OHIO SCIENTIFIC
6500 ASSEMBLER/EDITOR
USER'S MANUAL

6500 Assembler/Editor

User's Manual

Table of Contents

Introduction

The OSI 6500 Assembler/Editor provides assembly language programming capability that is fully format compatible with the standard MOS Technology assembly language. Therefore, all existing standard 6500 programs can be directly assembled without changes and the OSI Assembler is compatible with all existing MOS Technology 6500 documentation. The Assembler is easy to use and provides your choice of a full assembly listing, an errors-only listing, an object tape listing or an assembly of machine code directly to memory. The Editor provides easy-to-use, line-oriented file entry and editing with commands for selective source file printout, deletion of lines, re-numbering of lines and unlimited insertions between existing lines.

The 6500 Assembler/Editor is offered in a number of different versions for various system configurations. Refer to the appropriate appendix for information relevant to your particular version.

This manual thoroughly explains the operation and usage of the Assembler/Editor and provides a summary of the MOS Technology 6500 assembly language. For a detailed explanation of 6500 assembly language programming, the reader is referred to one of the available texts on the subject such as the MOS Technology Programming Manual or "How to Program Microcomputers" by William Barden.

Elements of the Assembly Language

The following elements of the assembly language are first defined.

Labels

Labels are used in assembly language programs to identify program points - specific locations in a program where an instruction sequence begins, a constant is located, or a variable is to be stored that needs to be referenced elsewhere in the program. The use of a label at a program point provides the means to refer to the program point mnemonically, or by name, rather than by the machine

address of the location. The use of a label also permits the Assembler to automatically update all references to a program point when its machine address changes due to changes in the program. A label may also be used synonomously for a numerical value which is not a program point. Such a label is often called a symbol.

A label consists of one to six characters from the set:

A-Z
Ø-9
.
:
$

The first character in a label must be a letter. The reserved names A, X, Y, S and P may not be used as labels.

There is one predefined label, "*", which refers to the current value of the location counter throughout an assembly.

Examples:	START	*
	TEN.1	LABEL:
	TABLE	A$B

Constants

Constants are used in assembly language programs for numerical values and character strings.

Although numerical constants are generally expressed internally in the computer in binary, the Assembler permits them to be expressed in binary, octal, hexidecimal or decimal. The absence of a prefix symbol indicates a decimal value. The other number bases are indicated by the prefixes:

% for binary

@ for octal

$ for hexidecimal

Numerical constants are always unsigned, positive numbers. They are limited in magnitude to 255 (decimal) if they are to be represented by a single byte or to 65535 if a double byte (or word).

Examples: 1000 @16400
%10110 1 $FACE

Character string constants are represented internally in the American Standard Code for Information Interchange (ASCII). The Assembler accepts character string constants as any sequence of printing ASCII characters preceeded by and followed by a single quote mark (an apostrophe). If an apostrophe is to be included within the string, two successive apostrophes must be used. The Assembler also accepts a single character preceeded by an apostrophe as the operand of an immediate mode instruction. (See Instruction Addressing Modes - Immediate)

Examples: 'THIS IS A CHARACTER STRING CONSTANT'
'@!"&#%$'
'12 dozen'
''''

Expressions

An expression is a sequence of one or more labels and/or numerical constants separated by the arithmetic operators +, -, * and / (for add, subtract, multiply and divide). An expression defines a value in terms of an algebraic expression that is evaluated at assembly time (not program execution time). The Assembler evaluates expressions strictly left to right; there is no operator precedence in expression evaluation and parentheses may not be used to alter the order of operations.

Examples: START TABLE-1
START+1 0-1
SIZE/256*256 %101+@10*15/$12

Assembler Statements

The Assembler recognizes three types of statements:

- Comment Statements
- Instruction Statements
- Directive Statements

Each statement is one line of assembler source code.

Comment Statements

Comment statements are included in assembly language programs to provide headings and other explanatory information. They, in no way, affect the machine code generated by the Assembler.

