To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

To all our customers

Regarding the change of names mentioned in the document, such as Mitsubishi Electric and Mitsubishi XX, to Renesas Technology Corp.

The semiconductor operations of Hitachi and Mitsubishi Electric were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Mitsubishi Electric, Mitsubishi Electric Corporation, Mitsubishi Semiconductors, and other Mitsubishi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Note : Mitsubishi Electric will continue the business operations of high frequency & optical devices and power devices.

Renesas Technology Corp.
Customer Support Dept.
April 1, 2003

User's Manual

M3T-SRA74 V.4.10

User's Manual

Relocatable Assembler for 740 Family

Renesas Electronics
www.renesas.com

Rev.1.00 2003.08

- Microsoft, MS-DOS, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the U.S. and other countries.
- Sun, Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. or other countries, and are used under license.
- Linux is a trademark of Linus Torvalds.
- Turbolinux and its logo are trademarks of Turbolinux, Inc.
- IBM and AT are registered trademarks of International Business Machines Corporation.
- Intel and Pentium are registered trademarks of Intel Corporation.
- Adobe, Acrobat, and Acrobat Reader are trademarks of Adobe Systems Incorporated.
- All other brand and product names are trademarks, registered trademarks or service marks of their respective holders.

Keep safety first in your circuit designs!

- Renesas Technology Corporation and Renesas Solutions Corporation put the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage. Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Renesas Technology product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation, Renesas Solutions Corporation or a third party.
- Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
- All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation and Renesas Solutions Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation, Renesas Solutions Corporation or an authorized Renesas Technology product distributor for the latest product information before purchasing a product listed herein. The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors. Please also pay attention to information published by Renesas Technology Corporation and Renesas Solutions Corporation by various means, including the Renesas home page (http://www.renesas.com).
- When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation and Renesas Solutions Corporation assume no responsibility for any damage, liability or other loss resulting from the information contained herein.
- Renesas Technology semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation, Renesas Solutions Corporation or an authorized Renesas Technology product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
- The prior written approval of Renesas Technology Corporation and Renesas Solutions Corporation is necessary to reprint or reproduce in whole or in part these materials.
- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination. Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
- Please contact Renesas Technology Corporation or Renesas Solutions Corporation for further details on these materials or the products contained therein.

For inquiries about the contents of this document or product, fill in the text file the installer generates in the following directory and email to your local distributor.

¥SUPPORT¥Product-name¥SUPPORT.TXT

Renesas Tools Homepage http://www.renesas.com/en/tools

Introduction

SRA74 is a structured relocatable macro assembler for the 740 Family. SRA74 can produce 740 Family machine language files and debug data files from source programs containing 740 Family structured language, assembly language, or a mixture of the two. This manual describes the functions and use of the four programs included in the SRA74 package.

1. SRA74 structured relocatable macro assembler

2. LINK74 linkage editor

3. LIB74 library

4. CRF74 cross-referencer

5.CV74 M37280 converter

The format of the manual

This manual consists of four separate parts. Each part covers one of the programs in the SRA74 package. As far as possible, the same sequence has been followed in the description for each of the programs. For instance, environmental variables are covered at the end of the chapter on operating method for each program.

- Part 1: SRA74 operation manual
  Describes how to use the SRA74 structured relocatable macro assembler, and how to enter SRA74 source programs.

- Part 2: LINK74 operation manual
  Describes how to use the LINK74 linker, and the part functions.

- Part 3: LIB74 operation manual
  Describes how to use the LIB74 library.

- Part 4: CRF74 operation manual
  Describes how to use the CRF74 cross-referencer.

- Part 5:CV74 operation manual
  Describes how to use the CV74 M37280 converter.

PART 1

RELOCATABLE ASSEMBLER FOR THE 740 FAMILY

SRA74 OPERATION MANUAL

RENESAS
Renesas Technology Corp.

Table of Contents

List of Figures

List of Tables

CHAPTER 1
The Format of the Manual

The SRA74 operation manual contains consists of the following chapters:

- Chapter 2 : Outline
  The basic functions of SRA74 and the files which it generates.

- Chapter 3 : Method of entering the source program
  How to enter the source program used by SRA74.

- Chapter 4 : Assembly language
  The 740 Family assembly language used with SRA74.

- Chapter 5 : Structured entry language
  The structured language used with SRA74.

- Chapter 6 : Pseudo-commands
  Pseudo-commands used with SRA74.

- Chapter 7 : Macro instructions
  Macro instructions used with SRA74.

- Chapter 8 : Operating method
  Method of operating SRA74.

- Appendix A : Error messages
  A list of the error messages used by SRA74 along with clarifications of the content of the messages and appropriate user responses.

- Appendix B : Commands
  All the commands in the 740 Family assembly language which can be used with SRA74.

- Appendix C : The commands listed by addressing mode
  The 740 Family assembly language commands listed for each addressing mode.

- Appendix D : Pseudo-commands
  A list of the contents of each pseudo-command used with SRA74.

- Appendix E : Macro instructions
  A list of the contents of each macro instruction used with SRA74.

- Appendix F : Structured commands
  A list of the contents of each of the structured commands used with SRA74.

- Reserved words : List of reserved words used with SRA74.

Note:
Parts of the programming examples contained in this manual use the sign “\” in place of “¥” to refer to the special page addressing mode. This is because the use of one or other of these signs depends on which operating system is being used. Nevertheless, since the codes are the same either sign can be used.

CHAPTER 2
Outline

SRA74 is the structured relocatable macro assembler for the 740 Family. The assembly language and the structured language are used to convert the source program (to be called the source file from now on) into relocatable files which can be processed by means of LINK74[1] and LIB74[2]. From now on this operation will be referred to as assembly. LINK74 can be used to convert the relocatable files into machine language.

2.1 Functions

The development of large scale software has made necessary the parallel development of functions which permit the sharing of programs for, for example, the exchange of data between groups of engineers or the reuse of existing software.

SRA74 is provided with the following functions to permit the efficient handling of such operations:-

1. The pseudo-command .SECTION enables you to allocate the name of your choice (section name) to each separate area. When linking, section names allocated in this way can be used to specify an address.

2. The number of sections which can be defined within any given file is unlimited thus enabling the setup to be matched to the user system's various ROM, RAM areas.

3. The pseudo-commands .LIB and .OBJ can be used to specify the link target file name in the source program (It will therefore be unnecessary to specify the link target file name during linking).

4. Declaration of the version using the pseudo-command .VER enables the version in the relocatable files to be confirmed during linking.

Other special features of SRA74 functions are outlined below:

1. It is possible to assemble a source file containing a mixture of assembly language and structured language (For assembly the structured language commands are converted into assembly language commands).

[1] Linkage editor for use with 740 Family

[2] Library for use with 740 Family

2. Files can be generated where the structured language has been converted into assembly language (Such files can be optimized by the user).

3. A tag file[3] can be generated to store error details (Assembly errors can be efficiently rectified).

4. The program can be used as an absolute assembler since both ROM and RAM areas can be introduced into a single file (Linking must, however, be used).

5. The programming environment can be defined clearly using the macro function (commands).

6. The editor and cross referencer can be activated by specifying the command parameters during assembly.

7. Source line debugging data can be output by specification of the command parameters during assembly.

2.2 Generating files

SRA74 generates the following five files. Unless specified with a command parameter, these files are placed in the same directory as the source file.

1. Relocatable file (Hereafter referred to as the R74 file)
   - File containing machine language data with the data for relocation.
   - Generates file regardless of contents of command parameters. It will not, however, generate a file in cases where SRA74 has undergone an abnormal termination due, for example, to an assembly error.
   - Output is in the specified directory where the command parameter “-O” has been specified and in the current directory where “-O” has not been specified.
   - Symbol data for symbolic debugging is included. However, with conventional values local symbol data is not output.
   - Local symbol data is output to an R74 file on specification of command parameter “-S”.
   - Source line debug data is output to an R74 file on specification of command parameter “-C” or pseudo instruction “.FUNC”.
   - Use of LINK74 generates an Intel HEX format machine language file.
   - The file attribute is .R74.

2. Assembly language file (Hereafter referred to as the I file)
   - A file which contains the expanded source file macros and structured commands converted to mnemonics.

[3] The name “tag” derives from the use of the word “tag” to label the location of an error or warning.

- File generated in the same directory as R74 files on specification of command parameter “-I”. No file is generated, however, if command parameter “-A” is specified at the same time.
- If the command parameter “-I” is not specified, this file is output as temporary file for assembly to the same directory as the TMP file and deleted automatically at the end of assembly.
- This is used as the input file (source file) for SRA74 pass 2 processing operations. The macro definition area, macro calling area and structured commands are all output as comment lines.
- In the source file a file read in by means of the pseudo-command .INCLUDE is also output with a file name having the attribute .In (n = 0-99).
- The I file should be used to optimize the source file at the assembly language level.
- The file attribute is .I.

3. Print file (Hereafter referred to as the PRN file)
   - This file shows the source file to be processed along with its location address and generation data.
   - The PRN file is generated on specification of command parameter “-L”.
   - Output is in the specified directory where the command parameter “-O” has been simultaneously specified and in the current directory where “-O” has not been specified.
   - When command parameter “-I” is simultaneously specified the structured command is converted to a mnemonic and output underneath the source line.
   - The PRN file should be used for operations such as print and debug.
   - The file attribute is .PRN.

4. The tag file (Hereafter referred to as the TAG file)
   - This file is used to store the error and warning messages generated in the course of an assembly operation.
   - The TAG file is output on specification of command parameter “-E”.
   - The TAG file should be used for reference when correcting editor errors.
   - The file attribute is .TAG.

5. The temporary file (Hereafter referred to as the TMP file)
   - This is a temporary file for use during assembly operations.
   - Output is in the set directory where the environment variable TMP has been set and in the current directory where “TMP” has not been set.
   - This file is automatically deleted when the assembly operation has been completed.
   - The file attribute is .$$n (n = 1-5).

Note:
The R74 file has a binary format and there should therefore be no output to the printer and VDU.

2.2.1 Configuration of the I file

Fig 2.1 gives an illustration of a source file and Figs.2.2/2.3 give examples of I files generated during its assembly. The I file is made up from the following data.

- The parts entered using structured language commands are converted to assembly language commands which are shown beneath the structured language commands. The structured language and macro commands are all output as comment lines. The I file can thus be treated as the source file and the assembly operation carried out by means of SRA74.

```
      .INCLUDE  ZERO.H                          ;section z
      .PAGE
      .SECTION  AD_CONVERT
      .ORG      $E000
      .SEXT     GET_AD_DATA                     ;sub routine
      .PUB      START
      .INCLUDE  MACRO.A74
START:
      D = 0
      T = 0
      S = STACK
      RAM_CL "WORK0, WORK1"                     ;ram clear
      [fAD_INT_OK] = OFF
      X = 2                                     ;A/D convert 2 times
      do
              JSR     \GET_AD_DATA
              [WORK0] = [WORK0] + [AD_RESULT]
              [WORK1] = [WORK1] + 0 with_c
              X = --X
      while  X
      C = 0                                     ;A/D DATA average
      [WORK1] = [WORK1] >> 1
      [WORK0] = [WORK0] >>1 with_c
      [AD_DATA] = [WORK0]
      .END
```

Fig 2.1 : Example of a source file

```
        .INCLUDE        ZERO.I00                    ;section z
                        ↑     The INCLUDE command line has name changed to the I
                              file generated by SRA74
        .PAGE
        .SECTION        AD_CONVERT
        .ORG            $E000
        .SEXT           GET_AD_DATA                 ;sub routine
        .PUB            START
        .INCLUDE        MACRO.I01
START:

;       D = 0     ← Structured commands output as comments and converted to assembly language.
                CLD
;       T = 0
                CLT
;       S = STACK
        LDX     #STACK
                TXS
;       RAM_CL  "WORK0, WORK1"                      ;ram clear
        LDA     #0
;       .REPEATI         ram, WORK0, WORK1        ←Macro commands output as comments and
                                                    converted into assembly language.
;       STA     ram
;       .ENDM
        STA     WORK0
        STA     WORK1
;       .ENDM
;       .ENDM
;       [fAD_INT_OK] = OFF
                CLB     fAD_INT_OK
;       X = 2                                       ;A/D convert 2 times
                LDX     #2
```

Fig 2.2 : Example of an I file (First half of source file)

```
;       do
        .DO:
                JSR     \GET_AD_DATA
;               [WORK0] = [WORK0] + [AD_RESULT]
                        LDA     WORK0
                        CLC
                        ADC     AD_RESULT
                        STA     WORK0
;               [WORK1] = [WORK1] + 0 with_c
                        LDA     WORK1
                        ADC     #0
                        STA     WORK1
;               X = --X
                        DEX
;       while X
                CPX  #0
                BNE  .D0
;       C = 0                                   ;A/D DATA average
                CLC
;       [WORK1] = [WORK1] >> 1
                LSR     WORK1
;       [WORK0] = [WORK0] >>1 with_c
                ROR     WORK0
;       [AD_DATA] = [WORK0]
                LDA     WORK0
                STA     AD_DATA
        .END
```

Fig 2.3 : Example of an I file (Second half of source file)

2.2.2 Configuration of PRN file

Figs 2.4 - 2.11 show examples of output PRN files. The PRN file contains the following data:-

1. Contents of source file plus data pertaining to corresponding address and generated data (first half of Figs 2.4 - 2.5 and 2.7 - 2.9).
   - Lines referring to external labels[4] show “E” beside the source.
   - Lines referring to public labels[5] show “P” beside the source.
   - Lines referring to local labels[6] show “L” beside the source.
   - Lines referring to symbols[7] show “S” beside the source.
   - Lines referring to bit symbols[8] show “B” beside the source.

2. Result of assembly operation (Second half of Fig 2.5 and Fig 2.9)
   Shows the number of errors, number of warnings, total number of lines, number of comment lines and memory capacity of each section.

3. List of symbols (First half of Fig 2.6 and whole of Fig 2.10)
   The symbols used in the program are divided into the following five types. A print file containing a symbol list can be generated by specifying the command parameter “-LS”.

   - USED (-d OPTION)
     Indicates symbol defined by command parameter “-D” from the command line and referred to in the program.

   - USED (EQUATE)
     Indicates symbol defined by pseudo-command .EQU and referred to in the program.

   - USED (BIT EQUATE)
     Indicates bit symbol defined by pseudo-command .EQU and referred to in the program.

   - UNUSED (-d or EQUATE)
     Indicates symbol defined in the above manner but not used in the program.

   - UNUSED (BIT EQUATE)
     Indicates bit symbol defined by pseudo-command .EQU but not referred to in the program.

[4] Indicates labels defined in other files. External labels and public labels are together referred to as global labels.

[5] Labels defined in this file which can be referred to from other files.

[6] Labels defined in this file which can be referred to only from within this file.

[7] Indicates symbols defined by the -D parameter and pseudo-command .EQU.

[8] Indicates bit symbols defined by pseudo-command .EQU.

4. List of labels (Second half of Fig 2.6 and whole of Fig 2.11)
   Lists the labels used in the program with their values divided into the following 3 groups:-

- USED (USER DEFINED)
  Indicates labels at section level which are defined and referred to in the program.

- USED (SYSTEM DEFINED)
  Indicates labels at section level which are defined by SRA74 and referred to in the program (Output only when command parameter “-I” is specified).

- UNUSED (USER DEFINED)
  Indicates labels at section level which are defined but which are not referred to in the program.

5. When the column number specified by pseudo-command . COL exceeds 132 characters then the time of assembly is shown in the list header section in the following manner.

```
   Sat Jan 16 15:06:42 1993
```

```
* 740 Family SRA74 V.4.00.00  *

SEQ.. LOC. OBJ...   S NM ....3....*....4....*....5....SOURCE STATEMENT....*....8

    1                 0          .INCLUDE        ZERO.H                ;section z
                    ↑Indicates include nest.
    2                 1          .SECTION        Z
    3                 1          .ORG            0
    4 0000 (0001)     1 :WORK0:          .BLKB   1
    5 0001 (0001)     1 :WORK1:          .BLKB   1
    6 0002 (0001)     1 :AD_RESULT:      .BLKB   1
    7 0003 (0001)     1 :AD_DATA:        .BLKB   1
            ↑Indicates size of reserved area.
    8                 1          .ORG            $FE
    9 00FE (0001)     1 :ICON:           .BLKB   1
                  ↓Indicates reference to public label.
   10 4,00FE        P 1 :fAD_INT_OK     .EQU    4, ICON
     ↑Indicates bit number.
   11                 1          .INCLUDE        DEFINE.H
   12 00BF            2 :STACK           .EQU    $BF
   13 0000            2 :OFF             .EQU    0
   14 0001            2 :ON              .EQU    1
```

Fig 2.4 : Example of a PRN file (First half of source file : No “-I” parameter)

```
* 740 Family SRA74 V.4.00.00  *

SEQ.. LOC. OBJ...   S NM ....3....*....4....*....5....SOURCE STATEMENT....*....8

   15                0          .PAGE
   16                0          .SECTION        AD_CONVERT
   17                0          .ORG            $E000
   18                0          .SEXT           GET_AD_DATA    ;sub routine
   19                0          .PUB            START
   20                0          .INCLUDE        MACRO.A74
   21                1 RAM_CL: .MACRO  RAMS
   22                1          LDA     #0
   23                1          .REPEATI        ram, RAMS
   24                1          STA     ram
   25                1          .ENDM
   26                1          .ENDM
   27 E000           0 START:
   28 E000           0          D = 0
   29 E001           0          T = 0
   30 E002           0          S = STACK
   31 E005           0          RAM_CL  "WORK0, WORK1"        ;ram clear
   32 E00B           0          [ fAD_INT_OK ] = OFF
   33 E00D           0          X = 2                         ;A/D convert 2 times
   34 E00F           0          do
   35 E00F 2200     E 0                  JSR     \GET_AD_DATA
                    ↑Indicates reference to external label.
   36 E011           0                   [ WORK0 ] = [ WORK0 ] + [ AD_RESULT ]
   37 E018           0                   [ WORK1 ] = [ WORK1 ] + 0 with_c
   38 E01E           0                   X = --X
   39 E01F           0          while  X
   40 E023           0          C = 0                          ;A/D DATA average
   41 E024           0          [ WORK1 ] = [ WORK1 ] >> 1
   42 E026           0          [ WORK0 ] = [ WORK0 ] >>1 with_c
   43 E028           0          [ AD_DATA ] = [ WORK0 ]
   44                0          .END

ERROR   COUNT           00000
WARNING COUNT           00000
STRUCTURED STATEMENT    00014 LINES
TOTAL   LINE ( SOURCE ) 00044 LINES
TOTAL   LINE ( OBJECT ) 00044 LINES
COMMENT LINE ( SOURCE ) 00000 LINES
COMMENT LINE ( OBJECT ) 00000 LINES
OBJECT  SIZE ( Z          ) 00005 (0005) BYTES
               ↑Indicates section name (up to 32 chars)
OBJECT  SIZE ( AD_CONVERT ) 00044 (002C) BYTES
```

Fig 2.5 : Example of a PRN file (Second half of source file : No “-I” parameter)

```
* 740 Family SRA74 V.4.00.00  *

*** USED   SYMBOLS ( TYPE = -d OPTION      ) ***


*** USED   SYMBOLS ( TYPE = EQUATE         ) ***
  ↓Symbol name output up to 32 characters.
OFF            0000p       STACK          00BFp
                  ↑Indicates public label.
*** USED   SYMBOLS ( TYPE = BIT EQUATE     ) ***

fAD_INT_OK     4,00FEp
              ↑Indicates the relevant bit with value from 1-7.
*** UNUSED SYMBOLS ( TYPE = -d or EQUATE   ) ***

ON             0001p

*** UNUSED SYMBOLS ( TYPE = BIT EQUATE     ) ***


*** USED   LABELS  ( TYPE = USER   DEFINED ) ***

EXTERNAL
  ↑Indicates external reference.
GET_AD_DATA    0000e
                  ↑Indicates external label
SECTION : Z
          ↑Indicates section name (up to 122 characters).
AD_DATA        0003p       AD_RESULT      0002p       ICON           00FEp
WORK0          0000p       WORK1          0001p

SECTION : AD_CONVERT


*** UNUSED LABELS  ( TYPE = USER   DEFINED ) ***

EXTERNAL


SECTION : Z


SECTION : AD_CONVERT

START          E000p
```

Fig 2.6 : Example of a PRN file (Symbol and label list : No “-l” parameter).

```
* 740 Family SRA74 V.4.00.00  *

SEQ.. LOC. OBJ...   S NM ....3....*....4....*....5....SOURCE STATEMENT....*....8

    1                 0         .INCLUDE        ZERO.H                  ;section z
    2                 1         .SECTION        Z
    3                 1         .ORG            0
    4 0000 (0001)     1 :WORK0:          .BLKB   1
    5 0001 (0001)     1 :WORK1:          .BLKB   1
    6 0002 (0001)     1 :AD_RESULT:      .BLKB   1
    7 0003 (0001)     1 :AD_DATA:        .BLKB   1
    8                 1         .ORG            $FE
    9 00FE (0001)     1 :ICON:           .BLKB   1
   10 4,00FE        P 1 :fAD_INT_OK     .EQU    4, ICON
   11                 1         .INCLUDE        DEFINE.H
   12 00BF            2 :STACK           .EQU    $BF
   13 0000            2 :OFF             .EQU    0
   14 0001            2 :ON              .EQU    1
```

Fig 2.7 : Example of a PRN file (First part of source file : Uses “-I” parameter)

```
* 740 Family SRA74 V.4.00.00  *

SEQ.. LOC. OBJ...   S NM ....3....*....4....*....5....SOURCE STATEMENT....*....8

   15                 0          .PAGE
   16                 0          .SECTION        AD_CONVERT
   17                 0          .ORG            $E000
   18                 0          .SEXT           GET_AD_DATA          ;sub routine
   19                 0          .PUB            START
   20                 0          .INCLUDE        MACRO.A74
   21                 1 RAM_CL:  .MACRO  RAMS
   22                 1          LDA     #0
   23                 1          .REPEATI        ram, RAMS
   24                 1          STA     ram
   25                 1          .ENDM
   26                 1          .ENDM
   27 E000            0 START:
   28                 0 ;         D = 0       ←Structured command becomes comment.
   29 E000 D8         0                   CLD   ←Structured inst. converted to assembly language
   30                 0 ;         T = 0
   31 E001 12         0                   CLT
   32                 0 ;         S = STACK
   33 E002 A2BF     S 0                   LDX     #STACK
                  ↑Indicates symbol reference.
   34 E004 9A         0                   TXS
   35 E005            0          RAM_CL   "WORK0, WORK1"         ;ram clear
   36 E005 A900       0+         LDA      #0
                    ↑Indicates macro expansion.
   37                 0+         .REPEATI        ram, WORK0, WORK1
   38                 0+         STA     ram
   39                 0+         .ENDM
   40 E007 8500     P 0+         STA     WORK0
   41 E009 8501     P 0+         STA     WORK1
   42                 0+         .ENDM
   43                 0 ;        [ fAD_INT_OK ] = OFF
   44 E00B 9FFE     B 0                   CLB     fAD_INT_OK
                  ↑Indicates reference to bit symbol.
   45                 0 ;        X = 2                          ;A/D convert 2 times
   46 E00D A202       0                   LDX     #2
```

Fig 2.8 : Example of a PRN file (Middle part of source file : Uses “-l” parameter)

```
   47                0 ;       do
   48 E00F           0         .D0:
   49 E00F 2200    E 0                 JSR     \GET_AD_DATA
   50                0 ;               [ WORK0 ] = [ WORK0 ] + [ AD_RESULT ]
   51 E011 A500    P 0                 LDA     WORK0
   52 E013 18        0                 CLC
   53 E014 6502    P 0                 ADC     AD_RESULT
   54 E016 8500    P 0                 STA     WORK0
   55                0 ;               [ WORK1 ] = [ WORK1 ] + 0 with_c
   56 E018 A501    P 0                 LDA     WORK1
   57 E01A 6900      0                 ADC     #0
   58 E01C 8501    P 0                 STA     WORK1
   59                0 ;               X = --X
   60 E01E CA        0                 DEX
   61                0 ;       while  X
   62 E01F E000      0                 CPX     #0
   63 E021 D0EC    L 0                 BNE     .D0

* 740 Family SRA74 V.4.00.00  *

SEQ.. LOC. OBJ...   S NM ....3....*....4....*....5....SOURCE STATEMENT....*....8

   64                0 ;       C = 0                           ;A/D DATA average
   65 E023 18        0                 CLC
   66                0 ;       [ WORK1 ] = [ WORK1 ] >> 1
   67 E024 4601    P 0                 LSR     WORK1
   68                0 ;       [ WORK0 ] = [ WORK0 ] >>1 with_c
   69 E026 6600    P 0                 ROR     WORK0
   70                0 ;       [ AD_DATA ] = [ WORK0 ]
   71 E028 A500    P 0                 LDA     WORK0
   72 E02A 8503    P 0                 STA     AD_DATA
   73                0         .END

ERROR   COUNT           00000
WARNING COUNT           00000
STRUCTURED STATEMENT    00014 LINES
TOTAL   LINE ( SOURCE ) 00043 LINES
TOTAL   LINE ( OBJECT ) 00073 LINES
COMMENT LINE ( SOURCE ) 00000 LINES
COMMENT LINE ( OBJECT ) 00014 LINES
OBJECT  SIZE ( Z          ) 00005 (0005) BYTES
OBJECT  SIZE ( AD_CONVERT ) 00044 (002C) BYTES
```

Fig 2.9 : Example of a PRN file (Final part of source file : Uses “-l” parameter)

```
* 740 Family SRA74 V.4.00.00  *
*** USED   SYMBOLS ( TYPE = -d OPTION      ) ***

*** USED   SYMBOLS ( TYPE = EQUATE         ) ***
OFF            0000p      STACK        00BFp
*** USED   SYMBOLS ( TYPE = BIT EQUATE     ) ***
fAD_INT_OK    4,00FEp
*** UNUSED SYMBOLS ( TYPE = -d or EQUATE   ) ***
ON             0001p
*** UNUSED SYMBOLS ( TYPE = BIT EQUATE     ) ***
```

Fig 2.10 : Example of a PRN file (Symbol list : Uses “- l” parameter)

```
*** USED   LABELS  ( TYPE = USER   DEFINED ) ***

EXTERNAL

GET_AD_DATA   0000e

SECTION : Z

AD_DATA       0003p      AD_RESULT     0002p      ICON          00FEp
WORK0         0000p      WORK1         0001p

SECTION : AD_CONVERT


*** USED   LABELS  ( TYPE = SYSTEM DEFINED ) ***

SECTION : Z


SECTION : AD_CONVERT

.D0           E00F'

*** UNUSED LABELS  ( TYPE = USER   DEFINED ) ***

EXTERNAL


SECTION : Z


SECTION : AD_CONVERT

START         E000p
```

Fig 2.11 : Example of a PRN file (Label list : Uses “-l” parameter)

2.2.3 Configuration of TAG file

Fig 2.12 gives an example of TAG file output. A TAG file contains the following data.

- Indicates name of file where error or warning has occurred, line number within the file, consecutive line number, error number and error message.

The TAG file should be printed out for use as a reference when correcting editor errors.

```
TEST.A74 115 ( TOTAL LINE 115 ) Error 18: Relative jump is out of range
TEST.A74 127 ( TOTAL LINE 127 ) Error 22: Value is out of range “data”
TEST.A74 593 ( TOTAL LINE 593 ) Error 23: “()” format error “;”
```

Fig 2.12 : Example of a TAG file

CHAPTER 3
Entering the Source Program

3.1 Configuration of the source program

The configuration of the source program takes the line as its basic unit. The rules for line entry are outlined below.

1. Each line is complete in itself. More than 1 line cannot therefore be used to enter a single command.

2. Each line contains a maximum of 256 characters including the line feed code. SRA74 ignores areas which exceed the 256 character limit.

3. Lines are divided into the following 5 groups in accordance with their content:-
   - Assembly language command line
     This line contains a 740 Family assembly language command.
     It also generates the corresponding machine language data.

   - Structured language command line
     This line contains a 740 Family structured language command.
     It also generates the corresponding machine language data.

   - Pseudo-command line
     This line contains a 740 Family pseudo-command.

   - Macro command line
     This line contains a 740 Family macro command.

   - Comment line
     This line is not processed by SRA74 and is therefore free for use as the user requires.

3.2 Configuration of a line

This section explains the configuration of each of the lines. An explanation and rules concerning the use of symbols for notation is also included.

1. △, ▲ indicates a space or tab code.
   The △ part is required but the ▲ part may be left out.

2. When entering a label a ":" (colon) is not an absolute requirement but if it is left out then a space or tab code must be inserted between each command. During assembly the -U parameter must be specified. You are therefore recommended to use the ":" as a matter of course.

3.2.1 The assembly language command line

The configuration of the assembly language command line is shown below. For further details concerning this line please refer to Chapter 4, Appendices B and C.
▲ Label :▲ Op-code △ Operand ▲; Comment <RET>

1. Label section
   Enter the label to enable reference to the line from another location.

2. Operation code section
   Enter the 740 Family assembly language mnemonic (Hereafter referred to as the op-code). The op-code makes no distinction between upper case and lower case alphabetic characters. NOP, nop or Nop would therefore all be valid.

   SRA74 recognizes the op-code as a reserved word so that when a label is not entered entry can be made from the head of the line.

3. Operand section
   Enter the op-code's target.
   - When the operand contains more than one item of data then the items should be separated from each other by a "," (comma).
   - A space or tab code can be entered on either side of a comma.

4. Comment section
   This section is not subject to processing by SRA74 and may therefore be utilized in whatever way the user wishes.

3.2.2 Structured language command line

The configuration of a structured language command (hereafter referred to as structured command) line is as shown below. For further details concerning this line please refer to Chapter 5 and Appendix F.
▲ Label :▲ Structured command △ Conditional expression ▲; Comment <RET>

1. Label section
   Enter a label to enable reference to be made to the line from another location.

2. Structured command section
   Enter a 740 Family structured command. Structured commands make no distinction between upper and lower case alphabetic characters. Either IF, if or If would therefore be equally valid.
   SRA74 recognizes the op-code as a reserved word so that when a label is not entered entry can be made from the head of the line.

3. Conditional expression section
   Enter the 740 Family structured command's target.

4. Comment section
   This section is not subject to processing by SRA74 and may therefore be utilized in whatever way the user wishes.

3.2.3 Pseudo-command line

The configuration of the pseudo-command line is shown below. For further details concerning this line please refer to Chapter 6 and Appendix D.

▲ Symbol / Bit symbol △ .EQU △ Operand ▲; Comment <RET>
▲ Label:▲ Pseudo-command △ Operand ▲; Comment <RET>

1. Symbol / Bit symbol section
   Enter a symbol or bit symbol with a value given by the pseudo-command .EQU.

2. Label section
   Enter the label to enable reference to the line from another location.

3. Pseudo-command section
   Enter a 740 Family pseudo-command. The pseudo-command makes no distinction between upper case and lower case alphabetic characters. Either .END, .end or .End would therefore be equally valid. SRA74 recognizes the pseudo-command as a reserved word so that when a label is not entered entry can be made from the head of the line.

4. Operand section
   Enter the pseudo-command's target.
   - When the operand contains more than one item of data then the items should be separated from each other by a "," (comma).
   - A space or tab code can be entered on either side of a comma.

5. Comment section
   This section is not subject to processing by SRA74 and may therefore be utilized in whatever way the user wishes.

3.2.4 Macro command line

The configuration of the macro command line is shown below. For further details concerning this line please refer to Chapter 7 and Appendix E.

