The first in a series of tutorials for the beginner to intermediate machine language programmer

Machine Language Made BASIC

Part I: General Math

By William P. Nee

When I first started to learn machine language, setting up graphics seemed like a very complicated procedure. All those VDGs and SAMs were really discouraging. But then I realized that when you type in commands such as PMODE 4 or PCLS 2 or SCREEN 1,1, the computer must do something with those commands and numbers.

Why not find those same routines in ROM so you can just enter, for example, the PMODE numbers you want and then let the computer do all the work of setting the VDGs and SAMs? This series of articles is the result of several months of studying ROM routines to see where you can enter them, and it should help to make machine language programming a little more BASIC. These programs have been assembled with the *EDTASM+* cartridge.

In the next 13 articles, we'll use machine language for basic math functions: RND, PRINT, PRINT USING, PMODE, PSET, LINE, PLAY, etc. We'll make a "dump" program for a seven-dot printer, create music with six voices and wind up with a 3-D rotation program that includes perspective. Most of the machine language programs will have either an explanation or a BASIC program listing for comparison.

Many of these programs are written for ease of understanding rather than for maximum speed. You will find places where you can cut down on the number of commands and refine the program. Just be sure you understand how the whole program works before modifying it. Thanks to Adrian Kotik for debugging the programs and proofreading and assembling the articles.

Machine language is not a complicated language, just different. Commands are written in an abbreviated format such as LD for Load, ST for Store, SUB for Subtract, etc. Most numbers are written in Hex format, using the numerical Base 16; however, *EDTASM+* lets us use the more familiar Base 10, which we will be using in most of the programs. The $ prefix will indicate an address or location in Hex; #$ indicates a Hex number; # indicates a Base 10 number.

Bill Nee reversed the "snowbird" trend by retiring to Wisconsin from a banking career in Florida. He spends the long, cold winters writing programs for his CoCo.

All examples and methods are based on using *EDTASM+*. Be sure the computer is turned off before inserting the cartridge. Examples will usually originate (ORG) at Hex Location $3000 for compatability with either a 16K or 32K computer. Each example will end with SWI, although to *avoid* the End Error message, you may add an extra line by pressing TAB, typing END, pressing TAB again and typing START. If you do, the line after the ORG line should begin with something like START. Any listing you want to save on tape should have END START as the last line of the program.

The following examples will show how to add, subtract, multiply, and divide using machine language.

Example 1: Put the number 5 into Register A. Store the contents of Register A in Location $D0. Increase the contents of Location $D0 by 1. Load Register A with the contents of Location $D0. To put the program into memory, type A/IM/AO and let the program list to check for any errors; press Z to go to ZBUG, then type G3000. To check the result, press R to examine the registers. Register A now contains 6. The command INC can be used to add the number 1 to registers A, B or a memory location.

Example 1A: In this example, we load Register A with the number 5 and add to Register A (ADDA) the number 4. After running the program, examine the registers to see the result of 9 in Register A.

Example 2: This time the DEC command is used. The results show that DEC can be used to subtract the number 1 from registers A, B or a memory location. The commands INC and DEC are useful for the counting portion of your programs; however, most math problems are a little more complicated than 1 + 1.

Example 2A: The SUB command will subtract a number from Register A or Register B. In many cases, however, both registers A and B may contain numbers we need to use later.

The next six examples illustrate how to add, subtract, and multiply these two registers together.

Example 3: Load registers A and B with different numbers, save Register B in the S stack (PSHS B), add to Register A the contents of the S stack and increase the S stack pointer by 1 back to its original location (ADDA ,S+). When we look at Register A we see that it contains F instead of 15. This is the computer's way of displaying numbers in Base 16, or Hex. In Base 16, 10=A, 11=B, 12=C, 13=D, 14=E, 15=F and 16=10. So our answer of "F" is correct.

Example 4: The same procedure is used to subtract Register B from Register A.

Example 5: Another subtraction problem, but now when we look at Register A we see FB, not -5. The computer doesn't recognize negative numbers, it merely counts backwards from zero. The highest number that can be put into registers A or B is 255, or #$FF. Counting backwards from zero in Base 16 you, get #$FF, #$FE, #$FD, #$FC, #$FB, #$FA, etc. The fifth number counting backwards from zero is #$FB, so -5=#$FB.

