# S-C Macro Assembler

## VERSION 2.0

**S-C Software Corporation**
2331 Gus Thomasson, Suite 125
P.O. Box 280300
Dallas, Texas 75228
(214) 324-2050

# S-C Macro Assembler Version 2.0

Here is your copy of the latest upgrade to the S-C Macro Assembler. The most important new feature of Version 2.0 is the capability of assembling the enhanced instruction sets of the 65C02 (both standard and Rockwell editions), 65802, and 65816 microprocessors. Next in importance is the revision of I/O handling to make it easier for you to customize the I/O driver for any sort of 80-column card you may have. 80-column drivers are included for the //e, //c, Videx, and STB-80 systems. In addition, Version 2.0 also includes many small enhancements which will be detailed herein. This upgrade packet also describes the additional features added with versions 1.1 and 1.2.

The disk contains two versions of the S-C Macro Assembler: one which loads at $1000, and another which loads at $D000. These versions are configured with a 40-column driver which will work in any Apple II series computer having at least 48K ($1000 version) or 64K ($D000 version) of RAM. I recommend, although it is not absolutely necessary, that Applesoft be resident in ROM on the motherboard.

The HELLO program checks to see whether you have an Apple //e, //c, or older version. If it is not a //e or //c, HELLO next presents a menu for you to choose a 40-column, Videx, or STB-80 version. Following that choice, you are presented with a menu to select the $1000 or $D000 version of the assembler. With all choices made, HELLO loads the selected version of the assembler, and then BLOADs the selected driver (if any). The $D000 version is more complicated to load, so it is controlled by an EXEC file.

As is true of all S-C products, the Version 2.0 disk is not copy-protected in any way. Make a backup copy now, and store the original in a safe place. The standard Apple COPYA is included on our release disk for your convenience. You may also use the FID program to make copies of individual files on your working disks.

You will probably wish to move the version of the assembler you usually use, with your favorite driver and EXEC loader, to your working disks. Do it!

It is also possible and advantageous to burn the assembler into EPROM. Some owners have burned it into 2716's for installation into an Apple firmware card; others have burned it into 2764's or a 27128 to plug into the SCRG quikLoader. We have done the latter too, and you may purchase a 27128 from us with Version 2.0 in it. Call for details.

## DOS Version

The release disk contains a slightly modified version of DOS 3.3. It is not the latest version of DOS 3.3 as distributed by Apple (which included some "final" patches for the APPEND command). Our version includes instead some patches which speed up the LOAD, BLOAD, and RUN commands tremendously. Of course, the S-C Macro Assembler will run with any version of Apple DOS 3.3, with or without our patches. However, it may not work with some of the enhanced DOS products. You should have no trouble with ProntoDOS, DavidDOS, or The DOS Enhancer; but some other brands, including DiversiDOS, are known to conflict with our use of DOS when you use the .TF assembler directive.

Version 2.0 is compatible with the Corvus hard disk system; it has not been tested with other brands. If there is a problem with other brands, it will occur during assembly of programs which use both the .IN and .TF directives with the files residing in different volumes on the hard disk.

## I/O Drivers

The copies of the Version 2.0 on the release disk include 40-column display drivers. The release disk also includes the source and object code for three 80-column drivers. The HELLO program will automatically BLOAD the 80-column driver of your choice. They are up to 244 bytes long, and load at either $3700 with the $1000-based assembler or $F700 with the $D000-based assembler.

Once you have the assembler in memory with your selected driver, you could BSAVE the composite as your own private version. Use your own file name, an address of $1000 or $D000, and a length of $27F4.

I have attempted to make operation identical regardless of which driver is in place. Nevertheless, there are still some differences. The constraints of 244 bytes per driver, sometimes "crazy" firmware, and the wide variety of features have meant compromises. Since you will probably settle on just one version, the minor differences between it and the ones you do not use should cause no problems.

1. //e-//c Driver:

The best version is the //e version, running in either a //c or in a //e with the new CDEF ROM set (copyright 1984). All features work well with this configuration. When the assembler with this driver is loaded, the screen stays in whatever mode was already set. If you were in 40-column mode, it stays in 40-column; if in 80, it stays it 80. Once in the assembler, you can go to 80-columns by typing PR#3, or back to 40-columns by hitting ctrl-RESET. In a //e with the older firmware, you can also revert to 40-columns by typing esc-crtl-Q. In a //c or a //e with the new firmware, esc-ctrl-Q does nothing; however, you can revert to 40-columns by typing PR#0.

In a //e or //c, you may leave the assembler by typing the FP command. If you are using the $D000-based version, you may return to the assembler with the INT command. (The INT command cold starts the assembler, just as it would cold-start Integer BASIC if that language were loaded into the $D000 area. This means any source program which may have been in memory earlier is cleared.)

