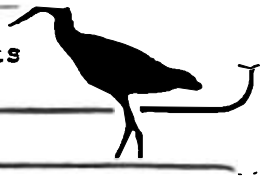
6502 CROSS-ASSEMBLER

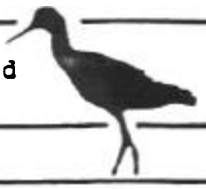
USER'S MANUAL

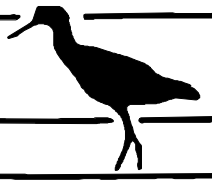
XASM65 Assembler Version 1.96
Manual Revision 1.3 (17-Mar-81)

AVOCET SYSTEMS, INC.
804 SOUTH STATE ST.
DOVER, DELAWARE 19901

(302) 734-0151







1.0 Introduction

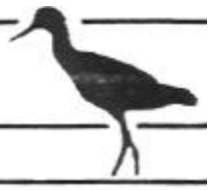
XASM65 is a cross-assembler designed to run on 8080 and Z-80 based microcomputers, under the CP/M* operating system. It generates machine code for the 6502 microprocessor, accepting standard 6502 mnemonics and syntax.

XASM65 accepts its input from a CP/M text file, and generates as output an object code file, an assembly listing, and an alphabetized listing of all symbols defined in the assembly. The object file may be in Intel HEX format or in KIM format or may be omitted altogether, at the user's discretion. The listings are normally sent to the system's LST: device, but may be directed to the console or to a disk file.

The assembler features a variety of pseudo-operations. In addition to the usual storage-definition instructions, there are facilities for conditional assembly, for control of listing format, and for including multiple source files in an assembly. For compatibility with other assemblers, a feature is offered whereby the user may define new mnemonics as synonyms for opcodes and pseudo-ops, thus minimizing the need to re-edit existing source programs.

This document is intended as a reference manual for the experienced user. As such, it assumes familiarity with the 6502 instruction set, with the operation of CP/M, and with assemblers in general. The user who needs tutorial assistance in these areas may wish to consult one of the many introductory texts on 6502 programming, as well as the manuals which accompany CP/M.

*Trademark of Digital Research.



2.0 Running the Cross-Assembler

2.1 XASM65 Command Strings

The 6502 cross-assembler is invoked by typing

XASM65 {d:}filename{.ext} {d:} {switches}

The d:'s are optional drive specifiers (eg. A:, B:, etc.). The file extension (.ext) is also optional; if omitted, it defaults to .ASM. The second d: specifies a disk drive for the output files (object and listing); if it is omitted, these files are placed on the currently-logged drive. The switches are (optional) single characters which control the output of the assembler, as described below.

2.2 XASM65 Switches

The switches following the filename, if supplied, control various options as follows:

L Listing only; ie. no object file.

X No assembly listing.

Y No symbol-table listing

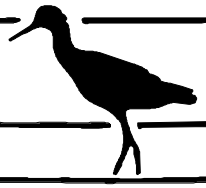
O Object only; equivalent to XY.

C Send listing, if any, to console instead of LST: device.

D Send listing, if any, to disk file with same name as source file and extension "PRN".

K Use KIM format for the object file, and give it extension KIM. Otherwise, object file is in Intel HEX format and has extension HEX. (See Appendix B).

N Suppress pagination of assembly listing; ie. no page headings or page ejects.



The switches may be used in combination; for example,

XASM65 GRINCH YCK

will suppress the symbol-table listing, send the assembly listing to the console, and use KIM object format. Note that errors are always listed, even if listing is turned off. Thus,

XASM65 GRINCH LXYC

will list errors only, on the console.

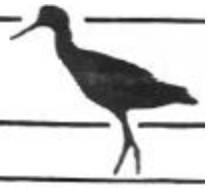
2.3 Summary of Defaults

The default output options, in the absence of their respective specifiers, are as follows:

The source file is assumed to have extension "ASM" and to reside on the currently-logged drive.

An object file is generated. It is in Intel HEX format and is placed on the currently-logged drive.

An assembly listing and a symbol-table listing are generated; they are sent to the CP/M LST: device.



3.0 Syntax of Assembler Source Files

3.1 Statements

labels can only have certain characters in them or assembler will hang up

Source files consist of a sequence of statements of one of the forms:

{label:} operator {arguments} {;comment}

symbol operator {arguments} {;comment}

{;comment}

If a label or a leading symbol is present, it must begin in column 1. Labels must be followed by a colon. Blank lines are treated as comments. Elements of a statement may be separated by blanks or tabs.