Wait a minute — isn't #$FB also a positive number? Yes, in Base 10 it is (F*16)+B, or (15*16)+11=251. How does the computer — or operator — know which number is correct? In most math operations the computer will use "signed" numbers. If the left-most bit of an 8- or 16-bit number written in Base 2 is a 1, the number is considered to be negative; if it is a zero, the number is positive. So in registers A or B, numbers that appear to be greater than 127 may actually be negative numbers for math purposes.

In Example 5 our answer was #$FB. Since #$FB is greater than 127, we know the answer is actually a negative number. To use the "signed number check" you must convert #$FB in Hex to 11111011 in Base 2. (An easy way to convert Hex numbers to the Base 2 is to convert one Hex digit at a time. #$F equals 15, or 1111 in the Base 2; #$B equals 11 or 1011 in the Base 2, so #$FB=11111011.)

To prove that #$FB as a "signed" number is the same as -5, use the "Two's Compliment" procedure. Write the number in Base 2, reverse all the 0s to 1s and all the 1s to 0s, then add 1. In doing so, 11111011 becomes 00000100, and adding 1 gives us a result of 00000101. This equals 5, so our answer is -5!

Example 6: This time we loaded registers A and B each with a number and used MUL to multiply them. Since the result could be greater than the space available in just one register, we must read registers A and B together all as one number. Registers A and B together are called Register D, although the D does not appear on the screen. Examine Register D (A and B) to find the answer $32, which is 3*16+2, or 50.

Example 7: In this example, we have to read all of Register D to find an answer of $2710, which is (2*16)**3 + (7*16)**2 + 1*16 + 0, or 10000. (** is used here as a symbol for exponential.)

Example 8: Before we check the answer, an important point to remember is that MUL does not use "signed" numbers. So when we try to put in -100 or #$9C, the computer uses #$9C=156 and multiplies that by 100. The answer of Hex 3CF0 in Register D is 15600. Even larger numbers can be multiplied by using a ROM subroutine at Address $9FB5.

Example 9: Load Register D with a Hex number and Register X with another Hex number. Jumping to the subroutine at $9FB5 (JSR $9FB5) will put the product of Register D times Register X in Register Y and Register U. If we read Register Y and Register U together, the product is $06260060. You can check the result this time by converting the answer to the Base 10. Does this subroutine work with negative numbers?

So far, we've been working only with whole numbers, but that rarely occurs in math. The computer has two locations where it stores numbers up to nine digits long, including positive and negative numbers. These locations are called Floating Point Accumulator 1 and 2, or FP1 and FP2. To get a "signed" number from Register D to FP1, use the ROM routine at Address $B4F4; to get a number from FP1 back to Register D, use the routine at $B3ED. Remember, Register D can only hold a whole number.

Example 1

```
ORG   $3000
LDA   #5
STA   $D0
INC   $D0
LDA   $D0
SWI
```

Example 1A

```
ORG   $3000
LDA   #5
ADDA  #4
SWI
```

Example 2

```
ORG   $3000
LDA   #5
STA   $D0
DEC   $D0
LDA   $D0
SWI
```

Example 2A

```
ORG   $3000
LDA   #5
SUBA  #4
SWI
```

Example 3

```
ORG   $3000
LDA   #10
LDB   #5
PSHS  B
ADDA  ,S+
SWI
```

Example 4

```
ORG   $3000
LDA   #10
LDB   #5
PSHS  B
SUBA  ,S+
SWI
```

Example 5

```
ORG   $3000
LDA   #5
LDB   #10
PSHS  B
SUBA  ,S+
SWI
```

Example 6

```
ORG   $3000
LDA   #5
LDB   #10
MUL
SWI
```

Example 7

```
ORG   $3000
LDA   #100
LDB   #100
MUL
SWI
```

Example 8

```
ORG   $3000
LDA   #-100
LDB   #100
MUL
SWI
```

Example 9

```
ORG   $3000
LDD   #$1234
LDX   #$5678
JSR   $9FB5
SWI
```

Certain numbers already stored in the computer's memory that are used to perform internal calculations and their locations are:

Number	Location
-0.50	$843C
0.25	$BFC2
0.50	$BEC0
1.00	$BAC5
10.00	$BB7D

Each number is five bytes long and is in floating point format.

