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.

Note that the following URLs in this document are not available:
http://www.necel.com/
http://www2.renesas.com/

Please refer to the following instead:
Development Tools | http://www.renesas.com/tools
Download | http://www.renesas.com/tool_download

For any inquiries or feedback, please contact your region.
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.

User's Manual

CA850 Ver. 3.20

C Compiler Package

Assembly Language

Target Device
V850 Series

Document No. U18514EJ1V0UM00 (1st edition)
Date Published May 2007 CP(K)

Printed in Japan

[MEMO]

- **The information in this document is current as of May, 2007. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC 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 NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".
  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.

"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.
"Special":	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, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)

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

(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

M8E 02. 11-1

[MEMO]

INTRODUCTION

Target devices

The V850 Series C compiler package generates object code for the NEC Electronics's V850 Series RISC microcontrollers.
This manual explains the CA850 C compiler package.

Readers

This manual is intended for user engineers who wish to develop an application system using the V850 Series C compiler package.

Purpose

This manual explains the assembly language specifications supported by the assembler (as850) included in the CA850 C compiler package.

Organization

This manual contains the following information:

- OVERVIEW
- ASSEMBLY LANGUAGE SPECIFICATIONS
- INSTRUCTION SET
- THE INSTRUCTION OF THE ASSENBLY LANGUAGE
- QUASI DIRECTIVES

Note on reading this manual

- Each program name of the C compiler package is described in this manual as follows:

C compiler package	→	CA850
Assembler	→	as850
C compiler	→	ca850

- The functions and features specific to the V850E in the V850 Series are identified in the title or by [V850E], whereas the functions and features specific to the V850E2 are identified in the title or by [V850E2].

Related Documents Read this manual together with the following documents.
The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Documents related to development tools (user's manuals)

Document Name		Document No.
CA850 Ver. 3.20 C Compiler Package	Operation	U18512E
	C Language	U18513E
	Assembly Language	This manual
	Link Directives	U18515E
PM+ Ver. 6.30 Project Manager		U18416E
ID850 Ver. 3.00 Integrated Debugger	Operation	U17358E
ID850NW Ver. 3.10 Integrated Debugger	Operation	U17369E
ID850QB Ver. 3.20 Integrated Debugger	Operation	U17964E
SM+ System Simulator	Operation	U17246E
	User Open Interface	U18212E
SM850 Ver. 2.50 System Simulator	Operation	U16218E
SM850 Ver. 2.00 or Later System Simulator	External Part User Open Interface Specifications	U14873E
RX850 Ver. 3.20 or Later Real-Time OS	Basics	U13430E
	Installation	U17419E
	Technical	U13431E
	Task Debugger	U17420E
RX850 Pro Ver. 3.21 Real-Time OS	Basics	U18165E
	Internal Structure	U18164E
	Task Debugger	U17422E
RX850V4 Ver. 4.22 Real-Time OS	Functionalities	U16643E
	Internal Structure	U16644E
	Task Debugger	U16811E
AZ850 Ver. 3.30 System Performance Analyzer		U17423E
AZ850V4 Ver. 4.10 System Performance Analyzer		U17093E
TW850 Ver. 2.00 Performance Analysis Tuning Tool		U17241E

[MEMO]

CONTENTS

LIST OF FIGURES

Figure No. Title Page

LIST OF TABLES

Table No. Title Page

CHAPTER 1 ASSEMBLY LANGUAGE SPECIFICATIONS

This chapter explains the assembly language specifications supported by the CA850 assembler (as850).

1.1 Organization of Assembly Language Statements

An assembly language statement consists of a "label", a "mnemonic", "operands", and a "comment".

```
[label]: [mnemonic] [operand], [operand] -- [comment]
```

It is irrelevant whether blanks are used or not in the following cases (1) to (4).

(1) Between the label name and colon

(2) Between the colon and mnemonic

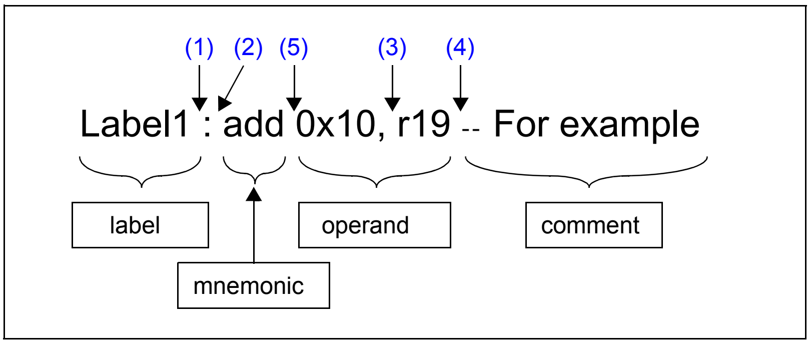
(3) Before the second and subsequent operands

(4) Before "--" that indicates the beginning of a comment

One or more blank is necessary in the following case.

(5) Between the mnemonic and the first operand

Figure 1 - 1 Organization of Assembly Language Statement



Basically, one assembly language statement is described on one line, with a line feed (return) at the end of the statement. Two or more statements can be described on one line by using ";" (semicolon).

1.1.1 Label

A label is a "name plate" that can be described on any line of a program.

A label can be used as the name of a branch destination if a conditional branch is executed or if execution branches to a subroutine.

For example, when the "jr" instruction, one of the branch instructions, is used, describe a label as follows.

```
        jr  Label1
```

When this instruction is executed, execution branches to the location of Label1.

When a label is described as name Label1, describe as follows.

```
Label1 :
```

Different labels can be defined over several lines.

```
Label1 :
Label2 :
```

However, two or more labels must not be specified on one line.

```
Label1: Label2: -- Two or more labels must not be specified on one line.
```

It is irrelevant whether blanks are inserted between the label name and colon.

Before using a label, a "definition" or "declaration" must be made. For how to make a definition or declaration, refer to "1.2.2 Label".

1.1.2 Mnemonic and operands

A mnemonic is a character string assigned to each instruction (V850 machine code).

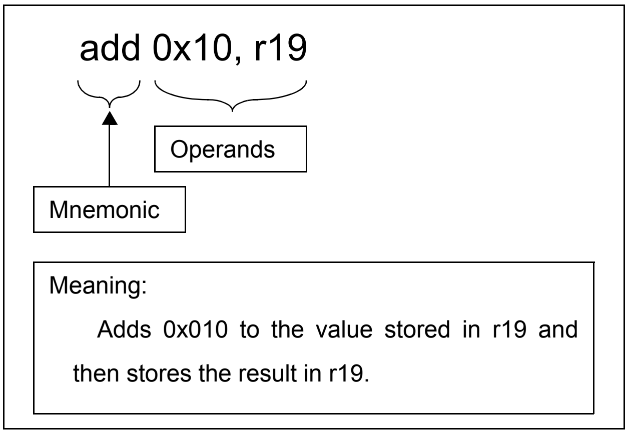
Machine codes are hard for human beings to understand as is. Therefore, a name is assigned to each machine code. This name is a "mnemonic". A mnemonic means the instruction itself. A mnemonic is expressed in close to word notation (based on English) so that the operation it stands for can be easily inferred.

For example, the mnemonic "add" means "addition", and "mul" means "multiplication".

An operand is an object to be manipulated by each instruction. If the mnemonic is "add" (addition), the operand is subject to the operation of addition. An operand must be described next to (on the right of) a mnemonic.

One or more blank is necessary between the mnemonic and the first operand.

Figure 1 - 2 Mnemonic and Operands



```
add 0x10, r19
```

Meaning:

Adds 0x010 to the value stored in r19 and then stores the result in r19.

An assembly instruction consists of a "mnemonic" and "operand(s)". The number of operands differs from one mnemonic to another.

For the list of the assembly instructions provided in the V850 microcontrollers and their specifications, refer to "CHAPTER 3 ASSEMBLY LANGUAGE INSTRUCTIONS".

1.1.3 Comment

Comments can be described in an assembly language program.

The as850 recognizes the description after the following marks to the end of the line as a comment.

```
--
```

```
#
```

In the case of "#", however, the statement to the end of the line is recognized as a comment only if "#" is at the beginning of the statement[Note].

```
# comment
    add 0x10, r19        -- comment-1
    sub r18, r19         -- comment-2
```

Note The blank at the beginning of the line is not regarded as a part of it. When there is a blank in front of the "#", the comment is composed of the characters from the "#" to the end of the statement.

1.1.4 Character set

The character set that can be used in a source program (assembly language) supported by the as850, and the use for the characters are as follows.

Table 1 - 1 Character Set and Usage of Characters

Character	Usage
Lowercase letter (a-z)	Constitutes a mnemonic, identifier, and constant
Uppercase letter (A-Z)	Constitutes an identifier and constant
_ (underscore)	Constitutes an identifier
. (period)	Constitutes an identifier and constant
Numeral	Constitutes an identifier and constant
: (colon)	End of label
, (comma)	Delimits an operand
- (hyphen)	Indicates a negative sign, subtraction operator, and the beginning of a comment
#	References the absolute address of a label and indicates the beginning of a comment
; (semicolon)	End of statement
' (single quotate)	Start and end of character constant
" (double quotate)	Start and end of string constant
$	References the gp offset of label
[ ]	Specifies the base register
+	Addition operator
*	Multiplication operator
/	Division operator
%	Offset reference of label in section (without instruction expansion) or remainder operator
<<	Left shift operator
>>	Right shift operator
!	Absolute address reference of label (without instruction expansion) or negation operator
&	Logical product operator
|	Logical sum operator
^	Exclusive OR operator
( )	Specifies an operation sequence

1.1.5 Example of assembly language statement

Here is a simple example of an assembly language program.

```
# sample program
    .extern __tp_TEXT, 4
    .extern __gp_DATA, 4
    .extern _main
    .section "RESET", text      -- Reset Handler address
    jr      __boot              -- Jump to __boot
    .text                       -- Text section
    .align 4                    -- Code alignment
    .globl __boot               -- Alignment
__boot:
    mov     #__tp_TEXT, tp      -- Set tp
    mov     #__gp_DATA, gp      -- Set gp

    .extern __ssbss, 4
    .extern __esbss, 4

# start of bss initialize
    mov     #__ssbss, r13
    mov     #__esbss, r13
    cmp     r12, r13
    jnl     sbss_init_end
sbss_init_loop:
    st.w    r0, 0[r13]add 4, r13
    cmp     r12, r13
    jl      sbss_init_loop
sbss_init_end:
# end of bss initialize

    jarl    _main, lp       -- Call main function
    .data
    .align 4
data_area:
    .word 0x00              -- data1
    .hword 0x01             -- data2
    .byte 0xff ; .byte 0xfe -- data3, data4
```

1.2 Organization of Assembly Language Program

1.2.1 Symbol

A symbol is a name having a value (integer value) which is defined by the user.

The ".set" quasi directive is used to define a symbol.

```
     .set     sym1, 0x10       -- sym1 is a symbol having the value 0x10.
     mov      sym1, r10        -- Stores the value(0x10) of sym1 in a register.
```

The as850 assumes a reference to a symbol appearing between the beginning of a file and the first .set quasi directive as a "reference to a symbol undefined at that point", and distinguishes this symbol from a reference to a defined symbol (also refer to "(1) Absolute expression" in "1.2.6 Expressions").

(1) Characters usable in symbol

The following characters shown in "1.1.4 Character set" can be used in symbols.

- Lowercase letters
- Uppercase letters
- _ (underscore)
- . (period)
- Numerals

However, a numeral cannot be used at the beginning of a symbol. If a symbol that begins with a numeral is specified, the as850 outputs the following message and stops assembling.

```
E3249: illegal syntax
```

Moreover, the reserved words shown in "1.2.4 Reserved words" cannot be used in symbols.

Caution Note that a symbol starting with "_" (underscore) may match a symbol name output by the compiler, and may therefore cause an unexpected operation. Also, avoid using symbols that start with "." (period) as much as possible because such symbols may be reserved in the future.

(2) Maximum number of characters of symbol and maximum number of symbols

A symbol consists of up to 1,037 characters. If a symbol of 1,038 or more characters is specified, the as850 outputs the following message and stops assembling.

```
E3260: token too long
```

The maximum number of symbols that can be defined depends on the size of the available memory area.

1.2.2 Label

A label is a name that can be described on any line of a program and is defined by the user.

A label is defined or declared as follows.

(1) Defining label

A label may be defined in two ways.

(a) Defined as local label when ":" is suffixed to a name at the beginning of a statement

```
label1:
```

This method is generally used to define a local label, and is hereafter referred to as "normal label definition".

(b) Defined as local label by the .lcomm quasi directive

```
    .lcomm label1, 0x100, 4
```

The above statement means 'allocates size of "0x100 bytes" from an address aligned to 4 bytes and uses the first label of that area as "label1"'.

(2) Declaring label

A label may be declared in four ways.

(a) Declared as an undefined external label by the .comm quasi directive

```
    .comm label1, 4, 4
```

This statement means 'undefined external label "label1" of size "4 bytes" is declared in an alignment condition of 4 bytes'.

(b) Declared as an external label by the .extern quasi directive (label not having a definition in a specified file)

```
    .extern label1
```

(c) Declared as an external label by the .globl quasi directive (label having a definition in a specified file)

```
    .globl label1
```

(d) Declared as an external label by not making a definition in a file

```
    .mov label1, r10
```

If the definition of label1 is not in the same file, label1 is regarded as an external label.

(3) Characters that may be used in labels

The following characters shown in "1.1.4 Character set" can be used in labels.

- Lowercase letters
- Uppercase letters
- _ (underscore)
- . (period)
- Numerals

However, a numeral cannot be used at the beginning of a label. If a label that begins with a numeral is specified, the as850 outputs the following message and stops assembling.

```
E3249: illegal syntax
```

Moreover, the reserved words shown in "1.2.4 Reserved words" cannot be used in symbols.

Caution Note that a symbol starting with "_" (underscore) may match a symbol name output by the compiler, and may therefore cause an unexpected operation. Also, avoid using symbols that start with "." (period) as much as possible because such symbols may be reserved in the future.

(4) Maximum number of characters of label and maximum number of labels

A label consists of up to 1,037 characters. If a label of 1,038 or more characters is specified, the as850 outputs the following message and stops assembling.

```
E3260: token too long
```

The maximum number of labels that can be defined depends on the size of the available memory area.

(5) Normal label definition in sbss/bss-attribute section

If a normal label definition is made in the sbss/bss-attribute section, the as850 outputs the following message and stops assembling.

```
E3246: illegal section
```

If this error is output, use the .lcomm quasi directive to define a label.

1.2.3 Macro

A macro is described by registering a pattern with a set sequence and by using this pattern.

A macro is defined by the user. A macro is defined as follows.

```
.macro PUSH REG     -- The following two statements constitute the macro body.
    add     -4, sp
    st,w    REG, 0x0[sp]
.endm
```

The macro body is enclosed by ".macro" and ".endm". If the following description is made after the above definition has been made, the macro is replaced by a code that "stores r19 in the stack".

```
    PUSH    r19
```

In other words, the macro is expanded into the following codes.

```
    add     -4, sp
    st,w    r19, 0x0[sp]
```

1.2.4 Reserved words

The as850 has reserved words. Reserved words cannot be used in symbols, labels, and section names.

If a reserved word is specified, the as850 outputs the following message and stops assembling.

```
E3245:identifier is reserved word
```

The reserved words are as follows.

- Instructions (such as add, sub, and mov)
- QUASI DIRECTIVES(such as .section, .lcomm, and .globl)
- hi, lo, hi1 (because they are used as hi(), lo(), and hi1(). Refer to "2.2.7 hi()/lo()/hi1()".)
- Register names

1.2.5 Constants

The as850 can handle "Numerical constants", "Character constant", and "String constant" as constants.

(1) Numerical constants

Numerical constants are divided into "Integer constants" and "Floating-point constant".

(a) Integer constants

An integer constant has a width of 32 bits. A negative value is expressed as a 2's complement. If an integer value that exceeds the range of the values that can be expressed by 32 bits is specified, the as850 uses the value of the lower 32 bits of that integer value and continues processing (it does not output any message).

(i) Binary constants

A binary constant consists of "0b" or "0B" followed by a numeric string of one or more "0" or "1" digits.

Example

```
0b00010110111101010111111010010111
```

(ii) Octal constant

An octal constant consists of "0" followed by a numeric string of one or more "0" to "7" digits.

Example

```
02675277227
```

(iii) Decimal constant

A decimal constant consists of one or more numerals starting with other than "0".

Example

```
385187479
```

(iv) Hexadecimal constant

A hexadecimal constant consists of "0x" or "0X" followed by a numeric string of one or more "0" to "9" digits, and a character string of "a" to "f" or "A" to "F".

Example

```
0x16f57e97
```

(b) Floating-point constant

A floating-point constant has a 32-bit width and consists of the following elements.

(i) Sign of mantissa ("+" can be omitted.)

(ii) Mantissa

(iii) "e" or "E" indicating exponent

(iv) Sign of exponent ("+" can be omitted.)

(v) Exponent

The exponent and mantissa are specified as decimal constants. If no exponent is used, however, (iii), (iv), and (v) are not used.

Example

```
123.4
-100.
10e-2
-100.2E+5
```

A floating-point constant can also be indicated by placing "0f" or "0F" at the beginning of a mantissa (for example, the as850 regards 10 as being an integer constant but 0f10 as being a floating-point constant).

A numeric string that starts with "0" and which has no decimal point, such as "060", must not be specified (only "0" can be specified).

(2) Character constant

A character constant consists of a single character enclosed by a pair of single quotation marks (' ') and indicates the value of the enclosed character[Note]. If any of the escape sequences listed in Table 1 - 2 is enclosed in single quotation marks, the as850 regards the sequence as being a single character.

Example

```
'a'
'\0'
'\012'
'\x0a'
```

Note If a character constant is specified, the as850 assumes that an integer having the value of that character constant is specified.

Table 1 - 2 Value and Meaning of Escape Sequences

Escape Sequence	Value	Meaning
\0	0x00	null character
\a	0x07	Alert
\b	0x08	Backspace
\f	0x0c	Form feed
\n	0x0a	New line
\r	0x0d	Carriage return
\t	0x09	Horizontal tab
\v	0x0b	Vertical tab
\\	0x5c	Backslash
\'	0x27	Single quotation mark
\"	0x22	Double quotation mark
\?	0x3f	Question mark
\ddd	0 - 0377	Octal number of up to 3 digits ($0 \le d \le 7$) [Note]
\xhh	0 - 0xff	Hexadecimal number of up to 2 digits ($0 \le h \le 9$, $a \le h \le f$, or $A \le h \le F$)

Note If a value exceeding "\377" is specified, the value of the escape sequence becomes the lower 1 byte. An octal number exceeding 0377 thus cannot be specified. For example, "\777" is assumed to be 0377.

(3) String constant

A string constant consists of a character string enclosed by a pair of double quotation marks ("") and indicates the enclosed string. If any of the escape sequences listed in Table 1 - 2 is enclosed in double quotation marks, the as850 regards the sequence as being a single character. If a numeral other than "0" to "7" is used as the escape sequence in "\ddd" format, the as850 regards the characters immediately before that numeral as an escape sequence of this format.

Example

"abc"	’a’, ’b’, ’c’
"ABC\n"	’A’, ’B’, ’C’, ’\n’
"\033abc\t\0"	’\033’, ’a’, ’b’, ’c’, ’\t’, ’\0’
"\12345"	’\123’, ’4’, ’5’
"\12845"	’\12’, ’8’, ’4’, ’5’

1.2.6 Expressions

An expression consists of a "constant", "symbol", "label reference", "operator", and "parentheses". It indicates a value consisting of these elements.

The as850 distinguishes between Absolute expression and Relative expressions.

(1) Absolute expression

An expression indicating a constant is called an "absolute expression".

An absolute expression can be used when an operand is specified for an instruction or when a value, size, alignment condition, filling value, or bit width is specified for a quasi directive.

An absolute expression usually consists of a constant or symbol (refer to "2.2.3 Symbols").

The as850 treats expressions in the format described below as absolute expressions.

However, an absolute expression in a format other than "constant expression" must not be specified for quasi directives other than the .byte, .hword, .shword **[V850E]**, and .word quasi directives without a bit width specification and quasi directives other than the .frame quasi directive (absolute expressions in all formats below can be specified for the .byte, .hword, .shword **[V850E]**, and .word quasi directives without a bit width specification to specify a value, while absolute expressions in "symbol" format can be specified for the .frame quasi directive to specify size, in addition to the "constant expression" format).

(a) Constant expression

Example

```
    .set     sym1, 0x100 -- Defines the symbol sym1.
    mov      sym1, r10   -- sym1, already defined, is treated as a constant expression.
```

If a reference to a previously defined symbol is specified, the as850 assumes that the constant of the value defined for the symbol has been specified.

Therefore, a defined symbol reference can be used in a constant expression.

(b) Symbol

The expressions related to symbols are the following ("±" is either "+" or "-").

- Symbol
- Symbol ± constant expression
- Symbol - symbol
- Symbol - symbol ± constant expression

A "symbol" here means an undefined symbol reference at that point. If a reference to a previously defined symbol is specified, the as850 assumes that the "constant" of the value defined for the symbol has been specified.

Example

```
    add      SYM1 + 0x100, r11    -- SYM1 is an undefined symbol at this point.
    .set     SYM1, 0x10           -- Defines SYM1.
```

(c) Label reference

The following expressions are used to reference a label ("±" is either "+" or "- ").

- Label reference - label reference
- Label reference - label reference ± constant expression

Here is an example of an expression related to a label reference.

Example

```
    mov $label1-$label2, r11
```

A "reference to two labels" as shown in this example must be referenced as follows.

- The same section has a definition in the specified file.
- Same reference method (such as $label and $label, and #label and #label)

If a reference to a label having no definition in the specified file is specified, the as850 outputs the following message and stops assembling.

```
E3209:illegal expression(labels must be defined)
```

If a reference to two labels having no definition in the same section is specified, the as850 outputs the following message and stops assembling.

```
E3209:illegal expression(labels in different sections)
```

If a reference to two labels by different reference methods is specified, the as850 outputs the following message and stops assembling.

```
E3209:illegal expression(labels have different reference types)
```

However, if a reference to the absolute address of a label not having a definition in the specified file is specified as label reference on one side of "- label reference" in an "expression related to label reference", it is assumed that the same reference method as that of the label on the other side is used, because of the current organization of the assembler.

Note that an absolute expression in this format cannot be specified for a branch instruction. If such an expression is specified, the as850 outputs the following message and stops assembling.

```
E3221:illegal operand(label-label)
```

(2) Relative expressions

An expression indicating an offset from a specific address[Note 1] is called a "relative expression".

A relative expression is used to specify an operand by an instruction or to specify a value by the .byte, .hword, or .word quasi directive.

A relative expression usually consists of a label reference (refer to "2.2.4 Label references").

The as850 regards expressions in the following formats[Note 2] as being relative expressions.

(a) Label reference

The following expressions are related to label reference ("±" is either "+" or "-").

- Label reference
- Label reference ± constant expression
- Label reference - symbol
- Label reference - symbol ± constant expression

Here is an example of an expression related to label reference.

Example

```
     add      #label1 + 0x10, r10
     add      #label2 - SIZE, r10
     .set     SIZE, 0x10
```

Notes 1 This address is determined when the linker (ld850) in the CA850 is executed. Therefore, the value of this offset may also be determined when the linker is executed.

2 The as850 can regard an expression in the format of "-symbol + label reference", for example, as being an expression in the format of "label reference - symbol," but it cannot regard an expression in the format of "label reference - (+symbol)" as being an expression in the format of "label reference - symbol" (the same applies to an absolute expression). Therefore, use parentheses only in constant expressions.

1.2.7 Operators

An operator can be used to specify the operation to be performed by an expression.

(1) Types of operators

Operators are classified into four types: "Arithmetic operators", "Shift operators", "Bitwise logical operators", and "Comparison operators".

"-" can be used as either a unary or binary operator.

Table 1 - 3 Operators

Type	Operator
Arithmetic operators	`+` `-` `*` `/` `%`
Shift operators	`<<` `>>`
Bitwise logical operators	`!` `|` `&` `^`
Comparison operators	`==` `<` `<=` `!=` `>` `>=` `&&` `||`

In the description below, the operand to the left of the operator is called the first operand, while the operand to the right of the operator is called the second operand. The operand for a unary operator is simply called an operand.

(a) Arithmetic operators

(i) +

Calculates the sum of the first and second operands.

(ii) -

Calculates the difference between the first and second operands.

If this operator is used as a unary operator, it calculates the 2's complement of the operand.

(iii) *

Calculates the product of the first and second operands.

(iv) /

Calculates the quotient of the first and second operands.

(v) %

Calculates the remainder resulting from dividing the first operand by the second operand.

(b) Shift operators

(i) <<

Shifts the first operand to the left by the number of bits specified by the second operand.

As many 0s as the specified number of bits are inserted on the right side (LSB[Note 1]) of the first operand.

Example

`0x12345678 << 4`	`0x23456780`

(ii) >>

Shifts the first operand to the right by the number of bits specified by the second operand. If the first operand is positive (MSB is 0), as many 0s as the specified number of bits are inserted on the left side of the first operand (MSB[Note 2]). If the first operand is negative (MSB is 1), as many 1s as the specified number of bits are inserted on the left side of the first operand.

Example

`0x12345678 >> 4`	`0x01234567`
`0x87654321 >> 4`	`0xF8765432`

Notes 1 LSB is an abbreviation of Least Significant Bit (bit corresponding to the lowest digit).

2 MSB is an abbreviation of Most Significant Bit (bit corresponding to the highest digit).

(c) Bitwise logical operators

(i) !

Logically negates each bit of the operand value.

Example

`!0x12345678`	`0xEDCBA987`

(ii) |

Calculates the logical sum of the first and second operands.

Example

`0x1234 | 0x5678`	`0x567C`

(iii) &

Calculates the logical product of the first and second operands.

Example

`0x1234 & 0x5678`	`0x1230`

(iv) ^

Calculates the exclusive OR of the first and second operands.

Example

`0x1234 ^ 0x5678`	`0x444C`

(d) Comparison operators

(i) ==

Compares the first operand with the second operand. If the two operands are equal, returns 1. Otherwise, returns 0.

Example

`1 == 1`	`1`
`1 == 0`	`0`

(ii) <

Compares the first and second operands. Returns 1 if the first operand is less than or equal to the second operand, and returns 0 if the first operand is greater than the second operand.

Example

`1 < 10`	1
`10 < 1`	0

(iii) <=

Compares the first and second operands. Returns 1 if the first operand is less than or equal to the second operand, and returns 0 if the first operand is greater than the second operand.

Example

`1 <= 1`	1
`1 <= 2`	1
`1 <= 0`	0

(iv) !=

Compares the first and second operands. Returns 0 if both the operands are equal, and returns 1 otherwise.

Example

`1 != 0`	1
`1 != 1`	0

(v) >

Compares the first and second operands. Returns 1 if the first operand is greater than the second operand, and returns 0 if the first operand is less than or equal to the second operand.

Example

`1 > 0`	1
`1 > 2`	0

(vi) >=

Compares the first and second operands. Returns 1 if the first operand is greater than or equal to the second operand, and returns 0 if the first operand is less than the second operand.

Example

`1 >= 1`	1
`1 >= 0`	1
`1 >= 2`	0

(vii) &&

Calculates the logical product of the logical value of the first and second operands.

Example

`1 != 3 && 1 <= 3`	1
`1 == 1 && 1 != 1`	0
`1 != 1 && 3 <= 1`	0

(viii) \|\|

Calculates the logical sum of the logical value of the first and second operands.

Example

`1 != 3 || 1 <= 3`	1
`1 == 1 || 1 != 1`	1
`1 != 1 || 3 <= 1`	0

(2) Priority of operators

Table below shows the priorities of the operators. If two operators having the same priority are specified, and if either is enclosed in parentheses, the operator in parentheses is executed first. If neither operator is enclosed in parentheses, or if both are enclosed in parentheses, the one on the left is executed first[Note].

Note However, use parentheses only for constant expressions (refer to "1.2.6 Expressions").

Table 1 - 4 Priority of Operators

Priority	Operator
High ↑ ↓ Low	`- ! (unary operator)`
	`* / << >> %`
	`& | ^`
	`+ -`
	`== < <= != > >=`
	`&& ||`

(3) Operation rules

The operation rules of the as850 are as follows[Note].

Note However, the rule explained in "1.2.6 Expressions" takes precedence for an expression including a reference to a symbol or label that has not yet been defined at that point.

(a) Unary operation

Only an absolute expression can be specified as the operand of a unary operator.

An expression that handles a floating-point value cannot be specified as the operand of the unary operator !.

(b) Binary operation

Table 1 - 5 lists the valid combinations of integer value expressions that can be specified as the operands of binary operators.

In this table, the following symbols are used in expressions consisting of operators and operands.

abs	Absolute expression
rel	Relative expression "referencing a label with a definition in the specified file"
ext	Relative expression "referencing a label with no definition in the specified file"
---	Indicates that the specified combination of the operator and operand is not supported by the as850

For floating-point values, however, the operation must be between floating-point values, and a floating-point value must not exist together with a relative expression in the same expression.

Table 1 - 5 Operation Rules for Binary Operation

Operand		Operator											
		+			-			*, /			Other		
Second operand		abs	rel	ext	abs	rel	ext	abs	rel	ext	abs	rel	ext
First operand	abs	abs	rel	ext	abs	---	---	abs	---	---	abs	---	---
	rel	rel	---	---	rel	abs[Note]	---	---	---	---	---	---	---
	ext	ext	---	---	ext	---	---	---	---	---	---	---	---

Note For details, refer to "1.2.6 Expressions".

1.3 Identifiers

An identifier is a name used for a symbol, label, or macro.

The following characters shown in "1.1.4 Character set" can be used in identifiers.

- Lowercase letters
- Uppercase letters
- _ (underscore)
- . (period)
- Numerals

However, a numeral must not be used at the beginning of a name.

Note that a symbol starting with "_" (underscore) may match a label name output by the compiler, and may therefore cause an unexpected operation. Also, avoid using identifiers that start with "." (period) as much as possible because such identifiers may be reserved in the future.

CHAPTER 2 INSTRUCTION SET

This chapter describes the instruction set supported by the CA850 assembler (as850).

2.1 Description of Symbols

Next table lists the meanings of the symbols used in this chapter and those that follow.

Table 2 - 1 Meanings of Symbols

Symbol	Meaning
reg, reg1, reg2 r0, R0 R1 gp ep [reg]	Register Zero register Assembler-reserved register (r1) Global pointer (r4) Element pointer (r30) Base register
disp	Displacement (32 bits unless otherwise stated)
imm	Immediate (32 bits unless otherwise stated)
bit#3	3-bit data for bit number specification
#label label $label !label %label	Absolute address reference of label Offset reference of label in section or PC offset reference For a section allocated to a segment for which a tp symbol is to be generated, however, offset from the tp symbol instead of offset reference in section gp offset reference of label Absolute address reference of label (without instruction expansion) Offset reference of label in section (without instruction expansion)
hi (value)	Higher 16 bits of *value*
lo (value)	Lower 16 bits of *value*
hi1 (value)	Higher 16 bits of *value* + value of bit 15 of *value* *value* : LSB(Least Significant Bit) is bit 0.
addr	Address
PC	Program counter
PSW	Program status word
regID	System register number (0 to 31)
vector	Trap vector (0 to 31)
BITIO	Peripheral I/O register (for 1-bit manipulation only)

2.2 Operand

This section describes the description formats of the operands of the as850. With the as850, registers, constants, symbols, label reference, reference of constants, symbols, and labels, operators (refer to "1.2.7 Operators"), and expressions enclosed in parentheses (refer to "1.2.6 Expressions") can be specified as the operands of instructions and quasi directives.

2.2.1 Registers

The registers that can be specified with the as850 are listed below[Note].

Note For the ldsr and stsr instructions, the PSW and system registers are specified using numbers. With the as850, PC cannot be specified as an operand

```
r0, zero, r1, r2, hp, r3, sp, r4, gp, r5, tp, r6, r7, r8, r9,
r10, r11, r12, r13, r14, r15, r16, r17, r18, r19,
r20, r21, r22, r23, r24, r25, r26, r27, r28, r29,
r30, ep, r31, lp
```

r0 and zero (zero register), r2 and hp (handler stack pointer), r3 and sp (stack pointer), r4 and gp (global pointer), r5 and tp (text pointer), r30 and ep (element pointer), and r31 and lp (link pointer) are the same registers, respectively.

(1) r0

r0 always has a value of 0. This register does not substitute the result of an operation even if used as a destination register. If r0 is specified as a destination register, the as850 outputs the following message[Note], then continues assembling.

Note Output of this message can be suppressed by specifying the warning suppression (-w) option upon starting the as850.

```
    mov 0x10, r0
```

```
W3013: register r0 used as destination register
```

(a) If r0 is specified in any of the following instructions as a destination register when the V850Ex is used as the target device, the as850 outputs an error message, not a warning message.

`dispose`,Syntaxes (1) and (2) in `divh` instruction,`ld.bu`,
`ld.hu`, Syntax (2) in `mov` instruction, `movea`, `movhi`,
`mulh`, `mulhi`, `satadd`, `satsub`, `satsubi`, `satsubr`,
`sld.bu`, `sld.hu`

```
     divh    r10, r0
```

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

(b) If r0 is specified in any of the following instructions as a source register when the V850Ex is used as the target device, the as850 outputs an error message, not a warning message.

Syntaxes (1) in `divh` instruction, `switch`

(2) r1

The assembler-reserved register (r1) is used as a temporary register when instruction expansion is performed using the as850. If r1 is specified as a source or destination register, the as850 outputs the following message[Note], then continues assembling.

Note Output of this message can be suppressed by specifying the warning suppression (-w) option upon starting the as850.

```
     mov 0x10, r1
```

```
W3013: register r1 used as destination register
```

```
     mov r1, r10
```

```
W3013: register r1 used as source register
```

2.2.2 Constants

As the constituents of the absolute expressions or relative expressions that can be used to specify the operands of the instructions and quasi directives in the as850, integer constants and character constants can be used.

For the ld/st and bit manipulation instructions, a peripheral I/O register name, defined in the device file, can also be specified as an operand, thus enabling input/output of a port address.

Moreover, floating-point constants can be used to specify the operand of the .float quasi directive, and string constants can be used to specify the operand of the .str quasi directive.

2.2.3 Symbols

The as850 supports the use of symbols as the constituents of the absolute expressions or relative expressions that can be used to specify the operands of instructions and quasi directives.

2.2.4 Label references

With the as850, label references can be used as the constituents of the relative expressions that can be used to specify the operand of the following instructions/quasi directive:

- Memory reference instructions (load/store and bit manipulation instructions)
- Operation instructions (arithmetic instructions, logical instructions, and saturation operation instructions)
- Branch instructions
- Area allocation quasi directive (only .word/.hword/.byte quasi directive)

The meaning of a label reference varies with the reference method and the differences in the instructions/ quasi directives. Detail is shown below.

Table 2 - 2 Label Referencing

Reference Method	Instruction Used	Meaning
#label	Memory reference instructions, operation instructions, jmp instruction	The absolute address of the position at which the definition of the label label exists (the offset from address 0Note 1). This has a 32-bit address and must be expanded into two instructions.
	Area allocation quasi directives (.word/.hword/.byte)	The absolute address of the position at which the definition of the label label exists (the offset from address 0Note 1). Note that the 32-bit address is a value masked in accordance with the size of the area secured.
label	Memory reference instructions, operation instructions	The offset in the section at the position at which the definition of the label label exists (the offset from the first address of the section where the definition of the label label existsNote 2). This has a 32-bit offset and must be expanded into two instructions. Note that for a section allocated to a segment for which a tp symbol is to be generated, the offset is referenced from the tp symbol.
	Branch instructions except jmp instruction	The PC offset at the position at which the definition of the label label exists (the offset from the first address of the instruction using the reference of the label label).
	Area allocation quasi directives (.word/.hword/.byte)	The offset in the section at the position at which the definition of the label label exists (the offset from the first address of the section where the definition of the label label existsNote 2). Note that the 32-bit offset is a value masked in accordance with the size of the area secured.
$label	Memory reference instructions, operation instructions	The gp offset at the position at which the definition of the label label exists (the offset from the address pointed to by the global pointerNote 3)

Table 2 - 2 Label Referencing

Reference Method	Instruction Used	Meaning
!label	Memory reference instructions, operation instructions	The absolute address at the position at which the definition of the label label exists (the offset from address 0Note 1). This has a 16-bit address and cannot be instruction expanded if instructions with 16-bit displacement or immediate data are specified. If any other instructions are specified, expansion into appropriate 1-instruction units is possible. If the address defined by the label label is not within a range expressible by 16 bits, an error will be output at linking.
	Area allocation quasi directives (.word/.hword/.byte)	The absolute address of the position at which the definition of the label label exists (the offset from address 0Note 1). Note that the 32-bit address is a value masked in accordance with the size of the area secured.
%label	Memory reference instructions, operation instructions	The offset in the section at the position at which the definition of the label label exists (the offset from the first address of the section where the definition of the label label existsNote 2). This has a 16-bit address and cannot be instruction expanded if instructions with 16-bit displacement or immediate data are specified. If any other instructions are specified, expansion into appropriate 1-instruction units is possible. If the address defined by the label label is not within a range expressible by 16 bits, an error will be output at linking. The ep offset at the position at which the definition of the label label exists (the offset from the address pointed to by the element pointer).
	Area allocation quasi directives (.word/.hword/.byte)	The offset in the section at the position at which the definition of label label exists (the offset from the first address of the section where the definition of the label label existsNote 2). Note that the 32-bit offset is a value masked in accordance with the size of the area secured.

Notes 1 The offset from address 0 in linked object file

2 The offset from the first address of the section (output section) to which the section in which the definition of label label exists is allocated in the linked object file

3 The offset from the address indicated by the value of the text pointer symbol + value of the global pointer for the segment to which the above output section is allocated.

The meanings of label references for memory reference instructions, operation instructions, branch instructions, and area allocation quasi directives are shown below.

Table 2 - 3 Memory Reference Instructions

Reference Method	Meaning
#label [reg]	The absolute address of the label label is regarded as a displacement. This has a 32-bit value and must be expanded into two instructions. By setting #label[r0], referencing by an absolute address can be specified. [reg] can be omitted. If omitted, the as850 assumes that [r0] has been specified.
label [reg]	The offset in the section of the label label is regarded as a displacement. This has a 32-bit value and must be expanded into two instructions. By specifying a register indicating the first address of the section as reg and thereby setting label[reg], general register relative referencing can be specified. For a section allocated to a segment for which a tp symbol is to be generated, however, the offset from the tp symbol is regarded as a displacement.
$label [reg]	The gp offset of the label label is regarded as a displacement. This has either a 32-bit or 16-bit value, depending on the section defined by the label label, and its instruction expansion pattern changes accordingly[Note] . If an instruction with a 16-bit value is expanded and the offset calculated by the address defined by the label label is not within a range that can be expressed in 16 bits, an error is output at linking. By setting $label[gp], relative referencing of the gp register (called a gp offset reference) can be specified. [reg] can be omitted. If omitted, the as850 assumes that [gp] has been specified.
!label [reg]	The absolute address of the label label is regarded as a displacement. This has a 16-bit value and is not instruction expanded. If the address defined by the label label cannot be expressed in 16 bits, an error is output at linking. By setting !label[r0], referencing by an absolute address can be specified. [reg] can be omitted. If omitted, the as850 assumes that [r0] is specified. Unlike #label[reg] referencing, however, instruction expansion is not executed.
%label [reg]	The offset in the section of the label label is regarded as a displacement. If the label label is allocated to a section that is the ep symbol, the offset from the ep symbol is regarded as a displacement. This has either a 16-bit value, or depending on the instruction a value lower than this, and if it is not a value that can be expressed within this range, an error is output at linking. [reg] can be omitted. If omitted, the as850 assumes that [ep] has been specified.

