ASM - DOS/65 ASSEMBLER

VERSION 2.1

TABLE OF CONTENTS

SECTION 1 - INTRODUCTION

1.1 OVERVIEW AND CONCEPT

The DOS/65 assembler, file ASM.COM, assembles source files of type .ASM from disk and generates a hexadecimal code file of type .KIM. The .KIM file may be converted to a binary file using the program MAKECOM.COM and then executed. The .KIM file is also a normal ASCII text file and may be edited using the DOS/65 editor, EDIT.COM. The .KIM file may also be loaded (with or without an offset) and debugged using the DOS/65 debugger DEBUG.COM. The assembler also creates a listing file of type .PRN showing the source code and the object code.

1.2 KEY FACTS

The assembler conforms to the MOS Technology standards and has a few extensions. Specifically, the assembler:

a. Allows use of the horizontal tab character ($09) or (ctl-i)) as a field separator instead of a blank. Actually the tab can be used anywhere a space would be appropriate. Upon input of the source text, the tab is expanded to a modulo 8 column. The use of a tab within an ASCII string is also permissible but will produce results which may not be what was desired. For example, if "t" is the tab character, the string .BYT 'ABtCD' will be translated to .BYT 'AB CD', 'AB CD', etc. as a function of the source column, including any line number, at which the tab character appeared. If a tab is desired as part of a character string, the correct way is to use the syntax

```
        .BYT    'AB',$9,'CD'
```

b. Supports the use of < and >. as prefixes to an operand to signify the high and low bytes of the 16 bit value of the operand. The following examples may help in understanding this feature.

```
        LABEL   =       $FA23
23              .BYT    <LABEL
FA              .BYT    >LABEL
```

c. Supports use of ASCI characters as operands in an expression. Only single characters are allowed and the only allowable form is 'x' where x is the desired ASCII character. Confusion can arise since MOS Technology standards allow ASCII characters used as the only operands of an immediate instruction to drop the closing quote. Thus the following formats are legal:

```
        .BYT    'A'
        .BYT    'A'+3
        LDA     #'A
        LDA     #'A'
```

```
          LDA     #'A'+3
```

while the following formats are illegal:

```
          .BYT    'A
          .byt    "A
          LDA     #'A+3
```

d. ASM.COM accepts both upper and lower case input and thus the following two lines would result in the same code:

```
          lda     #45
          LDA     #45
```

Lowercase to uppercase conversion is not done within a string and thus the following `.BYT` directives would result in different code:

```
          .byt    'ABC'
          .byt    'abc'
```

NOTE

In this manual parentheses will be used to enclose the names of single characters, e.g., (cr) for carriage return. Parentheses are also used to enclose entire fields. The use will be evident from the context.

SECTION 2 EXECUTION

2.1 COMMON DRIVE

If the .ASM, .KIM, and .PRN files are to be located on the same drive the assembler is executed by use of a CCM command line similar to the examples which follow (user inputs are underlined):

CCM INPUT	ACTION
`A>asm source`	ASM.COM is loaded from the default drive (A). Assembles file SOURCE.ASM on the default drive. Object code file SOURCE.KIM is written to the default drive and the listing file, SOURCE.PRN is written to the default drive.
`A>asm b:source`	Same as first example except that SOURCE.ASM is read from drive B and SOURCE.KIM and SOURCE.PRN are written to drive B.
`A>b:asm source`	Same as first example except that ASM.COM is loaded from drive B. SOURCE.ASM, SOURCE.KIM, and SOURCE.PRN remain on drive A.
`A>b:asm b:source`	In this example ASM.COM, SOURCE.ASM, SOURCE.KIM, and SOURCE.PRN are all on drive B.