A comment statement consists of a semicolon as the leftmost, non-blank character followed by any commentary whatever.

Examples:
```
; CONVERSION PROGRAM
; **Output Subroutine**
;
```

Instruction Statements

Instruction statements specify a 6500 assembly language instruction mnemonic or "opcode". The opcode may be preceeded by a label and must be followed by an appropriate "operand" corresponding to the instruction addressing mode as described below. The operand may be followed by any commentary desired. Thus, the fields in an instruction statement are:

(label) (opcode) (operand) (comment)

where the (label) and (comment) fields are optional and the (operand) is as appropriate to the instruction addressing mode. At least one space must exist between the fields.

Examples:
```
START   LDA VALUE     GET STARTING VALUE
        STX XTEMP     SAVE X-REG

TEST    SEC
```

Instruction Addressing Modes

The 6500 Assembler recognizes eight different assembly language addressing modes. These are further analyzed at assembly time and result in the choice of an instruction from one of the thirteen machine language addressing modes. Each assembly language addressing mode has a unique operand syntax. The form of each is as defined below.

Implied

The implied addressing mode has no operand. The operand on which the instruction is to operate is implied by the instruction opcode itself.

```
Examples:  CLC        PHA
           INX        RTS
           BRK        TYA
```

Accumulator

The accumulator addressing mode is used to specify the accumulator register as the operand.

Operand Form: A

```
Examples:  LSR A
           ROR A
```

Immediate

The immediate addressing mode is used to specify a single byte constant as the operand.

Operand Form: #expression or single character constant optionally followed by an expression

```
Examples:  LDA #Ø          LDX #99-%1Ø1
           ADC #'A         CMP #'?-1
           AND #$8F        LDA #LABEL/256
```

Symbolic

The symbolic addressing mode is used to reference an absolute (above page zero) or page zero memory location. When a symbolic operand is evaluated, the Assembler may choose either an absolute or a page zero machine language instruction as required by the operand value. If the opcode is that of a branch instruction, the relative addressing mode is always used.

Operand Form: expression

Examples:

```
LDA VALUE        CMP LIMIT-1
ASL BITS+2       SBC SIZE/2+START
STA Ø            JMP START
BNE NEXT         BVS OVRF
```

Note: If any label in a symbolic operand expression is undefined when that operand is encountered during pass one of an assembly and the instruction has both zero page and absolute addressing modes, the Assembler assumes that the absolute addressing mode and thus a two byte operand will be required. If on pass two the operand evaluates to a page zero location and one byte could be used, a warning error is printed noting a forward reference to page zero memory (error 19). This is not a serious error and only indicates that an extra byte was used.

Indexed

The indexed addressing mode is used to reference an absolute (above page zero) or a page zero location as specified by the instruction operand plus the contents of the specified index register at instruction execution time. When an indexed operand is evaluated, the Assembler may choose either an absolute or a page zero machine language instruction as required by the operand value.

Operand Form: expression,X or

expression,Y

Examples:

```
LDA TABLE-1,X        LDX LIST,Y
STA MARK,Y           LDY STACK,X
DEC Ø,X              ROR DIVISR,X
ASL TWO,X            STX ZIP,Y
```

See the note under Symbolic.

Indexed Indirect

The indexed indirect addressing mode is used to reference a location as specified by the address in the two page zero locations indicated by the instruction operand plus the contents of the X index register at instruction execution time.

Operand Form: (expression,X)

Examples:

```
LDA (LIST,X)         CMP (LIMITS+2,X)
STA (ADRS,X)         ORA (BUFFER,X)
```

Indirect Indexed

The indirect indexed addressing mode is used to reference a location as specified by the address formed by adding the Y index register to the address in the two page zero locations indicated by the instruction operand.

Operand Form: (expression),Y

Examples:

```
LDA (LIST),Y         CMP (LIMITS+2),Y
STA (ADRS),Y         ORA (BUFFER),Y
```

Indirect

The indirect addressing mode is used to reference a location as specified by the address in the two locations indicated by the instruction operand.