▲ Label:▲ .MACRO △ Operand ▲; Comment <RET>
▲ Label:▲ Macro command △ Operand ▲; Comment <RET>

1. Macro name section
   Name for accessing the macro definition.

2. Label section
   Enter the label (name) to enable reference to the line from another location.

3. Macro command section
   Enter a 740 Family macro command. The macro command makes no distinction between upper case and lower case alphabetic characters. Either .REPEATI, .repeati or .RepeatI would therefore all be equally valid. SRA74 recognizes the op-code as a reserved word so that when a label is not entered entry can be made from the head of the line.

4. Operand section
   Enter the macro command's operand.
   - When the operand contains more than one item of data then the items should be separated from each other by a "," (comma).
   - A space or tab code can be entered on either side of a comma.

5. Comment section
   This section is not subject to processing by SRA74 and may therefore be utilized in whatever way the user wishes.

3.2.5 Comment line

The first character on a comment line (with the exception of ▲) should always be a ";" (semi-colon). The configuration of a comment line is shown below.

▲; Comment <RET>

Note:
If the first character on a comment line is anything other than a ";" then SRA74 will not recognize the line as a comment and will include it in the assembly operation. Care must therefore be taken since this will result in a variety of problems such as assembly error or even the generation of false codes.

3.3 Section entry method

This section deals only with those sections where the entry format is the same for each command line. For further details of those sections requiring more than one entry format please refer to Chapters 4, 5, 6 and 7.

3.3.1 Symbol / Bit symbol / Label sections

These sections require the same entry format for each command line. SRA74 distinguishes symbols, bit symbols and labels[1] from each other and processes them accordingly. The rules of entry are given below:-

1. The name can be entered using either alphanumeric characters or the special characters "_" (underline), "." (period) and "?" (question mark). The first character must, however, be either an alphabetic or special character. The maximum number of permissible characters, including ":", is 255.

2. A distinction is made between upper and lower case characters. BIG and Big would therefore be discriminated as different names.

3. Reserved words may not be used as names. SRA74 processes all register names, flag names, op-codes, structured commands, pseudo-commands and macro commands as reserved words.

4. The following labels starting with the special character "." (period) should not be used as they are reserved by SRA74:-

   - .D0 - .D65535
   - .F0 - .F65535
   - .I0 - .I65535
   - .S0 - .S65535

5. The following labels starting with the special character ".." (double period) should not be used since they are reserved by SRA74. Also avoid labels beginning with '..' because they are reserved for SRA74 function enhancement.
   - ..0 - ..65535

6. Labels starting with the special character "??" (double question mark) are treated local labels valid only within that section. It is thus possible to use the same label name again in a different section. In case of reference only labels within that section will be referred to.

[1] SRA74 treats anything defined by the pseudo-command .EQU or parameter "-D" as a symbol or bit symbol and anything defined in any other way as a label.

Example:

```
                .SECTION PROG1
      ??MAIN:   NOP
                  :
                BRA     ??MAIN      This will result in a jump to "??MAIN"
                                         within the section named "PROG1".

                .SECTION PROG2

      ??MAIN:   NOP
                  :
                BRA     ??MAIN       This will result in a jump to "??MAIN'
                                        within the section named "PROG2".
```

7. The .PUB declaration can be omitted if a ":" is inserted immediately before a symbol, bit symbol or label.

   Example:

```
           :SYMBOL    .EQU 10
                      :
                      .SECTION   PROG2
           :LABEL:    NOP
                      :
```

When entering a label a ":" (colon) can be inserted immediately prior to the name. We recommend that a ":" should be inserted before entry in order to make it easy to distinguish between a label and a symbol and to enable label access to be carried out effectively by the editor. Take care, however, since it is an error to insert a ":" when entering a symbol or bit symbol.

3.3.2 Comment section

The user is free to enter whatever data he wishes here. The entry format is as follows:-

1. A ";" (semi-colon) must be inserted at the head of the comment before entry.

2. Any kind of character can be entered in the comment section.

CHAPTER 4
Assembly Language

4.1 Addressing mode

The addressing mode is the basic means (mode) by which a command specifies the data which it processes. The 740 Family has 19 addressing modes each with its own specific operand format. The addressing modes with related operand entry are outlined below:-

1. Implied
   This is an op-code command only. There is no operand specification.
   Example: `BRK`

2. Accumulator
   Method by which the contents of the accumulator are specified as the target data.
   Example: `ASL  A`

3. Immediate
   Method by which the operand is specified directly as the target data. The value entered in the operand should be preceded by "#".
   Example: `ADC  #IMMDATA`

4. Zero page
   Method by which the zero page area (00_{16} - FF_{16}) is specified as the target data.
   Example: `ADC  ZWORK`

5. Zero page X
   Method by which the zero page address is qualified by register X as the target data. The register name X should be entered after a "," (comma).
   Example: `ADC  ZWORK,X`

6. Zero page Y
   Method by which the zero page address is qualified by register Y as the target data. The register name Y should be entered after a "," (comma).
   Example: `LDX  ZWORK,Y`

7. Zero page indirect
   Method by which the target data is indicated to be in indirect memory. The zero page address which holds the target address is entered into the operand and the 2 byte target address is stored in the memory. The operand value should be enclosed by ().
   Example: `JMP (ZWORK)`

8. Zero page indirect X
   Method by which target data is indicated to be in indirect memory and qualified by register X. The zero page address which holds the target address is entered into the operand and the 2 byte target address is stored in the memory. The register name “X” is entered after a “,” (comma) and the operand value should be enclosed by ().
   Example: `ADC (ZWORK,X)`

9. Zero page indirect Y
   Method by which target data is indicated to be in indirect memory and qualified by register Y. The zero page address which holds the target address is entered into the operand and the 2 byte target address is stored in the memory. The register name “Y” is entered after a “,” (comma) and the operand value should be enclosed by ().
   Example: `ADC (ZWORK),Y`

10. Absolute
    Method by which a general page area (0100_{16} - $FFFF_{16}$) is specified as the target data.
    Example: `ADC WORK`

11. Absolute X
    Method by which a general page address is qualified by register X as the target data. The register name “X” should be entered after a “,” (comma).
    Example: `ADC WORK,X`

12. Absolute Y
    Method by which a general page address is qualified by register Y as the target data. The register name “Y” should be entered after a “,” (comma).
    Example: `ADC WORK,Y`

13. Absolute indirect
    Method by which target data is indicated to be in indirect memory. The general page address which holds the target address is entered into the operand and the 2 byte target address is stored in the memory. The operand value should be enclosed by ().
    Example: `JMP (WORK)`

14. Special page
    Method by which a special page area ($FF00_{16}$ - $FFFF_{16}$) is specified as the target data. The operand value should be indicated by “¥” or “\”.
    Example: `JSR \WORK`

15. Zero page bit
    Method by which a special page area (00_{16} - FF_{16}) specific bit is specified as the target data. When a bit symbol in entered into the operand the value and address of that bit symbol will be referred to .
    Example: `CLB 0,ZWORK` ⇒ Specifies bit 0 of “ZWORK”.
    Example: `CLB BITSYMBOL`

16. Accumulator bit
    Method by which a specific accumulator bit is specified as the target data. If the bit symbol is entered into the first operand then the value of that bit symbol only will be referred to.
    Example: `CLB 1,A` ⇒ Specifies bit 1 of the accumulator.
    Example: `CLB BITSYMBOL,A`

17. Relative
    Causes a jump to an address located relative to the head of the main command at a position measured in terms of the content of the operand. The relative value itself cannot be entered into the operand. If a label or target address is entered into the operand SRA74 will calculate the relative value.
    Example: `BRA *-12`
    Example: `BRA NEXT`

18. Zero page bit relative
    Specifies a zero page area (00_{16} - FF_{16}) specific bit as the target data. It is a method whereby if a bit symbol is entered into the first operand then the bit value and address of that bit symbol will be referred to and, in accordance with the status of that bit, a jump will be made to an address located relative to the head address of the main command at a distance measured in terms of the content of the final operand. The relative value itself cannot be entered into the final operand. If the label or target address is entered then SRA74 will calculate the relative value.
    Example: `BBC 2,ZWORK,NEXT` ⇒ Specifies bit 2 of “ZWORK”.
    Example: `BBC BITSYMBOL,NEXT`

19. Accumulator bit relative
    Specifies a specific accumulator bit as the target data. It is a method whereby if a bit symbol is entered into the first operand then only the bit value of that bit symbol will be referred to and, in accordance with the status of that bit, a jump will be made to an address located relative to the head address of the main command at a distance measured in terms of the content of the final operand. The relative value itself cannot be entered into the final operand. If the label or target address is entered then SRA74 will calculate the relative value.
    Example: `BBC 3,A,NEXT` ⇒ Specifies bit 3 of the accumulator.
    Example: `BBC BITSYMBOL,A,NEXT`

The entry formats for each command in each addressing mode are given for reference in Appendix C.

4.2 Operand data format

The 4 types of data format shown below may be entered into the operand.

1. Numerical constant
   - Positive and negative values are shown by inserting the operators “+” or “-” before the numerical constant. Where neither operator is inserted then the value will be processed as if positive.
   - A space or tab cannot be inserted between the sign which indicates the type of number and the number itself.
     Example: `.BYTE  $ 64` ⇒ This is an error.
   - Either binary, octal, decimal or hexadecimal numbers may be used as numerical constants.
   - Binary numbers ⇒ Should be composed of binary figures with either a “%” at the head or a “B” or “b” at the end.
     Example: `.BYTE  %100110`
     Example: `.BYTE  100110B`
   - Octal numbers ⇒ Should be composed of octal figures with either a “@” at the head or “O”, “o”, “Q” or “q” at the end.
     Example: `.BYTE  @70`
     Example: `.BYTE  70O`
     Example: `.BYTE  70Q`
   - Decimal numbers ⇒ Should be composed of decimal figures without a further special identification. Only integers such as 23 or 256 may be entered.
     Example: `.BYTE  100`
   - Hexadecimal numbers ⇒ Should be composed of hexadecimal figures with either a “$” at the head or “H” or “h” at the end. Where the head starts with the alphabetic characters A-F please insert 0 at the beginning.
     Example: `.BYTE  $64`
     Example: `.BYTE  64H`
     Example: `.BYTE  0ABH`

2. Character constants
   - Characters defined by the ASCII code may be used.
   - Character constants should be enclosed within either ‘ (single quotes) or “ (double quotes). Each character corresponds to 7 bit ASCII code (most significant bit 0).
     Example: `.BYTE  ‘A’` ⇒ To set 41_{16}.

3. Symbolic constant
   - There are 4 types of “*” symbol which are used to indicate a symbol, bit symbol, label or the head of the current statement. Bit symbols have the value allocated to the bit and the address to which that bit belongs. Symbols have an absolute value whereas labels and “*” have either relative or absolute values.

Example: `.WORD SUB` ⇒ Sets the address of the label SUB.
Example: `BRA *+2` ⇒Causes a jump to an address located by adding 2 to the current address.

4. Expressions
   - Expressions are composed of a mixture of numerical constants, character constants, symbolic constants and operators. A space or tab may be inserted as required between any operator and the related item.
     Example: `TBL + 1`
   - Expressions are processed from left to right (Operators do not have priority).
     Example: `2*3` ⇒ The result is 6.
     Example: `2+6/2` ⇒ The result is 4.

4.3 Operators

Table 4.1 lists all the operators which can be used for operand data entry.

Table 4.1 : List of operators

Type	Operator[1]	Description
Single term operator	+	Indicates positive number
	-	Indicates negative number
	!	Takes ones complement
	<	Extracts 8 high order bits of label or symbol
	>	Extracts 8 low order bits of label or symbol
	SIZEOF[2]	Finds size of section
	BK[3]	Obtains the bank value of label
	BL[3]	Obtains the extra area value [4] of label
Dual term operator	+	Addition
	-	Subtraction
	*	Multiplication
	/	Division
	&	AND of each bit
	|	OR of each bit

Notes:

1. With SRA74 V.4.00.00 and later, format checks are processed as integers attached with a 32-bit code. However, results are handled as such only in the following two cases.
   • Operands of the .ORG command
   • Labels when the -BANK option is specified
   With the following written format, operation results can differ from that obtained with V.2.00.10 and earlier.
   Example 2223h + 0FFFFh / 2
   • Results for V.2.00.10 and earlier: 1111H
   • Results for V.4.00.00 and later: 9111H

2. The SIZEOF value is determined at the time of the link operation without regard to the reference section or whether it is relocatable or absolute. The SIZEOF operator can only be used with a section name. When using the SIZEOF operator a space must be left between the operator and the section name.
   Example: `.WORD SIZEOF DATA`

3. For the operands of operators BK and BL, specify labels whose values are defined in assembly execution.

4. The extra area value is obtained by adding 1000H to the low-order 12 bits of the operand (label).

5. Operations are carried out from left to right (operators have no order of priority).
   Example: `2+6/2` ⇒ The result is 4.

6. Bit symbol operations and operations between labels which refer by means of .ZEXT and .SEXT are not possible.

CHAPTER 5
Structured Language

5.1 Function of the structured command

The readability of a program is severly affected by the frequent use of branch instructions and labels. The structured language allows coding of conditional branch structures and repeat structures without the use of labels; thus improving program readability. In addition, instructions that directly operate on registers and flags are available to enable generation of efficient codes.

5.2 Types of statement

The structured language statements can be classified into seven types as shown below. When the character l is prefixed to each instruction, a long branch using the “JMP” instruction is generated. The following examples show the coding example on the left with equivalent assembler program on the right. Refer to Appendix F for the details of each statement.

1. Assignment statement
   Assigns the right to the left.

```
[ MEM ] = 10             ; LDM   #10,MEM
[ MEM ] = [ MEM1 ]       ; LDA   MEM1
                         ; STA   MEM
```

2. (l)if - (l)else - endif statement
   The “if” statement is an instruction to change the control stream into two directions with the branch direction being determined by means of a conditional expression.

```
if [ MEM ]               ;          LDA   MEM
        [ WORK ] = 1     ;          BEQ   ELSE
else                     ;          LDM   #1,WORK
        [ WORK ] = 2     ;          BRA   ENDIF
endif                    ; ELSE:
                         ;          LDM   #2,WORK
                         ; ENDIF:
```

3. (l)for - next statement

   The “for” statement is an instruction for the control of repetition whereby a statement is executed repeatedly for as long as the specified conditional expression holds true.

```
for [ MEM ]              ; FOR:
        jsr   output     ;         LDA   MEM
next                     ;         BEQ   NEXT
                         ;         JSR   OUTPUT
                         ;         BRA   FOR
                         ; NEXT:
```

4. (l)do - while statement

   The “do” statement is executed repeatedly for as long as a specified conditional expression holds true.

```
do                       ; DO:
        jsr   output     ;         JSR   OUTPUT
while  [ MEM ]           ;         LDA   MEM
                         ;         BNE   DO
```

5. (l)switch - case - ends statement

   The “switch” statement switches control to some another statement in accordance with the value of a conditional expression.

```
switch  [ MEM ]          ;         LDA   MEM
    case  1              ;         CMP   #1
        jsr  output1     ;         BNE   CASE2
        break            ;         JSR   OUTPUT1
    case  2              ;         BRA   ENDS
        jsr  output2     ; CASE2:
        break            ;         CMP   #2
    case  3              ;         BNE   CASE3
        jsr  output3     ;         JSR   OUTPUT2
        break            ;         BRA   ENDS
    default              ; CASE3:
        jsr  output4     ;         CMP   #3
ends                     ;         BNE   DEFAULT
                         ;         JSR   OUTPUT3
                         ;         BRA   ENDS
                         ; DEFAULT:
                         ;         JSR   OUTPUT4
                         ; ENDS:
```

6. (l)break statement
   The “break” statement terminates the execution of a given “for”, “do” or “switch” statement and passes control to the subsequent statement.

```
for [ MEM1 ]            ; FOR:
        if  [ MEM2 ]    ;         LDA   MEM1
                  break ;         BEQ   NEXT
        endif           ;         LDA   MEM2
        jsr  output     ;         BEQ   ENDIF
next                    ;         BRA   NEXT
                        ; ENDIF:
                        ;         JSR   OUTPUT
                        ;         BRA   FOR
                        ; NEXT:
```

7. (l)continue statement

   The “continue” statement inserts a dummy statement to which it passes control after the final statement of the minimum “for” or “do” statement which contains it.

```
for [ MEM1 ]            ; FOR:
        if  [ MEM2 ]    ;         LDA   MEM1
               continue ;         BEQ   NEXT
        endif           ;         LDA   MEM2
        jsr  output     ;         BEQ   ENDIF
next                    ;         BRA   FOR
                        ; ENDIF:
                        ;         JSR   OUTPUT
                        ;         BRA   FOR
                        ; NEXT:
```

5.3 Notes concerning entry procedures

The following pages contain explanations of some of the points which should be borne in mind when carrying out programming using structured language.

1. Memory areas to be referred to via any one of the 740 Family addressing modes should be enclosed within “[]” or “{}” when entering them in the conditional expression section of an assignment statement or any of the control statements (the “if” statement, for example).

   Refer to the assignment statement generation example in Appendix F.3.1 for details.

2. Selected bits which may be referred to by bit symbols should be enclosed within “[]” or “{}” when entering them in the conditional expression section of an assignment statement or any of the control statements (for example, “if”). The reserved words shown below must, however, be used when referring to a selected accumulator bit. In this case there is no need to use the “[]” or “{}” enclosure procedure. Since no distinction is made between upper and lower case characters either BIT_A0 or bit_a0 would be equally valid.

BIT_A0 Accumulator bit 0 BIT_A1 Accumulator bit 1
BIT_A2 Accumulator bit 2 BIT_A3 Accumulator bit 3
BIT_A4 Accumulator bit 4 BIT_A5 Accumulator bit 5
BIT_A6 Accumulator bit 6 BIT_A7 Accumulator bit 7

Refer to the register and flag assignment statement generation example in Appendix F.3.1 A) for details.

3. All the 740 Family registers should be entered without specific enclosure in the conditional expression section of an assignment statement or any of the control statements (for example, “if”). This is because SRA74 uses the following types of names as reserved words. Since no distinction is made between upper and lower case characters either A or a would be equally valid.

   A Accumulator X Index register X
   Y Index register Y S Stack pointer
   P Processor status register

   Refer to the register and flag assignment statement generation example in Appendix F.3.1 A) for details.

4. The 740 Family status register flags should be entered without specific enclosure into the conditional expression section of an assignment statement or any control statement (for example, “if”). This is because SRA74 uses the following types of names as reserved words. Since no distinction is made between upper and lower case characters either C or c would be equally valid.

   C Carry flag Z Zero flag
   I Interrupt inhibit flag D Decimal mode flag
   T X qualified operation mode flag V Overflow flag
   N Negative flag

   Refer to the register and flag assignment statement generation example in Appendix F.3.1 A) and flag condition generation expansion example in Appendis F.3.2 for details.

5. SRA74 processes (using, for example, LDA commands to generate the codes) the following types of entry (front reference) as labels. If, therefore, “BITSYM” is only defined afterwards as a bit symbol then the generation code will not match (and SRA74 will register an error). The program should be rewritten such that when entering a bit symbol (BITSYM) reference should be made to it only after it has been defined.

   Example:

```
;if [BITSYM]
     LDA       BITSYM    Generation code
     BEQ       .I0       Generation code
     :
;endif
.I0:
   :
BITSYM   .equ     1,80h          Definition of bit symbol
```

For further details please refer to the syntax map in Appendix F.

6. Coding to reduce generated code size.
   - For IF and FOR statements, >= is more efficient than > and < is more efficient than <=. For DO statement, < is more efficient than >= and <= is more efficient than <.
     Refer to the memory conditional expression generation example in Appendix F3.2 for details.
   - DO statement is more efficient than FOR statement if the operation is to be performed more than once.
   - For SWITCH statement, it is more efficient to use the lbreak statement where necessary than to use the lswitch statement.

LSWITCH statement	Assembly language	LBREAK statement	Assembly language

`lswitch [ MEM ]`
`  case 1`
`  NOP`
`  break`
`  case 2`
`  :`
`  case 16`
`  NOP`
`  break`
`  default`
`  NOP`
`ends`
	`  CMP  # 1`
	`  BEQ  .Z0`
	`  JMP  .S2`
	`.Z0:`
	`  NOP`
	`  JMP  .S0`
	`.S2:`
	`  :`
	`.S16:`
	`  CMP  # 2`
	`  BEQ  .Z15`
	`  JMP  .S17`
	`.Z15:`
	`  NOP`
	`  JMP  .S0`
	`.S17:`
	`  NOP`
	`.S0:`
		`switch [ MEM ]`
		`  case 1`
		`  NOP`
		`  lbreak`
		`  case 2`
		`  :`
		`  case 16`
		`  NOP`
		`  break`
		`  default`
		`  NOP`
		`ends`
			`  CMP  # 1`
			`  BNE  .S2`
			`  NOP`
			`  JMP  .S0`
			`.S2:`
			`  :`
			`.S16:`
			`  CMP  # 2`
			`  BNE  .S17`
			`  NOP`
			`  BRA  .S0`
			`.S17:`
			`  NOP`
			`.S0:`

7. Accumulator value

   Most structured language operations use the accumulator. Therefore, note that the value in the accumulator will change after executing the following type of structured language statement.

```
;[ WORK ] = [ DATA1 ] + [ DATA2 ]
     LDA  DATA1
     CLC
     ADC  DATA2
     STA  WORK
;IF [ WORK ] & 0FH > 4
     LDA  WORK
     AND  #0FH
     CMP  #4
     BEQ  .I0
     BCC  .I0
;    X = ++X
     INX
;ENDIF
.I0:
```

   However, transfer to and comparison of registers are generated into efficient assembly language without the use of accumulators.

```
;X = [ WORK ]
     LDX  WORK
;IF X > [ WORK ]
     CPX  WORK
     BEQ  .I0
     BCC  .I0
;    Y = ++Y
     INY
;ENDIF
.I0:
```

   Refer to the dual term operation assignment statement generation example in Appendix F.3.1 for details.

5.4 Structured operators

Table 5.1 lists the operators which may be used with the structured language.

Table 5.1 : List of structured language operators

Type	Operator[1]	Description
Single term operator	+	Indicates positive number
	-	Indicates negative number
	~	Takes ones complement
	++	Increment
	--	Decrement
Dual term operator	+[1]	Addition
	-[1]	Subtraction
	*[1]	Multiplication
	/[2]	Division
	%[2]	Division surplus
	&	Each bit AND
	|	Each bit OR
	^	Each bit exclusive OR
	&&[3]	Logic AND
	||[3]	Logic OR
	<<	Left shift
	>>	Right shift
Comparative operator	<	Smaller
	>	Larger
	==	Equivalent
	!=	Non-equivalent
	<=	Smaller or equivalent
	>=	Larger or equivalent

Notes:

1. All operations are carried out as 8 bit unsigned numbers. When the results of operations (+,- only) are compared for size, for example, then the overflow and borrow may be used (In the following example, if the “work” contents are assumed to be FF16 then the result of the operation will be 0116 and the condition will be fulfilled).

   Example:

```
if [work] + 2 > 10
        :
else
        :
endif
```

2. When using *, / or % in a dual term operator the related subroutine library (SRA74.A74) should be linked after assembly as follows:

   a. Assemble SRA74.A74 and generate the relocatable file SRA74.R74.

   b. Define the labels corresponding to the operators as external reference.
   The labels corresponding to the operators in the following table must be defined as external label because SRA74 generates machine codes to call the subroutine in SRA74.R74 from operators *, /, and %.

Operator	Label	
*	.mult_8	Multiply routine
/	.div_8	Divide routine
%	.mod_8	Modulo routine

Coding example

```
.ext .mult_8, .div_8
```

c. Link the user program with SRA74.R74.

This can be used on system using one page (beyond address 010016) for stack area, but in this case the following definition and work area reservation must be made.

- Set the stack page (.SPPAGE label definition). The setting 0 secures page 0 and the setting 1 secures page 1. (Take care since this procedure simply provides the required indication to the assembler and it does not mean that the stack is now set).

  Example: .SPPAGE .EQU 1

- Secure 3 bytes in page 0 in the RAM as the operation work area (.syswk). The operation result will be set in the case of a multiplication from low order to high order in .syswk and .syswk+1. (The user himself should carry out the reference if he wishes to refer to the high order of the operation result).

  Example: .syswk .blkb 3

- Care should be taken when using the operation work area (.syswk) for an interrupt processing routine. In such cases there are a number of other operations which will need to be carried out such as the modification of the related library source (SRA74.A74) by the user or the saving of the operation work area to the stack.

3. Up to six structured commands (if, for while) can be coded using dual term operators && and ||.

   These logical operators are have no priority and are generated from left to right as follows:

```
;IF BIT_A1 == 1 || BIT_A2 == 1 && BIT_A3 == 1 || BIT_A4 == 1
     BBS  BIT_A1,A,.I0
     BBC  BIT_A2,A,.I1
     BBS  BIT_A3,A,.I0
     BBC  BIT_A4,A,.I1
.I0:
;    X = ++X
          INX
;ENDIF
.I1:
```

CHAPTER 6
Pseudo-Commands

6.1 Functions of pseudo-commands

A pseudo-command makes a specification[1] to SRA74 such that it will generate the machine language data which is the objective of the command. SRA74 uses 48 different pseudo- commands but these can be classified in terms of 6 main groups in accordance with their functions.

1. Assembly control
   - The pseudo-commands themselves do not generate data but control the assembly processing stream.
   - They do not affect address updating.
   - This group consists of the following seven pseudo-commands:

.ASSERT	Assemble assertion declaration
.END	Program termination declaration
.ERROR	Assembly error declaration
.IF (.ELSE) .ENDIF	Conditional assembly
.INCLUDE	Read file in

2. Address control
   - Data set pseudo-commands which generate constant data.
   - They carry out address updates.
   - This group consists of the following seven pseudo-commands:

.EQU (=)	Synonymous definitions
.ORG (*=)	Address specification
.BLKB	Secures RAM area
.BYTE .WORD	Set data

1 The pseudo-command names are referred to as “declarative” where they make a specification with regard to SRA74 and “specific” where they have an effect on an output file.

3. Link control
   - Carries out the controls associated with link processing operations.
   - This group consists of the 18 following pseudo-commands:-

.BEXT .ZBEXT	External reference specification (bit symbol)
.EXT .SEXT .ZEXT	External reference specification (symbol, label)
.PUB	Public specification (bit, symbol, label)
.SECTION	ROM and Ram area specification
.SECTION P(.PMOD)	ROM area specification (general page)
.SECTION R(.RMOD)	RAM area specification (general page)
.SECTION S(.SMOD)	ROM area specification (special page)
.SECTION Z(.ZMOD)	RAM area specification (zero page)
.OBJ .LIB	Link file name specification
.VER	Version specification

4. List control
   - Carries out controls related to PRN file output.
   - This group contains the 7 following pseudo-commands:-

.COL .LINE	List format (numbers of columns and lines) specification
.LIST .NLIST	List output/inhibit specification
.LISTM .NLISTM	Macro generation section list output/inhibit specification
.PAGE	Page feed and title specification

5. Debug support
   - Carries out the controls relating to the source line debugger.
   - This group consists of the 2 following pseudo-commands:-

.FUNC .ENDFUNC	Function specification

6. Reserved pseudo-commands
   - These are pseudo-commands which have been reserved for future expansion. These pseudo-commands have no effect on the assembly processing operations.
   - This group consists of the 9 following pseudo-commands:-

.PROGNAME	Program name declaration
.IO .ENDIO .RAM .ENDRAM	Area name declarations
.PROCMAIN .PROCSUB .PROCINT	Module name declarations
.ENDPROC	Module termination declaration

The following pages describe the functions of the pseudo- commands group by group.

6.2 Assembly controls

1. Assebly assertion declaration
   .ASSERT
   Displays the character string specified as operand on the screen during assembly.

2. Assembly termination declaration
   .END
   Declares the termination of the source program. SRA74 will not process any data on subsequent lines.

3. Assembly error declaration
   .ERROR
   This pseudo-command causes the character string in the operand to be displayed on the VDU and the assembly operation to be terminated.

4. Conditional assembly
   .IF (.ELSE) .ENDIF
   Specifies the assembly location in accordance with the contents of the symbol value. These pseudo-commands can be used for various operations such as the handling of programs with varying specifications by means of a single source program or the control of test routine assembly.

5. Read file in
   .INCLUDE
   The contents of another file are read in to the location in which this command is entered. It can be used for splitting up and editing large source programs.

6.3 Address control

1. Synonymous definition
   .EQU or =
   Ascribes an absolute value to a symbol. It also defines the values of bits from 0-7 and 0000_{16} - $FFFF_{16}$ values.

2. Address declaration
   .ORG or *=
   Declares the addressing of the following lines. The section in which this pseudo-command is entered becomes an absolute attribute and cannot be addressed during link operations. It can be used for interrupt vectors and other areas with fixed addresses.

3. Securing an area

   .BLKB

   The amount of memory area specified in the operand is secured as RAM area.

4. Data setting

   .BYTE .WORD

   The data specified in the operand is generated in the ROM area.

6.4 Link control

1. Global label specification

   .BEXT .ZBEXT

   Specifies a bit symbol name for external reference. Any bit symbol name which is specified here must not have public specification in another file.

   .EXT .SEXT .ZEXT

   Specifies a label or symbol name for external reference. Any label which is specified here must not have public specification in another file.

   .PUB

   Labels, symbols or bit symbols which are defined in this file may be referred to from any other file.

2. Area specification

   .SECTION

   This pseudo-command specifies that the area from this line onwards shall have the name specified by the operand. The area attribute will be determined automatically by SRA74 in accordance with the subsequent command. This specification remains valid until a further area specification command is received.

   .SECTION P or .PMOD

   Specifies that the area from this line onwards is a general page ROM area. This specification remains valid until a further area specification command is received.

   .SECTION R or .RMOD

   Specifies that the area from this line onwards is a general page RAM area. This specification remains valid until a further area specification command is received.

   .SECTION S or .SMOD

   Specifies that the area from this line onwards is a special page ROM area. This specification remains valid until a further area specification command is received.

.SECTION Z or .ZMOD

Specifies that the area from this line onwards is a zero page RAM area. This specification remains valid until a further area specification command is received.

3. Link file name specification

   .OBJ .LIB

   Specifies the names of the link target R74 files and library files. Mnemonic command input of files named here is enabled during link operations to facilitate LINK74 (linker) automatic reference.

4. Version specification

   .VER

   Specifies the R74 file version. If the LINK74 “-V” command parameter is specified then the correspondence between R74 files can be verified.

6.5 List control

1. Page feed and title specification

   .PAGE

   Specifies the page feed and title for a list.

2. List format specification

   .COL .LINE

   Specifies the number of columns and lines in a list. These pseudo-commands may only be entered once into the source file.

3. List output/inhibit specification

   .LIST .NLIST

   Controls list output to the print file. These pseudo- commands should be used at times when only parts of lists are required such as during the debugging of part of a program.

4. Macro generation section list output/inhibit specification

   .LISTM .NLISTM

   Controls the output of lists of macro generation sections to the PRN file.

6.6 Debug support

SRA74 generates a source level debug data when the command parameter “-C” is specified. However, it is also possible to generate source line debug data only for statements enclosed between the .FUNC and .ENDFUNC statements. Memory usage on the host machine can be reduced by limiting the generation of debug data to the necessary statements. The command parameter “-C” can still be used for source file containing the .FUNC pseudo command to generate the source line debug data for the entire file.

1. Function start specification
   .FUNC
   Specifies the start of the “function”.

2. Function end specification
   .ENDFUNC
   Specifies the end of the “function”.

Note:

1. When specifying -C for assembly with SRA74, the .FUNC and .ENDFUNC pseudo-command lines cannot be checked. Accordingly, errors related to these pseudo-command lines cannot be detected.