The assembler command sequence can be generalized as follows:

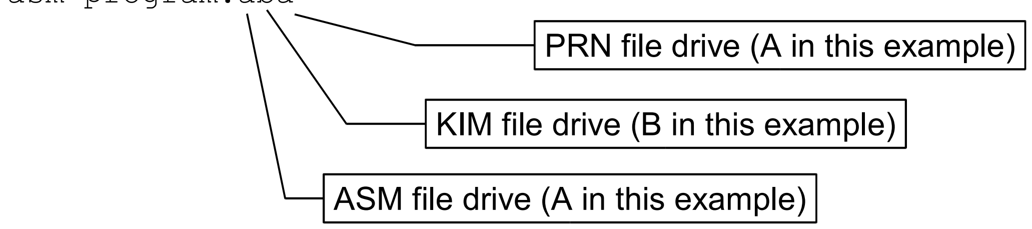
```
(drive:)asm (drive:)name
```

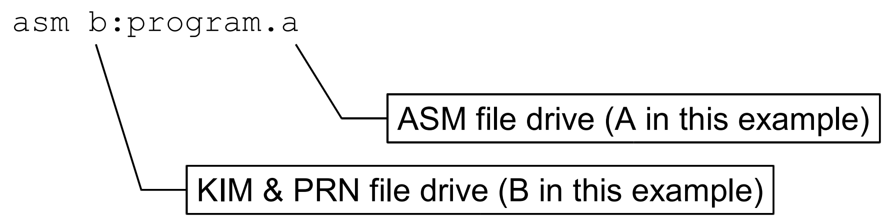
where the (drive:) terms represent optional items as discussed above.

2.2 INDEPENDENT DRIVES

ASM.COM also allows independent selection of the drive for the .ASM file, the drive for the .KIM file, and the drive for the .PRN file. The command lines shown above required the drive for those files to be the same and to be either the default or the drive specified by a prefix to the file name. The following command lines illustrate how to alter that drive assignment:

```
asm program.aba
```


PRN file drive (A in this example)
KIM file drive (B in this example)
ASM file drive (A in this example)


asm b:program.a
ASM file drive (A in this example)
KIM & PRN file drive (B in this example)

A logical description of the logic used by ASM to determine the drive for each file is as follows:

```
DRIVE(.ASM)=DEFAULT
DRIVE(.KIM)=DEFAULT
DRIVE(.PRN)=DEFAULT
IF PREFIX
  THEN
    DRIVE(.ASM)=PREFIX
    DRIVE(.KIM)=PREFIX
    DRIVE(.PRN)=PREFIX
IF (FIRST CHAR AFTER .  = BLANK)
  THEN
    EXIT
  ELSE
    DRIVE(.ASM)=FIRST CHAR AFTER .
IF (SECOND CHAR AFTER .  = BLANK)
  THEN
    EXIT
  ELSE
    DRIVE(.KIM)=SECOND CHAR AFTER .
IF (THIRD CHAR AFTER .  = BLANK)
  THEN
    EXIT
  ELSE
    DRIVE(.PRN)=THIRD CHAR AFTER .
EXIT
```

2.3 PRN REDIRECTION OR DELETION

If it is desired to send the .PRN file only to the console then the command line format shown in Section 2.2 can be used except that the third position after the "." is entered as a X. Thus the command line

```
asm program.abx
```

would read the .ASM file from drive A, write the .KIM file to drive B, and send the .PRN file to the console. If it is desired to completely eliminate the .PRN file then the command line format shown in section 2.2 can be used except that the third position after the "." is entered as a Z. Thus the command line

```
   asm program.abz
```

would function like the previous example but would not produce a .PRN file.

2.4 KIM DELETION

If it is desired to completely eliminate the .KIM file then the command line format shown in Section 2.2 can be used except that the second position after the "." is entered as a Z. Thus the command line

```
   asm program.azx
```

would read the .ASM file from drive A, produce no .KIM file, and would send the .PRN file to the console.

SECTION 3 OPERATION

As discussed in Section 1.2, the assembler is compatible with the MOS Technology standards with respect to operands, opcodes, labels and comments. It does not provide the same set of assembler directives as defined in either the Cross Assembler or the Microcomputer Family KIM Assembler Manual. The directives provided and each element of the syntax will be discussed in detail in the following sections.

3.1 LINE FORMAT

The general format for an input line to the assembler is

```
   (line number) (label) (opcode) (operands) (comments)(cr)(lf)
```

The parentheses indicate that the applicable field is not absolutely necessary in all lines. The only elements which must appear in each line are the (cr) and (lf) characters at the end of the line. Each field, if present, must be separated from adjacent fields by a (blank) or a (tab). If certain fields are present, it may then be necessary to include other fields. In particular, many of the legal opcodes require an operand and thus the operand field must be present when those opcodes are used. The comment field is always optional and the label field is only needed when the line must be referenced by another line. Lines are limited to a length of 80 characters in addition to the (cr) and (lf).

