General Notice

When using this document, keep the following in mind:

1. This document is confidential. By accepting this document you acknowledge that you are bound by the terms set forth in the non-disclosure and confidentiality agreement signed separately and /in the possession of SEGA. If you have not signed such a non-disclosure agreement, please contact SEGA immediately and return this document to SEGA.

2. This document may include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new versions of the document. SEGA may make improvements and/or changes in the product(s) and/or the program(s) described in this document at any time.

3. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without SEGA'S written permission. Request for copies of this document and for technical information about SEGA products must be made to your authorized SEGA Technical Services representative.

4. No license is granted by implication or otherwise under any patents, copyrights, trademarks, or other intellectual property rights of SEGA Enterprises, Ltd., SEGA of America, Inc., or any third party.

5. Software, circuitry, and other examples described herein are meant merely to indicate the characteristics and performance of SEGA's products. SEGA assumes no responsibility for any intellectual property claims or other problems that may result from applications based on the examples describe herein.

6. It is possible that this document may contain reference to, or information about, SEGA products (development hardware/software) or services that are not provided in countries other than Japan. Such references/information must not be construed to mean that SEGA intends to provide such SEGA products or services in countries other than Japan. Any reference of a SEGA licensed product/program in this document is not intended to state or simply that you can use only SEGA's licensed products/programs. Any functionally equivalent hardware/software can be used instead.

7. SEGA will not be held responsible for any damage to the user that may result from accidents or any other reasons during operation of the user's equipment, or programs according to this document.

NOTE: A reader's comment/correction form is provided with this document. Please address comments to :

SEGA of America, Inc., Developer Technical Support (att. Evelyn Merritt)
150 Shoreline Drive, Redwood City, CA 94065

SEGA may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

SEGA OF AMERICA, INC.
Consumer Products Division

SCU DSP
Assembler
User's Manual

Doc. # ST-240-A-042795

READER CORRECTION/COMMENT SHEET

Keep us updated!

If you should come across any incorrect or outdated information while reading through the attached document, or come up with any questions or comments, please let us know so that we can make the required changes in subsequent revisions. Simply fill out all information below and return this form to the Developer Technical Support Manager at the address below. Please make more copies of this form if more space is needed. Thank you.

General Information:

Your Name _____ **Phone** _____

Document number ST-240-A-042795 **Date** _____

Document name SCU DSP Assembler User's Manual

Corrections:

Chpt.	pg. #	Correction

Questions/comments: _____

Where to send your corrections:

Fax: (415) 802-1717
Attn: Evelyn Merritt,
Developer Technical Support

Mail: SEGA OF AMERICA
Attn: Evelyn Merritt,
Developer Technical Support
150 Shoreline Dr.
Redwood City, CA 94065

1. Overview

The SCU DSP assembler is designed to develop DSP instruction code and to simulate their execution under MS-DOS and UNIX environments. Linking of code is not required since the assembler outputs code in Motorola S format. The DSP assembler requires a substantial knowledge of the hardware; therefore, the user is advised to have a thorough understanding of the DSP hardware prior to use.

2. Running the Assembler

```
dspasm [option] <source filename>
```

1) The following options are available (files can be created only after the program terminates execution without errors.)

-l[Filename]:	Output list
-a[Filename]:	Output data in SH assembler format
-c[Filename]:	Output data in C format
-m:	To use the MODEL M development target

2) There are no default file extensions set for source filenames.
3) Only the errors detected in the initial search are displayed. Correct the errors and assemble the code repeatedly until all errors are eliminated.

3. How to Write a Program

```
[label] [Δoperation [Δoperand]] ... [comment(s)]
```

Ex: `LABEL:  MOV MC0, X  ;  comment(s)`

1) Labels
   - Defined by the programmer, and used as the destination address for the JMP instruction.
   - When writing labels, begin from the first column, or use a colon ":" at the end of the word (ex. **`LABEL:`**).
   - Labels can be as long as 32 characters in length, and upper or lower case English letters, numbers, and underscores(_) may be used. Numbers may not be used as the first character. Also, the labels are not case- sensitive.
2) Operations
   - Write the DSP execution instructions.
   - When writing code that begins with an operation, enter a blank space before the operation.
   - As many as six operand instructions can be listed under one operation (applicable to operand instructions only).
3) Operands
   - List operands required for the execution of operations.
   - Insert a space between operands.

4) Comments
   - Comments can be written to make the program easier to understand.
   - Start comments with a semi-colon ";" and end the comment at the end of the line.