3.2 Symbols

Symbols may be up to 8 characters in length, and may include any of the following characters:

A..Z 0..9 $

The first character of a symbol must be a letter or a period.

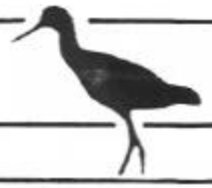
3.3 Numeric Constants

Numeric constants consist of a sequence of digits, optionally preceeded or followed by a radix specifier. The first character must be either a leading radix specifier or a decimal digit (0..9). The default radix is ten. For compatiblity with existing assemblers, other bases may be denoted by either a leading or a trailing specifier. The leading radix specifiers are:

% (binary) @ (octal) $ (hex)

The trailing specifiers are:

B (binary) Q (octal) H (hex)



Thus, for example, the following are equivalent:

@1111111 1111111B @177 177Q $7F 7FH

3.4 Character Constants

Character constants may be used wherever a numeric value is allowed. A character constant consists of one or two characters enclosed in single or double quotes (' or "). The single quote may be used as a character between double quotes, and vice-versa.

Thus, the following are equivalent:

'A' "A" and 41H

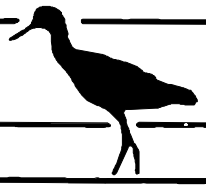
'AB' "AB" and 4142H

3.5 Location-Counter Reference

The character "$" may be used as an element in expressions. Its value is the value of the location counter at the beginning of the current statement.

3.6 Expressions

Arithmetic expressions are composed of symbols, numeric constants, character constants, and operators. All operators except +, -, *, and / must be separated from their operands by at least one space. Symbols which are operators are reserved, and may not be redefined as user symbols.
A description of the operators follows:



3.61 Arithmetic Operators

These treat their operands as 16-bit unsigned quantities, and return 16-bit results. No overflow checking is performed.

+ and -	Sum and Difference. Operands and results may be thought of as unsigned or as twos-complement quantities.
unary +	+x is defined as 0+x
unary -	-x is defined as 0-x
* /	Product and Quotient (unsigned)
MOD	Remainder; x MOD y gives the remainder of x/y

3.62 Shift Operators

SHL	Binary left shift. x SHL y yields x shifted left y places (ie. x multiplied by 2^y).
SHR	Binary right shift, logical. x SHR y yields x shifted right y places (ie. x divided by 2^y).

If the right argument is negative, then the direction of the shift is reversed.

3.63 Byte-Extraction Operators

HIGH	Returns the value of the most significant byte of its argument.
LOW	Returns the value of the least significant byte of its argument

These are unary operators, taking an argument on the right. For example: HIGH 1122H is 11H, and LOW 1122H is 22H.

3.64 Boolean Operators

NOT — Unary logical negation. Complements all the bits in its argument.

AND — Logical product; ie. each bit of the result is obtained by ANDing together the corresponding bits in the arguments.

OR — Logical sum.

XOR — Exclusive-OR.

These are all bitwise operators; that is, the same operation is performed on each operand bit position, with no carry from one bit position to the next.

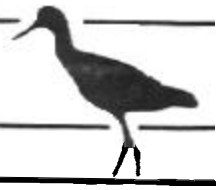
For example:
```
NOT 0 is 0FFFFH
101B AND 010B is 0
101B OR  010B is 111B
101B XOR 010B is 111B
101B XOR 100B is 001B
```

3.65 Relational Operators

These perform unsigned 16-bit comparisons of their operands, returning 1 for TRUE and 0 for FALSE.

For comparison x R y, where R is a relational operator, the results are as follows:

EQ	TRUE iff x and y are equal
NE	TRUE iff x and y are not equal
LE	TRUE iff x is less than or equal to y
LT	TRUE iff x is strictly less than y
GE	TRUE iff x is greater than or equal to y
GT	TRUE iff x is strictly greater than y.