6.7 Reserved pseudo-commands

Reserved pseudo-commands are pseudo-commands which are being reserved to enable the later expansion of SRA74. These pseudo-commands can be entered in the source file without resulting in error. They do not affect the outcome of the assembly operation in any way. Generally speaking they can be used in combination with any of the character string retrieval programs on general sale in order, for example, to confirm the contents of the source file.

If the pseudo-command .PROCMAIN is entered into the source file as shown below then retrieval of the main program can be carried out by using a character string retrieval program to retrieve “.PROCMAIN”.

Example:
```
.PROCMAIN     KEY-SCAN  ; Key scan program section entry
```

CHAPTER 7
Macro Commands

7.1 Functions of macro commands

The definition of the 740 Family programs, which use assembly language and structured language, in terms of macro commands enables the user to enter the assigned names (macro names) into the source program operand sections in just the same way as he enters op-codes or structured commands. It is possible, therefore, by preparing a variety of macro definitions, to use the MELPS 740 as a new, extended CPU for use when programming. In this way the macro command functions provide the user with a way of fine tuning his own programming environment.

7.2 Types of macro command

Macro commands are divided into 2 broad areas, those which come with SRA74 and those which are defined by the user himself.

1. System macro commands
   - .REPEATI - .ENDM
     Repeats the processing operation in accordance with the number of arguments contained in the operand.
   - .REPEATC - .ENDM
     Repeats the processing operation in accordance with the number of characters presented by the operand as arguments.
   - .REPEAT - .ENDM
     Repeats the processing operation a specific number of times.

2. User macro commands
   - .MACRO - .ENDM
     Defines the macro commands.
   - .EXITM
     Forces the termination of macro generation.
   - .LOCAL
     Changes the labels used in the macro definitions into intra- macro local labels.

For further details please refer to Appendix E.

Notes:

1. User macro commands must be defined prior to use. Macro commands should normally therefore be defined at the beginning of a program or alternatively a macro definition file should be inserted at the head by means of the pseudo- command .INCLUDE. User macro definitions may be nested up to 20 levels (however, this depends on the available host machine memory).

2. If the macro definition file is treated as a separate file (macro library) then the macro can be used simply by “including” it in the beginning of a program and it ceases to be necessary to enter the macro definition into each program.

3. Macro generated sections are indicated by a + at the side of the source in the print file.

4. LOCAL labels are assembled in order by allocation of the label ..n (where n is a decimal number between ..0 - ..65535). Care should be taken by the user since he may not use the label ..n.

5. When a pair of square brachets [ ] or curry braces { } indicating memory reference in structured description are used for the macro argument in each macro call, enclose [ ] or { } in double quotation marks.
   Example:

```
mac:   .macro   para1, para2
       para1 = para2
       .endm
mac    "[work]",10h
```

7.3 Macro operators

Table 7.1 lists the operators which can be used with macro commands.

Table 7.1 : List of macro operators

Operator	Description
\[1]	Causes recognition as argument of the character following the “\” which is placed in front of a special character which cannot be used as a macro argument. [format] \ character
;;[2]	Defines the comment which is not generated in a macro definition. [format] ;;comment
”[3]	Used when calling a macro where there is, for example, a space, tab, comma (,) or reserved word included in the argument. [format] ”character string”
$[4]	Used when there is concatenation of character strings either before or after the argument. [format] (1) argument $ character string (2) character string $ argument

Notes:

1. By acting as escape characters they negate the special meanings of the following characters:-

   Example:

   [Macro definition]

```
DATA: .MACRO    VAL
      .BYTE     VAL
      .ENDM
```

   [Call example]

```
      DATA "\"HELLO!\""
```

   [Macro generation]

```
      .BYTE"HELLO !"
      .ENDM
```

2. When outputting the result of a macro generation to the print file, comments preceded by a single semi-colon (";") will be output on the occasion of each macro generation whereas those preceded by a double semi-colon (";;") will not.

   Example:

   [Macro definition]

```
LOOP:   .MACRO
        .LOCAL  LOOP1
        LDA     #20             ;  comment
LOOP1:  DEC     A               ;; comment
        BNE     LOOP1           ;; comment
        .ENDM
```

   [Call example]

```
        LOOP
```

   [Macro generation]

```
        LDA     #20             ;  comment
..0:    DEC     A
        BNE     ..0
        .ENDM
```

3. When the argument supplied by the operand contains, for example, a space, tab, comma (,) or reserved word then everything should be enclosed within double quotes (”).

   Example:

   [Source entry]

```
SUB:        .REPEATI INST,”NOP”,”LDA #1”,”JSR SUB1”,”RTS”
            INST
           .ENDM
```

   [Macro generation]

```
SUB:
             NOP
             LDA #1
             JSR SUB1
             RTS
             .ENDM
```

4. Used when there is character string concatenation either before or after an argument and the name given by the argument is changed. There must be no space or tab inserted between the “$” and the character string.

   Example:

   [Source entry]

```
ADDWI:     .MACRO     MEM,IMM
      CLC
      LDA   MEM$_2
      ADC   #>IMM
      STA   MEM$_1
      LDA   #<IMM
      ADC   MEM$_2+1
      STA   MEM$_1+1
      .ENDM
```

   [Call example]

```
ADDWI RAM,1000H
```

   [Macro generation]

```
CLC
LDA   RAM_2
ADC   #>1000H
STA   RAM_1
LDA   #<1000H
ADC   RAM_2+1
STA   RAM_1+1
```

5. The strings enclosed in parentheses are treated as single argument.

   Example:

   [Macro definition]

```
ADDI: .MACRO    MEM,IMM
      LDA       MEM
      CLC
      ADC       #IMM
      STA       MEM
      .ENDM
```

   [Call example]

```
      ADDI      ( RAM,X ),5
```

   [Macro generation]

```
      LDA       ( RAM,X )
      CLC
      ADC       #5
      STA       ( RAM,X )
```

CHAPTER 8
Operation Method

8.1 Activation method

Before using SRA74 the following data (input parameters) must be entered:-

1. Source file name (required item)

2. Command parameters

With SRA74, these data can specified from the operating system command line or defined with the environment variable SRA74.

Section 8.2 describes the input parameters, section 8.3 describes how to enter the command with examples, and section 8.5 describes how to define the environment variable SRA74.

8.2 Input parameters

8.2.1 Source file name

1. Specifies the name of the source file which forms the object of the assembly operation. Only one source file may be specified.

2. When the file attribute (.A74) is omitted then the attribute .A74 is selected as the default value.

3. By specifying the full name of a file, files with attributes other than .A74 (eg. .ASM) may also be assembled.

4. The directory path can also be specified within the file name. If only the file name is specified then the processing operation will be carried out on a file contained within the current directory in the current drive.

Example: `A>SRA74 C:\WORK\TEST<RET>`

8.2.2 Command parameters

1. A command parameter consists of a minus sign followed by one or two characters.

2. Both upper case and lower case characters are remain valid for command parameters.

3. Each parameter is capable of multiple specification at the same time. In such cases each parameter should be separated from the next by a space prior to entry.

4. Two consecutive minus signs can be used to disable a command parameter. For example, --L suppresses the generation of PRN file.

5. Command parameters specified with the environment variable SRA74 are processed first.

Table 8.1 describes the contents of the command parameters.

Table 8.1 : List of command parameters

Command parameter	Description
-.	Inhibits the output of messages other than error messages to the VDU. This parameter should be used when using SRA74 with batch files, for example, and only error messages are required to be shown on the screen.
-!8	Makes the calculation result the lower 8 bits of the value after carrying out a ! operation.
-A	Specifies that the program is entered only in assembly language[1].
-BANK	Expands the address area upper limit from FFFFH to 1FFFFH. Operators BK and BL can be used. Section E information is not output to relocatable files or list files.
-C	Outputs source line debug data for all lines in the file.
-D	Sets a symbol's numerical value. The function of the command is equivalent to that of the pseudo-command .EQU. The specification format is as follows (When defining more than one symbol at a time each symbol and value entry should be separated from the next by a ":"): -Dsymbol=numerical value [:symbol=numerical value....: symbol=numerical value] Example: A>SRA74 SRCFILE -DS1=10:S2=20<RET>
-E	Generates the TAG file and activates the editor. The format for specification of the editor's program name is as follows[2]: -E[editor name] Example: A>SRA74 SRCFILE -EMI<RET> The part contained within the brackets [] may be omitted. If omitted, however, only a TAG file will be generated. In cases where the editor name has been specified the editor will be activated with the TAG file as argument after completion of the assembly operation. If, however, no error occurs then the editor will not be activated even though its name has been specified.
-I	Generates the source file (.I) in which the structured language commands are expanded into assembly language commands. When this parameter is specified at the same time as -L then the expanded sections will also be output to the print file which has been generated.

Command parameter	Description
-K	Suppresses the output of label information (such as ".IO") generated by SRA74 to the R74 file. This causes only user defined local labels to be output when the -S option is specified.
-L	Generates a PRN file. A PRN file will not be generated without this specification.
-LS	Generates a PRN file with symbol list.
-M	Outputs the result of macro generation to the PRN file. Without this specification the result of the macro generation will not be output into the list.
-O	Specifies the output destination path for the generated file. Both the directory and the drive name can be specified in the path. In the absence of such specification output will use the same path as the source file. The specification format is as follows: -O path name Example: `A>SRA74 SRCFILE -OB:\USR<RET>`
-S	Outputs local bit symbols, symbols and labels to the R74 file.
-U	Ignores the label entry ":" (colon). Without such specification and in cases where the label was entered by the program then the absence of the ":" will be treated as an error.
-X	Activates the cross-referencer after the assembly operation has been completed[3]. Example: `A>SRA74 SRCFILE -X<RET>`

Notes:

1. When this specification is made then SRA74 will not create temporary files during the assembly operation. The duration of the assembly operation will therefore be reduced.
   This command parameter should also be specified when an I file is input as the source file.

2. If there is no CRF74 in the current directory or in the command path then this will register as a system error.

8.3 Method of entry

SRA74 is activated by the input of command lines at the operating system prompt. Fig 8.1 shows an example of the input of activation command lines.
If a mistake is detected during the input of the command lines then a help page is displayed as shown in Fig 8.2 and the assembly operation is halted. If, on the other hand, the command lines have been input correctly then the assembly operation will be initiated. On completion of the assembly operation the number of errors, the number of warnings, the total number of lines, the number of comment lines and the section level memory capacity will all be displayed on the screen. An example of the appearance of the VDU screen following the successful completion of an assembly operation is shown in Fig 8.3.

```
A>SRA74 FILENAME -L -E <RET>
           ↓        ↓
Name of source file  Command parameter
to be assembled
```

Fig 8.1 : Example of the input of activation commands

```
A>SRA74<RET>
740 Family SRA74 V.4.00.00
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Usage: sra74 <filename> [ options ]
       -.   : all message suppressed
       -a   : assmble assembly language source file
       -c   : source line data output to .R74 file
       -d   : define symbols ( syntax -ds1=1:s2=2 )
       -e   : make tag file and start editor ( syntax -e or -eeditor name )
       -i   : make assembler source file
       -k   : suppress system label information to R74 file
       -l[s]: make list file or symbol list file
       -m   : macro listing
       -o   : select drive and directory for output ( syntax -otmp )
       -s   : local symbol data output to .R74 file
       -u   : don't care ':' at end of label
       -x   : execute crf74
A>
```

Fig 8.2 : Help page for command error

```
A>SRA74 TEST<RET>
740 Family SRA74 V.4.00.00
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

now processing pass 1  ( TEST.A74 )
----*----
now processing pass 2  ( TEST.A74 )
----*----

ERROR   COUNT           00000
WARNING COUNT           00000
STRUCTURED STATEMENT    00702 LINES
TOTAL   LINE ( SOURCE ) 00994 LINES
TOTAL   LINE ( OBJECT ) 00994 LINES
COMMENT LINE ( SOURCE ) 00247 LINES
COMMENT LINE ( OBJECT ) 00147 LINES
OBJECT  SIZE ( Z      ) 00010 (000A) BYTES
OBJECT  SIZE ( R      ) 00053 (0035) BYTES
OBJECT  SIZE ( P      ) 01938 (0792) BYTES

A>
```

Fig 8.3 : VDU display on normal termination

8.4 Errors

8.4.1 Types of error

The types of errors which occur during the operation of SRA74 have the following causes:-

1. Errors relating to the operating system

   These are errors such as insufficient disc or memory capacity which relate to SRA74's operating system. Please refer to the list of error messages in Appendix A and proceed according to the operating system commands.

2. Errors relating to the input of SRA74 command lines

   These are errors relating to the input of the command lines to activate SRA74. Please study the contents of this chapter and then reinput the relevant commands.

3. Errors relating to the contents of the source file constituting the object of the assembly operation

   These are errors relating to the source file such as the dual definition of a label or reference to a symbol which has not been defined. Please correct the relevant contents of the source file and carry out the assembly operation again. When an assembly error is detected the R74 file will not be generated.

Whenever an error or warning is detected SRA74 outputs the contents (file name, line number in file, consecutive line number, error number and error message) of the error onto the VDU screen and the print file in the format shown in Fig 8.4. Please study the list of errors in error number order in Appendix A and take action accordingly.

```
A>SRA74 TEST<RET>
740 Family SRA74 V.4.00.00
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

now processing pass 1  ( TEST.A74 )
----*----
now processing pass 2  ( TEST.A74 )
-
  115 E025  EAEA        BCC     LOOP2
TEST.A74 115 ( TOTAL LINE 115 ) Error 18: Relative jump is out of range
  127 E031  EAEA        LDA     #data
TEST.A74 127 ( TOTAL LINE 127 ) Error 22: Value is out of range "data"
---*
  551 E42B  EAEA        BRA     TEST2
TEST.A74 551 ( TOTAL LINE 551 ) Error 18: Relative jump is out of range
  593 E4FC  EAEAEA      LDA     (work,x               ; data set
TEST.A74 593 ( TOTAL LINE 593 ) Error 23: "()" format error ";"
----

ERROR   COUNT           00004
WARNING COUNT           00000
STRUCTURED STATEMENT    00702 LINES
TOTAL   LINE ( SOURCE ) 00994 LINES
TOTAL   LINE ( OBJECT ) 00994 LINES
COMMENT LINE ( SOURCE ) 00247 LINES
COMMENT LINE ( OBJECT ) 00147 LINES
OBJECT  SIZE ( Z      ) 00010 (000A) BYTES
OBJECT  SIZE ( R      ) 00053 (0035) BYTES
OBJECT  SIZE ( P      ) 01938 (0792) BYTES

A>
```

Fig 8.4 : Example of error display

8.4.2 Value for return to operating system

When entering an execution command into an operating system batch file, for example, there are times when you may wish to change the contents of the processing operation in accordance with the outcome of the execution. In SRA74 the execution results are divided into 4 error levels as shown in Table 8.2 and returned to the operating system.

Table 8.2 : List of error levels

Error level	Description of execution results
0	Normal termination
1	Error relating to content of source file which constitutes the object of the assembly operation
2	Error relating to input of SRA74 commands
3	Error relating to the operating system
4	Forced termination caused by ^C (cntl-C).

8.5 Environment variables

SRA74 makes use of the following environment variables:-

- TMP
  Specifies the name of the directory for creation of the temporary files generated during the assembly operation. In cases where this environment variable has not been set the temporary files will be generated in the same directory as the source file. An example of the setting of an environment variable is shown below:

  Example: `set TMP=A:\TMP`

- INC
  Specifies the name of the directory holding the files to be included during the assembly operation. In cases where this environment variable is not specified, files to be read in by means of the pseudo-command .INCLUDE are retrieved from the current directory. An example of the setting of this environment variable is shown below:

  Example: `set INC=A:\INC74`

- SRA74
  This environment variable can be used to specify the command name and command parameters normally entered from the command line. An example of this command variable is shown below:

  Example: `SET SRA74=-L -I`

  This environment variable can be used to define frequently used command paraemters thus eliminating the need to enter them from the command line each time. The command parameter defined with this environment varialbe can be disabled from the command line if it is unnecessary.

  The specifications made with the environment variable SRA74 are processed before those on the command line. For example, the following specifications (PC):

  ```
  SET  SRA74=-L -I
  SRA74  FILE --L -S
  ```

  are equivalent to the following specification from the command line:

  ```
  SRA74  -L  -I  FILE  --L  -S
  ```

  In other words, the command parameter “-L” is disabled with the specification from the command line and no PRN file is generated.

APPENDIX A
Error Messages

A.1 System errors

When a system error is detected during the assembly operation an error message is displayed on the VDU and assembly is suspended. Table A.1 lists all the system errors.

Table A.1 : List of system errors

Error Message	Description and User Action
Usage: sra74 <filename> [options]	The command has been input wrongly. ⇒ Refer to the help page and input the command again.
Can't open xxx[1]	File not found. ⇒ Check the source file name and input again.
Can't create xxx	This file cannot be generated. ⇒ Check the specification of the -O parameter and input again.
Out of disk space	There is not enough space left on the disk selected for output. ⇒ Create more space on the disk.
Out of heap space	The memory space required for the operation of the assembler is insufficient[2]. ⇒ Reduce the number of symbols or labels.
Can't find crf74.exe	The CRF74 cannot be found. ⇒ Copy the CRF74 into the current directory or else into a directory in the command path of the operating system.
Can't find command.com for execute xxx	The COMMAND.COM file which is required to activate the editor specified by the -E option cannot be found. ⇒ Check the specification of the operating system command path.

Notes:

1. The total number of symbols and labels handled by SRA74 is dependent on the memory capacity of the SRA74 operating system.

A.2 List of assembly errors

When an assembly error is detected an error message is output to both the VDU and the PRN file. Table A.2 contains a list of assembly errors with descriptions of their contents.

Table A.2 : List of assembly errors

Error number	Error Message	Description and User Action

1	Already had same statement	A pseudo-command which should only be used once in the source file has been used more than once.
		Example: `.LINE 60`
		`:`
		`.LINE 80`
		⇒ Reduce the number of times the declaration is used to one.

2	Reference to forward label or symbol	A pseudo-command refers to a forward label or symbol. Example: `.ORG TOP` `TOP:` ⇒ Carry out the definition of the label or symbol prior to the reference.
3	Division by 0	A numerical expression contains a division by 0. ⇒ Check the entry of the numerical expression.

Error number	Error Message	Description and User Action
4	Illegal operand	The operand contains a character which should not have been used. Example: `LDA #&10` ⇒ Check the entry of the operand.
5	Improper operand type	The combination of mnemonic and operand is at fault. Example: `ASL work,Y` ⇒ Check the entry format of the command.
6	Invalid label definition	The definition of the label has been entered into an improper area. Example 1: `LABEL1: .LINE 60` ⇒ Delete the label. Example 2: `LABEL2: .EQU 100` ⇒ Change the label to a symbol.
7	Invalid symbol definition	The definition of the symbol has been entered into an improper area. Example: `SYMBOL .LINE 60` ⇒ Delete the symbol.
8	Out of maximum program size	The address is outside 0FFFF16. Example: `.ORG 0FFF0H` `.WORD 1,2,3,4,5,6,7,8,9` ⇒ Modify the program such that the address falls within the permissible boundaries.
9	Label or symbol is multiple defined	The same label or symbol is defined more than once. Example: `MAIN: NOP` `MAIN: NOP` ⇒ Check the label or symbol name.
10	Nesting error	The maximum nesting level has been exceeded. ⇒ Reduce the number of nests to within the allowed maximum.

Error number	Error Message	Description and User Action
11	No .END statement	There is no .END statement in the source file. ⇒ Enter an .END statement at the end of the program.
12	No symbol definition	The symbol has not been entered. Example: `.EQU 60` ⇒ Enter the symbol.
13	No ';' at the top of comment	There is no ';' at the head of the comment section. Example: `LDA #CNT` counter set ⇒ Add a ';' at the head of the comment section.

14	Not in conditional block	~.ELSE or .ENDIF has been entered regardless of the absence of a corresponding
		~.IF statement.
		(This error will also occur when the corresponding .IF statement is in error.)
		Example:
		`.IF DATA1`
		`:`
		`.ENDIF`
		`:`
		`.ELSE`
		`:`
		`.ENDIF`
		⇒ Check the .IF statement.

15	Operand is expected	The operand required by the command is insufficient. Example: `.BYTE` ⇒ Check the entry of the operand.
16	Questionable syntax	The mnemonic is spelled incorrectly. Example: `ADD #DATA` ⇒ Check the spelling of the mnemonic.

17	Reference to multi defined label or symbol	Reference has been made to a label or symbol which has been defined more than once.
		Example:
		`MAIN: NOP`
		`MAIN: NOP`
		`BRA MAIN`
		⇒ Check the name of the label or symbol.

Error number	Error Message	Description and User Action
18	Relative jump is out of range	The jump destination address of the relative jump command is outside the permissible boundaries. ⇒ Either remap the program or modify the command.
19	Label or symbol is reserved word	A name which is identical to a reserved word has been used in the label or symbol. Example: `A .EQU 1FFH` ⇒ Modify the name of the label or symbol.
20	Reference to undefined label or symbol	Reference is being made to a label or symbol which has not been defined. ⇒ Check the label or symbol.
21	Value error	The data entry format is incorrect Example : `ADC #'A` ⇒ Check the data entry format
22	Value is out of range	The data boundaries are outside the permissible range. Example: `ADC #100H` ⇒ Check the operand entry format.
23	“()” format error	The number of left brackets ‘(‘ and right brackets ‘)’ does not match. Example: `ADC (WORK` ⇒ Check the operand entry format.

24	Relocatable error	The pseudo-command .ORG has been entered into a relocatable section.
		Example:
		`.SECTION PROG`
		`LDA WORK`
		`:`
		`.ORG 1000H`
		`:`
		⇒ Separate the sections.

25	No .SECTION statement	The pseudo-command .SECTION has not been entered. ⇒ Enter the pseudo-command .SECTION prior to program entry.

Error number	Error Message	Description and User Action
26	Reference to undefined section	Reference is being made to a section name which has not been defined. Example: `LDA #SIZEOF UNDEF` ⇒ Check the relevant section.
27	Section type mismatch	A command or data set pseudo-command (.BYTE, etc.) has been mixed up with an area secure command (.BLKB, etc.). Example : `LDA #WORK` `.BLKB 1` ⇒ Separate the sections.
28	Constant value is required	Relative attribute label or external reference symbol cannot be used. ⇒ Use an absolute value.
30	else not associated with if	There is no “if” to correspond with “else”. ⇒ Check the program.
31	endif not associated with if	There is no “if” to correspond with “endif”. ⇒ Check the program.
32	next not associated with for	There is no “for” to correspond with “next”. ⇒ Check the program.
33	while not associated with do	There is no “do” to correspond with “while”. ⇒ Check the program.
34	break not inside for, do or switch	“break” has been used in an improper location. ⇒ Check the program.
35	case not inside switch	“case” has been entered outside the “switch” range. ⇒ Check the program.
36	duplicate case value	The “case” value has been used more than once. ⇒ Check the program.
37	more than one default	There is more than one “default” in the same “switch” statement. ⇒ Check the program.
38	default not inside switch	“default” has been entered outside the “switch” statement range. ⇒ Check the program.
39	ends not associated with switch	There is no “switch” to correspond with “ends”. ⇒ Check the program.
40	continue not inside for or do	“continue” has been entered into an improper location. ⇒ Check the program.

A.3 List of warnings

When a warning is detected a warning message is output to the VDU and the PRN file. Table A.3 lists all the warning messages along with a description of their contents.

Table A.3 : List of warnings

Warning number	Warning message	Description and user action

1	Phase warning	1) The address specified by pseudo-command .ORG precedes an address prior to that one.
		
		Example:
		`.ORG 0E000H`
		`MAIN: LDA WORK`
		`:`
		`.ORG 0C000H`
		
		2) The command is referring to a label or symbol which is defined after this line.
		
		Example:
		`LDA WK,X`
		`:`
		`WK .EQU 80H`
		
		⇒ Define the label or symbol prior to the line which refers to it.

2	.END statement in include file	The pseudo-command .END has been entered into the include file. ⇒ Enter .END into the source file.
3	statement has no effect	The statement has no meaning as a statement. ⇒ Check the program.
4	not case values for switch statement	The “switch” statement does not contain any “case” values. ⇒ Check the program.
5	statement not preceded by case or default	The statement comes before “case” or “default” in a “switch” statement. ⇒ Check the program.
6	.EQU symbol is multiple defined	Pseudo-command .EQU is defined more than once with the same symbol.
7	'.SECTION E' requires command option '-BANK'	Programs section E. Specify the -BANK command option to assemble programs.

APPENDIX B
List of Commands

B.1 List of symbols

Table B.1 indicates the significance of the symbols used in the list of commands.

Table B.1 : List of symbols

Symbol	Description	Symbol	Description
A	Accumulator	X	Index register X
Y	Index register Y	imm	Immediate data
zz	Zero page address	hhll	General page address
i	Bit value (0-7)	bitsym	Bit symbol
#	Immediate mode	\	Special page mode

B.2 List of commands

Table B.2 lists all the commands which can be used in SRA74. Its addressing mode name and entry format is given alongside each command.

Table B.2 : List of commands

Command	Addressing mode	Entry format	
ADC	Immediate	ADC	#imm
	Zero page	ADC	zz
	Zero page X	ADC	zz,X
	Zero page indirect X	ADC	(zz,X)
	Zero page indirect Y	ADC	(zz),Y
	Absolute	ADC	hhll
	Absolute X	ADC	hhll,X
	Absolute Y	ADC	hhll,Y
AND	Immediate	AND	#imm
	Zero page	AND	zz
	Zero page X	AND	zz,X
	Zero page indirect X	AND	(zz,X)
	Zero page indirect Y	AND	(zz),Y
	Absolute	AND	hhll
	Absolute X	AND	hhll,X
	Absolute Y	AND	hhll,Y
ASL	Accumulator	ASL	A
	Zero page	ASL	zz
	Zero page X	ASL	zz,X
	Absolute	ASL	hhll
	Absolute X	ASL	hhll,X
BBC	Accumulator bit relative	BBC	i,A,hhll
		BBC	bitsym,A,hhll
	Zero page bit relative	BBC	i,zz,hhll
		BBC	bitsym,hhll
BBS	Accumulator bit relative	BBS	i,A,hhll
		BBS	bitsym,A,hhll
	Zero page bit relative	BBS	i,zz,hhll
		BBS	bitsym,hhll

Command	Addressing mode	Entry format	
BCC	Relative	BCC	hhll
BCS	Relative	BCS	hhll
BEQ	Relative	BEQ	hhll
BIT	Zero page	BIT	zz
	Absolute	BIT	hhll
BMI	Relative	BMI	hhll
BNE	Relative	BNE	hhll
BPL	Relative	BPL	hhll
BRA	Relative	BRA	hhll
BRK	Implied	BRK	
BVC	Relative	BVC	hhll
BVS	Relative	BVS	hhll
CLB	Accumulator bit	CLB	i,A
		CLB	bitsym,A
CLB	Zero page bit	CLB	i,zz
		CLB	bitsym
CLC	Implied	CLC	
CLD	Implied	CLD	
CLI	Implied	CLI	
CLT	Implied	CLT	
CLV	Implied	CLV	
CMP	Immediate	CMP	#imm
	Zero page	CMP	zz
	Zero page X	CMP	zz,X
	Zero page indirect X	CMP	(zz,X)
	Zero page indirect Y	CMP	(zz),Y
	Absolute	CMP	hhll
	Absolute X	CMP	hhll,X
	Absolute Y	CMP	hhll,Y
COM	Zero page	COM	zz

Command	Addressing mode	Entry format	
CPX	Immediate	CPX	#imm
	Zero page	CPX	zz
	Absolute	CPX	hhll
CPY	Immediate	CPY	#imm
	Zero page	CPY	zz
	Absolute	CPY	hhll
DEC	Accumulator	DEC	A
	Zero page	DEC	zz
	Zero page X	DEC	zz,X
	Absolute	DEC	hhll
	Absolute X	DEC	hhll,X
DEX	Implied	DEX	
DEY	Implied	DEY	
DIV	Zero page X	DIV	zz,X
EOR	Immediate	EOR	#imm
	Zero page	EOR	zz
	Zero page X	EOR	zz,X
	Zero page indirect X	EOR	(zz,X)
	Zero page indirect Y	EOR	(zz),Y
	Absolute	EOR	hhll
	Absolute X	EOR	hhll,X
	Absolute Y	EOR	hhll,Y
FST	Implied	FST	
INC	Accumulator	INC	A
	Zero page	INC	zz
	Zero page X	INC	zz,X
	Absolute	INC	hhll
	Absolute X	INC	hhll,X
INX	Implied	INX	
INY	Implied	INY	
JMP	Absolute	JMP	hhll
	Zero page indirect	JMP	(zz)
	Absolute indirect	JMP	(hhll)

Command	Addressing mode	Entry format	
JSR	Absolute	JSR	hhll
	Special page	JSR	\hhll
	Zero page indirect	JSR	(zz)
LDA[1]	Immediate	LDA	#imm
	Zero page	LDA	zz
	LDA	bitsym	
	Zero page X	LDA	zz,X
	Zero page indirect X	LDA	(zz,X)
	Zero page indirect Y	LDA	(zz),Y
	Absolute	LDA	hhll
		LDA	bitsym
	Absolute X	LDA	hhll,X
	Absolute Y	LDA	hhll,Y
LDM	Zero page	LDM	#imm,zz
LDX	Immediate	LDX	#imm
	Zero page	LDX	zz
	Zero page Y	LDX	zz,Y
	Absolute	LDX	hhll
	Absolute Y	LDX	hhll,Y
LDY	Immediate	LDY	#imm
	Zero page	LDY	zz
	Zero page X	LDY	zz,X
	Absolute	LDY	hhll
	Absolute X	LDY	hhll,X
LSR	Accumulator	LSR	A
	Zero page	LSR	zz
	Zero page X	LSR	zz,X
	Absolute	LSR	hhll
	Absolute X	LSR	hhll,X
MUL	Zero page X	MUL	zz,X
NOP	Implied	NOP	

Note:

1. When a bit symbol is entered into the operand of an LDA command only the address of the bit symbol is valid.

Command	Addressing mode	Entry format	
ORA	Immediate	ORA	#imm
	Zero page	ORA	zz
	Zero page X	ORA	zz,X
	Zero page indirect X	ORA	(zz,X)
	Zero page indirect Y	ORA	(zz),Y
	Absolute	ORA	hhll
	Absolute X	ORA	hhll,X
	Absolute Y	ORA	hhll,Y
PHA	Implied	PHA	
PHP	Implied	PHP	
PLA	Implied	PLA	
PLP	Implied	PLP	
ROL	Accumulator	ROL	A
	Zero page	ROL	zz
	Zero page X	ROL	zz,X
	Absolute	ROL	hhll
	Absolute X	ROL	hhll,X
ROR	Accumulator	ROR	A
	Zero page	ROR	zz
	Zero page X	ROR	zz,X
	Absolute	ROR	hhll
	Absolute X	ROR	hhll,X
RRF	Zero page	RRF	zz
RTI	Implied	RTI	
RTS	Implied	RTS	
SBC	Immediate	SBC	#imm
	Zero page	SBC	zz
	Zero page X	SBC	zz,X
	Zero page indirect X	SBC	(zz,X)
	Zero page indirect Y	SBC	(zz),Y
	Absolute	SBC	hhll
	Absolute X	SBC	hhll,X
	Absolute Y	SBC	hhll,Y
SEB	Accumulator bit	SEB	i,A
		SEB	bitsym,A

Command	Addressing mode	Entry format
SEB	Zero page bit	SEB i,zz
		SEB bitsym
SEC	Implied	SEC
SED	Implied	SED
SEI	Implied	SEI
SET	Implied	SET
SLW	Implied	SLW
STA[1]	Zero page	STA zz
		STA bitsym
	Zero page X	STA zz,X
	Zero page indirect X	STA (zz,X)
	Zero page indirect Y	STA (zz),Y
	Absolute	STA hhll
		STA bitsym
	Absolute X	STA hhll,X
	Absolute Y	STA hhll,Y
STP	Implied	STP
STX	Zero page	STX zz
	Zero page Y	STX zz,Y
	Absolute	STX hhll
STY	Zero page	STY zz
	Zero page X	STY zz,X
	Absolute	STY hhll
TAX	Implied	TAX
TAY	Implied	TAY
TST	Zero page	TST zz
TSX	Implied	TSX
TXA	Implied	TXA
TXS	Implied	TXS
TYA	Implied	TYA
WIT	Implied	WIT

Note:

1. When a bit symbol is entered into the operand of the STA command only the address of the bit symbol is valid.

APPENDIX C
Commands Listed by Addressing Mode

C.1 List of commands by addressing mode

Listed below are command coding formats by addressing mode. The symbols used in the description are listed in the symbol table in Appendix B.1.