3.1.1 LINE NUMBER

The line number field is always optional and even if present will be ignored by the assembler as long as it consists only of contiguous decimal digits.

3.1.2 LABEL

A label is a string of alphanumeric characters which must begin with an uppercase or lowercase alphabetic character (A through Z or a through z) and is limited in length to a predetermined value. The standard length is 16 characters, however, as explained in the Appendix, the assembler can be configured to limit label lengths to other values.

The characters A, S, P, X and Y (uppercase or lowercase) are reserved for special uses by the assembler and may not be used as labels nor may any of the 56 legal opcodes. Also, labels may not contain any characters other than alphabetic characters or the decimal digits 0 through 9. Labels need not begin in the first column of each line, but if used must begin with the first non-blank or non-tab character in the line. The following are examples of legal and illegal labels:

LEGAL	ILLEGAL	REASON
LOAD	X	reserved
A1	AND	opcode
opcode	2	first not alpha
TWO1T	B$	non-alphanumeric
theand	A1234567890123456	too long
ANDORA		

3.1.3 OPCODE
The opcode field may contain one of the 56 legal opcodes or one of the legal assembler directives. The legal opcodes are listed in Table 1 and the legal assembler directives are listed in Table 2. The user is referred to the 6502 Programming Manual for the details about each opcode and its allowable addressing modes or operands. Except for the equate (=), the assembler directives are distinguished from the opcodes by use of a leading period. While the assembler directives are listed in Table 2 using the "full" name, only the first three characters are significant and are actually used by the assembler when matches are attempted between the input and the legal assembler directives. Thus, the mnemonics shown in the minimum name column are all that need be used.

Neither the `.END` nor `.PAG` directive can have an operand field. The `=`, `.BYT`, `.WOR`, and `.OPT` directives must have one or more operands. If more than one operand is used, then they are separated by commas. The value of each operand for the .BYT directive must be between $00 and $FF (inclusive) while the value of each operand for the = and .WOR directives must be between $0000 and $FFFF (inclusive). The operands for the .OPT directive are not operands in the usual sense but are parameters which are used to set the various flags maintained by the assembler. Table 3 shows the full set of parameters which may be used. As was the case for the directives only the first three characters need be used. The parameters may be used in any order and any number of them may be used as long as the line length restrictions are not violated. If complementary parameters (e.g., SYM and NOS) are used in a single line, the last (rightmost) parameter used will be in effect upon printing of that line and processing of the next line. If no .OPT directive is used, the conditions marked as default will be in effect for the assembly.

3.1.4 OPERAND
Except for the .OPT assembler directive parameters discussed in Section 3.1.1.3, operands must be legal expressions. The only exceptions to that rule are a) those operands which can address the accumulator (e.g., ASL A) in which case an A is used as the operand, b) those opcodes which have no operand (e.g., CLC), and c) those

opcodes which, because of the addressing mode used (e.g., LDA #VALUE or LDA (VALUE),Y) have a complex operand format. In the latter case a portion of the operand field must still be a legal expression. For the examples shown the term "VALUE" must be a legal expression as defined in Section 3.1.2. The special formats required for the various addressing modes are summarized in Table 4. In all cases (expr) must be greater than or equal to zero. For further detail see the 6502 Programming Manual.