If you wish to engage a printer, first get into the 40-column mode. Then type PR#1 (or PR# and whatever slot your printer card is in). After the printing is complete, you may return to 80-column mode. Usually, 80-column screen display and printer interfaces should not be mixed.

There is one thorny configuration that you may possibly have: The older //e firmware with the MouseText character generator ROM. With this configuration, inverse capital letters appear on the screen as MouseText icons. The alternative is to turn off the ALTCHRSET mode, in which case inverse letters appear as flashing letters. If this is your configuration, hopefully you will upgrade your firmware as soon as Apple's upgrade kit is available.

The older //e firmware handled the "escape" key in a very difficult manner, so that I cannot support my own escape commands within the driver. Therefore the esc-L, esc-S, and esc-U commands do not function with the older firmware. However, you can get the same results by using the open-apple key instead of the escape key with these three commands.

The DELETE key on the Apple //e and //c generates the code $FF when it is pressed. This code is interpreted as a backspace, the same as the left arrow ($08).

2. STB-80 Driver:

The STB-80 was one of the best 80-column cards on the market. STB Systems no longer makes it though, because they have opted for the IBM marketplace. I have one, and so do many of you, so I wrote a driver for it.

When you load the assembler with the STB-80 driver, it automatically switches to 80-column mode. It is not possible to switch back to 40-column mode in the assembler. To leave the assembler, type the FP command and then hit RESET or ctrl-RESET.

If you wish to do some printing, turn on your printer with the PR#slot command. The cursor will remain active on the screen, but you will see no characters as you type. The printer will remain connected until you use the NEW command, finish an assembly, or get a syntax error in a command line. To dis-connect on purpose without assembling or erasing your source program, purposely generate a syntax error by typing "X" and a RETURN.

3. Videx Driver:

The Videx card that I have is version 2.4, with the softswitch. The firmware is such that to offer the same features you are used to in the S-C 40-column version I have to directly call two internal firmware routines, and to directly program the cursor generator on the Videx board. This means that my driver may not work with your Videx card, if your card has different firmware at these two addresses. It also means that my Videx driver will almost certainly not function with so-called "Videx-compatible" cards.

The two internal routines are CHRGET (at $C39E in version 2.4) and NTSHFT (at $C870 in version 2.4). If you find the corresponding routines in your Videx firmware, you should be able to substitute the addresses in my driver and reassemble it. If you need assistance, call us.

Printer operation and leaving the assembler are the same as with the STB-80 card. Of course, if you have no "soft switch" with your Videx card, you will need to re-enter 80-column mode to get any display after leaving the assembler.

4. Writing your own driver:

You may customize the existing drivers, or write your own. All of the source code for my three drivers is on the release disk, well-commented. Follow the same outline, and you will succeed. Assemble two versions, one with LOCATION .EQ $1000 and the other with LOCATION .EQ $D000.

Remember that the driver should be no longer than 244 bytes ($3700-37F3 and $F700-F7F3). The last 12 bytes in the driver page are reserved for use by the Laumer Research Full Screen Editor. Of course, if you are quite sure you will never use the Full Screen Editor, then you could use these additional 12 bytes in your driver.

If you have any questions about drivers, call us at (214) 324-2050. When you finish one for another 80-column card or perhaps an Apple clone, you might consider sending it to us for others who might have the same needs.

## Enhancements at the Editor Level

1. Shorthand Features:

Previous versions of the S-C Macro Assembler supported three "shorthand" features: ctrl-E as a handy synonym for "EDIT", esc-L in column 1 to trace over a file name and automatically LOAD it from disk, and esc-L after a line number to generate a star-dash line.

Version 2.0 continues to support those, and adds three more. Ctrl-C in column 1 will spell out "CATALOG" and wait for you to type RETURN, or append slot and drive data and then type RETURN.

Esc-S in column 1 is an Automatic-SAVE command. It depends for operation on your cooperation: you must place a special comment line somewhere near the beginning of your source program (as one of the first ten lines). This comment line has the form:

```
      1000 *HHHHHHSAVE filename
```

The "H" characters are actually ctrl-H characters, which you type in by a series of six ctrl-O,ctrl-H pairs. As you are typing they will appear in inverse on the screen. If you LIST the line, the ctrl-H characters will make the "SAVE filename" part print over the top of the line number, so all you will see is:

```
      SAVE filename
```

When you type esc-S, the first ten lines will be scanned looking for a comment line with a letter "S" following the "*" (any arbitrary characters may separate the "*" and "S"). If such a line is found, it will be listed on the screen. Then the line as it appears on the screen will be picked up and placed in the input buffer, and the cursor placed at the end of the line. You may then type RETURN to SAVE the file. Or you may append slot and drive data and then type RETURN. Or you my type ctrl-X to abort the command.

