EPSON

CMOS 32-bit Single Chip Microcomputer

E0C33 Family C Compiler Package

Quick Reference for Development

Memory Map and Trap Table

E0C33000 Core CPU

Memory Map

			Area size
0xFFFFFFF	Area 18	External memory	64MB
	Area 17	External memory	64MB
	Area 16	External memory	32MB
	Area 15	External memory	32MB
	Area 14	External memory	16MB
	Area 13	External memory	16MB
	Area 12	External memory	8MB
0x1000000	Area 11	External memory	8MB
0x0C00000	Area 10	External memory	4MB
	Area 9	External memory	4MB
	Area 8	External memory	2MB
	Area 7	External memory	2MB
	Area 6	External I/O	1MB
	Area 5	External memory	1MB
	Area 4	External memory	1MB
0x0100000	Area 3	On-chip ROM	512KB
0x0080000	Area 2	Reserved	128KB
0x0060000	Area 1	Internal I/O	128KB
0x0040000	Area 0	On-chip RAM	256KB
0x0000000			

Trap Table

	Vector address
Reset	base + 0
Reserved	base + 4–12
Zero division	base + 16
Reserved	base + 20
Address error	base + 24
NMI	base + 28
Reserved	base + 32–44
Software exception 0	base + 48
:	:
Software exception 3	base + 60
External maskable interrupt 0	base + 64
:	:
External maskable interrupt 215	base + 924

base: Trap table start address
= 0x0080000 (when booting by on-chip ROM)
= 0x0C00000 (when booting by external ROM)

Registers

E0C33000 Core CPU

General-purpose registers (16)

31 ... 0
R15
R14
R13
:
R4
R3
R2
R1
R0

Special registers (5)

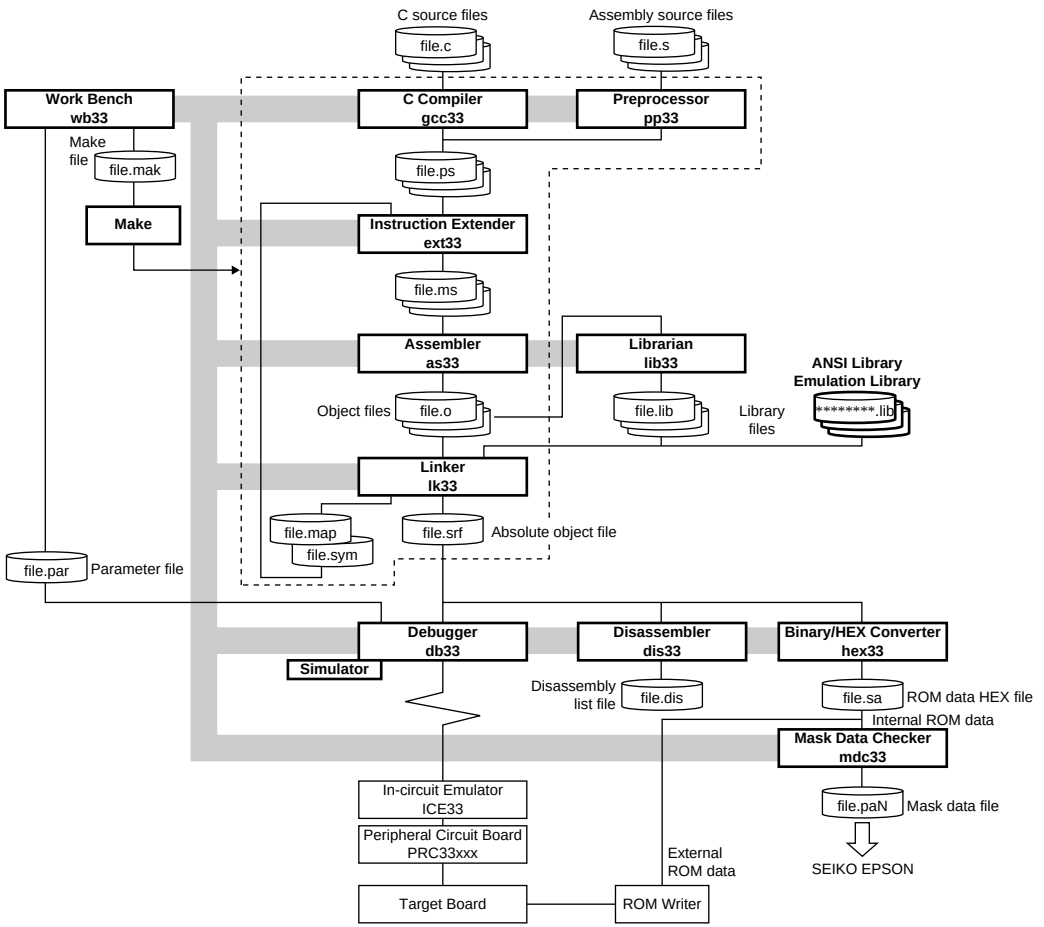
31 ... 0	
PC	Program counter
PSR	Processor status register
SP	Stack pointer
ALR	Arithmetic operation low register
AHR	Arithmetic operation high register