To handle internal calculations, the computer has five ROM routines that add, subtract, multiply, or divide. The symbol X means "the number at Location X." These routines are:

Function	Address
FP1=(X)+FP1	$B9C2
FP1=(X)-FP1	$B9B9
FP1=(X)*FP1	$BACA
FP1=(X)/FP1	$BB8F
FP1=FP2/(X)	$BB88

Example 10: Load Register D with the number 6, put that in FP1 (JSR $B4F4); load Register X with the location of the number 10 (LDX #$BB7D), add the number at the location in Register X to FP1 (JSR $B9C2). Transfer the result back to Register D (JSR $B3ED). Examine the register to find Register D is Hex 10, or 16.

Example 11: Use the routine at $B9B9 to subtract FP1 from the number at the location in Register X.

Example 12: The routine at $BACA is used to multiply the number at the location in Register X by FP1. The answer of 3C in Register D is 60.

Example 13: The routine at $BB8F divides the number at the location in Register X by FP1. Since Register D can only hold whole numbers, it shows a 1. The complete answer is in FP1, so we need a routine to display the entire answer instead of rounding it off to Register D. By using a combination of ROM routines, we can display the complete answer, including negative numbers, up to nine digits. (Putting a number back into Register D will lose the number in FP1.)

Example 13A: Type and run this example to see the entire

Example 10

```
ORG   $3000
LDD   #6
JSR   $B4F4
LDX   #$BB7D
JSR   $B9C2
JSR   $B3ED
SWI
```

Example 11

```
ORG   $3000
LDD   #6
JSR   $B4F4
LDX   #$BB7D
JSR   $B9B9
JSR   $B3ED
SWI
```

Example 12

```
ORG   $3000
LDD   #6
JSR   $B4F4
LDX   #$BB7D
JSR   $BACA
JSR   $B3ED
SWI
```

Example 13

```
ORG   $3000
LDD   #6
JSR   $B4F4
LDX   #$BB7D
JSR   $BB8F
JSR   $B3ED
SWI
```

Example 13A

```
ORG   $3000
LDD   #6
JSR   $B4F4
LDX   #$BB7D
JSR   $BB8F
JSR   $BDD9
LEAX  -1,X
JSR   B99C
JSR   $B958
SWI
```

answer displayed with all nine digits (and in the Base 10). The following routines are used in this example:

Function	Address
Transfer FP1 to a buffer at $3DA	$BDD9
Decrease location for sign	LEAX -1,X
Print buffer contents	$B99C
Print a carriage return	$B958

The main FP routines are:

Function	Address
Transfer FP1 to FP2 (and FP1)	$BC5F
Transfer FP2 TO FP1 (and FP2)	$BC4A
Register B (-128 to +127) to FP1	$BC7C
Register B (-128 to +127) + FP1	$BD99
Register D (-32768 to +32767) to FP1	$B4F4
(X) to FP1	$BC14
(X) to FP2	$BB2F
FP1 to (X)	$BC35
10*FP1	$BB6A
-1*FP1	$BEE9
FP1/10	$BB82
FP2/FP1	$BB91

Some other numbers stored in the computer's memory are:

Number	Location
-32768	$B3DF
PI/2	$83AB
2*PI	$BFBD or $BFE1
SQR(2)/2	$8432
SQR(2)	$8437
LOG(2)	$8441

The $B4F4 routine is used in most function commands. Once a number is in FP1, you can jump to the routine for any of the following:

Function	Location
SIN	$BF78
COS	$8378
TAN	$8381
SQR	$8480
EXP	$84F2
LOG	$8446
RND	$BF1F
INT	$BCCE

You can print the results in FP1 or use the routine at $B3ED to put the results back in Register D, but only as a whole number. For practice, develop a machine language program that will print the SIN of any angle you load into Register D. Remember, angles must be converted to radians, since trigonometric functions in the Color Computer are in radians. (Hint: one degree = 2*PI/360 radians.)