2. Enhancements to the COPY Command:

We have slightly modified the way the COPY command works. In prior versions a range of lines copied retained the original line numbers, even though that meant you had out of sequence numbers in memory. The new version generates a series of new line numbers for the copied lines. The new copy of the lines will all have the same line number as the target line of the COPY command. For example, COPY 1200,1240,1520 would copy lines 1200-1240 just before line 1520, and assign all the new lines the number 1520.

The new version also allows the option of deleting the original lines, turning COPY into a move command. After the lines are copied, the question "DELETE ORIGINAL?" is asked; a "Y" response will cause the original lines to be deleted. Any other response will cause the original lines to remain.

After a COPY operation is complete, you should RENUMBER the source lines. However, it need not be done immediately. There are cases when it is advantageous to postpone the RENUMBERing a short while.

For example, suppose I want to copy three separate blocks so that they end up one after the other, all before a given spot in the source code...

```
:COPY 3000,3150,4900
DELETE ORIGINAL? Y
:COPY 2700,2720,4900
DELETE ORIGINAL? Y
:COPY 6750,7040,4900
DELETE ORIGINAL? Y
:REN
```

3. Case Toggle

If you are using the 40-column version, or either Videx or STB-80 versions, you can turn the upper-case lock off and on by typing ctrl-S. On the //e or //c, use the case lock button on the keyboard.

4. Linkage to the Full Screen Editor:

We have simplified the linkage to the Laumer Research Full Screen Editor. We built in a trap for the "/" command character which switches you over to the FSE, so that the patch code which must be loaded at the end of page $F7 is much shorter. The last 12 bytes only of that page are now used for the patch, from $F7F4-$F7FF. (The lower 244 bytes of that page are used for the I/O driver.)

Two files are included on our release disk which when EXECed will load in the FSE and patch the assembler for its use: "LOADER FSE" and "LOADER FSE & ASM".

5. New User Vectors:

We added several new user vectors for your own enhancements. These are in addition to the PRT and USR commands already included in previous versions.

In case you would like to add another escape command, we left an opening in the table. Esc-U (or open-apple-U on //e) vectors to a JMP instruction at $D00C. Change the jump address so that it jumps to your own escape handler code, and you will gain control anytime esc-U is typed during line input. Terminate your escape handler code with an RTS opcode. When you get control, the line as it has been typed so far is in the buffer starting at $200, and the number of characters so far is in the X-register.

We also added a single character command similar to the "&" in Applesoft. If you type a "." as the first character of a command line, then when you type the RETURN key control will branch to a vector at $D00F. The instruction at $D00F is a JMP instruction. Insert your own address into that instruction, and you can decode the entire line in whatever manner you wish.

Because of the addition of these vectors, some other vectors and customization data items have moved. Here is the latest table of vectors and parameters:

| $1000 Based | $D000 Based | Description | Contents |
|---|---|---|---|
| $1000 | $D000 | Hard Entry | JMP HARD.INIT |
| $1003 | $D003 | Soft Entry | JMP SOFT |
| $1006 | $D006 | USR Vector | JMP SOFT |
| $1009 | $D009 | PRT Vector | JMP SOFT |
| $100C | $D00C | Esc-U Vector | JMP RDL.ERR |
| $100F | $D00F | "." Vector | JMP SOFT |
| $1012 | $D012 | Object Vector | JMP STORE.OBJECT.BYTE |
| $1015 | $D015 | .US Vector | JMP COMMENT |
| $1018 | $D018 | Tab Character | .DA #$89 (Ctrl-I) |
| $1019 | $D019 | Tab Settings | .DA #14,#18,#27,#32,#0 |
| $101E | $D01E | Esc-L Char | .AS -/-/ (star-dash) |
| $101F | $D01F | Compression Limit | .DA #4 |
| $1020 | $D020 | Wild Card Char | .DA #$17 (Ctrl-W) |
| $1021 | $D021 | Char Out | JMP MON.COUT ($FDED) |
| $1024 | $D024 | Low Mem Unprotect | .DA $0000 |
| $1026 | $D026 | High Mem Unprotect | .DA $0000 |
| $1028 | $D028 | LDA # Opcode | .HS A9 |
| $1029 | $D029 | Starting page for symbol table ($38 or $10) | |

6. New Insert Character for EDIT:

When editing a line with the EDIT command, the insert mode is now invoked by ctrl-A (mnemonic for ADD), rather than ctrl-I. This was done because the TAB key on the //e and //c produces a ctrl-I code. We wanted TAB to mean tab! The TAB or ctrl-I keys now perform a clear-to-tab function when operating under the EDIT command. Skip-to-tab is still invoked by ctrl-T.