Operand Form: (expression)

Examples:

```
JMP (RETURN)
JMP (RETABL+RND)
```

Not all 6500 instructions can have all addressing modes. The table on the following page defines for each instruction the available assembly language and machine code level addressing modes.

/

Table 1

6500 Instruction Addressing Modes

	Assembler Addressing Modes	AC	IM	Symbolic			Indexed				Indirect		
	Machine Language Addressing Modes	A C	I M	Z P	A b s	R e l	Z P X	A b s X	Z P Y	A b s Y	I n X	I n Y	I n
General	ADC AND CMP EOR LDA ORA SBC		X	X	X		X	X		X	X	X	
Shift	ASL LSR ROL ROR	X		X	X		X	X					
Bit Test	BIT			X	X								
Compare Index	CPX CPY		X	X	X								
Decrement/ Increment	DEC INC			X	X		X	X					
Jump	JMP JSR				X X								X
Load Index	LDX LDY		X X	X X	X X		X	X	X	X			
Store Index	STX STY			X X	X X		X		X				
Store	STA			X	X		X	X		X	X	X	
Branch	BCC BCS BEQ BMI BNE BPL BVC BVS					X							
Implied	BRK NOP RTI RTS CLC CLD CLI CLV DEX DEY INX INY PHA PHP PLA PLP SEC SED SEI TAX TAY TSX TXA TYA TXS	Implied											

AC - Accumulator Abs - Absolute
IM - Immediate Rel - Relative
ZP - Zero Page In - Indirect

Directive Statements

Directive statements specify an assembler directive (as opposed to a 6500 instruction). Directives are used to define numerical and character string constants, to set the program location origin, reserve space and equate labels to values. All directives begin with a period and appear in the opcode field of a directive statement. The directive may be preceeded by a label and must be followed by an appropriate operand. The operand may be followed by any commentary desired. Thus, the fields in a directive statement are:

(label) (directive) (operand) (comment)

where the (label) and (comment) fields are optional and the (operand) is as appropriate to the directive. At least one space must exist between fields.

The following directives are provided in the 6500 Assembler:

.BYTE Directive

This directive is used to generate single byte numerical constants and character string constants of any length.

Operand Form: expression or character string constant, ...

Examples:
```
        .BYTE Ø
        .BYTE %1Ø,@1Ø,1Ø,$1Ø,'1Ø',TEN
MESG    .BYTE 'MESSAGE ',255,'.',$D
        .BYTE SIZE/2+1
```

.WORD Directive

This directive is used to generate two byte numerical constants in the low, high order by byte used for machine addresses.

Operand Form: expression, ...

Examples: .WORD $ABCD

.WORD START,START+2

.WORD 1Ø,1ØØ,1ØØØ,1ØØØØ

.DBYTE Directive

This directive is used to generate two byte numerical constants in high, low order by byte.

Operand Form: expression, ...

Examples: .DBYTE $ABCD

.DBYTE START,START+2

.DBYTE 1Ø,1ØØ,1ØØØ,1ØØØØ

Note: Any other character string following a period such as .END, is ignored by the Assembler.

Equals (=) Directive

This directive is used to set the program location origin, reserve space and equate labels to values. Each of these operations uses a distinct statement form and is shown separately below.

Setting the Program Location Origin

Statement Form: * = expression

Examples: * = $Ø3ØØ

* = START

* = LAST+15

Reserving Space

Statement Form: label * = *+expression (label is optional) or

label * = expression+*

Examples: * = *+1

TABLE * = *+128

ARRAY * = XDIM*YDIM+*

Equating Labels to Values

Statement Form: label = expression

Examples: TEN = 1Ø

START = $8CØØ

SIZE = XDIM*YDIM

Note: In all forms of the equals directive, all labels used in an expression must be defined previously in the assembler source file or the Assembler cannot determine the location of subsequent labels during pass one of the assembly. This can result in "label previously defined" errors (error 12) for the subsequent labels during pass two.

