MACHINE LANGUAGE AND THE 8563

by Rick Kephart 5/21/88

This shows how to take whatever is in the memory area 8192-16191 (the normal HIRES screen on the 128), such as a DOODLE picture (which can be loaded with LOAD "DDfilename",B0,p7168), or a picture made by 128-BASIC, and put it up on the RGB screen.

The first part of this file shows to do it with a C-128 with only 16K video memory (unenhanced). The picture will be in black-and-white. The second part of this file shows how to do the same thing with 64K video memory (as in a C-128D), to produce a color picture.

To run the program, simply BLOAD"HIRES80",B15,P4864, make sure something is in the HIRES bitmap at 8192, switch to 80-columns, and type SYS4864, or whatever start address you load the program in at (it is completely relocatable: you can BLOAD"HIRES80",B15,P5000, the SYS5000, or use any available starting address, but you Must remain in Bank 15!). Then you can switch back to the Text screen simply by pressing the RUN/STOP key.

This was written using only the 128's built-in ML monitor. I will explain each step.

The first thing we will want to do is switch to FAST mode. This is a simple process. (This is actually simply BASIC's FAST routine). First we set Bit 4 in the Control Register at 53265 (HEX=$D011) to 0, by reading the value there with LDA and then ANDing the value there with Binary 11101111 (HEX=$6F), which turns off Bit 4 but does not affect anything else. This is the Bit which blanks the 40-column screen.

```
1300  AD 11 D0 LDA $D011
1303  29 6F    AND #$6F
1305  8D 11 D0 STA $D011
```

Then we go to the Clock Rate Register at 53296 (HEX=$D030) and turn on Bit 0. This changes the Clock Rate from 1 MHz (SLOW) to 2 MHz (FAST). The other bits are unimportant and need not be retained, so we can simply put a 1 in that register.

```
1308  A9 01    LDA #$01
130A  8D 30 D0 STA $D030
```

The next step is to set up the 8563 to Bitmap display. This is controlled by Register #25 (HEX=$19). The Register will also turn off Attribute memory, which we must also do. Attribute memory controls individual character colors, as well as Reverse and Blink. Since we will not have any room available for any attributes, they must be turned off.

To read or write to the 8563, we use the doorway at memory locations 54784 and 54785 (HEX=$D600 and $D601). It is only at these two memory locations that we can communicate with the 8563 Video Chip. This is done by storing the Register we want to read or write to $D600, then waiting

for Bit 7 at $D600 to be set to 1 (BPL will branch while Bit 7 is off), indicating that $D601 is ready for action. As soon as that happens, we go to work on $D601, reading or writing our value.

Register $19 will contain a different value depending on the particular 8563 chip in the particular 128 being used (there are two versions). Therefore, we cannot simply put a value there with Attributes-Off and Bitmap-On. Instead, we must first read the value that is there.

```
130D  A2 19    LDX #$19
130F  8E 00 D6 STX $D600
1312  2C 00 D6 BIT $D600
1315  10 FB    BPL $1312
1317  AD 01 D6 LDA $D601
```

Then we will put this normal value for Register $19 away. Let's use location $16, a Zero-page location normally used by BASIC, but available to us now since we're not using BASIC.

```
131A  85 16    STA $16
```

Now we must turn off Attributes and turn on Bitmap. Bits 6 and 7 of the register control these two 8563 things. To turn on the Bitmap, we must set Bit 7 to 1. We'll do this by ORing it with Binary 10000000 (HEX=$80) to turn on Bit 7 without affecting any other bit. To turn off the Attributes, we've got to get Bit 6 to 0. Let's do that by ANDing it with Binary 10111111 (HEX=$BF), which will turn Bit 6 off without affecting any other bit.

```
131C  09 80    ORA #$80
131E  29 BF    AND #$BF
```

Now we will use BASIC's built-in routine to write a vaue to a register, which is at 52684 (HEX=$CDCC). To do this, the register must be in the X-register, and the value to write must be in the Accumulator. The routine writes the register in X to $D600, waits for Bit 7 of $D600 to be on, then writes the value in the Accumulator to $D601. (The X-register still contains $19.)

```
1320  20 CC CD JSR $CDCC
```