7. Full 5-digit Line Numbers:

Line numbers may now have up to five digits, in the range from 0 through 65535. Versions prior to 1.1 restricted line numbers to the range 0-9999. If the numbers are less than 10000, they will print as four-digit numbers with leading zeroes if necessary. Larger numbers, of course, will print as five-digit numbers.

8. Improved HIDE Operation:

The HIDE command now performs an automatic MERGE before hiding the current section of source code. This allows a string of LOAD filenamel, HIDE, LOAD filename 2, HIDE, ...MERGE commands to merge a lot of source files together.

9. To make room for all these enhancements, cassette tape LOAD and SAVE commands have now been removed. We bet hardly a soul will miss them!

## Enhancements at the Assembler Level

1. .OP Directive:

The .OP directive is used to enable the assembly of a particular instruction set. I suppose "OP" could stand for either "OPcodes" or "OPtion". The normal default case is the instruction set of the plain unadorned 6502.

If you wish to assemble Sweet-16 opcodes, you now need to tell the assembler so in advance by a line like:

```
       1000    .OP SW16
```

This has the advantage of catching those cases in which a typing error such as BM1 instead of BMI is made. In previous versions of the assembler, the Sweet-16 BM1 opcode would be assembled, with no error message, even though you expected the 6502 BMI opcode. In version 2.0 you would not have enabled Sweet-16 assembly, so the typing error would be caught.

Here are all the possible values which may be used with the .OP directive:

```
       .OP 6502
       .OP SW16
       .OP 65C02
       .OP 65R02
       .OP 65802
       .OP 65816
```

The .OP processor actually only scans for several key characters. If none of them are found, the 6502 mode is set. If "S" is found, Sweet-16 mode is set. If "C" is found, the normal 65C02 mode is set. If "R" is found, the Rockwell extended 65C02 mode is set. If "8" is found, the 65802/65816 mode is set.

Two aliases are included in the opcode table, by popular demand. "BGE" is an alias for "BCS", since "BCS" is often used to mean "Branch if Greater than or Equal". Likewise, "BLT" is and alias for "BCC", meaning "Branch if Less Than".

All of the modes include as a subset the standard 6502 opcodes and addressing modes. The following tables indicate which opcodes and addressing modes are added with each .OP selection.

65C02 Mode -- adds the following new opcodes with the addressing modes shown:

| | |
|---|---|
| BRA reladdr | Branch Always |
| PHX | Push X-register |
| PHY | Push Y-register |
| PLX | Pull X-register |
| PLY | Pull Y-register |
| STZ zp | Store Zero |
| STZ zp,X | Store Zero |
| STZ abs | Store Zero |
| STZ abs,X | Store Zero |
| TRB zp | Test and Reset Bits |
| TSB zp | Test and Set Bits |

The 65C02 mode also adds the following addressing modes to 6502 opcodes:

| | |
|---|---|
| BIT #val8 | Test Bits |
| BIT zp,X | Test Bits |
| BIT abs,X | Test Bits |
| INC | Increment A-register |
| DEC | Decrement A-register |
| JMP (abs,X) | Jump indexed indirect |
| ADC (zp) | Direct Indirect mode |

(also for AND, CMP, EOR, LDA, ORA, SBC, and STA)

65R02 Mode -- includes all of the 65C02 opcodes and addressing modes, and adds four more opcodes:

```
SMB bit,zp
RMB bit,zp
BBR bit,zp,reladdr
BBS bit,zp,reladdr
```

These actually use up 32 opcode values, because the bit # (0-7) becomes part of the opcode byte.

----------------------

| | |
|---|---|
| Abbreviations: | reladdr...8-bit relative address |
| | zp........8-bit address in page zero |
| | abs.......16-bit address |
| | val8......8-bit immediate value |
| | bit.......3-bit bit number, 0-7 |

65802/65816 Mode -- includes all of the 65C02 opcodes and addressing modes, and adds the following new opcodes with the addressing modes shown:

| | |
|---|---|
| PEA val16 | Push 16-bit value on stack |
| PEI val8 | Push 8-bit value on stack |
| PER longreladdr | Push 16-bit relative address |
| BRL longreladdr | Branch Always, 16-bit reladdr |
| JML (abs) | Jump Long Absolute Indirect |
| JSL longabs | Jump Long Absolute |
| RTL | Long Return from Subroutine |
| MVP longabs,longabs | Block Move Up |
| MVN longabs,longabs | Block Move Down |
| PHB | Push Data Bank Register |
| PLB | Pull Data Bank Register |
| PHD | Push D-register |
| PLD | Pull D-register |
| PHK | Push Program Bank Register |
| REP #val8 | Reset Status Bits |
| SEP #val8 | Set Status Bits |
| TCD | Transfer C to D |
| TDC | Transfer D to C |
| TCS | Transfer C to S |
| TSC | Transfer S to C |
| TXY | Transfer X to Y |
| TYX | Transfer Y to X |
| XBA | Exchange B and A |
| XCE | Exchange Carry-bit with Emulation-Bit |
| COP | Co-Processor Interrupt |
| STP | Stop Clock until RESET |
| WAI | Stop Clock until Ready plus NMI or IRQ |
| WDM | No-operation, reserved for future systems |