Note Refer to "2.2.6 gp offset reference".

Table 2 - 4 Operation Instructions

Reference Method	Meaning
#label	The absolute address of the label label is regarded as an immediate value. This has a 32-bit value and must be expanded into two instructions.
label	The offset in the section of the label label is regarded as an immediate value. This has a 32-bit value and must be expanded into two instructions. For a section allocated to a segment for which a tp symbol is to be generated, however, the offset from the tp symbol is regarded as an immediate value.
$label	The gp offset of the label label is regarded as an immediate value. This has a 32-bit value and must be expanded into two instructions. This has either a 32-bit or 16-bit value, depending on the section defined by the label label, and its instruction expansion pattern changes accordingly[Note 1]. If an instruction with a 16-bit value is expanded and the offset calculated by the address defined by the label label is not within a range that can be expressed in 16 bits, an error is output at linking.
!label	The absolute address of the label label is regarded as an immediate value. This has a 16-bit value, and if operation instructions of an architecture for which a 16-bit value can be specified[Note 2] as immediate are specified, instruction expansion is not executed. If the add, mov, and mulh instructions are specified, expansion into appropriate 1-instruction units is possible. No other instructions can be specified. If the value is not within a range that can be expressed in 16 bits, an error is output at linking.
%label	The offset in the section of the label label is regarded as an immediate value. If the label label is allocated to a section that is a target of the ep symbol, the offset from the ep symbol is regarded as a displacement. This has a 16-bit value, and if operation instructions of an architecture for which a 16-bit value can be specified[Note 2] as immediate are specified, instruction expansion is not executed. Unlike label referencing, however, instruction expansion is not executed. This referencing method can be specified only for operation instructions of an architecture for which a 16-bit value can be specified as immediate, as well as the add, mov, and mulh instructions. Note that if the add, mov, and mulh instructions are specified, expansion into appropriate 1-instruction units is possible. No other instructions can be specified. If the value is not within a range that can be expressed in 16 bits, an error is output at linking.

Notes 1 Refer to "2.2.6 gp offset reference".

2 The instructions for which a 16-bit value can be specified as immediate are the addi, andi, movea, mulhi, ori, satsubi, and xori instructions.

Table 2 - 5 Branch Instructions

Reference Method	Meaning
#label	The absolute address of the label label for the jmp instruction is regarded as the jump destination address. This has a 32-bit value and must be expanded into three instructions.
label	The PC offset of the label label for branch instructions other than the jmp instruction is regarded as being a displacement. This is a 22-bit value, and if it is not within a range that can be expressed in 22 bits, an error is output at linking.

Table 2 - 6 Area Allocation Quasi Directives

Reference Method	Meaning
#label !label	The absolute address of the label label for the .word/.hword/.byte quasi instructions is regarded as a value. This has a 32-bit value, but is masked in accordance with the bit width of the relevant quasi directive.
label %label	The offset in the section defined by the label label for the .word/.hword/.byte quasi instructions is regarded as a value. This has a 32-bit value, but is masked in accordance with the bit width of the relevant quasi directive.
$label	The gp offset of the label label for the .word/.hword/.byte quasi instructions is regarded as a value. This has a 32-bit value, but is masked in accordance with the bit width of the relevant quasi directive.

2.2.5 ep offset reference

This section describes the ep offset reference. The CA850 assumes that data explicitly stored in internal RAM is shown below.

Referenced by the offset from the address indicated by the element pointer (ep).

Data in the internal RAM is divided into the following two groups.

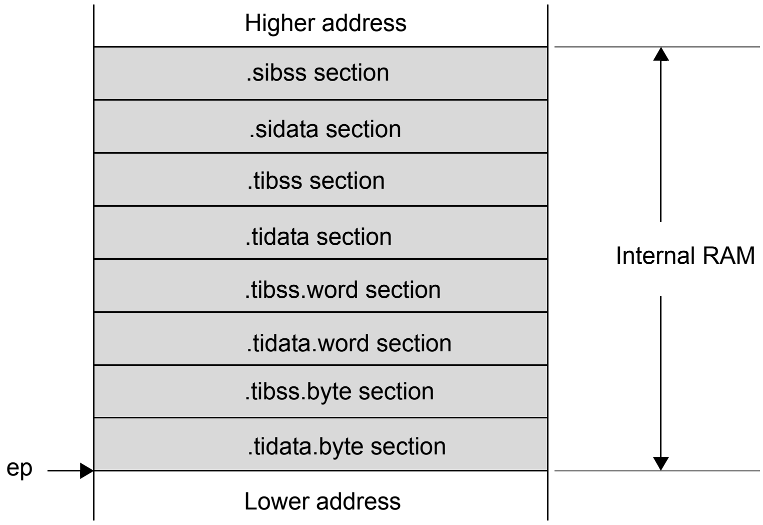
(1) .tidata/.tibss/.tidata.byte/.tibss.byte/.tidata.word/.tibss.word section

Data referenced by memory reference instructions (sld/sst) and having a small code size

(2) .sidata/.sibss section

Data referenced by memory reference instructions (ld/st) and having a large code size

Figure 2 - 1 Memory Location Image of Internal RAM



(1) Data allocation

Data is allocated to the sections in internal RAM as follows:

(a) When developing a program in C

(i) Allocate data by specifying the tidata or sidata section in the #pragma section command.

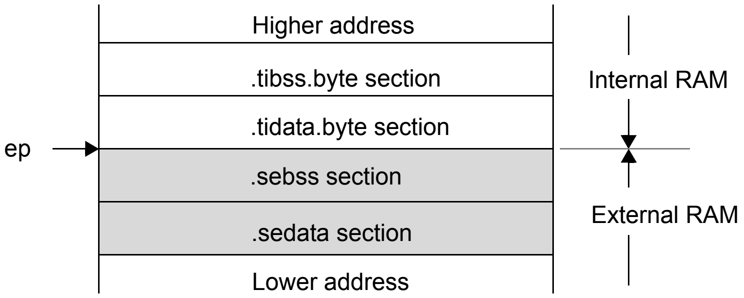
(ii) Allocate data by specifying the tidata or sidata section in the section file. Input the section file during compilation using a C compiler option.

(b) When developing a program in assembly language

Data is allocated to the .tidata, .tibss, .tidata.byte, .tibss.byte, .tidata.word, .tibss.word, .sidata, or .sibss section by a section definition quasi directive.

ep offset reference can also be executed with respect to data in a specific range of external RAM by allocating the data to sections .sedata and .sebss in the same manner as above.

Figure 2 - 2 Memory Allocation Image for External RAM (.sedata Section)



(2) Data reference

Using the data allocation method explained above, the as850 generates a machine instruction string that performs as follows:

(a) .Reference by ep offset for %label reference to data allocated to the .tidata, .tidata.byte, .tidata.word, .tibss, .tibss.byte, .tibss.word, .sidata, .sibss, .sedata, or .sebss section

(b) Reference by inter-section offset for %label reference to data allocated to other than that above

Example

```
    .sidata
sidata: .hword 0xfff0
    .data
data:   .hword 0xfff0

    .text
    ld.h    %sidata, r20     -- (1)
    ld.h    %data, r20       -- (2)
```

The as850 generates a machine instruction string for %label reference because:

- The as850 regards the code in (1) as being a reference by ep offset because the defined data is allocated to the .sidata section
- The as850 regards the code in (2) as being a reference by in-section offset

The as850 performs processing, assuming that the data is allocated to the correct section. If the data is allocated to other than the correct section, it cannot be detected by the as850.

Example

```
     .text
     ld.h     %label[ep], r20
```

Instructions are coded to allocate a label to the .sidata section and to perform reference by ep offset. Here, however, label is allocated to the .data section because of the allocation error. In this case, the as850 loads the data in the base register ep symbol value + offset value in the .data section of label.

```
     .text
     ld.h     %label1[r10], r20    --(1)
     .option ep_label
     ld.h     %label2[ep], r21     --(2)
     .option no_ep_label
     ld.h     %label3[r10], r22    --(3)
```

- For (1), reference by ep offset or by in-section offset is performed according to the section in which the defined data is allocated (default).
- For (2), reference by ep offset is performed regardless of the section in which the defined data is allocated, because label is within the range specified by the .option ep_label quasi directive.
- For (3), the operation is the same as (1) because label is within the range specified by the .option no_ep_label quasi directive.

2.2.6 gp offset reference

This section describes gp offset reference.

The CA850 assumes that data stored in external RAM (other than the .sedata or .sebss section explained on the previous page) is basically shown below.

Referenced by the offset from the address indicated by the global pointer (gp).

If r0-relative memory allocation for internal ROM or RAM is not done with the #pragma section command of C, the section file to be input to the C compiler, or an assembly language section definition quasi directive, all data is subject to gp offset reference.

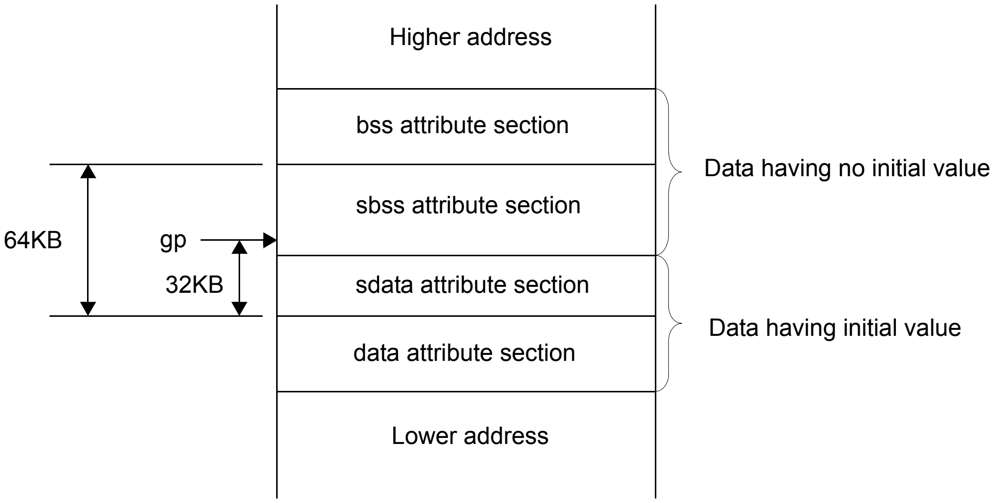
(1) Data allocation

The memory reference instruction (ld/st) of the machine instruction of the V850 microcontrollers can only accept 16-bit immediate as a displacement. For this reason, the CA850 classifies data into the following two types:

(a) Data allocated to a memory range that can be referenced by using the global pointer (gp) and a 16-bit displacement

(b) Data allocated to a memory range that can be referenced by the global pointer (gp) and a 32-bit displacement (consisting of two or more instructions). Data of the former type is allocated to the sdata- or sbss-attribute section, while that of the latter type is allocated to the data- or bss-attribute section.
Data having an initial value is allocated to the sdata/data-attribute section, while data without an initial value is allocated to the sbss/bss-attribute section. By default, the CA850 allocates data to the data-, sdata-, sbss-, then bss-attribute sections, starting from the lowest address. Moreover, it is assumed that the global pointer (gp) is set by a start up module to point to the address resulting from addition of 32 KB to the first address of the sdata-attribute section.

Figure 2 - 3 Memory Location Image of gp Offset Reference Section



Remark The sum of sdata- and sbss-attribute sections is 64 KB. gp is 32 KB below the first byte of the sdata-attribute section.

Data in the sdata- and sbss-attribute sections can be referenced by using a single instruction. To reference data in the data- and bss-attribute sections, however, two or more instructions are necessary.

Therefore, the more data allocated to the sdata- and sbss-attribute sections, the higher the execution efficiency and object efficiency of the generated machine instructions.

However, the size of the memory range that can be referenced with a 16-bit displacement is limited. If all the data cannot be allocated to the sdata- and sbss-attribute sections, it becomes necessary to determine which data is to be allocated to the sdata- and sbss-attribute sections.

The CA850 "allocates as much data as possible to the sdata- and sbss-attribute sections." . By default, all data items are allocated to the sdata- and sbss-attribute sections. The data to be allocated can be selected as follows:

(i) When the -G*num* option is specified

By specifying the -G*num* option upon starting the C compiler (ca850) or assembler (as850), data of less than *num* bytes is allocated to the sdata- and sbss-attribute sections.

(ii) When using a program to specify the section to which data will be allocated

Explicitly allocate data that will be frequently referenced to the sdata- and sbss-attribute sections. For allocation, use a section definition quasi directive when using the assembly language, or the #pragma section command when using C.

(iii) Specifying with the section file

In C, allocate data by specifying the sdata section in the section file. Input the section file during compilation with a C compiler option.

(2) Data reference

Using the data allocation method explained above, the as850 generates a machine instruction string that performs:

(a) Reference by using a 16-bit displacement for gp offset reference to data allocated to the sdata- and sbss-attribute sections

(b) Reference by using a 32-bit displacement (consisting of two or more machine instructions) for gp offset reference to data allocated to the data- and bss-attribute sections

Example

```
    .data
data:   .word 0xfff00010        --(1)

    .text
    ld.w    $data[gp], r20      --(2)
```

The as850 generates a machine instruction string, equivalent to the following instruction string for the ld.w instruction in (2), that performs gp offset reference of the data defined in (1)[Note].

```
    movhi   hi1($data), gp, r1
    ld.w    lo($data)[r1], r20
```

Note For details of hi1()/lo(), refer to "2.2.7 hi()/lo()/hi1()".

The as850 processes files on a one-by-one basis. Consequently, it can identify to which attribute section data having a definition in a specified file has been allocated, but cannot identify the section to which data not having a definition in a specified file has been allocated.

Therefore, the as850 generates machine instructions as follows[Note 2], when the -G*num* option is specified[Note 1] at start-up, assuming that the allocation policy described above (i.e., data smaller than a specific size is allocated to the sdata- and sbss-attribute sections) is observed.

Notes 1 If the as850 is started from the ca850, the -G*num* option, specified upon starting the ca850, is passed to the as850.

2 The data, for which data or sdata is specified by the .option quasi directive, is assumed to be allocated in the .data or .sdata section regardless of its size.

(c) Generates machine instructions that perform reference by using a 16-bit displacement for gp offset reference to data not having a definition in a specified file and which consists of less than *num* bytes.

(d) Generates a machine instruction string that performs reference by using a 32-bit displacement (consisting of two or more machine instructions) for gp offset reference to data having no definition in a specified file and which consists of more than *num* bytes.

To identify these conditions, however, the size of the data not having a definition in a specified file, and which is referenced by a gp offset, must be identified.

To develop a program in an assembly language, therefore, specify the size of the data (actually, a label for which there is no definition in a specified file and which is referenced by a gp offset) for which there is no definition in a specified file, by using the .extern quasi directive.

Example

```
    .extern data, 4              --(1)

    .text
    ld.w    $data [gp], r20      --(2)
```

When -G2 is specified upon starting the as850, the as850 generates a machine instruction string, equivalent to the following instruction string, for the ld.w instruction in (2) that performs gp offset reference to the data declared in (1)[Note].

```
    movhi   hi1($data), gp, r1
    ld.w    lo($data)[r1], r20
```

Note For hi1()/lo(), refer to "2.2.7 hi()/lo()/hi1()".

To develop a program in C, the C compiler (ca850) of the CA850 automatically generates the .extern quasi directive, thus outputting code which specifies the size of data not having a definition in the specified file (actually, a label for which there is no definition in a specified file and which is referenced by a gp offset).

[Summary]

The handling of gp offset reference (specifically, memory reference instructions that use a relative expression having the gp offset of a label as their displacement) by the as850 is summarized below:

(1) If the data has a definition in a specified file

(a) If the data is to be allocated to the sdata- or sbss-attribute sectionNote

Generates a machine instruction that performs reference by using a 16-bit displacement.

(b) If the data is not allocated to the sdata- or sbss-attribute section

Generates a machine instruction string that performs reference by using a 32-bit displacement.

Note If the value of the constant expression of a relative expression in the form of "label ± constant expression" exceeds 16 bits, the as850 generates a machine instruction string that performs reference using a 32-bit displacement.

(2) If the data does not have a definition in a specified file

(a) If the -G*num* option is specified upon starting the assembler

If a size of other than 0, but less than *num* bytes is specified for the data (label referenced by gp offset) by the .comm, .extern, .globl, .lcomm, or .size quasi directive.

Assumes that the data is to be allocated to the sdata- or sbss-attribute section and generates a machine instruction that performs reference by using a 16-bit displacement.

Other than above, assumes that the data is not allocated to the sdata- or sbss-attribute section and generates a machine instruction string that performs reference using a 32-bit displacement.

(b) If the -G*num* option is not specified upon starting the assembler

Assumes that the data is to be allocated to the sdata- or sbss-attribute section and generates a machine instruction that performs reference using a 16-bit displacement.

2.2.7 hi( )/lo( )/hi1( )

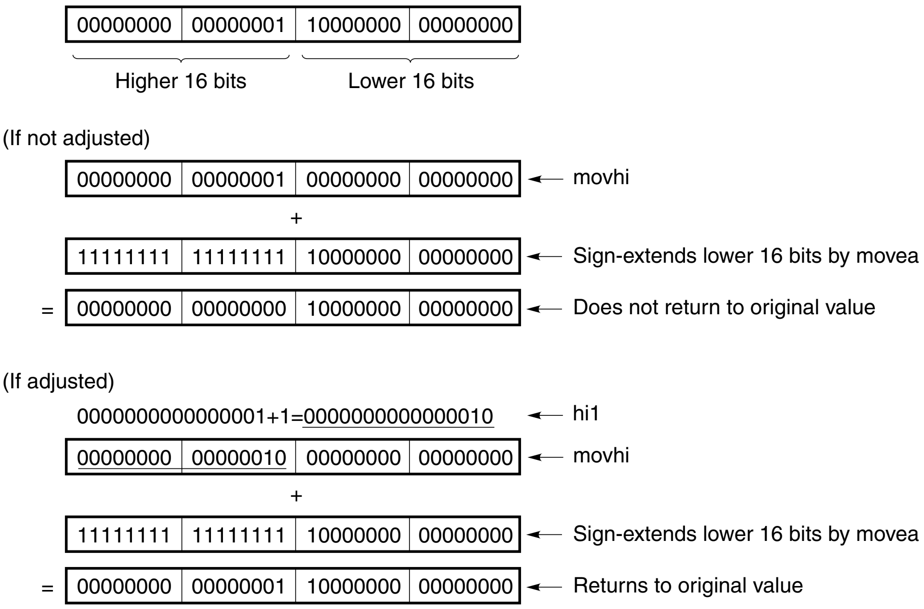
(1) To store 32-bit constant value in a register

The V850 microcontrollers does not support a machine instruction that can store a 32-bit constant value in a register with a single instruction. To store a 32-bit constant value in a register, therefore, the as850 performs instruction expansion, and generates an instruction string, by using the movhi and movea instructions. These divide the 32-bit constant value into the higher 16 bits and lower 16 bits.

Example

`mov     0x18000, r11`	`movhi   hi1(0x18000), r0, r1`
	`movea   lo(0x18000), r1, r11`

At this time, the movea instruction, used to store the lower 16 bits in the register, sign-extends the specified 16-bit value to a 32-bit value[Note]. To adjust the sign-extended bits, the as850 does not merely store the higher 16 bits in a register when using the movhi instruction, instead it stores the value of "the higher 16 bits + the most significant bit (i.e., bit 15) of the lower 16 bits" in the register.



00000000 | 00000001 | 10000000 | 00000000

Higher 16 bits   Lower 16 bits

(If not adjusted)

00000000 | 00000001 | 00000000 | 00000000 ← movhi

+

11111111 | 11111111 | 10000000 | 00000000 ← Sign-extends lower 16 bits by movea

= 00000000 | 00000000 | 10000000 | 00000000 ← Does not return to original value

(If adjusted)

0000000000000001+1=0000000000000010 ← hi1

00000000 | 00000010 | 00000000 | 00000000 ← movhi

+

11111111 | 11111111 | 10000000 | 00000000 ← Sign-extends lower 16 bits by movea

= 00000000 | 00000001 | 10000000 | 00000000 ← Returns to original value

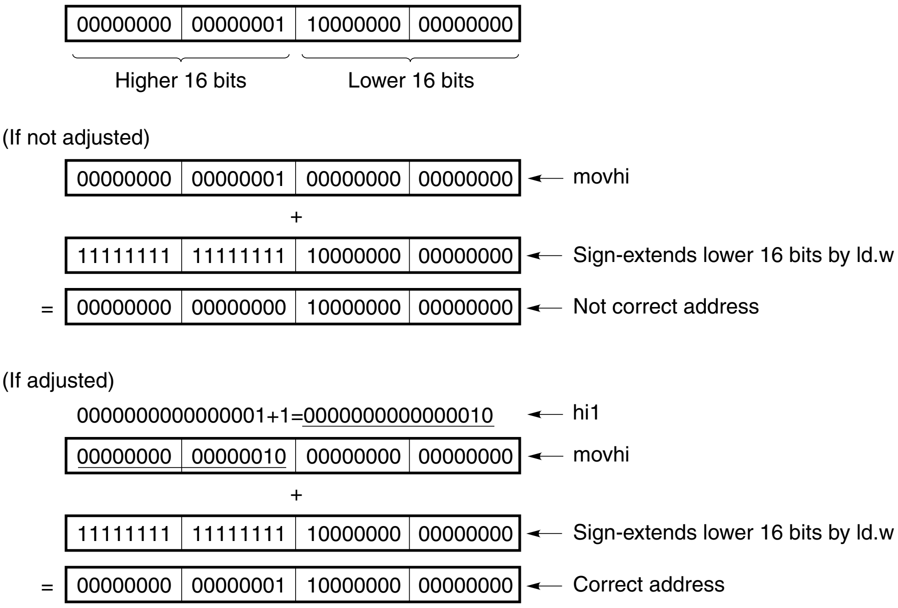
(2) To reference memory by using 32-bit displacement

The memory reference instruction (Load/store and bit manipulation instructions) of the machine instructions of the V850 microcontrollers can take only a 16-bit immediate as a displacement. Consequently, the as850 performs instruction expansion to reference the memory by using a 32-bit displacement, and generates an instruction string that performs the reference, by using the movhi and memory reference instructions and thereby constituting a 32-bit displacement from the higher 16 bits and lower 16 bits of the 32-bit displacement.

Example

`ld.w    0x18000[r11], r12`	`movhi   hi1(0x18000), r11, r1`
	`ld.w    lo(0x18000)[r1], r12`

At this time, the memory reference instruction that uses the lower 16 bits as a displacement sign-extends the specified 16-bit displacement to a 32-bit value. To adjust the sign-extended bits, the as850 does not merely configure the displacement of the higher 16 bits by using the movhi instruction, instead it configures the displacement of "the higher 16 bits + most significant bit (i.e., bit 15) of the lower 16 bits".



(3) hi( )/lo( )/hi1( )

In the next table, the as850 can specify the higher 16 bits of a 32-bit value, the lower 16 bits of a 32-bit value, and the value of the higher 16 bits + bit 15 of a 32-bit value by using hi( ), lo( ), and hi1( )[Note] .

Note If this information cannot be internally resolved by the assembler, it is reflected in the relocation information and subsequently resolved by the link editor (ld850).

Table 2 - 7 Meanings of hi( ) /lo( ) /hi1( )

hi( ) /lo( ) /hi1( )	Meaning
hi(value)	Higher 16 bits of value
lo(value)	Lower 16 bits of value
hi1(value)	Higher 16 bits of value + value of bit 15 of value

Example

```
    .data
L1 :
         :
    .text
    movhi   hi($L1), r0, r10    -- Stores the higher 16 bits of the gp offset
                                -- value of L1 in the higher 16 bits of r10,
                                -- and the lower 16 bits to 0

    movea   lo($L1), r0, r10    -- Sign-extends and stores the lower 16 bits of
                                -- gp offset value of L1 in r10
         :

    movhi   hi1($L1), r0, r1    -- Stores the gp offset value of L1 in r10
    movea   lo($L1), r1, r10
```

2.3 Runtime Library

The architecture of the V850 microcontrollers does not support floating-point operation instructions. To satisfy the ANSI standard language specifications, therefore, the CA850 executes all floating-point operations by calling from the runtime library of the libc.a file.

Because the devices in the V850 microcontrollers other than the V850Ex do not have 32-bit data multiplication, division, and remainder instructions, these instructions are called from the runtime library in the same manner as floating-point operations.

The runtime library is a routine that is used when the ca850 compiles a C language source program. It can also work with source programs in assembly language. In this case, libc.a must be linked with the ld850 when an executable object file is generated.

2.4 Macro Operators

This section describes a tilde (~), used as a zero-length delimiter in a macro body, and a dollar ($), used to specify a symbol value as an argument in a macro call.

2.4.1 Tilde symbol

The as850 handles a tilde (~) in a macro body as a zero-length delimiter. If, however, the tilde appears in a string constant or comment, it is not regarded as being a delimiter, but as a normal tilde (~).

Example1

```
.macro abc x
    abc~x:
        mov r10, r20
        sub def~x, r20
.endm
abc NECEL
```

The expansion result of the above example is shown below:

```
abcNECEL:
    mov     r10, r20
    sub     defNECEL, r20
```

Example2

```
.macro abc x, xy
    a_~xy:  mov     r10, r20
    a_~x~y: mov     r20, r10
.endm
abc necel, NECEL
```

The expansion result of the above example is shown below:

```
a_NECEL:  mov r10, r20
a_necely: mov r20, r10
```

Example3

```
.macro abc x, xy
    ~ab: mov    r10, r20
.endm
abc necel, NECEL
```

The expansion result of the above example is shown below:

```
ab: mov     r10, r20
```

2.4.2 Dollar symbol

If a symbol prefixed with a dollar symbol ($) is specified as an actual argument for a macro call, the as850 assumes the symbol to be specified as an actual argument.

If, however, an identifier other than a symbol or an undefined symbol name is specified immediately after the dollar symbol ($), the as850 outputs the following message then stops assembling.

```
$ must be followed by defined symbol
```

```
.macro mac1 x
    mov     x, r10
.endm
.macro mac2
    .set    value, 10
    mac1    value
    mac1    $value
.endm
mac2
```

The expansion result of the above example is shown below:

```
    .set    value, 10
    mov     value, r10
    mov     10, r10
```

CHAPTER 3 ASSEMBLY LANGUAGE INSTRUCTIONS

This section describes the instructions of the assembly language supported by the CA850 assembler (as850).

3.1 Description of Format

This section describes the instructions of the assembly language supported by the CA850 assembler (as850). For details of the machine instructions generated by the as850, refer to the Relevant Device’s Architecture User’s Manual of the V850 microcontrollers.

Instruction

[Overview]

Indicates the meaning of the instruction.

[Syntax]

Indicates the syntax of the instruction.

[Function]

Indicates the function of the instruction.

[Description]

Indicates the operation performed by the instruction.

[Flag]

Indicates the flag value after the execution of the instruction. Note, however, that the value of the flag before execution is indicated for the clr1, not1, and set1 instructions.

"---" indicates that the flag value is not affected by instruction execution.

[Caution]

Indicates the points to be noted when using the instruction.

3.2 Load/Store Instructions

This section describes the load/store instructions.

Next table lists the instructions described in this section

Table 3 - 1 Load/Store Instructions

Instruction		Meaning
ld	ld.b	Load (byte)
	ld.bu	Load (unsigned byte) **[V850E]**
	ld.h	Load (halfword)
	ld.hu	Load (unsigned halfword) **[V850E]**
	ld.w	Load (word)
sld	sld.b	Byte data load (short format)
	sld.bu	Unsigned byte data load (short format) **[V850E]**
	sld.h	Halfword data load (short format)
	sld.hu	Unsigned halfword data load (short format) **[V850E]**
	sld.w	Word data load (short format)
sst	sst.b	Byte data store (short format)
	sst.h	Halfword data store (short format)
	sst.w	Word data store (short format)
st	st.b	Byte data store
	st.h	Halfword data store
	st.w	Word data store

ld

[Overview]

Data load

[Syntax]

```
(1) ld.b     disp[reg1], reg2
(2) ld.h     disp[reg1], reg2
(3) ld.w     disp[reg1], reg2
(4) ld.bu    disp[reg1], reg2  [V850E]
(5) ld.hu    disp[reg1], reg2  [V850E]
```

The following can be specified for displacement (disp):

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with hi( ), lo( ), or hi1( ) applied

[Function]

The ld.b, ld.bu, ld.h, ld.hu, and ld.w instructions load data of 1 byte, 1 halfword, and 1 word, from the address specified by the first operand, int the register specified by the second operand.

[Description]

- If any of the following is specified for disp, the as850 generates one ld machine instruction[Note].
  In the following explanations, ld denotes the ld.b / ld.h / ld.w / ld.bu / ld.hu instructions.

(a) Absolute expression having a value in the range of -32,768 to +32,767

`ld  disp16[reg1], reg2`	`ld  disp16[reg1], reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`ld  $label[reg1], reg2`	`ld  $label[reg1], reg2`

(c) Relative expression having !label or %label

`ld  !label[reg1], reg2`	`ld  !label[reg1], reg2`

`ld  %label[reg1], reg2`	`ld  %label[reg1], reg2`

(d) Expression with hi( ), lo( ), or hi1( )

`ld  disp16[reg1], reg2`	`ld  disp16[reg1], reg2`

Note The ld machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xffff8000 to 0x7fff) as the displacement.

- If any of the following is specified for disp, the as850 performs instruction expansion to generate multiple machine instructions.

(a) Absolute expression having a value exceeding the range of -3,2768 to +32,767

`ld      disp[reg1], reg2`	`movhi   hi1(disp), reg1, r1`
	`ld      lo(disp)[r1], reg2`

(b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`ld      #label[reg1], reg2`	`movhi   hi1(#label), reg1, r1`
	`ld      lo(#label)[r1], reg2`

`ld      label[reg1], reg2`	`movhi   hi1(label), reg1, r1`
	`ld      lo(#label)[r1], reg2`

`ld      $label[reg1], reg2`	`movhi   hi1($label), reg1, r1`
	`ld      lo($label)[r1], reg2`

- If disp is omitted, the as850 assumes 0.
- If a relative expression having #label, or a relative expression having #label and with hi( ), lo( ), or hi1( ) applied is specified as disp, [reg1] can be omitted. If omitted, the as850 assumes that [r0] is specified.
- If a relative expression having $label, or a relative expression having $label and with hi( ), lo( ), or hi1( ) applied, is specified as disp, [reg1] can be omitted. If omitted, the as850 assumes that [gp] is specified.
- If a peripheral I/O register name defined in the device file is specified as disp, [reg1] can be omitted. If omitted, the as850 assumes that [r0] is specified.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- ld.b and ld.h sign-extend the data of 1 byte and 1 halfword, respectively, and load the data into a register as 1 word.
- ld.bu and ld.hu zero-extend the data of 1 byte and 1 halfword, respectively, and load the data into a register as 1 word.
- If a value that is not a multiple of 2 is specified as disp of ld.h, ld.w, or ld.hu, the as850 aligns disp with 2 and generates a code. Then, the as850 outputs either one of the messages below.

```
W3010: illegal displacement in inst instruction.
```

```
W4659: relocated value(value) of relocation entry (symbol: symbol, file: file,
section: section, offset: offset, type: relocation type) for load/store com-
mand become odd value.
```

- If r0 is specified as the second operand of ld.bu and ld.hu, the as850 outputs the following message and stops assembling

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

sld

[Overview]

Short format Load

[Syntax]

```
(1) sld.b      disp7[ep], reg2
(2) sld.h      disp8[ep], reg2
(3) sld.w      disp8[ep], reg2
(4) sld.bu     disp4[ep], reg2  [V850E]
(5) sld.hu     disp5[ep], reg2  [V850E]
```

The following can be specified for displacement (disp4/5/7/8):

- Absolute expression having a value of up to 7 bits for sld.b, 8 bits for sld.h and sld.w, 4 bits for sld.bu, and 5 bits for sld.hu.
- Relative expression

[Function]

The sld.b, sld.bu, sld.h, sld.hu, and sld.w instructions load the data of 1 byte, 1 halfword, and 1 word, from the address obtained by adding the displacement specified by the first operand to the contents of register ep, to the register specified by the second operand.

[Description]

- The as850 generates one sld machine instruction.
  Base register specification "[ep]" can be omitted.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- sld.b and sld.h sign-extend and store data of 1 byte and 1 halfword, respectively, in the register as 1 word.
- sld.bu and sld.hu zero-extend and store data of 1 byte and 1 halfword, respectively, in the register as 1 word.
- If a value that is not a multiple of 2 is specified as disp8 of sld.h or disp5 of sld.hu, and if a value that is not a multiple of 4 is specified as disp8 of sld.w, the as850 aligns disp8 or disp5 with multiples of 2 and 4, respectively, and generates a code. Then, the as850 outputs either one of the messages below.

```
W3010: illegal displacement in inst instruction.
```

```
W4659: relocated value(value) of relocation entry (symbol: symbol, file: file, section: section, offset: offset, type: relocation type) for load/store command become odd value.
```

- If a value exceeding 127 is specified for disp7 of sld.b, a value exceeding 255 is specified for disp8 of sld.h and sld.w, a value exceeding 16 is specified for disp4 of sld.bu, and a value exceeding 32 is specified for disp5 of sld.hu, the as850 outputs the following message, and generates code in which disp7, disp8, disp4, and disp5 are masked with 0x7f, 0xff, 0xf, and 0x1f, respectively.

```
W3011: illegal operand (range error in immediate)
```

- If r0 is specified as the second operand of the sld.bu and sld.hu, the as850 outputs the following message and stops assembling.

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

sst

[Overview]

Short format Store

[Syntax]

```
(1) sst.b      reg2, disp7[ep]
(2) sst.h      reg2, disp8[ep]
(3) sst.w      reg2, disp8[ep]
```

The following can be specified for displacement (disp7/8):

- Absolute expression having a value of up to 7 bits for sst.b or 8 bits for sst.h and sst.w
- Relative expression

[Function]

The sst.b, sst.h, and sst.w instructions store the data of the lower 1 byte, lower 1 halfword, and 1 word, respectively, of the register specified by the first operand to the address obtained by adding the displacement specified by the second operand to the contents of register ep.

[Description]

- The as850 generates one sst machine instruction.
  Base register specification "[ep]" can be omitted.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If a value that is not a multiple of 2 is specified as disp8 of sst.h, and if a value that is not a multiple of 4 is specified as disp8 of sst.w, the as850 aligns disp8 with multiples of 2 and 4, respectively, and generates a code. Then, the as850 outputs either one of the messages below.

```
W3010: illegal displacement in inst instruction.
```

```
W4659: relocated value(value) of relocation entry (symbol: symbol, file: file, section: section, offset: offset, type: relocation type) for load/store command become odd value.
```

- If a value exceeding 127 is specified as disp7 of sst.b, and if a value exceeding 255 is specified as disp8 of sst.h and sst.w, the as850 outputs the following message, and generates codes disp7 and disp8, masked with 0x7f and 0xff, respectively.

```
W3011: illegal operand (range error in immediate)
```

st

[Overview]

Store

[Syntax]

```
(1) st.b      reg2, disp[reg1]
(2) st.h      reg2, disp[reg1]
(3) st.w      reg2, disp[reg1]
```

The following can be specified as a displacement (disp):

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with hi( ), lo( ), or hi1( ) applied

[Function]

The st.b, st.h, and st.w instructions store the data of the lower 1 byte, lower 1 halfword, and 1 word, respectively, of the register specified by the first operand to the address specified by the second operand.

[Description]

- If any of the following is specified as disp, the as850 generates one st machine instruction[Note].
  In the following explanations, st denotes the st.b/st.h instructions.

(a) Absolute expression having a value in the range of -32,768 to +32,767

`st reg2, disp16[reg1]`	`st reg2, disp16[reg1]`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`st reg2, $label[reg1]`	`st reg2, $label[reg1]`

(c) Relative expression having !label or %label

`st reg2, !label[reg1]`	`st reg2, !label[reg1]`

`st reg2, %label[reg1]`	`st reg2, %label[reg1]`

(d) Expression with hi( ), lo( ), or hi1( ) applied

`st reg2, disp16[reg1]`	`st reg2, disp16[reg1]`

Note The st machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xffff8000 to 0x7fff) as the displacement.

- If any of the following is specified as disp, the as850 executes instruction expansion to generate two or more machine instructions.

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767

`st reg2, disp[reg1]`	`movhi hi1(disp), reg1, r1`
	`st reg2, lo(disp)[r1]`

(b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section.

`st reg2, #label[reg1]`	`movhi hi1(#label), reg1, r1`
	`st reg2, lo(#label)[r1]`

`st reg2, label[reg1]`	`movhi hi1(label), reg1, r1`
	`st reg2, lo(label)[r1]`

`st reg2, $label[reg1]`	`movhi hi1($label), reg1, r1`
	`st reg2, lo($label)[r1]`

- If disp is omitted, the as850 assumes 0.
- If a relative expression with #label, or a relative expression with #label and with hi( ), lo( ), or hi1( ) applied is specified as disp, [reg1] can be omitted. If omitted, the as850 assumes that [r0] is specified.
- If a relative expression with $label, or a relative expression with $label and with hi( ), lo( ), or hi1( ) applied is specified as disp, [reg1] can be omitted. If omitted, the as850 assumes that [gp] is specified.
- If a peripheral I/O register name defined in the device file is specified as disp, [reg1] can be omitted. If omitted, the as850 assumes that [r0] is specified.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If a value that is not a multiple of 2 is specified as the disp of st.h or st.w, the as850 aligns disp with 2 and generates a code. Then, the as850 outputs either one of the messages below.

```
W3010: illegal displacement in inst instruction.
```

```
W4659: relocated value(value) of relocation entry (symbol: symbol, file: file,
section: section, offset: offset, type: relocation type) for load/store com-
mand become odd value.
```

3.3 Arithmetic Operation Instructions

This section describes the arithmetic operation instructions. Next table lists the instructions described in this section.