***Note on writing:**
- The basic rule is to write the operation and operand on one line; however, when this is not possible, enter "\" before pressing **Return** to continue on to the next line. To follow an operation after a comment, enter "\" before ";". Also, do not exceed 255 characters per line.
- Operations and operands are not case sensitive; use either upper or lower case English letters.
- Specify **`$xx`** for hexadecimal, **`xxx`** for decimal, and **`%xxxxxxxx`** for binary.
- Output code addresses can be specified by the ORG directive.
- Although the program area in DSP only has a maximum capacity of 256 instructions, it can issue a "warning" and output code containing up to 2048 instructions to facilitate tasks such as the splitting of processes or optimization. However, only the SCU DSP Simulator can support this code. Therefore, it is necessary to edit the code down to its 256 instruction limit during assembly, if the code is actually used in the DSP. Also, note that if the number of address labels exceeds 256 instructions, assignment is not possible with 8-bit values.

* **Note on reserved words:**
- The following names are reserved for operands and may not be used for labels.
  `{ALH ALL ALU M0 M1 M2 M3 MC0 MC1 MC2 MC3 MUL}`

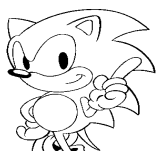
* **Note on numeric operations:**
- The following operators can be used when setting values on labels, or when using numerical values for operands (When the following are used as operands, do not enter any spaces. Ex. **`JMP $+2`** is correctly written, while **`JMP $ + 2`** is incorrect.)

Operators

+	addition
–	subtraction
*	multiplication
/	division
%	remainder
~	bit negation
&	bit product
|	bit sum
^	exclusive bit sum
<<	left shift
>>	right shift

Operator Priority

1.	+ – ~ (monadic operator)
2.	* / %
3.	+ –
4.	<< >>
5.	&
6.	| ^



4. Summary of Instructions

1) Operation instructions:
   NOP AND OR XOR ADD SUB AD2 SR RR SL RL RL8 CLR MOV

2) "Load immediate" instruction:
   MVI

3) DMA instructions:
   DMA DMAH

4) JUMP instruction:
   JMP

5) LOOP BOTTOM instructions:
   BTM LPS

6) END instructions:
   END ENDI

Directive summary:

EQU(=) Defines labels.

ORG Specifies starting address where instructions are located.

ENDS Enter at the end of the program, anything beyond this point is ignored.

IF <numerical value, label>
When the resulting calculated numerical or label value is any value other than 0, the program assembles from that point on to ELSE or ENDIF.

IFDEF <label>
When labels are defined first, the program assembles from that point to ELSE or ENDIF (Up to 16 levels of IF and IFDEF nestings are supported).

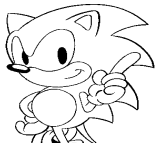
5. Sample Programs

1) Copying internal RAM0 data of the DSP to internal RAM1.

```
;  ——sample (1) start——

COPY_SIZE   =     12          ; Copy size
RAM0_ADR    =     $00         ; Source address
RAM1_ADR    =     $00         ; Destination address

      MOV RAM0_ADR, CT0             ; Set source RAM0 address
      MOV RAM1_ADR, CT1             ; Set destination RAM1 address
      MOV COPY_SIZE-1, LOP          ; Set transfer size-1 in the LOP
                                      register
      LPS                           ; Execute 1 instruction loop
      MOV MCO, MC1                  ; Transfer from RAM0 to RAM1
      ENDI

;  ——sample (1) end——
```