3.66 Evaluation of Expressions

Parentheses may be used to specify the order of evaluation of subexpressions. In the absence of parentheses, this order is determined by operator precedence; higher-precedence operators are evaluated first. In the case of operators with equal precedence, evaluation proceeds from left to right. The operators are listed below in groups according to precedence. Operators in the same horizontal group have the same precedence:

unary +, unary - (HIGHEST PRECEDENCE)

HIGH LOW

/ MOD SHR SHL

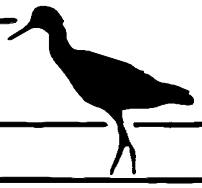
\+ -

EQ NE LT LE GT GE

NOT

AND

OR XOR (LOWEST PRECEDENCE)



4.0 Addressing Modes

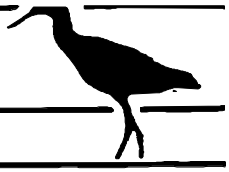
4.1 Syntax

The addressing modes in 6502 instructions are denoted by their standard forms:

Immediate:	#expression	
Direct:	expression	
Zero-Page:	expression	(expression < 256)
Indexed:	expression,X expression,Y	
Zero-Page Indexed:	expression,X	(expression < 256)
Indirect:	(expression,X) (expression),Y	

4.2 Selection of Zero-Page Modes

The assembler automatically selects the Zero-Page and Zero-Page Indexed modes according to the magnitude of the operand; these modes are used (if available for the specified instruction) whenever the operand has a value less than 256. However, since XASM65 is a two-pass assembler, this selection must take place during the first assembly pass. Therefore, if the value of the operand expression is not known during Pass 1 (ie. if the expression contains a forward reference) then the appropriate absolute addressing mode is always selected. This would typically occur if the expression contains a symbol which is not defined until later in the source program.



5.0 Pseudo-Operations

5.1 Storage Definition

DB arg[,arg...]

Define Bytes. Each arg may be either an expression or a string. Expressions must evaluate to 8-bit values (high byte either 0 or 255). Strings may be delimited by single or double quotes, as for character constants.

For each expression, a single byte of storage is reserved, initialized to the low byte of the expression's value. For each string, the characters of the string are stored in sequential reserved bytes.

If a compound expression beginning with a character constant is used in a DB, then the expression must be enclosed in parentheses to keep it from being incorrectly parsed as a string. For example,

```
DB ('A'+1)
```

will give the expected result, while

```
DB 'A'+1
```

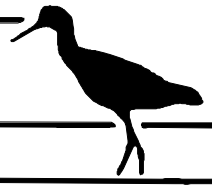
would be in error.

DW expression[,expression...]

Each expression reserves one word (2 bytes), initialized to the value of the expression. Each word will be placed in memory with its low-order byte first.

DS expression

Reserves n bytes, where n is the value of the expression. The bytes are not initialized.



5.2 ORG and END

ORG expression

Set program origin. This statement should precede the first code-generating statement in the source file. It sets the program counter initial value to the value of the expression, thus setting the location of code which follows. Additional ORG statements may be used to generate program segments which will load at different locations.

END [expression]

The last statement of the source file must be an END statement. An optional argument is allowed; if this is supplied, its value becomes the start address specified in the last record of the HEX object file.

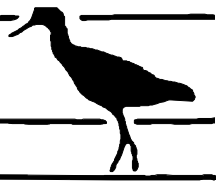
Symbol Definition

EQU and SET

These take the forms:

```
symbol    EQU  expression
symbol    SET  expression
```

They cause the symbol to be defined and given the value of the argument expression. Symbols defined with EQU serve as symbolic constants, and may not have their values changed. Symbols defined with SET are treated as variables; their values may be changed by additional SET's.



5.4 Conditional Assembly

IF, ELSE, and ENDIF

The construct

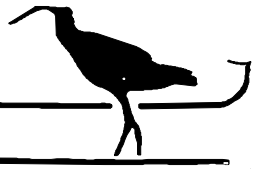
```
IF expression
 statement
 .
 .
 .
 ENDIF
```

behaves as follows: If the value of the expression is non-zero, then the statements between the IF and the ENDIF are assembled. Otherwise, these statements are ignored. Similarly,

```
IF expression
statement


.
ELSE
 statement
 .
 .
 .
 ENDIF
```

causes the first sequence of statements to be processed if the expression is TRUE (non-zero), and the second sequence to be processed otherwise.

Conditionals may be nested to a depth of 10. The value of the expression in the IF statement must be known in Pass 1.



5.5 Listing Control

PAGE (expression)

If the argument is omitted, then this causes an immediate page eject (ie. a skip to the top of the next page). If an argument is supplied and has value n, then an eject occurs only if there are less than n lines remaining on the current page.

WIDTH expression

Sets the assumed width of the listing page. The value of the argument may be between 32 and 132, and defaults to 132. WIDTH statements are not shown in the assembly listing.