The next thing we must do is set up the 8563 to start reading data, and storing it at location $0000 in the 8563's RAM memory. We have to start at $0000 because we need every byte from $0000 to the highest location, $3FFF. The Screen Memory's location, which is controlled by Registers 12 and 13 (HEX=$0C and $0D) are already set to 0 to start the screen at $0000, so we don't have to worry about them. But we do need to set the Current Memory Address registers 18 and 19 (HEX=$12 and $13) to $0000, so when we start writing to the 8563's RAM, the data will start at $0000. Once these have been set for the address at which we want to start writing, it is updated automatically for each byte we write (nice of it to

do that for us, isn't it?). (Unlike all other addresses with the 6502 or 6510 or 8510 microprocessors, addresses in the 8563 are written high-byte First, then low-byte, the opposite order. In this case, though, since the high- and low-bytes of the address are the same, this peculiarity is not visible here.) To do this, we will again use our built-in write-to-the-8563-chip routine at $CDCC. We will set the Accumulator to 0 and the X-register to $12. Then we will simply use INX to increase the X-register to write the low-byte to $13.

```
1323  A9 00    LDA #$00
1325  A2 12    LDX #$12
1327  20 CC CD JSR $CDCC
132A  E8       INX
132B  20 CC CD JSR $CDCC
```

The next byte in the program is just an NOP, separating the preliminaries from the actual main routine to write the one memory into the other.

```
132E  EA       NOP
```

Now the real work begins! The hardest part of this project is reading the VIC HIRES screen, because the VIC stores Bitmap memory in vertical blocks of 8 bytes in horizontal rows of 40 blocks across, whereas the 8563 uses 80 sequential bytes across each row. What we must do is read bytes from the VIC Bitmap in horizontal rows, and not sequentially. Here is a BASIC way to do this: FOR A = 8192 TO 16191 STEP 320: FOR B = 0 TO 7: FOR C = 0 TO 312 STEP 40: X = PEEK(A+B+C): NEXT C,B,A

That's what we have to do in Machine Language! Not an easy task. It is worth it, though, Because this BASIC routine is very slow. I decided to convert this exact routine into machine language. Here's how I did it:

I picked a couple of BASIC's Zero-page locations for a counter for 8192 to 16191. I chose locations $10 and $11, and stored 8192 (HEX=$2000) there to begin.

```
132F  A9 20    LDA #$20
1331  85 11    STA $11
1333  A9 00    LDA #$00
1335  85 10    STA $10
```

Here is where the biggest loop begins! We're going to copy the address stored in $10 & $11 to a couple more of BASIC's Zero-page locations (I'm sure BASIC won't mind), $12 and $13. This way, we can update the address in groups of 8 until we get to the end of a 40-byte row, and then update the address by 320 to go on to the next row.

```
1337  A5 10    LDA $10
1339  85 12    STA $12
133B  A5 11    LDA $11
133D  85 13    STA $13
```

I'm going to use another Zero-page location, $14, to count from 0 to 7. This will be used as a Y-index. We have to have a memory location to store it, because we'll be needing the Y-register later on.

```
133F  A9 00    LDA #$00
1341  85 14    STA $14
```

One more of BASIC's Zero-page locations, $15, we'll use to count from 1 to 40, to tell us when we've reached the end of a HIRES row. But let's use it as a countdown, starting by putting 40 (HEX=$28) and wait 'till we reach zero.

```
1343  A9 28    LDA #$28
1345  85 15    STA $15
```

Now let's get started, at last! We'll put the value in $14 (something between 0-7) into the Y-register, and read a byte from the VIC HIRES screen at 8192-16191.

```
1347  A4 14    LDY $14
1349  B1 12    LDA ($12),Y
```

Now comes the really interesting part. Since the 8563 Bitmap display is 640X199 bits and the VIC Bitmap is only 320X199 bits, every bit of the VIC display must be doubled to fill the entire 8563 Bitmap. This is one of those rare cases of numerical manipulations which are actually much easier to do in Machine Language than in BASIC! What will happen is each byte will be doubled into two byte, one with each of the first 4 bits doubled, and the other with the other 4 bits of the original number doubled (8 bits doubled = 16 bits, or 2 bytes).