Table 3 - 2 Arithmetic Operation Instructions

Instruction	Meaning
add	Addition
addi	Addition (immediate)
cmov	Transfers data depending on the flag condition **[V850E]**
cmp	Comparison
div	Signed division (word)) **[V850E]**
divh	Signed division (halfword)
divhu	Unsigned division (halfword) **[V850E]**
divu	Unsigned division (word) **[V850E]**
mov	Moves data
mov32	Moves data (32-bit) **[V850E]**
movea	Addition (32-bit immediate)
movhi	Addition (16-bit immediate)
mul	Signed multiplication (word) **[V850E]**
mulh	Signed multiplication (halfword)
mulhi	Signed multiplication (halfword immediate)
mulu	Unsigned multiplication **[V850E]**
mac	Signed word data multiply and add **[V850E2]**
macu	Unsigned word data multiply and add **[V850E2]**
sasf	Sets the flag condition after a logical left shift **[V850E]**
setf	Sets flag condition
sub	Subtraction
subr	Reverse subtraction
adf	Add with condition flag **[V850E2]**
sbf	Subtract with condition flag **[V850E2]**

add

[Overview]

Add

[Syntax]

```
(1) add         reg1, reg2
(2) add         imm, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  Adds the value of the register specified by the first operand to the value of the register specified by the second operand, and stores the result into the register specified by the second operand.

- Syntax (2)

  Adds the value of the absolute expression or relative expression specified by the first operand to the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

[Description]

- If this instruction is executed in syntax (1), the as850 generates one add machine instruction.
- If the following is specified as imm in syntax (2), the as850 generates one add machine instruction[Note].

(a) Absolute expression having a value in the range of -16 to +15

`add     imm15, reg`	`add     imm5, reg`

Note The add machine instruction takes a register or immediate value in the range of -16 to +15 (0xfffffff0 to 0xf) as the first operand.

- If the following is specified for imm in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions

(a) Absolute expression having a value exceeding the range of -16 to +15

`add     imm16, reg`	`addi    imm16, reg, reg`

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

add imm, reg	movhi hi(imm), r0, r1 add r1, reg

Else

add imm, reg	movhi hi1(imm), r0, r1 movea lo(imm), r1, r1 add r1, reg

(c) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

add imm, reg	movhi hi(imm), r0, r1 add r1, reg

Else

add imm, reg	mov imm, r1 add r1, reg

(d) Relative expression having !label or %label, or that having $label for a label with a definition in the sdata/sbss-attribute section

add $label, reg	addi !label, reg, reg

add %label, reg	addi %label, reg, reg

add $label, reg	addi $label, reg, reg

(e) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

add #label, reg	movhi hi1(#label), r0, r1 movea lo(#label), r1, r1 add r1, reg

add label, reg	movhi hi1(label), r0, r1 movea lo(label), r1, r1 add r1, reg

add $label, reg	movhi hi1($label), r0, r1 movea lo($label), r1, r1 add r1, reg

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

`add     #label, reg`	`mov     #label, r1`
	`add     r1, reg`

`add     label, reg`	`mov     label, r1`
	`add     r1, reg`

`add     $label, reg`	`mov     $label, r1`
	`add     r1, reg`

[Flag]

CY	1 if a carry occurs from MSB (Most Significant Bit), 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

addi

[Overview]

Add Immediate

[Syntax]

```
(1) addi        imm, reg1, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with hi( ), lo( ), or hi1( ) applied

[Function]

Adds the value of the absolute expression, relative expression, or expression with hi( ), lo( ), or hi1( ) applied, specified by the first operand, to the value of the register specified by the second operand, and stores the result into the register specified by the third operand.

[Description]

- If the following is specified for imm, the as850 generates one addi machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`addi    imm16, reg1, reg2`	`addi    imm16, reg1, reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`addi    $label, reg1, reg2`	`addi    $label, reg1, reg2`

(c) Relative expression having !label or %label

`addi    !label, reg1, reg2`	`addi    !label, reg1, reg2`

`addi    %label, reg1, reg2`	`addi    %label, reg1, reg2`

(d) Expression with hi( ), lo( ), or hi1( )

`addi    imm16, reg1, reg2`	`addi    imm16, reg1, reg2`

Note The addi machine instruction takes an immediate value in the range of -32,768 to +32,767 as the first operand.

- If the following is specified for imm, the as850 executes instruction expansion to generate two or more machine instructions.

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

addi imm, reg1, reg2	movhi hi(imm), r0, reg2 add reg1, reg2

If all the lower 16 bits of the value of imm are 0 and when reg2 is r0

addi imm, reg1, r0	movhi hi(imm), r0, r1 add reg1, r1

Else

addi imm, reg1, reg2	movhi hi1(imm), r0, r1 movea lo(imm), r1, reg2 add reg1, reg2

Other than above and when reg2 is r0

addi imm, reg1, r0	movhi hi1(imm), r0, r1 movea lo(imm), r1, r1 add reg1, r1

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

addi imm, reg1, reg2	movhi hi(imm), r0, reg2 add reg1, reg2

If all the lower 16 bits of the value of imm are 0 and when reg2 is r0

addi imm, reg1, r0	movhi hi(imm), r0, r1 add reg1, r1

Else

addi imm, reg1, reg2	mov imm, reg2 add reg1, reg2

Other than above and when reg2 is r0

addi imm, reg1, r0	mov imm, r1 add reg1, r1

(c) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

If reg2 is r0

`addi    #label, reg1, r0`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, r1`
	`add     reg1, r1`

`addi    label, reg1, r0`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, r1`
	`add     reg1, r1`

`addi    $label, reg1 r0`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, r1`
	`add     reg1, r1`

Else

`addi    #label, reg1, reg2`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, reg2`
	`add     reg1, reg2`

`addi    label, reg1, reg2`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, reg2`
	`add     reg1, reg2`

`addi    $label, reg1 reg2`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, reg2`
	`add     reg1, reg2`

(d) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

If reg2 is r0

`addi    #label, reg1, r0`	`mov     #label, r1`
	`addi    reg1, r1`

`addi    label, reg1, r0`	`mov     label, r1`
	`add     reg1, r1`

`addi    $label, reg1, r0`	`mov     $label, r1`
	`add     reg1, r1`

Else

`addi    #label, reg1, reg2`	`mov     #label, reg2`
	`addi    reg1, reg2`

`addi    label, reg1, reg2`	`mov     label, reg2`
	`add     reg1, reg2`

`addi    $label, reg1, reg2`	`mov     $label, reg2`
	`add     reg1, reg2`

[Flag]

CY	Most Significant Bit) , 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

cmov

[V850E]

[Overview]

Conditional Move

[Syntax]

```
(1) cmov        imm4, reg1, reg2, reg3
(2) cmov        imm4, imm, reg2, reg3
(3) cmovcond    reg1, ret2, reg3
(4) cmovcond    imm, reg2, reg3
```

The following can be specified for imm4:

- Constant expression having a value of up to 4 bits[Note]

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits

Note The cmov machine instruction takes an immediate value in the range of 0 to 15 (0x0 to 0xf) as the first operand.

[Function]

- Syntax (1)

  Compares the flag condition indicated by the value of the lower 4 bits of the value of the constant expression specified by the first operand with the current flag condition. If a match is found, the register value specified by the second operand is stored in the register specified by the fourth operand; otherwise, the register value specified by the third operand is stored in the register specified by the fourth operand.

- Syntax (2)

  Compares the flag condition indicated by the value of the lower 4 bits of the constant expression specified by the first operand with the current flag condition. If a match is found, the value of the absolute expression specified by the second operand is stored in the register specified by the fourth operand; otherwise, the register value specified by the third operand is stored in the register specified by the fourth operand.

- Syntax (3)

  Compares the flag condition indicated by string *cond* with the current flag condition. If a match is found, the register value specified by the first operand is stored in the register specified by the third operand; otherwise, the register value specified by the second operand is stored in the register specified by the third operand.

- Syntax (4)

Compares the flag condition indicated by string *cond* with the current flag condition. If a match is found, the value of the absolute expression specified by the first operand is stored in the register specified by the third operand; otherwise, the register value specified by the second operand is stored in the register specified by the third operand.

Table 3 - 3 cmov*cond* Instruction List

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
cmovgt	((S xor OV) or Z) = 0	Greater than (signed)	cmov 0xf
cmovge	(S xor OV) = 0	Greater than or equal (signed)	cmov 0xe
cmovlt	(S xor OV) = 1	Less than (signed)	cmov 0x6
cmovle	((S xor OV) or Z) = 1	Less than or equal (signed)	cmov 0x7
cmovh	(CY or Z) = 0	Higher (Greater than)	cmov 0xb
cmovnl	CY = 0	Not lower (Greater than or equal)	cmov 0x9
cmovl	CY = 1	Lower (Less than)	cmov 0x1
cmovnh	(CY or Z) = 1	Not higher (Less than or equal)	cmov 0x3
cmove	Z = 1	Equal	cmov 0x2
cmovne	Z = 0	Not equal	cmov 0xa
cmovv	OV = 1	Overflow	cmov 0x0
cmovnv	OV = 0	No overflow	cmov 0x8
cmovn	S = 1	Negative	cmov 0x4
cmovp	S = 0	Positive	cmov 0xc
cmovc	CY = 1	Carry	cmov 0x1
cmovnc	CY = 0	No carry	cmov 0x9
cmovz	Z = 1	Zero	cmov 0x2
cmovnz	Z = 0	Not zero	cmov 0xa
cmovt	always 1	Always 1	cmov 0x5
cmovsa	SAT = 1	Saturated	cmov 0xd

[Description]

- If the instruction is executed in syntax (1), the as850 generates one cmov machine instruction[Note].

Note The cmov machine instruction takes an immediate value in the range of -16 to +15 as the second operand.

- If the following is specified as imm in syntax (2), the as850 generates one cmov machine instruction.

(a) Absolute expression having a value in the range of -16 to +15

`cmov    imm4, imm5, reg2, reg3`	`cmov    imm4, imm5, reg2, reg3`

- If the following is specified as imm in syntax (2), the as850 executes instruction expansion to generate two or more machine instructions.

(a) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`cmov    imm4, imm16, reg2, reg3`	`movea   imm16, r0, r1`
	`cmov    imm4, r1, reg2, reg3`

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`cmov    imm4, imm, reg2, reg3`	`movhi   hi(imm), r0, r1`
	`cmov    imm4, r1, reg2, reg3`

Else

`cmov    imm4, imm, reg2, reg3`	`mov     imm, r1`
	`cmov    imm4, r1, reg2, reg3`

(c) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`cmov    imm4, $label, reg2, reg3`	`movea   $label, r0, r1`
	`cmov    imm4, r1, reg2, reg3`

(d) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`cmov    imm4, #label, reg2, reg3`	`mov     #label, r1`
	`cmov    imm4, r1, reg2, reg3`

`cmov    imm4, label, reg2, reg3`	`mov     label, r1`
	`cmov    imm4, r1, reg2, reg3`

`cmov    imm4, $label, reg2, reg3`	`mov     $label, r1`
	`cmov    imm4, r1, reg2, reg3`

- If the instruction is executed in syntax (3), the as850 generates the corresponding cmov instruction (refer to Table 3 - 3) and expands it to syntax (1).

- If the following is specified as imm in syntax (4), the as850 generates the corresponding cmov instruction (refer to Table 3 - 3) and expands it to syntax (2).

(a) Absolute expression having a value in the range of -16 to +15

- If the following is specified as imm in syntax (4), the as850 executes instruction expansion to generate two or more machine instructions.

(a) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

cmov*cond* imm16, reg2, reg3	movea imm16, r0, r1 cmov*cond* r1, reg2, reg3

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

cmov*cond* imm, reg2, reg3	movhi hi(imm), r0, r1 cmov*cond* r1, reg2, reg3

Else

cmov*cond* imm, reg2, reg3	mov imm, r1 cmov*cond* r1, reg2, reg3

(c) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

cmov*cond* $label, reg2, reg3	movea $label, r0, r1 cmov*cond* r1, reg2, reg3

(d) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

cmov*cond* #label, reg2, reg3	mov #label, r1 cmov*cond* r1, reg2, reg3

cmov*cond* label, reg2, reg3	mov label, r1 cmov*cond* r1, reg2, reg3

cmov*cond* $label, reg2, reg3	mov $label, r1 cmov*cond* r1, reg2, reg3

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If a constant expression having a value exceeding 4 bits is specified as imm4 of the cmov instruction, the as850 outputs the following message. If the value exceeds 4 bits, the as850 masks the value with 0xf and continues assembling.

```
W3011: illegal operand (range error in immediate)
```

- If anything other than a constant expression[Note] is specified as imm4 of the cmov instruction, the as850 outputs the following message and stops assembling.

```
E3249: illegal syntax
```

Note Undefined symbol and label reference.

cmp

[Overview]

Compare

[Syntax]

```
(1) cmp         reg1, reg2
(2) cmp         imm, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  Compares the value of the register specified by the first operand with the value of the register specified by the second operand, and indicates the result using a flag. Comparison is performed by subtracting the value of the register specified by the first operand from the value of the register specified by the second operand.

- Syntax (2)

  Compares the value of the absolute expression or relative expression specified by the first operand with the value of the register specified by the second operand, and indicates the result using a flag. Comparison is performed by subtracting the value of the register specified by the first operand from the value of the register specified by the second operand.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one cmp machine instruction.

- If the following is specified as imm in syntax (2), the as850 generates one cmp machine instruction[Note].

(a) Absolute expression having a value in the range of -16 to +15

`cmp     imm5, reg`	`cmp     imm5, reg`

Note The cmp machine instruction takes a register or immediate value in the range of -16 to +15 as the first operand.

- If the following is specified as imm in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions

(a) Absolute expression having a value exceeding the range of -16 to +15

`cmp     imm16, reg`	`movea   imm16, r0, r1`
	`cmp     r1, reg`

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`cmp     imm, reg`	`movhi   hi(imm), r0, r1`
	`cmp     r1, reg`

Else

`cmp     imm, reg`	`movhi   hi1(imm), r0, r1`
	`movea   lo(imm), r1, r1`
	`cmp     r1, reg`

(c) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

`cmp     imm, reg`	`movhi   hi(imm), r0, r1`
	`cmp     r1, reg`

Else

`cmp     imm, reg`	`mov     imm, r1`
	`cmp     r1, reg`

(d) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`cmp     $label, reg`	`movea   $label, r0, r1`
	`cmp     r1, reg`

(e) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`cmp     #label, reg`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, r1`
	`cmp     r1, reg`

`cmp     label, reg`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, r1`
	`cmp     r1, reg`

`cmp     $label, reg`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, r1`
	`cmp     r1, reg`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

`cmp     #label, reg`	`mov     #label, r1`
	`cmp     r1, reg`

`cmp     label, reg`	`mov     label, r1`
	`cmp     r1, reg`

`cmp     $label, reg`	`mov     $label, r1`
	`cmp     r1, reg`

[Flag]

CY	1 if a borrow occurs from MSB (Most Significant Bit) , 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is negative, 0 if not
SAT	---

div

[V850E]

[Overview]

Divide Word

[Syntax]

```
(1) div          reg1, reg2, reg3
(2) div          imm, reg2, reg3
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  Divides the register value specified by the second operand by the register value specified by the first operand as a signed value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.

- Syntax (2)

  Divides the register value specified by the second operand by the value of the absolute or relative expression specified by the first operand as a signed value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one div machine instruction[Note].
- If the instruction is executed in syntax (2), the as850 executes instruction expansion to generate two or more machine instructions[Note].

(a) 0

`div      0, reg2, reg3`	`div      r0, reg2, reg3`

(b) Absolute expression having a value of other than 0 whithin the range of -16 to +15

`div      imm5, reg2, reg3`	`mov      imm5, r1`
	`div      r1, reg2, reg3`

(c) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`div     imm16, reg2, reg3`	`movea   imm16, r0, r1`
	`div     r1, reg2, reg3`

(d) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`div     imm, reg2, reg3`	`movhi   hi(imm), r0, r1`
	`div     r1, reg2, reg3`

Else

`div     imm, reg2, reg3`	`mov     imm, r1`
	`div     r1, reg2, reg3`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`div     $label, reg2, reg3`	`movea   $label, r0, r1`
	`div     r1, reg2, reg3`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`div     #label, reg2, reg3`	`mov     #label, r1`
	`div     r1, reg2, reg3`

`div     label, reg2, reg3`	`mov     label, r1`
	`div     r1, reg2, reg3`

`div     $label, reg2, reg3`	`mov     $label, r1`
	`div     r1, reg2, reg3`

Note The div machine instruction does not take an immediate value as an operand.

[Flag]

CY	---
OV	1 if an Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

divh

[Overview]

Divide Half-word

[Syntax]

```
(1) divh        reg1, reg2
(2) divh        imm, reg2
(3) divh        reg1, reg2, reg3  [V850E]
(4) divh        imm, reg2, reg3  [V850E]
```

The following can be specified for imm16:

- Absolute expression[Note] having a value of up to 16 bits
- Relative expression

Note The as850 does not check whether the value of the expression exceeds 16 bits. The generated machine instruction performs execution using the lower 16 bits.

[Function]

- Syntax (1)

  Divides the register value specified by the second operand by the value of the lower halfword data of the register specified by the first operand as a signed value, and stores the quotient in the register specified by the second operand.

- Syntax (2)

  Divides the register value specified by the second operand by the value of the lower halfword data of the absolute or relative expression specified by the first operand as a signed value and stores the quotient in the register specified by the second operand.

- Syntax (3)

  Divides the register value specified by the second operand by the value of the lower halfword data of the register specified by the first operand as a signed value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.

- Syntax (4)

  Divides the register value specified by the second operand by the value of the lower halfword data of the absolute or relative expression specified by the first operand as a signed value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.

[Description]

- If the instruction is executed in syntaxes (1) and (3), the as850 generates one divh machine instruction.
- If the instruction is executed in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions[Note].

(a) 0

`divh    0, reg`	`divh    r0, reg`

(b) Absolute expression having a value of other than 0 whithin the range of -16 to +15

`divh    imm5, reg`	`mov     imm5, r1`
	`divh    r1, reg`

(c) Absolute expression having a value of other than 0 whithin the range of -16 to +15 **[V850E]**

`divh    imm5, reg`	`mov     imm5, r1`
	`divh    r1, reg`

(d) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`divh    imm16, reg`	`movea   imm16, r0, r1`
	`divh    r1, reg`

(e) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`divh    imm, reg`	`movhi   hi(imm), r0, r1`
	`divh    r1, reg`

Else

`divh    imm, reg`	`movhi   hi1(imm), r0, r1`
	`movea   lo(imm), r1, r1`
	`divh    r1, reg`

(f) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

`divh    imm, reg`	`movhi   hi(imm), r0, r1`
	`divh    r1, reg`

Else

`divh    imm, reg`	`mov     imm, r1`
	`divh    r1, reg`

(g) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

divh $label, reg	movea $label, r0, r1 divh r1, reg

(h) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

divh #label, reg	movhi hi1(#label), r0, r1 movea lo(#label), r1, r1 divh r1, reg

divh label, reg	movhi hi1(label), r0, r1 movea lo(label), r1, r1 divh r1, reg

divh $label, reg	movhi hi1($label), r0, r1 movea lo($label), r1, r1 divh r1, reg

(i) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

divh #label, reg	mov #label, r1 divh r1, reg

divh label, reg	mov label, r1 divh r1, reg

divh $label, reg	mov $label, r1 divh r1, reg

Note The divh machine instruction does not take an immediate value as an operand.

- If the instruction is executed in syntax (4), the as850 executes instruction expansion to generate one or more machine instructions

(a) 0

divh 0, reg2, reg3	divh r0, reg2, reg3

(b) Absolute expression having a value of other than 0 whithin the range of -16 to +15

`divh    imm5, reg2, reg3`	`mov     imm5, r1`
	`divh    r1, reg2, reg3`

(c) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`divh    imm16, reg2, reg3`	`movea   imm16, r0, r1`
	`divh    r1, reg2, reg3`

(d) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`divh    imm, reg2, reg3`	`movhi   hi(imm), r0, r1`
	`divh    r1, reg2, reg3`

Else

`divh    imm, reg2, reg3`	`mov     imm, r1`
	`divh    r1, reg2, reg3`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`divh    $label, reg2, reg3`	`movea   $label, r0, r1`
	`divh    r1, reg2, reg3`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`divh    #label, reg2, reg3`	`mov     #label, r1`
	`divh    r1, reg2, reg3`

`divh    label, reg2, reg3`	`mov     label, r1`
	`divh    r1, reg2, reg3`

`divh    $label, reg2, reg3`	`mov     $label, r1`
	`divh    r1, reg2, reg3`

[Flag]

CY	---
OV	1 if an Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not

SAT	---

[Caution]

- If r0 is specified by the first operand in syntax (1) when the V850Ex is used as the target device, the as850 outputs the following message and stops assembling.

```
E3239: illegal operand (can not use r0 as source in V850E mode)
```

With a device other than the V850Ex, the as850 outputs the following message and continues assembling.

```
W3013: register r0 used as source register
```

- If r0 is specified by the second operand in syntaxes (1) and (2) when the V850Ex is used as the target device, the as850 outputs the following message and stops assembling.

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

With a device other than the V850Ex, the as850 outputs the following message and continues assembling.

```
W3013: register r0 used as destination register
```

divhu

[V850E]

[Overview]

Divide Half-word Unsigned

[Syntax]

```
(1) divhu       reg1, reg2, reg3
(2) divhu       imm, reg2, reg3
```

The following can be specified for imm:

- Absolute expression having a value of up to 16 bits[Note]
- Relative expression

Note The as850 does not check whether the value of the expression exceeds 16 bits.
The generated machine instruction uses only the lower 16 bits for execution.

[Function]

- Syntax (1)

  Divides the register value specified by the second operand by the value of the lower halfword data of the register value specified by the first operand as an unsigned value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.

- Syntax (2)

  Divides the register value specified by the second operand by the value of the lower halfword data of the absolute or relative expression specified by the first operand as an unsigned value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one divhu machine instruction.
- If the instruction is executed in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions[Note].

(a) 0

`divhu    0, reg2, reg3`	`divhu    r0, reg2, reg3`

(b) Absolute expression having a value of other than 0 whithin the range of -16 to +15

`divhu   imm5, reg2, reg3`	`mov     imm5, r1`
	`divhu   r1, reg2, reg3`

(c) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`divhu   imm16, reg2, reg3`	`movea   imm16, r0, r1`
	`divhu   r1, reg2, reg3`

(d) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`divhu   imm, reg2, reg3`	`movhi   hi(imm), r0, r1`
	`divhu   r1, reg2, reg3`

Else

`divhu   imm, reg2, reg3`	`mov     imm, r1`
	`divhu   r1, reg2, reg3`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`divhu   $label, reg2, reg3`	`movea   $label, r0, r1`
	`divhu   r1, reg2, reg3`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`divhu   #label, reg2, reg3`	`mov     #label, r1`
	`divhu   r1, reg2, reg3`

`divhu   label, reg2, reg3`	`mov     label, r1`
	`divhu   r1, reg2, reg3`

`divhu   $label, reg2, reg3`	`mov     $label, r1`
	`divhu   r1, reg2, reg3`

Note The divhu machine instruction does not take an immediate value as an operand.

[Flag]

CY	---
OV	1 if an Integer-Overflow occurs, 0 if not
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

divu

[V850E]

[Overview]

Divide Word Unsigned

[Syntax]

```
(1) divu        reg1, reg2, reg3
(2) divu        imm, reg2, reg3
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  Divides the register value specified by the second operand by the register value specified by the first operand as an unsigned value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.

- Syntax (2)

  Divides the register value specified by the second operand by the value of the absolute or relative expression specified by the first operand as an unsigned value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one divu machine instruction.
- If the instruction is executed in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions[Note] .

Note The divu machine instruction does not take an immediate value as an operand.

(a) 0

`divu    0, reg2, reg3`	`divu    r0, reg2, reg3`

(b) Absolute expression having a value of other than 0 whithin the range of -16 to +15

`divu    imm5, reg2, reg3`	`mov     imm5, r1`
	`divu    r1, reg2, reg3`

(c) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`divu    imm16, reg2, reg3`	`movea   imm16, r0, r1`
	`divu    r1, reg2, reg3`

(d) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`divu    imm, reg2, reg3`	`movhi   hi(imm), r0, r1`
	`divu    r1, reg2, reg3`

Else

`divu    imm, reg2, reg3`	`mov     imm, r1`
	`divu    r1, reg2, reg3`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`divu    $label, reg2, reg3`	`movea   $label, r0, r1`
	`divu    r1, reg2, reg3`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`divu    #label, reg2, reg3`	`mov     #label, r1`
	`divu    r1, reg2, reg3`

`divu    label, reg2, reg3`	`mov     label, r1`
	`divu    r1, reg2, reg3`

`divu    $label, reg2, reg3`	`mov     $label, r1`
	`divu    r1, reg2, reg3`

[Flag]

CY	---
OV	1 if an Integer-Overflow occurs, 0 if not
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

mov

[Overview]

Move

[Syntax]

```
(1) mov        reg1, reg2
(2) mov        imm, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  Stores the value of the register specified by the first operand in the register specified by the second operand.

- Syntax (2)

  Stores the value of the absolute expression or relative expression specified by the first operand in the register specified by the second operand.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one mov machine instruction.

- If the following is specified as imm in syntax (2), the as850 generates one mov machine instruction[Note].

(a) Absolute expression having a value in the range of -16 to +15

`mov     imm5, reg`	`mov     imm5, reg`

Note The mov machine instruction for the V850 is in 16-bit format. A 48-bit format is supported with the V850Ex. For the V850, therefore, this instruction takes a register or immediate value in the range of -16 to +15 (0xfffffff0 to 0xf) as the first operand. For the V850Ex, in addition to these register and immediate values, mov takes an immediate value in the range of -2,147,483,648 to -2,147,483,647 (0x80000000 to 0x7fffffff).

- If the following is specified as imm in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions.

(a) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`mov     imm16, reg`	`movea   imm16, r0, reg`

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`mov imm, reg`	`movhi hi(imm), r0, reg`

Else

`mov imm, reg`	`movhi hi1(imm), r0, r1`
	`movea lo(imm), r1, reg`

(c) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

`mov imm, reg`	`movhi hi(imm), r0, reg`

Else[Note]

`mov imm, reg`	`mov imm, reg`

Note A 16-bit mov instruction is replaced by a 48-bit mov instruction.

(d) Relative expression having !label or %label, or that having $label for a label with a definition in the sdata/sbss-attribute section

`mov %label, reg`	`movea !label, r0, reg`

`mov %label, reg`	`movea %label, r0, reg`

`mov $label, reg`	`movea $label, r0, reg`

(e) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`mov     #label, reg`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, reg`

`mov     label, reg`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, reg`

`mov     $label, reg`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, reg`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section[Note] **[V850E]**

`mov     #label, reg`	`mov     #label, reg`

`mov     label, reg`	`mov     label, reg`

`mov     $label, reg`	`mov     $label, reg`

Note A 16-bit mov instruction is replaced by a 48-bit mov instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If r0 is specified by both the first and the second operand of syntax(1), the result of assembly becomes a nop instruction code.
- When the V850Ex is used as the target device, if an absolute expression having a value in the range between -6 and 15 is specified by the first operand and r0 is specified by the second operand of syntax (2), the as850 outputs the following message and stops assembling.

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

- If an absolute expression having a value exceeding the range of -32,768 to +32,767, #label, or a relative expression having label, and a relative expression having $label without a definition in the sdata/sbss attribute section are specified as the first operand of an instruction in syntax (2), and if instruction expansion is suppressed with quasi directive .option nomacro specified, when the target device is the V850Ex, the as850 outputs the following message and stops assembling.

```
E3249: illegal syntax
```

In this case, use the mov32 instruction.

mov32

[V850E]

[Overview]

32bit Move

[Syntax]

```
(1) mov32       imm, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

Stores the value of the absolute or relative expression specified as the first operand in the register specified as the second operand.

[Description]

- The as850 generates one 48-bit machine language mov instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

movea

[Overview]

Move Effective Address

[Syntax]

```
(1) movea      imm, reg1, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with hi( ), lo( ), or hi1( ) applied

[Function]

Adds the value of the absolute expression, relative expression, or expression with hi( ), lo( ), or hi1( ) applied, specified by the first operand, to the value of the register specified by the second operand, and stores the result in the register specified by the third operand.

[Description]

- If the following is specified for imm, the as850 generates one movea machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`movea   imm16, reg1, reg2`	`movea   imm16, reg1, reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`movea   $label, reg1, reg2`	`movea   $label, reg1, reg2`

(c) Relative expression having !label or %label

`movea   !label, reg1, reg2`	`movea   !label, reg1, reg2`

`movea   %label, reg1, reg2`	`movea   %label, reg1, reg2`

(d) Expression with hi( ) , lo( ) , or hi1( )

`movea imm16, reg1, reg2`	`movea imm16, reg1, reg2`

Note The movea machine instruction takes an immediate value in a range of -32,768 to +32,767 (0xffff8000 to 0x7fff) as the first operand.

- If the following is specified for imm, the as850 executes instruction expansion to generate one or more machine instructions.

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`movea imm, reg1, reg2`	`movhi hi(imm), reg1, reg2`

Else

`movea imm, reg1, reg2`	`movhi hi1(imm), reg1, r1`
	`movea lo(imm), r1, reg2`

(b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`movea #label, reg1, reg2`	`movhi hi1(#label), reg1, r1`
	`movea lo(#label), r1, reg2`

`movea label, reg1, reg2`	`movhi hi1(label), reg1, r1`
	`movea lo(label), r1, reg2`

`movea $label, reg1, reg2`	`movhi hi1($label), reg1, r1`
	`movea lo($label), r1, reg2`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If r0 is specified by the third operand when the V850Ex is used as the target device, the as850 outputs the following message and stops assembling.

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

With a device other than the V850Ex, the as850 outputs the following message and continues assembling

```
W3013: register r0 used as destination register
```

movhi

[Overview]

Move High half-word

[Syntax]

```
(1) movhi      imm16, reg1, reg2
```

The following can be specified for imm16:

- Absolute expression having a value of up to 16 bits
- Relative expression
- Either an absolute expression or relative expression with hi( ), lo( ), or hi1( ) applied

[Function]

Adds word data for which the higher 16 bits are specified by the first operand and the lower 16 bits are 0, to the value of the register specified by the second operand, and stores the result in the register specified by the third operand.

[Description]

- The as850 generates one movhi machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of 0 to 65,535 is specified as imm16, the as850 outputs the following message and stops assembling.

```
E3231: illegal operand (range error in immediate)
```

- If r0 is specified by the third operand when the V850Ex is used as the target device, the as850 outputs the following message and stops assembling.

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

With a device other than the V850Ex, the as850 outputs the following message and continues assembling.

```
W3013: register r0 used as destination register
```

mul

[V850E]

[Overview]

Multiply Word

[Syntax]

```
(1) mul         reg1, reg2, reg3
(2) mul         imm, reg2, reg3
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  Multiplies the register value specified by the first operand by the register value specified by the second operand as a signed value and stores the lower 32 bits of the result in the register specified by the second operand, and the higher 32 bits in the register specified by the third operand. If the same register is specified by the second and third operands, the higher 32 bits of the multiplication result are stored in that register.

- Syntax (2)

  Multiplies the value of the absolute or relative expression specified by the first operand by the register value specified by the second operand as a signed value and stores the lower 32 bits of the result in the register specified by the second operand, and the higher 32 bits in the register specified by the third operand. If the same register is specified by the second and third operands, the higher 32 bits of the multiplication result are stored in that register.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one mul machine instruction.
- If the instruction is executed in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions.

(a) 0

`mul      0, reg2, reg3`	`mul      r0, reg2, reg3`

(b) Absolute expression having a value of other than 0 whithin the range of -256 to +255

`mul      imm9, reg2, reg3`	`mul      imm9, reg2, reg3`

(c) Absolute expression exceeding the range of -256 to +255, but within the range of -32,768 to +32,767

`mul     imm16, reg2, reg3`	`movea   imm16, r0, r1`
	`mul     r1, reg2, reg3`

(d) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`mul     imm, reg2, reg3`	`movhi   hi(imm), r0, r1`
	`mul     r1, reg2, reg3`

Else

`mul     imm, reg2, reg3`	`mov     imm, r1`
	`mul     r1, reg2, reg3`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`mul     $label, reg2, reg3`	`movea   $label, r0, r1`
	`mul     r1, reg2, reg3`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`mul     #label, reg2, reg3`	`mov     #label, r1`
	`mul     r1, reg2, reg3`

`mul     label, reg2, reg3`	`mov     label, r1`
	`mul     r1, reg2, reg3`

`mul     $label, reg2, reg3`	`mov     $label, r1`
	`mul     r1, reg2, reg3`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If these three conditions for the instructions in syntax (1) are met: reg1 and reg3 are the same register, reg2 is a different register from reg1 and reg3, and reg1 and reg3 are neither r0 nor r1, the as850 performs instruction expansion and generates multiple machine-language instructions.

```
    mov     reg1, r1
    mul     r1, reg2, reg3
```

- If these three conditions for the instructions in syntax (1) are met: reg1 and reg3 are the same register, reg2 is a different register from reg1 and reg3, and reg1 and reg3 are r1, the as850 outputs the following messages and stops assembling.

```
W3013: register r1 used as source register
W3013: register r1 used as destination register
E3259: can not use r1 as destination in mul/mulu
```

- If these two conditions for the instructions in syntax (2) are met: reg2 and reg3 are the same register, and reg3 is r1, the as850 outputs the following message and stops assembling.

```
W3013: register r1 used as destination register
E3259: can not use r1 as destination in mul/mulu
```

If the warning message suppressing option -wr1- is specified, the as850 outputs the following message and stops assembling.

```
E3259: can not use r1 as destination in mul/mulu
```

mulh

[Overview]

Multiply Half-word

[Syntax]

```
(1) mulh        reg1, reg2
(2) mulh        imm, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 16 bits[Note]
- Relative expression

Note The as850 does not check whether the value of the expression exceeds 16 bits. The generated mulh instruction performs the operation by using the lower 16 bits.

[Function]

- Syntax (1)

  Multiplies the value of the lower halfword data of the register specified by the first operand by the value of the lower halfword data of the register specified by the second operand as a signed value, and stores the result in the register specified by the second operand.

- Syntax (2)

  Multiplies the value of the lower halfword data of the absolute expression or relative expression specified by the first operand by the value of the lower halfword data of the register specified by the second operand as a signed value, and stores the result in the register specified by the second operand.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one mulh machine instruction.
- If the following is specified as imm in syntax (2), the as850 generates one mulh machine instruction[Note].

(a) Absolute expression having a value in the range of -16 to +15

`mulh    imm15, reg`	`mulh    imm5, reg`

Note The mulh machine instruction takes a register or immediate value in the range of -16 to +15 (0xfffffff0 to 0xf) as the first operand.

- If the following is specified for imm in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions

(a) Absolute expression having a value exceeding the range of -16 to +15

`mulh    imm16, reg`	`mulhi   imm16, reg, reg`

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`mulh    imm, reg`	`movhi   hi(imm), r0, r1`
	`mulh    r1, reg`

Else

`mulh    imm, reg`	`movhi   hi1(imm), r0, r1`
	`movea   lo(imm), r1, r1`
	`mulh    r1, reg`

(c) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

`mulh    imm, reg`	`movhi   hi(imm), r0, r1`
	`mulh    r1, reg`

Else

`mulh    imm, reg`	`mov     imm, r1`
	`mulh    r1, reg`

(d) Relative expression having !label or %label, or that having $label for a label with a definition in the sdata/sbss-attribute section

`mulh    $label, reg`	`mulhi   !label, reg, reg`

`mulh    %label, reg`	`mulhi   %label, reg, reg`

`mulh    $label, reg`	`mulhi   $label, reg, reg`

(e) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

mulh #label, reg	movhi hi1(#label), r0, r1 movea lo(#label), r1, r1 mulh r1, reg

mulh label, reg	movhi hi1(label), r0, r1 movea lo(label), r1, r1 mulh r1, reg

mulh $label, reg	movhi hi1($label), r0, r1 movea lo($label), r1, r1 mulh r1, reg

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

mulh #label, reg	mov #label, r1 mulh r1, reg

mulh label, reg	mov label, r1 mulh r1, reg

mulh $label, reg	mov $label, r1 mulh r1, reg

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If r0 is specified by the second operand when the V850Ex is used as the target device, the as850 outputs the following message and stops assembling.

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

With a device other than the V850Ex, the as850 outputs the following message and continues assembling.

```
W3013: register r0 used as destination register
```

mulhi

[Overview]

Multiply Half-word Immediate

[Syntax]

```
(1) mulhi      imm, reg1, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 16 bits[Note]
- Relative expression
- Either of the above expressions with hi( ), lo( ), or hi1( ) applied

Note The as850 does not check whether the value of the expression exceeds 16 bits. The generated mulhi machine instruction performs the operation by using the lower 16 bits.

[Function]

Multiplies the value of the absolute expression, relative expression, or expression with hi( ), lo( ), or hi1( ) applied specified by the first operand by the value of the register specified by the second operand, and stores the result in the register specified by the third operand.

[Description]

- If the following is specified for imm, the as850 generates one mulhi machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`mulhi   imm16, reg1, reg2`	`mulhi   imm16, reg1, reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`mulhi   $label, reg1, reg2`	`mulhi   $label, reg1, reg2`

(c) Relative expression having !label or %label

`mulhi   !label, reg1, reg2`	`mulhi   !label, reg1, reg2`

`mulhi   %label, reg1, reg2`	`mulhi   %label, reg1, reg2`

(d) Expression with hi( ), lo( ), or hi1( )

`mulhi   imm16, reg1, reg2`	`mulhi   imm16, reg1, reg2`

Note The mulhi machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xffff8000 to 0x7fff) as the first operand.

- If the following is specified for imm, the as850 executes instruction expansion to generate two or more machine instructions

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`mulhi   imm, reg1, reg2`	`movhi   hi(imm), r0, reg2`
	`mulh    reg1, reg2`

If all the lower 16 bits of the value of imm are 0 and when reg2 is r0

`mulhi   imm, reg1, r0`	`movhi   hi(imm), r0, r1`
	`mulh    reg1, r1`

Else

`mulhi   imm, reg1, reg2`	`movhi   hi1(imm), r0, r1`
	`movea   lo(imm), r1, reg2`
	`mulh    reg1, reg2`

Other than above and when reg2 is r0

`mulhi   imm, reg1, r0`	`movhi   hi1(imm), r0, r1`
	`movea   lo(imm), r1, r1`
	`mulh    reg1,r1`

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

`mulhi   imm, reg1, reg2`	`movhi   hi(imm), r0, reg2`
	`mulh    reg1, reg2`

Else

`mulhi   imm, reg1, reg2`	`mov     imm, reg2`
	`mulh    reg1, reg2`

(c) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

If reg2 is r0

mulhi #label, reg1, r0	movhi hi1(#label), r0, r1 movea lo(#label), r1, r1 mulh reg1, reg2

mulhi label, reg1, r0	movhi hi1(label), r0, r1 movea lo(label), r1, r1 mulh reg1, r1

mulhi $label, reg1 r0	movhi hi1($label), r0, r1 movea lo($label), r1, r1 mulh reg1, r1

Else

mulhi #label, reg1, reg2	movhi hi1(#label), r0, r1 movea lo(#label), r1, reg2 mulh reg1, reg2

mulhi label, reg1, reg2	movhi hi1(label), r0, r1 movea lo(label), r1, reg2 mulh reg1, reg2

mulhi $label, reg1 reg2	movhi hi1($label), r0, r1 movea lo($label), r1, reg2 mulh reg1, reg2

(d) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

mulhi #label, reg1, reg2	mov #label, reg2 mulhi reg1, reg2

mulhi label, reg1, reg2	mov label, reg2 mulh reg1, reg2

mulhi $label, reg1, reg2	mov $label, reg2 mulh reg1, reg2

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If r0 is specified by the second operand when the V850Ex is used as the target device, the as850 outputs the following message and stops assembling.

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

With a device other than the V850Ex, the as850 outputs the following message and continues assembling.