PGLEN expression

Sets the number of lines which will be printed on each page of the listing. Note that the page heading takes up 7 of these lines. The value of the argument may be between 8 and 255, and defaults to 58. PGLEN statements are not shown in the assembly listing.

TITLE dtextd

Causes the specified text to become the listing page title, beginning with the next page header printed. The delimiter d may be any printing character.

If no TITLE statement is used, then XASM65 supplies a default title consisting of the text "SOURCE FILE NAME: " followed by the name and extension of the input file.

SBTTL dtextd

Just like TITLE, but sets the listing page subtitle, which is printed on the line after the title line.

LIST and NOLIST

These allow selective listing of portions of a program. NOLIST turns off the assembly listing, and LIST turns it back on. If listing has been turned off with NOLIST, then the next LIST encountered will begin a new page. Command-line switches which disable listing (ie. X and O) will take precedence over LIST. NOLIST does not turn off listing of the symbol table.

5.6 External Source Files

INCLUD d:name.ext

This pseudo-op causes the specified file to be included as if it were present at this point in the source file. INCLUD's may not be nested; that is, the file read by INCLUD may not contain another INCLUD statement. The file must end with an END statement.

5.7 Operator Synonyms

The statement

```
symbol    OPSYN    operator
```

causes the given symbol to be defined as a synonym for the operator or pseudo-op specified as argument. This is particularly useful when assembling source files written for another assembler. For example, if a program uses .BYTE instead of DB, it could be correctly assembled by including the statement

```
.BYTE     OPSYN    DB
```

6.0 Errors

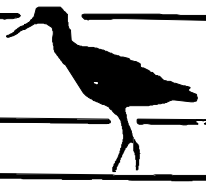
Fatal errors result in the printing of an error message on the console and immediate return to CP/M. Fatal errors may be caused by missing source files, inadequate disk space for output files, or overflow of the symbol table or various internal stacks.

Non-fatal errors are flagged with a character in the first column of the assembly listing. Lines containing errors are always listed, even if listing is turned off. A count is maintained of lines containing errors; if the count is non-zero at the end of the assembly, it is printed on the console in the message:

***** nnn LINES CONTAINED ERRORS *****

Only one error is listed per line; hence, if a line contains multiple errors some may not be caught until successive assembler runs.

The fatal error messages and non-fatal error flags are described in Appendix A.



7.0 Format of Listings

7.1 Page Headings

All listings begin with a heading consisting of seven lines:

```
(blank line)
(blank line)
(assembler name and version)
(blank line)
(title and page number)
(subtitle)
(blank line)
```

If no title is supplied in the source program, then the assembler provides a default title consisting of the message

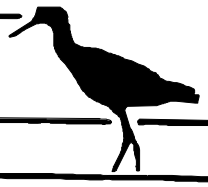
```
SOURCE FILE NAME:
```

followed by the name of the input file in the form name.ext. The page number is listed at the right-hand end of this same line, always within the specified page width.

If no subtitle is supplied, the subtitle line is left blank. Both title and subtitle will be truncated, if necessary, to satisfy page width constraints.

7.2 Line Headings

Each code-generating line of the listing begins with the error flag (blank if no error) and the 4-digit hexadecimal value of the location counter as of the start of the line. This is followed by up to four bytes of generated code, also in hexadecimal with two digits per byte. Statements which generate more than 4 bytes will be assembled correctly, but only the first four bytes are listed. Lines which do not generate code but which evaluate an operand (such as EQU) list the operand value in their headers, in place of the location counter.



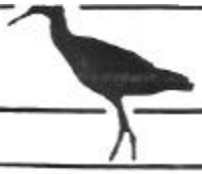
7.3 Symbol-Table Listing

The symbol-table listing shows all symbols defined in the current assembly, with their hexadecimal values. Only user-defined symbols are listed. Symbols are in vertical columns, sorted alphabetically according to the ASCII collating sequence. The number of columns is adjusted automatically to fit in the specified page width. All pages of the symbol-table listing are automatically subtitled:

```
---- SYMBOL TABLE ----
```

Because the sorting scheme "alphabetizes" symbol names even if they end in numeric characters, the listing order may not be what you expect. For example, a typical sequence of symbols might appear as follows:

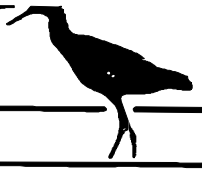
```
SYM19
SYM2
SYM20
SYM21
SYM3
SYM4
```



APPENDIX A. Error Messages and Flags

Non-Fatal Error Flags

C	Conditional Err	Unmatched IF, ELSE, or ENDIF; or conditionals nested too deep.
I	INCLUD Err	File not found, or nested INCLUD's.
L	Label Err	Label too long. (more than 8 chars)
M	Multiple Defn	Symbol already defined.
O	Operator Err	Undefined or illegal operator.
P	Phase Err	Symbol had different value on Pass 2 than on Pass 1.
R	Range Err	Argument out of bounds, branch out of range, or illegal register/port number.
S	Syntax Err	Ill-formed argument or expression.
U	Undefined	Undefined symbol(s) in operand field.
X	Index Err	Illegal or unrecognized index specifier.