(AHR, ALR: Option for Multiplication & Accumulation, Multiplication, and Division)

PSR

31–12	11–8	7	6	5	4	3	2	1	0
Reserved	IL	MO	DS	–	IE	C	V	Z	N

IL: Interrupt level (0–15: Enabled interrupt level)
MO: MAC overflow flag (1: MAC overflow, 0: Not overflown)
DS: Dividend sign flag (1: Negative, 0: Positive)
IE: Interrupt enable (1: Enabled, 0: Disabled)
Z: Zero flag (1: Zero, 0: Non zero)
N: Negative flag (1: Negative, 0: Positive)
C: Carry flag (1: Carry/borrow, 0: No carry)
V: Overflow flag (1: Overflow, 0: Not overflown)



1. Programming

Create C source and assembly source files using an editor.

2. Compilation, Assembly and Link

2-1) Start up the Work Bench wb33.
2-2) Create a Make file using the [Make gen] button of the wb33, then customize the contents if necessary.
2-3) Execute the Make tool using the [MAKE] button of the wb33.
The Make tool executes the C Compiler gcc33, Preprocessor pp33, Instruction Extender ext33, Assembler as33 and Linker lk33 sequentially to generate an executable object file (.srf). Each software tool can also be executed individually.

3. Debugging

3-1) Create a Parameter file using the [Par gen] button of the wb33, then customize the contents if necessary.
3-2) Start up the Debugger db33 using the [DB33] button of the wb33.
3-3) Load the executable object file (.srf), then debugging it using the db33 debug commands.

4. Mask Data Creation

When the program development has been completed, create the Mask data file.
4-1) Create the ROM data HEX file using the Binary/HEX Converter hex33.
4-2) Convert the ROM data HEX file into a Mask data file using the Mask Data Checker mdc33.
4-3) Submit the Mask data file to Seiko Epson.

