immediate)	1/7
*	ORA	$01	A <- (A) V M	(Ind,X)	6/2
	JAM	$02	[locks up machine]	(Implied)	1/-
	SLO	$03	M <- (M >> 1) + A + C	(Ind,X)	2/8
	NOP	$04	[no operation]	(Z-Page)	2/3
*	ORA	$05	A <- (A) V M	(Z-Page)	2/3
*	ASL	$06	C <- A7, A <- (A) << 1	(Z-Page)	2/5
	SLO	$07	M <- (M >> 1) + A + C	(Z-Page)	2/5
*	PHP	$08	Stack <- (P)	(Implied)	1/3
*	ORA	$09	A <- (A) V M	(Immediate)	2/2
*	ASL	$0A	C <- A7, A <- (A) << 1	(Accumalator)	1/2
	ANC	$0B	A <- A /\ M, C=~A7	(Immediate)	1/2
	NOP	$0C	[no operation]	(Absolute)	3/4
*	ORA	$0D	A <- (A) V M	(Absolute)	3/4
*	ASL	$0E	C <- A7, A <- (A) << 1	(Absolute)	3/6
	SLO	$0F	M <- (M >> 1) + A + C	(Absolute)	3/6
*	BPL	$10	if N=0, PC = PC + offset	(Relative)	2/2'2
*	ORA	$11	A <- (A) V M	((Ind),Y)	2/5'1
	JAM	$12	[locks up machine]	(Implied)	1/-
	SLO	$13	M <- (M >. 1) + A + C	((Ind),Y)	2/8'5
	NOP	$14	[no operation]	(Z-Page,X)	2/4
*	ORA	$15	A <- (A) V M	(Z-Page,X)	2/4
*	ASL	$16	C <- A7, A <- (A) << 1	(Z-Page,X)	2/6
	SLO	$17	M <- (M >> 1) + A + C	(Z-Page,X)	2/6
*	CLC	$18	C <- 0	(Implied)	1/2
*	ORA	$19	A <- (A) V M	(Absolute,Y)	3/4'1
	NOP	$1A	[no operation]	(Implied)	1/2
	SLO	$1B	M <- (M >> 1) + A + C	(Absolute,Y)	3/7
	NOP	$1C	[no operation]	(Absolute,X)	2/4'1
*	ORA	$1D	A <- (A) V M	(Absolute,X)	3/4'1
*	ASL	$1E	C <- A7, A <- (A) << 1	(Absolute,X)	3/7
	SLO	$1F	M <- (M >> 1) + A + C	(Absolute,X)	3/7
*	JSR	$20	Stack <- PC, PC <- Address	(Absolute)	3/6
*	AND	$21	A <- (A) /\ M	(Ind,X)	2/6
	JAM	$22	[locks up machine]	(Implied)	1/-
	RLA	$23	M <- (M << 1) /\ (A)	(Ind,X)	2/8
*	BIT	$24	Z <- ~(A /\ M) N<-M7 V<-M6	(Z-Page)	2/3
*	AND	$25	A <- (A) /\ M	(Z-Page)	2/3
*	ROL	$26	C <- A7 & A <- A << 1 + C	(Z-Page)	2/5
	RLA	$27	M <- (M << 1) /\ (A)	(Z-Page)	2/5'5
*	PLP	$28	A <- (Stack)	(Implied)	1/4
*	AND	$29	A <- (A) /\ M	(Immediate)	2/2
*	ROL	$2A	C <- A7 & A <- A << 1 + C	(Accumalator)	1/2
	ANC	$2B	A <- A /\ M, C <- ~A7	(Immediate9	1/2
*	BIT	$2C	Z <- ~(A /\ M) N<-M7 V<-M6	(Absolute)	3/4
*	AND	$2D	A <- (A) /\ M	(Absolute)	3/4
*	ROL	$2E	C <- A7 & A <- A << 1 + C	(Absolute)	3/6
	RLA	$2F	M <- (M << 1) /\ (A)	(Absolute)	3/6'5
*	BMI	$30	if N=1, PC = PC + offset	(Relative)	2/2'2
*	AND	$31	A <- (A) /\ M	((Ind),Y)	2/5'1
	JAM	$32	[locks up machine]	(Implied)	1/-
	RLA	$33	M <- (M << 1) /\ (A)	((Ind),Y)	2/8'5
	NOP	$34	[no operation]	(Z-Page,X)	2/4
*	AND	$35	A <- (A) /\ M	(Z-Page,X)	2/4
*	ROL	$36	C <- A7 & A <- A << 1 + C	(Z-Page,X)	2/6
	RLA	$37	M <- (M << 1) /\ (A)	(Z-Page,X)	2/6'5
*	SEC	$38	C <- 1	(Implied)	1/2
*	AND	$39	A <- (A) /\ M	(Absolute,Y)	3/4'1
	NOP	$3A	[no operation]	(Implied)	1/2
	RLA	$3B	M <- (M << 1) /\ (A)	(Absolute,Y)	3/7'5

	NOP	$3C	[no operation]	(Absolute,X)	3/4'1
*	AND	$3D	A <- (A) /\ M	(Absolute,X)	3/4'1
*	ROL	$3E	C <- A7 & A <- A << 1 + C	(Absolute,X)	3/7
	RLA	$3F	M <- (M << 1) /\ (A)	(Absolute,X)	3/7'5
*	RTI	$40	P <- (Stack), PC <-(Stack)	(Implied)	1/6
*	EOR	$41	A <- (A) \-/ M	(Ind,X)	2/6
	JAM	$42	[locks up machine]	(Implied)	1/-
	SRE	$43	M <- (M >> 1) \-/ A	(Ind,X)	2/8
	NOP	$44	[no operation]	(Z-Page)	2/3
*	EOR	$45	A <- (A) \-/ M	(Z-Page)	2/3
*	LSR	$46	C <- A0, A <- (A) >> 1	(Absolute,X)	3/7
	SRE	$47	M <- (M >> 1) \-/ A	(Z-Page)	2/5
*	PHA	$48	Stack <- (A)	(Implied)	1/3
*	EOR	$49	A <- (A) \-/ M	(Immediate)	2/2
*	LSR	$4A	C <- A0, A <- (A) >> 1	(Accumalator)	1/2
	ASR	$4B	A <- [(A /\ M) >> 1]	(Immediate)	1/2
*	JMP	$4C	PC <- Address	(Absolute)	3/3
*	EOR	$4D	A <- (A) \-/ M	(Absolute)	3/4
*	LSR	$4E	C <- A0, A <- (A) >> 1	(Absolute)	3/6
	SRE	$4F	M <- (M >> 1) \-/ A	(Absolute)	3/6
*	BVC	$50	if V=0, PC = PC + offset	(Relative)	2/2'2
*	EOR	$51	A <- (A) \-/ M	((Ind),Y)	2/5'1
	JAM	$52	[locks up machine]	(Implied)	1/-
	SRE	$53	M <- (M >> 1) \-/ A	((Ind),Y)	2/8
	NOP	$54	[no operation]	(Z-Page,X)	2/4
*	EOR	$55	A <- (A) \-/ M	(Z-Page,X)	2/4
*	LSR	$56	C <- A0, A <- (A) >> 1	(Z-Page,X)	2/6
	SRE	$57	M <- (M >> 1) \-/ A	(Z-Page,X)	2/6
*	CLI	$58	I <- 0	(Implied)	1/2
*	EOR	$59	A <- (A) \-/ M	(Absolute,Y)	3/4'1
	NOP	$5A	[no operation]	(Implied)	1/2
	SRE	$5B	M <- (M >> 1) \-/ A	(Absolute,Y)	3/7
	NOP	$5C	[no operation]	(Absolute,X)	3/4'1
*	EOR	$5D	A <- (A) \-/ M	(Absolute,X)	3/4'1
	SRE	$5F	M <- (M >> 1) \-/ A	(Absolute,X)	3/7
*	RTS	$60	PC <- (Stack)	(Implied)	1/6
*	ADC	$61	A <- (A) + M + C	(Ind,X)	2/6
	JAM	$62	[locks up machine]	(Implied)	1/-
	RRA	$63	M <- (M >> 1) + (A) + C	(Ind,X)	2/8'5
	NOP	$64	[no operation]	(Z-Page)	2/3
*	ADC	$65	A <- (A) + M + C	(Z-Page)	2/3
*	ROR	$66	C<-A0 & A<- (A7=C + A>>1)	(Z-Page)	2/5
	RRA	$67	M <- (M >> 1) + (A) + C	(Z-Page)	2/5'5
*	PLA	$68	A <- (Stack)	(Implied)	1/4
*	ADC	$69	A <- (A) + M + C	(Immediate)	2/2
*	ROR	$6A	C<-A0 & A<- (A7=C + A>>1)	(Accumalator)	1/2
	ARR	$6B	A <- [(A /\ M) >> 1]	(Immediate)	1/2'5
*	JMP	$6C	PC <- Address	(Indirect)	3/5
*	ADC	$6D	A <- (A) + M + C	(Absolute)	3/4
*	ROR	$6E	C<-A0 & A<- (A7=C + A>>1)	(Absolute)	3/6
	RRA	$6F	M <- (M >> 1) + (A) + C	(Absolute)	3/6'5
*	BVS	$70	if V=1, PC = PC + offset	(Relative)	2/2'2
*	ADC	$71	A <- (A) + M + C	((Ind),Y)	2/5'1
	JAM	$72	[locks up machine]	(Implied)	1/-
	RRA	$73	M <- (M >> 1) + (A) + C	((Ind),Y)	2/8'5
	NOP	$74	[no operation]	(Z-Page,X)	2/4
*	ADC	$75	A <- (A) + M + C	(Z-Page,X)	2/4
*	ROR	$76	C<-A0 & A<- (A7=C + A>>1)	(Z-Page,X)	2/6
	RRA	$77	M <- (M >> 1) + (A) + C	(Z-Page,X)	2/6'5
*	SEI	$78	I <- 1	(Implied)	1/2
*	ADC	$79	A <- (A) + M + C	(Absolute,Y)	3/4'1
	NOP	$7A	[no operation]	(Implied)	1/2
	RRA	$7B	M <- (M >> 1) + (A) + C	(Absolute,Y)	3/7'5
	NOP	$7C	[no operation]	(Absolute,X)	3/4'1
*	ADC	$7D	A <- (A) + M + C	(Absolute,X)	3/4'1