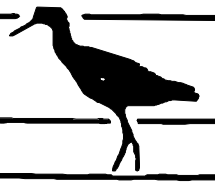


Fatal Error Messages

SOURCE FILE NOT FOUND	The specified source file doesn't exist.
UNABLE TO CREATE OUTPUT FILE	The directory is full on the disk specifed for output.
OUTPUT FILE WRITE ERROR	The output disk is full.
EVALUATION STACK FULL	An arithmetic expression was encountered which had too many levels of parentheses or of precedence nesting.
SYMBOL TABLE FULL	Not enough memory remains to create a table entry for a symbol being defined.

Non-Fatal Error Messages

NO ROOM FOR SYMBOL-TABLE SORT	Not enough memory is available to sort the symbol table. Symbol-table listing is therefore omitted.
END STATEMENT MISSING	End-of-File was reached in the source file, or in an include file, without encountering an END statement. The assembler inserts an END statement, flagged by a string of asterisks in the comment field.



APPENDIX B. Object File Formats

Object files are in either of two formats, both of which represent binary data bytes as two-digit ASCII hexadecimal numbers. An object file consists of a sequence of data records, followed by a single end record.

B-1 Intel HEX Format

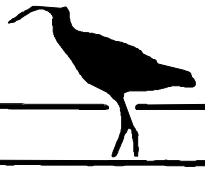
Data Record:

Byte 1	Colon (:)
2..3	Number of binary data bytes in this record.
4..5	Load address for this record, high byte.
6..7	Load address, " " " low byte.
8..9	Unused, should be "00".
10..x	Data bytes, two characters each.
x+1..x+2	Checksum (2 characters).
x+3..x+4	CR/LF

End Record:

Like data record, but number of data bytes is zero and the load address field contains the program starting address.

The checksum is the two's complement of the 8-bit sum, without carry, of all the data bytes, the two bytes of load address, and the byte count.



B-2 KIM Format

This format is used by the MOS Technology KIM-1 and by a number of other 6502-based microcomputers.

Data Record:

Byte	1	Semicolon (;)
	2..3	Number of data bytes.
	4..5	Load address, high byte.
	6..7	Load address, low byte.
	8..x	Data bytes.
	x+1..x+2	Checksum, high byte.
	x+3..x+4	Checksum, low byte.
	x+5..x+6	CR/LF
	x+7..x+12	6 NUL characters

End Record:

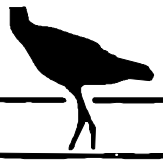
Like data record, but byte count is zero and load-address field contains total number of records in the file. End record is followed by an XOFF (DC3) character.

The checksum is a 16-bit sum without carry, uncomplemented.

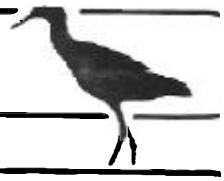
APPENDIX C. Instruction Mnemonics and Opcodes

This is an actual XASM65 assembly listing, showing all of the instruction mnemonics, the available addressing modes for each, and the associated hex opcodes.