```
W3013: register r0 used as destination register
```

mulu

[V850E]

[Overview]

Multiply Word Unsigned

[Syntax]

```
(1) mulu        reg1, reg2, reg3
(2) mulu        imm, reg2, reg3
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  Multiplies the register value specified by the first operand by the register value specified by the second operand as an unsigned value and stores the lower 32 bits of the result in the register specified by the second operand, and the higher 32 bits in the register specified by the third operand. If the same register is specified by the second and third operands, the higher 32 bits of the multiplication result are stored in that register.

- Syntax (2)

  Multiplies the value of the absolute or relative expression specified by the first operand by the register value specified by the second operand as an unsigned value and stores the lower 32 bits of the result in the register specified by the second operand, and the higher 32 bits in the register specified by the third operand. If the same register is specified by the second and third operands, the higher 32 bits of the multiplication result are stored in that register.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one mulu machine instruction.
- If the instruction is executed in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions.

(a) 0

`mulu    0, reg2, reg3`	`mulu    r0, reg2, reg3`

(b) Absolute expression having a value in the range of 1 to +511

`mulu    imm9, reg2, reg3`	`mulu    imm9, reg2, reg3`

(c) Absolute expression exceeding the range of 0 to +511, but within the range of -32,768 to +32,767

`mulu    imm16, reg2, reg3`	`movea   imm16, r0, r1`
	`mulu    r1, reg2, reg3`

(d) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`mulu    imm, reg2, reg3`	`movhi   hi(imm), r0, r1`
	`mulu    r1, reg2, reg3`

Else

`mulu    imm, reg2, reg3`	`mov     imm, r1`
	`mulu    r1, reg2, reg3`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`mulu    $label, reg2, reg3`	`movea   $label, r0, r1`
	`mulu    r1, reg2, reg3`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`mulu    #label, reg2, reg3`	`mov     #label, r1`
	`mulu    r1, reg2, reg3`

`mulu    label, reg2, reg3`	`mov     label, r1`
	`mulu    r1, reg2, reg3`

`mulu    $label, reg2, reg3`	`mov     $label, r1`
	`mulu    r1, reg2, reg3`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If these three conditions for the instructions in syntax (1) are met: reg1 and reg3 are the same register, reg2 is a different register from reg1 and reg3, and reg1 and reg3 are neither r0 nor r1, the as850 performs instruction expansion and generates multiple machine-language instructions.

```
    mov     reg1, r1
    mulu    r1, reg2, reg3
```

- If these three conditions for the instructions in syntax (1) are met: reg1 and reg3 are the same register, reg2 is a different register from reg1 and reg3, and reg1 and reg3 are r1, the as850 outputs the following messages and stops assembling.

```
W3013: register r1 used as source register
W3013: register r1 used as destination register
E3259: can not use r1 as destination in mul/mulu
```

- If these two conditions for the instructions in syntax (2) are met: reg2 and reg3 are the same register, and reg3 is r1, the as850 outputs the following message and stops assembling.

```
W3013: register r1 used as destination register
E3259: can not use r1 as destination in mul/mulu
```

If the warning message suppressing option -wr1- is specified, the as850 outputs the following message and stops assembling.

```
E3259: can not use r1 as destination in mul/mulu
```

mac

[V850E2]

[Overview]

Signed Word Data Multiply and Add (Multiply Word and Add)

[Syntax]

```
(1) mac         reg1, reg2, reg3, reg4
```

[Function]

Adds the multiplication result of the general-purpose register reg2 word data and the general-purpose register reg1 word data with the 64-bit data made up of general-purpose register reg3 as the lower 32 bits and general-purpose register reg3+1 (for example, if reg3 were r6, "reg3+1" would be r7) as the upper 32 bits, and stores the upper 32 bits of that result (64-bit data) in general-purpose register reg4+1 and the lower 32 bits in general-purpose register reg4.

The contents of general-purpose registers reg1 and reg2 are treated as 32-bit signed integers.

General-purpose registers reg1, reg2, reg3, and reg3+1 are unaffected.

[Description]

- The as850 generates one mac machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

The general-purpose registers that can be specified to reg3 or reg4 are limited to even numbered registers (r0, r2, r4, ..., r30). When specifying an odd numbered register, the following message is output, and assembly continues, specifying the register as an even numbered register (r0, r2, r4, ..., r30).

```
W3026: illegal register number, aligned odd register(rXX) to be even register(rYY).
```

macu

[V850E2]

[Overview]

Unsigned Word Data Multiply and Add (Multiply Word Unsigned and Add)

[Syntax]

```
(1) macu        reg1, reg2, reg3, reg4
```

[Function]

Adds the multiplication result of the general-purpose register reg2 word data and the general-purpose register reg1 word data with the 64-bit data made up of general-purpose register reg3 as the lower 32 bits and general-purpose register reg3+1 (for example, if reg3 were r6, "reg3+1" would be r7) as the upper 32 bits, and stores the upper 32 bits of that result (64-bit data) in general-purpose register reg4+1 and the lower 32 bits in general-purpose register reg4.

The contents of general-purpose registers reg1 and reg2 are treated as 32-bit unsigned integers.

General-purpose registers reg1, reg2, reg3, and reg3+1 are unaffected.

[Description]

- The as850 generates one macu machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

The general-purpose registers that can be specified to reg3 or reg4 are limited to even numbered registers (r0, r2, r4, ..., r30). When specifying an odd numbered register, the following message is output, and assembly continues, specifying the register as an even numbered register (r0, r2, r4, ..., r30).

```
W3026: illegal register number, aligned odd register(rXX) to be even register(rYY).
```

sasf

[V850E]

[Overview]

Shift And Set Flag Condition

[Syntax]

```
(1) sasf       imm4, reg
(2) sasfcond   reg
```

The following can be specified for imm4:

- Absolute expression having a value of up to 4 bits

[Function]

- Syntax (1) (sasf)

  Compares the flag condition indicated by the value of the lower 4 bits of the absolute expression specified by the first operand (refer to Table 3 - 4) with the current flag condition. If a match is found, the contents of the register specified by the second operand are shifted logically 1 bit to the left and ORed with 1, and the result stored in the register specified by the second operand; otherwise, the contents of the register specified by the second operand are logically shifted 1 bit to the left and the result stored in the register specified by the second operand.
- Syntax (2) (sasf*cond*)

  Compares the flag condition indicated by string *cond* with the current flag condition. If a match is found, the contents of the register specified by the second operand are shifted logically 1 bit to the left and ORed with 1, and the result stored in the register specified by the second operand; otherwise, the contents of the register specified by the second operand are shifted logically 1 bit to the left and the result stored in the register specified by the second operand.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one sasf machine instruction.
- If the instruction is executed in syntax (2), the as850 generates the corresponding sasf instruction (refer to Table 3 - 4) and expands it to syntax (1).

Table 3 - 4 sasf*cond* Instruction List

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
sasfgt	((S xor OV) or Z) = 0	Greater than (signed)	sasf 0xf
sasfge	(S xor OV) = 0	Greater than or equal (signed)	sasf 0xe
sasflt	(S xor OV) = 1	Less than (signed)	sasf 0x6
sasfle	((S xor OV) or Z) = 1	Less than or equal (signed)	sasf 0x7
sasfh	(CY or Z) = 0	Higher (Greater than)	sasf 0xb
sasfnl	CY = 0	Not lower (Greater than or equal)	sasf 0x9
sasfl	CY = 1	Lower (Less than)	sasf 0x1
sasfnh	(CY or Z) = 1	Not higher (Less than or equal)	sasf 0x3
sasfe	Z = 1	Equal	sasf 0x2
sasfne	Z = 0	Not equal	sasf 0xa
sasfv	OV = 1	Overflow	sasf 0x0
sasfnv	OV = 0	No overflow	sasf 0x8
sasfn	S = 1	Negative	sasf 0x4
sasfp	S = 0	Positive	sasf 0xc
sasfc	CY = 1	Carry	sasf 0x1
sasfnc	CY = 0	No carry	sasf 0x9
sasfz	Z = 1	Zero	sasf 0x2
sasfnz	Z = 0	Not zero	sasf 0xa
sasft	always 1	Always 1	sasf 0x5
sasfsa	SAT = 1	Saturated	sasf 0xd

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding 4 bits is specified as imm4 of the sasf instruction, the as850 outputs the following message and continues assembling using four low-order bits of a specified value.

```
W3011: illegal operand (range error in immediate).
```

setf

[Overview]

Set Flag Condition

[Syntax]

```
(1) setf       imm4, reg
(2) setfcond   reg
```

The following can be specified for imm4:

- Absolute expression having a value of up to 4 bits

[Function]

- Syntax (1) (setf)

  Compares the status of the flag specified by the value of the lower 4 bits of the absolute expression specified by the first operand with the current flag condition. If they are found to match, 1 is stored in the register specified by the second operand; otherwise, 0 is stored in the register specified by the second operand.

- Syntax (2) (setf*cond*)

  Compares the status of the flag indicated by string *cond* (refer to Table 3 - 5) with the current flag condition. If they are found to match, 1 is stored in the register specified by the second operand; otherwise, 0 is stored in the register specified by the second operand.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one sasf machine instruction.

- If the instruction is executed in syntax (2), the as850 generates the corresponding setf instruction (refer to Table 3 - 5) and expands it to syntax (1).

Table 3 - 5 setf*cond* Instruction List

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
setfgt	((S xor OV) or Z) = 0	Greater than (signed)	setf 0xf
setfge	(S xor OV) = 0	Greater than or equal (signed)	setf 0xe
setflt	(S xor OV) = 1	Less than (signed)	setf 0x6
setfle	((S xor OV) or Z) = 1	Less than or equal (signed)	setf 0x7
setfh	(CY or Z) = 0	Higher (Greater than)	setf 0xb
setfnl	CY = 0	Not lower (Greater than or equal)	setf 0x9
setfl	CY = 1	Lower (Less than)	setf 0x1
setfnh	(CY or Z) = 1	Not higher (Less than or equal)	setf 0x3
setfe	Z = 1	Equal	setf 0x2
setfne	Z = 0	Not equal	setf 0xa
setfv	OV = 1	Overflow	setf 0x0
setfnv	OV = 0	No overflow	setf 0x8
setfn	S = 1	Negative	setf 0x4
setfp	S = 0	Positive	setf 0xc
setfc	CY = 1	Carry	setf 0x1
setfnc	CY = 0	No carry	setf 0x9
setfz	Z = 1	Zero	setf 0x2
setfnz	Z = 0	Not zero	setf 0xa
setft	always 1	Always 1	setf 0x5
setfsa	SAT = 1	Saturated	setf 0xd

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding 4 bits is specified as imm4 of the setf instruction, the as850 outputs the following message and continues assembling using four low-order bits of a specified value.

```
W3011: illegal operand (range error in immediate).
```

sub

[Overview]

Subtract

[Syntax]

```
(1) sub         reg1, reg2
(2) sub         imm, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  Subtracts the value of the register specified by the first operand from the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

- Syntax (2)

  Subtracts the value of the absolute expression or relative expression specified by the first operand from the value of the register specified by the second operand, and stores the result into the register specified by the second operand.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one sub machine instruction.
- If the instruction is executed in syntax (2), the as850 executes instruction expansion and generates one or more machine instructions[Note] .

(a) 0

`sub     0, reg`	`sub     r0, reg`

(b) Absolute expression having a value of other than 0 whithin the range of -16 to +15

`sub     imm5, reg`	`mov     imm5, r1`
	`sub     r1, reg`

(c) Absolute expression having a value of other than 0 whithin the range of -16 to +15 **[V850E]**

`sub     imm5, reg`	`mov     imm5, r1`
	`sub     r1, reg`

(d) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`sub     imm16, reg`	`movea   imm16, r0, r1`
	`sub     r1, reg`

(e) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`sub     imm, reg`	`movhi   hi(imm), r0, r1`
	`sub     r1, reg`

Else

`sub     imm, reg`	`movhi   hi1(imm), r0, r1`
	`movea   lo(imm), r1, r1`
	`sub     r1, reg`

(f) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

`sub     imm, reg`	`movhi   hi(imm), r0, r1`
	`sub     r1, reg`

Else

`sub     imm, reg`	`mov     imm, r1`
	`sub     r1, reg`

(g) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`sub     $label, reg`	`movea   $label, r0, r1`
	`sub     r1, reg`

(h) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`sub     #label, reg`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, r1`
	`sub     r1, reg`

`sub     label, reg`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, r1`
	`sub     r1, reg`

`sub     $label, reg`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, r1`
	`sub     r1, reg`

(i) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

`sub     #label, reg`	`mov     #label, r1`
	`sub     r1, reg`

`sub     label, reg`	`mov     label, r1`
	`sub     r1, reg`

`sub     $label, reg`	`mov     $label, r1`
	`sub     r1, reg`

Note The sub machine instruction does not take an immediate value as an operand.

[Flag]

CY	1 if a borrow occurs from MSB (Most Significant Bit) , 0 if not
OV	1 if an Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

subr

[Overview]

Subtract Reverse

[Syntax]

```
(1) subr        reg1, reg2
(2) subr        imm, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  Subtracts the value of the register specified by the first operand from the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

- Syntax (2)

  Subtracts the value of the absolute expression or relative expression specified by the first operand from the value of the register specified by the second operand, and stores the result into the register specified by the second operand.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one subr machine instruction.
- If the instruction is executed in syntax (2), the as850 executes instruction expansion and generates one or more machine instructions[Note] .

(a) 0

`subr    0, reg`	`subr    r0, reg`

(b) Absolute expression having a value of other than 0 whithin the range of -16 to +15

`subr    imm5, reg`	`mov     imm5, r1`
	`subr    r1, reg`

(c) Absolute expression having a value of other than 0 whithin the range of -16 to +15 **[V850E]**

`subr    imm5, reg`	`mov     imm5, r1`
	`subr    r1, reg`

(d) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`subr    imm16, reg`	`movea   imm16, r0, r1`
	`subr    r1, reg`

(e) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`subr    imm, reg`	`movhi   hi(imm), r0, r1`
	`subr    r1, reg`

Else

`subr    imm, reg`	`movhi   hi1(imm), r0, r1`
	`movea   lo(imm), r1, r1`
	`subr    r1, reg`

(f) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

`subr    imm, reg`	`movhi   hi(imm), r0, r1`
	`subr    r1, reg`

Else

`subr    imm, reg`	`mov     imm, r1`
	`subr    r1, reg`

(g) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`subr    $label, reg`	`movea   $label, r0, r1`
	`subr    r1, reg`

(h) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

subr #label, reg	movhi hi1(#label), r0, r1 movea lo(#label), r1, r1 subr r1, reg

subr label, reg	movhi hi1(label), r0, r1 movea lo(label), r1, r1 subr r1, reg

subr $label, reg	movhi hi1($label), r0, r1 movea lo($label), r1, r1 subr r1, reg

(i) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

subr #label, reg	mov #label, r1 subr r1, reg

subr label, reg	mov label, r1 subr r1, reg

subr $label, reg	mov $label, r1 subr r1, reg

Note The subr machine instruction does not take an immediate value as an operand.

[Flag]

CY	1 if a borrow occurs from MSB (Most Significant Bit) , 0 if not
OV	1 if an Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

adf

[V850E2]

[Overview]

Add with Condition Flag (Add on Condition Flag)

[Syntax]

```
(1) adf         imm4, reg1, reg2, reg3
(2) adfcond     reg1, reg2, reg3
```

The following can be specified for imm4:

- Absolute expression having a value up to 4 bits (0xd cannot be specified)

[Function]

- adf

  Adds the word data of the register specified by the second operand to the word data of the register specified by the third operand.

  It then compares the flag condition of the addition result with the flag condition indicated by the value of the lower 4 bits of the absolute expression (refer to Table 3 - 6) specified by the first operand. If the values match, 1 is added to the addition result and that result is stored in the register specified by the fourth operand; otherwise, 0 is added to the addition result and that result is stored in the register specified by the fourth operand.

- adf*cond*

  Adds the word data of the register specified by the first operand to the word data of the register specified by the second operand.

  It then compares the flag condition of the addition result with the flag condition indicated by the string in the *cond*"part. If the values match, 1 is added to the addition result and that result is stored in the register specified by the third operand; otherwise, 0 is added to the addition result and that result is stored in the register specified by the third operand.

[Description]

- For the adf instruction, the as850 generates one adf machine instruction.

- For the adcond instruction, the as850 generates the corresponding adf instruction (refer to Table 3 - 6) and expands it to syntax (1).

Table 3 - 6 adf*cond* Instruction List

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
adfgt	((S xor OV)or Z) = 0	Greater than (signed)	adf 0xf
adfge	(S xor OV)= 0	Greater than or equal (signed)	adf 0xe
adflt	(S xor OV)= 1	Less than (signed)	adf 0x6
adfle	((S xor OV)or Z) = 1	Less than or equal (signed)	adf 0x7
adfh	(CY or Z) = 0	Higher (Greater than)	adf 0xb
adfnl	CY = 0	Not lower (Greater than or equal)	adf 0x9
adfl	CY = 1	Lower (Less than)	adf 0x1
adfnh	(CY or Z) = 1	Not higher (Less than or equal)	adf 0x3
adfe	Z = 1	Equal	adf 0x2
adfne	Z = 0	Not equal	adf 0xa
adfv	OV = 1	Overflow	adf 0x0
adfnv	OV = 0	No overflow	adf 0x8
adfn	S = 1	Negative	adf 0x4
adfp	S = 0	Positive	adf 0xc
adfc	CY = 1	Carry	adf 0x1
adfnc	CY = 0	No carry	adf 0x9
adfz	Z = 1	Zero	adf 0x2
adfnz	Z = 0	Not zero	adf 0xa
adft	always 1	Always 1	adf 0x5

[Flag]

CY	1 if there is carry from MSB, 0 if not
OV	1 if overflow occurred, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

[Caution]

- If an absolute expression having a value exceeding 4 bits is specified as imm4 of the adf instruction, the following message is output, and assembly continues using the lower 4 bits of the specified value.

```
W3011: illegal operand (range error in immediate).
```

- If 0xd is specified as imm4 of the adf instruction, the following message is output, and assembly is stopped.

```
E3261: illegal condition code
```

sbf

[V850E2]

[Overview]

Subtract with Condition Flag (Subtract on Condition Flag)

[Syntax]

```
(1) sbf          imm4, reg1, reg2, reg3
(2) sbfcond      reg1, reg2, reg3
```

The following can be specified for imm4:

- Absolute expression having a value up to 4 bits (0xd cannot be specified)

[Function]

- sbf

  Subtracts the word data of the register specified by the second operand from the word data of the register specified by the third operand.

  It then compares the flag condition of the subtraction result with the flag condition indicated by the value of the lower 4 bits of the absolute expression (refer to Table 3 - 7) specified by the first operand. If the values match, 1 is subtracted from the subtraction result and that result is stored in the register specified by the fourth operand; otherwise, 0 is subtracted from the subtraction result and that result is stored in the register specified by the fourth operand.

- sbf*cond*

  Subtracts the word data of the register specified by the first operand from the word data of the register specified by the second operand.

  It then compares the flag condition of the subtraction result with the flag condition indicated by the string in the "cond" part. If the values match, 1 is subtracted from the subtraction result and that result is stored in the register specified by the third operand; otherwise, 0 is subtracted from the subtraction result and that result is stored in the register specified by the third operand.

[Description]

- For the sbf instruction, the as850 generates one sbf machine instruction.

- For the adcond instruction, the as850 generates the corresponding sbf instruction (refer to Table 3 - 7) and expands it to syntax (1).

Table 3 - 7 sbf*cond* Instruction List

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
sbfgt	((S xor OV)or Z) = 0	Greater than (signed)	sbf 0xf
sbfge	(S xor OV)= 0	Greater than or equal (signed)	sbf 0xe
sbflt	(S xor OV)= 1	Less than (signed)	sbf 0x6
sbfle	((S xor OV)or Z) = 1	Less than or equal (signed)	sbf 0x7
sbfh	(CY or Z) = 0	Higher (Greater than)	sbf 0xb
sbfnl	CY = 0	Not lower (Greater than or equal)	sbf 0x9
sbfl	CY = 1	Lower (Less than)	sbf 0x1
sbfnh	(CY or Z) = 1	Not higher (Less than or equal)	sbf 0x3
sbfe	Z = 1	Equal	sbf 0x2
sbfne	Z = 0	Not equal	sbf 0xa
sbfv	OV = 1	Overflow	sbf 0x0
sbfnv	OV = 0	No overflow	sbf 0x8
sbfn	S = 1	Negative	sbf 0x4
sbfp	S = 0	Positive	sbf 0xc
sbfc	CY = 1	Carry	sbf 0x1
sbfnc	CY = 0	No carry	sbf 0x9
sbfz	Z = 1	Zero	sbf 0x2
sbfnz	Z = 0	Not zero	sbf 0xa
sbft	always 1	Always 1	sbf 0x5

[Flag]

CY	1 if a borrow occurs from MSB (Most Significant Bit) , 0 if not
OV	1 if overflow occurred, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

[Caution]

- If an absolute expression having a value exceeding 4 bits is specified as imm4 of the sbf instruction, the following message is output, and assembly continues using the lower 4 bits of the specified value.

```
W3011: illegal operand (range error in immediate).
```

- If 0xd is specified as imm4 of the sbf instruction, the following message is output, and assembly is stopped.

```
E3261: illegal condition code
```

3.4 Saturation Operation Instructions

This section describes the saturation operation instructions.

Next table lists the instructions described in this section.

Table 3 - 8 Saturation Operation Instructions

Instruction	Meaning
satadd	Saturated addition
satsub	Saturated subtraction
satsubi	Saturated subtraction (immediate)
satsubr	Reverse subtraction with saturation

satadd

[Overview]

Saturated Add

[Syntax]

```
(1) satadd     reg1, reg2
(2) satadd     imm, reg2
(3) satadd     reg1, reg2, reg3
```
[V850E2]

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

Adds the value of the register specified by the first operand to the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

If the result exceeds the maximum positive value of 0x7fffffff, however, 0x7fffffff is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the second operand. In both cases, the SAT flag is set to 1.

- Syntax (2)

Adds the value of the absolute expression or relative expression specified by the first operand to the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

If the result exceeds the maximum positive value of 0x7fffffff, however, 0x7fffffff is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the second operand. In both cases, the SAT flag is set to 1.

- Syntax (3)

Adds the value of the register specified by the first operand to the value of the register specified by the second operand, and stores the result in the register specified by the third operand.

If the result exceeds the maximum positive value of 0x7fffffff, however, 0x7fffffff is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the third operand. In both cases, the SAT flag is set to 1.

[Description]

- If the instruction is executed in syntax (1) or (3), the as850 generates one satadd machine instruction.

- If the following is specified for imm in syntax (2), the as850 generates one satadd machine instruction[Note].

(a) Absolute expression having a value in the range of -16 to +15

`satadd imm5, reg`	`satadd imm5, reg`

Note The satadd machine instruction takes a register or immediate value in the range of -16 to +15 (0xfffffff0 to 0xf) as the first operand.

- If the following is specified for imm in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions.

(a) Absolute expression having a value exceeding the range of -16 to +15

`satadd imm16, reg`	`movea   imm16, r0, r1`
	`satadd  r1, reg`

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`satadd imm, reg`	`movhi   hi(imm), r0, r1`
	`satadd  r1, reg`

Else

`satadd imm, reg`	`movhi   hi1(imm), r0, r1`
	`movea   lo(imm), r1, r1`
	`satadd  r1, reg`

(c) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

`satadd imm, reg`	`movhi   hi(imm), r0, r1`
	`satadd  r1, reg`

Else

`satadd imm, reg`	`mov     imm, r1`
	`satadd  r1, reg`

(d) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`satadd $label, reg`	`movea   $label, r0, r1`
	`satadd  r1, reg`

(e) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`satadd #label, reg`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, r1`
	`satadd  r1, reg`

`satadd label, reg`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, r1`
	`satadd  r1, reg`

`satadd $label, reg`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, r1`
	`satadd  r1, reg`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

`satadd #label, reg`	`mov     #label, r1`
	`satadd  r1, reg`

`satadd label, reg`	`mov     label, r1`
	`satadd  r1, reg`

`satadd $label, reg`	`mov     $label, r1`
	`satadd  r1, reg`

[Flag]

CY	1 if a carry occurs from MSB (Most Significant Bit) , 0 if not
OV	1 if an Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	1 if OV = 1, - if not

[Caution]

- If the instruction is executed in syntax (1) or (2), if the target device is V850Ex and r0 is specified as the second operand, the following message is output and assembly is stopped.

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

With a device other than the V850Ex, the as850 outputs the following message and continues assembling.

```
W3013: register r0 used as destination register
```

satsub

[Overview]

Saturated Subtract

[Syntax]

```
(1) satsub      reg1, reg2
(2) satsub      imm, reg2
(3) satsub      reg1, reg2, reg3
```
[V850E2]

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

Subtracts the value of the register specified by the first operand from the value of the register specified by the second operand, and stores the result in the register specified by the third operand.

If the result exceeds the maximum positive value of 0x7fffffff, however, 0x7fffffff is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the second operand. In both cases, the SAT flag is set to 1.

- Syntax (2)

Subtracts the value of the absolute expression or relative expression specified by the first operand from the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

If the result exceeds the maximum positive value of 0x7fffffff, however, 0x7fffffff is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the second operand. In both cases, the SAT flag is set to 1.

- Syntax (3)

Subtracts the value of the register specified by the first operand from the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

If the result exceeds the maximum positive value of 0x7fffffff, however, 0x7fffffff is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the third operand. In both cases, the SAT flag is set to 1.

[Description]

- If the instruction is executed in syntax (1) or (3), the as850 generates one satsub machine instruction.
- If the instruction is executed in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions[Note]

(a) 0

`satsub 0, reg`	`satsub r0, reg`

(b) Absolute expression having a value in the range of -32,768 to +32,767

`satsub imm16, reg`	`satsubi imm16, reg, reg`

(c) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`satsub imm, reg`	`movhi  hi(imm), r0, r1`
	`satsub r1, reg`

Else

`satsub imm, reg`	`movhi  hi1(imm), r0, r1`
	`movea  lo(imm), r1, r1`
	`satsub r1, reg`

(d) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

`satsub imm, reg`	`movhi  hi(imm), r0, r1`
	`satsub r1, reg`

Else

`satsub imm, reg`	`mov    imm, r1`
	`satsub r1, reg`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`satsub $label, reg`	`satsubi $label, reg, reg`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`satsub #label, reg`	`movhi  hi1(#label), r0, r1`
	`movea  lo(#label), r1, r1`
	`satsub r1, reg`

`satsub label, reg`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, r1`
	`satsub  r1, reg`

`satsub $label, reg`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, r1`
	`satsub  r1, reg`

(g) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

`satsub #label, reg`	`mov     #label, r1`
	`satsub  r1, reg`

`satsub label, reg`	`mov     label, r1`
	`satsub  r1, reg`

`satsub $label, reg`	`mov     $label, r1`
	`satsub  r1, reg`

Note The satsub machine instruction does not take an immediate value as an operand.

[Flag]

CY	1 if a carry occurs from MSB (Most Significant Bit) , 0 if not
OV	1 if an Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	1 if OV = 1, - if not

[Caution]

- If the instruction is executed in syntax (1) or (2), if the target device is V850Ex and r0 is specified as the second operand, the following message is output and assembly is stopped.

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

With a device other than the V850Ex, the as850 outputs the following message and continues assembling.

```
W3013: register r0 used as destination register
```

satsubi

[Overview]

Saturated Subtract Immediate

[Syntax]

```
(1) satsubi    imm, reg1, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with hi( ), lo( ), or hi1( ) applied

[Function]

Subtracts the value of the absolute expression, relative expression, or expression with hi( ), lo( ), or hi1( ) applied specified by the first operand from the value of the register specified by the second operand, and stores the result in the register specified by the third operand.

If the result exceeds the maximum positive value of 0x7fffffff, however, 0x7fffffff is stored in the register specified by the third operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the third operand. In both cases, the SAT flag is set to 1.

[Description]

- If the following is specified for imm, the as850 generates one satsubi machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`satsubi imm16, reg1, reg2`	`satsubi imm16, reg1, reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`satsubi $label, reg1, reg2`	`satsubi $label, reg1, reg2`

(c) Relative expression having !label or %label

`satsubi !label, reg1, reg2`	`satsubi !label, reg1, reg2`

`satsubi %label, reg1, reg2`	`satsubi %label, reg1, reg2`

(d) Expression with hi( ), lo( ), or hi1( )

`satsubi imm16, reg1, reg2`	`satsubi imm16, reg1, reg2`

Note The satsubi machine instruction takes an immediate value, in the range of -32,768 to +32,767 (0xffff8000 to 0x7fff), as the first operand.

- If the following is specified for imm, the as850 executes instruction expansion to generate one or more machine instructions.

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`satsubi imm, reg1, reg2`	`movhi   hi(imm), r0, reg2`
	`satsubr reg1, reg2`

If all the lower 16 bits of the value of imm are 0 and when reg2 is r0

`satsubi imm, reg1, r0`	`movhi   hi(imm), r0, r1`
	`satsubr reg1, r1`

Else

`satsubi imm, reg1, reg2`	`movhi   hi1(imm), r0, r1`
	`movea   lo(imm), r1, reg2`
	`satsubr reg1, reg2`

Other than above and when reg2 is r0

`satsubi imm, reg1, r0`	`movhi   hi1(imm), r0, r1`
	`movea   lo(imm), r1, r1`
	`satsubr reg1, r1`

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

`satsubi imm, reg1, reg2`	`movhi   hi(imm), r0, reg2`
	`satsubr reg1, reg2`

If all the lower 16 bits of the value of imm are 0 and when reg2 is r0

`satsubi imm, reg1, r0`	`movhi   hi(imm), r0, r1`
	`satsubr reg1, r1`

Else

`satsubi imm, reg1, reg2`	`mov     imm, reg2`
	`satsubr reg1, reg2`

Other than above and when reg2 is r0

`satsubi imm, reg1, r0`	`mov     imm, r1`
	`satsubr reg1, r1`

(c) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

If reg2 is r0

`satsubi #label, reg1, r0`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, r1`
	`satsubr reg1, r1`

`satsubi label, reg1, r0`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, r1`
	`satsubr reg1, r1`

`satsubi $label, reg1, r0`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, r1`
	`satsubr reg1, r1`

Else

`satsubi #label, reg1, reg2`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, reg2`
	`satsubr reg1, reg2`

`satsubi label, reg1, reg2`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, reg2`
	`satsubr reg1, reg2`

`satsubi $label, reg1, reg2`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, reg2`
	`satsubr reg1, reg2`

(d) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

If reg2 is r0

`satsubi #label, reg1, r0`	`movhi   #label, r1`
	`satsubr reg1, r1`

`satsubi label, reg1, r0`	`mov     label, r1`
	`satsubr reg1, r1`

`satsubi $label, reg1, r0`	`mov     $label, r1`
	`satsubr reg1, r1`

Else

`satsubi #label, reg1, reg2`	`movhi   #label, reg2`
	`satsubr reg1, reg2`

`satsubi label, reg1, reg2`	`mov     label, reg2`
	`satsubr reg1, reg2`

`satsubi $label, reg1, reg2`	`mov     $label, reg2`
	`satsubr reg1, reg2`

[Flag]

CY	1 if a carry occurs from MSB (Most Significant Bit) , 0 if not
OV	1 if an Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	1 if OV = 1, - if not

[Caution]

- If r0 is specified by the second operand when the V850Ex is used as the target device, the as850 outputs the following message and stops assembling.

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

With a device other than the V850Ex, the as850 outputs the following message and continues assembling.

```
W3013: register r0 used as destination register
```

satsubr

[Overview]

Saturated Subtract Reverse

[Syntax]

```
(1) satsubr    reg1, reg2
(2) satsubr    imm, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

Subtracts the value of the register specified by the second operand from the value of the register specified by the first operand, and stores the result in the register specified by the second operand.

If the result exceeds the maximum positive value of 0x7fffffff, however, 0x7fffffff is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the second operand. In both cases, the SAT flag is set to 1.

- Syntax (2)

Subtracts the value of the register specified by the second operand from the value of the absolute expression or relative expression specified by the first operand, and stores the result in the register specified by the second operand.

If the result exceeds the maximum positive value of 0x7fffffff, however, 0x7fffffff is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the second operand. In both cases, the SAT flag is set to 1.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one satsubr machine instruction.
- If the instruction is executed in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions[Note].

(a) 0

`satsubr 0, reg`	`satsubr r0, reg`

(b) Absolute expression having a value of other than 0 whithin the range of -16 to +15

`satsubr imm5, reg`	`mov     imm5, r1`
	`satsubr r1, reg`

(c) Absolute expression having a value of other than 0 whithin the range of -16 to +15 **[V850E]**

`satsubr imm5, reg`	`mov     imm5, r1`
	`satsubr r1, reg`

(d) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`satsubr imm16, reg`	`movea   imm16, r0, r1`
	`satsubr r1, reg`

(e) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`satsubr imm, reg`	`movhi   hi(imm), r0, r1`
	`satsubr r1, reg`

Else

`satsubr imm, reg`	`movhi   hi1(imm), r0, r1`
	`movea   lo(imm), r1, r1`
	`satsubr r1, reg`

(f) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

`satsubr imm, reg`	`movhi   hi(imm), r0, r1`
	`satsubr r1, reg`

Else

`satsubr imm, reg`	`mov     imm, r1`
	`satsubr r1, reg`

(g) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`satsubr $label, reg`	`movea   $label, r0, r1`
	`satsubr r1, reg`

(h) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

satsubr #label, reg	movhi hi1(#label), r0, r1 movea lo(#label), r1, r1 satsubr r1, reg

satsubr label, reg	movhi hi1(label), r0, r1 movea lo(label), r1, r1 satsubr r1, reg

satsubr $label, reg	movhi hi1($label), r0, r1 movea lo($label), r1, r1 satsubr r1, reg

(i) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

satsubr #label, reg	mov #label, r1 satsubr r1, reg

satsubr label, reg	mov label, r1 satsubr r1, reg

satsubr $label, reg	mov $label, r1 satsubr r1, reg

Note The satsubr machine instruction does not take an immediate value as an operand.

[Flag]

CY	1 if a carry occurs from MSB (Most Significant Bit) , 0 if not
OV	1 if an Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	1 if OV = 1, - if not

[Caution]

- If r0 is specified by the second operand when the V850Ex is used as the target device, the as850 outputs the following message and stops assembling.

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

- If r0 is specified by the second operand when the V850Ex is used as the target device, the as850 outputs the following message and stops assembling.

```
W3013: register r0 used as destination register
```

3.5 Logical Instructions

This section describes the logical instructions.

Next tablelists the instructions described in this section.

Table 3 - 9 Logical Instructions

Instruction	Meaning
and	Logical product
andi	Logical product (immediate)
bsh	Byte swap of halfword data **[V850E]**
bsw	Byte swap of word data **[V850E]**
hsh	Half-word data half-word swap **[V850E2]**
hsw	Halfword swap of word data **[V850E]**
not	Logical negation (takes 1’s complement)
or	Logical sum
ori	Logical sum (immediate)
sar	Arithmetic right shif
shl	Logical left shift
shr	Logical right shif
sxb	Sign extension of byte data **[V850E]**
sxh	Sign extension of halfword data **[V850E]**
tst	Test
xor	Exclusive OR
xori	Exclusive OR (immediate)
zxb	Zero extension of byte data **[V850E]**
zxh	Zero extension of halfword data **[V850E]**
sch0l	Bit (0) search from MSB side **[V850E2]**
sch0r	Bit (0) search from LSB side **[V850E2]**
sch1l	Bit (1) search from MSB side **[V850E2]**
sch1r	Bit (1) search from LSB side **[V850E2]**

and

[Overview]

And

[Syntax]

```
(1) and         reg1, reg2
(2) and         imm, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  ANDs the value of the register specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

- Syntax (2)

  ANDs the value of the absolute expression or relative expression specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

[Description]

- When this instruction is executed in syntax (1), the as850 generates one and machine instruction.
- When this instruction is executed in syntax (2), the as850 executes instruction expansion to generate one or more machine instruction[Note]

(a) 0

`and     0, reg`	`and     r0, reg`

(b) Absolute expression having a value in the range of +1 to +65,535

`and     imm16, reg`	`andi    imm16, reg, reg`

(c) Absolute expression having a value in the range of -16 to -1

`and     imm5, reg`	`mov     imm5, r1`
	`and     r1, reg`

(d) Absolute expression having a value in the range of -32,768 to -17

and imm16, reg	movea imm16, r0, r1 and r1, reg

(e) Absolute expression exceeding the above ranges

If all the lower 16 bits of the value of imm are 0

and imm, reg	movhi hi(imm), r0, r1 and r1, reg

Else

and imm, reg	movhi hi1(imm), r0, r1 movea lo(imm), r1, r1 and r1, reg

(f) Absolute expression exceeding the above ranges **[V850E]**

If all the lower 16 bits of the value of imm are 0

and imm, reg	movhi hi(imm), r0, r1 and r1, reg

Else

and imm, reg	mov imm, r0, r1 and r1, reg

(g) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

and $label, reg	movea $label, r0, r1 and r1, reg

(h) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`and     #label, reg`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, r1`
	`and     r1, reg`

`and     label, reg`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, r1`
	`and     r1, reg`

`and     $label, reg`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, r1`
	`and     r1, reg`

(i) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

`and     #label, reg`	`mov     #label, r1`
	`and     r1, reg`

`and     label, reg`	`mov     label, r1`
	`and     r1, reg`

`and     $label, reg`	`mov     $label, r1`
	`and     r1, reg`

Note The and machine instruction does not take an immediate value as an operand.

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

andi

[Overview]

And Immediate

[Syntax]

```
(1) andi        imm, reg1, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with hi( ), lo( ), or hi1( ) applied

[Function]

ANDs the value of the absolute expression, relative expression, or expression with hi( ), lo( ), or hi1( ) applied specified by the first operand with the value of the register specified by the second operand, and stores the result into the register specified by the third operand.

[Description]

- If the following is specified as imm, the as850 generates one andi machine instruction[Note].