*	ROR	$7E	C<-A0 & A<- (A7=C + A>>1)	(Absolute,X)	3/7
	RRA	$7F	M <- (M >> 1) + (A) + C	(Absolute,X)	3/7'5
	NOP	$80	[no operation]	(Immediate)	2/2
*	STA	$81	M <- (A)	(Ind,X)	2/6
	NOP	$82	[no operation]	(Immediate)	2/2
	SAX	$83	M <- (A) /\ (X)	(Ind,X)	2/6
*	STY	$84	M <- (Y)	(Z-Page)	2/3
*	STA	$85	M <- (A)	(Z-Page)	2/3
*	STX	$86	M <- (X)	(Z-Page)	2/3
	SAX	$87	M <- (A) /\ (X)	(Z-Page)	2/3
*	DEY	$88	Y <- (Y) - 1	(Implied)	1/2
	NOP	$89	[no operation]	(Immediate)	2/2
*	TXA	$8A	A <- (X)	(Implied)	1/2
	ANE	$8B	M <-[(A)\/$EE] /\ (X)/\(M)	(Immediate)	2/2^4
*	STY	$8C	M <- (Y)	(Absolute)	3/4
*	STA	$8D	M <- (A)	(Absolute)	3/4
*	STX	$8E	M <- (X)	(Absolute)	3/4
	SAX	$8F	M <- (A) /\ (X)	(Absolute)	3/4
*	BCC	$90	if C=0, PC = PC + offset	(Relative)	2/2'2
*	STA	$91	M <- (A)	((Ind),Y)	2/6
	JAM	$92	[locks up machine]	(Implied)	1/-
	SHA	$93	M <- (A) /\ (X) /\ (PCH+1)	(Absolute,X)	3/6'3
*	STY	$94	M <- (Y)	(Z-Page,X)	2/4
*	STA	$95	M <- (A)	(Z-Page,X)	2/4
	SAX	$97	M <- (A) /\ (X)	(Z-Page,Y)	2/4
*	STX	$96	M <- (X)	(Z-Page,Y)	2/4
*	TYA	$98	A <- (Y)	(Implied)	1/2
*	STA	$99	M <- (A)	(Absolute,Y)	3/5
*	TXS	$9A	S <- (X)	(Implied)	1/2
	SHS	$9B	X <- (A) /\ (X), S <- (X) M <- (X) /\ (PCH+1)	(Absolute,Y)	3/5
	SHY	$9C	M <- (Y) /\ (PCH+1)	(Absolute,Y)	3/5'3
*	STA	$9D	M <- (A)	(Absolute,X)	3/5
	SHX	$9E	M <- (X) /\ (PCH+1)	(Absolute,X)	3/5'3
	SHA	$9F	M <- (A) /\ (X) /\ (PCH+1)	(Absolute,Y)	3/5'3
*	LDY	$A0	Y <- M	(Immediate)	2/2
*	LDA	$A1	A <- M	(Ind,X)	2/6
*	LDX	$A2	X <- M	(Immediate)	2/2
	LAX	$A3	A <- M, X <- M	(Ind,X)	2/6
*	LDY	$A4	Y <- M	(Z-Page)	2/3
*	LDA	$A5	A <- M	(Z-Page)	2/3
*	LDX	$A6	X <- M	(Z-Page)	2/3
	LAX	$A7	A <- M, X <- M	(Z-Page)	2/3
*	TAY	$A8	Y <- (A)	(Implied)	1/2
*	LDA	$A9	A <- M	(Immediate)	2/2
*	TAX	$AA	X <- (A)	(Implied)	1/2
	LXA	$AB	X04 <- (X04) /\ M04 A04 <- (A04) /\ M04	(Immediate)	1/2
*	LDY	$AC	Y <- M	(Absolute)	3/4
*	LDA	$AD	A <- M	(Absolute)	3/4
*	LDX	$AE	X <- M	(Absolute)	3/4
	LAX	$AF	A <- M, X <- M	(Absolute)	3/4
*	BCS	$B0	if C=1, PC = PC + offset	(Relative)	2/2'2
*	LDA	$B1	A <- M	((Ind),Y)	2/5'1
	JAM	$B2	[locks up machine]	(Implied)	1/-
	LAX	$B3	A <- M, X <- M	((Ind),Y)	2/5'1
*	LDY	$B4	Y <- M	(Z-Page,X)	2/4
*	LDA	$B5	A <- M	(Z-Page,X)	2/4
*	LDX	$B6	X <- M	(Z-Page,Y)	2/4
	LAX	$B7	A <- M, X <- M	(Z-Page,Y)	2/4
*	CLV	$B8	V <- 0	(Implied)	1/2
*	LDA	$B9	A <- M	(Absolute,Y)	3/4'1
*	TSX	$BA	X <- (S)	(Implied)	1/2
	LAE	$BB	X,S,A <- (S /\ M)	(Absolute,Y)	3/4'1
*	LDY	$BC	Y <- M	(Absolute,X)	3/4'1