1. Implied

Commands	Coding Format
BRK, CLC, CLD, CLI, CLT, CLV, DEX, DEY, INX, INY, NOP, PHA, PHP, PLA, PLP, RTI, RTS, SEC, SED, SEI, SET, SLW, STP, TAX, TAY, TSX, TXA, TXS, TYA, WIT	BRK

2. Immediate

Commands	Coding Format
ADC, AND, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA, SBC	ADC #imm

3. Accumulator

Commands	Coding Format
ASL, DEC, INC, LSR, ROL, ROR	ASL A

4. Zero page

Commands	Coding Format
ADC, AND, ASL, BIT, CMP, COM, CPX, CPY, DEC, EOR, INC, LDA, LDX, LDY, LSR, ORA, ROL, ROR, RRF, SBC, STA, STX, STY, TST	ADC zz
LDM	LDM #imm,zz

5. Zero page index X

Commands	Coding Format
ADC, AND, ASL, CMP, DEC, DIV, EOR, INC, LDA, LDY, LSR, MUL, ORA, ROL, ROR, SBC, STA, STY	ADC zz,X

6. Zero page index Y

Commands	Coding Format
LDX, STX	LDX zz,Y

7. Zero page indirect

Commands	Coding Format
JMP, JSR	JMP (zz)

8. Zero page indirect index X

Commands	Coding Format
ADC, AND, CMP, EOR, LDA, ORA, SBC, STA	ADC (zz,X)

9. Zero page indirect index Y

Commands	Coding Format
ADC, AND, CMP, EOR, LDA, ORA, SBC, STA	ADC (zz),Y

10. Absolute

Commands	Coding Format
ADC, AND, ASL, BIT, CMP, CPX, CPY, DEC, EOR, INC, JMP, JSR, LDA, LDX, LDY, LSR, ORA, ROL, ROR, SBC, STA, STX, STY	ADC hhll

11. Absolute index X

Commands	Coding Format
ADC, AND, ASL, CMP, DEC, EOR, INC, LDA, LDY, LSR, ORA, ROL, ROR, SBC, STA	ADC hhll,X

12. Absolute index Y

Commands	Coding Format
ADC, AND, CMP, EOR, LDA, LDX, ORA, SBC, STA	ADC hhll,Y

13. Absolute indirect

Commands	Coding Format
JMP	JMP (hhll)

14. Special page

Commands	Coding Format
JSR	JSR \hhhll

15. Zero page bit

Commands	Coding Format
CLB, SEB	CLB i,zz CLB bitsym

16. Accumulator bit

Commands	Coding Format
CLB, SEB	CLB i,A CLB bitsym,A

17. Relative

Commands	Coding Format
BCC, BCS, BEQ, BMI, BNE, BPL, BRA, BVC, BVS	BCC hhll

18. Zero page bit relative

Commands	Coding Format
BCC, BBS	BBC i,zz,hhll BBC bitsym,hhll

19. Accumulator bit relative

Commands	Coding Format
BBC, BBS	BBC i,A,hhll
	BBC bitsym,hhll

APPENDIX D
The Pseudo-Commands

D.1 How to use the pseudo-command reference section

All the pseudo-commands which can be used in SRA74 are described in alphbetical order. Points which should be noted with regard to the notation style in this section are outlined below:-

1. The use of square brackets [] indicates that the section may be omitted as required.
2. △, ▲ are used to indicate the use of a space or a tab code. The △ represents a required section whereas the ▲ section may be omitted if necessary.
3. In the following descriptions the space between a label and a pseudo-command is indicated by a ▲. When entering a label a ":" (colon) is not absolutely necessary but if it is omitted then a space or tab code must be inserted between the label and a pseudo-command.

D.2 Summary of the pseudo-commands

.ASSERT — Assemble assertion directive

Format

▲.ASSERT △'character string'

Description

- Displays the character string to be entered into an operand on the VDU during the course of the assembly operation.

Example

```
.IF        MODE
 :
.ELSE
.ASSERT    'MODE is FALSE'
.ENDIF
```

.BEXT — External reference setting (General page bit symbol)

Format

▲.BEXT △bit symbol[,bit symbol,..,bit symbol]

Description

- Specifies that the bit symbol specified in the operand may be subject to external reference.
- A bit symbol specified by this pseudo-command is understood to be located on a general page.
- This pseudo-command should be entered before the line which refers to the label.

Example

```
.BEXT   BIT0,BIT1,BIT2
          :
```

.BLKB

RAM area secure (unit : byte)

Format

▲[label:] ▲.BLKB △numerical expression

Description

- Secures an area of a size specified by a numerical expression (unit : 1 byte).
- Any labels or symbols used in the numerical expression should be defined prior to the line in question.
- Labels with relocatable values cannot be entered into the operand.

Example

```
label:        .BLKB   10      ; Secures a 10 byte area.
                :
```

.BYTE

Byte data setting

Format

▲[label:] ▲.BYTE △numerical expression

Description

- Sets 1 byte of constant data.
- If more than 1 byte of data is to be set then each item of data must be separated by a “,”. Up to 255 items of data may be specified.
- Global labels may be entered into the operand.

Example

```
label1: .BYTE   10      ; Sets 0AH.
label2: .BYTE   'A','B' ; Sets 41H and 42H.
        :
```

.COL

Column number setting (Default setting : 512)

Format

▲.COL △numerical expression

Description

- Sets the number of columns (from 80 to 512) on one line of a list.
- A setting of 79 columns or less will be registered as 80 and a setting of 513 or more will be registered as 512.
- This pseudo-command can be used only once during a program.

Example

```
.COL    80      ; Sets the number of columns to 80.
:
```

.END

Program termination directive

Format

▲.END

Description

- Specifies the end of the source program.
- Lines following this pseudo-command will not form part of the assembly operation.

Example

```
:
.END            ; Directs the termination of the program.
```

.ENDFUNC

Function termination specification

Format

▲.ENDFUNC △label

Description

- Specifies the end of the function (subroutine).
- The use of this command enables a source line debug in the SDB74.

Example

```
        .FUNC   SUB
SUB:    LDA     #1
         :
         :
        RTS
        .ENDFUNC SUB   ; Specifies the end of the function.
```

.EQU (or else ‘=’)

Synonymous definition

Format 1

symbol △.EQU(or’=’)△numerical expression

Format 2

bit symbol △.EQU△(or’=’)△numerical expression,numerical expression

Description

- Allocates a numerical value to the left hand symbol.
- Format 1 is used to allocate a 16 bit integer value to the symbol. Format 2 is used to allocate 0-7 bit values and a 16 bit integer value (address) to the symbol.
- Any labels or symbols used in the numerical expression should be defined prior to the line in question.
- Labels with relocatable values may not be entered into the operand.
- Symbols may be redefined.
- If the command parameter “-Q’” is specified, a warning is issued when a symbol is redefined.

Example

```
symbol          .EQU 1    ;    1 is assigned to symbol
   :
symbol          .EQU 2    ;    2 is assigned to symbol
   :
symbol          .EQU 3    ;    3 is assigned to symbol
   :
bitsym          .EQU 1,23H;    bit 1 at 23H is assigned to bitsym
```

.ERROR — Assembly error directive

Format

▲.ERROR ▲'character string'

Description

- Used in combination with the conditional assembly command (.IF).
- If something is specified which cannot exist as a condition then the character string entered in the operand is displayed on the VDU and the assembly operation is terminated.

Example

```
.IF     MODE
     :
.ELSE
.ERROR  "undefined assemble mode'
.ENDIF
```

.EXT — External reference specification (General page)

Format

▲.EXT △label or symbol[,label or symbol,..,label or symbol]

Description

- Specifies that the label specified in the operand may make external reference in the absolute addressing mode. If you wish to carry out the assembly operation in the zero page addressing mode or in the special page addressing mode then the specification should be modified to .ZEXT or .SEXT respectively.
- This pseudo-command should be entered prior to the line in which reference is made to the label.

Example

```
.EXT    WORK1,WORK2,WORK3
  :
```

.FUNC

Function start specification

Format

▲.FUNC △label

Description

- Specifies the start of the function (subroutine)
- The use of this command enables a source line debugging.
- The label used with the pseudo-command ".FUNC" cannot be used on another ".FUNC".

Example

```
        .FUNC   SUB     ; Specifies the start of the function.
SUB:    LDA     #1
        :
        :
        RTS
        .ENDFUNC SUB
```

.IF (.ELSE) .ENDIF

Conditional assembly

Format

▲.IF△expression
<statement 1>
▲.ELSE
<statement 2>
▲.ENDIF

Description

- Numeric or string expression may be used as operand.
- When the operand is a numeric expression, assembles statement 1 if the result is true (not 0) or statement 2 if the result is false (0).
- When the operand is a string expression, assembles statement 1 if the string is true (string data) or statement 2 if the string is false (no string data).
- This command may be nested up to 20 levels. (Depending on the available host system memory.)
- Multiple lines may be specified for statements 1 and 2.
- Label with relocatable value may not be specified in the operand field.
- The operand may contain the following logical operators.

<	Less than
>	Greater than
==	Equal
!=	Not equal
<=	Less than or equal
>=	Greater than or equal

- If an undefined symbol is used in a numeric expression, it is treated as a symbol with value 0. If an undefined symbol is used in a string expression, it is treated as a character string.
- Up to six conditional expressions can be combined as operand using logical operators (‘||’, ‘&&’). However, the expressions have no priority and are evaluated from left to right.

Example

(1)

```
  .IF FLAG        ; Assembles lines through .ELSE if FLAG is true.
     :
     :
  .ELSE           ; Assembles lines through .ENDIF if FLAG is false.
     :
     :
  .ENDIF
```

```
(2)
 ADD: .MACRO     OP1,OP2,OP3
      .IF        "OP3"      ; Assemble through .ELSE if argument
                            ; OP3 exists
      ADC        OP1,OP2,OP3
      .ELSE
      ADC        OP1,OP2
      .ENDIF
```

```
(3)
 .IF  FLAG1 && FLAG2 || FLAG3 && FLAG4
 :  Whether to assemble this section depends on the
 :  truth table shown below.
 .ENDIF
```

FLAG1	FLAG2	FLAG3	FLAG4	Result
TRUE	TRUE	-	-	TRUE
TRUE	FALSE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	-	FALSE
FALSE	-	-	-	FALSE

"-’ may be either.

.INCLUDE — File read in

Format

▲.INCLUDE △file name

Description

- The contents of the file specified in the operand will be read into the location in which this pseudo-command is entered.
- The name of the file should always be specified in full.
- This pseudo-command can be nested up to 4 levels.
- The nesting level is output to the print file.

Example

```
.INCLUDE    TEST.INC    ; Reads in the contents of TEST.INC
:
```

.LIB — Library file name specification

Format

▲.LIB △file name[,file name,..,file name]

Description

- Specifies the name of a library file forming the target of a link operation.
- Only files with the attribute .LIB may be specified. The specification of any other file will result in an error being registered during the link operation.
- This pseudo-command cannot be nested.
- The directory path and file attribute (.LIB) may not be entered in the file name.

Example

```
 .LIB    LIB1,LIB2,LIB3
:
```

.LINE

Specification of number of lines per page (Default value : 54)

Format

▲.LINE △numerical expression

Description

- Specifies the number of lines (5-255) per page of a list.
- This pseudo-command may be used only once in a program.

Example

```
.LINE   60      ; Sets the number of lines to 60.
  :
```

.LIST

Initiate list output (Preset value)

Format

▲.LIST

Description

- Performs the output of a list to the PRN file.
- This is used to restart the output of a list following suspension of output to the PRN file by means of the pseudo- command .NLIST.

Example

```
.NLIST      ; Inhibits the output of the list.
 :          ; There will be no output to the PRN file until ".LIST".
 :
.LIST       ; Initiates the output of the list.
 :          ; Output is to the PRN files which follow this pseudo-command.
 :
```

.LISTM — Initiation of list output of macro generation section (Preset value)

Format

▲.LISTM

Description

- Outputs the macro command generation section to the PRN file.
- Used to restart list output following suspension of the macro generation section output by means of the pseudo- command .NLIST.
- In cases where the pseudo-command .NLIST has been used to inhibit total list output then the .LISTM command is invalid.

Example

```
.NLISTM  ; Inhibits output of the macro generation section list.
 :       ; The macro generation section up to ".LISTM" will not be output.
 :
.LISTM   ; Initiates the output of a macro generation section list.
 :       ; The macro generation section from this pseudo-command
 :       ; onwards will be output.
 :
```

.NLIST — List output inhibition

Format

▲.NLIST

Description

- Inhibits output to the PRN file.
- This condition can be reversed by means of the pseudo- command .LIST.

Example

```
.NLIST      ; Inhibits list output.
 :          ; There will be no output to the PRN file until ".LIST".
 :
.LIST       ; Initiates list output.
 :          ; Output will be made to the PRN file from this pseudo-command

 :          ; onwards.
 :
```

.NLISTM — Inhibition of macro generation section list output

Format

▲.NLISTM

Description

- The macro command generation section will not be output to the PRN file.
- This condition can be reversed by means of the pseudo- command .LISTM

Example

```
.NLISTM  ; Inhibits output of the macro generation section list.
 :       ; The macro generation section will not be output until ".LISTM".
 :
.LISTM   ; Initiates the output of the macro generation section list.
 :       ; The macro generation section from this pseudo-command
 :       ; onwards will be output.
 :
```

.OBJ — Relocatable file name specification

Format

▲.OBJ △file name[,file name,..,file name]

Description

- Specifies the name of a relocatable file which forms the target of a link operation.
- Only files with the attribute .R74 may be specified. The specification of any other file will result in the registration of an error during the link operation.
- This pseudo-command may not be nested.
- The directory path and file attribute (.R74) may not be entered in the file name.

Example

```
.OBJ   OBJ1,OBJ2,OBJ3
 :
```

.ORG (or else “*=”)

Address directive (Preset value : 0000H)

Format

▲.ORG(or”*=”)△numerical expression

Description

- Directs the initiation address from this line onwards.
- In the absence of any specification the initiation address will be 0000H.
- The section in which this pseudo-command is entered will become an absolute attribute. Any section not containing this pseudo-command will become a relative attribute.
- Any labels or symbols used in the numerical expression should be defined prior to the line in question.
- Labels with relocatable values may not be entered into the operand.

Example

```
.ORG    0C000H  ; Sets the location to C000H.
 :
*=      0E000H  ; Sets the location to E000H.
```

.PAGE

List page feed and title setting

Format

▲.PAGE △[‘title’]

Description

- A list page feed is carried out immediately prior to this command and the title specified in the operand is output to the header section of the list. The title should be enclosed within ’ (single quotes) or ” (double quotes) for entry.
- The maximum number of characters permitted in title is 20 if column specification is 80, 45 if column specification is from 105 to 512, or 60 subtracted from the number of columns if column specification is 81 to 104. If title is not specified, only skip to new page is performed.

Example

```
.PAGE   ’PROG1’; PROG1 will be output to the header section of the PRN file.
       :
```

.PMOD

ROM area specification (General page)

Format

▲.PMOD

Description

- Specifies that the area from this command onwards is a general page ROM area.
- This command is equivalent to the insertion of ‘P’ into the operand of the pseudo-command .SECTION.
- This specification will remain valid until another area specification command is received.
- In the absence of an area specification command at the head of a file then .PMOD will be selected as the standard value. Area specification commands may therefore be omitted in respect of the first ROM area of a file.

Example

```
                .PMOD
                .EXT    MAIN,SUB
                .PUB    ENZAN
    ENZAN:
                CLT
                :
```

.PUB

Public specification

Format

▲.PUB △bit symbol or symbol or label,....

Description

- Enables the bit symbol, symbol or label specified in the operand to be referred from any other source file.
- In a RAM area only file consisting only of Z section and R section, all labels are treated as global. Therefore, .PUB specification for these labels may be omitted.
- This pseudo-command should be entered prior to the line in which the label or symbol is defined.

Example

```
            .RMOD
            .PUB    WORK1,WORK2,WORK3
WORK1:      .BLKB   1
WORK2:      .BLKB   1
WORK3:      .BLKB   1
   :
```

.RMOD

RAM area specification (General page)

Format

▲.RMOD

Description

- Specifies that the area from this command onwards is a general page RAM area.
- This command is equivalent to the insertion of ‘R’ into the operand of the pseudo-command .SECTION.
- This specification will remain valid until another area specification command is received.
- In a RAM area only file consisting only of Z section and R section, all labels are treated as global. Therefore, .PUB specification for these labels may be omitted.

Example

```
            .RMOD
WORK1:      .BLKB   1
WORK2:      .BLKB   1
WORK3:      .BLKB   1
            :
```

.SECTION

Area specification

Format

▲.SECTION △section name

Description

- Specifies that the area from this line onwards will be the area with the name specified in the operand.
- When reserved section names (P, R, S, Z) have been entered in the operand SRA74 will recognize each as its own area attribute from this pseudo-command onwards and process them accordingly but where another optional section name has been entered then the area attribute will be determined in accordance with a later command. In such cases, however, the areas will only be recognized as general page ROM or RAM areas.
- This specification will remain valid until another area specification command is received.
- It is no problem to have more than one section with the same name in any file.
- In the absence of an area specification command at the head of a file, .SECTION P will be selected as the standard value. Area specification commands may therefore be omitted in respect of the first ROM area of a file.

Example

```
            .SECTION  DATA  ; Marks the start of the DATA section.
datatop:
nulldt:     .BLKB     8
            :
            .SECTION  STACK ; Marks the start of the STACK section.
            .BLKB     16
stacktop:
            :
            .SECTION  PROG ; Marks the start of the PROG section.
_init:
            LDX        #stacktop
            TXS
            LDA        #SIZEOF DATA
            LDX        #datatop
            :
```

.SEXT — External reference specification (Special page)

Format

▲.SEXT△label or symbol[,label or symbol,..,label or symbol]

Description

- Specifies that the label specified in the operand may be subject to external reference in the special page addressing mode (JSR \ only) or in the absolute addressing mode. If you wish to carry out the assembly operation in the zero page addressing mode then the specification should be made with .ZEXT.

- This pseudo-command should be entered prior to the line in which reference is made to the label.

Example

```
            .SECTION P
            .EXT     WORK1,WORK2
            .SEXT    SUB
START:
            LDA      WORK1
            JSR      \SUB
            :
```

.SMOD — ROM area specification (special page)

Format

▲.SMOD

Description

- Specifies that the area from this command onwards is a special page ROM area.

- This command is equivalent to the insertion of ‘S’ into the operand of the pseudo-command .SECTION.

- This specification will remain valid until another area specification command is received.

Example

```
            .SMOD
            .EXT   MAIN,SUB
            .PUB   ENZAN
ENZAN:
            CLT
            :
```

.VER — Program version specification

Format

▲.VER △'character string'

Description

- Specifies the version of a relocatable.
- If the parameter "-V" is specified then the LINK74 will check that there is correspondence between the version specifications in each relocatable file. It is thereby possible to check the version coordination between different relocatable files. For further information concerning the "-V" parameter please refer to the LINK74 operation manual.
- The correspondence between versions is checked by means of a comparison of character strings. Care should be taken when entering since a distinction is made between upper case and lower case characters.
- This pseudo-command may only be entered once during a program.

Example

```
        .VER    'V.1.0' ; Specifies version ''V.1.0''.
        :
```

.WORD — Word data specification

Format

▲[label:] ▲.WORD △numerical expression

Description

- Sets values using numerical expressions (unit : word).
- If more than one item of data is set then each item should be separated by a ",". A maximum of 16 items may be specified on a single line.
- The data is set from the low order byte.
- Global labels may be entered into the operand.

Example

```
label:     .WORD   0E000H  ; Sets 00H and E0H.
           .WORD   symbol  ; Sets the value of a symbol from the low order byte.
                :
```

.ZBEXT — External reference specification (Zero page bit symbol)

Format

▲.ZBEXT △bit symbol[,bit symbol,..,bit symbol]

Description

- Specifies that the bit symbol specified in the operand may be subject to external reference.
- A bit symbol specified by this pseudo-command is understood to be located on a zero page.
- This pseudo-command should be entered before the line which refers to the label.

Example

```
        .ZBEXT  BIT0,BIT1,BIT2
         :
```

.ZEXT — External reference specification (Zero page)

Format

▲.ZEXT △label or symbol[,label or symbol,..,label or symbol]

Description

- Specifies that the label specified in the operand may be the subject of external reference in the zero page addressing mode. If you wish to carry out the assembly operation in the absolute addressing mode or in the special page addressing mode then the specification should be modified to .EXT or .SEXT respectively.

- This pseudo-command should be entered prior to the line in which reference is made to the label.

Example

```
                .SECTION Z
                .ZEXT   WORK1,WORK2
                .SEXT   SUB
      START:
                LDA     WORK1
                JSR     \SUB
                :
```

.ZMOD — RAM area specification (Zero page)

Format

▲.ZMOD

Description

- Specifies that the area from this command onwards is a zero page RAM area.
- This command is equivalent to the insertion of ‘Z’ into the operand of the pseudo-command .SECTION.
- This specification will remain valid until another area specification command is received.
- In a RAM area only file consisting only of Z section and R section, all labels are treated as global. Therefore, .PUB specification for these labels may be omitted.

Example

```
            .ZMOD
WORK1:      .BLKB   1
WORK2:      .BLKB   1
WORK3:      .BLKB   1
            :
```

D.3 List of reserved pseudo-commands

The pseudo-commands listed below have been reserved to facilitate expansion in the future. Even if these pseudo-commands are entered there will be no effect on the assembly operation.

.ENDIO

I/O area termination directive (Reserved)

Format

▲.ENDIO

Description

- Directs the termination of an I/O area.
- All labels and symbols entered between .IO and .ENDIO will be deemed to fall within the I/O area.

Example

```
            .IO
port0       .EQU   00H
port1       .EQU   01H
            .ENDIO
```

.ENDPROC

Program module termination directive (Reserved)

Format

▲.ENDPROC

Description

- Directs the termination of all modules of a main program, subprogram or interrupt program.
- The area enclosed by either .PROCINT, .PROCMAIN or .PROCSUB and .ENDPROC is deemed to be a single program module.

Example

```
            .PROCMAIN
MAIN:
            :
            JMP   MAIN
            .ENDPROC
```

.ENDRAM — RAM area termination directive (Reserved)

Format

▲.ENDRAM

Description

- Directs the termination of the RAM area.
- Any labels and symbols entered between .RAM and .ENDRAM are deemed to belong to the RAM area.

Example

```
            .RAM
work0:      .BLKB   1
work1:      .BLKB   1
            .ENDRAM
```

.IO — I/O area start directive (Reserved)

Format

▲.IO

Description

- Directs the start of an I/O area.
- Any labels or symbols entered between .IO and .ENDIO are deemed to belong to the I/O area.

Example

```
            .IO
port0       .EQU    00H
port1       .EQU    01H
            .ENDIO
```

.PROCINT — Interrupt program start directive (Reserved)

Format

▲.PROCINT △[label]

Description

- Directs the start of an interrupt program.
- The area between .PROCINT and .ENDPROC is deemed to be an interrupt program.
- SRA74 will process any label entered in the operand as the label for this line.

Example

```
.PROCINT INT
 :
 :
.ENDPROC
```

.PROCMAIN — Main program start directive (Reserved)

Format

▲.PROCMAIN △[label]

Description

- Directs the start of the main program.
- The area between .PROCMAIN and .ENDPROC is deemed to be the main program.
- SRA74 will process any label entered in the operand as the label for this line.

Example

```
.PROCMAIN MAIN
 :
.ENDPROC
```

.PROCSUB — Subprogram start directive (Reserved)

Format

▲.PROCSUB △[label]

Description

- Directs the start of a subprogram.
- The area between .PROCSUB and .ENDPROC is deemed to be a subprogram.
- SRA74 will process any label entered in the operand as the label for this line.

Example

```
.PROCSUB SUB
 :
 :
.ENDPROC
```

.PROGNAME — Program name directive (Reserved)

Format

▲.PROGNAME △program name

Description

- Directs the naming of a program.
- The contents of the operand will be deemed to be the title of the program.

Example

```
.PROGNAME printer control program
```

.RAM

RAM area start directive (Reserved)

Format

▲.RAM

Description

- Directs the start of the RAM area.
- Any labels and symbols entered between .RAM and .ENDRAM are deemed to be part of the RAM area.

Example

```
            .RAM
work0:      .BLKB   1
work1:      .BLKB   1
            .ENDRAM
```

APPENDIX E
List of Macro Commands

E.1 How to use the macro command reference section

All the macro commands which can be used in SRA74 are described in alphbetical order. Points which should be noted with regard to the notation style in this section are outlined below:-

1. The use of square brackets [ ] indicates that the section may be omitted as required.
2. △, ▲ are used to indicate the use of a space or a tab code. The △ represents a required section whereas the ▲ section may be omitted if necessary.
3. n the following descriptions the space between a label and a macro command is indicated by a ▲. When entering a label a ":" (colon) is not absolutely necessary but if it is omitted then a space or tab code must be inserted between the label and a macro command.

E.2 Summary of the macro commands

.ENDM — ENDM macro command

Format

▲.ENDM

Description

- This command indicates the end of all macro definitions.

Example

[Macro definition]

```
ADD:  .MACRO    VAL
      CLC
      ADC       VAL
      .ENDM
```

[Call example]

```
      ADC       #10
```

[Macro generation]

```
      CLC
      ADC       #10
      .ENDM
```

.EXITM

EXITM macro command

Format

▲.EXITM

Description

- This command suspends macro generation and transfers control to the nearest .ENDM.

Example

[Macro definition]

```
DATA1:  .MACRO   VAL
        .IF     LABEL
        .BYTE   VAL
        .EXITM
        .ENDIF
        .WORD   VAL
        .ENDM
```

[Call example]

```
LABEL   .EQU    1
        DATA1   10
```

[Macro generation]

```
        .IF     LABEL
        .BYTE   10
        .EXITM
        .ENDIF
        .ENDM
```

.LOCAL — LOCAL macro command

Format

▲.LOCAL △label,[label,..,label]

Description

- This command makes a label which is defined among the macro definitions into an intra-macro local label.
- Labels which are made LOCAL are assembled in order by allocating them the label ..n (where n is a decimal number between ..0 - ..65535). Care should be taken by the user since use of the ..n label is not permitted.

Example

[Macro definition]

```
LOOP:   .MACRO
        .LOCAL   LOOP1
        LDA      #20
LOOP1:  DEC      A
        BNE      LOOP1
        .ENDM
```

[Call example]

```
        LOOP
```

[Macro generation]

```
        LDA      #20
..0:    DEC      A
        BNE      ..0
        .ENDM
```

.MACRO - .ENDM

MACRO macro command

Format

▲[label:] ▲.MACRO △[argument1,argument2,..,argumentn]

Description

- A macro definition starts from the line entered as .MACRO and ends on the line in which .ENDM is entered.
- The label allotted to the .MACRO command line becomes the name of this macro definition.
- A macro definition may or may not have arguments attached but where a macro with arguments is used then the necessary arguments must be transferred by the macro call statement.
- When a macro call is made and the arguments are generated then they will be transferred in the order in which the dummy arguments have been entered into the macro definition.
- Either assembly language commands, structured language commands, system macro commands or pseudo-commands can be entered between .MACRO and .ENDM. However, macro definition must not be nested for more than 20 levels.
- Providing a macro call follows a macro definition then it may be located anywhere in the program. If, however, a macro definition appears only after the macro call then this will be registered as an error.
- The arguments which are specified in the operand by the macro call are substituted in order from the left for the corresponding dummy arguments in the macro definition. It does not matter if the number of actual arguments and the number of dummy arguments entered at the original definition stage do not agree but if the number of actual arguments exceeds the number of dummy arguments then the surplus is ignored and if the number falls short then the shortfall is treated as null characters (a character string with zero length).
- Any number of actual arguments may be specified regardless of the number of dummy arguments used in the macro definition but they must all be fitted within the compass of a single line. Since arguments are divided by a ‘,’ it is necessary when using a comma or a space as an argument to enclose it within ‘“‘. However, commas inside parentheses are not treated as argument separators.
- A macro generated area is marked in the PRN file by a “+”.

* Multiple macro definitions may be made using the same macro name. When such macro is called, the most recent definition is used.

Example

Example 1: Macro definition without operand

[Macro definition]

```
ADDa:   .MACRO
        LDA     ABC
        LDX     #DEF
        CLC
        ADC     TABLE,X
        STA     GHI
        .ENDM
```

[Call example]

```
        ADDa
```

[Macro generation]

```
        LDA     ABC
        LDX     #DEF
        CLC
        ADC     TABLE,X
        STA     GHI
        .ENDM
```

Example 2: Macro definition with operand

[Macro definition]

```
ADDb:   .MACRO  V1,IMM,V2; Dummy arguments (V1,IMM,V2)
        LDA     V1
        LDX     #IMM
        CLC
        ADC     TABLE,X
        STA     V2
        .ENDM
```

[Call example]

```
        ADDb    WORK1,10,WORK2
```

[Macro generation]

```
        LDA     WORK1
        LDX     #10
        CLC
        ADC     TABLE,X
        STA     WORK2
        .ENDM
```

Example 3: Macro nesting

[Macro definition]

```
ADD:  .MACRO     SRC
      CLC
      ADC        SRC
      .ENDM
```

[Call example]

```
ADDW: .MACRO     SRC
      ADD        SRC   ; Macro call within a macro
      ADC        SRC+1
      .ENDM
```

Examplee 4: Recursive macro call

[Macro defnition]

```
MAC:  .MACRO               ; First definition
DATA: .BLKB      1
MAC:  .MACRO     VALUE     ; Second definition
      LDM        #VALUE,DATA
      .ENDM
      .ENDM
```

[Call example]

```
      MAC              ; Area reservation and new macro definition
       :
      MAC  10H         ; Newly defined macro call
```

.REPEAT - .ENDM

REPEAT macro command

Format

▲[label:] ▲.REPEAT △number of times

Description

- A given 740 Family command is repeatedly assembled between .REPEAT and .ENDM in accordance with the number of times specified by the operand.
- The label given to the .REPEAT command line becomes the first label of the generated line.
- Assembly language commands, structured language commands, macro commands, and pseudo commands other than .INCLUDE may be coded between .REPEAT and .ENDM. However, macros must not be nested for more than 20 levels.
- Numeric and symbolic constants (labels) may be entered in the operand but labels with relocatable values may not.