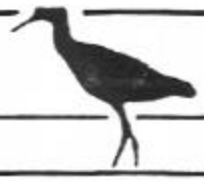
```
0100                    ORG     100H

0025            SMAL    EQU     25H
4455            BIG     EQU     4455H
0100            LONG    EQU     $


0100 6905               ADC     #5
0102 6525               ADC     SMAL
0104 7525               ADC     SMAL,X
0106 6D5544             ADC     BIG
0109 7D5544             ADC     BIG,X
010C 795544             ADC     BIG,Y
010F 6125               ADC     (SMAL,X)
0111 7125               ADC     (SMAL),Y

0113 2905               AND     #5
0115 2525               AND     SMAL
0117 3525               AND     SMAL,X
0119 2D5544             AND     BIG
011C 3D5544             AND     BIG,X
011F 395544             AND     BIG,Y
0122 2125               AND     (SMAL,X)
0124 3125               AND     (SMAL),Y

0126 0A                 ASL     A
0127 0625               ASL     SMAL
0129 1625               ASL     SMAL,X
012B 0E5544             ASL     BIG
012E 1E5544             ASL     BIG,X

0131 90FE               BCC     $
0133 B0FE               BCS     $
0135 F0FE               BEQ     $

0137 2425               BIT     SMAL
0139 2C5544             BIT     BIG
```



```
013C 30FE               BMI     $
013E D0FE               BNE     $
0140 10FE               BPL     $

0142 00                 BRK

0143 50FE               BVC     $
0145 70FE               BVS     $
0147 18                 CLC
0148 D8                 CLD
0149 58                 CLI
014A B8                 CLV

014B C905               CMP     #5
014D C525               CMP     SMAL
014F D525               CMP     SMAL,X
0151 CD5544             CMP     BIG
0154 DD5544             CMP     BIG,X
0157 D95544             CMP     BIG,Y
015A C125               CMP     (SMAL,X)
015C D125               CMP     (SMAL),Y

015E E005               CPX     #5
0160 E425               CPX     SMAL
0162 EC5544             CPX     BIG

0165 C005               CPY     #5
0167 C425               CPY     SMAL
0169 CC5544             CPY     BIG

016C C625               DEC     SMAL
016E D625               DEC     SMAL,X
0170 CE5544             DEC     BIG
0173 DE5544             DEC     BIG,X

0176 CA                 DEX
0177 88                 DEY

0178 4905               EOR     #5
017A 4525               EOR     SMAL
017C 5525               EOR     SMAL,X
017E 4D5544             EOR     BIG
0181 5D5544             EOR     BIG,X
0184 595544             EOR     BIG,Y
0187 4125               EOR     (SMAL,X)
0189 5125               EOR     (SMAL),Y

018B E625               INC     SMAL
018D F625               INC     SMAL,X
018F EE5544             INC     BIG
0192 FE5544             INC     BIG,X
```



```
0195 E8              INX
0196 C8              INY

0197 4C0001          JMP     LONG
019A 6C0001          JMP     (LONG)
019D 200001          JSR     LONG

01A0 A905            LDA     #5
01A2 A525            LDA     SMAL
01A4 B525            LDA     SMAL,X
01A6 AD5544          LDA     BIG
01A9 BD5544          LDA     BIG,X
01AC B95544          LDA     BIG,Y
01AF A125            LDA     (SMAL,X)
01B1 B125            LDA     (SMAL),Y

01B3 A205            LDX     #5
01B5 A625            LDX     SMAL
01B7 B625            LDX     SMAL,Y
01B9 AE5544          LDX     BIG
01BC BE5544          LDX     BIG,Y

01BF A005            LDY     #5
01C1 A425            LDY     SMAL
01C3 B425            LDY     SMAL,X
01C5 AC5544          LDY     BIG
01C8 BC5544          LDY     BIG,X

01CB 4A              LSR     A
01CC 4625            LSR     SMAL
01CE 5625            LSR     SMAL,X
01D0 4E5544          LSR     BIG
01D3 5E5544          LSR     BIG,X

01D6 EA              NOP

01D7 0905            ORA     #5
01D9 0525            ORA     SMAL
01DB 1525            ORA     SMAL,X
01DD 0D5544          ORA     BIG
01E0 1D5544          ORA     BIG,X
01E3 195544          ORA     BIG,Y
01E6 0125            ORA     (SMAL,X)
01E8 1125            ORA     (SMAL),Y

01EA 48              PHA
01EB 08              PHP
01EC 68              PLA
01ED 28              PLP
```



```
01EE 2A             ROL     A
01EF 2625           ROL     SMAL
01F1 3625           ROL     SMAL,X
01F3 2E5544         ROL     BIG
01F6 3E5544         ROL     BIG,X

01F9 6A             ROR     A
01FA 6625           ROR     SMAL
01FC 7625           ROR     SMAL,X
01FE 6E5544         ROR     BIG
0201 7E5544         ROR     BIG,X

0204 40             RTI
0205 60             RTS

0206 E905           SBC     #5
0208 E525           SBC     SMAL
020A F525           SBC     SMAL,X
020C ED5544         SBC     BIG
020F FD5544         SBC     BIG,X
0212 F95544         SBC     BIG,Y
0215 E125           SBC     (SMAL,X)
0217 F125           SBC     (SMAL),Y

0219 38             SEC
021A F8             SED
021B 78             SEI

021C 8525           STA     SMAL
021E 9525           STA     SMAL,X
0220 8D5544         STA     BIG
0223 9D5544         STA     BIG,X
0226 995544         STA     BIG,Y
0229 8125           STA     (SMAL,X)
022B 9125           STA     (SMAL),Y

022D 8625           STX     SMAL
022F 9625           STX     SMAL,Y
0231 8E5544         STX     BIG

0234 8425           STY     SMAL
0236 9425           STY     SMAL,X
0238 8C5544         STY     BIG

023B AA             TAX
023C A8             TAY
023D 98             TYA
023E BA             TSX
023F 8A             TXA
0240 9A             TXS
0000                END
```