(a) Absolute expression having a value in the range of 0 to 65,535

`andi    imm16, reg1, reg2`	`andi    imm16, reg1, reg2`

(b) Relative expression having !label or %label

`andi    !label, reg1, reg2`	`andi    !label, reg1, reg2`

`andi    %label, reg1, reg2`	`andi    %label, reg1, reg2`

(c) Expression with hi( ), lo( ), or hi1( )

`andi    imm16, reg1, reg2`	`andi    imm16, reg1, reg2`

Note The andi machine instruction takes an immediate value of 0 to 65,535 (0 to 0xffff) as the first operand

- If the following is specified for imm, the as850 executes instruction expansion to generate one or more machine instructions.

(a) Absolute expression having a value in the range of -16 to -1

andi imm5, reg1, reg2	mov imm5, reg2 and reg1, reg2

(b) Absolute expression having a value in the range of -32,768 to -17

If reg2 is r0

andi imm16, reg1, r0	movea imm16, r0, r1 and reg1, r1

Else

andi imm16, reg1, reg2	movea imm16, r0, reg2 and reg1, reg2

(c) Absolute expression exceeding the above ranges

If all the lower 16 bits of the value of imm are 0

andi imm, reg1, reg2	movhi hi(imm), r0, reg2 and reg1, reg2

If all the lower 16 bits of the value of imm are 0 and when reg2 is r0

andi imm, reg1, r0	movhi hi(imm), r0, r1 and reg1, r1

Else

andi imm, reg1, reg2	movhi hi1(imm), r0, r1 movea lo(imm), r1, reg2 and reg1, reg2

Other than above and when reg2 is r0

andi imm, reg1, r0	movhi hi1(imm), r0, r1 movea lo(imm), r1, r1 and reg1, r1

(d) Absolute expression exceeding the above ranges **[V850E]**

If all the lower 16 bits of the value of imm are 0

andi imm, reg1, reg2	movhi hi(imm), r0, reg2 and reg1, reg2

If all the lower 16 bits of the value of imm are 0 and when reg2 is r0

andi imm, reg1, r0	movhi hi(imm), r0, r1 and reg1, r1

Else

andi imm, reg1, reg2	mov imm, reg2 and reg1, reg2

Other than above and when reg2 is r0

andi imm, reg1, r0	mov imm, r1 and reg1, r1

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

If reg2 is r0

andi $label, reg1, r0	movea $label, r0, r1 and reg1, r1

Else

andi $label, reg1, reg2	movea $label, r0, reg2 and reg1, reg2

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

If reg2 is r0

andi #label, reg1, r0	movhi hi1(#label), r0, r1 movea lo(#label), r1, r1 and reg1, reg2

andi label, reg1, r0	movhi hi1(label), r0, r1 movea lo(label), r1, r1 and reg1, r1

andi $label, reg1, r0	movhi hi1(label), r0, r1 movea lo(label), r1, r1 and reg1, r1

Else

`andi    #label, reg1, reg2`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, reg2`
	`and     reg1, reg2`

`andi    label, reg1, reg2`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, reg2`
	`and     reg1, reg2`

`andi    $label, reg1, reg2`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, reg2`
	`and     reg1, reg2`

(g) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

If reg2 is r0

`andi    #label, reg1, r0`	`mov     #label, r1`
	`and     reg1, r1`

`andi    label, reg1, r0`	`mov     label, r1`
	`and     reg1, r1`

`andi    $label, reg1, r0`	`mov     $label, r1`
	`and     reg1, r1`

Else

`andi    #label, reg1, reg2`	`mov     #label, reg2`
	`and     reg1, reg2`

`andi    label, reg1, reg2`	`mov     label, reg2`
	`and     reg1, reg2`

`andi    $label, reg1, reg2`	`mov     $label, reg2`
	`and     reg1, reg2`

[Flag]

CY	---
OV	0
S	1 if the result MSB is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

bsh

[V850E]

[Overview]

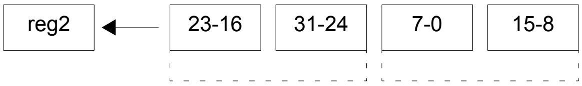
Byte Swap Half-word

[Syntax]

```
(1) bsh         reg1, reg2
```

[Function]

Byte-swaps the register value specified by the first operand in halfword units and stores the result in the register specified by the second operand.



Byte-swap of reg1 in halfword units
(numbers indicate bit numbers)

[Description]

- The as850 generates one bsh machine instruction.

[Flag]

CY	1 if either or both of the bytes in the lower halfword of the register is 0, 0 if not
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the lower half-word data of the result is 0, 0 if not
SAT	---

bsw

[V850E]

[Overview]

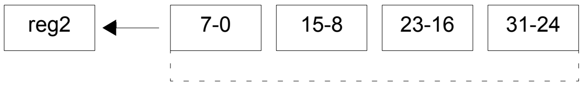
Byte Swap Word

[Syntax]

```
(1) bsw        reg1, reg2
```

[Function]

Byte-swaps the register value specified by the first operand and stores the result in the register specified by the second operand.



Byte-swap of reg1 for entire word
(numbers indicate bit numbers)

[Description]

- The as850 generates one bsw machine instruction.

[Flag]

CY	1 if one or more bytes of the word in the register is 0, 0 if not
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the word data of the result is 1, 0 if not
SAT	---

hsh

[V850E2]

[Overview]

Half-word Data Half-word Swap (Half-word Swap Half-word)

[Syntax]

```
(1) hsh        reg2, reg3
```

[Function]

Stores the register value specified by the first operand in the register specified by the second operand, and stores the flag assessment result in the PSW register.

[Description]

- The as850 generates one hsh machine instruction.

[Flag]

CY	1 if the lower half-word data of the result is 0, 0 if not
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the lower half-word data of the result is 0, 0 if not
SAT	---

hsw

[V850E]

[Overview]

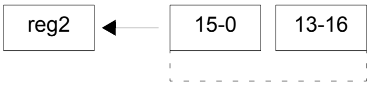
Half-word Swap Word

[Syntax]

```
(1) hsw        reg1, reg2
```

[Function]

Halfword-swaps the register value specified by the first operand and stores the result in the register specified by the second operand.



Halfword swap of reg1
(numbers indicate bit numbers)

[Description]

- The as850 generates one hsw machine instruction.

[Flag]

CY	1 if one or more halfwords in the word of the register is 0, 0 if not
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the word data of the result is 1, 0 if not
SAT	---

not

[Overview]

Not

[Syntax]

```
(1) not         reg1, reg2
(2) not         imm, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  NOTs (1's complement) the value of the register specified by the first operand, and stores the result in the register specified by the second operand.

- Syntax (2)

  NOTs (1's complement) the value of the absolute expression or relative expression specified by the first operand, and stores the result in the register specified by the second operand.

[Description]

- When this instruction is executed in syntax (1), the as850 generates one not machine instruction.
- When this instruction is executed in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions[Note]

(a) 0

`not     0, reg`	`not     r0, reg`

(b) Absolute expression having a value of other than 0 whithin the range of -16 to +15

`not     imm5, reg`	`mov     imm5, r1`
	`not     r1, reg`

(c) Absolute expression having a value of other than 0 whithin the range of -16 to +15 **[V850E]**

`not     imm5, reg`	`mov     imm5, r1`
	`not     r1, reg`

(d) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`not imm16, reg`	`movea imm16, r0, r1`
	`not r1, reg`

(e) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

`not imm, reg`	`movhi hi(imm), r0, r1`
	`not r1, reg`

Else

`not imm, reg`	`movhi hi1(imm), r0, r1`
	`movea lo(imm), r1, r1`
	`not r1, reg`

(f) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

`not imm, reg`	`movhi hi(imm), r0, r1`
	`not r1, reg`

Else

`not imm, reg`	`mov imm, r1`
	`not r1, reg`

(g) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`not $label, reg`	`movea $label, r0, r1`
	`not r1, reg`

(h) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

not #label, reg	movhi hi1(#label), r0, r1 movea lo(#label), r1, r1 not r1, reg

not label, reg	movhi hi1(label), r0, r1 movea lo(label), r1, r1 not r1, reg

not $label, reg	movhi hi1($label), r0, r1 movea lo($label), r1, r1 not r1, reg

(i) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

not #label, reg	mov #label, r1 not r1, reg

not label, reg	mov label, r1 not r1, reg

not $label, reg	mov $label, r1 not r1, reg

Note The not machine instruction does not take an immediate value as an operand.

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

or

[Overview]

Or

[Syntax]

```
(1) or          reg1, reg2
(2) or          imm, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  ORs the value of the register specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

- Syntax (2)

  ORs the value of the absolute expression or relative expression specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

[Description]

- When this instruction is executed in syntax (1), the as850 generates one or machine instruction.
- When this instruction is executed in syntax (2), the as850 executes instruction expansion to generate one or more machine instructions[Note]

(a) 0

`or        0, reg`	`or        r0, reg`

(b) Absolute expression having a value in the range of 1 to 65,535

`or        imm16, reg`	`ori       imm16, reg, reg`

(c) Absolute expression having a value in the range of -16 to -1

`or        imm5, reg`	`mov       imm5, r1`
	`or        r1, reg`

(d) Absolute expression having a value in the range of -32,768 to -17

or imm16, reg	movea imm16, r0, r1 or r1, reg

(e) Absolute expression exceeding the above ranges

If all the lower 16 bits of the value of imm are 0

or imm, reg	movhi hi(imm), r0, r1 or r1, reg

Else

or imm, reg	movhi hi1(imm), r0, r1 movea lo(imm), r1, r1 or r1, reg

(f) Absolute expression exceeding the above ranges **[V850E]**

If all the lower 16 bits of the value of imm are 0

or imm, reg	movhi hi(imm), r0, r1 or r1, reg

Else

or imm, reg	mov imm, r0, r1 or r1, reg

(g) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

or $label, reg	movea $label, r0, r1 or r1, reg

(h) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

or #label, reg	movhi hi1(#label), r0, r1 movea lo(#label), r1, r1 or r1, reg

or label, reg	movhi hi1(label), r0, r1 movea lo(label), r1, r1 or r1, reg

or $label, reg	movhi hi1($label), r0, r1 movea lo($label), r1, r1 or r1, reg

(i) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

`or      #label, reg`	`mov     #label, r1`
	`or      r1, reg`

`or      label, reg`	`mov     label, r1`
	`or      r1, reg`

`or      $label, reg`	`mov     $label, r1`
	`or      r1, reg`

Note The or machine instruction does not take an immediate value as an operand.

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

ori

[Overview]

Or Immediate

[Syntax]

```
(1) ori         imm, reg1, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with hi( ), lo( ), or hi1( ) applied

[Function]

ORs the value of the absolute expression, relative expression, or expression with hi( ), lo( ), or hi1( ) applied specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the third operand.

[Description]

- If the following is specified for imm, the as850 generates one ori machine instruction[Note].

(a) Absolute expression having a value in the range of 0 to 65,535

`ori     imm16, reg1, reg2`	`ori     imm16, reg1, reg2`

(b) Relative expression having !label or %label

`ori     !label, reg1, reg2`	`ori     !label, reg1, reg2`

`ori     %label, reg1, reg2`	`ori     %label, reg1, reg2`

(c) Expression with hi( ), lo( ), or hi1( )

`ori     imm16, reg1, reg2`	`ori     imm16, reg1, reg2`

Note The ori machine instruction takes an immediate value of 0 to 65,535 (0 to 0xffff) as the first operand.

- If the following is specified for imm, the as850 executes instruction expansion to generate one or more machine instructions.

(a) Absolute expression having a value in the range of -16 to -1

ori imm5, reg1, reg2	mov imm5, reg2 or reg1, reg2

(b) Absolute expression having a value in the range of -32,768 to -17

If reg2 is r0

ori imm16, reg1, r0	movea imm16, r0, r1 or reg1, r1

Else

ori imm16, reg1, reg2	movea imm16, r0, reg2 or reg1, reg2

(c) Absolute expression exceeding the above ranges

If all the lower 16 bits of the value of imm are 0

ori imm, reg1, reg2	movhi hi(imm), r0, reg2 or reg1, reg2

If all the lower 16 bits of the value of imm are 0 and when reg2 is r0

ori imm, reg1, r0	movhi hi(imm), r0, r1 or reg1, r1

Else

ori imm, reg1, reg2	movhi hi1(imm), r0, r1 movea lo(imm), r1, reg2 or reg1, reg2

Other than above and when reg2 is r0

ori imm, reg1, r0	movhi hi1(imm), r0, r1 movea lo(imm), r1, r1 or reg1, r1

(d) Absolute expression exceeding the above ranges **[V850E]**

If all the lower 16 bits of the value of imm are 0

`ori     imm, reg1, reg2`	`movhi   hi(imm), r0, reg2`
	`or      reg1, reg2`

If all the lower 16 bits of the value of imm are 0 and when reg2 is r0

`ori     imm, reg1, r0`	`movhi   hi(imm), r0, r1`
	`or      reg1, r1`

Else

`ori     imm, reg1, reg2`	`mov     imm, reg2`
	`or      reg1, reg2`

Other than above and when reg2 is r0

`ori     imm, reg1, r0`	`mov     imm, r1`
	`or      reg1, r1`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

If reg2 is r0

`ori     $label, reg1, r0`	`movea   $label, r0, r1`
	`or      reg1, r1`

Else

`ori     $label, reg1, reg2`	`movea   $label, r0, reg2`
	`or      reg1, reg2`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

If reg2 is r0

`ori     #label, reg1, r0`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, r1`
	`or      reg1, r1`

`ori     label, reg1, r0`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, r1`
	`or      reg1, r1`

`ori     $label, reg1, r0`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, r1`
	`or      reg1, r1`

Else

ori #label, reg1, reg2	movhi hi1(#label), r0, r1 movea lo(#label), r1, reg2 or reg1, reg2

ori label, reg1, reg2	movhi hi1(label), r0, r1 movea lo(label), r1, reg2 or reg1, reg2

ori $label, reg1, reg2	movhi hi1(label), r0, r1 movea lo(label), r1, reg2 or reg1, reg2

(g) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

If reg2 is r0

ori #label, reg1, r0	mov #label, r1 or reg1, r1

ori label, reg1, r0	mov label, r1 or reg1, r1

ori $label, reg1, r0	mov $label, r1 or reg1, r1

Else

ori #label, reg1, reg2	mov #label, reg2 or reg1, reg2

ori label, reg1, reg2	mov label, reg2 or reg1, reg2

ori $label, reg1, reg2	mov $label, reg2 or reg1, reg2

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

sar

[Overview]

Shift Arithmetic Right

[Syntax]

```
(1) sar        reg1, reg2
(2) sar        imm5, reg2
(3) sar        reg1, reg2, reg3 [V850E2]
```

The following can be specified for imm5:

- Absolute expression having a value of up to 5 bits

[Function]

- Syntax (1)

Arithmetically shifts to the right the value of the register specified by the second operand by the number of bits indicated by the lower 5 bits of the register value specified by the first operand, then stores the result in the register specified by the second operand.

- Syntax (2)

Arithmetically shifts to the right the value of the register specified by the second operand by the number of bits specified by the value of the absolute expression specified by the first operand, then stores the result in the register specified by the second operand.

- Syntax (3)

Arithmetically shifts to the right the value of the register specified by the second operand by the number of bits indicated by the lower 5 bits of the register value specified by the first operand, then stores the result in the register specified by the third operand.

[Description]

- The as850 generates one sar machine instruction.

[Flag]

CY	1 if the value of the bit shifted out last is 1, 0 if not (0 if the specified number of bits is 0)
OV	0
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of 0 to 31 is specified for imm5 in syntax (2), the as850 outputs the following message, and continues assembling using the lower 5 bits[Note] of the specified value

```
W3011: illegal operand (range error in immediate).
```

Note The sar machine instruction takes an immediate value of 0 to 31 (0x0 to 0x1f) as the first operand.

shl

[Overview]

Shift Logical Left

[Syntax]

```
(1) shl         reg1, reg2
(2) shl         imm5, reg2
(3) shl         reg1, reg2, reg3  [V850E2]
```

The following can be specified for imm5:

- Absolute expression having a value of up to 5 bits

[Function]

- Syntax (1)

  Logically shifts to the left the value of the register specified by the second operand by the number of bits indicated by the lower 5 bits of the register value specified by the first operand, then stores the result in the register specified by the second operand.

- Syntax (2)

  Logically shifts to the left the value of the register specified by the second operand by the number of bits specified by the value of the absolute expression specified by the first operand, then stores the result in the register specified by the second operand.

- Syntax (3)

  Logically shifts to the left the value of the register specified by the second operand by the number of bits indicated by the lower 5 bits of the register value specified by the first operand, then stores the result in the register specified by the third operand.

[Description]

- The as850 generates one shl machine instruction.

[Flag]

CY	1 if the value of the bit shifted out last is 1, 0 if not (0 if the specified number of bits is 0)
OV	0
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of 0 to 31 is specified for imm5 in syntax (2), the as850 outputs the following message, and continues assembling by using the lower 5 bits[Note] of the specified value.

```
W3011: illegal operand (range error in immediate).
```

Note The shl machine instruction takes an immediate value of 0 to 31 (0x0 to 0x1f) as the first operand.

shr

[Overview]

Shift Logical Right

[Syntax]

```
(1) shr         reg1, reg2
(2) shr         imm5, reg2
(3) shr         reg1, reg2, reg3
```
[V850E2]

The following can be specified for imm5:

- Absolute expression having a value of up to 5 bits

[Function]

- Syntax (1)

Logically shifts to the right the value of the register specified by the second operand by the number of bits indicated by the lower 5 bits of the register value specified by the first operand, then stores the result in the register specified by the second operand.

- Syntax (2)

Logically shifts to the right the value of the register specified by the second operand by the number of bits specified by the value of the absolute expression specified by the first operand, then stores the result in the register specified by the second operand.

- Syntax (3)

Logically shifts to the right the value of the register specified by the second operand by the number of bits indicated by the lower 5 bits of the register value specified by the first operand, then stores the result in the register specified by the third operand.

[Description]

- The as850 generates one shr machine instruction.

[Flag]

CY	1 if the value of the bit shifted out last is 1, 0 if not (0 if the specified number of bits is 0)
OV	0
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of 0 to 31 is specified as imm5 in syntax (2), the as850 outputs the following message, and continues assembling by using the lower 5 bits[Note] of the specified value

```
W3011: illegal operand (range error in immediate).
```

Note The shr machine instruction takes an immediate value of 0 to 31 (0x0 to 0x1f) as the first operand.

sxb

[V850E]

[Overview]

Sign Extend Byte

[Syntax]

```
(1) sxb        reg
```

[Function]

Sign-extends the data of the lowermost byte of the register specified by the first operand to word length.

[Description]

- The as850 generates one sxb machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

sxh

[V850E]

[Overview]

Sign Extend Half-word

[Syntax]

```
(1) sxh        reg
```

[Function]

Sign-extends the data of the lower 2 bytes of the register specified by the first operand to word length.

[Description]

- The as850 generates one sxh machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

tst

[Overview]

Test

[Syntax]

```
(1) tst         reg1, reg2
(2) tst         imm, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  ANDs the value of the register specified by the second operand with the value of the register specified by the first operand, and sets only the flags without storing the result.

- Syntax (2)

  ANDs the value of the register specified by the second operand with the value of the absolute expression or relative expression specified by the first operand, and sets only the flags without storing the result.

[Description]

- When this instruction is executed in syntax (1), the as850 generates one tst machine instruction.
- When this instruction is executed in syntax (2), the as850 executes instruction expansion to generate two or more machine instructions[Note].

(a) 0

`tst 0, reg`	`tst r0, reg`

(b) Absolute expression having a value of other than 0 whithin the range of -16 to +15

`tst imm5, reg`	`mov imm5, r1`
	`tst r1, reg`

(c) Absolute expression having a value of other than 0 whithin the range of -16 to +15 **[V850E]**

`tst imm5, reg`	`mov imm5, r1`
	`tst r1, reg`

(d) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

tst imm16, reg	movea imm16, r0, r1 tst r1, reg

(e) Absolute expression having a value exceeding the range of -32,768 to +32,767

If all the lower 16 bits of the value of imm are 0

tst imm, reg	movhi hi(imm), r0, r1 tst r1, reg

Else

tst imm, reg	movhi hi1(imm), r0, r1 movea lo(imm), r1, r1 tst r1, reg

(f) Absolute expression having a value exceeding the range of -32,768 to +32,767 **[V850E]**

If all the lower 16 bits of the value of imm are 0

tst imm, reg	movhi hi(imm), r0, r1 tst r1, reg

Else

tst imm, reg	mov imm, r1 tst r1, reg

(g) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

tst $label, reg	movea $label, r0, r1 tst r1, reg

(h) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`tst     #label, reg`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, r1`
	`tst     r1, reg`

`tst     label, reg`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, r1`
	`tst     r1, reg`

`tst     $label, reg`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, r1`
	`tst     r1, reg`

(i) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

`tst     #label, reg`	`mov     #label, r1`
	`tst     r1, reg`

`tst     label, reg`	`mov     label, r1`
	`tst     r1, reg`

`tst     $label, reg`	`mov     $label, r1`
	`tst     r1, reg`

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

xor

[Overview]

Exclusive Or

[Syntax]

```
(1) xor         reg1, reg2
(2) xor         imm, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax (1)

  Exclusive-ORs the value of the register specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

- Syntax (2)

  Exclusive-ORs the value of the absolute expression or relative expression specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

[Description]

- When this instruction is executed in syntax (1), the as850 generates one xor machine instruction.
- When this instruction is executed in syntax (2), the as850 executes instruction expansion to generate two or more machine instructionsNote

(a) 0

`xor     0, reg`	`xor     r0, reg`

(b) Absolute expression having a value in the range of 1 to 65,535

`xor     imm16, reg`	`xori    imm16, reg, reg`

(c) Absolute expression having a value in the range of -16 to -1

`xor     imm5, reg`	`mov     imm5, r1`
	`xor     r1, reg`

(d) Absolute expression having a value in the range of -32,768 to -17

xor imm16, reg	movea imm16, r0, r1 xor r1, reg

(e) Absolute expression exceeding the above ranges

If all the lower 16 bits of the value of imm are 0

xor imm, reg	movhi hi(imm), r0, r1 xor r1, reg

Else

xor imm, reg	movhi hi1(imm), r0, r1 movea lo(imm), r1, r1 xor r1, reg

(f) Absolute expression exceeding the above ranges **[V850E]**

If all the lower 16 bits of the value of imm are 0

xor imm, reg	movhi hi(imm), r0, r1 xor r1, reg

Else

xor imm, reg	mov imm, r0, r1 xor r1, reg

(g) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

xor $label, reg	movea $label, r0, r1 xor r1, reg

(h) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`xor     #label, reg`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, r1`
	`xor     r1, reg`

`xor     label, reg`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, r1`
	`xor     r1, reg`

`xor     $label, reg`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, r1`
	`xor     r1, reg`

(i) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

`xor     #label, reg`	`mov     #label, r1`
	`xor     r1, reg`

`xor     label, reg`	`mov     label, r1`
	`xor     r1, reg`

`xor     $label, reg`	`mov     $label, r1`
	`xor     r1, reg`

Note The xor machine instruction does not take an immediate value as an operand.

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

xori

[Overview]

Exclusive Or Immediate

[Syntax]

```
(1) xori        imm, reg1, reg2
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with hi( ), lo( ), or hi1( ) applied

[Function]

Exclusive-ORs the value of the absolute expression, relative expression, or expression with hi( ), lo( ), or hi1( ) applied specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the third operand.

[Description]

- If the following is specified for imm, the as850 generates one xori machine instruction[Note].

(a) Absolute expression having a value in the range of 0 to 65,535

xori imm16, reg1, reg2	xori imm16, reg1, reg2

(b) Relative expression having !label or %label

xori !label, reg1, reg2	xori !label, reg1, reg2

xori %label, reg1, reg2	xori %label, reg1, reg2

(c) Expression with hi( ), lo( ), or hi1( )

xori imm16, reg1, reg2	xori imm16, reg1, reg2

Note The xori machine instruction takes an immediate value of 0 to 65,535 (0 to 0xffff) as the first operand.

- If the following is specified for imm, the as850 executes instruction expansion to generate one or more machine instructions

(a) Absolute expression having a value in the range of -16 to -1

`xori    imm5, reg1, reg2`	`mov     imm5, reg2`
	`xor     reg1, reg2`

(b) Absolute expression having a value in the range of -32,768 to -17

If reg2 is r0

`xori    imm16, reg1, r0`	`movea   imm16, r0, r1`
	`xor     reg1, r1`

Else

`xori    imm16, reg1, reg2`	`movea   imm16, r0, reg2`
	`xor     reg1, reg2`

(c) Absolute expression exceeding the above ranges

If all the lower 16 bits of the value of imm are 0

`xori    imm, reg1, reg2`	`movhi   hi(imm), r0, reg2`
	`xor     reg1, reg2`

If all the lower 16 bits of the value of imm are 0 and when reg2 is r0

`xori    imm, reg1, r0`	`movhi   hi(imm), r0, r1`
	`xor     reg1, r1`

Else

`xori    imm, reg1, reg2`	`movhi   hi1(imm), r0, r1`
	`movea   lo(imm), r1, reg2`
	`xor     reg1, reg2`

Other than above and when reg2 is r0

`xori    imm, reg1, r0`	`movhi   hi1(imm), r0, r1`
	`movea   lo(imm), r1, r1`
	`xor     reg1, r1`

(d) Absolute expression exceeding the above ranges **[V850E]**

If all the lower 16 bits of the value of imm are 0

`xori    imm, reg1, reg2`	`movhi   hi(imm), r0, reg2`
	`xor     reg1, reg2`

If all the lower 16 bits of the value of imm are 0 and when reg2 is r0

`xori    imm, reg1, r0`	`movhi   hi(imm), r0, r1`
	`xor     reg1, r1`

Else

`xori    imm, reg1, reg2`	`mov     imm, reg2`
	`xor     reg1, reg2`

Other than above and when reg2 is r0

`xori    imm, reg1, r0`	`mov     imm, r1`
	`xor     reg1, r1`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

If reg2 is r0

`xori    $label, reg1, reg2`	`movea   $label, r0, reg2`
	`xor     reg1, reg2`

Else

`xori    $label, reg1, reg2`	`movea   $label, r0, reg2`
	`xor     reg1, reg2`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

If reg2 is r0

`xori    #label, reg1, r0`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, r1`
	`xor     reg1, r1`

`xori    label, reg1, r0`	`movhi   hi1(label), r0, r1`
	`movea   lo(label), r1, r1`
	`xor     reg1, r1`

`xori    $label, reg1, r0`	`movhi   hi1($label), r0, r1`
	`movea   lo($label), r1, r1`
	`xor     reg1, r1`

Else

`xori #label, reg1, reg2`	`movhi hi1(#label), r0, r1`
	`movea lo(#label), r1, reg2`
	`xor reg1, reg2`

`xori label, reg1, reg2`	`movhi hi1(label), r0, r1`
	`movea lo(label), r1, reg2`
	`xor reg1, reg2`

`xori $label, reg1, reg2`	`movhi hi1($label), r0, r1`
	`movea lo($label), r1, reg2`
	`xor reg1, reg2`

(g) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section **[V850E]**

If reg2 is r0

`xori #label, reg1, r0`	`mov #label, r1`
	`xor reg1, r1`

`xori label, reg1, r0`	`mov label, r1`
	`xor reg1, r1`

`xori $label, reg1, r0`	`mov $label, r1`
	`xor reg1, r1`

Else

`xori #label, reg1, reg2`	`mov #label, reg2`
	`xor reg1, reg2`

`xori label, reg1, reg2`	`mov label, reg2`
	`xor reg1, reg2`

`xori $label, reg1, reg2`	`mov $label, reg2`
	`xor reg1, reg2`

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

zxb

[V850E]

[Overview]

Zero Extend Byte

[Syntax]

```
(1) zxb         reg
```

[Function]

Zero-extends the data of the lowermost byte of the register specified by the first operand to word length.

[Description]

- The as850 generates one zxb machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

zxh

[V850E]

[Overview]

Zero Extend Half-word

[Syntax]

```
(1) zxh         reg
```

[Function]

Zero-extends the data of the lower halfword of the register specified by the first operand to word length.

[Description]

- The as850 generates one zxh machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

sch0l

[V850E2]

[Overview]

Bit (0) Search from MSB Side (Search zero from left)

[Syntax]

```
(1) sch0l       reg2, reg3
```

[Function]

Searches the word data of the register specified by the first operand, from the left (MSB side), and stores the position of the first bit (0) found in the register specified by the second operand in hexadecimal. (For example, if bit 31 of the register specified by the first operand is 0, 01H is stored in the register specified by the second operand.)

If no bit (0) is found, 0 is written into the register specified by the second operand, and the Z flag is simultaneously set (1). If a bit (0) is found at the end, the CY flag is set (1).

[Description]

- The as850 generates one sch0l machine instruction.

[Flag]

CY	1 if a bit (0) is found at the end, 0 if not
OV	0
S	0
Z	1 if a bit (0) is not found, 0 if not
SAT	---

sch0r

[V850E2]

[Overview]

Bit (0) Search from LSB Side (Search zero from right)

[Syntax]

```
(1) sch0r       reg2, reg3
```

[Function]

Searches the word data of the register specified by the first operand, from the right (LSB side), and stores the position of the first bit (0) found in the register specified by the second operand in hexadecimal. (For example, if bit 0 of the register specified by the first operand is 0, 01H is stored in the register specified by the second operand.)

If no bit (0) is found, 0 is written into the register specified by the second operand, and the Z flag is simultaneously set (1). If a bit (0) is found at the end, the CY flag is set (1).

[Description]

- The as850 generates one sch0r machine instruction.

[Flag]

CY	1 if a bit (0) is found at the end, 0 if not
OV	0
S	0
Z	1 if a bit (0) is not found, 0 if not
SAT	---

sch1l

[V850E2]

[Overview]

Bit (1) Search from MSB Side (Search one from left)

[Syntax]

```
(1) sch1l      reg2, reg3
```

[Function]

Searches the word data of the register specified by the first operand, from the left (MSB side), and stores the position of the first bit (1) found in the register specified by the second operand in hexadecimal. (For example, if bit 31 of the register specified by the first operand is 1, 01H is stored in the register specified by the second operand.)

If no bit (1) is found, 0 is written into the register specified by the second operand, and the Z flag is simultaneously set (1). If a bit (0) is found at the end, the CY flag is set (1).

[Description]

- The as850 generates one sch1l machine instruction.

[Flag]

CY	1 if a bit (1) is found at the end, 0 if not
OV	0
S	0
Z	1 if a bit (1) is not found, 0 if not
SAT	---

sch1r

[V850E2]

[Overview]

Bit (1) Search from LSB Side (Search zero from right)

[Syntax]

```
(1) sch1r      reg2, reg3
```

[Function]

Searches the word data of the register specified by the first operand, from the right (LSB side), and stores the position of the first bit (1) found in the register specified by the second operand in hexadecimal. (For example, if bit 0 of the register specified by the first operand is 1, 01H is stored in the register specified by the second operand.)

If no bit (1) is found, 0 is written into the register specified by the second operand, and the Z flag is simultaneously set (1). If a bit (1) is found at the end, the CY flag is set (1).

[Description]

- The as850 generates one sch1r machine instruction.

[Flag]

CY	1 if a bit (1) is found at the end, 0 if not
OV	0
S	0
Z	1 if a bit (1) is not found, 0 if not
SAT	---

3.6 Branch Instructions

This section describes the branch instructions.

Next table lists the branch instructions described in this section.

Table 3 - 10 Branch Instructions

Instruction	Meaning
jarl	Jump and register link
jarl22	Jump and register link **[V850E2]**
jarl32	Jump and register link **[V850E2]**
jcond	Conditional branch
jmp	Unconditional branch
jmp32	Unconditional branch (jump) **[V850E2]**
jr	Unconditional branch (PC relative)
jr22	Unconditional branch (PC relative) **[V850E2]**
jr32	Unconditional branch (PC relative) **[V850E2]**

jarl

[Overview]

Jump and Register Link

[Syntax]

```
(1) jarl        disp22, reg2
(2) jarl        disp32, reg1
```
[V850E2]

The following can be specified as the displacement (disp22):

- Absolute expression having a value of up to 22 bits
- Relative expression having a PC offset reference of label

[Function]

- Syntax (1)

  Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand.

  The return address is stored in the register specified by the second operand.

- Syntax (2)

  Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand.

  The return address is stored in the register specified by the second operand.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one jarl machine instruction[Note] if any of the following expressions are specified for disp22.

  (a) Absolute value in the range of -2,097,152 to +2,097,151

  (b) Relative expression that has a PC offset reference of label having a definition in the same section and the same file as this instruction, and which has a value in the range of -2,097,152 to +2,097,151

  (c) Relative expression having a PC offset reference of a label having no definition in the same file or section as this instruction

Note The jarl machine instruction takes an immediate value in the range of -2,097,152 to +2,097,151 (0xfe00000 to 0x1fffff) as the displacement.

- If the instruction is executed in syntax (2), the as850 generates one jarl machine instruction (6-byte long instruction).

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression that exceeds the range of -2,097,152 to +2,097,151, or a relative expression having a PC offset reference of label with a definition in the same section and the same file as this instruction and having a value that falls outside the range of -2,097,152 to +2,097,151 is specified as disp22, the as850 outputs the following message and stops assembling.

```
E3230: illegal operand (range error in displacement)
```

- If an absolute expression having an odd-numbered value, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction and having an odd-numbered value, is specified as disp22, the as850 outputs the following message and stops assembling.

```
E3226: illegal operand (must be even displacement)
```

- When the assembler option -Xfar_jump is not specified, and an absolute expression outside of the range -2,097,152 to +2,097,151 or a relative expression outside of the range -2,097,152 to +2,097,151, having a label PC offset reference with a definition in the same file and same section as this instruction, is specified as disp32, the following message is output and assembly is stopped.

```
E3230: illegal operand (range error in displacement)
```

jarl22

[V850E2]

[Overview]

Jump and Register Link

[Syntax]

```
(1) jarl22     disp22, reg1
```

The following can be specified as the displacement (disp22):

- Absolute expression having a value of up to 22 bits
- Relative expression having a PC offset reference of label

[Function]

Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand.

The return address is stored in the register specified by the second operand.

[Description]

- If the following is specified for disp22, the as850 generates one jarl machine instruction[Note].

(a) Absolute value in the range of -2,097,152 to +2,097,151

(b) Relative expression that has a PC offset reference of label having a definition in the same section and the same file as this instruction, and which has a value in the range of -2,097,152 to +2,097,151

(c) Relative expression having a PC offset reference of a label having no definition in the same file or section as this instruction

Note The jarl machine instruction takes an immediate value in the range of -2,097,152 to +2,097,151 (0xfe00000 to 0x1fffff) as the displacement.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression that exceeds the range of -2,097,152 to +2,097,151, or a relative expression having a PC offset reference of label with a definition in the same section and the same file as this instruction and having a value that falls outside the range of -2,097,152 to +2,097,151 is specified as disp22, the as850 outputs the following message and stops assembling.

```
E3230: illegal operand (range error in displacement)
```

- If an absolute expression having an odd-numbered value, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction and having an odd-numbered value, is specified as disp22, the as850 outputs the following message and stops assembling.

```
E3226: illegal operand (must be even displacement)
```

jarl32

[V850E2]

[Overview]

Jump and Register Link

[Syntax]

```
(1) jarl32     disp32, reg1
```

[Function]

Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand.

The return address is stored in the register specified by the second operand.

[Description]

- The as850 generates one jarl machine instruction (6-byte long instruction).

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

jcond

[Overview]

Jump on Condition

[Syntax]

```
(1) jcond       disp22
```

The following can be specified as the displacement (disp22):

- Absolute expression having a value of up to 22 bits
- Relative expression having a PC offset reference of label

[Function]

Compares the flag condition indicated by string *cond* (refer to Table 3 - 11) with the current flag condition. If they are found to be the same, transfers control to the address obtained by adding the value of the absolute expression or relative expression specified by the operand to the current value of the program counter (PC)[Note].

Note Mnemonic b*cond* can be used for the j*cond* instruction other than jbr. Mnemonic br can be used for the jbr instruction (there is no functional difference).

Table 3 - 11 j*cond* Instruction List

Instruction	Flag Condition	Meaning of Flag Condition
jgt	((S xor OV) or Z) = 0	Greater than (signed)
jge	(S xor OV) = 0	Greater than or equal (signed)
jlt	(S xor OV) = 1	Less than (signed)
jle	((S xor OV) or Z) = 1	Less than or equal (signed)
jh	(CY or Z) = 0	Higher (Greater than)
jnl	CY = 0	Not lower (Greater than or equal)
jl	CY = 1	Lower (Less than)
jnh	(CY or Z) = 1	Not higher (Less than or equal)
je	Z = 1	Equal
jne	Z = 0	Not equal
jv	OV = 1	Overflow
jnv	OV = 0	No overflow
jn	S = 1	Negative
jp	S = 0	Positive
jc	CY = 1	Carry
jnc	CY = 0	No carry
jz	Z = 1	Zero
jnz	Z = 0	Not zero
jbr	-	Always (Unconditional)
jsa	SAT = 1	Saturated

[Description]

- If the following is specified for disp22, the as850 generates one bcond machine instruction[Note].

(a) Absolute expression having a value in the range of -256 to +255

(b) Relative expression having a PC offset reference for a label with a definition in the same section and the same file as this instruction and having a value in the range of -256 to +255

`j`*`cond`*`   disp9`	`b`*`cond`*`   disp9`

Note The b*cond* machine instruction takes an immediate value in the range of -256 to +255 (0xffffff00 to 0xff) as the displacement.

- If the following is specified as disp22, the as850 executes instruction expansion and generates two or more machine instructions.

(a) Absolute expression having a value exceeding the range of -256 to +255 but within the range of -2,097,150 to +2,097,153[Note]

(b) Relative expression having a PC offset reference of label with a definition in the same section of the same file as this instruction and having a value exceeding the range of -256 to +255 but within the range of -2,097,150 to +2,097,153

(c) Relative expression having a PC offset reference of label without a definition in the same file or section as this instruction

Note The range of -2,097,150 to +2,097,153 applies to instructions other than jbr and jsa. The range for the jbr instruction is from -2,097,152 to +2,097,151, and that for the jsa instruction is from -2,097,148 to +2,097,155.

`jbr     disp22`	`    jr  disp22`

`jsa     disp22`	`    bsa Label1`
	`    br  Label2`
	`Label1:`
	`    jr  disp22 - 4`
	`Label2:`

`j`*`cond`*`   disp22`	`    bn`*`cond`*` Label`[Note]
	`    jr  disp22 - 2`
	`Label:`

Note bn*cond* denotes an instruction that effects control branches under opposite conditions, for example, bnz for bz or ble for bgt.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of -2,097,150 to +2,097,153, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction, and having a value exceeding the range of -2,097,150 to +2,097,153, is specified as disp22, the as850 outputs the following message and stops assembling.

```
E3230: illegal operand (range error in displacement)
```

- If an absolute expression having an odd-numbered value, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction, and having an odd-numbered value, is specified as disp22, the as850 outputs the following message and stops assembling.

```
E3226: illegal operand (must be even displacement)
```

jmp

[Overview]

Jump

[Syntax]

```
(1) jmp          [reg]
(2) jmp          disp32[reg]  [V850E2]
(3) jmp          addr
```

The following can be specified for addr:

- Relative expression having the absolute address reference of a label

[Function]

- Syntax (1)

  Transfers control to the address indicated by the value of the register specified by the operand.