Example

[Source entry example]

```
TIME5:  .REPEAT 5
        NOP
        .ENDM
```

[After macro generation]

```
TIME5:  .REPEAT 5
        NOP
        .ENDM
TIME5:
        NOP
        NOP
        NOP
        NOP
        NOP
```

.REPEATC - .ENDM

REPEATC macro command

Format

▲[label:] ▲.REPEATC △dummy argument,real argument

Description

- Characters are repeatedly assembled by the operand in accordance with the number of times they are given by the real argument until .ENDM is reached.
- One character is withdrawn from the real argument and transferred to the dummy argument each time the operation is repeated.
- The label given to the .REPEATC command line becomes the first label of the generated line.
- Assembly language commands, structured language commands, macro commands, and pseudo commands other than .INCLUDE may be coded between .REPEATC and .ENDM. However, macros must not be nested for more than 20 levels.
- If special characters such as a space, tab or comma (,) are included in a character string then the whole string must be specified within “”. In such cases the whole of the character string is used with the exception of the “”.

Example

Example 1:

[Source entry example]

```
DATA:   .REPEATC VAL,ABCDE
                 ↑    ↑
        Dummy argument  Real argument
            .BYTE    'VAL'
            .ENDM
```

[After macro generation]

```
DATA:
            .BYTE   'A'
            .BYTE   'B'
            .BYTE   'C'
            .BYTE   'D'
            .BYTE   'E'
```

Example 2:

[Source entry example]

```
DATA:   .REPEATC VAL,"ABC,;"
                 ↑    ↑
          Dummy argument  Real argument
            .BYTE   'VAL'
            .ENDM
```

[After macro generation]

```
DATA:
            .BYTE   'A'
            .BYTE   'B'
            .BYTE   'C'
            .BYTE   ','
            .BYTE   ';'
```

.REPEATI - .ENDM

REPEATI macro command

Format

▲[label:] ▲.REPEATI △dummy argument,real argument[,real argument,..,real argument]

Description

- Assembly is repeated until .ENDM is reached in accordance with the number of arguments entered in the operand.
- One argument is withdrawn from the arguments entered in the operand and transferred to the dummy argument each time the operation is repeated.
- The label given to the .REPEATI command line becomes the first label of the generated line.
- Assembly language commands, structured language commands, macro commands, and pseudo commands other than .INCLUDE may be coded between .REPEATC and .ENDM. However, macros must not be nested for more than 20 levels.
- Numerical constants, character constants, symbolic constants (labels) and character string constants may be entered as real arguments. Other macro commands may not be entered.
- If spaces, tabs or commas (,) are included in a real argument then the whole argument must be enclosed within ‘ “ ’.

Example

Example 1:

[Source entry example]

```
SUB:    .REPEATI INST,"NOP","LDA #1","JSR SUB1","RTS"
                 ↑          ↑
```

Dummy argument Real argument

```
        INST
        .ENDM
```

[After macro generation]

```
SUB:
        NOP
        LDA #1
        JSR SUB1
        RTS
        .ENDM
```

Example 2:

[Source entry example]

```
DATA:   .REPEATI VAL,0,1,2,"'HELLO !!'"
                 ↑     ↑
```

Dummy argument Real argument

```
        .BYTE    VAL
        .ENDM
```

[After macro generation]

```
DATA:
        .BYTE   0
        .BYTE   1
        .BYTE   2
        .BYTE   'HELLO !!'
        .ENDM
```

APPENDIX F
List of Structured Commands

F.1 Using the structured command reference section

All the structured commands which can be used in SRA74 are described in alphbetical order. Points which should be noted with regard to the notation style in this section are outlined below:-

1. △, ▲ are used to indicate the use of a space or a tab code.
   The △ represents a required section whereas the ▲ section may be omitted if necessary.

2. The items in brackets [ ] may be omitted.

3. When entering a label a ":" (colon) is not absolutely necessary but if it is omitted then a space or tab code must be inserted between the label and a structured command. As a general rule you are recommended to use a ":" at all times.

F.2 Summary of the structured commands

BREAK

BREAK statement

Format

▲[label:] ▲(L)BREAK

Description

- A “break” statement suspends the execution of the corresponding “for”, “do” or “switch” statement and passes control to the next statement to be executed.
- The “break” statement may only be used within a “for”, “do” or “switch” statement.

CONTINUE

CONTINUE statement

Format

▲[label:] ▲(L)CONTINUE

Description

- The “continue” statement uses a dummy statement to replace the final statement of the smallest repeat statement, “for” or “do”, which contains it and passes control to that dummy statement.
- It may only be used within a “for” or “do” statement.

DO - WHILE

DO statement

Format

▲[label:] ▲(L)DO
<statement>
▲[label:] ▲WHILE △conditional expression

Description

- The “do” statement continues to be executed so long as the conditional expression is fulfilled (so long as it remains true). The decision to repeat is made only after execution of the statement. For this reason the “do” statement is most useful when you wish to repeat an operation once more before terminating the repetition.

- If “ever” is inserted into a conditional expression it will form an endless loop.

- For further details concerning conditional expressions please refer to the syntax maps.

FOR - NEXT

FOR statement

Format

▲[label:] ▲(L)FOR △conditional expression
<statement>
▲[label:] ▲NEXT

Description

- The “for” statement is a command used for the control of the repetition of an operation which it will continue to repeat for so long as the specified conditional expression holds true.

- If “ever” is inserted into a conditional expression it will form an endless loop.

- For further details concerning conditional expressions please refer to the syntax maps.

IF - (ELSE) - ENDIF

IF statement

Format

▲[label:] ▲(L)IF △conditional expression
<statement>
▲[[label:] ▲(L)ELSE]
<statement>
▲[label:] ▲ENDIF

Description

- The “if” statement is a command which switches the control stream in one of two different directions where the selected direction for branching is determined in accordance with a conditional expression. The branch decision itself is based on whether the result of applying the condition is zero (false) or not (true). If the condition holds true then control is passed to the subsequent command but if it is found to be false then control is passed to the command immediately following “else” where there is an “else” and to the command following “endif” if there is no “else”.

- The “else” section can be omitted.

- There is no limit to the number of times the “if” statement can be nested.

- Where the “if” statement is nested in a complex manner then the relationship between “if” and “else” is determined such that the “if” and “else” pair which is closest together forms a sequential pair.

- For further details concerning conditional expressions please refer to the syntax maps.

SWITCH - CASE - ENDS

SWITCH statement

Format

▲[label:] ▲(L)SWITCH △conditional expression
▲[label:] ▲CASE △constant
<statement>
▲[label:] ▲CASE △constant
<statement>
:
▲[label:] ▲DEFAULT
<statement>
▲[label:] ▲ENDS

Description

- The “switch” statement passes control to another statement which is determined in accordance with the value of a conditional expression. The value of the expression is compared with a “case” prefix constant attached to the beginning of the statement and where there is a match then control is passed to the statement. Where there is no match between the value of the expression and the value specified by the “case” then control is passed to the statement which has a “default” prefix. Where there is no “default” none of the statements is executed and the “switch” statement is left.

- The “default” section may be omitted.

- When control has been passed to a matching “case” statement then all subsequent “case” statements will be executed in turn. If you do not wish to pass on to the next “case” statement but instead to leave the “switch” statement then this can be effected by the use of a “break” statement.

- For further details of conditional expressions and constants please refer to the syntax maps.

ASSIGNMENT

ASSIGNMENT statement

Format

▲[label:] ▲left side ▲=▲right side

Description

- Assigns the right side to the left. For assignment to memory bit variables, register bit variables or flag variables only numerical or symbolic constants with values of 0 or 1 can be used.

- For further details concerning left and right sides please refer to the syntax maps.

F.3 Generation example

The assembly language statements generated from structured language commands are described with examples.

F.3.1 Assignment statement generation example

The relationship between assembly language and structured language commands other than conditional branch commands are shown below. Conditional branch commands are described in F.3.2. The following table describes the symbols used in the assignment statement generation example.

Table F.1 Symbols used in examples

Symbol	Description	Symbol	Description
A	Accumulator	X	Index register X
Y	Index register Y	S	Stack pointer
P	Processor status register	C	Carry flag
Z	Zero flag	I	Interrupt disable flag
D	Decimal mode flag	T	X modifier operation mode flag
V	Overflow flag	N	Negative flag
imm	Immediate data	zz	Zero page address
hhll	General page address	#	Immediate mode
MEM	Memory	FLAG	Memory bit

A) Register and flag assignment statement generation examples

Table F.2 Register and flag assignment statement generation examples

Type	Structured command	Assembly command
Bit Operation	BIT_A0 = 0	CLB BIT_A0
	BIT_A0 = 1	SEB BIT_A0
Flags	C = 0	CLC
	C = 1	SEC
	D = 0	CLD
	D = 1	SED
	I = 0	CLI
	I = 1	SEI
	T = 0	CLT
	T = 1	SET
	V = 0	CLV

Appendix F. List of structured commands

Type	Structured command	Assembly command
Data transfer commands	A = imm	LDA # imm
	X = imm	LDX # imm
	Y = imm	LDY # imm
	X = A	TAX
	A = X	TXA
	Y = A	TAY
	A = Y	TYA
	X = S	TSX
	S = X	TXS
	[S] = A	PHA
	[S] = P	PHP
	A = [S]	PLA
	P = [S]	PLP
Arithmetic commands	A = A + imm WITH_C	ADC # imm
	A = A - imm WITH_C	SBC # imm
	A = ++ A	INC A
	A = -- A	DEC A
	X = ++ X	INX
	X = -- X	DEX
	Y = ++ Y	INY
	Y = -- Y	DEY
	A = A & imm	AND # imm
	A = A | imm	ORA # imm
	A = A ^ imm	EOR # imm
	A = A << imm	ASL A
	A = A >> imm	LSR A
	A = A << imm WITH_C	ROL A
	A = A >> imm WITH_C	ROR A

B) Memory assignment statement generation examples

Table F.3 Memory assignment statement generation examples

Type	Structured command	Assembly command
Data transfer commands	A = [zz]	LDA zz
	X = [zz]	LDX zz
	Y = [zz]	LDY zz
	[zz] = imm	LDM # imm, zz
	[zz] = A	STA zz
	[zz] = X	STX zz
	[zz] = Y	STY zz
Arithmetic commands	A = A + [zz] WITH_C	ADC zz
	A = A - [zz] WITH_C	SBC zz
	[zz] = ++ [zz]	INC zz
	[zz] = -- [zz]	DEC zz
	A = A & [zz]	AND zz
	A = A | [zz]	ORA zz
	A = A ^ [zz]	EOR zz
	[zz] = [zz] << imm	ASL zz
	[zz] = [zz] >> imm	LSR zz
	[zz] = [zz] << imm WITH_C	ROL zz
	[zz] = [zz] >> imm WITH_C	ROR zz
Bit operation	[FLAG] = 0	CLB FLAG
	[FLAG] = 1	SEB FLAG

C) Addressing mode assignment statement generation examples

Table F.4 Addressing mode assignment statement generation examples

Type	Structured command	Assembly command
Load commands	A = [zz,X]	LDA zz,X
	A = [hhll]	LDA hhll
	A = [hhll,X]	LDA hhll,X
	A = [hhll,Y]	LDA hhll,Y
	A = [(zz,X)]	LDA (zz,X)

Appendix F. List of structured commands

Type	Structured command	Assembly command
Load commands	A = [(zz),Y]	LDA (zz),Y
	X = [zz,Y]	LDX zz,Y
	X = [hhll]	LDX hhll
	X = [hhll,Y]	LDX hhll,Y
	Y = [zz,X]	LDY zz,X
	Y = [hhll]	LDY hhll
	Y = [hhll,X]	LDY hhll,X
Store commands	[zz,X] = A	STA zz,X
	[hhll] = A	STA hhll
	[hhll,X] = A	STA hhll,X
	[hhll,Y] = A	STA hhll,Y
	[(zz,X)] = A	STA (zz,X)
	[(zz),Y] = A	STA (zz),Y
	[zz,Y] = X	STX zz,Y
	[hhll] = X	STX hhll
	[zz,X] = Y	STY zz,X
	[hhll] = Y	STY hhll
Add subtract commands	A = A + [zz,X] WITH_C	ADC zz,X
	A = A + [hhll] WITH_C	ADC hhll
	A = A + [hhll,X] WITH_C	ADC hhll,X
	A = A + [hhll,Y] WITH_C	ADC hhll,Y
	A = A + [(zz,X)] WITH_C	ADC (zz,X)
	A = A + [(zz),Y] WITH_C	ADC (zz),Y
	A = A - [zz,X] WITH_C	SBC zz,X
	A = A - [hhll] WITH_C	SBC hhll
	A = A - [hhll,X] WITH_C	SBC hhll,X
	A = A - [hhll,Y] WITH_C	SBC hhll,Y
	A = A - [(zz,X)] WITH_C	SBC (zz,X)
	A = A - [(zz),Y] WITH_C	SBC (zz),Y
	[zz,X] = ++ [zz,X]	INC zz,X
	[hhll] = ++ [hhll]	INC hhll
	[hhll,X] = ++ [hhll,X]	INC hhll,X
	[zz,X] = -- [zz,X]	DEC zz,X
	[hhll] = -- [hhll]	DEC hhll
	[hhll,Y] = -- [hhll,Y]	DEC hhll,X

Type	Structured command	Assembly command
	A = A & [zz,X]	AND zz,X
	A = A & [hhll]	AND hhll
	A = A & [hhll,X]	AND hhll,X
Logical	A = A & [hhll,Y]	AND hhll,Y
	A = A & [(zz,X)]	AND (zz,X)
operation	A = A & [(zz),Y]	AND (zz),Y
	A = A | [zz,X]	ORA zz,X
commands	A = A | [hhll]	ORA hhll
	A = A | [hhll,X]	ORA hhll,X
	A = A | [hhll,Y]	ORA hhll,Y
	A = A | [(zz,X)]	ORA (zz,X)
	A = A | [(zz),Y]	ORA (zz),Y
	A = A ^ [zz,X]	EOR zz,X
Logical	A = A ^ [hhll]	EOR hhll
	A = A ^ [hhll,X]	EOR hhll,X
operation	A = A ^ [hhll,Y]	EOR hhll,Y
	A = A ^ [(zz,X)]	EOR (zz,X)
commands	A = A ^ [(zz),Y]	EOR (zz),Y
	[zz,X] = [zz,X] << imm	ASL zz,X
	[hhll] = [hhll] << imm	ASL hhll
Rotate	[hhll,X] = [hhll,X] << imm	ASL hhll,X
	[zz,X] = [zz,X] >> imm	LSR zz,X
shift	[hhll] = [hhll] >> imm	LSR hhll
	[hhll,X] = [hhll,X] >> imm	LSR hhll,X
	[zz,X] = [zz,X] << imm WITH_C	ROL zz,X
commands	[hhll] = [hhll] << imm WITH_C	ROL hhll
	[hhll,X] = [hhll,X] << imm WITH_C	ROL hhll,X
	[zz,X] = [zz,X] >> imm WITH_C	ROR zz,X
	[hhll] = [hhll] >> imm WITH_C	ROR hhll
	[hhll,X] = [hhll,X] >> imm WITH_C	ROR hhll,X

* Commands with no corresponding structured language command.

MUL, DIV, COM, BIT, TST, RRF, JSR, RTI, RTS, NOP, FST, SLW, WIT, STP, BRA, JMP

D) Dual term operation assignment statement generation examples

Table F.5 Dual term operation assignment statement generation examples

Type	Structured command	Assembly command
Rotate shift commands	[zz] = [zz] << 2	ASL zz ASL zz
	[zz] = [zz] << 2 WITH_C	ROL zz ROL zz
	[zz] = [zz] >> 2	LSR zz LSR zz
	[zz] = [zz] >> 2 WITH_C	ROR zz ROR zz
Add subtract commands	[zz] = [zz] + 4	LDA zz CLC ADC # 4 STA zz
	[zz] = [zz] + 4 WITH_C	LDA zz ADC # 4 STA zz
	[zz] = [zz] + [zz]	LDA zz CLC ADC zz STA zz
	[zz] = [zz] - 4	LDA zz SEC SBC # 4 STA zz
	[zz] = [zz] - 4 WITH_C	LDA zz SBC # 4 STA zz

Type	Structured command	Assembly command
Subtract commands	[zz] = [zz] - [zz]	LDA zz SEC SBC zz STA zz
Multiply	[zz] = [zz] * 4	LDA zz JSR .mult_8 .BYTE 4 STA zz
divide commands	[zz] = [zz] / 4	LDA zz JSR .div_8 .BYTE 4 STA zz
	[zz] = [zz] % 4	LDA zz JSR .mod_8 .BYTE 4 STA zz
Logical operation commands	[zz] = [zz] & 4	LDA zz AND # 4 STA zz
	[zz] = [zz] | 4	LDA zz ORA # 4 STA zz
	[zz] = [zz] ^ 4	LDA zz EOR # 4 STA zz

* Refer to Note 2 in section 5.4 for .mult_8, .div_8, and .mod_8.

F.3.2 Conditional expression generation examples

The conditional expression generation examples for IF, DO, and FOR statements are shown below.

The labels corespond to the following generation example.

```
- if (else) ~ endif

        if   [ MEM ] == 2       ;       LDA    MEM
             JSR   out          ;       CMP    #2
        endif                   ;       BNE    hhll
                                ;       JSR    out
                                ; hhll:

- do ~ while

        do                      ; hhll:
             JSR   out          ;       JSR    out
        while   [ MEM ] == 2    ;       LDA    MEM
                                ;       CMP    #2
                                ;       BEQ    hhll
- for ~ next

        for   [ MEM ] == 2      ; hhll1:
             JSR   out          ;       LDA    MEM
        next                    ;       CMP    #2
                                ;       BNE    hhll2
                                ;       JSR    out
                                ;       BRA    hhll1
                                ; hhll2:
```

Table F.6 Flag conditional expression generation example

Structured command	Assembly command	Long branch
• if [FLAG]== 1 • while [FLAG]== 0 • for [FLAG]== 1	BBC FLAG,hhll	BBS FLAG, .Z0 JMP hhll .Z0:
• if [FLAG] == 0 • while [FLAG]== 1 • for [FLAG]== 0	BBS FLAG,hhll	BBC FLAG, .Z0 JMP hhll .Z0:
• if C == 1 • while C == 0 • for C == 1	BCC hhll	BCS .Z0 JMP hhll .Z0:
• if C == 0 • while C == 1 • for C == 0	BCS hhll	BCC .Z0 JMP hhll .Z0:
• if Z == 1 • while Z == 0 • for Z == 1	BNE hhll	BEQ .Z0 JMP hhll .Z0:
• if Z == 0 • while Z == 1 • for Z == 0	BEQ hhll	BNE .Z0 JMP hhll .Z0:
• if N == 1 • while N == 0 • for N == 1	BPL hhll	BMI .Z0 JMP hhll .Z0:
• if N == 0 • while N == 1 • for N == 0	BMI hhll	BPL .Z0 JMP hhll .Z0:
• if V == 1 • while V == 0 • for V == 1	BVC hhll	BVS .Z0 JMP hhll .Z0:
• if V == 0 • while V == 1 • for V == 0	BVS hhll	BVC .Z0 JMP hhll .Z0:

Table F.7 Memory conditional expression generation example

Type	Structured command	Assembly command	Long branch
IF statement	if [MEM] == 2	LDA MEM CMP # 2 BNE hhll	LDA MEM CMP # 2 BEQ .Z0 JMP hhll .Z0:
	if [MEM] != 2	LDA MEM CMP # 2 BEQ hhll	LDA MEM CMP # 2 BNE .Z0 JMP hhll .Z0:
	if [MEM] > 2	LDA MEM CMP # 2 BEQ hhll BCC hhll	LDA MEM CMP # 2 BEQ .Z0 BCS .Z1 .Z0: JMP hhll .Z1
	if [MEM] >= 2	LDA MEM CMP # 2 BCC hhll	LDA MEM CMP # 2 BCS .Z0 JMP hhll .Z0:
	if [MEM] < 2	LDA MEM CMP # 2 BCS hhll	LDA MEM CMP # 2 BCC .Z0 JMP hhll .Z0:
	if [MEM] <= 2	LDA MEM CMP # 2 BEQ .Z0 BCS hhll .Z0:	LDA MEM CMP # 2 BEQ .Z0 BCC .Z0 JMP hhll .Z0:

Type	Structured command	Assembly command	Long branch
DO statement	while [MEM]== 2	LDA MEM CMP # 2 BEQ hhll	LDA MEM CMP # 2 BNE .Z0 JMP hhll .Z0:
	while [MEM]!= 2	LDA MEM CMP # 2 BNE hhll	LDA MEM CMP # 2 BEQ .Z0 JMP hhll .Z0:
	while [MEM]> 2	LDA MEM CMP # 2 BEQ .Z0 BCS hhll .Z0:	LDA MEM CMP # 2 BEQ .Z0 BCC .Z0 JMP hhll .Z0:
	while [MEM]>= 2	LDA MEM CMP # 2 BCS hhll	LDA MEM CMP # 2 BCC .Z0 JMP hhll .Z0:
	while [MEM]< 2	LDA MEM CMP # 2 BCC hhll	LDA MEM CMP # 2 BCS .Z0 JMP hhll .Z0:
	while [MEM]<= 2	LDA MEM CMP # 2 BEQ hhll BCC hhll	LDA MEM CMP # 2 BEQ .Z0 BCS .Z1 .Z0: JMP hhll .Z1:

Type	Structured command	Assembly command	Long branch
FOR statement	for [MEM] == 2	LDA MEM CMP # 2 BNE hhll2	LDA MEM CMP # 2 BEQ .Z0 JMP hhll2 .Z0:
	for [MEM] != 2	LDA MEM CMP # 2 BEQ hhll2	LDA MEM CMP # 2 BNE .Z0 JMP hhll2 .Z0:
	for [MEM] > 2	LDA MEM CMP # 2 BEQ hhll2 BCC hhll2	LDA MEM CMP # 2 BEQ .Z0 BCS .Z1 .Z0: JMP hhll2 .Z1:
	for [MEM] >= 2	LDA MEM CMP # 2 BCC hhll2	LDA MEM CMP # 2 BCS .Z0 JMP hhll2 .Z0:
	for [MEM] < 2	LDA MEM CMP # 2 BCS hhll2	LDA MEM CMP # 2 BCC .Z0 JMP hhll2 .Z0:
	for [MEM] <= 2	LDA MEM CMP # 2 BEQ .Z0 BCS hhll2 .Z0:	LDA MEM CMP # 2 BEQ .Z0 BCC .Z1 .Z0: JMP hhll2 .Z1:

Table F.8 Register conditional expression generation example

Type	Structured command	Assembly command	Long branch
A register	if A == [MEM]	CMP MEM BNE hhll	CMP MEM BEQ .Z0 JMP hhll .Z0:
	if A != [MEM]	CMP MEM BEQ hhll	CMP MEM BNE .Z0 JMP hhll .Z0:
	if A > [MEM]	CMP MEM BEQ hhll BCC hhll	CMP MEM BEQ .Z0 BCS .Z1 .Z0: JMP hhll .Z1:
	if A >= [MEM]	CMP MEM BCC hhll	CMP MEM BCS .Z0 JMP hhll .Z0:
	if A < [MEM]	CMP MEM BCS hhll	CMP MEM BCC .Z0 JMP hhll .Z0:
	if A <= [MEM]	CMP MEM BEQ .Z0 BCS hhll .Z0:	CMP MEM BEQ .Z0 BCC .Z0 JMP hhll .Z0:
X register	if X == [MEM]	CPX MEM BNE hhll	CPX MEM BEQ .Z0 JMP hhll .Z0:
	if X != [MEM]	CPX MEM BEQ hhll	CPX MEM BNE .Z0 JMP hhll .Z0:
	if X > [MEM]	CPX MEM BEQ hhll BCC hhll	CPX MEM BEQ .Z0 BCS .Z1 .Z0: JMP hhll .Z1:

Appendix F. List of structured commands

Type	Structured command	Assembly command	Long branch
X regisrer	if X >= [MEM]	CPX MEM BCC hhll	CPX MEM BCS .Z0 JMP hhll .Z0:
	if X < [MEM]	CPX MEM BCS hhll	CPX MEM BCC .Z0 JMP hhll .Z0:
	if X <= [MEM]	CPX MEM BEQ .Z0 BCS hhll .Z0:	CPX MEM BEQ .Z0 BCC .Z0 JMP hhll .Z0:
Y register	if Y == [MEM]	CPY MEM BNE hhll	CPY MEM BEQ .Z0 JMP hhll .Z0:
	if Y != [MEM]	CPY MEM BEQ hhll	CPY MEM BNE .Z0 JMP hhll .Z0:
	if Y > [MEM]	CPY MEM BEQ hhll BCC hhll	CPY MEM BEQ .Z0 BCS .Z1 .Z0: JMP hhll .Z1:
	if Y >= [MEM]	CPY MEM BCC hhll	CPY MEM BCS .Z0 JMP hhll .Z0:
	if Y < [MEM]	CPY MEM BCS hhll	CPY MEM BCC .Z0 JMP hhll .Z0:
	if Y <= [MEM]	CPY MEM BEQ .Z0 BCS hhll .Z0:	CPY MEM BEQ .Z0 BCC .Z0 JMP hhll .Z0:

F.4 Syntax maps of structured commands

Syntax maps are used to illustrate the grammar of the structured commands which can be used by SRA74.
The meanings of the most important words are illustrated below:

- Variable
  Variable is used as a general term covering memory variables, memory bit variables, register variables, register bit variables and flag variables.

- Memory variable
  This refers to any selected memory or stack location which is referred to in any of the addressing modes of the 740 Family. Memory variables should always be enclosed for entry in [ ] or { }.

[ZZ]	Zero page	[ZZ,X]	Zero page X
[ZZ,Y]	Zero page Y	[HHLL]	Absolute
[HHLL,X]	Absolute X	[HHLL,Y]	Absolute Y
[(ZZ,X)]	Zero page indirect X	[(ZZ),Y]	Zero page indirect Y
[S]	Stack		

- Memory bit variable
  This refers to any selected bit which is referred to in any of the bit addressing modes of the 740 Family. Memory bit variables should always be enclosed for entry in [ ] or { }. These variables must always be defined in the following way by the pseudo-command .EQU prior to reference.

  Example:

```
BITSYM    .EQU 1,1000H     Bit symbol definition
          if [ BITSYM ]    Bit symbol reference
                 :
          else
                 :
          endif
```

- Register variable
  This refers to the various registers of the 740 Family. There is no need for enclosure in [] or {} for entry. They should be entered as they are. SRA74 uses the following names as reserved words for the registers. Since no distinction is made between upper and lower case characters either A or a is equally valid.

A	Accumulator	X	Index register X
Y	Index register Y	S	Stack pointer
P	Program counter		

- Register bit variable
  This refers to any selected accumulator bit referred to in the 740 Family accumulator bit addressing mode. SRA74 uses the following names as reserved words for the register bits. Since no distinction is made between upper and lower case characters either BIT_A0 or bit_a0 is equally valid.

BIT_A0	Accumulator bit 0	BIT_A1	Accumulator bit 1
BIT_A2	Accumulator bit 2	BIT_A3	Accumulator bit 3
BIT_A4	Accumulator bit 4	BIT_A5	Accumulator bit 5
BIT_A6	Accumulator bit 6	BIT_A7	Accumulator bit 7

- Flag variable

  This refers to the flags in the 740 Family status registers. SRA74 uses the following names as reserved words for the flags. Since no distinction is made between upper and lower case characters either C or c is equally valid.

C	Carry flag	Z	Zero flag
I	Interrupt inhibit flag	D	Decimal mode flag
T	X qualified operation mode flag	V	Overflow flag
N	Negative flag		

- WITH_C

  This refers to the specification of an operation with carry. This is prepared as a reserved word in SRA74. Since no distinction is made between upper and lower case characters either WITH_C or with_c is equally valid.

  Example:

```
[ work ] = [ work ] << 2 with_c
```

  The contents of work including the carry is shifted 2 places to the left.

- EVER

  This refers to the specification of an endless loop. This is prepared as a reserved word in SRA74. Since no distinction is made between upper and lower case characters either EVER or ever is equally valid.

  Example:

```
for ever
 :
next
```

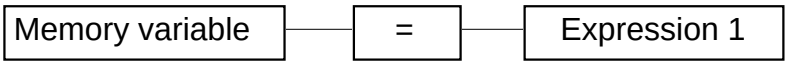
  Specifies an endless loop

- Constant

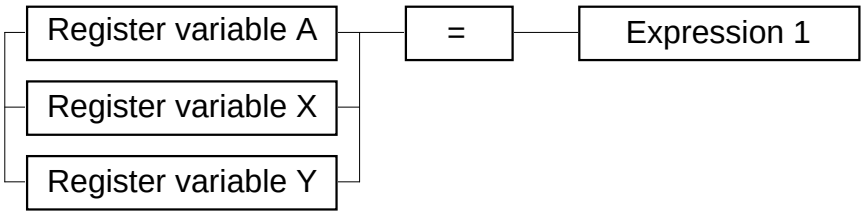
  This is a general term referring to numerical constants, character constants, symbolic constants or combinations of these with operators.

