A FORTH ASSEMBLER FOR THE 6502#

by William F. Ragdale

Table of Contents

Source: 6502 Assembler in Forth

INTRODUCTION#

This article should further polarize the attitudes of those outside the growing community of FORTH users. Some will be fascinated by a label-less, macro-assembler whose source code is only 96 lines. Others will be repelled by reverse Polish syntax and the absence of labels.

The author immodestly claim that this is the best FORTH assembler ever distributed. It is the only such assembler that detects all errors in op-code generation and conditional structuring. It is released to the public domain as a defense mechanism. Three good 6502 assemblers were submitted to the FORTH Interest Group but each had some lack. Rather than merge and edit for publication, I chose to publish mine with all the submitted features plus several more.

Imagine having an assembler in 1300 bytes of object code with:

1. User macros (like IF,UNTIL,) definable at any time.
2. Literal values expressed in any numeric base, alterable at any time.
3. Expressions using any resident computation capability.
4. Nested control structures without labels, with error control.
5. Assembler source itself in a portable high level language.

OVERVIEW#

Forth is provided with a machine language assembler to create execution procedures that would be time inefficient, if written as colon-definitions. It is intended that "code" be written similarly to high level, for clarity of expression. Functions may be written first in high-level, tested, and then re-coded into assembly, with a minimum of restructuring.

THE ASSEMBLY PROCESS#

Code assembly just consists of interpreting with the ASSEMBLER vocabulary as CONTEXT. Thus, each word in the input stream will be matched according the Forth practice of searching CONTEXT first then CURRENT:

```
ASSEMBER   (now CONTEXT)
FORTH      (chained to ASSEMBLER)
user's     (CURRENT if one exists)
FORTH      (chained to the user's vocab)
try for literal number
else, do error abort
```

The above sequence is the usual action of Forth's text interpreter, which remains in control during assembly.

During assembly of CODE definitions, Forth continues interpretation of each word encountered in the input stream (not in the compile mode). These assembler words specify operands, address modes, and opcodes. At the conclusion of the CODE definition a final error check verifies correct completion by "unsmudging" the definition's name, to make it available for dictionary searches.

RUN-TIME, ASSEMBLY-TIME#

One must be careful to understand at what time a particular word definition executes. During assembly, each assembler word interpreted executes. Its function at that instant is called 'assembling' or 'assembly-time'. This function may involve op-code generation, address calculation, mode selection, etc.

The later execution of the generated code is called 'run-time'. This distinction is particularly important with the conditionals. At assembly time each such word (i.e., IF,UNTIL, BEGIN, etc.) itself 'runs' to produce machine code which will later execute at what is labeled 'run-time' when its named code definition is used.

AN EXAMPLE#

As a practical example, here's a simple call to the system monitor, via the NMI address vector (using the BRK opcode).

```
CODE MON ( exit to monitor )
     BRK,
NEXT JMP,
END-CODE
```

The word CODE is first encountered, and executed by Forth. CODE builds the following name "MON" into a dictionary header and calls ASSEMBLER as the CONTEXT vocabulary.

The "(" is next found in FORTH and executed to skip til ")". This method skips over comments. Note that the name after CODE and the ")" after "(" must be on the same text line.

OP-CODES#

BRK, is next found. in the assembler as the op-code. When BRK, executes, it assembles the byte value 00 into the dictionary as the op-code for "break to monitor via "NMI".

Many assembler word names end in ",". The significance of this is:

1. The comma shows the conclusion of a logical grouping that would be one line of classical assembly source code.
2. "," compiles into the dictionary, thus a comma implies the point at which code is generated.
3. The "," distinguishes op-codes from possible hex numbers ADC and ADD.

NEXT#

Forth executes your word definition under control of the address interpreter, named NEXT. This short code routine moves execution from one definition, to the next. At the end of your code definition, you must return control to NEXT or else to code which returns to NEXT.

RETURN OF CONTROL#

Most 6502 systems can resume execution after a break, since the monitor saves the CPU register contents. Therefore, we must return control to Forth after a return from the monitor. NEXT is a constant that specifies the machine address of Forth's address interpreter (say $0242). Here it is the operand for "JMP,". As "JMP," executes, it assembles a machine code jump to the address of NEXT from the assembly time stack value.

SECURITY#

Numerous tests are made within the assembler for user errors:

1. All parameters used in CODE definitions must be removed.
2. Conditionals must be properly nested and paired.
3. Address modes and operands must be allowable for the op-codes

These tests are accomplished by checking the stack position (in CSP) at the creation of the definition name and comparing it with the position at END-CODE. Legality of address modes and operands is insured by means of a bit mask associated with each operand.

Remember that if an error occurs during assembly, END-CODE never executes. The result is that the "smudged" condition of the definition name remains in the "smudged" condition and will not be found during dictionary searches.

The user should be aware that one error not trapped is referencing a definition in the wrong vocabulary: i.e., `0=` of ASSEMRLER when you want `0=` of FORTH

(Editor's note: the listing assumes that the figFORTH error messages are already available in the system, as follows: ?CSP issues the error message "DEFINITION NOT FINISHED" if the stack position differs from the value saved in the user variable CSP, which is set at the creation of the definition name.

?PAIRS issues the error message "CONDITIONALS NOT IMPAIRED" if its two arguments do not match.

3 ERROR prints the error message "HAS INCORRECT ADDRESS MODE".)

SUMMARY

The object code of our example is:

```
3059  83 4D 4F CE     CODE MON
305D  4D 30           link field
305F  61 30           code field
3061  00              BRK
```

```
3062  4C 42 02       JMP NEXT
```

OP-CODES, revisited#

The bulk of the assembler consists of dictionary entries for each op-code. 6502 one mode op-codes are:

BRK, CLC, CLD, CLI, CLV, DEX, DEY, INX, INY, NOP, PHA, PHP, PLA, PLP, RTI, RTS, SEC, SED, SEI, TAX, TAY, TSX, TXS, TXA, TYA,

When any of these are executed, the corresponding op-code byte is assembled into the dictionary.

The multi-mode op-codes are:

ADC, AND, CMP, EOR, LDA, ORA, SBC, STA, ASL, DEC, INC, LSR, ROL, ROR, STX, CPX, CPY, LDX, LDY, STY, JSR, JMP, BIT,

These usually take an operand, which must already be on the stack. An address mode may also be specified. If non is given, the op-code uses z-page or absolute addressing. The address modes are determined by:

Symbol	Mode	Operand
.A	accumulator	none
#	immediate	8 bits only
,X	indexed X	z-page or absolute
,Y	indexed Y	z-page or absolute
X)	indexed indirect X	z-page only
)Y	indirect index Y	z-page only
)	indirect	absolute only
one	memory	z-page or absolute

EXAMPLES

Here are examples of Forth vs. conventional assembler. Note that the operand comes first, followed by any mode modifier, and then the op-code mnemonic. This makes best use of the stack at assembly time. Also, each assembler word is set off by blanks, as is required for all Forth source text.

```
     .A ROL,      -->  ROL A
    1 # LDY,      -->  LDY #1
 DATA ,X STA,     -->  STA DATA,X
 DATA ,Y CMP,     -->  CMP DATA,Y
    6 X) ADC,     -->  ADC (06,X)
POINT )Y STA,     -->  STA (POINT),Y
VECTOR ) JMP,     -->  JMP (VECTOR)
```

(work in Progress)