*	LDA	$BD	A <- M	(Absolute,X)	3/4'1
*	LDX	$BE	X <- M	(Absolute,Y)	3/4'1
	LAX	$BF	A <- M, X <- M	(Absolute,Y)	3/4'1
*	CPY	$C0	(Y - M) -> NZC	(Immediate)	2/2
*	CMP	$C1	(A - M) -> NZC	(Ind,X)	2/6
	NOP	$C2	[no operation]	(Immediate)	2/2
	DCP	$C3	M <- (M)-1, (A-M) -> NZC	(Ind,X)	2/8
*	CPY	$C4	(Y - M) -> NZC	(Z-Page)	2/3
*	CMP	$C5	(A - M) -> NZC	(Z-Page)	2/3
*	DEC	$C6	M <- (M) - 1	(Z-Page)	2/5
	DCP	$C7	M <- (M)-1, (A-M) -> NZC	(Z-Page)	2/5
*	INY	$C8	Y <- (Y) + 1	(Implied)	1/2
*	CMP	$C9	(A - M) -> NZC	(Immediate)	2/2
*	DEX	$CA	X <- (X) - 1	(Implied)	1/2
	SBX	$CB	X <- (X)/\(A) - M	(Immediate)	2/2
*	CPY	$CC	(Y - M) -> NZC	(Absolute)	3/4
*	CMP	$CD	(A - M) -> NZC	(Absolute)	3/4
*	DEC	$CE	M <- (M) - 1	(Absolute)	3/6
	DCP	$CF	M <- (M)-1, (A-M) -> NZC	(Absolute)	3/6
*	BNE	$D0	if Z=0, PC = PC + offset	(Relative)	2/2'2
*	CMP	$D1	(A - M) -> NZC	((Ind),Y)	2/5'1
	JAM	$D2	[locks up machine]	(Implied)	1/-
	DCP	$D3	M <- (M)-1, (A-M) -> NZC	((Ind),Y)	2/8
	NOP	$D4	[no operation]	(Z-Page,X)	2/4
*	CMP	$D5	(A - M) -> NZC	(Z-Page,X)	2/4
*	DEC	$D6	M <- (M) - 1	(Z-Page,X)	2/6
	DCP	$D7	M <- (M)-1, (A-M) -> NZC	(Z-Page,X)	2/6
*	CLD	$D8	D <- 0	(Implied)	1/2
*	CMP	$D9	(A - M) -> NZC	(Absolute,Y)	3/4'1
	NOP	$DA	[no operation]	(Implied)	1/2
	DCP	$DB	M <- (M)-1, (A-M) -> NZC	(Absolute,Y)	3/7
	NOP	$DC	[no operation]	(Absolute,X)	3/4'1
*	CMP	$DD	(A - M) -> NZC	(Absolute,X)	3/4'1
*	DEC	$DE	M <- (M) - 1	(Absolute,X)	3/7
	DCP	$DF	M <- (M)-1, (A-M) -> NZC	(Absolute,X)	3/7
*	CPX	$E0	(X - M) -> NZC	(Immediate)	2/2
*	SBC	$E1	A <- (A) - M - ~C	(Ind,X)	2/6
	NOP	$E2	[no operation]	(Immediate)	2/2
	ISB	$E3	M <- (M) - 1,A <- (A)-M-~C	(Ind,X)	3/8'1
*	CPX	$E4	(X - M) -> NZC	(Z-Page)	2/3
*	SBC	$E5	A <- (A) - M - ~C	(Z-Page)	2/3
*	INC	$E6	M <- (M) + 1	(Z-Page)	2/5
	ISB	$E7	M <- (M) - 1,A <- (A)-M-~C	(Z-Page)	2/5
*	INX	$E8	X <- (X) +1	(Implied)	1/2
*	SBC	$E9	A <- (A) - M - ~C	(Immediate)	2/2
*	NOP	$EA	[no operation]	(Implied)	1/2
	SBC	$EB	A <- (A) - M - ~C	(Immediate)	1/2
*	SBC	$ED	A <- (A) - M - ~C	(Absolute)	3/4
*	CPX	$EC	(X - M) -> NZC	(Absolute)	3/4
*	INC	$EE	M <- (M) + 1	(Absolute)	3/6
	ISB	$EF	M <- (M) - 1,A <- (A)-M-~C	(Absolute)	3/6
*	BEQ	$F0	if Z=1, PC = PC + offset	(Relative)	2/2'2
*	SBC	$F1	A <- (A) - M - ~C	((Ind),Y)	2/5'1
	JAM	$F2	[locks up machine]	(Implied)	1/-
	ISB	$F3	M <- (M) - 1,A <- (A)-M-~C	((Ind),Y)	2/8
	NOP	$F4	[no operation]	(Z-Page,X)	2/4
*	SBC	$F5	A <- (A) - M - ~C	(Z-Page,X)	2/4
*	INC	$F6	M <- (M) + 1	(Z-Page,X)	2/6
	ISB	$F7	M <- (M) - 1,A <- (A)-M-~C	(Z-Page,X)	2/6
*	SED	$F8	D <- 1	(Implied)	1/2
*	SBC	$F9	A <- (A) - M - ~C	(Absolute,Y)	3/4'1
	NOP	$FA	[no operation]	(Implied)	1/2
	ISB	$FB	M <- (M) - 1,A <- (A)-M-~C	(Absolute,Y)	3/7
	NOP	$FC	[no operation]	(Absolute,X)	3/4'1
*	SBC	$FD	A <- (A) - M - ~C	(Absolute,X)	3/4'1

```
*   INC      $FE       M <- (M) + 1               (Absolute,X)     3/7
    ISB      $FF       M <- (M) - 1,A <- (A)-M-~C (Absolute,X)     3/7

'1 - Add one if address crosses a page boundry.
'2 - Add 1 if branch succeeds, or 2 if into another page.
'3 - If page boundry crossed then PCH+1 is just PCH
'4 - Sources disputed on exact operation, or sometimes does not work.
'5 - Full eight bit rotation (with carry)

Sources:
  Programming the 6502, Rodney Zaks, (c) 1983 Sybex
  Paul Ojala, Post to Comp.Sys.Cbm (po87553@cs.tut.fi / albert@cc.tut.fi)
  D John Mckenna, Post to Comp.Sys.Cbm (gudjm@uniwa.uwa.oz.au)

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

==========

Simple Hires Line Drawing Package for the C-128 80-Column Screen

Copyright (c) 1992 Craig Bruce <csbruce@ccnga.uwaterloo.ca>

1. GRAPHICS PACKAGE OVERVIEW

The graphics package this article explains is BLOADed into memory at address $1300 on bank 15 and has three entry points:

$1300 = move the pixel cursor or draw a line: .AX=x, .Y=y, .C=cmd
$1303 = activate graphics mode and clear the screen
$1306 = exit graphics mode and reload the character set

To move the pixel cursor to the start point of a line, load the .AX registers with the X coordinate (0-639), load the .Y register with the Y coordinate (0-199), clear the carry flag, and call $1300. (Make sure that Bank 15 is in context). This can be done in BASIC as follows:

```
SYS 4864, X AND 255, X/256, Y, 0
```

To draw a line from the pixel cursor location to a given point, load the .AX and .Y registers like before, set the carry flag, and call $1300. The pixel cursor will then be set to the end point of the line just drawn, so you do not have to set it again if you are drawing a continuous object (like a square).