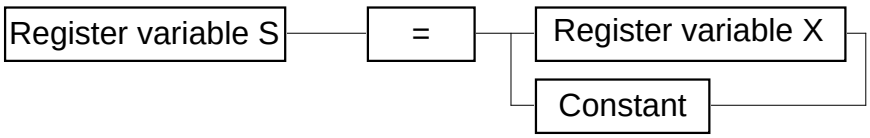
- **Assignment statement**

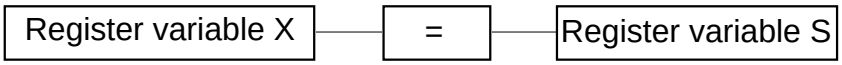
Memory variable assignment statement

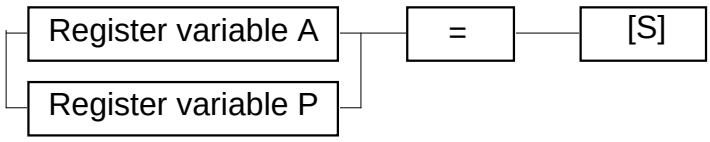


Register variable assignment statement

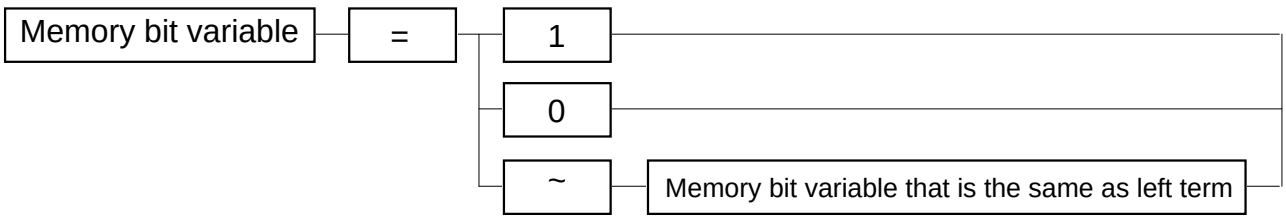




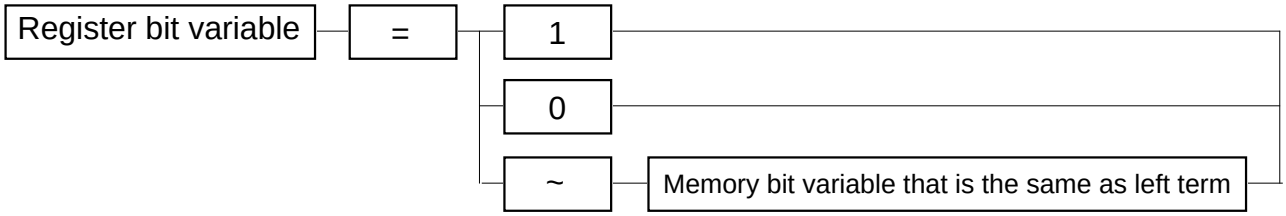




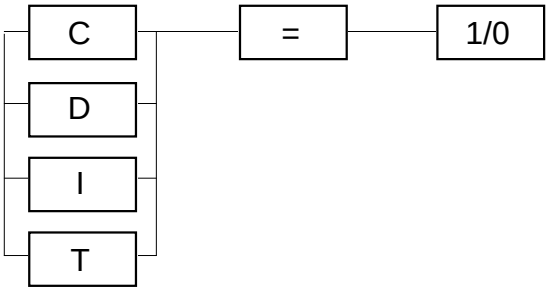
Memory bit variable assignment statement

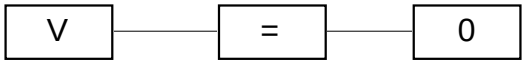


Register bit variable assignment statement

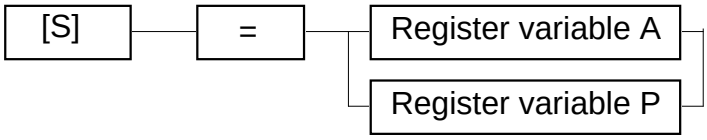


Flag variable assignment statement

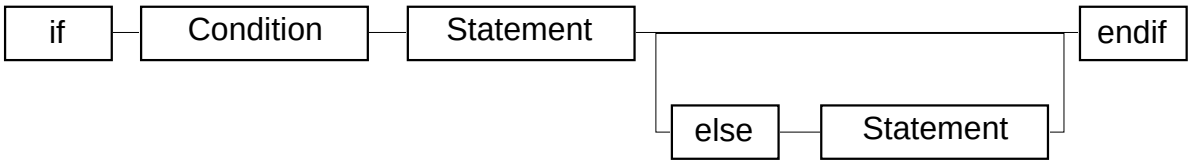




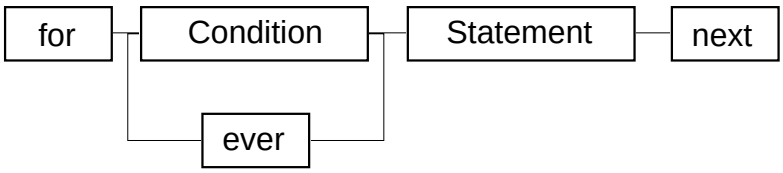
Stack frame variable assignment statement



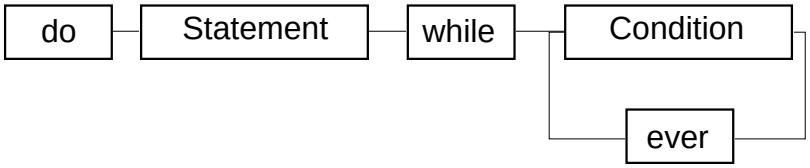
• **if statement**



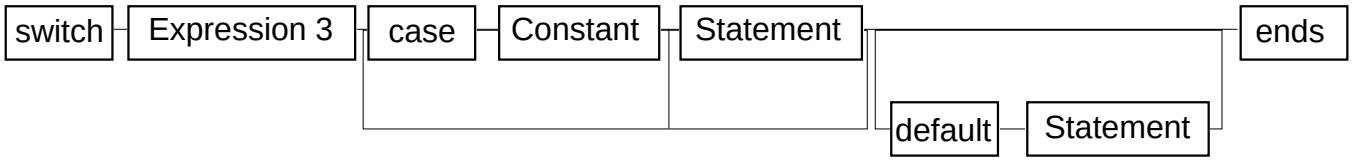
• **for statement**



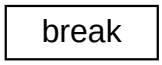
• **do statement**



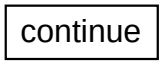
• **switch statement**



• **break statement**



• **continue statement**

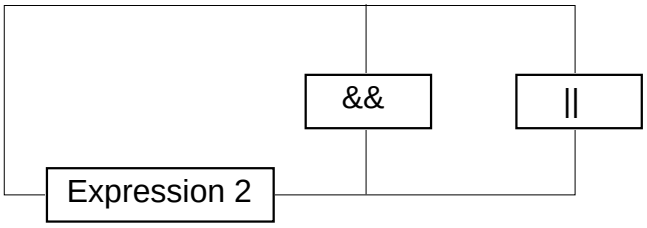


• **Expression 1**

Constant

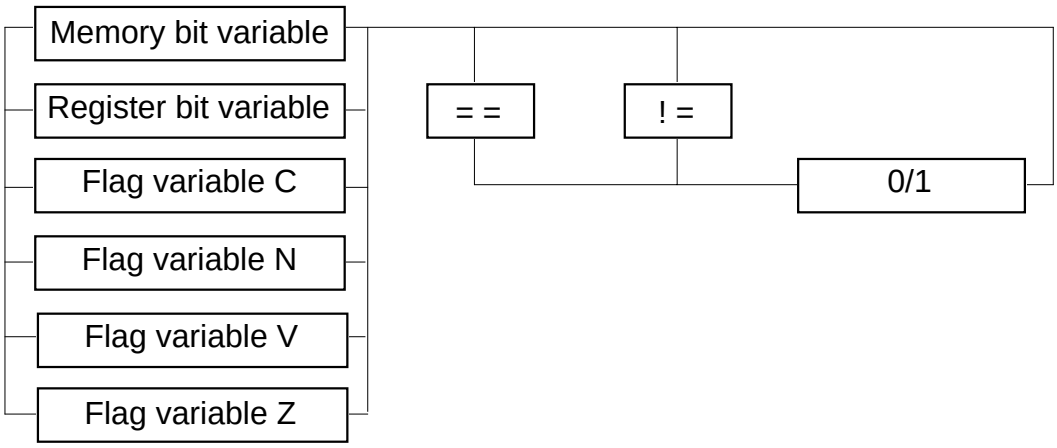
Expression 3

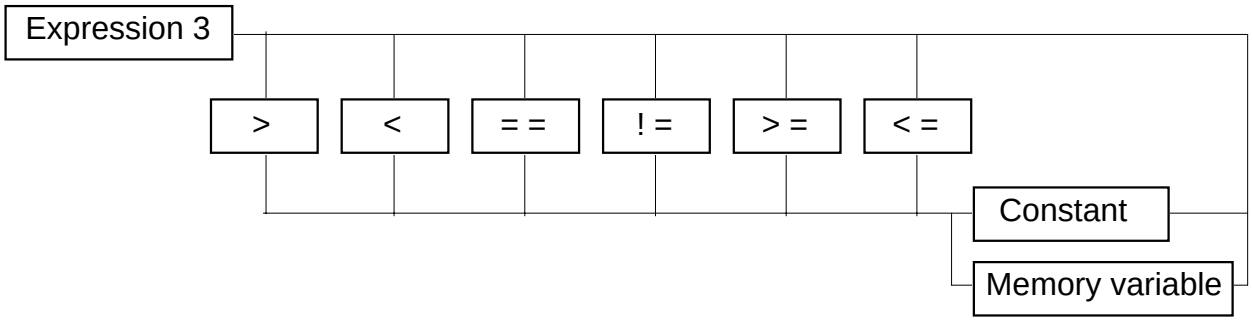
• **Condition expression**



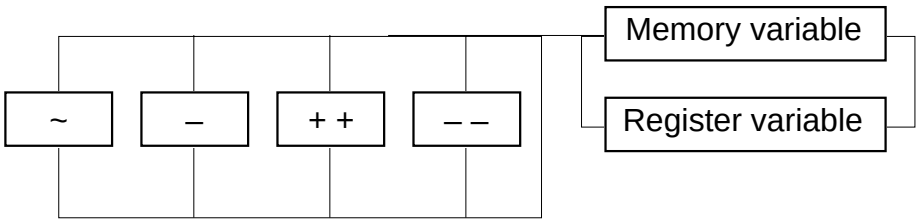
Expression 2 can be concatenated up to six times

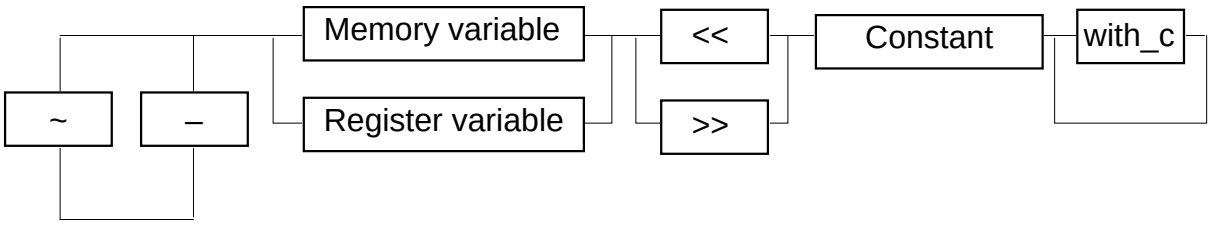
• **Expression 2**

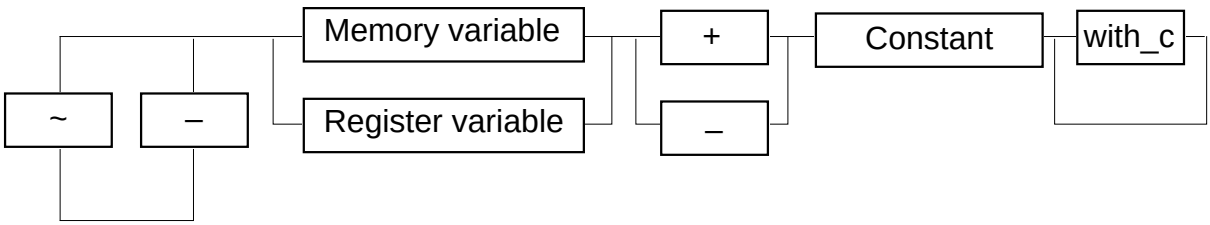


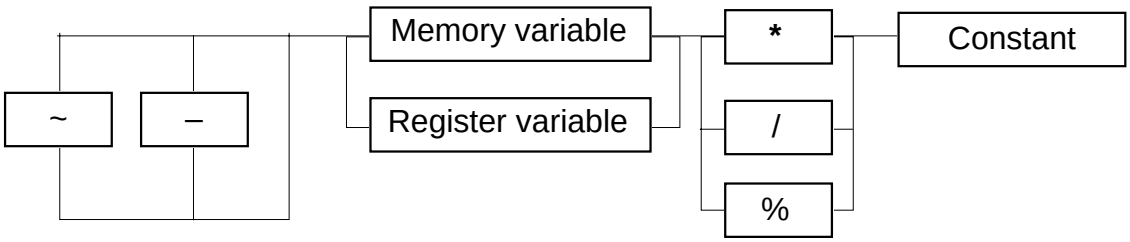


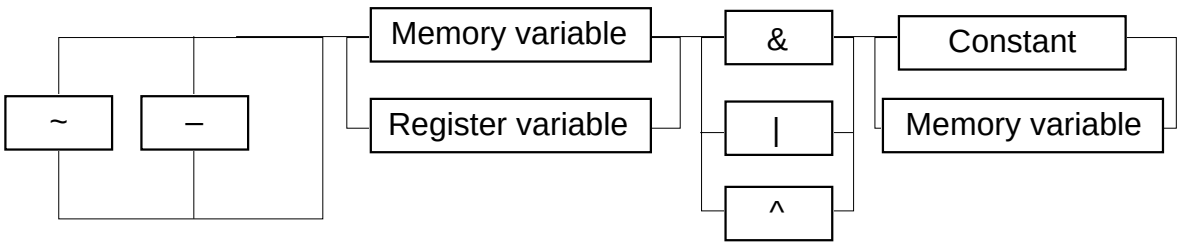
• **Expression 3**





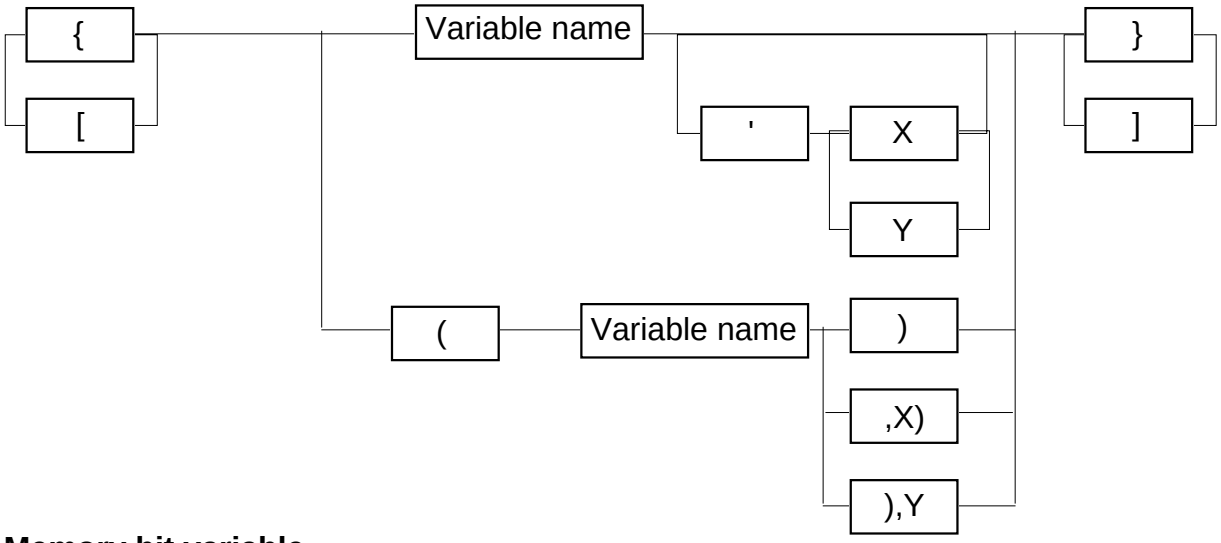




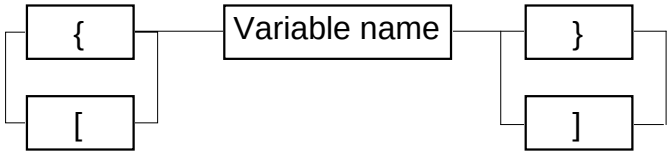


• **Variable**

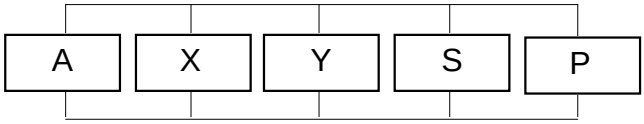
Memory variable



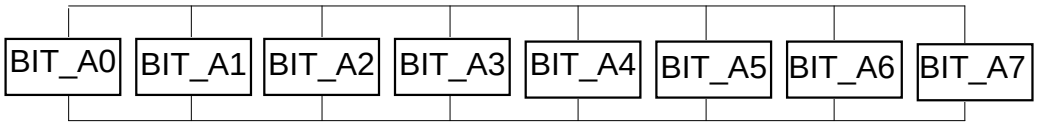
Memory bit variable



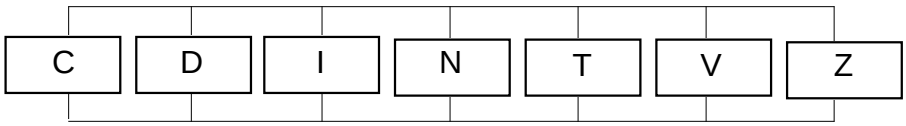
Register variable



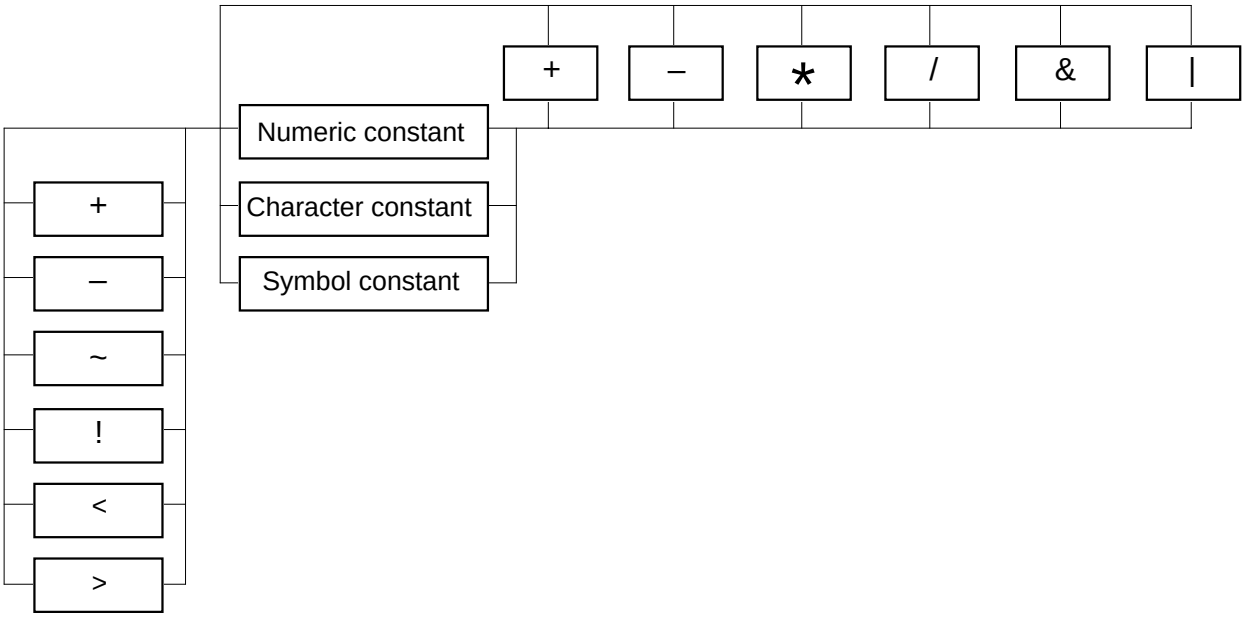
Register bit variable



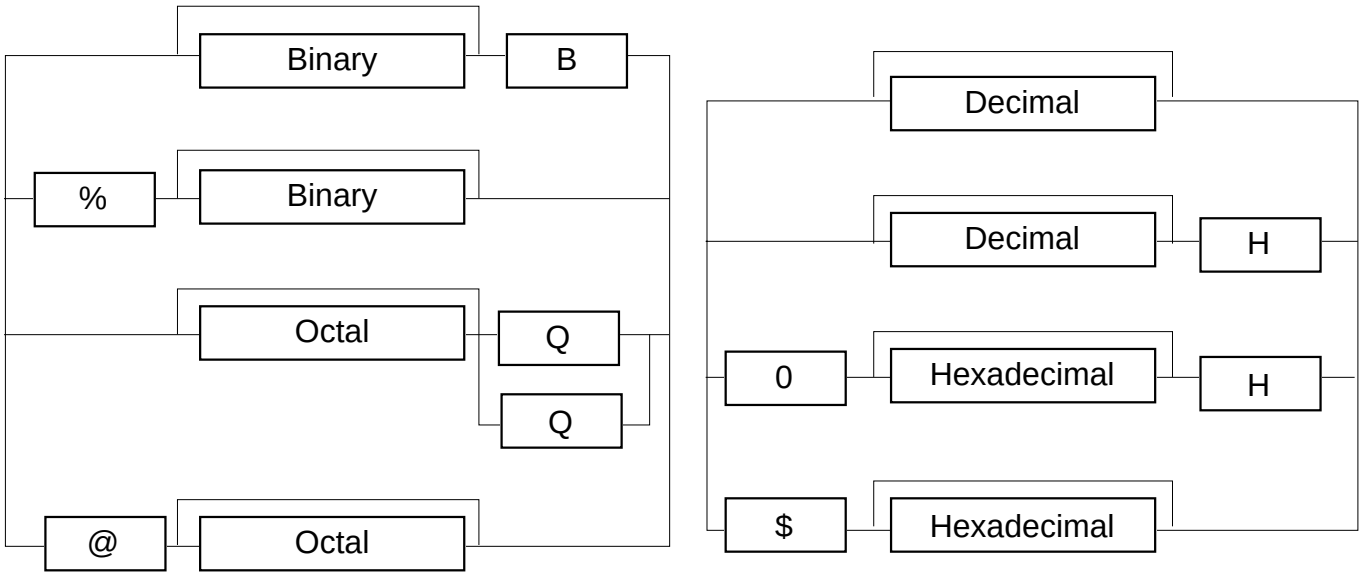
Flag variable



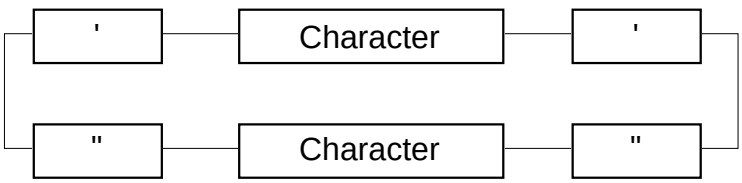
Constant



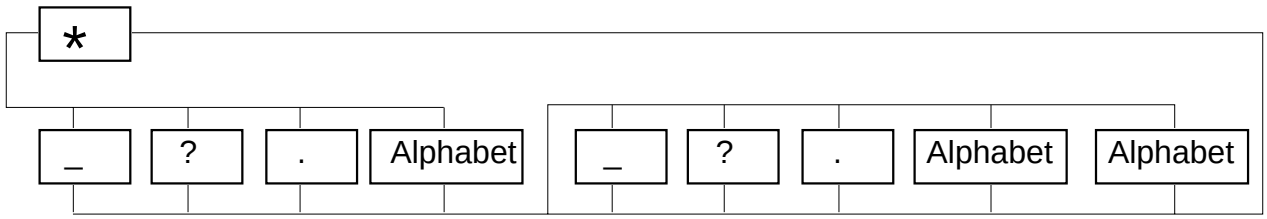
Numeric constant



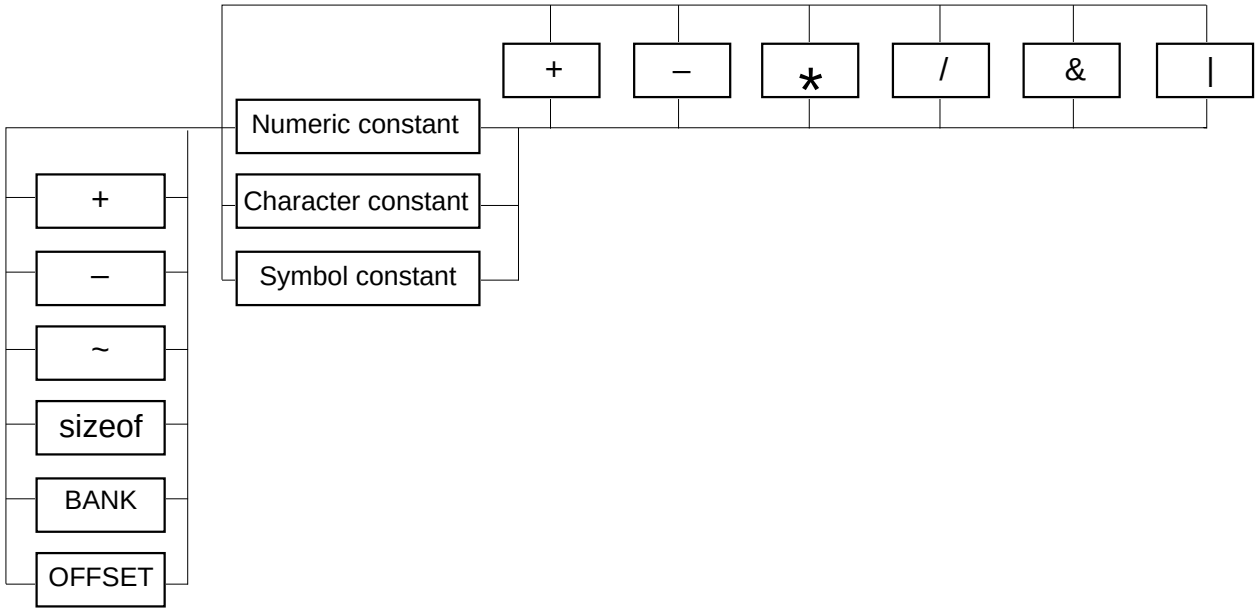
Character constant



Symbol constant



Variable name



List of Reserved Words

Symbol

..0~..65535 label
.ASSERT pseudo command
.BEXT pseudo command
.BLKB pseudo command
.BYTE pseudo command
.COL pseudo command
.D0~.D65535 label
.ELSE pseudo command
.END pseudo command
.ENDFUNC pseudo command
.ENDIF pseudo command
.ENDIO reserved pseudo command
.ENDM macro command
.ENDPROC reserved pseudo command
.ENDRAM reserved pseudo command
.EQU pseudo command
.ERROR pseudo command
.EXITM macro command
.EXT pseudo command
.F0~.F65535 label
.FUNC pseudo command
.I0~.I65535 label
.IF pseudo command
.INCLUDE pseudo command
.IO reserved pseudo command
.LIB pseudo command
.LINE pseudo command
.LIST pseudo command
.LISTM pseudo command
.LOCAL macro command
.MACRO macro command
.NLIST pseudo command
.NLISTM pseudo command
.OBJ pseudo command
.ORG pseudo command
.PAGE pseudo command
.PMOD pseudo command
.PROCINT reserved pseudo command
.PROCMAIN reserved pseudo command
.PROCSUB reserved pseudo command
.PROGNAME reserved pseudo command
.PUB pseudo command
.RAM reserved pseudo command
.REPEAT macro command
.REPEATC macro command
.REPEATI macro command
.RMOD pseudo command
.S0~.S65535 label
.SECTION pseudo command
.SEXT pseudo command
.SMOD pseudo command
.VER pseudo command
.WORD pseudo command
.ZBEXT pseudo command
.ZEXT pseudo command
.ZMOD pseudo command
??0~??65535 label

A

A accumulator
ADC mnemonic
AND mnemonic
ASL mnemonic

B

BBC mnemonic
BBS mnemonic
BCC mnemonic
BCS mnemonic
BEQ mnemonic
BIT mnemonic
BIT_A0 symbol
BIT_A1 symbol
BIT_A2 symbol
BIT_A3 symbol
BIT_A4 symbol
BIT_A5 symbol
BIT_A6 symbol
BIT_A7 symbol
BMI mnemonic
BNE mnemonic
BPL mnemonic
BRA mnemonic
BREAK structured command
BRK mnemonic

STA mnemonic
STP mnemonic
STX mnemonic
STY mnemonic
SWITCH structured command

T

T X modifier flag
TAX mnemonic
TAY mnemonic
TST mnemonic
TSX mnemonic
TXA mnemonic
TXS mnemonic
TYA mnemonic

V

V overflow flag

W

WHILE structured command
WIT mnemonic
WITH_C structured command

X

X index register X

Y

Y index register Y

Z

Z zero flag

PART 2

LINKAGE EDITOR FOR THE 740 FAMILY

LINK74 OPERATION MANUAL

RENESAS
Renesas Technology Corp.

Table of Contents

List of Figures

List of Tables

CHAPTER 1
The Format of the Manual

The LINK74 operation manual contains the following chapters:-

- Chapter 2: Outline

  The basic functions of LINK74 and the files which it generates.

- Chapter 3: Section functions

  The sections, the basic operational units of LINK74.

- Chapter 4: Operating method

  How to enter the commands used by LINK74.

- Appendix A: Error messages

  A list of the error messages used by LINK74 along with clarifications of the content of the messages and appropriate user responses.

CHAPTER 2
Outline

LINK74 links the relocatable files produced by SRA74 with library files to produce machine language data files for the 740 Family.

2.1 Functions

LINK74 can be used in conjunction with debugger for 740 family. The following functions are provided to ensure that the RTT74 and LIB74 functions can be used to the full.

1. LINK74 joins together in series all areas from separate relocatable files which have the same section[1] name.

2. The section sequence and start address for individual section units can be specified.

3. Library files created by LIB74 can be used.

4. LINK74 produces the map files required for debugging.

5. LINK74 produces the symbol files required for symbolic debugging.

[1] Element differing physically from other elements, such as the ROM area and RAM area making up a program.

2.2 Generating files

LINK74 generates 3 file types as outlined below:-

1. Machine language file (Hereafter referred to as the HEX file)
    - Intel HEX format machine language file.
    - The file attribute is .HEX.
2. Map file (Hereafter referred to as the MAP file)
    - Contains address data for the eventual locations of each section
    - This file can be output to the printer and used for debugging or understanding the memory requirements of each section.
    - MAP file generated on specification of command parameter “-M”.
    - The file attribute is .MAP.
3. Symbol file (Hereafter referred to as the SYM file)
    - File containing the information required for symbolic debugging.
    - This file is not in a format designed for listing, so it should not be printed out.
    - File generated on specification of command parameter “-S”.
    - The file attribute is .SYM.

2.3 Configuration of the MAP file

Fig 2.1 shows an example of an output MAP file. The MAP file contains the following data:-

1. Information in section units, pertaining to how much data has been linked, and from which relocatable files. The following types of information are output.

   - ATR: Indicates relative or absolute attribute[2]. REL indicates relative, and ABS indicates absolute.
   - TYPE: Indicates whether RAM area or ROM area.
   - START: Indicates the start address.
   - LENGTH: Indicates the size of the area in bytes.
   - If a library file is linked, both the name of the library file and the name of the relocatable file are indicated. The name of the relocatable file is indicated within brackets ().

2. Global label list

   Indicates global labels[3] in the program, together with their absolute addresses. This is only output when the command parameter “-MS” has been specified.

3. Global symbol list

   Indicates global symbols[4] in the program, together with their absolute addresses. This is only output when the command parameter “-MS” has been specified.

4. Global bit symbol list

   Indicates global bit symbols[5] in the program, together with their bit values and absolute addresses. This is only output when the command parameter “-MS” has been specified.

[2] In the assembly language source code, start addresses specified by the pseudo-command .ORG (or *=) become absolute attributes.

[3] Indicates labels declared by the pseudo-command .PUB.

[4] Indicates symbols declared by the pseudo-command .PUB.

[5] Indicates bit symbols declared by the pseudo-command .PUB.

```
740 Family LINKER V.4.00.00    MAP FILE      Thu Dec 10 18:30:58 1992

SECTION          FILENAME             ATR.  TYPE  START  LENGTH

WORKRAM          MAIN.R74             ABS   RAM   0000   0080
            SUB.R74               REL   RAM   0800   0100
            UTIL.LIB              REL   RAM   0180   0008
            (CALC.R74)
PROM        MAIN.R74              REL   ROM   C000   1080
            SUB.R74               REL   ROM   D800   1500
            UTIL.LIB              REL   ROM   ED00   0820
            (CALC.R74)
DROM        MAIN.R74              REL   ROM   F520   0023
            SUB.R74               REL   ROM   F544   0030

GLOBAL LABEL INFORMATION

ADCNT           0030   COUNT          009C    DATA0            00A4
DATA1           00A6   MAIN           C000    TIME             00C6

GLOBAL SYMBOL INFORMATION

GLOBAL BIT SYMBOL INFORMATION
```

Fig 2.1: Example of a MAP file output

CHAPTER 3
Section Functions

3.1 The role of sections

Programs written in assembly language are generally structured with a RAM area, program area and fixed data area. A relocatable file is generated when an SRA74 source program is assembled, and a relocatable file contains at least one of these areas. Each area is called a section. Some examples are given below to show the contents of sections and the way that they are used.

Two of the most straightforward examples are the relocatable files MAIN.R74 and SUB.R74. As can be seen from fig 3.1, each of these relocatable files consists of a RAM area, program area and fixed data area.