Now for our comparison programs. Both will perform a sort of numbers and graphics from screen locations &H400 to &H5FF. You may want to time each program. Is there a difference in the random portion? Is there much difference in the sorting portion? We will review these programs in the next article. Meanwhile, try making up your own programs using the ROM routines we've discussed.

(Questions or comments concerning this tutorial may be directed to the author at Route 2, Box 216C, Mason, WI 54856-9302. Please enclose an SASE when requesting a reply.) □

Listing 1: SORTBAS

```
11Ø CLS
12Ø FOR X=&H4ØØ TO &H5FF
13Ø POKE X,RND(255)
18Ø NEXT
2ØØ FLAG=1
22Ø FOR X=&H4ØØ TO &H5FE
23Ø A=PEEK(X):B=PEEK(X+1)
25Ø IF A<=B THEN 3ØØ
27Ø TEMP=A:A=B:B=TEMP
28Ø POKE X,A:POKE X+1,B
29Ø FLAG=Ø
3ØØ NEXT
32Ø IF FLAG=Ø THEN 2ØØ
34Ø A$=INKEY$:IF A$="" THEN 34Ø
37Ø END
```

Listing 2: SORTBIN

```
3ØØØ                ØØ1ØØ         ORG     $3ØØØ
3ØØØ BD      A928   ØØ11Ø START   JSR     $A928       11Ø CLS
3ØØ3 1Ø8E    Ø4ØØ   ØØ12Ø         LDY     #$4ØØ       12Ø FOR X=&H4ØØ TO &H5FF
3ØØ7 CC      ØØFF   ØØ13Ø LOOP1   LDD     #255        13Ø POKE X,RND(255)
3ØØA BD      B4F4   ØØ14Ø         JSR     $B4F4
3ØØD BD      BF1F   ØØ15Ø         JSR     $BF1F
3Ø1Ø BD      B3ED   ØØ16Ø         JSR     $B3ED
3Ø13 E7      AØ     ØØ17Ø         STB     ,Y+
3Ø15 1Ø8C    Ø5FF   ØØ18Ø         CMPY    #$5FF       18Ø NEXT
3Ø19 23      EC     ØØ19Ø         BLS     LOOP1
3Ø1B 86      Ø1     ØØ2ØØ SORT    LDA     #1          2ØØ FLAG=1
3Ø1D B7      3Ø4Ø   ØØ21Ø         STA     FLAG
3Ø2Ø 8E      Ø4ØØ   ØØ22Ø         LDX     #$4ØØ       22Ø FOR X=&H4ØØ TO &H5FE
3Ø23 EC      8Ø     ØØ23Ø LOOP2   LDD     ,X+         23Ø A=PEEK(X):B=PEEK(X+1)
3Ø25 34      Ø4     ØØ24Ø         PSHS    B
3Ø27 A1      EØ     ØØ25Ø         CMPA    ,S+         25Ø IF A<=B THEN 3ØØ
3Ø29 23      Ø7     ØØ26Ø         BLS     CONT
3Ø2B 1E      89     ØØ27Ø         EXG     A,B         27Ø TEMP=A:A=B:B=TEMP
3Ø2D ED      1F     ØØ28Ø         STD     -1,X        28Ø POKE X,A:POKE X+1,B
3Ø2F 7F      3Ø4Ø   ØØ29Ø         CLR     FLAG        29Ø FLAG=Ø
3Ø32 8C      Ø5FE   ØØ3ØØ CONT    CMPX    #$5FE       3ØØ NEXT
3Ø35 23      EC     ØØ31Ø         BLS     LOOP2
3Ø37 7D      3Ø4Ø   ØØ32Ø         TST     FLAG        32Ø IF FLAG=Ø THEN 2ØØ
3Ø3A 27      DF     ØØ33Ø         BEQ     SORT
3Ø3C BD      ADFB   ØØ34Ø         JSR     $ADFB       34Ø A$=INKEY$:IF A$="" THEN 34Ø
3Ø3F 3F             ØØ35Ø         SWI
3Ø4Ø                ØØ36Ø FLAG    RMB     1
             3ØØØ   ØØ37Ø         END     START       37Ø END

ØØØØØ TOTAL ERRORS
```