FORTH FACTORY#

6502 DISASSEMBLER#

by JOHN MATTES

see also 6502 Disassembler

This article describes a Forth program to disassemble 6502 machine code instructions. Using it, you can get a listing of assembler mnemonics to help you figure out how programs written by others are put together. This is one of the best ways of improving your programming skills.

The article is divided into two parts. The first part provides some background information on the 6502 instruction set, to help you understand how the disassembler works. It is not necessary to read this part to get the disassembler working, but it will help you to understand the output. The second part describes the program itelf and gives a sample result from running it.

Machine instructions can contain up to three bytes, the first of which is the operation code (telling the machine what to do), and the remainder give the operand or its address. These "address" bytes can be interpreted in one of several different ways, depending on the "addressing mode".

Imagine that we are considering an instruction with opcode "OP" and the next two bytes contain the two hex numbers AB and CD, respectively (remember that each 8-bit byte contains 2 hex numbers):

```
_________________
: O P : A B : C D :
_________________
```

The "Table of Address Modes" lists the various 6502 addressing modes and describes how the hex numbers ABCD following the opcode are to be interpreted.

In the 6502, absolute addresses require two bytes and the most significint digits of the address are stored in the byte with the highest address. That is why the absolute addresses are shown as CDAB. The notation (X) indicates "the contents of the X register." In this notation (OOAB) + (X) indicates "the contents of the memory byte at address OOAB plus the contents of the X register. A comma is used to separate the high and low bytes of an address where clarity requires.

All multiple address mode instructions in the 6502 instruction set can be used in the absolute address mode. The numerical "mode number" shown in the Table of Address Modes is the difference between an instruction's absolute address opcode and its "mode" opcode (plus hex 10 to avoid negative mode numbers).

A table of the absolute address opcodes (+10) for the various multiple address instructions, called MULTIMODE, is included in screen 30. Given an arbitrary opcode (say 65) we can find the first entry in MULTIMODE which exceeds the opcode (7D in this case) and subtract to get the mode number (08, corresponding to Zero Page, X). The mnemonic can be read as ADC from the ninth entry in MULTINAME (7D is the Ienth entry in MULTIMODE).

The 22 entries in MULTIMODE account for 117 of the 151 valid 6502 opcodes (out of a maximum of 256 possible). The remaining 34 opcodes each identify a single address mode instruction and are dealt with by looking up tables called ONEMODE and ONENAME. These tables also include ten, renegade, multiple-address opcodes that, for reasons best known to the 6502 designers, don't result in correct mode number. The most irregular instruction is LDX, where only two of its five address modes fit the pattern. Th at is why LDX appears three times in ONENAME.

Given an arbitrary opcode, the first step in the disassembly process is to check the list of single address mode opcodes.

If the wanted opcode is present, the instruction type is known immediately. If not, the list of multiple mode opcodes is used to determine both the instruction type and the addressing mode.

Now for the disassembler program itself. The listing appears in screens 30 to 35. As is usual in Forth listings, the interesting part of the program appears last. The first few screens contain the building blocks from which the main program, called "DISASSEMBLE,'" or "DIS" for short, is constructed.

Before describing how DISASSEMBLE works, I shall define what each of the words used in the program does. POINTER A variable containing the address of the current opcode. ONEMODE A table containing the opcodes of those instructiclns which have only one addressing mode. Each entry consists of two bytes; the first byte gives the mode number and the second is the opcode. STRING Compile the following text stream into the dictionary. ONENAME A table containing the mnemonics of those instructions which have only one addressing mode. MULTIMODE A table of base codes for those instructions which have multiple address modes. The "base code" for an instruction is its absolute mode opcode plus hex 10. MULTINAME A table containing the mnemonics of those instructions which have multiple address modes. MODE A table containing mnemonics describing the various addressing modes. LENGTH A table giving the number of by which follow the opcode for the various addressing modes. SEARCH OP add len -- I f

Searches a table of two-byte words of length "len" beginning at address "add" for a match to the single byte OP. The table must be arranged in ascending order; I is the index number of the first table entry, which is equal to OP (f=1) or exceeds OP (f = 0). PRINTNAME I add - - Prints three characters (i.e, instruction mnemonic) beginning at address add + 3*I. PRINTMODE MODE/4 -- Print the two-character mnemonic corresponding to the addressing mode "MODE". PRINTADD MODE/4--f Prints the one- or two-byte "address" (if one exists) following the opcode. Sets f to 1 (to terminate disassembly) if the opcode is one of five instructions which can cause a jump. CHKMODE I X -- J MODE/4 Calculates the address mode of the current opcode against the Ith base code in the MULTIMODE table and checks whether this is modulo four (i.e, divisible by four with no remainder), if so, J = I, otherwise J = I + 1. DISASSEMBLE add -- Disassemble the code beginning at address "add". DIS A synonym for DISASSEMBLE.