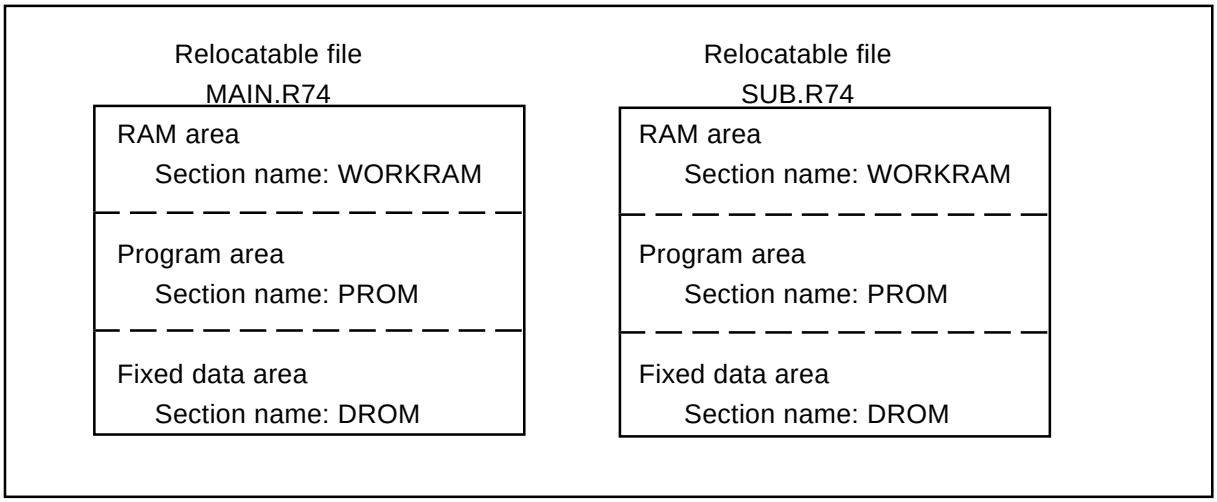


Fig 3.1: Relocatable file structure

To arrange these relocatable files in the sort of memory space shown in fig 3.2, the sections to be linked should be given the same name using the assembler pseudo-command .SECTION. Doing this ensures that the sections are assigned to continuous areas when linked. LINK74 allows the start address for each section to be specified at the time of linking.

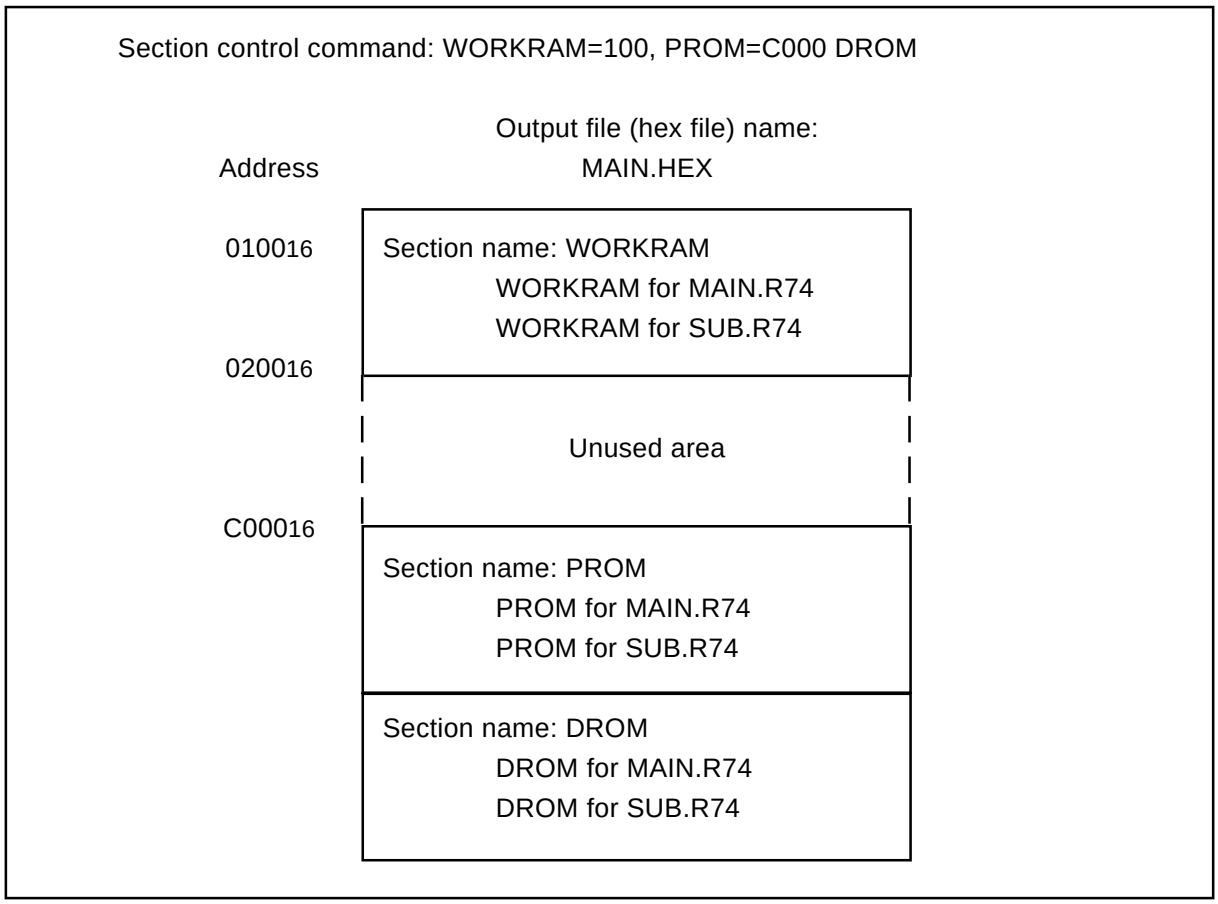


Fig 3.2 System memory map

As can be seen, LINK74 can be used to create machine language data corresponding to the final address in the customer's system.

3.2 Section attributes

Each section contains two different types of information, information about address assignment (relocatable or fixed) and information about the physical arrangement (ROM or RAM). The former is called the address attribute, and the latter is called the physical attribute. Details of each these attributes are given below.

3.2.1 Address attributes

There are two address attributes, relative attribute and absolute attribute. They are determined by the presence or absence of the pseudo-command .ORG (or *=) in the appropriate section of the source program. Details are given below.

1. Relative attribute
   - If there is no pseudo-command .ORG in a section, it becomes a relative attribute section. Relative attribute sections can be relocated, and their start addresses can be specified at the time of linking.

2. Absolute attribute
   - If there is a pseudo-command .ORG in a section, it becomes an absolute attribute section. Absolute attribute sections are located at the address specified by the pseudo-command .ORG.
   - The start address of an absolute attribute section cannot be specified at the time of linking.

Sections with the same name in different relocatable files can have different address attributes.

3.2.2 Physical attribute

There are two physical attributes, ROM attribute and RAM attribute. They indicate the physical characteristics of the area where the section will be located. Details are given below.

1. ROM attribute
   - Sections including code-generating statements in the assembly language source program are ROM attribute sections.
   - ROM attribute sections generate HEX files as the result of linking.
   - To avoid confusion with address attributes, sections with ROM attributes are called ROM type.

2. RAM attribute
   - RAM attribute sections do not generate HEX files as the result of linking.
   - Sections including the area securing pseudo-command .BLKB in the assembly language source program are RAM attribute sections.
   - To avoid confusion with address attributes, sections with RAM attributes are called RAM type.

Sections with the same name must all have the same physical attribute.

3.2.3 Reserved sections

LINK74 treats sections Z, S, P and R as reserved sections. These reserved sections are described below.

1. Z section
   The Z section is the RAM attribute section located in the 740 Family zero page (from address 00_{16} to FF_{16}). If a section name is declared as 'Z' (or 'z') by the SRA74 pseudo-command .SECTION or if the pseudo-command .ZMOD is declared, the section becomes the Z section.

   If the Z section is a relative attribute section, the start address can be specified by LINK74 command input. If the start address specification command is omitted, 00_{16} is selected as the start address.

2. S section
   The S section is the ROM attribute section located in the 740 Family special page (from address $FF00_{16}$ to $FFFF_{16}$). If a section name is declared as 'S' (or 's') by the SRA74 pseudo-command .SECTION or if the pseudo-command .SMOD is declared, the section becomes the S section.

   If the S section is a relative attribute section, the start address can be specified by LINK74 command input. If the start address specification command is omitted, $FF00_{16}$ is selected as the start address.

3. P section
   The P section is a ROM attribute section with program or fixed data. It can be located anywhere within the program area (from address 00_{16} to $FFFF_{16}$). If a section name is declared as 'P' (or 'p') by the SRA74 pseudo-command .SECTION or if the pseudo-command .PMOD is declared, the section becomes the P section.

   If the P section is a relative attribute section, 00_{16} is selected as the start address if the start address specification command is omitted.

4. R section
   The R section is a RAM attribute section. It can be located anywhere within the program area (from address 00_{16} to $FFFF_{16}$). If a section name is declared as 'R' (or 'r') by the SRA74 pseudo-command .SECTION or if the pseudo-command .RMOD is declared, the section becomes the R section.

   If the R section is a relative attribute section, 00_{16} is selected as the start address if the start address specification command is omitted.

5. E section
   Defined symbols in section E and source line debug information in the section are not output to relocatable files. When section E (small case letters accepted) is declared using the .SECTION pseudo-command of SRA74, that section becomes section E. Always specify the section address in advance with the .ORG pseudo-command. And, always specify the -BANK option to assemble programs when running SRA74. Unless these command options are specified, warning is output for the defined lines in section E.

3.3 Basic function of sections

1. During linking, sections with the same names are given locations next to each other. Other sections cannot be located between sections with the same name.

2. The sequence of sections with the same name is determined by the sequence in which the relocatable file names are specified in the link command.

CHAPTER 4
Operation Method

4.1 Activation method

Before using LINK74 the following data (input parameters) must be entered:-

1. Relocatable file name (required item)

2. Library file name

3. Section control data

4. Command parameters

In LINK74, this data can be input in any of three different ways according to the operating environment.

1. Interactive input

2. Command line input

3. Command file input

These different input methods are used for the same parameters. Section 4.2 below contains explanations of the input parameters and Section 4.3 outlines with the help of examples the different methods of entering the parameters.

4.2 Input parameters

4.2.1 Relocatable file name

1. Always enter the name of the relocatable file.

2. The relocatable file must have the file attribute .R74, but the attribute part of the filename may be omitted when the command is input.

3. The directory path can also be specified within the file name. If only the file name is specified then the processing operation will be carried out on a file contained within the current directory in the current drive.

4. The name for the output file is taken from the name of the first relocatable file specified. If an output file name is specified using the command parameter -F, this takes precedence.

5. The directory where the output file is output is taken from the directory for the first relocatable file specified. If a directory is specified using the command parameter -O, this takes precedence.

4.2.2 Library file name

1. The name of the library file may be omitted.

2. The library file must have the file attribute .LIB, but the attribute part of the filename may be omitted when the command is input.

3. The directory path can also be specified within the file name. If only the file name is specified then the processing operation will be carried out on a file contained within the current directory.

4. The library file is only accessed when there are global labels or global symbols which cannot be resolved within the relocatable file.

4.2.3 Section control

1. Section data may be omitted. If omitted, the sections are arranged in the order they are encountered in the relocatable file read in.

2. Specify section data for relative attribute files. Absolute attribute files are located at fixed addresses specified by the pseudo-command .ORG regardless of whether or not a section is specified.

3. Section locations should be specified in order starting with the lowest address. Separate each of the section names with spaces.

4. The start address for each section is specified by following the section name with “=” (equals sign), then by the address. Specify the address in hexadecimal. The initial “0” and final “H” are not required.

5. Section names can be specified with either upper case or lower case characters.

6. If the start address for a relative attribute file is omitted, the start address becomes 0000_{16}. Reserved section S starts from $FF00_{16}$.

7. Overlapping section address results in error. However, if the command parameter “-A” is specified, absolute addresses may overlap. This enables absolute address labels such as SFR area to be included in several program files with the pseudo-command .INCLUDE without defining it as external reference.

4.2.4 Command parameters

Command parameters control the output file from linking, presence or absence of a version check and similar items. Table 4.1 gives details of the command parameters.

Table 4.1: List of command parameters

Command parameter	Description
-A	Allows overlapping of absolute attribute sections with the same name. Useful when sharing global memory area.
-BANK	Expands the address area upper limit from FFFFH to 1FFFFH.
-F	Specifies the output file name. The specification format is as follows: -Fsample
-M	Outputs a MAP file. (section data only)
-MS	Outputs a MAP file with global labels, global bit symbols and a global symbol list.
-N	Causes references to the .R74 and .LIB files specified by means of pseudo-commands .OBJ and .LIB in the source file to be disregarded.
-O	Specifies the directory for the output file. The specification format is as follows: -OC:\USR\WORK This specifies the output directory as directory \USR\WORK on drive C.
-S	Outputs a SYM file.
-V	Checks that the versions of different relocatable files are consistent. In order for this check to be performed, the version must be specified in the assembly language source file using the pseudo-command .VER.

4.3 Methods of entry

4.3.1 Interactive input

Interactive input has the following features.

1. The relocatable file, library file, section control commands and command parameters are entered interactively in this sequence.

2. This method of entry is convenient when there are only a few relocatable files or sections, or when using trial and error to determine the best address.

3. If essential commands are not present during command line input or command file input, the system automatically switches to interactive input.

Interactive input is activated by entering LINK74<RET> at the operating system prompt. When activated, LINK74 outputs a display like the one below.

```
A>LINK74<RET>
740 Family LINKER V.4.00.00
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Relocatable files (.R74) >>
```

Fig 4.1: LINK74 initial display

The last line in fig 4.1 indicates that LINK74 is waiting for a relocatable file to be entered. Enter the relocatable file to be linked after “>>”. With interactive input, then LINK74 waits for the library file, section controls and command parameters to be entered in the same way. Enter all these as shown in fig 4.2. (To enter more than one file, separate individual file names with spaces. The same applies for the library file name input, section control and command parameters.)

```
A>LINK74<RET>
740 Family LINKER V.4.00.00
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Relocatable files (.R74) >> MAIN SUB<RET>
Libraries         (.LIB) >> UTIL1 UTIL2<RET>
Section information      >> WORKRAM=100 PROM=C000 DROM<RET>
Command parameter        >> -O\WORK -M -S<RET>
```

Fig 4.2: Display during interactive input

4.3.2 Command line input

Command line input has the following features.

1. With command line input, the whole command is entered on the operating system command line.

2. MS-DOS restricts the length of a command to 127 characters, so command line input should be used when the number of relocatable files or sections is small.

3. Command line input may also be used inside batch files and make files.

4. The four different types of data in the input parameters should be separated by commas. Fig 4.3 shows the same command as fig 4.2, but this time entered on the command line.

5. Even if the library name and subsequent parameters are not required, the commas must still be entered. For instance, fig 4.4 shows the command line input for the command in fig 4.3 if the library is not required.

6. In the special case where command parameters are not required, two commas are required to show clearly that there are no command parameters. The example from fig 4.4 is shown with the command parameters omitted in fig 4.5.

7. If the command parameters entered are insufficient, the system automatically switches to interactive input mode.

```
A>LINK74 MAIN SUB, UTIL1 UTIL2, WORKRAM=100 PROM=C000 DROM, -0\WORK -M -S<RET>
```

Fig 4.3: Command line input - Example 1

```
A>LINK74 MAIN SUB,, WORKRAM=100 PROM=C000 DROM, -0\WORK -M -S<RET>
```

Fig 4.4: Command line input - Example 2 (No library file)

```
A>LINK74 MAIN SUB,, WORKRAM=100 PROM=C000 DROM,,<RET>
```

Fig 4.5: Command line input - Example 3 (No command parameters)

4.3.3 Command file input

Command file input has the following features.

1. With command file input, a command file is created in advance and specified when LINK74 is activated.

2. Command file input is convenient when there are too many characters to use command line input.

3. The command file name is specified as shown in fig 4.6 when activating LINK74 by entering "@" in front of the file name. In fig 4.6, the contents of CMD.DAT are executed as the command.

4. The contents of the command file are the same as for command line input (the "LINK74" can be omitted, however). The <RET> is disregarded, so long commands can be entered on several lines. The example in fig 4.5 can be written as the command file shown in fig 4.7.

5. If the command parameters entered are insufficient, the system automatically switches to interactive input mode. For example, if the last two commas are omitted from fig 4.7, LINK74 outputs an interactive input display to request the command parameters.

```
A>LINK74 @CMD.DAT<RET>
```

Fig 4.6: Specifying a command file

```
MAIN SUB
,
,WORKRAM=100 PROM=C000 DROM
,,
```

Fig 4.7: Command file layout - Example

4.4 Errors

4.4.1 Types of error

The types of errors which occur during the operation of LINK74 have the following causes:-

1. Errors relating to the operating system

   These are errors such as insufficient disc or memory capacity which relate to LINK74's operating system. Please refer to the list of error messages in Appendix A and proceed according to the operating system commands.

2. Errors relating to the input of LINK74 command lines

   These are errors relating to the input of the command lines to activate LINK74. Please study the contents of this chapter and then enter the relevant commands again.

3. Errors relating to the contents of the relocatable files constituting the object of the linking operation

   These are errors relating to the relocatable files such as the dual definition of a global label or reference to an external label which has not been defined. Please correct the relevant contents of the source file and carry out the assembly operation again where required.

4. Error relating to LINK74 functions

   This error occurs if the SRA74 and LINK74 versions do not match. Please ask your supplier if you require a more detailed explanation.

LINK74 outputs the details of an error onto the VDU screen in the format shown in Fig 4.8. Please study the list of errors in error number order in Appendix A and take action accordingly.

```
A>LINK74 MAIN SUB,,WORKRAM=100 PROM=C000 DROM,,
MELPS 740 LINKER V.1.00.00C
Copyright 1989, 1990, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SOFTWARE CORPORATION
All Rights Reserved.

now processing pass 1
processing "MAIN.R74"
ERROR NO.2: Out of heap space

A>
```

Fig 4.8: Example of error display

4.4.2 Value for return to operating system

When entering an execution command into an operating system batch file, for example, there are times when you may wish to change the contents of the processing operation in accordance with the outcome of the execution. In LINK74 the execution results are divided into 5 error levels as shown in Table 4.2 and returned to the operating system.

Table 4.2: List of error levels

Error level	Description of execution results
0	Normal termination
1	Error relating to content of relocatable file which constitutes the object of linking
2	Error relating to input of LINK74 commands
3	Error relating to the operating system
4	Error relating to LINK74 functions

4.5 Environment variables

LIB74

LINK74 refers to the environment variables in LINK74 at the point when it reads in the library file. If the directory path for the library file is specified by setting the environment variable LIB74, the directory path does not need to be specified when the command is entered.

Details are given in ordinary OS manuals.

APPENDIX A
Error Messages

Table A.1: List of system errors

Error number	Error message	Description and user action
0	xxx file not found	The file specified in the input cannot be found. This error message can also appear for files specified by the pseudo- commands .OBJ or .LIB. ⇒ Enter the correct filename
1	Invalid command input	The command input has more than four parameters, or the total number of characters used for the parameters is 2048 or more. ⇒ Input the command again within the limits imposed by these restrictions.
2	Out of heap space	The memory space required for the operation of the linker is insufficient. ⇒ Reduce the number of public symbols.
3	Invalid section information	Section information is incorrect. ⇒ Input the command again in the format “Section name=address”
4	Invalid parameter input “xxx”	Command parameter specification is incorrect. ⇒ Input the command again correctly.
5	Non relocatable file name	No relocatable file name has been input. ⇒ Input the file name
6	Internal error	An internal error has occurred in LINK74. ⇒ Please notify your supplier.

Error number	Error message	Description and user action
7	xxx relocatable format is mismatch	The .R74 relocatable file format versions do not match. ⇒ This error occurs when there is a mismatch between the assembler or library and the linker being used. The .R74 file and .LIB file processed must be created by SRA74 and LIB74 having the same version number as LINK74.
8	Program version is different	Mismatch with the program version declared by pseudo-command .VER. ⇒ Either make the version specified in the relocatable file by pseudo-command .VER match, or use the -V parameter to cancel the parameter specification.
9	Unresolved label “xxx” in xxx	A symbol or label declared for external reference in the section is not defined. ⇒ Link a program with the label or symbol declared as public.
10	“xxx” is multiple defined in xxx. others in xxx	Label or symbol is defined more than once. ⇒ Modify the label or symbol name.
11	Location overlap. SECTION=xxx ADDRESS=xxx in xxx	The section address space overlaps. ⇒ Check the section address location, and ensure that an address overlap does not occur. (For absolute attribute sections, modify the pseudo-command .ORG in the source file.)
12	SECTION xxx is an absolute	A start address is specified for an absolute attribute section using a section control command. ⇒ Either stop specifying the address with the command input, or change the section to a relative attribute section.

Error number	Error message	Description and user action
14	Can't find SECTION xxx	The section cannot be found. ⇒ Specify the section information correctly. (Take care with upper and lower case letters - section names make a distinction between upper and lower case.)
15	Can't create xxx	This file cannot be generated. ⇒ Check the specification of the -O parameter and input again.
16	File seek error xxx	Seek failure. ⇒ This error involves the operating system. It most frequently occurs when there is a hardware problem with a disk drive.
17	Expression value is out of range. SECTION=xxx ADDRESS=xxx OFFSET=xxx	The value resulting from calculation of an expression is out of range. (The section name, absolute address and offset from the start of the section are given to indicate the location of the error.) ⇒ Modify the program to ensure that the expression value does not go out of range.
18	Out of disk space	There is not enough space left on the disk selected for output. ⇒ Create more space on the disk.
19	Relative jump is out of range. SECTION=xxx ADDRESS=xxx OFFSET=xxx	The jump destination address of the relative jump command cannot be accessed. ⇒ Modify the program so that the jump destination label is within range. (The section name, absolute address and offset from the start of the section are given to indicate the location of the error.)

Error number	Error message	Description and user action
22	Out of maximum program size	The program area exceeds the 64KB (FFFF16) maximum area. ⇒ Reduce the size of the program.
23	Section type mismatch in SECTION xxx	Section with ROM type and RAM type mixed. ⇒ Ensure that ROM and RAM type are not mixed in the same section.
25	Expression is out of ZERO page. SECTION=xxx ADDRESS=xxx OFFSET=xxx	The result of the formula processed for zero page addressing exceeds the range from 0016 to FF16. ⇒ Modify to ensure that the result is within the zero page range.
26	Expression is out of SPECIAL page. SECTION=xxx ADDRESS=xxx OFFSET=xxx	The result of the formula processed for special page addressing is outside the range from FF0016 to FFFF16. ⇒ Modify to ensure that the result is within the special page range.
27	label “xxx” type is mismatch	The type (label or bit symbol) of an externally referenced label differs from the declaration. ⇒ Correctly declare the pseudo-command for external reference.
28	Section “xxx” information is out of range	The section start address specified by command input is out of range. ⇒ Specify the start address correctly.
29	Bit value is out of range. SECTION=xxx ADDRESS=xxx OFFSET=xxx	The result of calculating a bit value is outside the range from 0 to 7. ⇒ Modify to ensure that the result is within the range.
30	Extend area requires command option '-BANK'	The expansion area is not defined. ⇒ Link it by specifying the -BANK command option.

PART 3

LIBRARIAN
FOR THE 740 FAMILY

LIB74 OPERATION MANUAL

RENESAS
Renesas Technology Corp.

Table of Contents

List of Figures

List of Tables

CHAPTER 1
The Format of the Manual

The LIB74 operation manual contains the following chapters:-

- Chapter 2: Outline

  The basic functions of LIB74 and the files which it generates.

- Chapter 3: Operating method

  How to enter the commands used by LIB74.

- Appendix A: Error messages

  A list of the error messages used by LIB74 along with clarifications of the content of the messages and appropriate user responses.

CHAPTER 2
Outline

LIB74 is a program for managing the structured relocatable files produced by SRA74 in the form of a library. Frequently used subroutines can be included in a library to reduce assembly time and promote the reuse of programs.

2.1 Functions

LIB74 processes the relocatable files created by SRA74. Files created by LIB74 can be accessed by LINK74[1]. The following functions are provided to ensure that their functions can be used to the full.

1. LIB74 creates and modifies library files that can be accessed by LINK74.
2. Relocatable files can be stored in library files.
3. Unrequired relocatable files can be deleted from library files.
4. Old relocatable files in a library file can be updated to create new relocatable files.
5. Relocatable files stored in a library file can be extracted.
6. Information of relocatable files in a library file can be displayed.

2.2 Advantages

1. Linking can be made faster

   Storing relocatable files in a library enables the fast access to the information required in linking, and accelerates the LINK74 linking operation.

2. When relocatable files in a library file are updated, the update dates can be compared, making it possible to cut down processing work by only processing new versions of relocatable files. (When the -U command parameter is specified.)

[1] A linkage editor for use with the 740 Family.

2.3 Generating files

LIB74 generates 4 file types as outlined below:-

1. Library file
   - File containing edited relocatable files with an index to labels and symbols.
   - In the library file, relocatable files are handled as individual modules. (Here, the relocatable files in a library file are referred to as modules.)
   - A module name is the same as the relocatable file name including the file attribute.
   - The file attribute of a library file is .LIB.
2. Program list file (Hereafter referred to as the LST file)
   - File generated on specification of command parameter “-L”.
   - Contains an index with relocatable file names, global labels and symbols.
   - The file attribute is .LST.
   - The structure of this file is described in the next section.
3. Relocatable file
   - File generated on specification of command parameter “-X”.
   - File regenerated from a relocatable file stored in a library file
   - A regenerated relocatable file is identical to the relocatable file created by SRA74 before it was stored in the library file.
   - The file attribute is .R74.
4. Backup file
   - Backup files are generated regardless of command parameter specifications, unless LIB74 terminates abnormally.
   - Whenever a library file is modified, the library file before modification is retained as a backup.
   - The file attribute is .BAK.

Note:
Library files, relocatable files, and backup files are in binary format, so they should not be output to the screen or printer.

2.4 Configuration of the LST file

Fig 2.1, fig 2.2 and fig 2.3 show examples of an output LST file. The LST file contains the following data:-

1. List of modules

   The following information stored in the library file is output.

   - Module_name: Names of modules stored in the library file The output sequence is the same sequence that the files were stored in the library file.
   - Offset: The number of bytes from the start of the module to start of the module (hexa-decimal)
   - Module_size: Module memory capacity (hexadecimal)

```
LIB74 librarian V.1.00.10            date 1998-Dec-10 15:30  page 1

Library file name :    CALC8.LIB
Relocatable format:    VER.A
Last update time  :    1998-Dec-10 15:30
Number of modules :    3
Number of global symbols:   10

Module_name:
key_scan ..... Offset: 00000000H Module size: 00000100H
multiply ..... Offset: 00000180H Module size: 00000080H
division ..... Offset: 00000200H Module size: 000000A5H
```

Fig 2.1: LST file output - example 1 (list of modules)

2. Global label and symbol list (alphabetic order)

   Two separate tables are output
   - PUBLIC symbol table
     Symbol_name: Indicates public labels, public symbols and public bit symbols. Public symbols declared with the pseudo-command .EQU are marked with “(e)”. Public bit symbols are marked with “(b)”.

     Module_name: Indicates the modules containing the public labels, public symbols and public bit symbols.

- EXTERN symbol table
  Symbol_name: Indicates external labels and external symbols.

  Module_name: Indicates the modules containing external reference specifications.

```
LIB74 librarian V.1.00.10            date 1998-Dec-10 15:30  page 2

PUBLIC symbol table (symbol count = 0010)

  Symbol_name      Module_name           Symbol_name      Module_name

  _dividel........ division              _division....... division
  _key_flg1(b).... key_scan              _key_flg2(b).... key_scan
  _key_scan....... key_scan              _key_scn1....... key_scan
  _multi1......... multiply              _multiply....... multiply
  _one(e)......... key_scan              _two(e)......... key_scan



LIB74 librarian V.1.00.10            date 1998-Dec-10 15:30  page 3

EXTERN symbol table (symbol count = 0010)

Symbol_name        Module_name           Symbol_name      Module_name

  _div_ansl....... division              _div_ansh....... division
  _key_buff1...... key_scan              _key_buff2...... key_scan
  _key_buff3...... key_scan              _key_buff4...... key_scan
  _mul_ansl....... multiply              _mul_ansh....... multiply
```

Fig 2.2: LST file output - example 2 (list of global labels and symbols)

3. List of global labels and symbols for each module

   Two tables are output, a PUBLIC symbol table and an EXTERN symbol table. Both show the module names, followed by the global labels and global symbols for each module.

```
LIB74 librarian V.1.00.10            date 1998-Dec-10 15:30  page 4

PUBLIC symbol table

Module name: key_scan (symbol count = 0006)

_key_flg1(b)      _key_flg2(b)      _key_scan         _key_scn1
_one(e)           _two(e)

Module name: multiply (symbol count = 0002)

_multi1           _multiply

Module name: division (symbol count = 0002)

_dividel          _division


LIB74 librarian V.1.00.10            date 1998-Dec-10 15:30  page 5

EXTERN symbol table

Module name: key_scan (symbol count = 0006)

_key_buff1        _key_buff2       _key_buff3        _key_buff4
_linecnty         _linecntx

Module name: multiply (symbol count = 0002)

_mul_ansl         _mul_ansh

Module name: division (symbol count = 0002)

_div_ansl         _div_ansh
```

Fig 2.3: LST file output - example 3 (list of global labels and symbols for each module)

CHAPTER 3
Operation Method

3.1 Activation method

Before using LIB74 the following data must be entered:-

1. Library file name (required item)

2. Relocatable file name

3. Command parameters

In LIB74, this data can be input in any of two different ways according to the operating environment.

1. Command line input

2. Command file input

These different input methods are used for the same parameters. The following sections contain explanations of the input parameters and outline with the help of examples the different methods of entering the parameters.

3.2 Input parameters

3.2.1 Library file name

1. Always enter the name of the library file.

2. The name for a library file to be edited should be entered after command parameter -O, and separated from the command parameter by a space or tab character.

3. The directory path can also be specified within the file name. If only the file name is specified then the system will search for the library file in the directory specified by the environmental variable LIB74. If the environmental variable has not been set, processing will be carried out on a file contained within the current directory.

4. The file attribute .LIB may be omitted.

3.2.2 Relocatable file name

1. Several relocatable files may be specified, separated by space or tab characters.

2. The names of the relocatable files to be processed should be entered after command parameter -F, and separated from the command parameter by a space or tab character.

3. The directory path can also be specified within the file name. If only the file name is specified then the processing operation will be carried out on a file contained within the current directory on the current drive.

4. The .R74 attribute part of the filename may be omitted when the command is input.

3.2.3 Command parameters

Command parameters control output file specification, specification of details of library file operation and similar items. Table 3.1 gives details of the command parameters.

Table 3.1: List of command parameters

Number[1]	Command parameter[2]	Description
1	-O (Output)	Specifies the library file to be edited.
2	-F (Files)	Specifies a relocatable file to be added to the library file, updated or extracted, or specifies a module to be deleted from the library file.
3	-A (Append)	Adds a relocatable file to the library file.
	-R (Replace)	Updates a module in the relocatable file.
	-D (Delete)	Deletes a specified module from the library file.
	-L (List)	Outputs a list of the modules stored in a library file.
	-X (Extract)	Extracts a specified module from the library file. A relocatable file that can be processed by LINK74 is created in the current directory from the extracted module.
4	-V (Verbose)	Displays the names of individual files on the screen as they are being processed.
	-U (Update)	When updating modules in the library file, only the module corresponding to the newest relocatable file is updated.

Notes:

1. The numbers indicate the following details.
    1. Essential parameter. Always enter the name of the library file to be edited.
    2. When one of the -A, -R and -X parameters is specified to append, replace or extract a relocatable file, the name of the relocatable file must be specified. To deleting a module, the name of the module must be specified.
    3. One of these five parameters must be specified.
    4. Specify as required.

2. No distinction is made between upper case and lower case letters for command parameters . Either -A or -a is acceptable.

3.2.4 Details of command parameters

The individual command parameters are described in detail below.