```
SYS 4864, X AND 255, X/256, Y, 1
```

The activate and exit routines are called without any parameters and work very simply. You should be sure to call exit before returning to the program editing mode or you will not be able to see what you are typing.

A BASIC demonstration program is also included in the UU section for this package. It starts by putting the pixel cursor at the center of the screen and then picks a random point to draw to, and repeats until you press a key to stop it. For an interesting effect, put a call to $1303 immediately before the call to draw the line.

The point plotting speed is about 4,100 pixels per second and the line drawing speed is a bit slower than this because of all of the calculations that have to be done to draw a line. There are faster pixel plotting and line drawing algorithms than the ones implemented here, but that is material for a future article.

2. INTRODUCTION TO THE VDC

Programming the 8563 Video Display Controller is quite straight forward. You access it a bit indirectly, but it can still be done at relatively high speeds using machine language. The VDC contains 37 control registers and from 16K to 64K of dedicated display memory that is separate from the main processor. The

memory must be accessed through the VDC registers.

The important VDC registers for this exercise are:

REG	BITS	DESC
$12	7-0	VDC RAM address high byte
$13	7-0	VDC RAM address low byte
$18	7	Block copy / block fill mode select
$19	7	Bitmap / Character mode select
$19	6	Color / Monochrome mode select
$1a	7-4	Foreground color
$1a	3-0	Background color
$1e	7-0	Copy / fill repetition count
$1f	7-0	VDC RAM data read / write

You access the VDC chip registers though addresses $D600 and $D601 on bank 15. Location $D600 selects the VDC register to use on write and returns the VDC status on read. The only important status information is bit 7 (value $80) which is the "ready" flag. The following two subroutines read or write the value in .A to VDC register number .X:

```
VdcRead:  stx $d600                VdcWrite: stx $d600
WaitLoop: bit $d600                WaitLoop: bit $d600
          bpl WaitLoop                       bpl WaitLoop
          lda $d601                          sta $d601
          rts                                rts
```

Once the current VDC register is selected at $d600, it remains selected. You may read or write it though $d601 as many times as you like as long as you wait for the VDC to be "ready" between accesses.

In order to access the VDC RAM, you must first put the high and low bytes of the VDC RAM address into registers $12 and $13 (high byte first) and then read or write through register $1f to read or write the data in the VDC RAM. After each access to register $1f, the VDC RAM address is incremented by one. So, if you repeatedly read or write to register $1f you can read or write a chunk of VDC memory very quickly.

3. ENTERING GRAPHICS MODE

Activating the graphics mode of the VDC is very simple - you just have to set bit 7 of VDC register $19 and poof! You should also clear bit 6 of that register to disable the character color mode. This graphics package supports only monochrome graphics since the standard 16K VDC does not have enough space to hold both the bitmap and the 8*8 pixel cell attributes. The 640*200 pixel display takes 128,000 bits or 16,000 bytes. This leaves 384 bytes of VDC RAM that is not needed by the bitmap but it is not large enough to do anything with, so it is wasted.

When you disable the character color mode, the VDC takes its foreground and background color values from register $1a. The foreground color is what color the "1" bits in the bitmap will be displayed in and the background color, the "0" bits.

Now that the bitmap mode is set up, we must clear the VDC memory locations 0 to 15999 (decimal) to clear the bitmap screen. This can be done very quickly using the VDC fill mode. If you poke a value into VDC register number $1e, the VDC will fill its memory from the location currently in the VDC RAM address registers for the number of bytes you just poked into register $1e with the value that you last poked into the VDC RAM data register. (This is assuming that "fill" mode is selected in register $18). If you poke a 0 into the repeat register it means to fill 256 bytes.

So, to clear the bitmap, poke a zero into both of the VDC RAM address

registers since the bitmap starts at location 0. Then poke a value of 0 into VDC RAM data register. This sets the fill value to 0 and pokes the first VDC RAM location. Then, go into a loop and put a zero into the VDC repeat register 63 times. This will fill 63 contiguous chunks of 256 bytes each. We end up filling 16,129 bytes, but that is not a problem since we have 384 "safety" bytes at the end of the bitmap. Internally, the VDC will fill its memory at a rate of about 1 Megabyte per second (if I remember my test results correctly), so clearing the screen is a PDFQ operation.

4. EXITING GRAPHICS MODE

To exit from graphics mode we have to reload the character set from the ROM on bank 14 and we have to go back into character mode and clear the text screen. The kernel provides its own character reload routine so I used that. The only problem with it is that it is a lot slower than it has to be. It takes about 0.45 seconds whereas the same job can be done in about 0.09 seconds. The kernel is so slow because it uses the kernel INDFETCH nonsense.

Then you just set the bitmap mode bit to zero and the character color mode to one. This gets you back to normal character mode. You also have to clear the text screen since it will be filled with garbage from the graphing.

5. POINT PLOTTING

The pixels on the screen accessed by their X and Y coordinates, 0 <= X <= 639, 0 <= Y <= 199. The formula to calculate the byte address in the VDC RAM given the X and Y coordinates is made simple by the mapping of bytes to the pixels on the screen. The bytes of VDC memory go across the screen rather than in 8*8 cells like the VIC screen. Each pixel row is defined by 80 consecutive bytes of VDC RAM.

The formula for the byte address of a pixel is: AD=Y*80+INT(X/8), and the formula for the bit number is simply BI=X AND 7. The bit number can be used as an index into a table of bit values: [$80,$40,$20,$10,$08,$04,$02,$01], such that index 0 contains $80, since the highest bit is the leftmost bit.

Calculating the bit number and looking up the bit value is very easy to do in machine language, but the byte address calculation requires a little more work. First we have to multiply the Y value by 80, using a 16-bit word for storage. This is done by shifting the Y value left twice, adding the original Y value, and shifting left four more times. Then we have to shift the X value right by three using a 16-bit word for storage to get INT(X/8), and we add the two results together and we have the byte address.

To plot the point, we have to peek into the VDC RAM at the byte address to see what is "behind" the pixel we want to plot. Then OR the new bit value on to the "background" value and poke the result back into VDC RAM at the byte address. Unfortunately, since the VDC RAM address register auto-increments after each reference, we will have to set it twice - once for the read and once for the write. That means that the VDC registers have to be accessed six times for each pixel. Fortunately, the VDC operates at its highest speed in monochrome bitmap mode (it has less work to do than in color character mode, so it is able to pay more attention to the CPU).

Effects other than just plotting the point can be achieved by using functions other than OR to put the point on the background. EOR would "flip" the pixel, and AND-NOT (achieved by LDA bitval : EOR #$ff : AND background) would erase the pixel.

6. LINE DRAWING

The line drawing routine that is implemented in the package is given by the following BASIC code (in fact, I programmed it in BASIC first to get it working; of course, the BASIC version is as slow as hell):

```
10 dx=x-lx:dy=y-ly
20 if abs(dx)>abs(dy) then begin r=dx:gy=dy/abs(dx):gx=sgn(dx)
30 bend:else r=dy:gx=dx/abs(dy):gy=sgn(dy)
40 px=lx+0.5:py=ly+0.5
50 fori=1to abs(r): <PLOT PX,PY> :px=px+gx:py=py+gy:next
60 lx=x:ly=y
```

This implements the Basic Incremental Algorithm for raster line drawing. The "lx" and "ly" are the position of the pixel cursor and "x" and "y" are the coordinates to draw the line to. The "dx" and "dy" are the differences in the X and Y directions. The idea is that we will increment the pixel cursor by a constant of 1 in one direction and by a fraction 0.0 <= g <= 1.0 in the other direction. This fraction is actually the slope of the line.

Lines 20 and 30 figure out the increments for the X and Y directions ("gx" and "gy"). These are signed fractional numbers on the range -1.0 <= g <= 1.0. We check the "dx" and "dy" to see which has the greatest absolute value and that will be the direction that is incremented by 1, 0, or -1 and the other direction will increment by the (fractional) slope of the line with respect to the other direction.

Line 40 starts the plotting at the current pixel cursor location PLUS 0.5. We add 1/2 to the X and Y positions to "center" onto the pixel cell. If we didn't do this, we would notice dis-symmetry in plotting to the left and to the right. For example, 50.0 - 0.3 = 49.7 and 50.0 + 0.3 = 50.3. If we truncate these values, going left by 0.3 moves us to position 49 whereas going right by 0.3 makes us stay in the same position. This is dis-symmetry and makes plots look a bit off. Adding 0.5 corrects the problem.

Line 50 goes into a loop for the longest dimension of the line (the one incremented by 1). The <PLOT PX,PY> is not exactly BASIC; you substitute the call to the point plot routine described in the previous section. It repeatedly adds the X and Y increment values until the line is finished. This algorithm draws the line in the direction that your end points imply.

6.1. FRACTIONAL NUMBER REPRESENTATION

There are only two real complications to the machine language implementation are the representation of the signed fractional numbers and the division of the fractional numbers. To represent the numbers I use a 32-bit format with a 16-bit signed integer portion (-32768 to +32767) and a 16-bit fractional portion (0.0 to 0.99998474 in 0.00001526 increments). The weight values of the bit positions are as follows:

```
POS     31... 22 21 20 19 18 17 16  15  14  13  12    11   10     9 ...      0
VAL -32768... 64 32 16  8  4  2  1 1/2 1/4 1/8 1/16 1/32 1/64 1/128 ...1/65536
```

For example, 0...00001011101.10011010000...0 (notice the BINARY point) is 64 + 16 + 8 + 4 + 1 + 1/2 + 1/16 + 1/32 + 1/128 = 93.6015625 in decimal. Or, as a short cut, you can consider the integer 16 bits and the fractional 16 bits as independent words and add 1/65536th of the second to the first. Thus, for the example above, the quantity is 93 + (32768+4096+2048+512)/65536 = 93.6015625. (Good, they both match). The first calculation uses true base 2 whereas the second calculation uses base 65536.

Two's complement representation is used to achieve signedness (,Park!). We should all know about two's comp. Well, it works just fine for fractional binary numbers as well. In fact, it makes no difference at all. You can just think of the quantity as an integer number of 65536ths of a pixel and you get the integer operations of complement, add, and subtract for free. The easy way to get the two's comp. representation of a number is to take the positive binary image of the number and subtract it from 0 (this makes perfect sense since 0 - x = -x).

Fractional binary division is a little more complicated. There is no problem

with the binary point, since the laws of mathematics say we can multiply the top and bottom by the same quantity (like 65536) without changing the result, but handling the two's complement is a problem. What I did was figure out what the sign of the result of the division was going to be (by EORing the sign bits together) and then converted the two operands to their positive value if they were originally negative. This lets me perform a 32-bit unsigned division operation and then I convert the result to a negative if the result is supposed to be negative. The 32-bit divide is not all that complicated; it is done the same way that you would do it on paper (remember those days) except you use binary digits. The divide subroutine does not exactly rival supercomputer speeds, but it does get the job done.

6.2. MACHINE LANGUAGE OPERATION

While drawing the line, the X and Y coordinates are 32-bit signed fractional numbers as well as the "gx" and "gy" vector components ("go" values). Lines 20 and 30 require quite a bit of work in 6502 machine language and use the 32-bit add, subtract, 2's complement, and divide operations described in the previous section. To find the ABSolute value of a number, check to see whether it is positive or negative. If it is positive, you are done. If it is negative, just do a 2's complement on it (makes sense: 0 - (-x) = x).

Line 40 is done by simply taking the pixel cursor X and Y coordinates (which are stored as unsigned integers and are remembered between line draw calls) and put them into the high word of a 32-bit field and then put $8000 (32768) into the low word (which gives V + 1/2 (or V + 32768/65536)).

Line 50 is easily implemented as a loop that decrements the "r" word until it reaches zero, while calling the point plot routine and doing the 32-bit add to add the "g" values to the X and Y coordinates. When the line is finished, the final X and Y coordinates are put back into the pixel cursor position storage and the package is ready for the next call.

7. CONCLUSION

Ha! This ain't no formal paper so I don't have to write a conclusion. So there! [Ed.Note - He is currently working on his Masters thesis, so you'll have to pardon 'im here.. (grin)]

```
================================================================================
begin 640 hires80.bin
M`!-,`15,$1-,2!-,$!-,$!.S8*D`C0#_J>"B&B#,S:F'HAD@S,VI`*(2(,S-
MZ"#,S:D`(,K-J2"B&"#,S:D`HAZ@/R#,S8C0^F`@#,ZIDR#2_Z(9J4=,S,V$
M^H7\F*``A/L*)OL*)OL89?J0`N;["B;["B;["B;["B;[A?JE_(;]1OUJ1OUJ
M1OUJ&&7ZA?JE_67[A?NE_"D'JKV>$X7\8(!`(!`(!`(!```@5Q.E^Z(2(,S-
MI?KH(,S-(-C-!?RHI?NB$B#,S:7ZZ"#,S9A,RLVI`(5DA66%9H5GA5*%4X54
MHB`&8"9A)F(F8R92)E,F5*54T`JE4L50I5/E49`1.*52Y5"%4J53Y5&%4[`"
MQE0F9"9E)F8F9\K0R&"$4*90$!`XA5"I`.502(10J0#E4*AH8$BI`(5@A6$@
MSQ-H$!DXJ0#E9(5DJ0#E985EJ0#E9H5FJ0#E9X5G8*4,I`T@&A2%^H3[I1"D
M$2`:%(7\A/VE_,7ZI?WE^[!$I?J%$H50I?N%$X51I?R%8J7]A6.E$2`Q%*(#
MM625#LH0^:4-,`JI`(4-J0&%#-`&J?^%#84,.*D`Y1*%$JD`Y1.%$V"E_(42
MA5"E_843A5&E^H5BI?N%8Z4-(#$4H@.U9)4*RA#Y3.H4+/__+/__+/__I1$P
M"JD`A1&I`840T`:I_X41A1!,KA2P!X6+AHR$C6"%!(8%A`BB!ZD`E0K*$/LX
MI03EBX4,I07EC(4-.*4(Y8V%$*D`L`*I_X41(%@4I8N%!*6,A06I@(4#A0>I
M`(4"A0:EC84(J0"%":4$I@6D""H$Z42!1/P.QBE`F4*A0*E`V4+A0.E!&4,
MA02E!64-A048I09E#H4&I0=E#X4'I0AE$(4(I0EE$84)YA+0N^833%05I02F
&!:0(3`,5
`
end
================================================================================
begin 640 hires.demo
M`1PQ'&0`BR#"*-$H(C$S,$8B*2D@L[$@T2@B0C,B*2"G(/X1(DA)4D53.#`N
M0DE.(@!!'&X`GB#1*"(Q,S`S(BD`4AQX`$12LM$H(C$S,#`B*0!S'((`GB!$
M4BPS,C`@KR`R-34L,S(PK3(U-BPQ,#`L,`"%'(P`6+*U*+LH,2FL-C0P*0"7
M')8`6;*U*+LH,2FL,C`P*0"R'*``GB!$4BQ8(*\@,C4U+%BM,C4V+%DL,0"[
```

```
L'*H`H2!!)`#-'+0`BR!!)+(B(B"G(#$T,`#='+X`GB#1*"(Q,S`V(BD```"[
`
end
```

```
;*************************************************************************
;*  "HIRES80.BIN" hires line plotting package for the Commodore 128 80-col   *
;*  screen.                                                                  *
;*                                                                           *
;*  This package contains a couple of irregularities that I discovered       *
;*  while I was commenting it.  I left them in rather than start all over,   *
;*  since this package was written with a monitor rather than an assembler.  *
;*************************************************************************

;*************************************************************************
;*  package entry points                                                     *
;*************************************************************************