TABLE 1. OP CODES

ADC Add Operand with Carry to Accumulator	AND And Operand with Accumulator	ASL Shift Operand Left One Bit	BCC Branch on Carry Clear
BCS Branch on Carry Set	BEQ Branch on Zero Result	BIT Test Bits in Memory with Accumulator	BMI Branch on Results Minus
BNE Branch on Result not Zero	BPL Branch on Result Plus	BRK Break	BVC Branch on Overflow Clear
BVS Branch on Overflow Set	CLC Clear Carry Flag	CLD Clear Decimal Mode	CLI Clear Interrupt Disable Bit
CLV Clear Overflow Flag	CMP Compare Accumulator and Operand	CPX Compare Index X and Operand	CPY Compare Index Y and Operand
DEC Decrement Operand by One	DEX Decrement Index X by One	DEY Decrement Index Y by One	EOR Exclusive-or Operand with Accumulator
INC Increment Operand by One	INX Increment X by One	INY Increment Y by One	JMP Jump to New Location
JSR Jump to New Location Saving Return Address	LDA Load Accumulator	LDX Load Index X	LDY Load Index Y
LSR Shift Operand One Bit Right	NOP No Operation	ORA Or Operand with Accumulator	PHA Push Accumulator on Stack
PHP Push Processor Status on Stack	PLA Pull Accumulator from Stack	PLP Pull Processor Status from Stack	ROL Rotate Operand One Bit Left
ROR Rotate Operand One Bit Right	RTI Return From Interrupt	RTS Return from Subroutine	SBC Subtract Operand and Carry from Accumulator
SEC Set Carry Flag	SED Set Decimal Mode	SEI Set Interrupt Disable Status	STA Store Accumulator
STX Store Index X	STY Store Index Y	TAX Transfer Accumulator to Index X	TAY Transfer Accumulator to Index Y
TSX Transfer Stack Register to Index X	TXA Transfer Index X to Accumulator	TXS Transfer Index X to Stack Register	TYA Transfer Index Y to Accumulator

TABLE 2. ASSEMBLER DIRECTIVES

FULL NAME	MINIMUM NAME	MEANING
`.END`	`.END`	End Assembly
`.BYTE`	`.BYT`	Define Byte Values
`.WORD`	`.WOR`	Define Word Values
`.OPTION`	`.OPT`	Set Assembler Options
`=`	`=`	Equate Label to Expression
`.PAGE`	`.PAG`	Insert Formfeed in List Output

NOTE

If the NOLIST option is set then the .PAGE directive will not cause a formfeed to be inserted in the .PRN file. The Formfeed character used by .PAGE is determined by the Formfeed character in the console definition block in SIM.

TABLE 3. OPTION PARAMETERS

FULL NAME	MINIMUM NAME	MEANING
`*SYMBOLS`	`SYM`	Print Symbol Table
`NOSYMBOLS`	`NOS`	Do Not Print Symbol Table
`*ERRORS`	`ERR`	If NOLIST Print Errors
`NOERRORS`	`NOE`	If NOLIST Do Not Print Errors
`*LIST`	`LIS`	Write Listing
`NOLIST`	`NOL`	Do Not Write Listing
`GENERATE`	`GEN`	Print Strings
`*NOGENERATE`	`NOG`	Do Not Print Strings
`*KIM`	`KIM`	Generate KIM file
`NOKIM`	`NOK`	Do Not Generate KIM File

*Default Values

TABLE 4. OPERAND FORMATS

MODE	FORMAT	COMMENTS
Accumulator	A	
Absolute	(expr)	
Zero Page	(expr)	(expr) must be < 256
Implied	no operand	
Immediate	(expr)	(expr) must be < 256
(Indirect, X)	((expr), X)	(expr) must be < 255
(Indirect), Y	((expr)), Y	(expr) must be < 255
Zero Page, X	(expr), X	(expr) must be < 256
Absolute, X	(expr), X	
Absolute, Y	(expr), Y	
Relative	(expr)	(expr)-* must be > -128 and < 128
Indirect	((expr))	
Zero Page, Y	(expr), Y	(expr) must be 256

NOTE
(expr) is an expression meeting the requirement of Section 3.1.2.

3.1.5 COMMENT

The comment field may contain any arbitrary text (upper or lower case) and is used to explain the programmer's intent. The comment field may begin at any time if preceded by a (;). If the preceding fields (i.e., opcode and operand) are present then the comment field need not begin with a (;). The following examples show legal and illegal

LEGAL	ILLEGAL
`;comment`	`.byt comment`
`.BYT 3 comment`	`LDA comment`

comments.

3.2 EXPRESSIONS
The key to the effective use of an assembler is an understanding of the power available in formulating expressions. The following sections define the terms which make up expressions and how they may be combined.