1. -O
   - Specifies the library file to be edited.
   - The library file name can be specified with a directory path name.

     Example: A>LIB74 -LO B:\USR\TEST<RET>
   - If the directory path specification is omitted, the system first searches in the directory specified by the environmental variable LIB74, then in the current directory. If the environmental variable is set as shown in the example shown below, a file in the \USR\LIB directory on drive B: will be processed.

     Example: A>SET LIB74=B:\USR\LIB
   - If the directory path specification is omitted and no environmental variable has been set, a file in the current directory on the current drive will be processed.
   - If the attribute part of the filename is omitted, the attribute LIB is selected as the default.
   - If the full filename is specified, it is possibly to specify files with attributes other than LIB.

2. -F
   - Specifies a relocatable file to be added to the library file, updated or extracted, or specifies a module to be deleted from the library file.
   - Several files can be specified, separated by space or tab characters.
   - The file name can be specified with a directory path name.

     Example: A>LIB74 -AO TEST.LIB -F B:\WORK\SUB1 C:SUB2<RET>
   - If the directory path specification is omitted, a file in the current directory on the current drive will be processed.
   - If the attribute part of the filename is omitted, the attribute R74 is selected as the default.
   - If the full filename is specified, it is possibly to specify files with attributes other than R74.

3. -A
   - Creates a new library file or adds a relocatable file to an existing library file.
   - When creating a new library file, the relocatable files specified using the -F parameter are added to the start of the new library in order of specification.
   - When adding relocatable files to an existing library file, the relocatable files specified using the -F parameter are added to the end of the library file in order of specification.

- No check is made to see if a specified relocatable file has the same name as a module already stored in the library.
- If two identical module names are specified at the same time, the multiple definition of labels and symbols is regarded as an error and processing stops.

4. -R
   - Updates a module in the library file.
   - Used together with -U, only modules in the relocatable file with the latest update date are updated.

5. -D
   - Deletes a specified module from the library file.

6. -L
   - Outputs a list (LST file) of the modules stored in a library file.
   - When a file is specified with the -F parameter, information concerning the specified relocatable file is output to the LST file.
   - When no file is specified with the -F parameter, information concerning all modules in the library is output to the LST file.
   - The file attribute is LST.

7. -X
   - Extracts a specified module from the library file, creating a relocatable file identical to the relocatable file before storage in the library.
   - The date of extraction becomes the update date for the relocatable file.
   - The extracted relocatable file can be processed by LINK74.
   - The extracted relocatable file is created in the current directory.
   - When no relocatable file is specified with the -F parameter, all modules in the library file are extracted.
   - The contents of the library file are not modified.
   - The file attribute is R74.

8. -V
   - Displays details of files on the screen to show how they are being processed.

9. -U

- When updating modules in the library file, the update date for the relocatable file specified with the -F option is compared with the update date of the module stored in the library file. The update is only performed if the update date of the relocatable file is the newest files.
- The update date for the relocatable file is taken form the operating system file management data.

3.3 Methods of entry

3.3.1 Command line input

Command line input has the following features.

1. With command line input, the whole command is entered on the operating system command line.

2. Command line input should be used when the number of relocatable files or parameters is small.

3. Command line input may also be used inside batch files and make files.

Fig 3.1 shows the example of deleting module FILE1.R74 from the library file TEST.LIB.

```
A>LIB74 -D -O TEST.LIB -F FILE1<RET>
```

Fig 3.1: Command line input - Example 1 (deleting a module)

Fig 3.2 shows the example of adding relocatable files FILE2.R74 and FILE3.R74 to the library file TEST.LIB.

```
A>LIB74 -AO TEST.LIB -F FILE2 FILE3<RET>
```

Fig 3.2: Command line input - Example 2 (Adding relocatable files)

3.3.2 Command file input

Command file input has the following features.

1. With command file input, a command file is created in advance using an editor and specified when LIB74 is activated.

2. Command file input is convenient when there are too many files or too many characters to use command line input.

3. The command file name is specified as shown in fig 3.3 when activating LIB74 by entering "@" in front of the file name. In fig 3.3, the contents of CMD.DAT are executed as the command.

4. The contents of the command file are the same as for command line input (the "LIB74" can be omitted, however). The <RET> is disregarded, so long commands can be entered on several lines. The example in fig 3.2 can be written as the command file shown in fig 3.4.

If the command has been entered correctly, LIB74 starts processing. As soon as the processing of each LIB74 command terminates, a termination message is shown on the screen and processing finishes. Fig 3.5 shows an example of the display when LIB74 processing has terminated normally.

```
A>LIB74 @CMD.DAT<RET>
```

Fig 3.3: Specifying a command file

```
-AO
TEST.LIB
-F
FILE2
FILE3
```

Fig 3.4: Command file layout - Example

```
A LIB74 -AVO TEST.LIB -F SUB_1 SUB_100<RET>
740 Family LIBRARY MANAGER V.1.00.10
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.
<test.lib > Create
APPEND FILE = sub_1,sub_100
MODULE COUNT         000002
GLOBAL SYMBOL COUNT  000019
A>
```

Fig 3.5: Example of display when LIB74 terminates normally

3.4 Errors

3.4.1 Types of error

The types of errors which occur during the operation of LIB74 have the following causes:-

1. Errors relating to the operating system

   These are errors such as insufficient disc or memory capacity which relate to LIB74's operating system. Please refer to the list of error messages in Appendix A and proceed according to the operating system commands.

2. Errors relating to the input of LIB74 command lines

   These are errors relating to the input of the command lines to activate LIB74. If the input is incorrect, a label screen such as the one shown in fig 3.6 is displayed. Please study the contents of this chapter and then re-enter the relevant commands.

3. Errors relating to the contents of the relocatable files being processed

   These are errors relating to the relocatable files such as dual definition of a public label. Refer to the LST file to make the necessary corrections.

```
A>LIB74<RET>
740 Family LIBRARY MANAGER V.1.00.10
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Usage: A>LIB74 -[ardxluv]o <filename> [-f <filename> ...]

       -o :  library file name
       -f :  relocatable file name
       -a :  append command
       -r :  replace command
       -d :  delete command
       -x :  extract command
       -l :  listout command
       -u :  update check (option)
       -v :  verbose (option)
```

Fig 3.6: Help screen for command line errors

LIB74 outputs the details of an error onto the VDU screen. Please study the list of errors in error number order in Appendix A and take action accordingly.

3.4.2 Value for return to operating system

When entering an execution command into an Operating system batch file, for example, there are times when you may wish to change the contents of the processing operation in accordance with the outcome of the execution. In LIB74 the execution results are divided into 4 error levels as shown in Table 3.2 and returned to the operating system.

Table 3.2: List of error levels

Error level	Description of execution results
0	Normal termination
1	Error relating to the library file or relocatable file to be processed
2	Error relating to input of LIB74 commands
3	Error relating to the operating system

3.5 Environment variables

LIB74 always processes the library file in the current directory first. If there is no library file in the current directory and a directory path is specified with the environment variable “LIB74,” then the library file in the directory specified with the environment variable is processed.

```
Example: A>SET LIB74=B:\USR\LIB<RET>
```

APPENDIX A
Error Messages

A.1 System errors

When a system error is detected during LIB74 operation, an error message is displayed on the VDU and processing is suspended. Table A.1 lists all the system errors.

A.2 Library manager errors

When a processing error is detected during LIB74 operation, an error message is displayed on the VDU and processing is suspended. Table A.2 lists all the library manager errors.

Table A.1: List of system errors

Error message	Description and user action
Usage: A>LIB74 -[adluvxz]o <filename> [-f<filename> ...]	The command has been input wrongly. ⇒ Refer to the help screen and input the command again.
Can't open xxx	File not found. ⇒ Check that the file specified with -O or -F parameter exists in the specified directory
Can't create xxx	This file cannot be generated. ⇒ Check the specification of the -O parameter and input again.
Out of disk space[1]	There is not enough space left on the disk selected for output. ⇒ Create more space on the disk.
Input file read error xxx	Error occurred during file read process ⇒ This error involves the operating system. It most frequently occurs when there is a hardware problem with a disk drive.
Internal error	An internal error has occurred in LIB74. ⇒ Please notify your supplier.
File seek error xxx	Seek failure. ⇒ This error involves the operating system. It most frequently occurs when there is a hardware problem with a disk drive.

Note:

1. LIB74 creates intermediate files when it is running. Disk space equivalent to approximately two or three times the size of the library file is required.

Table A.2: List of library manager errors

Error message	Description and user action
xxx is multiple defined in xxx. others in xxx	Public label or public symbol is defined more than once. ⇒ Use a LST file to check the public labels and symbols in the library file
xxx module is not in the library	The module specified cannot be found. ⇒ Use a LST file to check the modules in the library file
Invalid module or library	The module in the library file is in a different format from the relocatable file. ⇒ The library file to be edited and the relocatable file must be created with the same version of SRA74.
xxx command file not found	The command file specified cannot be found. ⇒ Check the command file specification
Out of heap space	The memory space required for the operation of the library manager is insufficient. ⇒ Reduce the number of global labels
Too many object modules	There are too many modules in the library file. ⇒ Split up the library file into two or more files. Each library file can hold up to 500 modules.
CPU number error	The specified library file or relocatable file has not been created by SRA74. ⇒ Check the library file or relocatable file specification

PART 4

CROSS REFERENCER FOR THE 740 FAMILY

CRF74 OPERATION MANUAL

RENESAS
Renesas Technology Corp.

Table of Contents

List of Figures

List of Tables

CHAPTER 1
The format of the Manual

The CRF74 operation manual contains the following chapters:-

- Chapter 2: Outline
  The basic functions of CRF74 and the files which it generates.

- Chapter 3: Operating method
  How to enter the commands used by CRF74.

- Appendix A: Error messages
  A list of the error messages used by CRF74 along with clarifications of the content of the messages and appropriate user responses.

CHAPTER 2
Outline

CRF74 is a program for creating a cross reference list of labels and symbols in a source file. This list is very useful in helping you to understand the interrelationships between different parts of a source file when you are modifying a program.

2.1 Functions

CRF74 is used in conjunction with SRA74[1]. The following functions are provided to make the process of understanding a source file more efficient.

1. Line numbers are displayed for commands accessing labels.

2. File read can be executed by means of the pseudo-command .INCLUDE.

3. Headers can be output using the pseudo-command .PAGE.

2.2 Generating files

CRF74 generates a cross reference lists (CRF files)

- A label and symbol cross reference list is displayed.
- Format is fixed. Each page has 57 lines of 80 characters each.
- This file should be output to the printer and used in case of debugging or editing.
- The file attribute is .CRF.

[1] A relocatable assembler for use with the 740 Family.

2.3 Configuration of the CRF file

Fig 2.1 shows an example of an output CRF file. The CRF file contains the following data:-

1. Labels and symbols, together with the line numbers where they are defined and accessed. The definition line is marked with “#”, and any access lines involving subroutine calls are marked with “&”.

2. The first 32 characters of label and symbol names are shown. The page format is determined by the longest name in the CRF file.

3. The list header shows any title specified using the pseudo-command .PAGE. (Only the first 30 characters are used. Any further characters are disregarded.)

4. CRF74 performs no evaluation of values of labels and symbols in the source program. This means that it cannot make evaluations of condition assembler.

```
740 Family CROSS REFERENCE V.1.00.10                                P. 001

A0          3926    4285    8549    9079    9100
AA          3884    5545    5668
ABEND       9396&   9465&   9587#
ABEND10     9588    9593#
ABENDRT     9590#   9605
ACCHK       1201#   1408
ACCHK5      1213    1237#
ACCHKE      1235    1239    1241    1249#
ADDING      4994    5006#
ADDING0     5013    5014    5016#
ADDING1     5015    5019#
ADDRESS      300     318    1025
ADR_CHK     9154#
ADR_OUT     8302    8336    9145#
ADR_PNT     8157#
ADR_PNT2          8149#
```

Fig 2.1: CRF file output

CHAPTER 3
Operation Method

3.1 Activation method

Before using CRF74 the following data (input parameters) must be entered:-

1. Source file name (required item)

2. Command parameters

3.2 Input parameters

3.2.1 Source file name

1. Always enter the name of the library file.

2. If the .A74 attribute part of the filename is omitted, the attribute .A74 will be selected as the default.

3. If the full filename is specified, files with attributes other than .A74 may be processed (Example: .ASM).

4. The drive name can also be specified within the file name. If only the file name is specified then the system will search for the library file on the current drive.

5. Up to 16 source file names can be specified.

3.2.2 Command parameters

Command parameters are used to specify whether or not to search for the pseudo-command .INCLUDE in the source file and to specify the drive name for the output file. Table 3.1 gives details of the command parameters.

Table 3.1: List of command parameters

Command parameter	Description
-O	Specifies the drive name and directory path for the CRF file to be output. They should be specified in the following format. Example: A>CRF74 SRCFILE -OC:\TMP <RET> (The '¥' symbol and '\' symbol are interchangeable. This is because the use of one or other of these signs depends on which operating system is being used. Since the codes are the same, either sign can be used.) The CRF file is output to the TMP directory on drive C.
-I	Disregard the pseudo-command .INCLUDE.

3.3 Methods of entry

3.3.1 Command line input

Command line input has the following features.

With command line input, the whole command is entered on the operating system command line to activate CRF74.

```
A>CRF74 SRCFILE1 SRCFILE2 SRCFILE3<RET>
```

Fig 3.1: Command line input - Example

If an error is detected in the command line, a help screen like the one shown in fig 3.2 is displayed and processing stops.

```
A>CRF74<RET>
740 Family CROSS REFERENCE V.1.00.10
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Usage:  crf74 <filename> [-ifilename,..] [-opath]
   -i : not include specified files ( use -ifilename,.... )
   -o : select drive and directory for output ( use -otmp )
```

Fig 3.2: Help screen for command line errors

3.4 Errors

3.4.1 Types of error

The types of errors which occur during the operation of CRF74 have the following causes:-

1. Errors relating to the operating system

   These are errors such as insufficient disc or memory capacity which relate to CRF74's operating system. Please refer to the list of error messages in Appendix A and proceed according to the operating system commands.

2. Errors relating to the input of CRF74 command lines

   These are errors relating to the input of the command lines to activate CRF74. Please study the contents of this chapter and then re-enter the relevant commands.

3. Errors relating to the contents of the source files being processed

   This error occurs when a source file specified by the pseudo-command .INCLUDE cannot be found.

When CRF74 detects an error, a message is displayed in the format shown in Fig. 3.3. Refer to the list of error messages in Appendix A and take the appropriate action.

```
A>CRF74 SRCFILE1<RET>
740 Family CROSS REFERENCE V.1.00.10
Copyright 1989-1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

now processing pass 1
now making cross reference ( SRCFILE1.A74 )
----*
Out of heap space

A>
```

Fig 3.3: Error display

3.4.2 Value for return to operating system

When entering an execution command into an operating system DOS batch file, for example, there are times when you may wish to change the contents of the processing operation in accordance with the outcome of the execution. In CRF74 the execution results are divided into 3 error levels as shown in Table 3.2 and returned to the operating system. For an explanation of how to make best use of these error levels please refer to a commercially available MS-DOS handbook.

Table 3.2: List of error levels

Error level	Description of execution results
0	Normal termination
1	Error due to not being able to find source file specified by pseudo-command .INCLUDE
2	Error relating to input of CRF74 commands
3	Error relating to the operating system

3.5 Environment variables

CRF74 does not use operating system environment variables.

APPENDIX A
Error Messages

A.1 System errors

When a system error is detected during CRF74 operation, an error message is displayed on the VDU and processing is suspended. Table A.1 lists all the system errors.

Table A.1: List of system errors

Error message	Description and user action
Usage: crf74 <filename> [-ifilename] [-opath]	The command has been input wrongly. ⇒ Refer to the help screen and input the command again.
Can't open xxx	File not found. ⇒ Check the source file name and input the command again.
Can't create xxx	This file cannot be generated. ⇒ Create more space on the disk.
Out of disk space	There is not enough space left on the disk for output. ⇒ Create more space on the disk.
Out of heap space	There is not enough available memory for the cross referencer to work properly.[1] ⇒ Reduce the number of symbols or labels.

Note:

1. The total number of symbols and labels which can be handled by CRF74 depends on the available memory capacity.

A.2 Cross reference errors

If an error is detected during cross reference creation, an error message is displayed on the VDU, but processing still continues. Table A.2 lists all the cross reference errors.

Table A.2: List of cross reference errors

Error message	Description and user action
Can't open include file xxx	The source file specified by pseudo-command .INCLUDE cannot be found. ⇒ Check the contents of the directory.

PART 5

M37280 CONVERTER FOR THE 740 FAMILY

CV74 OPERATION MANUAL

RENESAS
Renesas Technology Corp.

Table of Contents

List of figures

List of Tables

CHAPTER 1
The format of the Manual

The CV74 operation manual contains the following chapters:-

- Chapter 2: Outline
  The basic functions of CV74 and the files which it generates.

- Chapter 3: Operating method
  How to enter the commands used by CV74.

- Appendix A: Error messages
  A list of the error messages used by CV74 along with clarifications of the content of the messages and appropriate user responses.

CHAPTER 2
Outline

CV74 converts HEX files and symbol files generated with the assembler system (SRA74 V.3.00 and later) for the M37280 into a data format usable with a debugger. Use it together with a debugger to debug programs with address area data exceeding the 64 Kbytes of the M37280.
Before using CV74, generate a HEX file and symbol file with LINK74.

2.1 Functions

CV74 can convert HEX files and symbol files generated with the assembler system (SRA74 V.3.00 and later) for the M37280 into a data format usable with a debugger.

1. HEX files are divided into those whose address area data exceeds 64 Kbytes and that which is 64 Kbytes or less.

2. Symbol files are converted into a usable format for debuggers.

Note:
Proper debugging is not possible if the HEX files and symbol files are used before conversion with CV74.

2.2 Generating files

CV74 generates 3 file types as outlined below:-

1. HEX files storing address area data of 64 Kbytes or less

    - HEX files which can be debugged with a debugger
    - Files storing address area data of 64 Kbytes or less, sampled from HEX files generated by LINK74
    - The extension is .HEX.

2. HEX files storing address area data over 64 Kbytes

    - Files storing address area data over 64 Kbytes, sampled from HEX files generated by LINK74
    - The extension is .HXH.

3. Symbol files debuggable with a debugger

    - The extension is .SYM.

CHAPTER 3
Operation Method

3.1 Activation method

To start up CV74, it is necessary to input the following information (input parameters).

1. Name of file to convert
2. Command parameters

3.2 Input parameters

3.2.1 Name of file to convert

1. Be sure to input the name of the file to convert.
2. Prepare the target HEX files and symbol files in the same directory.
3. It is not possible to specify multiple file names.
4. File names cannot be specified with "." (period). The same goes for relative directories.

3.2.2 Command parameters

The command parameters specify the output file name or file to convert. They are described in Table 3.1.
Both HEX and symbol files are converted as long as either -H or -S is not specified.

Table 3.1: List of command parameters

Command parameter	Description
-O	Specifies the name of the output file. Be sure to input this option.
-H	Converts only HEX files. This command parameter cannot be specified at the same time as the -S option.
-S	Converts only symbol files. This command parameter cannot be specified at the same time as the -H option.

Note
To debug with a debugger, download the HEX and symbol files generated with CV74.

3.3 Methods of entry

With command line input, the whole command is entered on the operating system command line to activate CV74.

Figure 3.1 gives an example wherein a HEX file (master.hex) and symbol file (master.sym) generated with LINK74 are converted into a HEX file (change.hex) and symbol file (change.sym) of a format usable with a debugger.

```
A>CV74 master –Ochange <RET>
HEX, SYM file converter for 7200 Series V.1.00.00
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

== HEX file convert now == ( master.hex --> change.hex )
== SYM file convert now == ( master.sym --> change.sym )
```

Fig 3.1: Help screen for command line errors

3.4 Errors

3.4.1 Types of error

The types of errors which occur during the operation of CV74 have the following causes:-

1. Errors relating to the operating system

   These are errors such as insufficient disc or memory capacity which relate to CV74's operating system. Please refer to the list of error messages in Appendix A and proceed according to the operating system commands.

2. Errors relating to the input of CV74 command lines

   These are errors relating to the input of the command lines to activate CV74. Please study the contents of this chapter and then re-enter the relevant commands.

3. Errors relating to the contents of the source files being processed

   An error has occurred with the contents of the HEX and symbol files to be converted. Check the contents and reinput as necessary.

```
A> CV74 <RET>
HEX, SYM file converter for 7200 Series V.1.00.00
Copyright 1998, MITSUBISHI ELECTRIC CORPORATION
AND MITSUBISHI ELECTRIC SEMICONDUCTOR SYSTEMS CORPORATION
All Rights Reserved.

Error : No input files specified

Usage: cv74 Input-filename -Ooutput-filename [-S or -H]<cr>
```

Fig 3.2: Error display

If an error occurs while CV74 is running, an error message is displayed on the monitor. Remedy the error as explained in "Appendix A Error Messages".

3.4.2 Value for return to operating system

When entering an execution command into an operating system DOS batch file, for example, there are times when you may wish to change the contents of the processing operation in accordance with the outcome of the execution. In CV74 the execution results are divided into 5 error levels as shown in Table 3.2 and returned to the operating system. For an explanation of how to make best use of these error levels please refer to a commercially available OS handbook.

Table 3.2: List of error levels

Error level	Description of execution results
0	Normal termination
1	Forced termination caused by ^C (cntl-C).
2	Error relating to the operating system
3	Error relating to content of source file the which constitutes the object of the operation.
4	Error relating to input of CV74 commands

3.5 Environment variables

CV74 does not use operating system environment variables.

APPENDIX A
Error Messages

A.1 List of errors

Table A.1: List of errors

Error message	Description and user action
'-H' and '-S' are used	Both the -H and -S options were specified at the same time. ⇒ Specify one or the other.
. is specified in the filename	The specified file name contains a period (.). ⇒ Check the file name.
Can't create 'xx' file	The "xx" file cannot be generated. ⇒ Check directory capacity.
Can't create temporary file	A temporary file could not be generated. ⇒ Specify a directory in the environment variable TMP, so as to create a temporary file in a directory other than the current one.
Can't open 'xx' file	The "xx" file cannot be opened. ⇒ Check the file name.
Command line is too long	There are too many characters in the command line. ⇒ Reinput the command line.
File format error	The input file format is illegal. ⇒ Check the file format.
Hex files number exceed 1	Multiple target files have been specified. ⇒ Specify only 1 target file.
Invalid option 'xx' is used	Invalid command option "xx" was used. ⇒ The specified option does not exist. Reinput the command.
No input files specified	An input file was not specified. ⇒ Specify an input file.
No output files specified	An output file was not specified. ⇒ Specify an output file.
Not enough memory	There is not enough memory. ⇒ Divide up the file and reexecute the command, or increase memory.
Option 'xx' is multiple defined	Command option xx is defined twice. ⇒ Reinput the command.
Option 'xx' is not appropriate	The xx command option is not properly written. ⇒ Respecify the command option.

A.2 List of warning

Table A.2: List of warning

Error message	Description and user action
Output data to the 'xx' file doesn't exitst	The "xx" file cannot be generated because the specified data area is empty. ⇒ Check the contents of the input file.

****

Branch Instruction Optimization Tool LOOP74

Operation Guide

****

1. Function

This program converts branch instructions that generate errors because the addresses are out of range into enabled branch instructions.

If you will write all branch instructions with the shortest branch instructions and use loop74, you will be able to generate programs written with optimized branch instructions.

2. Generated files

- When the –ASM command parameter is specified

  LOOP74 outputs conversion results to the xxx.i file generated by SRA74. The xxx.i file is output to the same directory as the input file.

- When the –ASM command parameter is not specified

  LOOP74 outputs conversion results to the structured source file specified by LOOP74.

Note:) Source level debugging using a debugger is not possible with files converted with –ASM command.

3. Startup

3.1 Input parameters

To run LOOP74, it is necessary to input the below information (input parameters).

- Source file name

This name specifies the source file to be converted. Only one file can be specified. Input the extension with the file name. The only applicable extension is .A74. The directory path can be specified in the file name. When only the file name is specified, the file in the current directory of the current drive is processed.

LOOP74 command options

Option	Description
–.	Prevents the state of execution from being output to the monitor. (Not applied to error messages)
–ASM	Converts branch instructions developed from structured commands to the assembly commands. (When using this option, it becomes not possible to do source level debugging.)
–V	Outputs LOOP74 version and ends processing.
–COUNTn	Ends conversion after reaching a specified number n (decimal number) of conversion cycles. Specify n between 1 and 100. The default is 100.
–COMMENT	Attaches a comment indicating LOOP74 conversion has been performed to the converted line.

Capital and small case letters are distinguished in the above options, therefore commands must be specified exactly as indicated above in capital letters.

- SRA74 command options

These command options are specified when LOOP74 starts up SRA74. All command options which LOOP74 does not support itself are interpreted to be SRA74 command options and handed over to SRA74.

3.2 Startup

LOOP74 starts up with the below line.

```
A> LOOP74 SAMPLE.A74 –COMMENT -S -C -L<RET>
```

The above line specifies SAMPLE.A74 as the assembler source file to convert, specifies that LOOP74 shall attach a comment to the converted line and specifies -S, -C and -L as command options to be handed over to SRA74.
The same processing is achieved with the following line.

```
A> LOOP74 -L SAMPLE.A74 -S -COMMENT -C <RET>
```

In other words, there is not set sequence for specifying command parameters with LOOP74.

Note:) LOOP74 automatically generates tag files, therefore it is not necessary to write the -E command option of SRA74. In any case, problems are not created if the command option is written.

4. Input source file format

LOOP74 processes source files of the same format as the input source files of SRA74. Hence, source files which cannot be processed by SRA74 cannot be processed by LOOP74 either. LOOP74 also has the following restrictions.

- LOOP74 does not support labels with .EXT declarators, etc.
- LOOP74 does not support sources without a ":" in the label.
- LOOP74 does not support elements unprocessible by SRA74 such as the "FALSE" condition of if pseudo-commands.
- A warning is generated if the below relative addresses are written in the branch instruction operand. LOOP74 determines an address is relative if * is written in the branch instruction operand.

Example

```
BRA * + 5
BRA * + label
```

5. Output file format

LOOP74 outputs the branch instruction lines to be converted in the below format. Normally, LOOP74 deletes the instruction line prior to conversion, but the original instruction line can be kept as a comment by specifying the -COMMENT command option.

- When the -COMMENT command parameter is not specified (Default)

Line to convert [1]

```
BEQ     L1
```

LOOP74 conversion results

```
BNE   ..lop1    ; This is the line which loop74 generated
JMP   L1        ; This is the line which loop74 generated
..lop1:
```

Line to convert [2]

```
        BEQ     L1      ;goto L1
```

LOOP74 conversion results

```
        BNE   ..lop1    ; This is the line which loop74 generated
        JMP    L1       ;goto L1
        ..lop1:
```

With the above line format, a comment indicating that LOOP74 generated the line is attached to the branch instruction line that was converted. If a comment is written in the target line, LOOP74 outputs the comment written for the line of the last branch instruction output.

- When the -COMMENT command parameter is specified

Line to convert [1]

```
          BEQ    L1
```

LOOP74 conversion results

```
        ;   BEQ    L1
          BNE   ..lop1   ; This is the line which loop74 generated
          JMP   L1       ; This is the line which loop74 generated
        ..lop1:
```

Line to convert [2]

```
          BEQ    L1       ;goto L1
```

LOOP74 conversion results

```
        ;   BEQ     L1      ;goto L1
          BNE   ..lop1   ; This is the line which loop74 generated
          JMP   L1   ; This is the line which loop74 generated
        ..lop1:
```

With the above line format, the converted line is output as a comment because the -COMMENT command option was specified. If a comment is written in the target line, LOOP74 does not output the comment written for the line of the last branch instruction output.

6. Conversion rules

LOOP74 converts branch instructions according to the below rules.

input line	output line
BEQ L1	BNE ..lop1 JMP L1 ..lop1:
BNE L1	BEQ ..lop1 JMP L1 ..lop1:
BCC L1	BCS ..lop1 JMP L1 ..lop1:
BCS L1	BCC ..lop1 JMP L1 ..lop1:
BMI L1	BPL ..lop1 JMP L1 ..lop1:
BPL L1	BMI ..lop1 JMP L1 ..lop1:
BVC L1	BVS ..lop1 JMP L1 ..lop1:
BVS L1	BVC ..lop1 JMP L1 ..lop1:
BBC BIT,MEM,L1	BBS BIT,MEM,..lop1 JMP L1 ..lop1:
BBS BIT,MEM,L1	BBC BIT,MEM,..lop1 JMP L1 ..lop1:
BRA L1	JMP L1

Labels generated by LOOP74 are output with a format from ..lop1 to ..lop65535.

Also, if multiple conditional branch instructions which branch to the same jump destination are written consecutively, conversion takes place as follows.

input line	output line
Bcnd_1 L1 Bcnd_2 L1 : : Bcnd_n L1	Bcnd_1 ..lop1 Bcnd_2 ..lop1 : : BcndR_n ..lop2 ..lop1: JMP L1 ..lop2:

Bcnd_1, Bcnd_2 and Bcnd_n represent the below conditional branch instructions.

BEQ, BNE, BCC, BCS, BMI, BPL, BVC, BVS

BcndR_n represents the conditional branch instruction which returns to the Bcnd_n branch instruction.

If multiple conditional branch instructions which branch to the same jump destination are written consecutively, only the last instruction written is returned and developed.

A specific example is given below.

input line	output line
BEQ L1 BCS L1	BEQ ..lop1 BCC ..lop2 ..lop1: JMP L1 ..lop2:

7. Errors

If an error occurs while LOOP74 is running, an error message is displayed on the monitor and LOOP74 stops. An error list is given below.

Error message	Description/Remedial action
Can't create file 'filename'	The "file name" file cannot be generated. Check disk capacity.
Can't create Temporary file	A temporary file cannot be generated. Check if writing is disabled for the directory.
Can't open file 'filename'	The "file name" file cannot be opened.

	Check the specified file name.
Can't translate	The address of even the biggest branch instruction cannot be received. Check the content of the source program.
Command line is too long	There are too many characters in the command line. LOOP74 can take command lines up to a maximum 255 bytes. Reduce the number of characters.
Error occurred in executing 'sra74'	An error occurred while running SRA74. Reexecute SRA74.
Illegal source file	LOOP74 cannot analyze the source file format. Specify a source file of the correct format.
Illegal TAG file	LOOP74 cannot analyze the tag file format. Specify a tag file of the correct format.
No input files specified	An input file was not specified. Specify an input file.
Not enough memory	There is not enough memory. Divide up the input file or increase memory.
Option 'xx' is not appropriate	The xx command option is not properly written. Respecify the command option.
Source files number exceed 1	Multiple source files have been specified. Specify only 1 source file.
Too many branch error	There are too many branch instructions to convert. Divide up the source file and reduce the number of branch instructions (for LOOP74) to process per file.

8. Warning

If warning is detected while LOOP74 is running, a warning message will be displayed on the monitor. LOOP74 processing will continue. Warning messages are given below.

Warning message	Description/Remedial action
Relative address is specified	A relative address is written in the branch instruction operand. This line is not converted.

9. Values returned to the OS

When processing ends, LOOP74 returns the below values to the OS.

Returned value	Meaning
0	Ended successfully
1	Error in source file or tag file read by LOOP74
2	Error in command line input
3	Operating system error
4	Ended by ^C (control C) input

10. Environmental variables

LOOP74 does not use environmental variables.

MEMO

M3T-SRA74 V.4.10 User's Manual

Rev. 1.00
August 01, 2003
REJ10J0155-0100Z

M3T-SRA74 V.4.10
User's Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ10J0155-0100Z