Assembly Commands

The Assembler has four assemble commands:

A or AØ gives a fully assembly listing

A1 gives an errors-only listing

A2 gives an object tape listing

A3 assembles object code directly to memory

Full Assembly Listing - A

The A or AØ command gives a full assembly listing including, from left to right, for each line of assembly source code:

- source code line number
- program location
- object (machine) code - zero to three bytes
- source code

If any errors were detected in the source code, pointer(s) to the error and the appropriate error number(s) appear after the above line. The machine code generated in the case of an error depends on the type of error, but generally, is either the appropriate opcode byte with a zero operand or three NOP bytes to facilitate patching without requiring reassembly.

See Figure 1.

```
A

  10                ;  PAGE 0,1 SWAP SUBROUTINE
  20                ;
  30                ;    ENTRY:   NO PARAMS
  40                ;    EXIT:    STUFF SWAPPED
  50                ;             A SAVED
  60                ;    PURE PROCEDURE
  70                ;
  80 4700                * = $4700
  90                ;
 100                ;  SWAP STORAGE
 110 4900           SWAPB  * = *+512      PAGE 0,1
 120                ;
 130                ;  TEMP STORAGE
 140 4902           RTN    * = *+2        RETURN
 150                A      * = *+1        A (NOT SWAPPED)
-----------------------↑
E#    1
 160                ;
 170                ;  REMOVE RETURN FROM HIS STACK
 180 4902 EAEAEA    SWAP:  STA A
------------------------------↑
E#    5
 190 4905 68              PLA
 200 4906 18              CLC
 210 4907 6901            ADC #1          ADJUST AS RTS DOES
 220 4909 8D0049          STA RTN
 230 490C 68              PLA
 240 490D 6900            ADC #0
--------------------------------↑
E#   18
 250 490F 8D0149          STA RTN+1
 260                ;
 270                ;  SWAP PAGES 0,1
 280 4912 A200            LDX #0
 290 4914 BD0001 SWAP1    LDA 256,X         INTERCHANGE 2
 300 4917 BC0048          LDY SWAPB+256,X     PAGE 1 BYTES
 310 491A EAEAEA          STA SWAPB+256,S
---------------------------------------↑
E#    8
 320 491D 98              TYA
 330 491E 9D0001          STA 256,X
 340                ;
 350 4921 B500            LDA 0,X           THEN INTERCHNAGE 2
 360 4923 BC0047          LDY SWAPB,X         PAGE 0 BYTES
 370 4926 9D0047          STA SWAPB,X
 380 4929 98              TYA
 390 492A 9500            STA 0,X
 400 492C E8              INX
 410 492D D0E5            BNE SWAP1       DO 256 LOOPS
 420                ;
 430 492F EAEAEA          LDA A
------------------------------↑
E#    5
 440 4932 6C0049          JMP (RTN)
 450                ;
 460                      .END
```

Figure 1
Full Assembly Listing

Errors-Only Listing - A1

The A1 command produces the same output described above for the full assembly listing, but does so only for source lines which contain detected assembly errors.

See Figure 2.

Object Tape Listing - A2

The A2 command produces an object tape listing in the standard format which includes for each object tape "record":

- start of record character (a semicolon)
- number of data bytes in record up to 24 (18 hex)
- address (or location) of data bytes (4 hex digits)
- values of data bytes (2 hex digits each)
- record checksum (4 hex digits) - 16 bit sum modulo 65536
- carriage return, line feed, nulls

See Figure 3.

Assembly to Memory - A3

The A3 command produces no printed output, but places the object code resulting from the assembly directly into RAM at the specified program location, plus any offset previously entered with the M (Memory Offset) command.