Armed with the definitions of the "building blocks," we can now analyze the "main program." I have found the coding form used in the box headed "Description of Disassemble" useful for both analyzing and writing Forth code. The first column is an instruction number(for reference);the second, the contents of the stack; and the third, the instruction (Forth word). The number against a stack entry indentifies the instruction removing that entry from the stack.

You can use a form like the one I have just described to analyze the remaining code. Of course, you will need to know Forth or have in front of you the fig-Forth glossary. Finally, here is an example of using the disassembler to see how the Forth word C@ replaces an address on the stack with the contents of that address.

The process is started by entering 'C@DIS [RETURN]. This sequence puts the parameter field address of C@ on the stack and starts disassembly. The result looks like:

```
13FB  LDA X) 0
13FD  STA ,X 0
13FF  STY  ,X 1
1401 JMP   .....  F47      OK
```

Note that the address O,X points to the byte on the bottom of the data stack (it grows down!) and 1,X is the next byte up. F47 is the address of the Forth procedure NEXT, which passes execution to the next Forth word.

The first instruction loads the accumulator with the byte which was at the 16-bit address on the "top" (physically at the bottom) of the stack. The second instruction at 13FF stores the contents of the Y register (which you can count on being zero) into the high order byte on "top" of the stack. Thus the address on the "top" of the stack is replaced by the byte which was (and still is) stored at that address.

A word of warning: DISASSEMBLE will disassemble anything! It does not try to stop you from disassembling data, Forth code or even machine code starting at the wrong point. However, you can easily detect a listing of gibberish. The listing will tend to be long (over a screen), the addresses will be all over the place and rarely used instructions will pop up frequently.

Listing: DISASM.4TH

```
SCR #30
( FORTH DISASSEMBLER ANTIC 3/84 )
HEX 0 VARIABLE MULTIMODE -2 ALLOT
1D , 1E , 3C , 3D , 3E , 5C ,
5D , 5E , 7D , 7E , 9C , 9D ,
9E , BC , BD , BE , DC , DD ,
DE , FC , FD , FE ,

: SEARCH 1 + 0 DO OVER OVER I 2
    * + C@ - DUP 0= IF DROP
    DROP DROP I 1 LEAVE ELSE
    0 IF DROP DROP I 0 LEAVE
    ENDIF ENDIF LOOP ;

0< VARIABLE POINTER

-->

SCR #31
: STRING ( COMPILE TEXT )
    BL BLK @ IF BLK @ BLOCK ELSE
    TIB @ ENDIF IN @ + SWAP
    ENCLOSE IN +! OVER - >R +
    HERE R CMOVE R> ALLOT ;

0 VARIABLE MULTINAME -2 ALLOT

STRING ORAASLBITANDROLJMPEORLSR
STRING ADCRORSTYSTASTXLDYLDALDX
STRING CPYCMPDECCPXSBCINC

: CHKMODE DROP DUP 2 * '
    MULTIMODE + @ POINTER @ C@ -
    4 /MOD SWAP IF SWAP 1+ SWAP
    ENDIF ;   -->

SCR #32
0 VARIABLE ONEMODE -2 ALLOT
2C00 , 2C08 , 280A , 3010 ,
2C18 , 1020 , 2C28 , 282A ,
3030 , 2C38 , 2C40 , 2C48 ,
284A , 3050 , 2C58 , 2C60 ,
2C68 , 286A , 246C , 3070 ,
2C78 , 2C88 , 2C8A , 3090 ,
2096 , 2C98 , 2C9A , 14A0 ,
14A2 , 2CA8 , 2CAA , 30B0 ,
20B6 , 2CB8 , 2CBA , 04BE ,
2CC8 , 2CCA , 30D0 , 2CD8 ,
2CE8 , 2CEA , 30F0 , 2CF8 ,
00FF , ( 00FF IS A DUMMY )


-->
```

```
SCR #33
0 VARIABLE ONENAME -2 ALLOT
STRING BRKPHPASLBPLCLCJSRPLPROL
STRING BMISECRTIPHALSRBVCCLIRTS
STRING PLARORJMPBVSSEIDEYTXABCC
STRING STXTYATXSLDYLDXTAYTAXBCS
STRING LDXCLVTSXLDXINYDEXBNECLD
STRING INXNOPBEQSED???

0 VARIABLE MODE -2 ALLOT
STRING ,X,Y,X)Y..##0PX).Y().AIMRE

0 VARIABLE LENGTH -2 ALLOT
2 C, 2 C, 1 C, 1 C, 2 C, 1 C,
1 C, 1 C, 1 C, 2 C, 0 C, 0 C,
1 C, 0 C,
-->
```

```
SCR #34
: PRINTNAME SPACE SWAP 3 * +
            3 TYPE 2 SPACES ;

: PRINTMODE 2 * ' MODE + 2 TYPE
            2 SPACES ;

: PRINTADD POINTER @ C@ DUP 20
    = OVER 40 = OR OVER 4C = OR
    OVER 60 = OR SWAP 6C = OR
    SWAP ' LENGTH + C@ 1 POINTER
    +! POINTER @ OVER POINTER +!
    OVER 0= IF DROP DROP ELSE
    OVER 1 = IF C@ . DROP ELSE
    @ 0 D. DROP ENDIF ENDIF ;

-->
```

```
SCR #35
: DISASSEMBLE POINTER ! CR
  BEGIN CR
    POINTER @ DUP 0 D. 2 SPACES
    C@ ' ONEMODE 2D SEARCH
      IF ( FOUND ) DUP ' ONENAME
         PRINTNAME 2 * 1+ '
         ONEMODE + C@ 4 /
      ELSE ( NOT ) DROP POINTER @
```

```
         C@ ' MULTIMODE 16 SEARCH
         CHKMODE CHKMODE CHKMODE
         SWAP ' MULTINAME
         PRINTNAME ENDIF
    DUP PRINTMODE PRINTADD
    ?TERMINAL OR
  UNTIL ;
: DIS DISASSEMBLE ;
```