The first thing we'll do is set up the X-register as a counter for the high 4 bits and then the low 4 bits of each byte (that is, we must run through this routine twice: once for each set of 4 bits). This will also serve as a Zero-page Index!

```
134B  A2 01    LDX #$01
```

Now, we'll copy the byte into two Zero-page locations. Let's give BASIC a break and use $FD and $FE.

```
134D  85 FD    STA $FD
134F  85 FE    STA $FE
```

Now we'll put a 4 in the Y-register to count down the 4 bits to double at a time.

```
1351  A0 04    LDY #$04
```

Now is the time to start doubling! We'll use the ML instruction ROL. The

Carry will hold whether the bit is 0 or 1. We don't have to worry about CLC or ASL, because we'll be using all 8 bits, and it won't matter what ends up in $FD and $FE when we're done! As each bit is rolled out into the carry, we'll roll it into a Zero-page location. The 2 bytes which will hold the final two bytes will be $FB and $FC, pointed to by the X-register! We have the byte in two memory locations, so it will be a simple matter to double the bit simply by rolling each bit out twice.

```
1353  26 FD    ROL $FD
1355  36 FB    ROL $FB,X
1357  26 FE    ROL $FE
1359  36 FB    ROL $FB,X
135B  88       DEY
135C  D0 F5    BNE $1353
```

Now the top four bits of $FD and $FE contain what used to be the low 4 bits. It makes no difference what's now in the 4 low bits of $FD and $FE, we'll never see them.

```
135E  CA       DEX
135F  F0 F0    BEQ $1351
```

Now $FC and $FB contain the value from the VIC screen with each bit doubled. So now we're ready to write two bytes to the 8563 Bitmap. The Memory read/write gateway to the 8563 is Register 31 (HEX=$1F). Again, we'll use the routine at $CDCC to write the two bytes to the 8563 RAM memory, and thereby put them up on the Bitmap display.

```
1361  A2 1F    LDX #$1F
1363  A5 FC    LDA $FC
1365  20 CC CD JSR $CDCC
1368  A5 FB    LDA $FB
136A  20 CC CD JSR $CDCC
```

Time now to go to the next horizontal byte of the VIC Bitmap. This byte is 8 away from the previous byte. We'll do this with a simple addition routine, adding 8 to the base address in $12 & $13.

```
136D  18       CLC
136E  A9 08    LDA #$08
1370  65 12    ADC $12
1372  85 12    STA $12
1374  A9 00    LDA #$00
1376  65 13    ADC $13
1378  85 13    STA $13
```

Now we'll use our countdown counter we've set up in $15 to see if we've reached the end of a row yet, and look back if we haven't.

```
137A  C6 15    DEC $15
```

```
137C  D0 C9    BNE $1347
```

Now we've got one line of bytes. We have 8 more lines of 40 bytes each to get. Remember, we use location $14 to store our Y-index offset. Each byte of each row will be 1 byte higher than the previous row. First, we're getting the first byte of each 8-byte block, so we have one horizontal row. The next horizontal row will consist of the second byte of each 8-byte block. The third row will be the third byte of each block, and so on until we've reached all 8 bytes. As we go to each byte of the block, we must maintain the base address as the first byte of the first block, so our offset will point to the right byte.

```
137E  E6 14    INC $14
1380  A5 10    LDA $10
1382  85 12    STA $12
1384  A5 11    LDA $11
1386  85 13    STA $13
```

Now we check to see if we've finished the 8-byte block.

```
1388  A5 14    LDA $14
138A  C9 08    CMP #$08
138C  D0 B5    BNE $1343
```

Now the time has come to jump to the next row of 8-byte blocks. We do this by adding 320 (HEX=$0140) to the base address. This is a simple addition routine.

```
138E  18       CLC
138F  A9 40    LDA #$40
1391  65 10    ADC $10
1393  85 10    STA $10
1395  A9 01    LDA #$01
1397  65 11    ADC $11
1399  85 11    STA $11
```

Now we check to see if we're finished. We'll be finished when we have worked our way up to location 16191 (HEX=$3F3F). We already have the high byte in the accumulator, so we'll check and see if that's $3F yet. If it is, then we'll check to see if the low byte is higher than $3F.

```
139B  C9 3F    CMP #$3F
139D  D0 98    BNE $1337
139F  A5 10    LDA $10
13A1  C9 3F    CMP #$3F
13A3  90 92    BCC $1337
```

Hurrah! The VIC Bitmap is now on display on the 8563 RGB screen! Now we'll keep this on the screen, until the STOP key has been pressed. Location 145 (HEX=$91) is constantly updated by the Kernal to contain the

value of the column of the Keyboard scan which has the STOP key. Bit 7 is cleared to 0 whenever the STOP key is pressed. BMI loops as long as Bit 7 is 1, so it will loop until the STOP key is being pressed.

```
13A5  A5 91    LDA $91
13A7  30 FC    BMI $13A5
```