Current Version: 2.02 (Released 21-Oct-83)

Previous Versions: 2.01 (Released 04-Aug-83)
2.00 (Released 04-Oct-83)
1.98 (Released 25-May-83)

Manual: Rev. 1.5

Diskette Contents

1. XASM65.COM ..... Cross-Assembler, Executable File

2. TEST65.ASM ..... Demonstration source code file, showing all 6500-series instructions, addressing modes, and other assembler features.

V2.02 - New Features

1. Bug Fix: The BBRn and BBSn instructions now generate correct displacements.

2. Improvement: NUL characters are no longer output as part of the assembly-listing page heading. This was causing problems with certain printers (notably older Okidatas) which treated the NULs in a non-standard way (ie. failed to ignore them).

V2.01 - New Features

1. The pseudo-op EJECT has been added as a synonym for PAGE.

Version 2.00 - New Features

1. SUPPORT FOR NEW 6500-SERIES CHIPS

   The assembler now supports the new instructions and addressing modes of the 6511 (R6500/11) and 65C02 (R65C02, R65C102, R65C112) microprocessors. Examples of all new instructions and modes are shown in the test file (TEST65.ASM), with comments indicating which chips include them.

   Two new pseudo-ops - MOD11 and MODC02 - allow you to inform the assembler which chip your code is destined for. Any instructions not available in the specified chip will be flagged with a 'W' (Warning) error.

   If neither MOD11 nor MODC02 is specified, then only the standard 6502 instruction set will be considered valid. In terms of instructions, the 6502 is a proper subset of the 6511, which in turn is a proper subset of the 65C02. Each instruction generates the same object code regardless of the chip specified.

2 IMPROVED PARSING OF INSTRUCTION OPERANDS

The assembler has been substantially re-worked internally to provide better syntax checking for instruction operands. Junk on the end of the operand is now detected (causing a syntax error), and misuse of certain addressing modes is also caught. Note that this discussion applies only to machine instructions, not to pseudo-ops. Note, also, that some errors formerly flagged as 'X' (Index) are now flagged as 'S' (Syntax) errors.

By way of review, the indexed addresing modes are specified as follows:

syntax	mode
expression,X	Absolute or Zero-Page w/index X
expression,Y	Absolute or Zero-Page w/index Y
(expression)	Indirect
(expression,X)	Indirect w/index X
(expression),Y	Indirect w/index Y

When the assembler sees a left parenthesis as the first character of the operand field, regardless of the instruction type, it assumes that one of the indirect modes is present. Thus, you'll need to be careful about using expressions which begin with a left parenthesis. For example, if you write

```
(GRINCH+1)*5
```

intending to use Absolute mode, the assembler will treat this as an Indirect-mode operand with junk on the end, resulting in a syntax error. The solution is to either rearrange the expression so that it doesn't begin with a left parenthesis, as in-

```
5*(GRINCH+1)
```

- or to use an intermediate symbol, as:

```
FOO     EQU     (GRINCH+1)*5
        CMP     FOO
```

3. FORWARD REFERENCES IN INSTRUCTIONS

Instructions which have a zero-page mode without the corresponding absolute mode will now use the zero-page mode for forward-referenced operands. For example, the STX instruction has the ZeroPage,Y mode but not the Absolute,Y mode. Thus any STX operand of the form

```
expression,Y
```


will be assembled as Zero-Page,Y. A range error will occur if the expression turns out to be greater than 255.

4 FORWARD REFERENCES IN PSEUDO-OPS

An 'F' error will occur if a forward reference is present in the operand of any of the following pseudo-ops:

DS
EQU
IF
LOC
ORG
SET

In general, if an 'F' error is present in your assembly you can expect obscure problems; usually, some symbols will have values which don't correspond to the location counter. Thus, it's best to get rid of all illegal forward references before trusting the generated code.