2) Calculating 2 x 3 + 4 x 5. (RAM0 x RAM1 + RAM0 x RAM1 = RAM2)
(Sample 2b is an optimization of 2a)

```
;  ——sample (2a) start——

RAM0_ADR    =     $00         ; Store 2, 4 starting addresses
RAM1_ADR    =     $00         ; Store 3, 5 starting addresses
RAM2_ADR    =     $00         ; Store results at this address

      MOV RAM0_ADR, CT0         ; Set RAM0 address
      MOV RAM1_ADR, CT1         ; Set RAM1 address
      MVI #2, MC0               ; Set “2” in RAM0
      MVI #3, MC1               ; Set “3” in RAM1
      MVI #4, MC0               ; Set “4” in RAM0
      MVI #5, MC1               ; Set “5” in RAM1
      MOV RAM0_ADR, CT0         ; Set RAM0 address
      MOV RAM1_ADR, CT1         ; Set RAM1 address
      MOV RAM2_ADR, CT2         ; Set RAM2 address
      MOV MC0, X                ; Transfer data from RAM0 to RX
      MOV MC1, Y                ; Transfer data from RAM1 to RY
      MOV MUL, P                ; Store the product of RX and
                                  RY at PH, PL
      MOV MC0, X                ; Transfer data from RAM0 to RX
      MOV MC1, Y                ; Transfer data from RAM1 to RY
      CLR A                     ; Set ACH, ACL to “0”
      AD2 MOV ALU, A            ; Store the sum of PH, PL and ACH,
                                  ACL at ACH, ACL
```

```
      MOV MUL, P        ; Store the product between RX
                          and RY at PH, PL
      AD2 MOV ALL, MC2  ; Store the sum of PH, PL and ACH,
                          ACL in RAM2
      ENDI

;  ——sample (2a) end——
;  ——sample (2b) start——

RAM0_ADR   =   $00          ; Store 2, 4 starting addresses
RAM1_ADR   =   $00          ; Store 3, 5 starting addresses
RAM2_ADR   =   $00          ; Store results at this address

                                                  MOV RAM0_ADR, CT0
                                                  MOV RAM1_ADR, CT1
   MVI #2, MC0
   MVI #3, MC1
   MVI #4, MC0
   MVI #5, MC1
                                                  MOV RAM0_ADR, CT0
                                                  MOV RAM1_ADR, CT1
      MOV MC0,X              MOV MC1,Y            MOV RAM2_ADR, CT2
      MOV MC0,X   MOV MUL,P  MOV MC1,Y   CLR A
   AD2            MOV MUL,P                MOV ALU,A
   AD2                                             MOV ALL, MC2
   ENDI

;  ——sample (2b) end——
```

3) Calculating matrix multiplies. (RAM0 X RAM1 = RAM2)

```
 / M 00 M 01 M 02 M 03\ / 1 0 0 x\    /M 00 M 01 M 02 M 03\
| M 10 M 11 M 12 M 13 || 0 1 0 y | → | M 10 M 11 M 12 M 13 |
 \M 20 M 21 M 22 M 23/ | 0 0 1 z |    \M 20 M 21 M 22 M 23/
                       \0 0 0 1 /
```

```
;  ——sample (3) start——

DATA_TOP   =   $10000>>2    ; External memory address is 4 byte units
MAT_SIZE   =   $0C          ; Array size
RAM0_ADR   =   $00          ; Starting address that stores X, Y, Z changes
RAM1_ADR   =   $00          ; Address for array work
RAM2_ADR   =   $00          ; Original array address

;  (Transfers x, y, z translation arrays from external memory to RAM0)
;
           MVI DATA_TOP, RA0
```

```
                                                    MOV RAM0_ADR, CT0
            DMA  D0, MC0, #$02
;
;     (Copy matrix operands from RAM2 to RAM1)
                                                    MOV RAM2_ADR, CT2
                                                    MOV RAM1_ADR, CT1
                                                    MOV MAT_SIZE-1, LOP
            LPS
                                                    MOV MC2, MC1
WAITING:
            JMP TO, WAITING
;
;     (Calculate arrays)


                                                       MOV RAM0_ADR, CT0
                                                       MOV RAM1_ADR, CT1
      MOV MC0, X              MOV MC1, Y
      MOV MC0, X  MOV MUL, P  MOV MC1, Y  CLR A
AD2   MOV MC0, X  MOV MUL, P  MOV MC1, Y  MOV ALU, A  MOV RAM0_ADR, CT0
AD2               MOV MUL, P  MOV MC1, Y  MOV ALU, A  MOV #1, RX
AD2   MOV MC0, X  MOV MUL, P  MOV MC1, Y  MOV ALU, A  MOV RAM2_ADR+3, CT2
AD2   MOV MC0, X  MOV MUL, P  MOV MC1, Y  CLR A       MOV ALL, MC2
AD2   MOV MC0, X  MOV MUL, P  MOV MC1, Y  MOV ALU, A  MOV RAMO_ADR, CT0
AD2               MOV MUL, P  MOV MC1, Y  MOV ALU, A  MOV #1, RX
AD2   MOV MC0, X  MOV MUL, P  MOV MC1, Y  MOV ALU, A  MOV RAM2_ADR+7, CT2
AD2   MOV MC0, X  MOV MUL, P  MOV MC1, Y  CLR A       MOV ALL, MC2
AD2   MOV MC0, X  MOV MUL, P  MOV MC1, Y  MOV ALU, A  MOV RAMO_ADR, CT0
AD2               MOV MUL, P  MOV MC1, Y  MOV ALU, A  MOV #1, RX
AD2               MOV MUL, P              MOV ALU, A  MOV RAM2_ADR+11, CT2
AD2                                                   MOV ALL, MC2
ENDI

; ------- sample (3) end -------
                                                     END
```