----------------

Abbreviations:

| | |
|---|---|
| val8.......... | 8-bit immediate value |
| val16......... | 16-bit immediate value |
| longreladdr... | 16-bit relative address |
| abs........... | 16-bit address |
| longabs....... | 24-bit address |

[redacted]

The 65802/65816 mode also adds the following new addressing modes to 6502 opcodes:

To the ADC, AND, CMP, EOR, LDA, ORA, SBC, and STA opcodes, four new modes:

| | |
|---|---|
| ADC val8,S | Stack Relative |
| ADC (val8,S),Y | Stack Relative Indirect Indexed |
| ADC >(zp) | Direct Indirect Long |
| ADC >(zp),Y | Indirect Indexed Long |

To the ADC, AND, CMP, EOR, LDA, ORA, and SBC opcodes, a new 16-bit immediate mode. This is not a true mode, because the opcodes values are the same as the 8-bit immediate opcodes. At execution time a P-register bit determines whether one or two bytes of data will be used. The syntax in the S-C Macro Assembler for 8- or 16-bit immediate operands is as follows:

8-bit Immediate

| | |
|---|---|
| LDA #expr | low 8-bits |
| LDA /expr | mid 8-bits of 24-bits |
| LDA ^expr | high 8-bits of 24-bits |

16-bit Immediate

| | |
|---|---|
| LDA ##expr | low 16-bits |
| LDA //expr | mid 16-bits of 32-bits |
| LDA ^^expr | high 16-bits of 32-bits |

We decided to use the double-delimiter to indicate 16-bit immediate values so that it would be readily apparent upon reading your assembly source code what you intended. An alternative we considered was to use a new directive to tell the assembler whether to assemble 8- or 16-bit immediate values. We thought that would lead to more programming bugs as the simple reading of a source line would not indicate which mode was being used.

Note that you really only need the # and ## modes, because you can get the others by dividing the expression by 256 one or more times.

| | |
|---|---|
| JMP longabs | Jump with 24-bit address |
| JSR (abs,X) | Jump to Subroutine Indexed Indirect |

[ Strictly speaking, the 65802 level probably should not allow the long addressing modes. However, Version 2.0 makes no distinction between 65802 and 65816 at this time. ]

------------------

Abbreviations:

| | |
|---|---|
| val8...... | 8-bit immediate value |
| zp........ | 8-bit address |
| expr...... | 32-bit expression |
| abs....... | 16-bit address |
| longabs... | 24-bit address |

## 2. Expressions:

In previous versions of the S-C assemblers the operand expressions were limited to 16-bits. Since the 65816 uses a 24-bit address bus, this is no longer adequate. We decided to change to 32-bit expressions, even though 24 might have been enough. Up to 24 bits are meaningful in 65816 instructions, and up to 32 bits are meaningful in .DA directives.

The .DA directive already allowed you to generate either 8-bit or 16-bit data values. We have now added the capability of generating 24-bit and 32-bit values, using this syntax:

```
     .DA #expr         8-bits (low-order byte)
     .DA /expr         8-bits (next-to-lowest byte)

     .DA expr         16-bits (low-order 16 bits)
     .DA expr/256     16-bits (middle 16 bits of 32)
     .DA expr/65536   16-bits (high-order 16 bits)

     .DA <expr        24-bits (low-order 24 bits)
     .DA <expr/256    24-bits (high-order 24 bits)

     .DA >expr        32-bits
```

In all the multiple byte cases, the bytes will be stored highest-byte first; this is the normal 6502 way.