This command must be used with caution as there is no protection against the object code overwriting the Assembler or other needed RAM.

```
A1

  150 4902          A       * = *+1         A (NOT SWAPPED)
--------------------↑
E#     1
  180 4902 EAEAEA SWAP:   STA A
------------------------------↑
E#     5
  240 490D 6900           ADC #0
--------------------------------↑
E#    18
  310 491A EAEAEA         STA SWAPB+256,S
------------------------------------------↑
E#     8
  430 492F EAEAEA         LDA A
--------------------------------↑
E#     5
```

Figure 2
Errors-Only Listing

```
A2

;184902EAEAEA68186901BD00496869008D0149A200BD0001BC004808ED
;18491AEAEAEA989D0001B500BC00479D0047989500E8D0E5EAEAEA0D93
;0349326C00490133
```

Figure 3
Object Tape Listing

Assembly Errors

The following descriptions of assembly errors and their possible causes are provided to facilitate elimination of these errors in an assembly.

1. A, X, Y, S AND P ARE RESERVED NAMES

   One of these reserved names was found in the label field. No code is generated for a statement with a reserved name as a label. Use of a reserved name in an expression will give an "undefined label" error, error 18.

3. ADDRESS NOT VALID

   An address greater than 65535 (hex FFFF) was encountered.

4. FORWARD REFERENCE IN EQUATE, ORIGIN OR RESERVE DIRECTIVE

   An expression used in one of these directives includes a label that hasn't been previously defined in the assembly source file.

5. ILLEGAL OPERAND TYPE FOR THIS INSTRUCTION

   An operand was found which is not defined for the specified instruction opcode. Refer to Table 1 for the defined instruction addressing modes.

6. ILLEGAL OR MISSING OPCODE

   A defined opcode was not found. Refer to Table 1 for the defined opcodes.

7. INVALID EXPRESSION

   An expression was found that is not a valid sequence of numerical constants and/or labels separated by valid operators or is not a valid instruction operand form.

8. INVALID INDEX - MUST BE X OR Y

   An indexable instruction was found with an invalid index. Refer to Table 1.

9. LABEL DOESN'T BEGIN WITH ALPHABETIC CHARACTER

   A non-alphabetic character was encountered where a label was expected.

10. LABEL GREATER THAN SIX CHARACTERS

    A string of more than six valid label characters (A-Z, Ø-9, $, ., :) was found before a non-valid label character. This is a warning message. Assembly continues using the first six characters of the label.

12. LABEL PREVIOUSLY DEFINED

The identified label has previously occurred in the assembler source file or this occurrence of the label had a different value on pass one than on pass two. The latter error may be caused by previous errors in the assembly.

13. OUT OF BOUNDS ON INDIRECT ADDRESSING

An indirect-indexed or indexed-indirect address does not fall into page zero as required.

15. RAN OFF END OF LINE

An operand is required and wasn't found before the end of the line.

16. RELATIVE BRANCH OUT OF RANGE

The target address of a branch instruction is farther away then the minus 128 to plus 127 byte range of the instruction permits.

18. UNDEFINED LABEL

The identified label is not defined anywhere within the assembler source file.

19. FORWARD REFERENCE TO PAGE ZERO MEMORY

This warning message is generated when an instruction that has both page zero and absolute addressing modes has an operand that is defined later in the assembly source file to be a page zero address. During pass one of the assembly two bytes are allocated for the operand since its value is not yet known. Then during pass two the operand is found to require only a single byte so one byte is wasted. This is usually not a serious error because the generated code will generally execute as expected.

20. IMMEDIATE OPERAND GREATER THAN 255

An immediate operand expression evaluated to greater than 255, the maximum value that can be represented in a single byte immediate operand.

25. LABEL (SYMBOL) TABLE OVERFLOW

The size of the work space is insufficient to hold the current source file and a table for all of the labels encountered in the program. To assemble will require a reduction in either the size of the program source file or the number of symbols or an increase in the size of the work space.