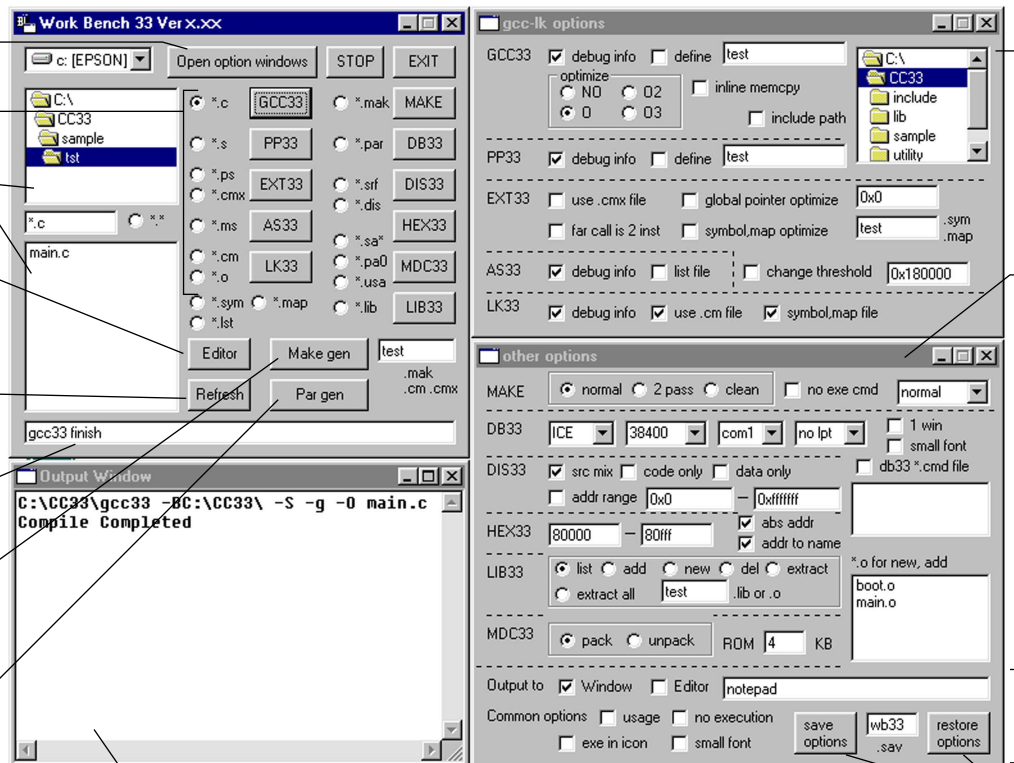
Outline

This wb33 enables the development tools to be started up from the wb33 window. The selection of files, major start-up options, and the startup of each tool can be executed by mouse operations alone.

Error Messages

Can not execute command	The command (tool) cannot be executed.
Can not open file	The source, make or option file cannot be opened.
Write error	Data cannot be written to the make or option file.
Read error	The source or option file cannot be read.
File size over 32KB, so cut down to 32KB	The source-file capacity exceeded the 32KB display range.

[Execution] window



[Open option window] button

Opens the two option windows.

Tool buttons

Clicking the button executes the tool.

Directory/file list boxes

Select the input file before starting up the tool.

[Editor] button

Opens the selected source file with the editor specified in the [other options] window.

[Refresh] button

Updates the file list displayed in the box.

Command line edit box

Displays start-up commands. They can also be modified in this box.

[Make gen] button

To create a make file, enter the file name in the box on the right, select the source files in the list box, then click this button.

[Par gen] button

Opens the [parameter file generator] window for making a parameter file.

[Stop] button

This button can stop tool execution. It does not affect the make and single file processing.

[Exit] button

Terminates the wb33.

[Output] window

Displays the contents of source files and results of tool execution. To display a source file, double click the file name listed in the file list box on the [Execution] window.

[gcc-lk options] window

The start-up options for the gcc33, pp33, ext33, as33 and lk33 can be selected.

[other options] window

The start-up options for the make, db33, dis33, hex33, lib33 and mdc33 can be selected. The wb33 options can also be selected.

wb33 options

[Window]: Displays the tool execution results to the [Output] window.

[Editor]: Displays the tool execution results with the editor.

Edit box: Enter the start-up command of the editor to be used.

[usage]: Usage output

[no execution]: Displays the tool start -up command in the command line edit box. Tools are not executed.

[exe in icon]: Executes tools in minimized status.

[small font]: Changes the font used in the [Output] window.

[save/restore options] buttons

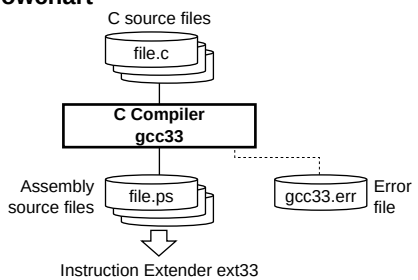
Save and restores the option settings on the option windows.