Of course, you may put more than one value on a single .DA line, connected by commas, and you may mix sizes on the line.

```
               1010 *--------------------------------
0800- 78       1020 BYTE    .DA #$12345678          LOW BYTE
0801- 56       1030         .DA /$12345678          2ND BYTE
0802- 56       1040         .DA #$12345678/256      2ND BYTE
0803- 34       1050         .DA #$12345678/65536    3RD BYTE
0804- 12       1060         .DA /$12345678/65536    4TH BYTE
0805- 12       1070         .DA #$12345678/65536/256   4TH BYTE
               1080 *--------------------------------
0806- 78 56    1090 WORD16  .DA $12345678          LOW
0808- 56 34    1100         .DA $12345678/256      MIDDLE
080A- 34 12    1110         .DA $12345678/65536    HIGH
               1120 *--------------------------------
080C- 78 56 34 1130 WORD24  .DA <$12345678        LOW
080F- 56 34 12 1140         .DA <$12345678/256    HIGH
               1150 *--------------------------------
0812- 78 56 34
0815- 12       1160 WORD32  .DA >$12345678
               1170 *--------------------------------
0816- 11 33 22
0819- 66 55 44
081C- AA 99 88
081F- 77       1180         .DA #$11,$2233,<$445566,>$778899AA
               1190 *--------------------------------
```

We also added three operators to those you may use in operand expressions. Boolean operations of logical product (AND), or (OR), and exclusive-or (EOR) are now available. Use the following operator characters:

```
AND -- &
 OR -- ! or |
EOR -- ^
```

Here are some examples:

```
$12345678&$F0F0F0F0 is $10305070
$12345678|$F0F0F0F0 is $F2F4F6F8
$12345678^$F0F0F0F0 is $E2C4A688
```

Binary constants are now supported. The syntax is "%11000011101" (up to 32 bits). To make it easier to read the binary constants, you may include optional periods as visual separators between the binary digits or groups of digits. Here are some examples:

```
0800- AD 08 01  1000       LDA %1.0000.1000
0803- 29 7F     1010       AND #%01111111
0805- 01 80     1020       .DA %1000000000000001
0807- 34 12     1030       .DA %0001.0010.0010.0100
```

ASCII literals with the high-bit set are now allowed, and are signified with the quotation mark. Note that a trailing quotation mark is optional, just as is a trailing apostrophe with the low-bit-zero ASCII literals.

```
0800- A9 58     1000       LDA #'X
0802- A9 D8     1010       LDA #"X
0804- C1 42 E3  1020       .DA #"A",#'B',#"c"
```

3. Force zero page, absolute, or long modes:

The 6502 assembly language has several ambiguous modes. For example,

```
1000 VALUE  .EQ $05
1010        LDA VALUE
```

could be assembled in two different ways, both perfectly valid. Since VALUE is in page zero, "A5 05" is one possible way to assemble it. However, "AD 05 00" is also perfectly valid. The assembler normally decides which mode it can use. In this case the assembler would use the zero page form, "A5 05". If you want to force the assembler to use the longer form, you could do so by putting the .EQ line later in the program. That works, but it is not a desirable technique.

Borrowing syntax from some other assemblers, I have added the capability to force the mode you desire. If you write a ">" character before the operand, the long mode will be forced. You can also use "<" to force the zero page mode. If you are writing code which will be assembled to execute inside page zero, you may find cases in which you need the "<" capability.

Big Mac, Merlin, and some other assemblers share this syntax. For some reason Apple's ToolKit uses the ">" and "<" in exactly the opposite sense.

The long mode may be forced by prefixing ">>" to the operand. Here are some examples:

```
                      1000        .OP 65816
                      1010 *---------------------------
000800- A5 03         1030        LDA 3
000802- AD 03 00      1040        LDA >3
000805- AF 03 00 00   1050        LDA >>3
000809- 4C 03 00      1060        JMP 3
00080C- 5C 03 00 00   1070        JMP >>3
000810- AD 34 12      1080        LDA $1234
000813- A5 34         1090        LDA <$1234
000815- AF 56 34 12   1100        LDA $123456
000819- AD 34 12      1110        LDA >$123456
00081C- 5C 56 34 12   1120        JMP $123456
000820- 4C 34 12      1130        JMP >$123456
```

4. .BS with Fill Byte:

In previous versions of the assembler, the .BS directive wrote zeroes on the target file. If object code was stored directly in RAM, no fill value at all was stored. In the new version, zeroes will be stored in RAM. For example, ".BS 5" is equivalent to ".HS 0000000000".

We have also added an optional parameter so that you can specify what the fill byte will be. Here are the syntax and some examples:

```
                    .BS count,value
       1000 SYMBOL .BS 4,$A0    4-byte variable,
                                 filled with $A0 bytes
       1010 VALUE  .BS 2,$FF    2-byte variable,
                                 filled with $FF bytes
       1020 BUFFER .BS 32       32-byte variable,
                                 filled with $00 bytes
```

A RANGE ERROR will be generated if the number of bytes is negative, or greater than 32767.