5. ASSEMBLY LISTING

The assembly listing now defaults to the disk (a .PRN file is produced) rather than to the printer. It may be diverted to the printer by using the 'P' switch in the assembler command line. The 'D' switch is no longer implemented.

Versions 1.98 and Later - New Features

1. LOC and ENDLOC Pseudo-Ops

These pseudo-ops make it possible to assemble (in line) code which will be moved to another location for execution. LOC is similar to ORG, but affects only the execution address, not the address at which the code will reside in the load module. Following a LOC, instructions and data continue to be laid down in line, but the values of the location counter (and thus of any new labels) reflect the origin specified in the LOC.

LOC takes a single arithmetic expression as argument; the value of the expression is the new starting value for the location counter. The expression should not contain any forward references. The location counter in the assembly listing will reflect the execution address set with LOC, rather than the load address.

ENDLOC, which takes no argument, cancels the effect of an extant LOC: it sets the execution address equal to the current load address, thus resuming normal assembly.

The effect of a LOC is also cancelled when an ORG is encountered. Thus, the load address should always be set first, with ORG, and the offset execution address then set with LOC.

An example of the use of LOC and ENDLOC may be found in the current version of TEST65.ASM.

2. LOWER CASE INPUT

The assembler now accepts input in lower case. No distinction is made between the upper and lower case versions of the same character; thus, for example,

grinch

is equivalent to

GRINCH

when used as a symbol. Upper and lower cases are distinguished properly within character constants and strings.

BUG FIX: PAGE HEADINGS

In previous versions of the assembler, the page heading lines of the assembly listing included some linefeed characters with the high-order bit set (but only when listing was sent directly to the printer). This caused erratic behavior, such as overprinted lines, with some printers.

4. PAGE EJECTS

A page eject is now issued before the first page of the assembly listing. This is for those folks with MP/M who were getting pieces of their listing printed on the tail end of somebody else's.

5. ERROR COUNT

The error count (number of lines containing errors) is now numerically correct, and is both displayed on the console and included in the assembly listing. If no errors were detected, the assembler will tell you.

6. PAGE NUMBERING

Page numbers in the assembly listing are now 3 digits.

7. BUG FIX: SYMBOL NAMES

Symbols beginning with '.' are now accepted, per the manual.

8. BUG FIX: SHIFT-RIGHT OPERATOR

The SHR arithmetic operator now works correctly.

9. BUG FIX: UNTERMINATED CONDITIONALS

The END statement is now recognized even if encountered within the skipped section of an IF/ELSE construct. This prevents runaway if you forget an ENDIF.

10. INCLUDE PSEUDO-OP

To remedy a past sin, INCLUDE is now recognized as a synonym for INCLUD.

Other Addenda and Errata To Manual

1. INTERRUPTING THE ASSEMBLER

Assembly may be interrupted at any time by striking a CTRL-C.

2. LABEL FIELD

Please note that labels (including those on the left-hand side of an EQU or SET) MUST begin in column 1. Conversely, anything that begins in column 1 is considered a label. We've seen several cases in which a user placed TITLE or SBTTL pseudo-ops in column 1, and then wondered why they didn't work.

3. COLONS

Despite what the manual says, labels don't require colons.

Current Versions

CP/M-86:	2.01C	Released 26-Oct-83
MSDOS/PCDOS:	2.01M	" " " "
Manual:	1.3	

Previous Versions: 2.00C, 2.00M (Released 04-Aug-83)

Diskette Contents
---

1. XASM65.CMD (CP/M-86) ..... Cross-Assembler, Executable File
   or
   XASM65.COM (MSDOS, PCDOS)

2. TEST65.ASM ..... Demonstration source code file, showing all 6500-series instructions, addressing modes, and other assembler features.

Version 2.01 - New Features
---

1. Bug Fix: The BBRn and BBSn instructions now generate the correct relative branch displacement.

2. Improvement: Page headers in the assembly listing no longer contain NUL characters. NULs should rightfully be ignored by all ASCII devices, but some printers (eg. older Okidatas) go haywire upon receiving them.

3. Synonym: EJECT is now recognized as a synonym for PAGE.

OTHER NOTES

Notes for V2.00 and earlier are identical to those for the 8080/Z80 version of the assembler. See preceeding pages (65README/80).