- Syntax (2)

  Transfers control to the address attained by adding the displacement specified by the operand and the register content.

- Syntax (3)

  Transfers control to the address indicated by the value of the relative expression specified by the operand.

[Description]

- When this instruction is executed in syntax (1), the as850 generates one jmp machine instruction.
- When this instruction is executed in syntax (2), the as850 generates one jmp (6-byte long instruction) machine instructions
- When this instruction is executed in syntax (3), the as850 executes instruction expansion and generates two or more machine instructions

`jmp     #label`	`movhi   hi1(#label), r0, r1`
	`movea   lo(#label), r1, r1`
	`jmp     [r1]`

[V850E]

`jmp     #label`	`mov     #label, r1`
	`jmp     [r1]`

[Flag]

CY	---

OV	---
S	---
Z	---
SAT	---

[Caution]

- If an expression other than a relative expression having the absolute address reference of a label is specified as addr in syntax (3), the as850 outputs the following message and stops assembling.

```
E3224: illegal operand (label reference for jmp must be #label)
```

jmp32

[V850E2]

[Overview]

Unconditional Branch (Jump)

[Syntax]

```
(1) jmp32       disp32[reg]
```

[Function]

Transfers control to the address attained by adding the displacement specified by the operand and the register content.

[Description]

- The as850 generates one jmp machine instruction (6-byte long instruction).

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

jr

[Overview]

Jump Relative

[Syntax]

```
(1) jr          disp22
(2) jr          disp32
```
[V850E2]

The following can be specified as the displacement (disp22):

- Absolute expression having a value of up to 22 bits
- Relative expression having a PC offset reference of label

[Function]

- Syntax (1)

  Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand.

- Syntax (2)

  Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand.

[Description]

- If the instruction is executed in syntax (1), the as850 generates one jr machine instruction[Note] if any of the following expressions are specified for disp22.

(a) Absolute expression having a value in the range of -2,097,152 to +2,097,151

(b) Relative expression that has a PC offset reference of label having a definition in the same section of the same file as this instruction, and having a value in the range of -2,097,152 to +2,097,151

(c) Relative expression having a PC offset reference of a label with no definition in the same file or section as this instruction

Note The jr machine instruction takes an immediate value in the range of -2,097,152 to +2,097,151 (0xfe00000 to 0x1fffff) as the displacement.

- If the instruction is executed in syntax (2), the as850 generates one jr machine instruction (6-byte long instruction).

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of -2,097,152 to +2,097,151, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction, and having a value exceeding the range of -2,097,152 to +2,097,151, is specified as disp22, the as850 outputs the following message and stops assembling.

```
E3230: illegal operand (range error in displacement)
```

- If an absolute expression having an odd-numbered value or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction, and having an odd-numbered value, is specified as disp22, the as850 outputs the following message and stops assembling.

```
E3226: illegal operand (must be even displacement)
```

- When the assembler option -Xfar_jump is not specified, and an absolute expression outside of the range -2,097,152 to +2,097,151 or a relative expression outside of the range -2,097,152 to +2,097,151, having a label PC offset reference with a definition in the same file and same section as this instruction, is specified as disp32, the following message is output and assembly is stopped.

```
E3230: illegal operand (range error in displacement)
```

jr22

[V850E2]

[Overview]

Unconditional Branch (PC Relative) (Jump Relative)

[Syntax]

```
(1) jr22        disp22
```

The following can be specified as the displacement (disp22):

- Absolute expression having a value of up to 22 bits
- Relative expression having a PC offset reference of label

[Function]

Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the operand.

[Description]

- If the following is specified for disp22, the as850 generates one jr machine instruction[Note].

(a) Absolute value in the range of -2,097,152 to +2,097,151

(b) Relative expression that has a PC offset reference of label having a definition in the same section and the same file as this instruction, and which has a value in the range of -2,097,152 to +2,097,151

(c) Relative expression having a PC offset reference of a label having no definition in the same file or section as this instruction

Note The jr machine instruction takes an immediate value in the range of -2,097,152 to +2,097,151 (0xfe00000 to 0x1fffff) as the displacement.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression that exceeds the range of -2,097,152 to +2,097,151, or a relative expression having a PC offset reference of label with a definition in the same section and the same file as this instruction and having a value that falls outside the range of -2,097,152 to +2,097,151 is specified as disp22, the as850 outputs the following message and stops assembling.

```
E3230: illegal operand (range error in displacement)
```

- If an absolute expression having an odd-numbered value, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction and having an odd-numbered value, is specified as disp22, the as850 outputs the following message and stops assembling.

```
E3226: illegal operand (must be even displacement)
```

jr32

[V850E2]

[Overview]

Unconditional Branch (PC relative) (Jump Relative)

[Syntax]

```
(1) jr32       disp32
```

[Function]

Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand.

[Description]

- The as850 generates one jr machine instruction (6-byte long instruction).

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

3.7 Bit Manipulation Instructions

This section describes the bit manipulation instructions.

Next table lists the instructions described in this section.

Table 3 - 12 Bit Manipulation Instructions

Instruction	Meaning
clr1	Bit clear
not1	Bit negation
set1	Bit set
tst1	Bit test

clr1

[Overview]

Clear Bit

[Syntax]

(1) `clr1        bit# 3, disp[reg1]`

(2) `clr1        reg2, [reg1]` **[V850E]**

(3) `clr1        BITIO`

The following can be specified as a displacement (disp):

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with hi( ), lo( ), or hi1( ) applied

The disp cannot be specified in syntax (2).

[Function]

- Syntax (1)

  Clears the bit specified by the first operand of the data indicated by the address specified by the second operand. The bits other than the one specified are not affected.

- Syntax (2)

  Clears the bit specified by the lower 3 bits of the register value specified by the first operand of the data indicated by the address specified by the register value of the second operand. The bits other than the one specified are not affected.

- Syntax (3)

  Clears the bit specified by the peripheral I/O register bit name (only reserved words defined in the device file) in the data indicated by the address specified by the first operand.

[Description]

- If the following is specified as disp, the as850 generates one clr1 machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`clr1    disp16[reg1], reg2`	`clr1    disp16[reg1], reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`clr1    $label[reg1], reg2`	`clr1    $label[reg1], reg2`

(c) Relative expression having !label or %label

`clr1    !label[reg1], reg2`	`clr1    !label[reg1], reg2`

`clr1    !label[reg1], reg2`	`clr1    %label[reg1], reg2`

(d) Expression with hi( ), lo( ), or hi1( )

`clr1    disp16[reg1], reg2`	`clr1    disp16[reg1], reg2`

Note The clr1 machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xffff8000 to 0x7fff) as the displacement.

- If the following is specified as disp, the as850 executes instruction expansion and generates two or more machine instructions.

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767

`clr1    disp[reg1], reg2`	`movhi   hi1(disp), reg1, r1`
	`clr1    lo(disp)[r1], reg2`

(b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`clr1    #label[reg1], reg2`	`movhi   hi1(#label), reg1, r1`
	`clr1    lo(#label)[r1], reg2`

`clr1    label[reg1], reg2`	`movhi   hi1(label), reg1, r1`
	`clr1    lo(label)[r1], reg2`

`clr1    $label[reg1], reg2`	`movhi   hi1($label), reg1, r1`
	`clr1    lo($label)[r1], reg2`

[Flag]

CY	---
OV	---
S	---
Z	1 if the specified bit is 0, 0 if not[Note]
SAT	---

Note The flag values shown here are those existing prior to the execution of this instruction, not those after the execution.

[Caution]

- If disp is omitted, the as850 assumes 0.
- If a relative expression with #label or a relative expression with #label and with hi( ), lo( ), or hi1( ) applied is specified as disp, [reg1] that follows the expression can be omitted. If omitted, the as850 assumes [r0] to be specified.
- If a relative expression having $label or a relative expression having $label and with hi( ), lo( ), or hi1( ) applied is specified as disp, [reg1] that follows the expression can be omitted. If omitted, the as850 assumes that [gp] is specified.
- If a peripheral I/O register name that is defined in the device file is specified as disp, [reg1] that follows the name can be omitted. If omitted, the as850 assumes that [r0] is specified.

not1

[Overview]

Not Bit

[Syntax]

```
(1) not1        bit# 3, disp[reg1]
(2) not1        reg2, [reg1]
(3) not1        BITIO
```

[V850E] (applies to syntax (2))

The following can be specified as a displacement (disp):

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with hi( ), lo( ), or hi1( ) applied

The disp cannot be specified in syntax (2).

[Function]

- Syntax (1)

  Inverts the bit specified by the first operand (0 to 1 or 1 to 0) of the data indicated by the address specified by the second operand. The bits other than the one specified are not affected.

- Syntax (2)

  Inverts the bit specified by the lower 3 bits of the register value specified by the first operand (0 to 1 or 1 to 0) of the data indicated by the address specified by the register value of the second operand. The bits other than the one specified are not affected.

- Syntax (3)

  Inverts (from 0 to 1 or 1 to 0) the bit specified by the peripheral I/O register bit name (only reserved words defined in the device file) in the data indicated by the address specified by the first operand.

[Description]

- If the following is specified for disp, the as850 generates one not1 machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`not1    disp16[reg1], reg2`	`not1    disp16[reg1], reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`not1    $label[reg1], reg2`	`not1    $label[reg1], reg2`

(c) Relative expression having !label or %label

`not1    !label[reg1], reg2`	`not1    !label[reg1], reg2`

`not1    !label[reg1], reg2`	`not1    %label[reg1], reg2`

(d) Expression with hi( ), lo( ), or hi1( )

`not1    disp16[reg1], reg2`	`not1    disp16[reg1], reg2`

Note The not1 machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xffff8000 to 0x7fff) as the displacement.

- If the following is specified as disp, the as850 executes instruction expansion and generates two or more machine instructions

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767

`not1    disp[reg1], reg2`	`movhi   hi1(disp), reg1, r1`
	`not1    lo(disp)[r1], reg2`

(b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`not1    #label[reg1], reg2`	`movhi   hi1(#label), reg1, r1`
	`not1    lo(#label)[r1], reg2`

`not1    label[reg1], reg2`	`movhi   hi1(label), reg1, r1`
	`not1    lo(label)[r1], reg2`

`not1    $label[reg1], reg2`	`movhi   hi1($label), reg1, r1`
	`not1    lo($label)[r1], reg2`

[Flag]

CY	---
OV	---
S	---
Z	1 if the specified bit is 0, 0 if not[Note]
SAT	---

Note The flag values shown here are those existing prior to the execution of this instruction, not those after the execution.

[Caution]

- If disp is omitted, the as850 assumes 0.
- If a relative expression with #label or a relative expression with #label and with hi( ), lo( ), or hi1( ) applied is specified as disp, [reg1] that follows the expression can be omitted. If omitted, the as850 assumes [r0] to be specified.
- If a relative expression having $label or a relative expression having $label and with hi( ), lo( ), or hi1( ) applied is specified as disp, [reg1] that follows the expression can be omitted. If omitted, the as850 assumes that [gp] is specified.
- If a peripheral I/O register name that is defined in the device file is specified as disp, [reg1] that follows the name can be omitted. If omitted, the as850 assumes that [r0] is specified.

set1

[Overview]

Set Bit

[Syntax]

```
(1) set1        bit #3, disp[reg1]
(2) set1        reg2, [reg1]
(3) set1        BITIO
```

[V850E] (applies to syntax (2))

The following can be specified as a displacement (disp):

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with hi( ), lo( ), or hi1( ) applied

The disp cannot be specified in syntax (2).

[Function]

- Syntax (1)

  Sets the bit specified by the first operand of the data indicated by the address specified by the second operand. The bits other than the one specified are not affected.

- Syntax (2)

  Sets the bit specified by the lower 3 bits of the register value specified by the first operand of the data indicated by the address specified by the register value of the second operand. The bits other than the one specified are not affected.

- Syntax (3)

  Sets the bit specified by the peripheral I/O register bit name (only reserved words defined in the device file) in the data indicated by the address specified by the first operand.

[Description]

- If the following is specified for disp, the as850 generates one set1 machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`set1    disp16[reg1], reg2`	`set1    disp16[reg1], reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`set1    $label[reg1], reg2`	`set1    $label[reg1], reg2`

(c) Relative expression having !label or %label

`set1 !label[reg1], reg2`	`set1 !label[reg1], reg2`

`set1 !label[reg1], reg2`	`set1 %label[reg1], reg2`

(d) Expression with hi( ), lo( ), or hi1( )

`set1 disp16[reg1], reg2`	`set1 disp16[reg1], reg2`

Note The set1 machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xffff8000 to 0x7fff) as the displacement.

- If the following is specified for disp, the as850 executes instruction expansion, then generates two or more machine instructions.

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767

`set1 disp[reg1], reg2`	`movhi hi1(disp), reg1, r1`
	`set1 lo(disp)[r1], reg2`

(b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`set1 #label[reg1], reg2`	`movhi hi1(#label), reg1, r1`
	`set1 lo(#label)[r1], reg2`

`set1 label[reg1], reg2`	`movhi hi1(label), reg1, r1`
	`set1 lo(label)[r1], reg2`

`set1 $label[reg1], reg2`	`movhi hi1($label), reg1, r1`
	`set1 lo($label)[r1], reg2`

[Flag]

CY	---
OV	---
S	---
Z	1 if the specified bit is 0, 0 if not[Note]
SAT	---

Note The flag values shown here are those existing prior to the execution of this instruction, not those after the execution.

[Caution]

- If disp is omitted, the as850 assumes 0.
- If a relative expression with #label or a relative expression with #label and with hi( ), lo( ), or hi1( ) applied is specified as disp, [reg1] that follows the expression can be omitted. If omitted, the as850 assumes [r0] to be specified.
- If a relative expression having $label or a relative expression having $label and with hi( ), lo( ), or hi1( ) applied is specified as disp, [reg1] that follows the expression can be omitted. If omitted, the as850 assumes that [gp] is specified.
- If a peripheral I/O register name that is defined in the device file is specified as disp, [reg1] that follows the name can be omitted. If omitted, the as850 assumes that [r0] is specified.

tst1

[Overview]

Test Bit

[Syntax]

```
(1) tst1        bit# 3, disp[reg1]
(2) tst1        reg2, [reg1]  [V850E]
(3) tst1        BITIO
```

The following can be specified as a displacement (disp):

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with hi( ), lo( ), or hi1( ) applied

The disp cannot be specified in syntax (2).

[Function]

- Syntax (1)

  Sets only a flag according to the value of the bit specified by the first operand of the data indicated by the address specified by the second operand. The value of the second operand and the specified bit are not changed.

- Syntax (2)

  Sets only a flag according to the value of the bit of the lower 3 bits of the register value specified by the first operand of the data indicated by the address specified by the second operand. The value of the second operand and the specified bit are not changed.

- Syntax (3)

  Sets only the flag in accordance with the value of the bit specified by the peripheral I/O register bit name (only reserved words defined in the device file) in the data indicated by the address specified by the first operand. The value of the peripheral I/O register bit is not affected.

[Description]

- If the following is specified for disp, the as850 generates one tst1 machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`tst1     disp16[reg1], reg2`	`tst1     disp16[reg1], reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`tst1     $label[reg1], reg2`	`tst1     $label[reg1], reg2`

(c) Relative expression having !label or %label

`tst1    !label[reg1], reg2`	`tst1    !label[reg1], reg2`

`tst1    !label[reg1], reg2`	`tst1    %label[reg1], reg2`

(d) Expression with hi( ), lo( ), or hi1( )

`tst1    disp16[reg1], reg2`	`tst1    disp16[reg1], reg2`

Note The tst1 machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xffff8000 to 0x7fff) as the displacement.

- If the following is specified for disp, the as850 executes instruction expansion, then generates two or more machine instructions.

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767

`tst1    disp[reg1], reg2`	`movhi   hi1(disp), reg1, r1`
	`tst1    lo(disp)[r1], reg2`

(b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`tst1    #label[reg1], reg2`	`movhi   hi1(#label), reg1, r1`
	`tst1    lo(#label)[r1], reg2`

`tst1    label[reg1], reg2`	`movhi   hi1(label), reg1, r1`
	`tst1    lo(label)[r1], reg2`

`tst1    $label[reg1], reg2`	`movhi   hi1($label), reg1, r1`
	`tst1    lo($label)[r1], reg2`

[Flag]

CY	---
OV	---
S	---
Z	1 if the specified bit is 0, 0 if not[Note]
SAT	---

Note The flag values shown here are those existing prior to the execution of this instruction, not those after the execution.

[Caution]

- If disp is omitted, the as850 assumes 0.
- If a relative expression with #label or a relative expression with #label and with hi( ), lo( ), or hi1( ) applied is specified as disp, [reg1] that follows the expression can be omitted. If omitted, the as850 assumes [r0] to be specified.
- If a relative expression having $label or a relative expression having $label and with hi( ), lo( ), or hi1( ) applied is specified as disp, [reg1] that follows the expression can be omitted. If omitted, the as850 assumes that [gp] is specified.
- If a peripheral I/O register name that is defined in the device file is specified as disp, [reg1] that follows the name can be omitted. If omitted, the as850 assumes that [r0] is specified.

3.8 Stack Manipulation Instructions

This section describes the stack manipulation instructions.

Next table lists the instructions described in this section.

Table 3 - 13 Stack Manipulation Instructions

Instruction	Meaning
pop	Pop from stack area (single register)
popm	Pop from stack area (multiple registers)
push	Push to stack area (single register)
pushm	Push to stack area (multiple registers)

pop

[Overview]

Pop

[Syntax]

```
(1) pop          reg
```

[Function]

Pops the value of the register specified by the operand from the stack area.

[Description]

- When the pop instruction is executed, the as850 executes instruction expansion to generate two or more machine instructions.

`pop      reg`	`ld.w     [sp], reg`
	`add      4, sp`

[Flag]

Set by the add instruction.

CY	1 if a carry occurs from MSB (Most Significant Bit) , 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

popm

[Overview]

Pop Multiple

[Syntax]

```
(1) popm        reg1, reg2, ..., regN
```

[Function]

Pops the values of the registers specified by the operand from the stack area in the sequence in which the registers are specified.

Up to 32 registers can be specified by the operand.

[Description]

- When the popm instruction is executed, the as850 executes instruction expansion to generate two or more machine instructions.

When there are three or fewer registers

`popm    reg1,..., regN`	`ld.w    4 * 0[sp], reg1`
	`    :`
	`ld.w    4 * (N - 1)[sp], regN`
	`add     4 * N, sp`

When there are four or more registers

`popm    reg1, reg2, ..., regN`	`ld.w    4 * 0[sp], reg1`
	`ld.w    4 * 1[sp], reg2`
	`    :`
	`ld.w    4 * (N - 1)[sp], regN`
	`addi    4 * N, sp, sp`

[Flag]

Set by the add/addi instruction.

CY	1 if a carry occurs from MSB (Most Significant Bit) , 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

push

[Overview]

Push

[Syntax]

```
(1) push        reg
```

[Function]

Pushes the value of the register specified by the operand to the stack area.

[Description]

- When the push instruction is executed, the as850 executes instruction expansion to generate two or more machine instructions.

`push    reg`	`add      -4. sp`
	`st.w    reg, [sp]`

[Flag]

Set by the add instruction.

CY	1 if a carry occurs from MSB (Most Significant Bit) , 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

pushm

[Overview]

Push Multiple

[Syntax]

```
(1) pushm       reg1, reg2, ..., regN
```

[Function]

Pushes the values of the registers specified by the operand to the stack area.

Up to 32 registers can be specified by the operand.

[Description]

- When the pushm instruction is executed, the as850 executes instruction expansion to generate two or more machine instructions.

When there are three or fewer registers

`pushm   reg1, reg2, ..., regN`	`add      -4 * N, sp`
	`st.w    regN, 4 * (N - 1)[sp]`
	`    :`
	`st.w    reg2, 4 * 1[sp]`
	`st.w    reg1, 4 * 0[sp]`

When there are four or more registers

`pushm   reg1, reg2, ..., regN`	`addi     -4 * N, sp, sp`
	`st.w    regN, 4 * (N - 1)[sp]`
	`    :`
	`st.w    reg2, 4 * 1[sp]`
	`st.w    reg1, 4 * 0[sp]`

[Flag]

Set by the add/addi instruction.

CY	1 if a carry occurs from MSB (Most Significant Bit), 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

3.9 Special Instructions

This section describes the special instructions.

Next table lists the instructions described in this section.

Table 3 - 14 Special Instructions

Instruction	Meaning
callt	Table reference call **[V850E]**
ctret	Returns from callt **[V850E]**
dbret	Returns from debug trap **[V850E]**
dbtrap	Debug trap **[V850E]**
di	Disables maskable interrupt
dispose	Deletes stack frame (postprocessing of function) **[V850E]**
ei	Enables maskable interrupt
halt	Stops the processor
ldsr	Loads to system register
nop	No operation
prepare	Generates stack frame (preprocessing of function) **[V850E]**
reti	Returns from trap or interrupt routine
stsr	Stores contents of system register
switch	Table reference branch **[V850E]**
trap	Software trap

callt

[V850E]

[Overview]

Call With Table Look Up

[Syntax]

```
(1) callt      imm6
```

The following can be specified as imm6:

- Absolute expression having a value of up to 6 bits

[Function]

- Performs processing in the following sequence[Note].

(1) Saves the values of the return PC and PSW to CTPC and CTPSW.

(2) Generates a table entry address by shifting the value specified by the operand 1 bit to the left as an offset value from CTBP(CALLT Base Pointer) and by adding it to the CTBP value.

(3) Loads unsigned halfword data from the generated table entry address.

(4) Adds the loaded value to the CTBP value to generate an address.

(5) Branches to the generated address.

Note For details of the system registers, refer to the Relevant Device's Architecture User's Manual of each device.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

ctret

[V850E]

[Overview]

Return From Callt

[Syntax]

```
(1) ctret
```

[Function]

- Returns from the processing by callt. Performs the processing in the following sequence[Note]:

(1) Extracts the return PC and PSW from CTPC and CTPSW.

(2) Sets the extracted values in the PC and PSW and transfers control.

Note For details of the system registers, refer to the Relevant Device's Architecture User's Manual of each device.

[Flag]

CY	Extracted value
OV	Extracted value
S	Extracted value
Z	Extracted value
SAT	Extracted value

dbret

[V850E]

[Overview]

Return From Debug Trap

[Syntax]

```
(1) dbret
```

[Function]

- Returns from debug trap[Note].

Note For details of the function, refer to the Relevant Device's Architecture User's Manual of each device.

[Flag]

CY	Extracted value
OV	Extracted value
S	Extracted value
Z	Extracted value
SAT	Extracted value

dbtrap

[V850E]

[Overview]

Debug Trap

[Syntax]

```
(1) dbtrap
```

[Function]

- Causes debug trap[Note].

Note For details of the function, refer to the Relevant Device's Architecture User's Manual of each device.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

di

[Overview]

Disable Interrupt

[Syntax]

```
(1) di
```

[Function]

- Sets the ID bit of the PSW to 1 and disables acknowledgement of maskable interrupts since this instruction has already been executed.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---
ID	1

dispose

[V850E]

[Overview]

Function Dispose

[Syntax]

```
(1) dispose    imm, list
(2) dispose    imm, list, [reg]
```

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits

The following can be specified as list. list specifies the 12 registers that can be popped by the dispose instruction.

- Register
  Specify the registers (r20 to r31) to be popped, delimiting each with a comma.
- Constant expression having a value of up to 12 bits
  The 12 bits and 12 registers correspond as follows:

bit11											bit0
r30	r24	r25	r26	r27	r20	r21	r22	r23	r28	r29	r31

The following two specifications are equivalent.

`dispose 0x10, r26, r29, r31`	`dispose 0x10, 0x103`

[Function]

The dispose instruction performs the postprocessing of a function.

- Syntax (1)

(1) Adds the value of the absolute expression specified by the first operand to the stack pointer (sp)[Note] and sets sp in the register saving area.

(2) Pops one of the registers specified by the second operand and adds 4 to sp.

(3) Repeatedly executes (2) until all the registers specified by the second operand have been popped.

Note Since the value actually added to sp by the machine instruction is imm shifted 2 bits to the left, the assembler shifts the specified imm 2 bits to the right in advance and reflects it in the code.

- Syntax (2)

(1) Adds the value of the absolute expression specified by the first operand to the stack pointer (sp)[Note] and sets sp in the register saving area.

(2) Pops one of the registers specified by the second operand and adds 4 to sp.

(3) [Repeatedly executes (2) until all the registers specified by the second operand have been popped.

(4) Sets the register value specified by the third operand in the program counter (PC).

Note Undefined symbol and label reference.

[Description]

- If the following is specified for imm, the as850 generates one dispose machine instruction.

(1) Absolute expression having a value in the range of 0 to 127

`dispose imm, list`	`dispose imm, list`

`dispose imm, list, [reg]`	`dispose imm, list, [reg]`

If anything other than a constant expression is specified as list, the as850 outputs the following message and stops assembling.

```
E3249: illegal syntax
```

- When the following is specified as imm, the as850 executes instruction expansion to generate two or more machine instructions.

(a) Absolute expression exceeding the range of 0 to 127, but within the range of 0 to 32,767

`dispose imm, list`	`movea   imm, sp, sp`
	`dispose 0, list`

`dispose imm, list, [reg]`	`movea   imm, sp, sp`
	`dispose 0, list, [reg]`

(b) Absolute expression having a value exceeding the range of 0 to 32,767

`dispose imm, list`	`mov     imm, r1`
	`add     r1, sp`
	`dispose 0, list`

`dispose imm, list, [reg]`	`mov     imm, r1`
	`add     r1, sp`
	`dispose 0, list, [reg]`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

Note If the add instruction is generated as a result of instruction expansion, the flag value may be affected.

[Caution]

- An address consisting of the two lower bits specified by sp is masked to 0 even though misalign access is enabled. In sp, set a value which is aligned with a four-byte boundary.
- If r0 is specified by the [reg] in syntax (2), the as850 outputs the following message and stops assembling.

```
E3240: illegal operand (can not use r0 as destination in V850E mode)
```

ei

[Overview]

Enable Interrupt

[Syntax]

```
(1) ei
```

[Function]

- Sets the ID bit of the PSW to 0, and enables acknowledgment of maskable interrupt from the next instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---
ID	0

halt

[Overview]

Halt

[Syntax]

```
(1) halt
```

[Function]

- Stops the processor and sets it in the HALT status. The HALT status can be released by a maskable interrupt, NMI, or reset.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

ldsr

[Overview]

Load System Register

[Syntax]

```
(1) ldsr        reg, regID
```

The following can be specified as regID:

- Absolute expression having a value of up to 5 bits

[Function]

- Stores the value of the register specified by the first operand in the system register[Note] indicated by the system register number specified by the second operand.

Note For details of the system registers, refer to the Relevant Device's Hardware User's Manual provided with the each device and the table below.

Table 3 - 15 System Register Numbers (ldsr)

Number	System Register	
0	Status saving register for interrupt	EIPC
1	Status saving register for interrupt	EIPSW
2	Status saving register for NMI	FEPC
3	Status saving register for NMI	FEPSW
4	Interrupt source register[Note]	ECR
5	Program status word	PSW
6-31	Reserved	

Note The interrupt source register cannot be specified by an operand and accessing it is prohibited.

Table 3 - 16 System Register Numbers [V850E/MS1] (ldsr)

Number	System Register	
0	Status saving register for interrupt	EIPC
1	Status saving register for interrupt	EIPSW
2	Status saving register for NMI	FEPC
3	Status saving register for NMI	FEPSW
4	Interrupt source register[Note]	ECR
5	Program status word	PSW
6-15	Reserved	
16	Status saving register for CALLT execution	CTPC
17	Status saving register for CALLT execution	CTPSW
18	Status saving register for exception/debug trap	DBPC
19	Status saving register for exception/debug trap	DBPSW
20	CALLT base pointer	CTBP
21-31	Reserved	

Note The interrupt source register cannot be specified by an operand and accessing it is prohibited.

Table 3 - 17 System Register Numbers [V850E1] (ldsr)

Number	System Register	
0	Status saving register for interrupt	EIPC
1	Status saving register for interrupt	EIPSW
2	Status saving register for NMI	FEPC
3	Status saving register for NMI	FEPSW
4	Interrupt source register[Note 1]	ECR
5	Program status word	PSW
6-15	Reserved	
16	Status saving register for CALLT execution	CTPC
17	Status saving register for CALLT execution	CTPSW
18	Status saving register for exception/debug trap[Note 2]	DBPC
19	Status saving register for exception/debug trap[Note 2]	DBPSW
20	CALLT base pointer	CTBP
21	Debug interface register[Note 2]	DIR
22	Break point control registers 0, 1[Notes 2, 3]	BPC0, BPC1
23	Program ID register	ASID
24	Break point address set registers 0, 1[Notes 2, 3]	BPAV0, BPAV1
25	Break point address mask registers 0, 1[Notes 2, 3]	BPAM0, BPAM1
26	Break point data set registers 0, 1[Notes 2, 3]	BPDV0, BPDV1
27	Break point data mask registers 0, 1[Notes 2, 3]	BPDM0, BPDM1
28-31	Reserved	

Notes 1 The interrupt source register cannot be specified by an operand and accessing it is prohibited.

2 Access is enabled only in the debug mode.

3 The register actually accessed is specified by the CS bit of the DIR register.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

If the program status word (PSW) is specified as the system register, the value of the corresponding bit of reg is set as each flag.

[Caution]

- When returning by the reti instruction after setting (1) bit 0 of EIPC, FEPC, or CTPC to 0 by the ldsr instruction, the value of bit 0 is ignored (because bit 0 of PC is fixed to 0). When setting a value to EIPC, FEPC, or CTPC, set an even value (bit 0 = 0).
- If an absolute expression having a value exceeding the range of 0 to 31 is specified as regID, the as850 outputs the following message, then continues assembling using the lower 5 bits[Note] of the specified value.

Note The ldsr machine instruction takes an immediate value in the range of 0 to 31 (0x0 to 0x1f) as the second operand.

```
W3011: illegal operand (range error in immediate)
```

- If a reserved register number, the number of a register which cannot be accessed (such as ECR) or the number of a register which can be accessed only in the debug mode is specified as regID, the as850 outputs the following message and continues assembling as is

```
W3018: illegal regID for ldsr
```

nop

[Overview]

No Operation

[Syntax]

```
(1) nop
```

[Function]

- Nothing is executed. This instruction can be used to allocate an area during an instruction sequence or to insert a delay cycle during instruction execution.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

prepare

[V850E]

[Overview]

Function Prepare

[Syntax]

```
(1) prepare    list, imm1
(2) prepare    list, imm1, imm2
(3) prepare    list, imm1, sp
```

The following can be specified as imm1/imm2.

- Absolute expression having a value of up to 32 bits

list specifies the 12 registers that can be pushed by the prepare instruction.The following can be specified as list.

- Register
  Specify the registers (r20 to r31) to be pushed, delimiting each with a comma.
- Constant expression having a value of up to 12 bits
  The 12 bits and 12 registers correspond as follows:

```
bit11                                        bit0
 r30 r24 r25 r26 r27 r20 r21 r22 r23 r28 r29 r31
```

The following two specifications are equivalent.

`prepare r26, r29, r31, 0x10`	`prepare 0x103, 0x10`

[Function]

The prepare instruction performs the preprocessing of a function.

- Syntax (1)

(1) Pushes one of the registers specified by the first operand and subtracts 4 from the stack pointer (sp).

(2) Repeatedly performs (1) until all the registers specified by the first operand have been pushed.

(3) Subtracts the value of the absolute expression specified by the second operand from sp[Note] and sets sp in the register saving area.

- Syntax (2)

(1) Pushes one of the registers specified by the first operand and subtracts 4 from sp.

(2) Repeatedly performs (1) until all the registers specified by the first operand have been pushed.

(3) Subtracts the value of the absolute expression specified by the second operand from sp[Note] and sets sp to the register saving area.

(4) Sets the value of the absolute expression specified by the third operand in ep.

- Syntax (3)

(1) Pushes one of the registers specified by the first operand and subtracts 4 from sp.

(2) Repeatedly performs (1) until all the registers specified by the first operand have been pushed.

(3) Subtracts the value of the absolute expression specified by the second operand from sp[Note] and sets sp in the register saving area.

(4) Sets the value of sp specified by the third operand in ep.

Note Since the value actually subtracted from sp by the machine instruction is imm1 shifted 2 bits to the left, the assembler shifts the specified imm1 2 bits to the right in advance and reflects it in the code.

[Description]

- If the following is specified for imm1, the as850 generates one prepare machine instruction.

(a) Absolute expression having a value in the range of 0 to 127

`prepare list, imm1`	`prepare list, imm1`

`prepare list, imm1, imm2`	`prepare list, imm1, imm2`

`prepare list, imm1, sp`	`prepare list, imm1, sp`

If anything other than a constant expression is specified as list, the as850 outputs the following message and stops assembling.

```
E3249: illegal syntax
```

- When the following is specified as imm1, the as850 executes instruction expansion to generate two or more machine instructions.

(a) Absolute expression exceeding the range of 0 to 127, but within the range of 0 to 32,767

`prepare list, imm1`	`prepare list, 0`
	`movea   -imm1, sp, sp`

`prepare list, imm1, imm2`	`prepare list, 0, imm2`
	`movea   -imm1, sp, sp`

`prepare list, imm1, sp`	`prepare list, 0, sp`
	`movea   -imm1, sp, sp`

(b) Absolute expression having a value exceeding the range of 0 to 32,767

`prepare list, imm1`	`prepare list, 0`
	`mov     imm1, r1`
	`sub     r1, sp`

`prepare list, imm1, imm2`	`prepare list, 0, imm2`
	`mov     imm1, r1`
	`sub     r1, sp`

`prepare list, imm1, sp`	`prepare list, 0, sp`
	`mov     imm1, r1`
	`sub     r1, sp`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

Note If a sub instruction is generated as a result of instruction expansion, the flag value may be affected.

[Caution]

- An address consisting of the two lower bits specified by sp is masked to 0 even though misalign access is enabled. In sp, set a value which is aligned with a four-byte boundary.

reti

[Overview]

Return from Trap or Interrupt

[Syntax]

```
(1) reti
```

[Function]

- Returns from a trap or interrupt routine[Note].

Note For details of the function, refer to the Relevant Device's Architecture User's Manual of each device.

[Flag]

CY	Extracted value
OV	Extracted value
S	Extracted value
Z	Extracted value
SAT	Extracted value

stsr

[Overview]

Store System Register

[Syntax]

```
(1) stsr        regID, reg
```

The following can be specified as regID:

- Absolute expression having a value of up to 5 bits

[Function]

- Stores the value of the system register[Note] indicated by the system register number specified by the first operand, to the register specified by the second operand.

Table 3 - 18 System Register Numbers (ldsr)

Number	System Register	
0	Status saving register for interrupt	EIPC
1	Status saving register for interrupt	EIPSW
2	Status saving register for NMI	FEPC
3	Status saving register for NMI	FEPSW
4	Interrupt source register[Note]	ECR
5	Program status word	PSW
6-31	Reserved	

Note For details of the system registers, refer to the Relevant Device's Hardware User's Manual provided with the each device and the table below.

Table 3 - 19 System Register Numbers [V850E/MS1] (stsr)

Number	System Register	
0	Status saving register for interrupt	EIPC
1	Status saving register for interrupt	EIPSW
2	Status saving register for NMI	FEPC
3	Status saving register for NMI	FEPSW
4	Interrupt source register	ECR
5	Program status word	PSW
6-15	Reserved	
16	Status saving register for CALLT execution	CTPC
17	Status saving register for CALLT execution	CTPSW
18	Status saving register for exception/debug trap	DBPC
19	Status saving register for exception/debug trap	DBPSW
20	CALLT base pointer	CTBP
21-31	Reserved	

Table 3 - 20 System Register Numbers [V850E1] (stsr)

Number	System Register	
0	Status saving register for interrupt	EIPC
1	Status saving register for interrupt	EIPSW
2	Status saving register for NMI	FEPC
3	Status saving register for NMI	FEPSW
4	Interrupt source register	ECR
5	Program status word	PSW
6-15	Reserved	
16	Status saving register for CALLT execution	CTPC
17	Status saving register for CALLT execution	CTPSW
18	Status saving register for exception/debug trap[Note 1]	DBPC
19	Status saving register for exception/debug trap[Note 1]	DBPSW
20	CALLT base pointer	CTBP
21	Debug interface register[Note 1]	DIR
22	Break point control registers 0, 1[Notes 1,2]	BPC0, BPC1
23	Program ID register	ASID
24	Break point address set registers 0, 1[Notes 1,2]	BPAV0, BPAV1
25	Break point address mask registers 0, 1[Notes 1,2]	BPAM0, BPAM1
26	Break point data set registers 0, 1[Notes 1,2]	BPDV0, BPDV1
27	Break point data mask registers 0, 1[Notes 1,2]	BPDM0, BPDM1
28-31	Reserved	

Notes 1 Access is enabled only in the debug mode.

2 The register actually accessed is specified by the CS bit of the DIR register.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- When returning by the reti instruction after setting (1) bit 0 of EIPC, FEPC, or CTPC to 0 by the ldsr instruction, the value of bit 0 is ignored (because bit 0 of PC is fixed to 0). When setting a value to EIPC, FEPC, or CTPC, set an even value (bit 0 = 0).
- If an absolute expression having a value exceeding the range of 0 to 31 is specified as regID, the as850 outputs the following message, then continues assembling using the lower 5 bits[Note] of the specified value.

Note The ldsr machine instruction takes an immediate value in the range of 0 to 31 (0x0 to 0x1f) as the second operand.

```
W3011: illegal operand (range error in immediate)
```

- If a reserved register number or the number of a register which can be accessed only in the debug mode is specified as regID, the as850 outputs the following message and continues assembling as is

```
W3018: illegal regID for stsr
```

switch

[V850E]

[Overview]

Jump With Table Look Up

[Syntax]

```
(1) switch      reg
```

[Function]

- Performs processing in the following sequence.

(1) Adds the value resulting from logically shifting the value specified by the operand 1 bit to the left to the first address of the table (address following the switch instruction) to generate a table entry address.

(2) Loads signed halfword data from the generated table entry address.

(3) Logically shifts the loaded value 1 bit to the left and sign-extends it to word length. Then adds the first address of the table to it to generate an address.

(4) Branches to the generated address.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If r0 is specified by reg, the as850 outputs the following message and stops assembling.

```
E3240: illegal operand (can not use r0 as source in V850E mode)
```

trap

[Overview]

Trap

[Syntax]

```
(1) trap        vector
```

The following can be specified for vector:

- Absolute expression having a value of up to 5 bits

[Function]

- Causes a software trap[Note].