.$1300  [4c 01 15]  jmp $1501     ;jmp to draw line/position pixel cursor
.$1303  [4c 11 13]  jmp $1311     ;jmp to enter hires mode
.$1306  [4c 48 13]  jmp $1348     ;jmp to exit hires mode
.$1309  [4c 10 13]  jmp $1310     ;reserved
.$130c  [4c 10 13]  jmp $1310     ;reserved
.$130f: $b3                       ;library loaded identifier
.$1310  [60      ]  rts

;*************************************************************************
;*  enter hires mode                                                         *
;*************************************************************************

.$1311  [a9 00   ]  lda #$00      ;switch to bank 15 (kernal bank)
.$1313  [8d 00 ff]  sta $ff00
.$1316  [a9 e0   ]  lda #$e0      ;set color to light grey on black
.$1318  [a2 1a   ]  ldx #$1a
.$131a  [20 cc cd]  jsr $cdcc     ; ROM routine to write VDC register
.$131d  [a9 87   ]  lda #$87      ;enter bitmap mode (note - for version 2 VDC)
.$131f  [a2 19   ]  ldx #$19
.$1321  [20 cc cd]  jsr $cdcc
.$1324  [a9 00   ]  lda #$00      ;set VDC RAM address high
.$1326  [a2 12   ]  ldx #$12
.$1328  [20 cc cd]  jsr $cdcc
.$132b  [e8      ]  inx           ;set VDC RAM address low
.$132c  [20 cc cd]  jsr $cdcc
.$132f  [a9 00   ]  lda #$00      ;set VDC RAM data register to $00
.$1331  [20 ca cd]  jsr $cdca
.$1334  [a9 20   ]  lda #$20      ;select block fill mode
.$1336  [a2 18   ]  ldx #$18
.$1338  [20 cc cd]  jsr $cdcc
.$133b  [a9 00   ]  lda #$00      ;fill 256*63 VDC bytes with 0
.$133d  [a2 1e   ]  ldx #$1e      ;  to clear the hires screen
.$133f  [a0 3f   ]  ldy #$3f
.$1341  [20 cc cd]  jsr $cdcc
.$1344  [88      ]  dey
.$1345  [d0 fa   ]  bne $1341
.$1347  [60      ]  rts

;*************************************************************************
;*  exit hires mode                                                          *
;*************************************************************************

.$1348  [20 0c ce]  jsr $ce0c     ;reload the character sets
.$134b  [a9 93   ]  lda #$93      ;clear the text screen
.$134d  [20 d2 ff]  jsr $ffd2
.$1350  [a2 19   ]  ldx #$19      ;restore color text mode
.$1352  [a9 47   ]  lda #$47
.$1354  [4c cc cd]  jmp $cdcc
```

```
;*************************************************************************
;*calculate the bitmap byte address and bit value for pixel given x=.AX,y=.Y *
;*************************************************************************

.$1357  [84 fa   ]  sty $fa       ;save .A and .Y
.$1359  [85 fc   ]  sta $fc
.$135b  [98      ]  tya           ;put pixel cursor y position into .A
.$135c  [a0 00   ]  ldy #$00      ;clear pixel cursor y position high byte
.$135e  [84 fb   ]  sty $fb
.$1360  [0a      ]  asl a         ;multiply pixel cursor y by 2 giving y*2
.$1361  [26 fb   ]  rol $fb       ;  and we must shift the high byte to
.$1363  [0a      ]  asl a         ;again, giving y*4
.$1364  [26 fb   ]  rol $fb
.$1366  [18      ]  clc           ;add the original y, giving y*5
.$1367  [65 fa   ]  adc $fa
.$1369  [90 02   ]  bcc $136d
.$136b  [e6 fb   ]  inc $fb
.$136d  [0a      ]  asl a         ;multiply by 2 again, giving y*10
.$136e  [26 fb   ]  rol $fb
.$1370  [0a      ]  asl a         ;again, giving y*20
.$1371  [26 fb   ]  rol $fb
.$1373  [0a      ]  asl a         ;again, giving y*40
.$1374  [26 fb   ]  rol $fb
.$1376  [0a      ]  asl a         ;again, giving y*80: ha!  we are done
.$1377  [26 fb   ]  rol $fb
.$1379  [85 fa   ]  sta $fa       ;save low byte of y*80
.$137b  [a5 fc   ]  lda $fc       ;restore x coordinate low byte
.$137d  [86 fd   ]  stx $fd       ;set up x coordinate high byte
.$137f  [46 fd   ]  lsr $fd       ;divide the x coordinate by 2 giving x/2
.$1381  [6a      ]  ror a         ;  we must ror the high byte, then the low
.$1382  [46 fd   ]  lsr $fd       ;again, giving x/4
.$1384  [6a      ]  ror a
.$1385  [46 fd   ]  lsr $fd       ;again, giving x/8: done
.$1387  [6a      ]  ror a
.$1388  [18      ]  clc           ;now add y*80 and x/8
.$1389  [65 fa   ]  adc $fa
.$138b  [85 fa   ]  sta $fa
.$138d  [a5 fd   ]  lda $fd
.$138f  [65 fb   ]  adc $fb
.$1391  [85 fb   ]  sta $fb       ;giving us the pixel byte address in ($fa)
.$1393  [a5 fc   ]  lda $fc       ;get x mod 8
.$1395  [29 07   ]  and #$07      ;  ha! we can just extract the low three bits
.$1397  [aa      ]  tax
.$1398  [bd 9e 13]  lda $139e,x   ;look up the bit value in the table
.$139b  [85 fc   ]  sta $fc       ;  and save it at $fc
.$139d  [60      ]  rts           ;exit with address in ($fa) and value in $fc

;*************************************************************************
;*  bit value table                                                      *
;*************************************************************************

.$139e: $80  $40  $20  $10        ;bit values stored left to right
.$13a2: $08  $04  $02  $01

.$13a6  [00      ]  brk           ;filler - I forget why I put it here
.$13a7  [00      ]  brk

;*************************************************************************
;*  plot pixel at x=.AX, y=.Y on bitmap screen                           *
;*************************************************************************

.$13a8  [20 57 13]  jsr $1357     ;calculate the pixel address and value
.$13ab  [a5 fb   ]  lda $fb       ;set VDC RAM address high to pixel address
.$13ad  [a2 12   ]  ldx #$12
```

```
.$13af  [20 cc cd]  jsr $cdcc
.$13b2  [a5 fa   ]  lda $fa       ;set VDC RAM address low to pixel address
.$13b4  [e8      ]  inx
.$13b5  [20 cc cd]  jsr $cdcc
.$13b8  [20 d8 cd]  jsr $cdd8     ;peek the VDC RAM address
.$13bb  [05 fc   ]  ora $fc       ;OR on the new pixel value
.$13bd  [a8      ]  tay           ;  and save the result (byte to poke back)
.$13be  [a5 fb   ]  lda $fb       ;reset the VDC RAM address to the pixel
.$13c0  [a2 12   ]  ldx #$12      ;  address; this is necessary since the
.$13c2  [20 cc cd]  jsr $cdcc     ;  VDC will increment its RAM address on
.$13c5  [a5 fa   ]  lda $fa       ;  every access
.$13c7  [e8      ]  inx
.$13c8  [20 cc cd]  jsr $cdcc
.$13cb  [98      ]  tya
.$13cc  [4c ca cd]  jmp $cdca     ;and poke the new pixel byte value

;************************************************************************
;*  perform the unsigned 32-bit divide with 16-bit denominator (bottom)      *
;*    [$63 $62 $61 $60] is the numerator (top)                               *
;*            [$51 $50] is the denominator (bottom)                          *
;*    [$67 $66 $65 $64] is the quotient (result)                             *
;*        [$54 $53 $52] is the remainder                                     *
;************************************************************************