3.2.1 CONSTANTS
The assembler recognizes both numeric and non-numeric constants. The numeric constants may be of the following types:

Binary	`%bbbbbbbbbbbbbbbb`
Octal	`@oooooo`
Decimal	`nnnnn`
Hexadecimal	`$hhhh`

b, o, n, or h represent single binary (0, 1), octal (0 through 7), decimal (0 through 9), or hexadecimal (0 through 9, A through F, and a through f) digits respectively. As few as one digit may be used or as many as required to represent values up to the limit (65535 or its equivalent) imposed by use of 16 bits for each value may be used.

Non-numeric constants may be single ASCII characters enclosed in quotes such as

```
'A'
'z'
'*'
```

or may be strings enclosed in quotes (e.g., `'ABC*$' or “Error Message”`). While single character constants can be used in any expression, strings must be used by themselves and only as an operand of a .BYT assembler directive. If a single ASCII character is used as the operand in an immediate addressing mode the closing quote is not required. If a quote is desired as a member of a string, then a pair is used. Thus

```
'a quote '' is needed'
```

would be loaded as the string

```
a quote ' is needed
```

3.2.2 VARIABLES
A variable is nothing more than a label. It must be defined somewhere in the program. When encountered during the assembly the symbol is replaced by its value. The only variable which has special meaning is the character `*`. This character represents the value of the program counter and if used in an expression will be replaced with the value the program counter had prior to the start of the line in which it is used. As the

character * is also used to signify multiplication operation, care must be exercised to ensure that the correct result is obtained.

3.2.3 OPERATIONS
The available operations include the following:

UNARY
- −
- >
- <

BINARY
- −
- +
- * (multiplication)
- / (division)

The unary operators < and > must be applied only to labels and yield a value equal to the most significant and least significant bytes respectively of the label. The binary operators have no precedence; hence, all expressions are evaluated in strict left to right order. If overflow results from an addition operation, the expression will be considered to be erroneous as will a final result which is negative. Overflow in multiplication is ignored as is the remainder resulting from a division operation. The following are examples of valid and invalid expressions under these rules.

EXPRESSION	VALUE	ERROR
`$FFF*$FFF`	`$E001`	No
`$F7FF+$1000`	`$07FF`	Yes (Overflow)
`10 - 15`	`$FFFB`	Yes (Negative)
`$15/$33`	`$0000`	No
`<LABEL+1`	`$0040`	No (for `LABEL = $173F`)
`**2+1`	`$0403`	No (for `* = $201`)
`*-$300`	`$FFO1`	Yes (Negative) (for `* = $201`)

3.3 CAUTIONS
Although it is impossible to protect the programmer from all possible errors some are discussed in the following sections.

3.3.1 PAGE ZERO DEFINITIONS
Perhaps the easiest error to fall prey to in using the assembler is the failure to define page zero addresses prior to their use in an operand field. This error will usually (not necessarily always) give a "Label Previously Defined" error at some line following the line which had the forward reference to page zero. The reason is simple - if during the first pass the assembler encounters an operand field containing an undefined label, it will allocate three bytes for the opcode and operand. If the label is later defined as a

page zero address the assembler will allocate only two bytes during the second pass. Subsequent labels will now be off by one and will cause the error message to be generated in all following lines in which labels are defined in terms of the current program counter. The solution is simple - ALWAYS DEFINE PAGE ZERO LABELS AT THE BEGINNING OF THE PROGRAM!

3.3.2 PAGE ZERO USAGE

While a program can be assembled for any location, the manner in which page zero usage is defined is important. For programs which are to be executed in the DOS/65 TEA, either of the following two approaches will work (assuming that TEA is defined in the code file!)

Example 1

```
;first variable definition follows and others
*       =       0
DATA    *=      *+1
;other variable definitions follow as needed
VAR     *=      *+1
;the next line is the last variable definition
LAST    *=      *+1
;now set the program counter for the start of code
*       =       TEA
;executable code follows
```

Example 2:

```
;first variable definition follows
DATA    =       2
;other variable definitions follow as needed
VAR     =       DATA+1 (or =3)
;the next line is the last variable definition
LAST    =       NXTLST+1 (or =location)
;now set the program counter for the start of code
*       =       TEA
;executable code follows
```

The following will assemble correctly but the .KIM file cannot be loaded by MAKECOM.COM.

```
*       =       0
;first variable definition follows and others
DATA    .BYT    0
VAR     .BYT    38
LAST    .BYT    $FF
;now set the program counter for the start of code
*       =       TEA
```

```
   ;executable code follows
```

This last approach will not work since the `.BYT` directive not only defines the location of each label, it also generates data (code) to be loaded at the defined location. `MAKECOM.COM` will flag that as an error since it cannot create `.COM` files for any location in page 0 or 1. The first two approaches will work since neither actually generates code.