Note For details of the function, refer to the Relevant Device's Architecture User's Manual of each device.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value falling outside the range of 0 to 31 is specified as vector, the as850 outputs the following message, continuing assembling using the lower 5 bits[Note] of the specified value.

```
W3011: illegal operand (range error in immediate)
```

Note The trap machine instruction takes an immediate value in the range of 0 to 31 (0x0 to 0x1f) as an operand.

CHAPTER 4 QUASI DIRECTIVES

This section describes the assembly language quasi directives supported by the CA850 assembler (as850).

4.1 Description of Format

The quasi directive of the assembly language supported by as850 are described in the following format.

A quasi directive performs the preprocessing necessary for the assembler to generate machine instructions and directs the assembler to define a section or input a file. It can also direct processing of output code and macro replacement.

Quasi directive

[Syntax]

Indicates the function of quasi directive syntax.

[Function]

Indicates the function of the quasi directive.

[Description]

Provides a supplementary description of the function of the quasi directive.

[Caution]

Describes the points to be noted when using the quasi directive.

[Example]

Provides an example of using the quasi directive.

4.2 Section Definition Quasi Directives

Using a section definition quasi directive, the as850 can allocate a code, generated for a source program (assembly language), to a specified section[Note].

Next table lists the section definition quasi directives described in this section.

Note The CA850 handles machine instructions and data in units called sections

Table 4 - 1 Section Definition Quasi Directives

Quasi directive	Meaning
.bss	Allocation to .bss section
.const	Allocation to .const section
.data	Allocation to .data section
.previous	(Re-)definition of section definition quasi directive preceding the section definition quasi directive that specifies the current section definition quasi directive
.sbss	Allocation to .sbss section
.sconst	Allocation to .sconst section
.sdata	Allocation to .sdata section
.sebss	Allocation to .sebss section
.section	Allocation to section of specified type
.sedata	Allocation to .sedata section
.sibss	Allocation to .sibss section
.sidata	Allocation to .sidata section
.text	Allocation to .text section
.tibss	Allocation to .tibss section
.tibss.byte	Allocation to .tibss.byte section
.tibss.word	Allocation to .tibss.word section
.tidata	Allocation to .tidata section
.tidata.byte	Allocation to .tidata.byte section
.tidata.word	Allocation to .tidata.word section
.vdbstrtab	Allocation to .vdbstrtab section
.vdebug	Allocation to .vdebug section
.vline	Allocation to .vline section

If the assembler source program does not contain a section definition quasi directive, all sections generated by that program will become .text sections.

.bss

[Syntax]

```
.bss
```

[Function]

Allocates, to the .bss section[Note], a code generated for the assembly language source program, between this quasi directive and the subsequent section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this pquasi directive and the end of the assembler source file.

Note Reserved section having section name .bss, section type NOBITS, and section attribute AW.

[Description]

The .bss section is allocated to a memory range which can be referenced by using gp and a 32-bit displacement, specified by two instructions. This section has no initial value.

[Example]

Used as .bss section until the next section definition quasi directive.

```
    .bss
    .lcomm __stack, 0x100, 4
```

.const

[Syntax]

```
.const
```

[Function]

Allocates, to the .const sectionNote, a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .const, section type PROGBITS, and section attribute A.

[Description]

The .const section is allocated to a memory range which can be referenced by using r0 and a 32-bit displacement, specified by two instructions. This section is used for constant data (read-only).

[Example]

Used as .const section until the next section definition quasi directive.

```
    .const
    .align 4
    .globl _p, 4
_p:
    .word 10
```

.data

[Syntax]

```
.data
```

[Function]

Allocates, to the .data section[Note], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .data, section type PROGBITS, and section attribute AW.

[Description]

The .data section is allocated to a memory range which can be referenced by using gp and a 32-bit displacement, specified by two instructions. This section has an initial value.

[Example]

Used as .data section until the next section definition quasi directive.

```
    .data
    .align 4
    .globl _p, 4
_p:
    .word 10
```

.previous

[Syntax]

```
.previous
```

[Function]

(Re-)specifies the section definition quasi directive preceding the section definition quasi directive specifying the current section definition quasi directive.

For example, if quasi directives .data, .text, then .previous are specified, the specification of the .previous quasi directive is equivalent to specifying the .data quasi directive.

[Example]

.previous is equivalent to .data.

```
    .data
    .align 4
    .globl _p, 4
_p:
    .word 10
    .text
lab:
    jbr     LL
    .previous
```

.sbss

[Syntax]

```
.sbss
```

[Function]

Allocates, to the .sbss sectionNote, a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .sbss, section type NOBITS, and section attribute AWG.

[Description]

The .sbss section is allocated to a memory range which can be referenced with a single instruction by using gp and a 16-bit displacement (up to 64 KB, including the size of the .sdata section). This section has no initial value.

[Example]

Used as .sbss section until the next section definition quasi directive.

```
    .sbss
    .globl _1, 4
    .lcomm _1, 4, 4
```

.sconst

[Syntax]

```
.sconst
```

[Function]

Allocates, to the .sconst section[Note], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .sconst, section type PROGBITS, and section attribute A.

[Description]

The .sconst section is allocated to a memory range which can be referenced with a single instruction by using r0 and a 16-bit displacement (up to 32 KB in the positive direction, relative to r0). This section is used for constant data (read-only).

[Example]

Used as .sconst section until the next section definition quasi directive.

```
    .sconst
    .align 4
    .globl _p, 4
_p:
    .word 10
```

.sdata

[Syntax]

```
.sdata
```

[Function]

Allocates, to the .sdata section[Note], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .sdata, section type PROGBITS, and section attribute AWG.

[Description]

The .sdata section is allocated to a memory range which can be referenced with a single instruction by using gp and a 16-bit displacement (up to 64 KB, including the size of the .sbss section). This section has an initial value.

[Example]

Used as .sdata section until the next section definition quasi directive.

```
    .sdata
    .align 4
    .globl _p, 4
_p:
    .word 10
```

.sebss

[Syntax]

```
.sebss
```

[Function]

Allocates, to the .sebss section[Note], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .sebss, section type NOBITS, and section attribute AW.

[Description]

The .sebss section is allocated to a memory range which can be referenced with a single instruction by using ep and a 16-bit displacement (up to 32 KB in the negative direction, relative to ep). It cannot be allocated, however, to the lower addresses used for the .sedata section within that range. This section has no initial value.

[Example]

Used as .sebss section until the next section definition quasi directive.

```
    .sebss
    .globl _1, 4
    .lcomm _1, 4, 4
```

.section

[Syntax]

```
.section "section-name"[, section-type]
```

[Function]

Allocates, to a section of the type specified by the second operand in the section name specified by the first operand, a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Seven section types are supported. These are listed in Table 3-19[Note].

Note Uppercase characters can also be used to specify a section type (for example, TEXT can be specified instead of text).

Table 4 - 2 Section Types

Type	Meaning
bss	bss-attribute section Section having section type NOBITS and section attribute AW
const	const-attribute section Section having section type PROGBITS and section attribute A
data	data-attribute section Section having section type PROGBITS and section attribute AW
sbss	sbss-attribute section Section having section type NOBITS and section attribute AWG
sdata	sdata-attribute section Section having section type PROGBITS and section attribute AWG
text	text-attribute section Section having section type PROGBITS and section attribute AX
comment	comment-attribute section Section with section type PROGBITS and without any section attribute

[Example]

Defines a data-attribute section named sec

```
    .section "sec", data
    .align 4
    .globl _p, 4
_p:
    .word 10
```

[Caution]

- Section names .pro_epi_runtime, .text, .data, .bss, .sdata, .sbss, .sconst, .const, .sidata, .sibss, .sedata, .sebss, .tidata, .tibss, .tidata.byte, .tibss.byte, .tidata.word, .tibss.word, and .version are reserved for use by the CA850. The correspondence between these reserved section names and the section types is detailed in the table below.

Table 4 - 3 Correspondence between These Reserved Section Names and The Section Types

Reserved Section Name	Section Type
.pro_epi_runtime .text	text
.data .sedata .sidata .tidata .tidata.byte .tidata.word	data
.bss .sebss .sibss .tibss .tibss.byte .tibss.word	bss
.sdata	sdata
.sbss	sbss
.const .sconst	const
.version	comment

If these section names are specified by the first operand, therefore, either the second operand must be omitted or the section type corresponding to each reserved section must be specified. If a type other than the corresponding type is specified, the as850 outputs the following message then stops assembling.

```
F3504: illegal section kind
```

- If a name other than that of one of the above reserved sections is specified by the first operand, and if the second operand is omitted, it is assumed that text is specified as the section type.
- If two or more different section types are specified for a single section having a specific name, the as850 outputs the following message then stops assembling.

```
F3504: illegal section kind
```

- If an interrupt request name defined in the device file is specified as the first operand, the link editor automatically allocates the section to the corresponding handler address. The allocation address, therefore, cannot be specified by using the link editor for a section for which an interrupt request name has been specified. An interrupt request name must not be specified for other than an interrupt handler section.

Example of using interrupt request name

```
    .section "RESET", text
    jr  __start
```

.sedata

[Syntax]

```
.sedata
```

[Function]

Allocates, to the .sedata section[Note], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .sedata, section type PROGBITS, and section attribute AW.

[Description]

The .sedata section is allocated to a memory range which can be referenced with a single instruction by using ep and a 16-bit displacement (up to 32 KB in the negative direction, relative to ep). It cannot be allocated, however, to the higher addresses used for the .sebss section within that range. This section has an initial value.

[Example]

Used as .sedata section until the next section definition quasi directive.

```
    .sedata
    .align 4
    .globl _p, 4
_p:
    .word 10
```

.sibss

[Syntax]

```
.sibss
```

[Function]

Allocates, to the .sibss section[Note], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .sibss, section type NOBITS, and section attribute AW.

[Description]

The .sibss section is allocated to a memory range that can be referenced with a single instruction by using ep and a 16-bit displacement (up to 32 KB in the positive direction from ep). It is allocated at an address higher by the size of the .tidata.byte, .tibss.byte, .tidata.word, .tibss.word, .tidata, .tibss, or .sidata section within that range. This section does not have an initial value (refer to Figure 2 - 1).

[Example]

Used as .sibss section until the next section definition quasi directive.

```
    .sibss
    .globl _1, 4
    .lcomm _1, 4, 4
```

.sidata

[Syntax]

```
.sidata
```

[Function]

Allocates, to the .sidata section[Note], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .sidata, section type PROGBITS, and section attribute AW

[Description]

The .sidata section is allocated to a memory range which can be referenced with a single instruction by using ep and a 16-bit displacement (up to 32 KB in the positive direction, relative to ep). It is allocated at an address higher by the size of the .tidata.byte, .tibss.byte, .tidata.word, .tibss.word, .tidata, or .tibss section within that range (refer to Figure 2 - 1).

[Example]

Used as .sidata section until the next section definition quasi directive.

```
    .sidata
    .align 4
    .globl _p, 4
_p:
    .word 10
```

.text

[Syntax]

```
.text
```

[Function]

Allocates, to the .text section[Note 1], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive.

Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file[Note 2].

Notes 1 Reserved section having section name .text, section type PROGBITS, and section attribute AX.

2 The as850 assumes .text to be specified two times before the assembly-language source program in a single assembler source file (for example, if ".word 1" is specified prior to a section definition quasi directive, it is allocated to the .text section). If, however, the .text section is not explicitly specified, and if a label definition, instruction, location counter control quasi directive, or area allocation quasi directive are not specified for the .text section that is specified as being the default section, the as850 does not generate the .text section.

[Example]

Used as .text section until the next section definition quasi directive.

```
    .text
    .align 4
    .globl __start
__start:
    mov #__tp_TEXT, tp
```

.tibss

[Syntax]

```
.tibss
```

[Function]

Allocates, to the .tibss section[Note], a code generated for the assembly language source program between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .tibss, section type NOBITS, and section attribute AW.

[Description]

The .tibss section is data without an initial value that is located in internal RAM of the V850 microcontrollers. Access to it is assumed to be by relative addressing using ep and the sld/sst instruction. The as850 and ld850 position .tibss at the address indicated by ep when none of .tidata.byte, .tibss.byte, .tidata.word, .tibss.word, and .tidata sections are used. When any of these sections is used, .tibss is positioned at the address obtained by adding the size of the .tidata.byte/.tibss.byte/.tidata.word/.tibss.word section used to the address indicated by ep (refer to Figure 2 - 1).

The range to be accessed when the sld and sst instructions are used varies with the data size. To effectively use the sld and sst instructions, therefore, it is recommended that byte data be allocated to the .tidata.byte/ .tibss.byte section and that halfword or larger data be allocated to the .tidata.word/.tibss.word section. If, however, the quantity of data to be stored in internal RAM is small, making such careful preparations for access areas unnecessary, this quasi directive can be used to allocate data to the .tibss section, thus eliminating the necessity to classify data by size.

[Example]

Used as .tibss section until the next section definition quasi directive.

```
    .tibss
    .globl _1, 4
    .lcomm _1, 4, 4
```

.tibss.byte

[Syntax]

```
.tibss.byte
```

[Function]

Allocates, to the .tibss.byte section[Note], a code generated for the assembly language source program between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .tibss.byte, section type NOBITS, and section attribute AW.

[Description]

The .tibss.byte section is located in internal RAM of the V850 microcontrollers. Access to it is assumed to be by relative addressing using ep and the sld/sst instruction. The sld/sst instruction can access

- Area of up to 128 bytes when byte data is accessed
- Area of up to 256 bytes when halfword or larger data is accessed

The as850 and ld850 classify sections into either .tidata.byte/.tibss.byte or .tidata.word/.tibss.word, depending on the size of the data, to position .tibss.byte at the address obtained by adding the size of the .tidata.byte section used to the address indicated by ep. This enables the area that can be accessed by the sld/sst instruction to be used effectively (refer to Figure 2 - 1).

It is recommended, therefore, that byte data without an initial value to be stored in internal RAM be allocated to the .tibss.byte section with this quasi directive[Note].

Note Byte data can be accessed even if allocated to the .tibss.word section.

[Example]

Used as .tibss.byte section until the next section definition quasi directive.

```
    .tibss.byte
    .globl  _1, 4
    .lcomm  _1, 4, 4
```

.tibss.word

[Syntax]

```
.tibss.word
```

[Function]

Allocates, to the .tibss.word section[Note], a code generated for the assembly language source program between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .tibss.word, section type NOBITS, and section attribute AW

[Description]

The .tibss.word section is located in internal RAM of the V850 microcontrollers. Access to it is assumed to be by relative addressing using ep and the sld/sst instruction. The sld/sst instruction can access

- Area of up to 128 bytes when byte data is accessed
- Area of up to 256 bytes when halfword or larger data is accessed

The as850 and ld850 classify sections into either .tidata.byte/.tibss.byte or .tidata.word/.tibss.word, depending on the size of the data, to position .tibss.word at the address obtained by adding the size of the .tidata.byte/.tibss.byte/.tidata.word section used to the address indicated by ep. This enables the area that can be accessed by the sld/sst instruction to be used effectively (refer to Figure 2 - 1).

It is recommended, therefore, that halfword or larger data without an initial value to be stored in internal RAM be allocated to the .tibss.word section with this quasi directive.

[Example]

Used as .tibss.word section until the next section definition quasi directive.

```
    .tibss.word
    .globl _1, 4
    .lcomm _1, 100000, 4
```

.tidata

[Syntax]

```
.tidata
```

[Function]

Allocates, to the .tidata section[Note], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .tidata, section type PROGBITS, and section attribute AW.

[Description]

The .tidata section is located in internal RAM of the V850 microcontrollers and is assumed to be accessed by relative addressing, using ep and the sld/sst instruction. The as850 and ld850 position .tidata at the address indicated by ep when none of .tidata.byte, .tibss.byte, .tidata.word, and .tibss.word sections are used. When any of these sections is used, .tidata is positioned at the address obtained by adding the size of the .tidata.byte/.tibss.byte/.tidata.word/.tibss.word section used to the address indicated by ep (refer to Figure 2 - 1).

For the sld and sst instructions, the range to be accessed varies with the data size. To effectively use the sld and sst instructions, therefore, it is recommended that byte data be allocated to the .tidata.byte/.tibss.byte section and that halfword or larger data be allocated to the .tidata.word/.tibss.word section. If, however, the amount of data to be stored in internal RAM is small, making such careful consideration for access areas unnecessary, this quasi directive can be used to allocate data to the .tidata section, thus eliminating the necessity to classify data by size.

[Example]

Used as .tidata section until the next section definition quasi directive.

```
    .tidata
    .align 4
    .globl _p, 4
_p:
    .word 10
```

.tidata.byte

[Syntax]

```
.tidata.byte
```

[Function]

Allocates, to the .tidata.byte section[Note], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .tidata.byte, section type PROGBITS, and section attribute AW.

[Description]

The .tidata.byte section is located in internal RAM of the V850 microcontrollers and is assumed to be accessed by relative addressing, using ep and the sld/sst instruction. The sld/sst instruction can access

- Area of up to 128 bytes when byte data is accessed.
- Area of up to 256 bytes when halfword or larger data is accessed.

The as850 and ld850 classify sections into either .tidata.byte/.tibss.byte or .tidata.word/.tibss.word, depending on the size of the data, to position .tidata.byte to the address indicated by ep, enabling effective use of the area that can be accessed by the sld/sst instruction (refer to Figure 2 - 1).

It is recommended, therefore, that byte data having an initial value to be stored in internal RAM be allocated to the .tidata.byte section by using this quasi directive[Note].

Note Byte data having an initial value can be accessed even if allocated to the .tidata.word section.

[Example]

Used as .tidata.byte section until the next section definition quasi directive.

```
    .tidata.byte
    .globl _p, 1
_p:
    .byte 1
```

.tidata.word

[Syntax]

```
.tidata.word
```

[Function]

Allocates, to the .tidata.word section[Note], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .tidata.word, section type PROGBITS, and section attribute AW.

[Description]

The .tidata.word section is located in internal RAM of the V850 microcontrollers and is assumed to be accessed by relative addressing, using ep and the sld/sst instruction. The sld/sst instruction can access

- Area of up to 128 bytes when byte data is accessed.
- Area of up to 256 bytes when halfword or larger data is accessed.

The as850 and ld850 classify sections into either .tidata.byte/.tibss.byte or .tidata.word/.tibss.word, depending on the size of the data, to position .tidata.word at the address obtained by adding the size of the .tidata.byte/.tibss.byte section used to the address indicated by ep. This enables the area that can be accessed by the sld/sst instruction to be used effectively (refer to Figure 2 - 1).

It is recommended, therefore, that halfword or larger data having an initial value to be stored in internal RAM be allocated to the .tidata.word section by using this quasi directive.

[Example]

Used as .tidata.word section until the next section definition quasi directive.

```
    .tidata.word
    .align 4
    .globl _p, 4
_p:
    .word 100000
```

.vdbstrtab

[Syntax]

```
.vdbstrtab
```

[Function]

Allocates, to the .vdbstrtab section[Note], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .vdbstrtab and section type STRTAB.

.vdebug

[Syntax]

```
.vdebug
```

[Function]

Allocates, to the .vdebug section[Note], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .vdebug and section type PROGBITS.

.vline

[Syntax]

```
.vline
```

[Function]

Allocates, to the .vline section[Note], a code generated for the assembly language source program, between this quasi directive and the next section definition quasi directive. Or, if there is no subsequent section definition quasi directive, allocates it between this quasi directive and the end of the assembler source file.

Note Reserved section having section name .vline and section type PROGBITS.

4.3 Symbol Control Quasi Directives

Using the symbol control quasi directives, the as850 can generate a symbol table entry, define symbols, and specify the size of the data indicated by a label.

Next table lists the symbol control quasi directives described in this section.

Table 4 - 4 Symbol Control Quasi Directives

Quasi directive	Meaning
.ext_ent_size	Flash table entry size
.ext_func	Generates a flash table entry
.file	Generates a symbol table entry (FILE type)
.frame	Generates a symbol table entry (FUNC type)
.set	Defines a symbol
.size	Specifies the size of the data indicated by label

Maintain the value of size[Note], as specified by the symbol control quasi directive, within 2^{31}. If a value of 2^{31} or more is specified, the as850 outputs the following message then stops assembling.

```
E3247: illegal size value
```

.ext_ent_size

[Syntax]

```
.ext_ent_size size
```

[Function]

Sets the value specified by the operand as the flash table entry size when an object file is generated.

Specify this instruction to use the function for relinking a flash area or external ROM.

[Description]

To specify a branch from an area that cannot be rewritten or replaced (boot area) to a rewritable or replaceable area (flash area), a branch table is generated at a specified address in the flash area by specifying this quasi directive and two-stage branch is performed via the table.

The entry size of this table is 4 bytes by default. A jr instruction is generated and execution can branch in a range of 22 bits from the branch instruction. If it is necessary to branch to an address exceeding the range of 22 bits from the branch instruction in this table, execution can branch over the entire 32-bit address space when 10 is specified by this instruction as the entry size in the case of the V850 core, and 8 is specified in the case of the V850Ex core.

[Caution]

- This quasi directive must be described in a source file which contains a relevant branch instruction (in the boot area) and a source file which contains a relevant label definition (in the flash area).
- The size specified by this quasi directive is the only value for the entire area, including the boot area and flash area. If a different size is specified, the as850 outputs the following message and stops assembling.

```
W3021: .ext_ent_size already specified, ignored.
```

If a different size is specified for two or more relocatable object files, an error occurs when linking is executed.

- It is recommended that all relevant label names be described in a single file and included in the source files of the boot area and flash area using the .include quasi directive. This prevents the contradictions described above.
- Specify 4 (default), 8 **[V850E]**, or 10 **[V850]** as the size. When a common object is created (when the -cn option is specified), 8 **[V850E]** must not be specified because the object must operate with both the V850 and V850Ex.

.ext_func

[Syntax]

```
.ext_func label-name, ID-value
```

[Function]

Generates a flash table entry having a label name and ID value specified by the operands when an object file is generated.

Specify this instruction to use the function for relinking a flash area or external ROM.

[Description]

To specify a branch from an area that cannot be rewritten or replaced (boot area) to a rewritable or replaceable area (flash area), a branch table is generated to a specified address in a flash area by specifying this quasi directive and two-stage branch is performed via the table.

[Caution]

This quasi directive must be written in a source file which contains a relevant branch instruction (in the boot area) and a source file which contains a relevant label definition (in the flash area).

- If the same label name is specified with a different ID value, the as850 outputs the following message then stops assembling

```
E3253: symbol "identifier" already defined as another id
```

- If the same ID value is specified with a different label name, the as850 outputs the following message then stops assembling.

```
E3252: id already defined as symbol "identifier"
```

- It is recommended that all relevant label names be written in a single file and included into source files of the boot area and flash area using the .include quasi directive. This prevents contradictions described above.
- The ID value must be a positive number. The size of a branch table to be allocated depends on the maximum ID value. NEC recommends that the ID value be specified without spaces.

.file

[Syntax]

```
.file "file-name"
```

[Function]

Generates a symbol table entry[Note] having a file name specified by the operand and type FILE when an object file is generated.

If this quasi directive does not exist in the input source file, it is assumed that ".file "input file name""has been specified, and a symbol table entry with the input file name and type FILE is generated.

Note The binding class is LOCAL.

.frame

[Syntax]

```
.frame label-name, size
```

[Function]

Generates a symbol table entry of a size specified by the second operand and type FUNC when the symbol table entry for the label specified by the first operand is generated upon the generation of the object file[Note].

Note This quasi directive is used for debugging at C language source level. Specify 0 in size to code for debugging at assembler level.

.set

[Syntax]

```
.set symbol-name, value
```

[Function]

Defines a symbol having a symbol name specified by the first operand and a value(Integer value) specified by the second operand.

If the .set quasi directive is specified for a given symbol more than once within a single assembler source file, reference to that symbol will have the following value, depending on the position of that reference.

- If the reference appears between the beginning of the file and the first .set quasi directive for that symbol
  Value specified with the last .set quasi directive for that symbol
- If the reference does not appear between a certain .set quasi directive and the next .set quasi directive, or if there is no subsequent .set quasi directive, between the first .set quasi directive and the end of the assembler source file
  Value specified by that .set quasi directive

[Caution]

- Any label reference or undefined symbol reference must not be used to specify a value. Otherwise, the as850 outputs the following message then stops assembling.

```
E3203: illegal expression (string)
```

- If a label name, a macro name defined by the .macro quasi directive, or a symbol of the same name as a formal parameter of a macro is specified, the as850 outputs the following message and stops assembling.

```
E3212: symbol already define as string
```

[Example]

Defines the value of symbol sym1 as 0x10

```
    .set sym1, 0x10
```

.size

[Syntax]

```
.size label-name, size
```

[Function]

Specifies the size specified by the second operand as the size of the data indicated by the label specified by the first operand[Note].

Note If the size has already been set, the previously specified value is overwritten.

[Caution]

If the -A option of the link editor of the CA850 is used, set the size of the data to be allocated to the sdata-attribute section (actually, the label subject to gp offset reference) by using this quasi directive or the .globl quasi directive when defining the data[Note].

Note Otherwise, valid information cannot be obtained by specifying the -A option of the link editor.

[Example]

Assumes size of label1 to be 15

```
    .size label1, 15
```

4.4 Location Counter Control Quasi Directives

Using the location counter control quasi directive, the as850 can align or advance the value of the location counter[Note].

Next table lists the location counter control quasi directives described in this section.

Note A location counter exists in each section and is initialized to 0 when the first section definition quasi directive for the corresponding section in that file appears.

Table 4 - 5 Location Counter Control Quasi Directives

Quasi Directive	Meaning
.align	Aligns the value of the location counter
.org	Advances the value of the location counter

If the location counter control quasi directive is specified in the sbss- or bss-attribute section, the as850 outputs the following message then stops assembling.

```
E3246: illegal section
```

.align

[Syntax]

```
.align alignment-condition[, fill-value]
```

[Function]

Aligns the value of the location counter for the current section, specified by the previously specified section definition quasi directive under the alignment condition specified by the first operand.

If a hole results from aligning the value of the location counter, it is filled with the fill value specified by the second operand, or with the default value of 0.

For example, if .align 4 is specified while the current value of the location counter is 3, the value of the location counter is aligned, according to the alignment condition of 4 (word boundary), to 4, and the 1-byte hole that results is filled with the default value of 0.

[Caution]

- Specify an even number of 2 or more, but less than 2^{31}, as the alignment condition. Otherwise, the as850 outputs the following message then stops assembling.

```
E3200: illegal alignment value
```

- Specify a 1-byte value as the fill value. If a value of more than 1 byte is specified, the lowermost 1-byte is used.
- If this quasi directive is used with an alignment condition of 4 or more, as specified by the sdata-attribute section, valid information may not be obtained when a guideline value for determining the size of the data to be allocated to the sdata/sbss-attribute section is displayed (by using the -A option of the ld850).
- This quasi directive merely aligns the value of the location counter in a specified file for the section. It does not align an absolute address[Note 1] or an offset in a section[Note 2].

Notes 1 Offset from address 0 in linked object file

2 Offset from the first address of the section (output section) to which that section is allocated in a linked object file

[Example]

Aligns at 16 bytes

```
    .align 16
```

.org

[Syntax]

```
.org value
```

[Function]

Advances the value of the location counter for the current section, specified by the previously specified section definition quasi directive, to the value(Less than 2^{31}) specified by the operand.

If a hole results from advancing the value of the location counter, it is filled with 0.

[Caution]

- If a value that is smaller than the current value of the location counter is specified, the as850 outputs the following message then stops assembling.

```
E3244: illegal origin value value
```

- If this quasi directive is used in the sdata-attribute section, valid information may not be obtained when a guideline value for determining the size of the data to be allocated to the sdata/sbss-attribute section is displayed (by using the -A option of the ld850).
- This quasi directive merely advances the value of the location counter in a specified file for the section. It does not specify either an absolute address[Note 1] or an offset in a section[Note 2].

Notes 1 Offset from address 0 in a linked object file.

2 Offset from the first address of the section (output section) to which that section is allocated in a linked object file.

[Example]

Advances the location counter value 16 bytes

```
    .org 16
```

4.5 Area Allocation Quasi Directives

Using area allocation quasi directives, the as850 can allocate an area and set a value for that area.

Next table lists the area allocation quasi directives described in this section.

Table 4 - 6 Area Allocation Quasi Directives

Quasi Directive	Meaning
.byte	Allocates a 1-byte area
.float	Sets a floating-point value
.hword	Allocates a 1-halfword area
.lcomm	Defines a label that allocates an area
.shword	Allocates a 1-halfword area **[V850E]**
.space	Allocates an area for size
.str	Allocates an area for string
.word	Allocates a 1-word area

If an area allocation quasi directive other than the .lcomm quasi directive is specified in the sbss- or bss-attribute section, the as850 outputs the following message then stops assembling.

```
E3246: illegal section
```

Maintain the values of size (Number of bytes) and alignment condition, specified with the area allocation quasi directive, within 231. If a value of 231 or more is specified, the as850 outputs the following message then stops assembling.

```
E3247: illegal size value
    or
E3200: illegal alignment value
```

.byte

[Syntax]

```
.byte value[, value] ...
.byte bit-width:value[, bit-width:value] ...
```

[Function]

- The first part of this quasi directive instructs the allocation of a 1-byte area for each operand, and the storing of the value of the lowermost byte of the specified value in the allocated area.
- The second part instructs the allocation of an area of the specified bit width and stores the specified value into the allocated area.

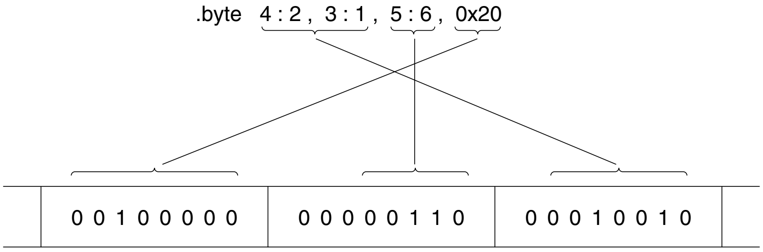
(1) Specify the bit width as a value between 0 and 8.

(2) If the specified bit width exceeds the byte width, it is masked by the byte width.

(3) A value specified first and having the bit width is allocated starting from the least significant bit of the byte area. If the area exceeds the byte boundary as a result of allocating an area immediately after the area to which the value with the previous bit width has been allocated, the second value is allocated starting from the byte boundary (refer to Figure 4 - 1).

(4) If a hole results, it is filled with 0.

Figure 4 - 1 Example of Allocation with Bit Width Specified



- The above two specifications can be made together with one .byte quasi directive (refer to Figure 4 - 1).

[Example]

Allocates 1 byte and stores 1

```
     .tidata.byte
     .align 4
     .globl _p, 4
_p:
     .byte 1
```

.float

[Syntax]

```
.float value [, value] ...
```

[Function]

Allocates a 1-word area for each operand, and stores the specified floating-point value in the allocated area[Note].

Note If an integer constant is specified, a 1-word area is allocated, and the specified integer constant is stored in the allocated area.

[Example]

Allocates 1 word and stores 1.2345

```
    .sidata
    .align 4
    .globl _p, 4
_p:
    .float 1.2345
```

.hword

[Syntax]

```
.hword value[, value] ...
.hword bit-width:value[, bit-width:value] ...
```

[Function]

- The first part of this quasi directive instructs the allocation of a 1-halfword area (2 bytes) for each operand, and the storing of the value of the lower 1 halfword of the specified value into the allocated area.
- The second part of this instruction instructs the allocation of an area of the specified bit width, and the storing of the specified value into the allocated area.

(1) Specify the bit width as a value between 0 and 16.

(2) If the specified value exceeds the halfword width, it is masked by the halfword width.

(3) A value declared first and having the bit width is allocated from the least significant bit position in the halfword area. If the halfword boundary of the area is exceeded as a result of allocating an area immediately after the area to which the value having the previous bit width has been allocated, the value having the bit width is allocated starting from the halfword boundary.

(4) If a hole results, it is filled with 0.

- The above two specifications can be made together for each .hword quasi directive.

[Example]

Allocates 1 halfword and stores 100

```
    .tidata
    .align 4
    .globl _p, 4
_p:
    .hword 100
```

.lcomm

[Syntax]

```
.lcomm label-name, size, alignment-condition
```

[Function]

Aligns the value of the location counter for the current section, specified by the previously specified section definition quasi directive, under the alignment condition specified by the third operand, allocates an area of the size specified by the second operand, and defines a local label[Note], having a label name specified by the first operand, at the first address of the allocated area.

Note Local symbol (symbol having binding class LOCAL).

[Caution]

- The current section, specified by the previously specified section definition quasi directive, must be an sbss- or bss-attribute section (refer to Table 4 - 2). If this quasi directive is specified for any other section, the as850 outputs the following message then stops assembling.

```
E3246: illegal section
```

- If this quasi directive is used by specifying an alignment condition of 4 or greater in the sbss-attribute section, valid information may not be obtained when a guideline value for determining the size of the data to be allocated to the sdata/sbss-attribute section is displayed (by using the -A option of the ld850).

[Example]

Assumes size of __stack label to be 0x100 for 4-byte alignment.

```
    .bss
    .lcomm __stack, 0x100, 4
```

.shword

[V850E]

[Syntax]

```
.shword value[, value] ...
.shword bit-width:value[, bit-width:value] ...
```

[Function]

- The first part of the .shword quasi directive allocates an area of 1 halfword to each operand, shifts a specified value 1 bit to the right, and stores it in the allocated area.
- The second part of the .shword quasi directive allocates an area of the specified bit width, shifts a specified value 1 bit to the right, and stores it in the allocated area.

(1) Specify the bit width as a value between 0 and 16.

(2) If the specified value exceeds the halfword width, it is masked by the halfword width.

(3) A value that is declared first and has the bit width is allocated from the least significant bit position in the halfword area. If the halfword boundary of the area is exceeded as a result of allocating an area immediately after the area to which the value with the previous bit width has been allocated, that value is allocated starting at the halfword boundary.

(4) If a hole results, it is filled with 0.

- The above two specifications can be made together for each .shword quasi directive.
- This quasi directive is suitable for creating a table for the switch instruction.

[Example]

Allocates an area for a string constant and stores a value in it

```
    .sdata
    .align 4
    .globl _t, 4
_t:
    .shword 10
```

.space

[Syntax]

```
.space size[, fill-value]
```

[Function]

Allocates an area of the size specified by the first operand and fills the allocated area with the fill value specified by the second operand (the default is 0).

- Specify a 1-byte fill value.
  If a larger value than this is specified, the 1 byte corresponding to the lowermost digit is used.

[Example]

Fills 4 bytes with 0

```
    .sidata
    .globl _p, 4
_p:
    .space 4
```

.str

[Syntax]

```
.str "string-constant"[, "string-constant"] ...
```

[Function]

Allocates an area for the specified string constant for each operand and stores the specified string in the allocated area[Note].

Note Unlike in the case of C, '\0' is not loaded as the default value at the end of a string.

[Example]

Allocates an area for a string constant and stores a value in it.

```
.str "hello"
```

.word

[Syntax]

```
.word value[, value] ...
.word bit-width:value[, bit-width:value] ...
```

[Function]

- The first part of this quasi directive instructs the allocation of a 1-word area for each operand, and the storing of the specified value in the allocated area.
- The second part of this quasi directive instructs the allocation of an area of a specified bit width, and the storing of the specified value in the allocated area.

(1) Specify the bit width as a value between 0 and 32.

(2) If the value exceeds the word width, it is masked by the word width.

(3) A value for which the bit width is declared first is allocated starting from the least significant bit position of the word area. If the word boundary of the area is exceeded as a result of allocating an area immediately after the area to which the value having a bit width has been allocated, the value having the bit width is allocated starting from the word boundary.

(4) If a hole results, it is filled with 0.

- The above two specifications can be made together for each .word quasi directive.

[Example]

Allocates an area of 1 word and fills it with 0xa**.**

```
    .sidata
    .align 4
    .globl _p, 4
_p:
    .word 0xa
```

4.6 Program Linkage Quasi Directives

Using the program linkage quasi directive, the as850 can declare an undefined external label[Note 1] or external label[Note 2] of a specified size, together with an alignment condition.

Next table lists the program linkage quasi directives described in this section.

Notes 1 Undefined external symbol (symbol having binding class GLOBAL and section header table index GPCOMMON or COMMON)

2 External symbol (symbol having binding class GLOBAL).

Table 4 - 7 Program Linkage Quasi Directives

Quasi Directive	Meaning
.comm	Declares an undefined external label
.extern	Declares an external label
.globl	Declares an external label

Maintain the values of the size (Number of bytes) and alignment condition, specified for a program linkage quasi directive, within 2^{31}. If a larger value than this is specified, the as850 outputs the following message then stops assembling.

```
E3247: illegal size value
    or
E3200: illegal alignment value
```

.comm

[Syntax]

```
.comm label-name, size, alignment-condition
```

[Function]

Declares an undefined external label[Note] having a label name specified by the first operand, a size specified by the second operand, and an alignment condition specified by the third operand.

Note Undefined external symbol (symbol having binding class GLOBAL and section header table index GPCOMMON or COMMON). If a definition for the undefined external symbol does not exist, the link editor (ld) of the CA850 allocates an area of the specified size, aligned under the specified alignment condition, to the .sbss section for an undefined external symbol having section header table index GPCOMMON, or to the .bss section for an undefined external symbol having section header table index COMMON. If two or more undefined external symbols of different sizes exist, the ld uses the larger size. If a definition already exists, it takes precedence.

- If the -G*num* option is specified upon starting the as850

(1) If the specified size is 1 or more, but no more than *num* bytes

Generates a symbol table entry having section header table index GPCOMMON upon generating the symbol table entry for the label when the object file is generated.

(2) If the specified size is 0 or more than *num* bytes

Generates a symbol table entry having section header table index COMMON upon generating the symbol table entry for the label when the object file is generated.

- If the -G*num* option is not specified upon starting the as850

Generates a symbol table entry having section header table index GPCOMMON upon generating the symbol table entry for the label when the object file is generated.

[Caution]

- If the same label name as that specified by the first operand is defined by means of normal label definition in the same file as this quasi directive

(a) If the label is declared as having symbol table entry index GPCOMMON and is defined by means of normal label definition in the data-attribute section, or if it is declared as having symbol table entry index COMMON by this quasi directive and is defined by means of normal label definition in the sdata-attribute section.

```
    .comm lab1, 4, 4    -- GPCOMMON if assembly is executed without -G
        :
    .data
lab1:                   -- Normal label definition in .sdata section
```

The as850 outputs the following message then stops assembling.

```
E3213: label identifier redefined
```

(b) Else

```
    .comm lab1, 4, 4    -- GPCOMMON if assembly is executed without -G
        :
    .sdata
lab1:                   -- Normal label definition in .sdata section
```

The label defined by means of normal label definition is regarded as being an external label and the specification of this quasi directive is ignored. Generates a symbol table entry having binding class GLOBAL upon generating the symbol table entry for the label when the object file is generated.