.$13cf  [a9 00   ]  lda #$00      ;set the result to 0
.$13d1  [85 64   ]  sta $64
.$13d3  [85 65   ]  sta $65
.$13d5  [85 66   ]  sta $66
.$13d7  [85 67   ]  sta $67

.$13d9  [85 52   ]  sta $52       ;clear the remainder
.$13db  [85 53   ]  sta $53
.$13dd  [85 54   ]  sta $54

.$13df  [a2 20   ]  ldx #$20      ;set the loop count to 32 bits
.$13e1  [06 60   ]  asl $60       ;shift out the high bit of the numerator
.$13e3  [26 61   ]  rol $61
.$13e5  [26 62   ]  rol $62
.$13e7  [26 63   ]  rol $63
.$13e9  [26 52   ]  rol $52       ;shift it into the remainder
.$13eb  [26 53   ]  rol $53
.$13ed  [26 54   ]  rol $54
.$13ef  [a5 54   ]  lda $54       ;check if the remainder is >= the denominator
.$13f1  [d0 0a   ]  bne $13fd
.$13f3  [a5 52   ]  lda $52
.$13f5  [c5 50   ]  cmp $50
.$13f7  [a5 53   ]  lda $53
.$13f9  [e5 51   ]  sbc $51
.$13fb  [90 11   ]  bcc $140e     ;if not, go to next bit
.$13fd  [38      ]  sec           ;subract the denominator from the remainder
.$13fe  [a5 52   ]  lda $52
.$1400  [e5 50   ]  sbc $50
.$1402  [85 52   ]  sta $52
.$1404  [a5 53   ]  lda $53
.$1406  [e5 51   ]  sbc $51
.$1408  [85 53   ]  sta $53
.$140a  [b0 02   ]  bcs $140e
.$140c  [c6 54   ]  dec $54
.$140e  [26 64   ]  rol $64       ;shift a "1" bit into the quotient.  Note
.$1410  [26 65   ]  rol $65       ;  the first "rol" should have been preceeded
.$1412  [26 66   ]  rol $66       ;  by a "sec"; this is a BUG!  However, it
.$1414  [26 67   ]  rol $67       ;  will fail only if denom >=32768 which
                                  ;  cannot happen in this application.
.$1416  [ca      ]  dex           ;go on to the next bit
.$1417  [d0 c8   ]  bne $13e1
```

```
.$1419  [60      ]  rts

;************************************************************************
;*  get the absolute value of the 2's comp number in .AY -> .AY         *
;************************************************************************

.$141a  [84 50   ]  sty $50
.$141c  [a6 50   ]  ldx $50
.$141e  [10 10   ]  bpl $1430     ;if the number is positive, exit
.$1420  [38      ]  sec           ;else take the 2's complement of the negative
.$1421  [85 50   ]  sta $50       ;  value to get the positive value
.$1423  [a9 00   ]  lda #$00
.$1425  [e5 50   ]  sbc $50
.$1427  [48      ]  pha
.$1428  [84 50   ]  sty $50
.$142a  [a9 00   ]  lda #$00
.$142c  [e5 50   ]  sbc $50
.$142e  [a8      ]  tay
.$142f  [68      ]  pla
.$1430  [60      ]  rts

;************************************************************************
;*  perform the fractional signed 32-bit divide                         *
;************************************************************************

.$1431  [48      ]  pha           ;remember the sign of the result
.$1432  [a9 00   ]  lda #$00      ;set the numerator fractional portion to .0
.$1434  [85 60   ]  sta $60
.$1436  [85 61   ]  sta $61
.$1438  [20 cf 13]  jsr $13cf     ;32-bit divide
.$143b  [68      ]  pla           ;if the sign of the result is supposed to be
.$143c  [10 19   ]  bpl $1457     ;  positive, then exit
.$143e  [38      ]  sec           ;if the sign of the result is negative, take
.$143f  [a9 00   ]  lda #$00      ;  get the 2's complement of the positive
.$1441  [e5 64   ]  sbc $64       ;  result
.$1443  [85 64   ]  sta $64
.$1445  [a9 00   ]  lda #$00
.$1447  [e5 65   ]  sbc $65
.$1449  [85 65   ]  sta $65
.$144b  [a9 00   ]  lda #$00
.$144d  [e5 66   ]  sbc $66
.$144f  [85 66   ]  sta $66
.$1451  [a9 00   ]  lda #$00
.$1453  [e5 67   ]  sbc $67
.$1455  [85 67   ]  sta $67
.$1457  [60      ]  rts

;************************************************************************
;*  get the X and Y plotting increments and the pixels-to-plot count    *
;************************************************************************

.$1458  [a5 0c   ]  lda $0c       ;get ABS(DX)
.$145a  [a4 0d   ]  ldy $0d
.$145c  [20 1a 14]  jsr $141a
.$145f  [85 fa   ]  sta $fa
.$1461  [84 fb   ]  sty $fb

.$1463  [a5 10   ]  lda $10       ;get ABS(DY)
.$1465  [a4 11   ]  ldy $11
.$1467  [20 1a 14]  jsr $141a
.$146a  [85 fc   ]  sta $fc
.$146c  [84 fd   ]  sty $fd

.$146e  [a5 fc   ]  lda $fc       ;compare ABS(DY) to ABS(DX)
.$1470  [c5 fa   ]  cmp $fa
```

```
.$1472  [a5 fd   ]  lda $fd
.$1474  [e5 fb   ]  sbc $fb
.$1476  [b0 44   ]  bcs $14bc     ;if ABS(DY) >= ABS(DX) then branch ahead

.$1478  [a5 fa   ]  lda $fa       ;set pixels-to-plot count to ABS(DX)
.$147a  [85 12   ]  sta $12
.$147c  [85 50   ]  sta $50
.$147e  [a5 fb   ]  lda $fb
.$1480  [85 13   ]  sta $13
.$1482  [85 51   ]  sta $51       ;set the numerator (top) to DY and the
.$1484  [a5 fc   ]  lda $fc       ;  denominator (bottom) to ABS(DX)
.$1486  [85 62   ]  sta $62
.$1488  [a5 fd   ]  lda $fd
.$148a  [85 63   ]  sta $63
.$148c  [a5 11   ]  lda $11       ;get the sign of DY - will be the sign of div
.$148e  [20 31 14]  jsr $1431     ;perform the signed fractional division
.$1491  [a2 03   ]  ldx #$03      ;store the result in the Y increment value
.$1493  [b5 64   ]  lda $64,x
.$1495  [95 0e   ]  sta $0e,x
.$1497  [ca      ]  dex
.$1498  [10 f9   ]  bpl $1493
.$149a  [a5 0d   ]  lda $0d       ;get the X increment
.$149c  [30 0a   ]  bmi $14a8
.$149e  [a9 00   ]  lda #$00      ;if DX is positive, X inc is +1
.$14a0  [85 0d   ]  sta $0d       ;  (note that DX cannot be 0 here so we don't
.$14a2  [a9 01   ]  lda #$01      ;   have to worry about that case)
.$14a4  [85 0c   ]  sta $0c
.$14a6  [d0 06   ]  bne $14ae
.$14a8  [a9 ff   ]  lda #$ff      ;if DX is negative, X inc is -1
.$14aa  [85 0d   ]  sta $0d
.$14ac  [85 0c   ]  sta $0c

.$14ae  [38      ]  sec           ;take the negative of the number of pixels
.$14af  [a9 00   ]  lda #$00      ;  to plot and exit
.$14b1  [e5 12   ]  sbc $12       ;I don't remember exactly why I use the
.$14b3  [85 12   ]  sta $12       ;  negative; there is not much of a speed
.$14b5  [a9 00   ]  lda #$00      ;  improvement.  Oh well, t'is done.
.$14b7  [e5 13   ]  sbc $13
.$14b9  [85 13   ]  sta $13
.$14bb  [60      ]  rts