Outline

The gcc33 is an ANSI standard C compiler (GNU C Compiler) and compiles C source files and generates the assembly source files.

∗ The GNU C Compiler is designed by Free Software Foundation, Inc.

Flowchart



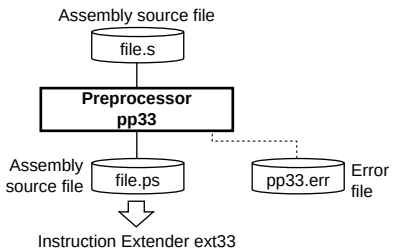
Start-up Command Usage

```
GNU C Compiler for E0C33 Ver 2.7.2 (Rev x.xx)
Usage:
    gcc33 -S [options] filename
Options:
    -g : generate debug information
    -O : optimize output code
    -E : preprocess source files and output the results to stdout
    -B<directory>\ : specifies the directoy which cc1.exe and cpp.exe exit
    -I<directory>  : specifies the directoy which include files exist
    -D<macro=defn> : define macro "macro" as "defn"
    -D<macro>      : define macro "macro" as '1'
    -merr          : produce log file (gcc33.err)
    -mno-memcpy    : produce string variable initializer statement inline
Output:
    Extended assembler source files(.ps) for ext33
Example:
    gcc33 -S -Bc:\usr\local\bin\ -O -g test.c
```

Outline

Starts the processing procedure of assembly source files. The pp33 expands the range of program-creating functions, such as for Macro statements and Include statements, and thus creates assembly source files to be entered into Instruction Extender ext33.

Flowchart



Start-up Command Usage

```
Pre Processor 33 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
Usage:
    pp33 [options] filename
Options:
    -e : produce log file (pp33.err)
    -g : generate debug information in output file
    -d string : define string
Output:
    Assembler source file for ext33 (.ps)
Example:
    pp33 -e -g -d TYPE1 test.s
```

Error Messages

Cannot open file.	The source file or included file cannot be opened.
Cannot allocate memory.	There is not enough memory.
Cannot open output file.	The output file cannot be opened.
Cannot open working file.	The temporary working file cannot be opened.
Cannot write file.	Data cannot be written to the file.
Cannot close file.	The file cannot be closed.
Cannot read file. Line size is too long.	The statement is too long to read. (Max. 255 characters)
Preprocessor limit : macro label number full.	The number of internal branch labels has exceeded the limit in macro expansion. (Max. 9999 labels)
Invalid syntax.	The preprocessor instruction description format is illegal.
Nesting level too deep.	Nesting of #include has exceeded the limit. (Max. 2 levels)
Unknown preprocessor instruction.	The preprocessor instruction is illegal.
Too many macro parameters.	The macro parameters have exceeded the limit. (Max. 32)
Invalid macro parameter.	The macro parameters are illegal. ($1–$32)
Invalid macro label.	The internal branch label in macro definition is abnormal. ($$1–$$64)
Invalid expression.	The operator description format is illegal.
Multi symbol.	The same name was defined for #define and #defnum.

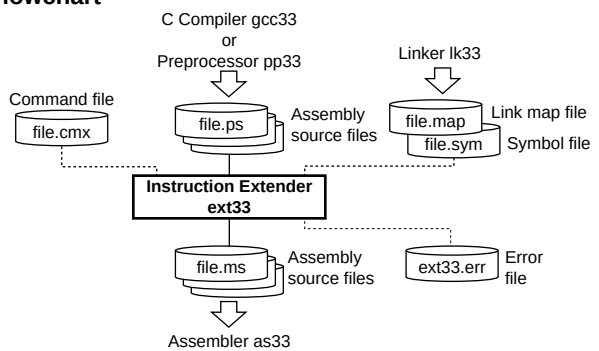
Warning Message

Multi define symbol.	The same define name or defnum name was multiply defined.

Outline

The ext33 expands the extended instructions described in the assembly source files into basic instructions for the as33. Further more, optimizes the source files by decreasing the ext instruction.