- If a label having the same name as that specified by the first operand is defined by the .lcomm quasi directive in the same file as this quasi directive

(a) If the size or alignment condition specified by the .lcomm quasi directive differs from the size or alignment condition specified by this quasi directive.

```
    .comm lab1, 4, 4
        :
    .sbss
    .lcomm lab1, 4, 2     -- Alignment condition differs
```

The as850 outputs the following message then stops assembling.

```
E3213: label identifier redefined
```

(b) If the label is declared, by this quasi directive, as having section header table index GPCOMMON and is defined in the bss-attribute section by the .lcomm quasi directive, or if it is declared by this quasi directive as having section header table index COMMON and is defined in the sbss-attribute section by the .lcomm quasi directive.

```
     .comm lab1, 4, 4     -- GPCOMMON if assembly is executed without -G
         :
     .bss
     .lcomm lab1, 4, 4    -- Definition in .bss section
```

The as850 outputs the following message then stops assembling.

```
E3213: label identifier redefined
```

(c) Else

```
     .comm lab1, 4, 4     -- GPCOMMON if assembly is executed without -G
         :
     .sbss
     .lcomm lab1, 4, 4    -- Definition in .bss section
```

The as850 regards the label defined by .lcomm as being an external label[Note], ignoring the specification made by this quasi directive. Generates a symbol table entry having binding class GLOBAL upon generating the symbol table entry for the label when the object file is generated.

- If a label having the same name as that specified by the first operand is (re-)defined by this quasi directive in the same file as this quasi directive.

(a) If the size or boundary condition is differen

```
     .comm lab1, 4, 4
         :
     .comm lab1, 2, 4     -- Size differs
```

The as850 outputs the following message then stops assembling.

```
E3213: label identifier redefined
```

(b) When the size and boundary conditions are the same

The as850 assumes the .comm quasi directive to be specified once only.

[Example]

Declares undefined external label of size 4 with alignment condition 4.

```
.sbss
.comm _p, 4, 4
```

.extern

[Syntax]

```
.extern label-name[, size]
```

[Function]

Declares a label having the same name as that specified by the first operand as an external label[Note]. If the second operand is specified, specifies a value as the size indicated by the data of the label.

This quasi directive is the same as the .globl quasi directive in that both declare an external label. However, use this quasi directive to declare a label that does not have a definition in the specified file as an external label, and use the .globl quasi directive to declare a label having a definition in the specified file as an external label.

Note External symbol (symbol having binding class GLOBAL).

[Caution]

- aWith the as850, by default, a label is declared as an external label if it does not have a definition in the specified file.

  Consequently, if a label having the same name as the label specified by the first operand does not have a definition in the specified file, this quasi directive specifies only the size of the data indicated by that label.
- Because the as850 judges whether to generate "a machine instruction that performs reference using 16-bit displacement" or "a machine instruction string (consisting of two or more machine instructions) that performs reference using 32-bit displacement" when executing gp offset reference to data that does not have a definition in the specified file, based on the size of the data, specify the size of the label that has no definition in the specified file and which is subject to gp offset reference, using this quasi directive.

[Example]

Declares external label _main (_main is not defined in file).

```
    .extern _main
```

.globl

[Syntax]

```
.globl label-name[, size]
```

[Function]

Declares a label having the same name as that specified by the first operand as an external label[Note]. If the second operand is specified, a value is specified as the size of the data indicated by the label.

This quasi directive is the same as the .extern quasi directive in that both declare an external label. However, use this quasi directive to declare a label having a definition in the specified file as an external label, and use the .extern quasi directive to declare a label that does not have a definition in the specified file as an external label.

Note External symbol (symbol having binding class GLOBAL).

[Caution]

- If a label having the same name as that of the label specified by the first operand is defined by this declaration, that label can be referenced from other assembler source files.
- When a guideline value for determining the size of the data to be allocated to the sdata/sbss-attribute section is to be displayed (by using the -A option of the ld850), the size of the data to be allocated to the sdata-attribute section (actually, the label subject to gp offset reference) must be specified by using either this or the .size quasi directive[Note].

Note Otherwise, valid information may not be obtained.

[Example]

Declares external label _func (_func is defined in file).

```
    .globl _func
```

4.7 Assembler Control Quasi Directive

The assembler control quasi directive can be used to control the processing performed by the as850.

Next table lists the assembler control quasi directive described in this section.

Table 4 - 8 Assembler Control Quasi Directive

Quasi Directive	Meaning
.option	Controls the assembler according to specified options

.option

[Syntax]

```
.option option
```

[Function]

Controls the assembler according to the options specified with the operand.

The following options can be specified[Note]:

Note Uppercase characters can also be used to specify the option (for example, NOMACRO can be specified instead of nomacro).

asm

This cancels c option specification for a syntax error that occurs after this quasi directive.

az_info_j

The address of the instruction immediately after this quasi directive is output to the address information section for AZ850 (The section name is az_info_j) . This option is specified to collect the address information for an instruction that calls a function.

az_info_r

The address of the instruction immediately after this quasi directive is output to the address information section for AZ850 (The section name is az_info_r) . This option is specified to collect the address information for an instruction which causes a return from a function.

az_info_ri

The address of the instruction immediately after this quasi directive is output to the address information section for AZ850 (The section name is az_info_ri) . This option is specified to collect the address information for an instruction which causes a return from an interrupt function.

C *linenum* ["*filename*"]

The line number of the error message and the file name for the syntax error subsequent to this quasi directive are overwritten by the specified items and output.

Second and subsequent "*filename*" specifications in the assembler source file can be omitted. If omitted, the file name is processed as the one specified for the preceding quasi directive. In this case, the presence of the asm option between this quasi directive and the preceding one is not checked.

If the first "*filename*" is omitted in the assembler source file, as850 outputs the following message then stops assembling.

```
E3249: illegal syntax
```

callt

A quasi directive which is reserved for the compiler

Caution Do not delete a callt instruction when it exists in the assembler source file output by the compiler. If it is deleted, the prologue epilogue runtime linking cannot be checked.

cpu devicename

Reads the device file on the target device specified by devicename.

To specify a device name to read the device file, the -cpu option can also be specified when starting the as850. A device name must always be specified when generating an object file. If a device name is not specified with the -cpu option, or with this quasi directive, the as850 outputs the following message then stops processing.

```
F3522: unknown cpu type
```

If a device name is specified by both the -cpu option and quasi directive, the as850 outputs a warning message. In this case, the specification made with the option takes precedence over that made with the quasi directive.

If two or more devices are specified by the option or quasi directive, the as850 outputs the following error message stops processing.

```
F3523: duplicated cpu type
```

Example

Specifies V850ES/SA2 as device to be used.

```
.option cpu 3201
```

To specify the device file to be used, specify the standard folder of the device file or the folder containing the device file with the -F option of the as850.

data *extern_symbol*

Assumes that external data having symbol name extern_symbol has been allocated to the data or bss attribute section, regardless of the size specified with the -G option of the ca850 or as850, and expands the instructions which reference that data.

This format is used when a variable for which "data" is specified in #pragma section or section file is externally referenced by an assembler source file.

Example

_d is used as the .data section regardless of the option and is expanded into instructions when referenced.

```
    .option data _d
    .text
    mov $_d, r11
```

ep_label

Performs a label reference by %label as a reference by ep offset for the subsequent instructions.

macro

Cancels the specification made with the nomacro option for the subsequent instructions.

mask_reg

Embeds information, which indicates the mask register function is used, in the relocatable object file generated by the as850.

This option is effective when, for example, an assembler source file output by an earlier C compiler that does not support the mask register function is used to specify the mask register function.

Since use of this option assumes that the mask register function is used, no error occurs when an object compiled with the mask register function specified is linked.

Caution When the mask register function is used, the C compiler uses r20 and r21 as mask registers. Do not allow the assembler source program to change the mask values set in these registers.

new_fcall

Embeds information, which indicates the new function call format[Note] is used, in the relocatable object file generated by the as850.

This option is effective when, for example, an assembler source file output by an earlier C compiler with different calling specifications is used with an object created by the current version of the C compiler.

Specifying this option assumes that the new call format is met, resulting in no error during a link with an object created in the default new call format of the C compiler.

no_ep_label

Cancels the specification made with the ep_label option for the subsequent instructions.

nomacro

Does not expand the subsequent instructions, other than the set*cond*/j*cond*/jmp/cmov*cond***[V850E]** / sasf*cond* **[V850E]** instructions.

nooptimize

Does not optimize instruction rearrangement for the subsequent instructions.

novolatile

Cancels the specification made with the nooptimize/volatile option for the subsequent instructions.

nowarning

Does not output warning messages for the subsequent instructions.

optimize

Has the same function as the novolatile option.

reg_mode *tnum pnum*

Embeds a register mode information section in the relocatable object file generated by the as850.

The register mode information section contains information relating to the number of work registers, and registers for register variables, used by the compiler. This instruction sets the number of work registers, and registers for register variables, as *tnum*, *pnum*.

When 22-register mode is used, *tnum* and *pnum* indicate five registers each. In 26-register mode, they indicate seven registers each.

Example

22-register mode is used.

```
    .option reg_mode 5 5
```

sdata *extern_symbol*

Assumes that external data having symbol name extern_symbol has been allocated to the sdata or sbss attribute section, regardless of the size specified with the -G option of the ca850 or as850, and does not expand the instructions which reference that data.

This format is used when a variable for which "sdata" is specified in the #pragma section or section file is externally referenced by an assembler source file.

Example

The _d is used as the .sdata section regardless of the option and is not expanded into instructions when referenced.

```
    .option sdata _d
    .text
    mov $_d, r11
```

volatile

Has the same function as the nooptimize option.

warning

Outputs warning messages for the subsequent instructions.

4.8 File Input Control Quasi Directives

Using the file input control quasi directive, the as850 can input an assembler source file or binary file to a specified position.

Next table lists the file input control quasi directives described in this section.

Table 4 - 9 File Input Control Quasi Directives

Quasi Directive	Meaning
.binclude	Inputs a binary file
.include	Inputs an assembler source file

.binclude

[Syntax]

```
.binclude "file-name"
```

[Function]

Assumes the contents of the binary file specified by the operand to be the result of assembling the source file at the position of this quasi directive.

The specified file is searched in the folder in which the source file including this quasi directive is placed. "file-name" can also be described with the relative path from the folder including the source file. When a folder is specified by the assembler option -I, the folder is searched first.

When there is no file in the folder in which the source file is placed, the folder in which C language source file is placed (specified by the .file quasi directive) and the current folder are searched.

[Caution]

- This quasi directive handles the entire contents of the binary files. When a relocatable file is specified, this quasi directive handles files configured in ELF format. Note that it is not just the contents of the .text selection, etc. that are handled.
- Enclose the file name to be specified with ".
- If a non-existent file is specified, the as850 outputs the following message then stops assembling.

```
F3503: can not open file file
```

[Example]

Includes aa.bin file.

```
    .binclude "aa.bin"
```

.include

[Syntax]

```
.include "file-name"
```

[Function]

ssumes that the contents of the file specified by the operand to be at the position of this quasi directive.

The specified file is searched in the folder in which the source file including this quasi directive is placed. "file-name" can also be described with the relative path from the folder including the source file. When a folder is specified by the assembler option -I, the folder is searched first.

When there is no file in the folder in which the source file is placed, the folder in which C language source file is placed (specified by the .file quasi directive and the current folder are searched.

[Caution]

- Enclose the file name to be specified with ".
- If a non-existent file is specified, the as850 outputs the following message then stops assembling.

```
F3503: can not open file file
```

- If the .include statement is nested 9 or more levels deep, the as850 outputs the following message then stops assembling.

```
F3517: include nest over
```

[Example]

Includes aa.s file.

```
    .include "aa.s"
```

4.9 Repetitive Assembly Quasi Directives

The as850 can repeatedly assemble an arrangement of statements (block) enclosed within a repetitive assembly quasi directive and corresponding .endm quasi directive, at the position of the repetitive assembly quasi directive.

Next table lists the repetitive assembly quasi directives described in this section.

Table 4 - 10 Repetitive Assembly Quasi Directives

Quasi Directive	Meaning
.irepeat	Repetition according to the parameter specification
.repeat	Repetition by the specified number of times

.irepeat

[Syntax]

```
.irepeat formal-parameter actual-parameter[, actual-parameter] ...
```

[Function]

Repeatedly assembles the arrangement of statements (block) enclosed within this quasi directive and the .endm quasi directive corresponding to this quasi directive, replacing the formal parameter specified by the first operand appearing in that block with the actual parameters specified by the second operands and those that follow. If the formal parameter is replaced by all the actual parameters specified by the second operand and those that follow, repetition is stopped.

[Caution]

- Always specify .irepeat and .endm as a pair. If .endm is omitted, the as850 outputs the following message then stops assembling.

```
F3513: unexpected EOF in .repeat/.irepeat
```

- If 33 or more actual parameters are specified, the as850 outputs the following message then stops assembling.

```
F3514: paramater table overflow
```

- If the same parameter name is specified for a formal parameter and an actual parameter, the as850 outputs the following message and stops assembling.

```
F3238: illegal operand (.irepeat parameter)
```

- If a parameter defined by a label or other quasi directive is specified for a formal parameter and an actual parameter, the as850 outputs the following message and stops assembling.

```
F3238: illegal operand (.irepeat parameter)
```

[Example]

```
.irepeat x a, b, c, d
    .word x
.endm
```

The expansion result of the above example is shown below:

```
    .word a
    .word b
    .word c
    .word d
```

.repeat

[Syntax]

```
.repeat absolute-value-expression
```

[Function]

Repeatedly assembles the arrangement of statements (block) enclosed within this quasi directive and the corresponding .endm quasi directive by the number of times specified by the absolute expression of the first operand.

[Caution]

- Always specify .repeat and .endm as a pair. If .endm is omitted, the as850 outputs the following message then stops assembling.

```
F3513: unexpected EOF in .repeat/.irepeat
```

- The value is evaluated as a 32-bit signed integer.
- If there is no arrangement of statements (block), nothing is executed.
- If the result of evaluating the expression is negative, the as850 outputs the following message, and continues assembling.

```
E3225: illegal operand (must be evaluated positive or zero)
```

[Example]

```
.repeat 2
    nop
.endm
```

The expansion result of the above example is shown below:

```
    nop
    nop
```

4.10 Conditional Assembly Quasi Directives

Using conditional assembly quasi directives, the as850 can control the range of assembly according to the result of evaluating a conditional expression.

Next table lists the conditional assembly quasi directives described in this section.

Table 4 - 11 Conditional Assembly Quasi Directives

Quasi Directive	Meaning
.else	Control based on absolute expression/symbol
.elseif	Control based on absolute expression (assembly performed when the value is true)
.elseifn	Control based on absolute expression (assembly performed when the value is false)
.endif	End of control range
.if	Control based on absolute expression (assembly performed when the value is true)
.ifdef	Control based on symbol (assembly performed when the symbol is defined)
.ifn	Control based on absolute expression assembly performed when the value is false)
.ifndef	Control based on symbol (assembly performed when the symbol is not defined)

If a conditional assembly quasi directive is nested 17 or more levels deep, the as850 outputs the following message then stops assemblin.

```
F3512: .if, .ifn, etc. too deeply nested
```

.else

[Syntax]

```
.else
```

[Function]

If the absolute expression of the .if, .elseif, or .ifdef quasi directive is evaluated as being false (= 0), or if the absolute expression of the .ifn, .elseifn, or .ifndef quasi directive corresponding to this quasi directive is evaluated as being true (≠0), assembles the arrangement of statements (block) enclosed within this quasi directive and the corresponding .endif quasi directive.

[Caution]

- If the .if, .ifn, .elseif, .elseifn, .ifdef, or .ifndef quasi directive corresponding to this quasi directive does not exist, the as850 outputs the following message then stops assembling.

```
F3510: .else unexpected
```

[Example]

```
.if 0
    .word 10
.else
    .str "a"
.endif

.if 10 > 20
    .word 20
.else
    .str "b"
.endif

.set expr, 0
.if expr
    .word expr
.else
    .str "c"
.endif
```

The expansion result of the above example is shown below:

```
    .str "a"
    .str "b"
    .str "c"
```

.elseif

[Syntax]

```
.elseif absolute-value-expression
```

[Function]

- If the absolute expression specified by the operand is evaluated as being true (≠0)

(1) If this quasi directive and the corresponding .else, .elseif, or .elseifn quasi directive exist, assembles the block enclosed within this quasi directive and the corresponding quasi directive.

(2) If none of the corresponding quasi directives detailed above exist, assembles the block enclosed within this quasi directive and the corresponding .endif quasi directive.

- If the absolute expression is evaluated as being false (= 0)

  Skips to the .else, .elseif, .elseifn, or .endif quasi directive corresponding to this quasi directive.

[Caution]

- If a corresponding quasi directive does not exist, the as850 outputs the following message then stops assembling.

```
F3511: .endif unmatched
```

[Example]

```
.if 0
    .word 10
.elseif 10
    .str "a"
.endif
.if 10 > 20
    .word 20
.elseif 10 == 20
    .str "b"
.endif
.set expr, 0
.if expr
    .word expr
.elseifn expr - 10
    .str "c"
.endif
```

The expansion result of the above example is shown below:

```
    .str "a"
```

.elseifn

[Syntax]

```
.elseifn absolute-value-expression
```

[Function]

- If the absolute expression specified by the operand is evaluated as being true (≠0)

  Skips to the .else, .elseif, .elseifn, or .endif quasi directive corresponding to this quasi directive.

- If the absolute expression is evaluated as being false (= 0)

(1) If this quasi directive and the corresponding .else, .elseif, or .elseifn quasi directive exist, assembles the block enclosed within this quasi directive and the corresponding quasi directive.

(2) If none of the corresponding quasi directives detailed above exist, assembles the block enclosed within this quasi directive and the corresponding .endif quasi directive.

[Caution]

- If the corresponding quasi directive does not exist, the as850 outputs the following message then stops assembling.

```
F3511: .endif unmatched
```

[Example]

```
.if 0
    .word 10
.elseifn 10
    .str "a"
.endif
.if 10 > 20
    .word 20
.elseifn 10 >= 20
    .str "b"
.endif
.set expr, 0
.if expr
    .word expr
.elseif expr - 10
    .str "c"
.endif
```

The expansion result of the above example is shown below:

```
    .str "b"
    .str "c"
```

.endif

[Syntax]

```
.endif
```

[Function]

Indicates the end of the control range of a conditional assembly quasi directive.

[Caution]

- If the .if, .ifn, .elseif, .elseifn, .ifdef, or .ifndef quasi directive corresponding to this quasi directive does not exist, the as850 outputs the following message then stops assembling.

```
F3510: .endif unexpected
```

.if

[Syntax]

```
.if absolute-value-expression
```

[Function]

- If the absolute expression specified by the operand is evaluated as being true (≠0)

(1) If this quasi directive and a corresponding .else, .elseif, or .elseifn quasi directive exist, assembles the block enclosed within this quasi directive and the corresponding quasi directive.

(2) If none of the corresponding quasi directives detailed above exist, assembles the block enclosed within this quasi directive and the corresponding .endif quasi directive.

- If the absolute expression is evaluated as being false (= 0)

  Skips to the .else, .elseif, .elseifn, or .endif quasi directive corresponding to this quasi directive.

[Caution]

- If an undefined symbol is specified by the operand, the as850 outputs the following message then stops assembling.

```
E3202: illegal expression
```

- If a corresponding quasi directive does not exist, the as850 outputs the following message then stops assembling.

```
F3511: .endif unmatched
```

[Example]

```
.if 10
    .word 10
.endif
.if 10 < 20
    .word 20
.endif
.set expr, 30
.if expr
    .word expr
.endif
```

The expansion result of the above example is shown below:

```
    .word 10
    .word 20
    .word 30
```

.ifdef

[Syntax]

```
.ifdef name
```

[Function]

- If the name specified by the operand is defined

(1) If this quasi directive and the corresponding .else, .elseif, or .elseifn quasi directive exist, assembles the block enclosed within this quasi directive and the corresponding quasi directive.

(2) If none of the corresponding quasi directives detailed above exist, assembles the block enclosed within this quasi directive and the corresponding .endif quasi directive.

- If the specified name is not defined

  Skips to the .else, .elseif, .elseifn, or .endif quasi directive corresponding to this quasi directive.

[Caution]

- A symbol, label, or macro name can be specified as the name, but a reserved word must not be specified. If a reserved word is specified, the as850 outputs the following message then stops assembling.

```
E3220: illegal operand ( identifier is reserved word)
```

- If the corresponding quasi directive does not exist, the as850 outputs the following message then stops assembling.

```
F3511: .endif unmatched
```

[Example]

```
define_symbol:
    .ifdef define_symbol
        .word 10
    .endif
    .ifdef undef_symbol
        .word 20
    .else
        .ifde define_symbol
            .str "x"
        .endif
    .endif
    .set expr, 20
    .ifdef expr
        .word expr
    .endif
```

The expansion result of the above example is shown below:

```
    .word 10
    .str "x"
    .word 20
```

.ifn

[Syntax]

.ifn `absolute-value-expression`

[Function]

- If the absolute expression specified by the operand is evaluated as being true (≠0)

  Skips to the .else, .elseif, .elseifn, or .endif quasi directive corresponding to this quasi directive.

- If the absolute expression is evaluated as being false (= 0)

(1) If this quasi directive and the corresponding .else, .elseif, or .elseifn quasi directive exist, assembles the block enclosed within this quasi directive and the corresponding quasi directive.

(2) If none of the corresponding quasi directives detailed above exist, assembles the block enclosed within this quasi directive and the corresponding .endif quasi directive.

[Caution]

- If the corresponding quasi directive does not exist, the as850 outputs the following message then stops assembling.

```
F3511: .endif unmatched
```

[Example]

```
.ifn 0
    .word 10
.endif
.ifn 10 > 20
    .word 20
.endif
.set expr, 0
.ifn expr
    .word expr
.endif
```

The expansion result of the above example is shown below:

```
    .word 10
    .word 20
    .word 0
```

.ifndef

[Syntax]

```
.ifndef name
```

[Function]

- If the name specified by the operand is defined

  Skips to the .else, .elseif, .elseifn, or .endif quasi directive corresponding to this quasi directive.

- If the specified name is not defined

(1) If this quasi directive and the corresponding .else, .elseif, or .elseifn quasi directive exist, assembles the block enclosed within this quasi directive and the corresponding quasi directive.

(2) If none of the corresponding quasi directives detailed above exist, assembles the block enclosed within this quasi directive and the corresponding .endif quasi directive.

[Caution]

- A symbol, label, or macro name can be specified as the name, but a reserved word must not be specified. If a reserved word is specified, the as850 outputs the following message then stops assembling.

```
E3220: illegal operand ( identifier is reserved word)
```

- If the corresponding quasi directive does not exist, the as850 outputs the following message then stops assembling.

```
F3511: .endif unmatched
```

[Example]

```
define_symbol:
    .ifndef define_symbol
        .word 10
    .else
        .str "a"
    .endif
    .ifndef undef_symbol
        .word 20
    .else
        .ifndef define_symbol
            .str "x"
        .endif
    .endif
    .set expr, 20
    .ifndef expr
        .word expr
    .endif
```

The expansion result of the above example is shown below:

```
    .str "a"
    .word 20
```

4.11 Skip Quasi Directives

Using the skip quasi directives, the as850 can skip the remaining repetitions of a repetitive assembly quasi directive.

Next table lists the skip quasi directives described in this section.

Table 4 - 12 Skip Quasi Directives

Quasi Directive	Meaning
.exitm	Skips outwards by one
.exitma	Skips to the outmost repetition

.exitm

[Syntax]

```
.exitm
```

[Function]

This quasi directive skips the repetitive assembly of the repetitive assembly quasi directives enclosing this quasi directive at the innermost position.

[Caution]

- If this quasi directive is not enclosed by repetitive assembly quasi directives, the as850 outputs the following message then stops assembling.

```
F3515: .exitm not in .repeat/.irepeat
```

[Example]

```
.repeat 2
    .set expr, 1
    .word 10
    .repeat 10
        .if expr < 5
            .byte expr
            .set expr, expr + 1
        .else
            .ifdef undefine_symbol
                .byte expr
                .set expr, expr + 1
            .else
                .exitm
            .endif
        .endif
    .endm
    .hword 20
    .hword 30
.endm
.word expr
```

The expansion result of the above example is shown below:

```
    .word 10
    .byte 1
    .byte 2
    .byte 3
    .byte 4
    .hword 20
    .hword 30
    .word 10
    .byte 1
    .byte 2
    .byte 3
    .byte 4
    .hword 20
    .hword 30
    .word 5
```

.exitma

[Syntax]

```
.exitma
```

[Function]

This quasi directive skips the repetitive assembly of the repetitive assembly quasi directives enclosing this quasi directive at the outermost position.

[Caution]

- If this quasi directive is not enclosed by repetitive assembly quasi directives, the as850 outputs the following message then stops assembling.

```
F3515: .exitma not in .repeat/.irepeat
```

[Example]

```
.repeat 2
    .set expr, 1
    .word 10
    .repeat 10
        .if expr < 5
            .byte expr
            .set expr, expr + 1
        .else
            .ifdef undefine_symbol
                .byte expr
                .set expr, expr + 1
            .else
                .exitma
            .endif
        .endif
    .endm
    .hword 20
    .hword 30
.endm
.word expr
```

The expansion result of the above example is shown below:

```
    .word 10
    .byte 1
    .byte 2
    .byte 3
    .byte 4
    .word 5
```

4.12 Macro Quasi Directives

Using a macro quasi directive, the as850 can define any arrangement of statements as a macro body corresponding to a specified macro name. By referencing this macro name in the source program, it can be assumed that the arrangement of statements corresponding to the macro name is described at the position of reference.

Next table lists the macro quasi directives described in this section.

Table 4 - 13 Macro Quasi Directives

Quasi Directive	Meaning
.endm	End of repetitive zone or end of macro definition
.local	Definition of local symbol
.macro	Beginning of macro definition

.endm

[Syntax]

```
.endm
```

[Function]

Indicates the end of a repetitive zone or a macro body.

[Caution]

- If the .repeat, .irepeat, or .macro quasi directive corresponding to this quasi directive does not exist, the as850 outputs the following message then stops assembling.

```
F3510: .endm unexpected
```

.local

[Syntax]

```
.local local-symbol[, local-symbol] ...
```

[Function]

Declares a specified string as a local symbol that is replaced by a specific identifier.

[Caution]

- If 33 or more local symbols are specified for the formal parameter of this quasi directive, the as850 outputs the following message then stops assembling.

```
F3514: paramater table overflow
```

-The local symbol name is generated by the assembler in the range between .??0000 and ??FFFF.

[Example]

```
.macro m1 x
    .local a, b
    a: .word a
    b: .word x
.endm
m1 10
m1 20
```

The expansion result of the above example is shown below:

```
.??0000: .word .??0000
.??0001: .word 10
.??0002: .word .??0002
.??0003: .word 20
```

.macro

[Syntax]

```
.macro macro-name [formal-parameter,] ...
```

[Function]

Defines the arrangement of the statements, enclosed within this quasi directive and the .endm quasi directive, as the macro body for the macro name specified by the first operand. If this macro name is referenced (a process referred to as "macro call"), it is assumed that the macro body corresponding to the macro name is described at the position of the macro call .

[Caution]

- If the .endm quasi directive corresponding to this quasi directive does not exist, the as850 outputs the following message then stops assembling.

```
F3513: unexpected EOF in .macro
```

- If a macro name is re-defined, and if this macro is subsequently called, the re-defined macro body becomes the macro body of the macro name.
- If 33 or more formal parameters are specified, the as850 outputs the following message then stops assembling.

```
F3514: paramater table overflow
```

- Any excess formal parameters that are not referenced in the macro body are ignored. Note that, in this case, the as850 outputs no message.
- If a shortage of actual parameters for macro call occurs, the as850 outputs the following message then stops assembling.

```
F3519: argument mismatch
```

- If an undefined macro is called in a macro body, the as850 outputs the following message then stops assembling.

```
E3249: illegal syntax
```

- If a currently defined macro is called in a macro body, the as850 outputs the following message then stops assembling.

```
F3518: unreasonable macro_call nesting
```

- If a parameter defined by a label or quasi directive is specified for a formal parameter, the as850 outputs the following message and stops assembling.

```
E3212: symbol already defined as string
```

- When calling a macro, only a label name, symbol name, numeric value, register, and instruction mnemonic can be specified for an actual parameter. If a label expression (LABEL-1), reference method specification label (#LABEL), or base register specification ([gp]) is specified, the as850 outputs a message dependent on the specified actual parameter and stops assembling.

[Example]

```
.macro PUSH REG
    add     -4, sp
    st.w    REG, 0x0[sp]
.endm
.macro POP REG
    ld.w    0x0[sp],REG
    add     0x4, sp
.endm
    PUSH    r10
    mov     10, r10
    add     r10, r20
    POP     r10
```

The expansion result of the above example is shown below:

```
    add     -4, sp
    st.w    r10, 0x0[sp]
    mov     10, r10
    add     r10, r20
    ld.w    0x0[sp], r10
    add     0x4, sp
```

APPENDIX A INSTRUCTION SUMMARY

In the next table, this appendix lists the instruction mnemonics and quasi directives supported by the CA850 assembler (as850), in alphabetical order.

Table A - 1 Instruction Mnemonics List

Instruction Mnemonics	Meaning
add	Addition
addi	Addition (immediate)
adf	Add with condition flag **[V850E2]**
and	Logical product
andi	Logical product (immediate)
bsh	Byte swap halfword **[V850E]**
bsw	Byte swap word **[V850E]**
callt	Table reference call **[V850E]**
clr1	Bit clear
cmov	Transfers data depending on the flag condition **[V850E]**
cmp	Comparison
ctret	Returns from callt **[V850E]**
dbret	Returns from debug trap **[V850E]**
dbtrap	Debug trap **[V850E]**
di	Disables maskable interrupt
dispose	Postprocessing of function (dispose) **[V850E]**
div	Signed division (word) **[V850E]**
divh	Signed division (halfword)
divhu	Unsigned division (halfword) **[V850E]**
divu	Unsigned division (word) **[V850E]**
ei	Enables maskable interrupt
halt	Stops the processor
hsh	Half-word data half-word swap **[V850E2]**
hsw	Halfword swap word **[V850E]**
jarl	Jump and register link
jarl22	Jump and register link **[V850E2]**
jarl32	Jump and register link **[V850E2]**

Table A - 1 Instruction Mnemonics List

Instruction Mnemonics	Meaning
jcond	Conditional branch
jmp	Unconditional branch
jmp32	Unconditional branch (jump) **[V850E2]**
jr	Unconditional branch (PC relative)
jr22	Unconditional branch (PC relative) **[V850E2]**
jr32	Unconditional branch (PC relative) **[V850E2]**
ld.b	Load (byte)
ld.bu	Load (unsigned byte) **[V850E]**
ld.h	Load (halfword)
ld.hu	Load (unsigned halfword) **[V850E]**
ld.w	Load (word)
ldsr	Loads to system register
mac	Signed word data multiply and add **[V850E2]**
macu	Unsigned word data multiply and add **[V850E2]**
mov	Moves data
mov32	Moves data (32-bit) **[V850E]**
movea	Addition (32-bit immediate)
movhi	Addition (16-bit immediate)
mul	Signed multiplication (word) **[V850E]**
mulh	Signed multiplication (halfword)
mulhi	Signed multiplication (immediate)
mulu	Unsigned multiplication (word) **[V850E]**
nop	No operation
not	Logical negation (takes 1's complement)
not1	Bit negation
or	Logical sum
ori	Logical sum (immediate)
pop	Pop from stack area (single register)
popm	Pop from stack area (multiple registers)
prepare	Preprocessing of function (prepare) **[V850E]**
push	Push to stack area (single register)
pushm	Push to stack area (multiple registers)

Table A - 1 Instruction Mnemonics List

Instruction Mnemonics	Meaning
reti	Returns from trap or interrupt routine
sar	Arithmetic right shift
sasf	Set the flag condition after a logical left shift **[V850E]**
satadd	Saturated addition
satsub	Saturated subtraction
satsubi	Saturated subtraction (immediate)
satsubr	Reverse subtraction with saturation
sch0l	Bit (0) search from MSB side **[V850E2]**
sch0r	Bit (0) search from MSB side **[V850E2]**
sch1l	Bit (1) search from MSB side **[V850E2]**
sch1r	Bit (1) search from MSB side **[V850E2]**
sbf	Subtract with condition flag **[V850E2]**
set1	Bit set
setf	Sets flag condition
shl	Logical left shift
shr	Logical right shift
sld.b	Byte data load (short format)
sld.bu	Unsinged byte data load (short format) **[V850E]**
sld.h	Halfword data load (short format)
sld.hu	Unsinged halfword data load (short format) **[V850E]**
sld.w	Word data load (short format)
sst.b	Byte data store (short format)
sst.h	Halfword data store (short format)
sst.w	Word data store (short format)
st.b	Byte data store
st.h	Halfword data store
st.w	Word data store
stsr	Stores contents of system register
sub	Subtraction
subr	Reverse subtraction
switch	Table reference jump **[V850E]**
sxb	Sign extension byte **[V850E]**

Table A - 1 Instruction Mnemonics List

Instruction Mnemonics	Meaning
sxh	Sign extension halfword **[V850E]**
trap	Software trap
tst	Test
tst1	Bit test
xor	Exclusive OR
xori	Exclusive OR (immediate)
zxb	Zero extension byte **[V850E]**
zxh	Zero extension halfword **[V850E]**

Table A - 2 Quasi Directives List

Quasi Directive	Meaning
.align	Aligns the value of the location counter
.binclude	Inputs a binary file
.bss	Allocation to .bss section
.byte	Allocates a 1-byte area
.comm	Declares an undefined external label
.const	Allocation to .const section
.data	Allocation to .data section
.else	Control based on absolute expression/symbol
.elseif	Control based on absolute expression (assembly performed when the value is true)
.elseifn	Control based on absolute expression (assembly performed when the value is false)
.endif	End of control range
.endm	End of repetitive zone or end of macro definition
.exitm	Skips outwards by one
.exitma	Skips to the outmost repetition
.extern	Declares an external label
.ext_ent_size	Flash table entry size
.ext_func	Generates a flash table entry
.file	Generates a symbol table entry (FILE type)
.float	Sets a floating-point value
.frame	Generates a symbol table entry (FUNC type)
.globl	Declares an external label
.hword	Allocates a 1-halfword area
.if	Control based on absolute expression (assembly performed when the value is true)
.ifdef	Control based on symbol (assembly performed when the symbol is defined)
.ifn	Control based on absolute expression (assembly performed when the value is false)
.ifndef	Control based on symbol (assembly performed when the symbol is not defined)
.include	Inputs an assembler source file
.irepeat	Repetition according to the parameter specification
.lcomm	Defines a label that allocates an area
.local	Definition of local symbol
.macro	Beginning of macro definition

Table A - 2 Quasi Directives List

Quasi Directive	Meaning
.option	Controls the assembler according to specified options
.org	Advances the value of the location counter
.previous	(Re-)definition of section definition quasi directive preceding the section definition quasi directive that specifies the current section definition quasi directive
.repeat	Repetition by the specified number of times
.sbss	Allocation to .sbss section
.sconst	Allocation to .sconst section
.sdata	Allocation to .sdata section
.sebss	Allocation to .sebss section
.section	Allocation to section of specified type
.sedata	Allocation to .sedata section
.set	Defines a symbol
.shword	Allocate a 1 halfword area (for switch instruction) **[V850E]**
.sibss	Allocation to .sibss section
.sidata	Allocation to .sidata section
.size	Specifies the size of the data indicated by label
.space	Allocates an area for size
.str	Allocates an area for string
.text	Allocation to .text section
.tibss	Allocation to .tibss section
.tibss.byte	Allocation to .tibss.byte section
.tibss.word	Allocation to .tibss.word section
.tidata	Allocation to .tidata section
.tidata.byte	Allocation to .tidata.byte section
.tidata.word	Allocation to .tidata.word section
.vdbstrtab	Allocation to .vdbstrtab section
.vdebug	Allocation to .vdebug section
.vline	Allocation to .vline section
.word	Allocates a 1-word area

APPENDIX B INDEX

For further information,
please contact:

NEC Electronics Corporation
1753, Shimonumabe, Nakahara-ku,
Kawasaki, Kanagawa 211-8668,
Japan
Tel: 044-435-5111
http://www.necel.com/

[America]

NEC Electronics America, Inc.
2880 Scott Blvd.
Santa Clara, CA 95050-2554, U.S.A.
Tel: 408-588-6000
800-366-9782
http://www.am.necel.com/

[Europe]

NEC Electronics (Europe) GmbH
Arcadiastrasse 10
40472 Düsseldorf, Germany
Tel: 0211-65030
http://www.eu.necel.com/

Hanover Office
Podbielskistrasse 166 B
30177 Hannover
Tel: 0 511 33 40 2-0

Munich Office
Werner-Eckert-Strasse 9
81829 München
Tel: 0 89 92 10 03-0

Stuttgart Office
Industriestrasse 3
70565 Stuttgart
Tel: 0 711 99 01 0-0

United Kingdom Branch
Cygnus House, Sunrise Parkway
Linford Wood, Milton Keynes
MK14 6NP, U.K.
Tel: 01908-691-133

Succursale Française
9, rue Paul Dautier, B.P. 52
78142 Velizy-Villacoublay Cédex
France
Tel: 01-3067-5800

Sucursal en España
Juan Esplandiu, 15
28007 Madrid, Spain
Tel: 091-504-2787

Tyskland Filial
Täby Centrum
Entrance S (7th floor)
18322 Täby, Sweden
Tel: 08 638 72 00

Filiale Italiana
Via Fabio Filzi, 25/A
20124 Milano, Italy
Tel: 02-667541

Branch The Netherlands
Steijgerweg 6
5616 HS Eindhoven
The Netherlands
Tel: 040 265 40 10

[Asia & Oceania]

NEC Electronics (China) Co., Ltd
7th Floor, Quantum Plaza, No. 27 ZhiChunLu Haidian
District, Beijing 100083, P.R.China
Tel: 010-8235-1155
http://www.cn.necel.com/

NEC Electronics Shanghai Ltd.
Room 2511-2512, Bank of China Tower,
200 Yincheng Road Central,
Pudong New Area, Shanghai P.R. China P.C:200120
Tel: 021-5888-5400
http://www.cn.necel.com/

NEC Electronics Hong Kong Ltd.
Unit 1601-1613, 16/F., Tower 2, Grand Century Place,
193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: 2886-9318
http://www.hk.necel.com/

NEC Electronics Taiwan Ltd.
7F, No. 363 Fu Shing North Road
Taipei, Taiwan, R. O. C.
Tel: 02-8175-9600
http://www.tw.necel.com/

NEC Electronics Singapore Pte. Ltd.
238A Thomson Road,
#12-08 Novena Square,
Singapore 307684
Tel: 6253-8311
http://www.sg.necel.com/

NEC Electronics Korea Ltd.
11F., Samik Lavied'or Bldg., 720-2,
Yeoksam-Dong, Kangnam-Ku,
Seoul, 135-080, Korea
Tel: 02-558-3737
http://www.kr.necel.com/