.$14bc  [a5 fc   ]  lda $fc       ;set the pixels-to-plot count to ABS(DY)
.$14be  [85 12   ]  sta $12
.$14c0  [85 50   ]  sta $50
.$14c2  [a5 fd   ]  lda $fd
.$14c4  [85 13   ]  sta $13
.$14c6  [85 51   ]  sta $51       ;set the numerator (top) to DX and the
.$14c8  [a5 fa   ]  lda $fa       ;  denominator(bottom) to ABS(DY)
.$14ca  [85 62   ]  sta $62
.$14cc  [a5 fb   ]  lda $fb
.$14ce  [85 63   ]  sta $63
.$14d0  [a5 0d   ]  lda $0d       ;get the sign of DX - will be the sign of div
.$14d2  [20 31 14]  jsr $1431     ;do the 32-bit signed fractional division
.$14d5  [a2 03   ]  ldx #$03      ;store the result in the X increment
.$14d7  [b5 64   ]  lda $64,x
.$14d9  [95 0a   ]  sta $0a,x
.$14db  [ca      ]  dex
.$14dc  [10 f9   ]  bpl $14d7
.$14de  [4c ea 14]  jmp $14ea     ;jump over the next section
;-------
.$14e1  [2c ff ff]  bit $ffff     ;This section contained junk before and I
.$14e4  [2c ff ff]  bit $ffff     ;  don't know how it got here.  I replaced
.$14e7  [2c ff ff]  bit $ffff     ;  it with BITs and now jump over it.
;-------
.$14ea  [a5 11   ]  lda $11
```

```
.$14ec  [30 0a   ]  bmi $14f8
.$14ee  [a9 00   ]  lda #$00      ;if DY is positive then Y inc is +1
.$14f0  [85 11   ]  sta $11       ;  (we do not have to worry about the case
.$14f2  [a9 01   ]  lda #$01      ;   of DY being zero since then the increment
.$14f4  [85 10   ]  sta $10       ;   would not be important)
.$14f6  [d0 06   ]  bne $14fe
.$14f8  [a9 ff   ]  lda #$ff      ;if DY is negative then Y inc is -1
.$14fa  [85 11   ]  sta $11
.$14fc  [85 10   ]  sta $10
.$14fe  [4c ae 14]  jmp $14ae     ;jump back to the exit

;*************************************************************************
;*  main routine: draw line or set pixel cursor position                 *
;*************************************************************************

.$1501  [b0 07   ]  bcs $150a     ;goto draw routine if .C=1
.$1503  [85 8b   ]  sta $8b       ;store x and y pixel cursor coordinates
.$1505  [86 8c   ]  stx $8c
.$1507  [84 8d   ]  sty $8d
.$1509  [60      ]  rts           ;exit set pixel cursor

.$150a  [85 04   ]  sta $04       ;save draw-to coordinates
.$150c  [86 05   ]  stx $05
.$150e  [84 08   ]  sty $08
.$1510  [a2 07   ]  ldx #$07      ;clear $0a-$0d and $0e-$11
.$1512  [a9 00   ]  lda #$00
.$1514  [95 0a   ]  sta $0a,x
.$1516  [ca      ]  dex
.$1517  [10 fb   ]  bpl $1514

.$1519  [38      ]  sec           ;get dx value = DrawToX - PixelCursorX
.$151a  [a5 04   ]  lda $04       ;  dx is at [$0d $0c . $0b $0a]
.$151c  [e5 8b   ]  sbc $8b
.$151e  [85 0c   ]  sta $0c
.$1520  [a5 05   ]  lda $05
.$1522  [e5 8c   ]  sbc $8c
.$1524  [85 0d   ]  sta $0d

.$1526  [38      ]  sec           ;get dy value = DrawToY - PixelCursorY
.$1527  [a5 08   ]  lda $08       ;  dy is at [$11 $10 . $0f $0e]
.$1529  [e5 8d   ]  sbc $8d
.$152b  [85 10   ]  sta $10
.$152d  [a9 00   ]  lda #$00
.$152f  [b0 02   ]  bcs $1533
.$1531  [a9 ff   ]  lda #$ff
.$1533  [85 11   ]  sta $11

.$1535  [20 58 14]  jsr $1458     ;calculate the X and Y plotting increments

.$1538  [a5 8b   ]  lda $8b       ;set the drawing X position to x+0.5
.$153a  [85 04   ]  sta $04       ;  X is at [$05 $04 . $03 $02]
.$153c  [a5 8c   ]  lda $8c
.$153e  [85 05   ]  sta $05
.$1540  [a9 80   ]  lda #$80
.$1542  [85 03   ]  sta $03
.$1544  [85 07   ]  sta $07
.$1546  [a9 00   ]  lda #$00
.$1548  [85 02   ]  sta $02
.$154a  [85 06   ]  sta $06
.$154c  [a5 8d   ]  lda $8d       ;set the drawing Y position to y+0.5
.$154e  [85 08   ]  sta $08       ;  Y is at [$09 $08 . $07 $06]
.$1550  [a9 00   ]  lda #$00
.$1552  [85 09   ]  sta $09

.$1554  [a5 04   ]  lda $04       ;get the pixel X and Y coordinates
```

```
.$1556  [a6 05   ]  ldx $05
.$1558  [a4 08   ]  ldy $08
.$155a  [20 a8 13]  jsr $13a8     ;plot the pixel
.$155d  [a5 12   ]  lda $12       ;check the pixels-to-plot count for zero
.$155f  [05 13   ]  ora $13
.$1561  [f0 3b   ]  beq $159e     ;if no more pixels to plot, exit loop
.$1563  [18      ]  clc           ;add the X increment to the X coordinate
.$1564  [a5 02   ]  lda $02
.$1566  [65 0a   ]  adc $0a
.$1568  [85 02   ]  sta $02
.$156a  [a5 03   ]  lda $03
.$156c  [65 0b   ]  adc $0b
.$156e  [85 03   ]  sta $03
.$1570  [a5 04   ]  lda $04
.$1572  [65 0c   ]  adc $0c
.$1574  [85 04   ]  sta $04
.$1576  [a5 05   ]  lda $05
.$1578  [65 0d   ]  adc $0d
.$157a  [85 05   ]  sta $05
.$157c  [18      ]  clc           ;add the Y increment to the Y coordinate
.$157d  [a5 06   ]  lda $06
.$157f  [65 0e   ]  adc $0e
.$1581  [85 06   ]  sta $06
.$1583  [a5 07   ]  lda $07
.$1585  [65 0f   ]  adc $0f
.$1587  [85 07   ]  sta $07
.$1589  [a5 08   ]  lda $08
.$158b  [65 10   ]  adc $10
.$158d  [85 08   ]  sta $08
.$158f  [a5 09   ]  lda $09
.$1591  [65 11   ]  adc $11
.$1593  [85 09   ]  sta $09
.$1595  [e6 12   ]  inc $12       ;increment the pixels to plot count
.$1597  [d0 bb   ]  bne $1554     ;  note that it is stored as the negative of
.$1599  [e6 13   ]  inc $13       ;  the count
.$159b  [4c 54 15]  jmp $1554     ;repeat plotting loop

.$159e  [a5 04   ]  lda $04       ;exit - set the pixel cursor position to the
.$15a0  [a6 05   ]  ldx $05       ;  last pixel plotted on the line
.$15a2  [a4 08   ]  ldy $08
.$15a4  [4c 03 15]  jmp $1503
================================================================================
THE END
```