Editor Operation

The Editor provides a very easy to use means for creating assembly language source files and for editing, correcting, printing and punching a tape copy of the created file. The Editor is general purpose in that it is well-suited for creation of any type of source file whether it be assembly language, BASIC, another computer language or other form of printed documentation.

Editor Commands

INIZ — clears any entered source lines from memory. To prevent inadvertently clearing the work space, the question "INIZ?(Y/N)" is asked after the INIZ command has been entered. The user must enter "YES" (or simply "Y") to complete the initialization operation.

(line number)(any text)

enters a source line. The line is placed into the source file at the position indicated by the specified line number. If a line number of zero (Ø) is entered, the line is placed immediately after the previously entered line unless another command has been executed in the interim. If a zero numbered line is entered after the execution of a PRINT command, the line will be placed after the last line referenced by that command. Line numbers may be 1 through 65535.

RESEQ — resequences all line numbers in the source file by 10, starting with 10. After the RESEQ command has been executed, the next sequential line number is output.

DELETE (line specification)

deletes the specified lines from the source file. The (line specification) may take any of the following forms:

first line - last line	deletes the specified lines
first line -	deletes first line through the end
- last line	deletes from the start through last line
line	deletes the specified line

PRINT (optional line specification)

prints the specified lines in order by line number. The (line specification) may take any of the forms listed above for DELETE or may be omitted in which case all lines in the source file are printed.

In the PRINT and DELETE commands any number of line specifications, spearated by commas may be entered after the command.

All commands can be abbreviated down to their first letter. All commands and source lines must be followed by a carriage return.

When entering commands or source text lines, corrections for typing errors can be made to a line anytime before the final carriage return is entered. An up-arrow character may be used to delete all characters so far entered on the line and a back-arrow may be used to successively delete previously entered single characters, one back-arrow for each character to be deleted.

Note: Up-arrow may be ↑, ∧ or shift/N; back-arrow may be ___ or shift/O on your keyboard.

If an uninterpretable command is input, an explanatory error message is output with a pointer to the specific part of the command which could not be interpreted. When such an error occurs, any correct commands that appeared on the line previous to the erroneous command have been completed; those after the point of the error were not.

Assembler/Editor Statistics

Source File Storage Requirements (per line):

two bytes for the line number plus,

one byte for each text character plus,

one byte for the line terminator character (ØD)

All repeated characters such as a sequence of spaces, occupy only two bytes; one for the character and one for a repeat count.

Symbol Table Storage Requirements:

Six bytes/symbol. 6500 opcodes and reserved names occupy no symbol table space.

Assembly Speed:

Approximately 600 lines per minute.

Appendix A

OS-65D V3.0 Version

of the

6500 Assembler/Editor

In OS-65D V3.0, the Assembler/Editor is loaded from disk and initiated by typing ASM after the A* prompter in the DOS kernel command mode. Whenever exiting to the DOS, you can return to the Assembler/Editor as long as it is loaded by typing RETURN ASM.

This version of the 6500 Assembler/Editor contains the following commands in addition to those described elsewhere in this manual.

EXIT	exits the Assembler/Editor and transfers control to the OS-65D kernel which then displays the A* prompter.
Hnnnn	sets the high memory limit to hexidecimal address nnnn.
Mnnnn	sets the memory offset for A3 assemblies to hexidecimal nnnn.
Control-I	tabs 8 spaces from the current print position. Also:

Control-U	7 spaces
Control-Y	6 spaces
Control-T	5 spaces
Control-R	4 spaces
Control-E	3 spaces

Control-C	aborts the current operation.
! command line	sends the command line to OS-65D to be executed then returns to the Assembler.