The Bitmap is nice to look at, but eventually we'll want to be able to see characters on the screen again! Remember when we stored the original value of Register $19 in location $16? Well, here's where we finally use it! This will turn the Bitmap off and turn the Attributes back on again.

```
13A9  A2 19    LDX #$19
13AB  A5 16    LDA $16
13AD  20 CC CD JSR $CDCC
```

But we've still got us a problem here! Since we used the entire 16K to display our Bitmap, we have overwritten the entire 8563 character-set. We know have the text screen set up, but no character data to be able to print characters to print to it! The Kernal will come to our rescue here. When the 8563 is first initialized, VIC's character set is copied into the 8563 RAM, in the character memory storage area (which, by the way, is located at 8192 to 16383 (HEX=$2000 to $3FFF) in the 8563's RAM memory). The 128 Kernal's Jump Table has a entry called INIT80, which carries out this copying procedure, at 65378 (HEX=$FF62). It jumps to the actual routine which is in in the Screen Editor ROM at 49191 (HEX=$C027).

```
13B0  20 62 FF JSR $FF62
```

Now the Text screen is being displayed by the 8563, and we have all our characters in memory so we can display them. But there's still one thing wrong. Attribute memory is turned on, but is filled with strange data, whatever was in the Bitmap display from 2048 to 4096 (HEX=$0800 to $1000) which is where the 8563 stores the Attributes in its memory. That's why you see that bizarre display for an instant after the Bitmap is switched out. Attribute memory is easiest to clear by simply clearing the screen, by printing the CLR/HOME character of CHR$(147) (HEX=$93) through the Kernal output routine at $FFD2.

```
13B3  A9 93    LDA #$93
13B5  20 D2 FF JSR $FFD2
```

And now we're finished! Back to BASIC. Bye bye!

```
13B8  60       RTS
```

Switch to FAST mode

```
1300  AD 11 D0 LDA $D011
1303  29 6F    AND #$6F
1305  8D 11 D0 STA $D011
1308  A9 01    LDA #$01
130A  8D 30 D0 STA $D030
```

Now we set up the VDC for BITMAP display, but do NOT turn off attributes!

```
130D  A2 19    LDX #$19
130F  8E 00 D6 STX $D600
1312  2C 00 D6 BIT $D600
1315  10 FB    BPL $1312
1317  AD 01 D6 LDA $D601
131A  85 16    STA $16
131C  09 80    ORA #$80
131E  20 CC CD JSR $CDCC
```

Now, change the screen memory from $0000-$4000 up to $8000€$C000. The screen memory is determined by the values in registers 12 and 13 ($0C & $0D) in high-byte/low-byte format:

```
1321  A9 80    LDA #$80
1323  A2 0C    LDX #$0C
1325  20 CC CD JSR $CDCC
1328  E8       INX
1329  A9 00    LDA #$00
132B  20 CC CD JSR $CDCC
```

Now we set the current write-address to $8000 by putting that value into registers 18 & 19:

```
132E  A9 80    LDA #$80
1330  A2 12    LDX #$12
1332  20 CC CD JSR $CDCC
1335  A9 00    LDA #$00
1337  E8       INX
1338  20 CC CD JSR $CDCC
```

No we convert the VIC bitmap to the VDC bitmap, just like in the 128 version

```
133B  A9 20    LDA #$20
133D  85 11    STA $11
133F  A9 00    LDA #$00
1341  85 10    STA $10
1343  A5 10    LDA $10
1345  85 12    STA $12
1347  A5 11    LDA $11
1349  85 13    STA $13
```

```
134B  A9 00    LDA #$00
134D  85 14    STA $14
134F  A9 28    LDA #$28
1351  85 15    STA $15
1353  A4 14    LDY $14
1355  B1 12    LDA ($12),Y
1357  A2 01    LDX #$01
1359  85 FD    STA $FD
135B  85 FE    STA $FE
135D  A0 04    LDY #$04
135F  26 FD    ROL $FD
1361  36 FB    ROL $FB,X
1363  26 FE    ROL $FE
1365  36 FB    ROL $FB,X
1367  88       DEY
1368  D0 F5    BNE $135F
136A  CA       DEX
136B  F0 F0    BEQ $135D
136D  A2 1F    LDX #$1F
136F  A5 FC    LDA $FC
1371  20 CC CD JSR $CDCC
1374  A5 FB    LDA $FB
1376  20 CC CD JSR $CDCC
1379  18       CLC
137A  A9 08    LDA #$08
137C  65 12    ADC $12
137E  85 12    STA $12
1380  A9 00    LDA #$00
1382  65 13    ADC $13
1384  85 13    STA $13
1386  C6 15    DEC $15
1388  D0 C9    BNE $1353
138A  E6 14    INC $14
138C  A5 10    LDA $10
138E  85 12    STA $12
1390  A5 11    LDA $11
1392  85 13    STA $13
1394  A5 14    LDA $14
1396  C9 08    CMP #$08
1398  D0 B5    BNE $134F
139A  18       CLC
139B  A9 40    LDA #$40
139D  65 10    ADC $10
139F  85 10    STA $10
13A1  A9 01    LDA #$01
13A3  65 11    ADC $11
13A5  85 11    STA $11
13A7  C9 3F    CMP #$3F
13A9  D0 98    BNE $1343
13AB  A5 10    LDA $10
```

```
13AD  C9 3F    CMP #$3F
13AF  90 92    BCC $1343
```