Flowchart



Start-up Command Usage

```
Extender 33 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
Usage:
    ext33 [options] filename
    ext33 [options] -c commandfile (.cmx)
Options:
    -e          : produce log file (ext33.err)
    -lk program  : optimize with program information (program.sym, program.map)
    -gp address  : optimize with global pointer (0x0 - 0xffffffff)
    -near        : specifies all xjmp and xcall extract 2 instruction
    -j threshold : specifies jump optimization threshold (0x100 - 0x1fffff)
Output:
    Assembler source file for as33(.ms)
Example:
    ext33 -e -lk test -gp 0x8000 test.ps
```

Error Messages

Cannot allocate memory.	There is not enough memory.
Cannot open input file "<file name>".	The input file cannot be opened.
Invalid input filename "<file name>".	An illegal file name was specified.
Cannot open output file "<file name>".	The output file cannot be opened.
Cannot open error file "ext33.err".	The error file cannot be opened.
Cannot open command file "<file name>".	The command file cannot be opened.
Cannot write error file.	The log cannot be written to the error file.
Cannot write output file "<file name>".	Data cannot be written to the file.
Too long filename "<file name>".	The file name is too long. (Max. 255 characters including path)
Invalid command file description "<description>".	There is an illegal description in the command file.
Invalid command file format.	The command file is not a text file.
Invalid jump threshold "<parameter>".	The parameter of the -j option is incorrect.
Invalid GP address "<parameter>".	The grobal pointer address is incorrect.
Too many input files.	Too many input files have been specified. (Max.682)
No input file is specified.	Input file has not been specified.
Invalid syntax.	The extended instruction has been described in illegal syntax.

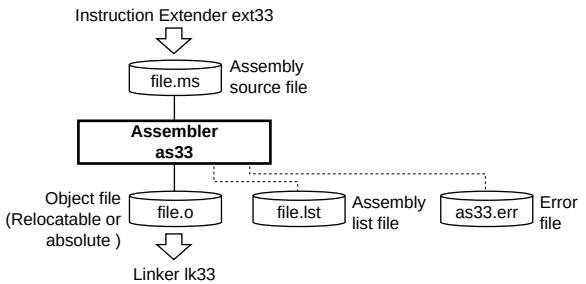
Warning Messages

Map file "<file name>" does not exist.	The link map file is not found.
Symbol file "<file name>" does not exist.	The symbol file is not found.
No map information in map file.	There is no map information for the input file in the link map file.
Invalid map file format.	The map file format is incorrect.
Invalid symbol file format.	The symbol file format is incorrect.
Cannot find the symbol "<symbol>" in symbol table.	There is no information for the symbol in the symbol file.
Operand exceeds maximum address.	The address operand has exceeded the effective range.
Invalid address operand.	The operand has specified illegal address boundary.
Invalid operand value.	The operand value is out of the range.

Outline

Converts the mnemonic of the source files into object codes (machine language) of the E0C33000. The results are output in a relocatable object file.

Flowchart



Start-up Command Usage

```
Assembler 33 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
Usage:
    as33 [options] <filename>
Options:
    -e : produce log file (as33.err)
    -g : generate debug information in object file
    -l : produce list file
Output:
    object file (.o)
    list file   (.lst)
    log file    (as33.err)
Example:
    as33 -e -g -l sample.ms
```