This version of the Assembler/Editor occupies memory from Ø2ØØ through 16FF. Its work space starts at 3179 (3279 in mini-floppy versions) and is utilized as shown below:

3179,317A	address of start of source (low, high) - normally 317E
317B,317C	address of end of source +1 (low, high)
317D	number of tracks required for source
317E	normal start of source

Note: It is possible to carry the Assembler's symbol table forward from one assembly to another. To do so, exit the Assembler after the first assembly

and enter the machine language monitor by typing "RE M". Change location Ø855 from ØA to 18 and read out the contents of locations ØØØA and ØØØB. Enter the values from these locations into locations 12FA and 12FB, respectively. Then re-enter the Assembler by re-entering the DOS kernel at 2A51 and typing "RE A". Now the symbols from the first assembly will remain in the symbol table for reference during the next assembly. Likewise, the symbols from the first and second assemblies will remain for the third assembly, etc. If you want to eliminate all but the symbols from the first assembly, exit the Assembler and immediately re-enter it by typing "RE A". To restore normal operation of the Assembler change location Ø855 back to ØA. This will cause the symbol table to be cleared at the beginning of each assembly.

/

Appendix B

Word Processor WP-1B Version

of the

6500 Assembler/Editor

The WP-1B diskette boots up with the following message:

OSI 6500/6800 SOFTWARE DEVELOPMENT SYSTEM VERSION 1.0

MEMORY SIZE?

Enter the memory size in decimal or hexidecimal preceeded by a $ or just hit the return key and all available memory will be used. The system then displays:

SELECT ASSEMBLY:

A. 6500
B. 6800

Type an A to select the 6500 Assembler/Editor or a B to select the 6800 Assembler which uses the same Editor. Instructions for use of the 6800 Assembler are provided in a separate manual.

This version of the 6500 Assembler/Editor contains many enhanced Editor commands as described in the Ohio Scientific Word Processor Manual. The section of that manual entitled "Extended Features for Advanced Programming Topics" also applies with the exception that BASIC is not included.

The disk operating system which can be entered by typing "EXIT" contains the following additional command of use to the 6800 Assembler user:

Xnnnn switch to the 6800 processor and transfer control to hexidecimal location nnnn.

The system also includes indirect file capability.

Using Indirect Files

Often it is desirable to be able to merge two or more assembler source files. The indirect file provides a mechanism for doing this.

In order to use an indirect file, you must have enough RAM to hold the required program(s) in the Assembler work space and another copy of the program(s) above the work space. The top of the work space can be appropriately set up

with the DOS Hnnnn command. Then the indirect file mechanism is set up with this address +1 by entering it into the following location:

21A2 indirect file input address (high) - normally = 80

The low part of this address is fixed at ØØ.

Transfers to and from the indirect file are then performed as follows:

Dumping Source from the Work Space to an Indirect File

1. Load the source into the Assembler work space with the LOAD command.
2. Output the source but type a [after typing PRINT and before hitting the RETURN key. This turns the indirect file output on.
3. At the completion of the output type a]. This will be echoed as]] and will turn the indirect file output off.

Loading Source from an Indirect File to the Work Space

1. Clear the work space by typing INIZ then YES. Or, load the source file into the work space into which the indirect file is to be merged.
2. Type a Control-X and hit the RETURN key. The indirect file data will be loaded into the work space. When the] character is input at the end of the file, the indirect file input will be terminated and you can containue with editing or assembly.

Note: It is possible to carry the Assembler's symbol table forward from one assembly to another. To do so, exit the Assembler after the first assembly and enter the machine language monitor by typing "RM". Change location Ø855 from ØA to 18 and read out the contents of locations ØØØA and ØØØB. Enter the values from these locations into locations 12FA and 12FB, respectively. Then re-enter the Assembler by re-entering the DOS at 25ØØ and typing "RA". Now the symbols from the first assembly will remain in the symbol table for reference during the next assembly. Likewise, the symbols from the first and second assemblies will remain for the third, etc. If you want to eliminate all but the symbols from the first assembly, exit the Assembler and immediately re-enter it by typing "RA". To restore normal operation of the Assembler change location Ø855 back to ØA. This will cause the symbol table to be cleared at the beginning of each assembly.