DESCRIPTION OF DISASSEMBLER#

Step	Stack	Instruction	Comment	
	OPadd (2)		Address of current opcode	
1		POINTER!	Store OPadd in POINTER	
3		CR	Start a new line.	
4		BEGIN CR	Start a loop with a new line.	
6	OPadd (11)	POINTER@	Fetch the opcode address.	
7	OPadd (9)	DUP		
8	O (9)	O	Print the address (double precision -	
9		D.	to avoid negative addresses!)	
10		2 SPACES	Leave 2 spaces	
11	OP (14)	C@	Fetch the opcode.	
12	OMad (14)	' ONEMODE	Calculate the start address for ONEMODE table.	
13	2D (14)	2D	ONEMODE is 2D (45) entries long.	
14	I (20)(27)	SEARCH	For OP in ONEMODE table. Leave -	
	f (15)		Index and flag.	
15		IF	Test f. False part starts Step 26.	
16	I (18)	DUP	TRUE part (f=1),i.e., OP on ONEMODE table.	
17	ONad (18)	' ONENAME	Start address for ONENAME table.	
18		PRINTNAME	Type the mnemonic	
20	2I + 1 (23)	2 * 1+		
22	OMad (23)	' ONEMODE	Start address of ONEMODE table.	
23	OMad+2I+1 (24)	+	Adress of MODE for entry I.	
24	MODE (25)	C@	Fetch MODE.	
25	MODE/4 (41)	4 /	True part jumps to Step 38.	
26		ELSE	FALSE part OP not in ONEMODE	
27		DROP	The index left by SEARCH (Step 14).	
28	OP	POINTER @ c @	Prepare to search MULTIMODE -	
29	MMad	' MULTIMODE	table for opcode. Table is	
30	16	16	16 (hex) entries long.	

31	I (32)	SEARCH	For OP. Leave Index and flag.	
	f (32)		f is not used in this case	
32	J (33)	CHKMODE	Check whether MODE for entry -	
	MODE/4 (33)		I in MULTIMODE table is -	
33	J (34)	CHKMODE	divisible by 4. If it is -	
	MODE/4 (34)		return J = I otherwise J = I + 1.	
34	J (35)	CHKMODE	It may be necessary to -	
	MODE/4 (41)		increment I twice (3 CHKMODEs).	
35	J (37)	SWAP		
36	MNad (37)	' MULTINAME	Start address for MULTINAME table	
37		PRINTNAME	Start address for MULTINMAE table	
38		ENDIF	terminates the IF at line 15.	
39	MODE/4 (40)	DUP		
40		PRINTMODE	Print address mode mnemonics	
41	f (43)	PRINTADD	Print the address part of theinstruction and update the pointer. f = 1 indicates a jump instruction (finish).	
42	f (43)	?TERMINAL	f = 1 indicates a key is pressed (finish).	
43	f (44)	OR		
44		UNTIL	Jump to BEGIN (step 4) if f = 0	
45		;	END	

'John Mattes, from Syndney, Australia, is an electrical engineer who has worked in telecommunications for 20 years. He says he is "absorbed" in using Forth with his Atari 800.'