Now here is where the color is added. First we move Attribute memory from its normal location at $0800 up to the unused area at $1000. This move is done so that the old attributes will be preserved when we switch back to the text display. The location of Attribute memory is determined by the values in registers 20 & 21 ($14 & $15)

```
13B1  A9 10    LDA #$10
13B3  A2 14    LDX #$14
13B5  20 CC CD JSR $CDCC
13B8  A9 00    LDA #$00
13BA  E8       INX
13BB  20 CC CD JSR $CDCC
```

Reset the current write to memory location

```
13BE  A9 10    LDA #$10
13C0  A2 12    LDX #$1213C2  20 CC CD JSR $CDCC
13C5  A9 00    LDA #$00
13C7  E8       INX
13C8  20 CC CD JSR $CDCC
```

40-column color memory is stored starting at location $1C00 (7168). Let's use a dynamic routine to read the colors, by storing the address within the program

```
13CB  A9 1C    LDA #$1C
13CD  8D D7 13 STA $13D7
13D0  A9 00    LDA #$00
13D2  8D D6 13 STA $13D6
```

Now we get the value, then convert it to 2 4-bit nibbles

```
13D5  AD 00 1C LDA $1C00
13D8  48       PHA
13D9  4A       LSR
13DA  4A       LSR
13DB  4A       LSR
13DC  4A       LSR
13DD  A8       TAY
```

There is a table of color translation between 40-column and 80-column stored in ROM starting at $CEC5. We'll put each nibble in the Y-register to use as an offset to get the equivalent 80-column color

```
13DE  B9 5C CE LDA $CE5C,Y
13E1  85 FE    STA $FE
```

```
13E3  68       PLA
13E4  29 0F    AND #$0F
13E6  A8       TAY
13E7  B9 5C CE LDA $CE5C,Y
```

  The two halves of the byte (foreground and background colors) are combined

```
13EA  0A       ASL
13EB  0A       ASL
13EC  0A       ASL
13ED  0A       ASL
13EE  05 FE    ORA $FE
13F0  A2 1F    LDX #$1F
```

  Each color must be stored in the 80-column bitmap twice, since the entire display is expanded twice as wide

```
13F2  20 CC CD JSR $CDCC
13F5  20 CC CD JSR $CDCC
```

  Loop back if not done (at $2000)

```
13F8  EE D6 13 INC $13D6
13FB  D0 D8    BNE $13D5
13FD  EE D7 13 INC $13D7
1400  AD D7 13 LDA $13D7
1403  C9 20    CMP #$20
1405  D0 CE    BNE $13D5
```

  Check for STOP key

```
1407  A5 91    LDA $91
1409  30 FC    BMI $1407
```

  Turn off bitmap and restore test screen

```
140B  A5 16    LDA $16
140D  A2 19    LDX #$19
140F  20 CC CD JSR $CDCC
```

  Put Attribute memory back at $0800

```
1412  A9 08    LDA #$08
1414  A2 14    LDX #$14
1416  20 CC CD JSR $CDCC
1419  E8       INX
141A  A9 00    LDA #$00
141C  20 CC CD JSR $CDCC
```

```
  Put screen memory back at $0000

141F  A9 00    LDA #$00
1421  A2 0C    LDX #$0C
1423  20 CC CD JSR $CDCC
1426  E8       INX
1427  20 CC CD JSR $CDCC
142A  60       RTS

  And return

End of file.
```

You can write to me at .

rmkq@Lphrc.org