5. Object Code Emission Vector:

A new user vector has been added which allows you to gain control over each byte of the object code as it is emitted from pass two of the assembler. Ordinarily the object code would be either stored at the target address in RAM or written on the target file. With this vector you may write a program to do other things with the object code. For example, one customer uses this vector to funnel code through a serial port to another computer. The vector is at $1012 or $D012, depending on where you have loaded the assembler. It consists normally of "JMP STORE.OBJECT.BYTE"; put your own address into the instruction, and you have total control.

6. Optional Separators in .HS Directive:

The .HS directive now allows optional "." characters before and after each pair of hex digits. This makes it easier to count the bytes, and to align the bytes with comments on the lines above or below the .HS lines.

7. Deeper .DO Nests:

.DO -- .FIN sections can now be nested to 63 levels, rather than the limit of 8 established in Version 1.0.

8. Additonal Comment Character:

Comment lines may begin with either "*" or ";". This increases compatiblity with other brands of assemblers. Some people prefer "*", some prefer ";". Believe it or not, some even use both!

9. Expanded and Flexible Memory Protection:

Memory protection during assembly has been expanded. The assembler now protects the ranges $001F-$02FF and $03D0-$07FF as well as the symbol tables, the assembler itself, and DOS.

New user parameters have been added to allow you to selectively override memory protection. You enter the first and last address of any range you want to UN-protect in these two parameters. The beginning address of the range goes at $1024 and $1025, low byte first ($D024 and $D025 in the high memory version). The end address of the un-protected range goes at $1026-$1027 ($D026-$D027).

10. .SE Directive

The .SE directive has been added, to allow re-definable symbols. Symbols which are originally defined by the .SE directive may be re-defined within the same assembly by additional .SE lines. This directive allows a counter within macro definitions, as shown in the file "EXAMPLE: .SE DIRECTIVE".

Labels defined with a .EQ directive, or by simply appearing in the label field, cannot be re-defined.

11. Assembly of Separate Phases:

The .PH and .EP directives are now available, to start and end a phase. With these directives you can assemble a section of code that is intended to be moved and exectued somewhere else, without having to create a separate Target File. .PH <expr> effectively sets the origin to <expr>, but keeps the target address unchanged. When the .EP directive is encountered, the origin is reset to match the target address.

```
0800- AD 03 9D  1000 START LDA DATA
0803- 60        1010       RTS
                1020 *-------------------------
                1030       .PH $9D00
9D00- A9 08     1040 PATCH LDA /ADDR
9D02- 60        1050       RTS
9D03-           1060 DATA  .BS 1
                1070       .EP
                1080 *-------------------------
0808- 00 9D     1090 ADDR  .DA PATCH

SYMBOL TABLE

0808- ADDR    9D03- DATA    9D00- PATCH   0800- START
```

Notice that the object code column and the symbol table show the code to be in different locations, but if you examine memory you will find all the code together, starting at $0800 and running through $0809:

```
$800.809
0800- AD 03 9D 60 A9 08 60 00
0808- 00 9D
```

12. Dummy Sections:

We added .DUMMY and .ED directives to start and end a dummy section. A dummy section assembles, but no object code bytes are produced. Dummy sections are useful when specifying data blocks, or when you want to run as assembly for syntax checking without generating code.

.DUMMY and .ED are equivalent to the DSECT and DEND directives in Apple's ToolKit Assembler, with the exception that the dummy origin is not automatically set to $0000. Any .OR directives within a dummy section will only be effective that section: .DUMMY saves the current origin, and .ED restores it.

```
0800- 34 12     1000 ADDR  .DA LABEL
                1010 *-------------------------
                1020       .DUMMY
                1030       .OR $1234
1234- AD 00 08  1040 LABEL LDA ADDR
1237-           1050 NEXT  .BS 1
1238-           1060 AGAIN .BS 1
                1070       .ED
                1080 *-------------------------
0802- AD 35 12  1090       LDA NEXT
```

If you include a .TF directive inside a dummy section, an amazing and useful thing happens: the assembly listing is written out to the target file! Be careful with the surprising feature. If you have several .TF lines in a source program, and some are in dummy sections, you may get some rather unpredictable results.

13. Change to Relative-Branch Expressions:

We changed the way the relative branches are assembled, so that "*" in expressions is equal to the location of the opcode byte. In Macro 1.0 and earlier versions, "*" equaled the address of the offset byte, which was non-standard. However, even with this correction, we do not recommend that you use "*+anything" or *-anything" address expressions in relative branches: it is a dangerous practice that almost always leads eventually to bugs. Use local labels instead.