3.3.3 SYMBOL LENGTH

The standard assembler is set to use labels of from one to sixteen characters in length. Before discussing how to configure the assembler for other label lengths, the effects of symbol length must be briefly discussed. Each symbol encountered during the assembly is stored in a table. Each entry requires two bytes for the symbol value and as many bytes as are specified as the maximum to hold the symbol itself.

If the assembler were configured for six character labels, a 16K system with the TEA beginning at $200 could hold the symbol table, the assembler and the necessary portions of DOS/65 even if as many as 500 symbols were used. If the allowable symbol length is doubled to 12, then only about 300 symbols can be stored. While larger systems will have no problem with long symbol length and large numbers of symbols, such is not the case for 16K systems.

3.3.4 ERRORS

If errors are encountered during the assembly, do not attempt to use the KIM file as the data contents and address structure will probably be in error. The KIM file might be loadable either with MAKECOM.COM or DEBUG.COM but the results of executing the erroneous program are unpredictable and potentially disastrous. The error codes listed in the appendix are self-explanatory.

APPENDIX A - ERROR MESSAGES

NUMBER	MESSAGE
1	Undefined Symbol
2	Label Previously Defined
3	Illegal or Missing Opcode
4	Address Not Valid
5	Accumulator Mode Not Allowed
6	Forward Reference in .BYT or .WOR
7	Ran Off End of Line
8	Label Does Not Begin with Alphabetic Character
9	Label Too Long
10	Label or Opcode Contains Non-Alphanumeric
11	Forward Reference in Equate or ORG
12	Invalid Index - Must be X or Y
13	Invalid Expression
14	Undefined Assembler Directive
15	Invalid Operand for Page Zero Mode
16	Invalid Operand for Absolute Mode
17	Relative Branch Out of Range
18	Illegal Operand Type for this Instruction
19	Out of Bounds on Indirect Addressing
20	A, X, Y, S, and P are Reserved Labels
21	Program Counter Negative - Reset to O
22	Invalid Character - Expecting "=" for ORG
23	Source Line Too Long
24	Divide by Zero in Expression
25	Symbol Table Overflow

APPENDIX B - ASSEMBLER VERSION DESIGNATION

Assembler version designation is coded using the following format:

x.yy-z

x is a number greater than zero which designates the basic version of the system.

yy is a number greater than or equal to zero which designates the revision number of the system.

z is a single letter which designates the TEA location at which the program is designed to be run. Currently assigned designations are

DESIGNATOR	**TEA**
S	$200
P	$400
A	$800
T	$1000
R	$1400
K	$2000

APPENDIX C – SYMBOL LENGTH

ASM.COM V2.1X is normally set to use labels up to 16 characters in length. That is a reasonable maximum that allows great flexibility and clarity in naming labels. However it has two drawbacks in that it uses more memory for the label table and is somewhat slower than a configuration that used a smaller limit, e.g., six characters.

Since the ASM source code is provided it is easy to change the maximum label length and re-assemble the software. However, that is a time consuming operation and there is an easier and faster way.

At TEA+3 in the TEA.COM file is a single byte defining the maximum label length. Normally that is set to 16 as shown in the source code. If it is changed to a smaller value, e.g., six, then ASM.COM will limit labels to that new value.

The easiest and fastest way to change that is to follow the following procedure. In this case the C64 version is assumed with TEA set to $800. User inputs are in BOLD & UNDERLINE.

```
A>debug asm.com
CAN NOT SET IRQ/BRK VECTOR
NEXT ADDRESS=2180
-d803,803
0803 10 .
-s803,6
-d803,803
0803 06 .
(control-c)
A>save 26 asm6.com
```

This will save a new copy of ASM.COM as ASM6.COM with the maximum label length set to six (6).