Appendix C

C2P - 1P Cassette Version

of the

6500 Assembler/Editor

This version of the Assembler/Editor is supplied on an auto-load cassette tape. The following procedure may be used to load the Assembler from tape:

Loading the Assembler/Editor

1. Apply power to your personal computer then reset it by pressing the break key. Load the cassette, label up, into the cassette machine and turn the cassette machine on with the volume at about mid-range.

2. Type "ML".

   The M initiates the 65VP monitor and the L starts the auto-load. In a few seconds the four zeros in the upper left portion of the video monitor should change to an incrementing address value with a rapidly changing data field. The value of the address is dependent on which auto-load cassette is being read. At this time a checksum loader is being read into memory in 65VP format. Upon completion (no more than 30 seconds), the checksum loader will load the rest of of the cassette. The Assembler comes up with the message INIZ?(Y/N).

Should a checksum error occur, the following message is printed:

```
OBJECT LOAD CHECKSUM ERR
REWIND PAST ERR - TYPE G TO RESTART
```

If a checksum error consistently happens at the same location, the cassette is probably bad - contact your OSI dealer concerning replacement. However, should checksum errors occur randomly at various locations, it is most likely that there is a problem with the cassette machine or the connection to the computer. Check for broken or frayed connections. Make sure that the playback head and pressure roller/capstan assembly is clean. With a minimal amount of care, no problems with auto-load cassettes should be encountered.

The cassette version of the Assembler/Editor permits loading and saving source code in a manner similar to ROM BASIC.

To save source code:

Type SAVE (CR); type PRINT (line specification), turn on the cassette

machine in record mode and type (CR). As in ROM BASIC, the save mode is disabled by typing LOAD (CR) followed by a space.

To load previously recorded source code:

Turn on cassette machine in play; type load, wait for leader to pass, they type (CR). The LOAD mode is disabled by typing a space.

This version of the Assembler/Editor also provides the following commands:

EXIT	causes the computer to execute its reset vector and display "C/W/M?". Great care must be taken never to type "C" as this will destroy the Assembler/Editor. The Assembler/Editor may then be re-entered by typing "M 12ØØ G".
/	
CONTROL-I	tabs 8 spaces from the current cursor location.

The above commands, as all other Assembler/Editor commands may be executed by typing the first letter only.

This version of the Assembler/Editor occupies memory from Ø24Ø through 139Ø (hexidecimal) and requires a minimum total of 8K of memory to operate. Its source file work space starts at 1391 and ends at 1FFF, as supplied. The entry location is hex 13ØØ. While running all of page zero is used. However, you can exit the Assembler/Editor, use page zero and re-enter it by typing "M 13ØØ G".

The following locations may be changed in the cassette version of the Assembler/Editor to suit your requirements:

12C9,12CA	the low, high memory address of the start of the source file work space. 1391 hex, as supplied.
12CB,12CC	the low, high memory address of the end of the source file work space. 1FFF, as supplied.
12FC,12FD	the low, high memory offset used to bias placement of object code during an A3 assembly. Ø, as supplied.
12FE,12FF	the low, high address of the next available source file storage location. These locations are initialized to the address of the start of the work space by the INIZ command and, thereafter, are automatically updated by the Editor.

Note: It is possible to carry the Assembler's symbol table forward from one assembly to another. To do so, exit the Assembler after the first assembly and enter the machine language monitor by "M". Change location Ø855 from ØA to 18 and read out the contents of locations ØØØA and ØØØB. Enter the values from those locations into locations 12FA and 12FB, respectively.

Then re-enter the Assembler by typing ".13ØØG". Now the symbols from the first assembly will remain in the symbol table for reference during the next assembly. Likewise, the symbols from the first and second assemblies will remain for the third assembly, etc. If you want to eliminate all but the symbols from the first assembly, exit the Assembler and immediately re-enter it by typing "M13ØØG". To restore normal operation of the Assembler change location Ø855 back to ØA. This will cause the symbol table to be cleared at the beginning of each assembly.