14. Additional Option on .LIST Directive:

.LIST CON allows you to include those lines that are skipped over by a .DO -- .FIN section in the assembly listing.

| | |
|---|---|
| .LIST CON | Show excluded lines |
| .LIST COFF | Omit excluded lines |

Here is an example:

```
                1000      .LIST CON
                1010      .DO 0    (FALSE)
                1020      NOP      (LISTED, BUT NOT ASSEMBLED)
                1030      .ELSE
0800- EA        1040      NOP      (LISTED AND ASSEMBLED)
                1050      .FIN
```

Without the .LIST CON, line 1020 would be omitted from the listing. The default condition is .LIST COFF.

## Brief History of S-C Assembler Versions

We have been working on the S-C Assemblers for over six years now, and it is interesting to see the progress we have made.

| | |
|---|---|
| Apr 78 | began working on assembler<br>various pre-releases (first manual only 1 page!) |
| Aug 78 | original tape version, $25<br>4-char labels<br>$1000-$1BFF |
| Jul 79 | Disk Version 3.2, $35. |
| Jul 80 | Disk Version 4.0, $55.<br>source code $95, first sold Oct 81<br>Paul Schlyter showed us how to move it to language card RAM.<br>Don Taylor and Rip Toren showed us how to use the Videx card. |
| Feb 82 | Macro 1.0, $80. |
| Apr 83 | Macro 1.1, $92.50<br>source code $100, first sold Apr 84 |
| Nov 83 | Macro 1.2 (un-official pre-release) |
| Nov 84 | Macro 2.0, $100<br>65C02 and 65816 assembly<br>$1000-$37FF or $D000-$F7FF |

Related products

| | |
|---|---|
| Oct 80 | started "Apple Assembly Line" |
| Nov 80 | Rak-Ware, Decision Systems, and Lee Meador disassemblers |
| Mar 81 | Rak-Ware DISASM version 2.0 |
| Apr 81 | Rak-Ware XREF for S-C 4.0 |
| May 81 | Rak-Ware utilities for 4.0 |
| Jan 83 | Attempted to produce Apple /// version, but did not ever finish the project. |
| Mar 83 | Laumer Research Full Screen Editor |
| May 83 | S-C XREF |

[redacted]

Cross Assemblers

| | | | |
|---|---|---|---|
| Nov 80 | 6800 | (4.0) | Bob S-C |
| Oct 81 | 6809 | (4.0) | Chris Wiggs |
| Jun 82 | 6800 | (Macro) | Bob S-C |
| Jun 82 | 6809 | (Macro) | Bobby Deen |
| Jul 82 | Z-80 | (Macro) | Bobby Deen |
| Sep 82 | 68000 | (Macro) | Bobby Deen |
| Dec 82 | 65C02 | (Macro) | Bob S-C |
| Dec 82 | 8048 | (Macro) | Bobby Deen |
| Jan 83 | 6805 | (Macro) | Bobby Deen |
| Mar 83 | 8051 | (Macro) | Bobby Deen |
| Mar 83 | 1802 | (Macro) | Bobby Deen |
| May 83 | PDP11 | (Macro) | Bobby Deen |
| Jun 83 | 8085 | (Macro) | Bobby Deen |
| Nov 83 | 6301 | (Macro) | Bob S-C |
| Mar 84 | Z-8 | (Macro) | Bobby Deen |
| Aug 84 | 1650 | (Macro) | Bob S-C |
| Aug 84 | 1670 | (Macro) | Bob S-C |

Version 4.0 was translated into Japanese by the friendly folk at ESD Laboratories in Tokyo. They published the Japanese version under license agreement for several years.

Minimum Assembler, with "The Fourth Leg of the Apple", published by Dr. Ray Brinker. Dr. Brinker's book/disk introduces the reader to assembly language, Forth, and other fascinating subjects. Plus, you get a working (albeit somewhat stripped-down) assembler, a working Forth, and other useful stuff. All for $50.

Synassembler, on Atari 400/800, converted from our Version 4.0 by Steve Hales and published for several years by Synapse. Now "out of print", as Synapse decided the market was too small.

An unofficial, bootleg copy turned up in the International Apple Core disk-of-the-month series during 1983. This was a slightly modified copy of my original tape version, which found its way from Texas to California by way of Florida and Ontario (Canada).

F-S Macro Assembler, converted from our Macro 1.0 version by Y. Lempereur at FunSoft Inc, and distributed by Stanton Products, 3710 Pacific Ave., Venice, CA 90291 for $50. Phone (213) 821-2425. Operates on any Atari with 48K RAM and a disk drive. Jeff Stanton is selling these to readers of his latest book, "Atari Graphics and Arcade Games".

Mainstay MacASM, also written by Y. Lempereur at FunSoft, on the model of the S-C Macro Assembler, to operate in the Macintosh. This is a full 68000 assembler, for $100 to $150. Available from S-C Software for $100 while supplies last.