Error Messages

Out of memory.	There is not enough memory.
File open error.	The fie cannot be opened.
File access error.	The file cannot be accessed.
Invalid file name.	The souce file has the same extension (.o) as the output file.
Filename length limit exceeded - 255.	The file name has exceeded 255 characters.
Directory path length limit exceeded - 255.	The path has exceeded 255 characters.
Line length limit exceeded - 255.	The number of characters in one line has exceeded the limit.
Symbol name length limit exceeded - 32.	The number of characters in the symbol name has exceeded the limit.
Token length limit exceeded - 64.	The numeric data has exceeded 64 digits.
Multiple statements on the same line.	Two or more statements were described in one line.
Invalid statement syntax.	An illegal character is described.
Invalid instruction. - "<instruction>"	A non-existing instruction was described.
Invalid register. - "<register>"	The register name has a specification error.
Invalid directive. - "<directive>"	A non-existing pseudo-instruction was described.
Invalid symbol mask. - "<mask>"	The symbol mask has a description error.
Invalid instruction syntax.	The instruction has a syntax error.
Invalid directive syntax.	The pseudo-instruction has a syntax error.
Multiply declared symbol. - "<symbol>"	A symbol of the same name was declared in multiple locations.
Multiply defined symbol. - "<symbol>"	A symbol of the same name (label) was defined in multiple locations.
Incorrect section type for statement.	A statement not permissible for the current section was described.
Memory mapping conflict.	The address is duplicated.
Section count limit exceeded - 256.	The number of sections has exceeded the limit.

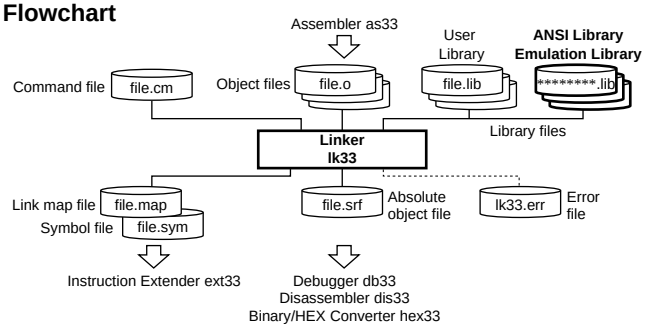
Warning Messages

Numeric range.	The immediate operand or parameter has exceeded the specifiable range.
Unknown escape sequence.	The .ascii pseudo-instruction contains an illegal escape sequence.
Escape sequence out of range for character.	The .ascii pseudo-instruction contains a code exceeded 0xff.
\x used with no following hex digits.	A hexadecimal number did not follow the \x in the .ascii pseudo-instruction.

Outline

Defines the memory locations of object codes created by the as33, and creates executable object codes. This tool puts together a multiplicity of object files and library objects into one file.

Flowchart



Start-up Command Usage

```
Linker 33 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
Usage:
    lk33 [options] <filenames>
Options:
    -e : produce log file (lk33.err)
    -g : generate debug information in output file
    -s : generate symbol information (.sym)
    -m : generate map information (.map)
    -c CommandFile : execute lk33 commands from CommandFile (.cm)
Output:
    SRF33 object file (.srf)
Example:
    lk33 -e -g -s -m -c test.cm
```

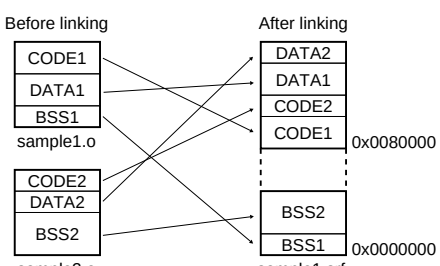
Linker Commands

-w	Sets warning level. (multiple defined global symbols in BSS do not issur a warning)
-l <library path>	Sets a library file search path. (Max. 4 paths)
-o <output file>	Sets an output file name.
-defsym <symbol> = <value>	Sets global symbols. (Max.255)
-d	Deletes the BSS area for the multiple defined global symbol.
-code <address/section>	Sets a relocatable code-section start address.
-code <address/section> { <files>}	Sets a code-section start address for the specified files.
-data <address/section>	Sets a relocatable data-section start address.
-data <address/section> { <files>}	Sets a data-section start address for the specified files.
-bss <address/section>	Sets a relocatable bss-section start address.
-ucode <address/section>	Sets a virtual code-section start address for relocatable code sections.
-ucode <address/section> { <files>}	Sets a shared code-section start address for the specified files.
-udata <address/section>	Sets a virtual data-section start address for relocatable data sections.
-udata <address/section> { <files>}	Sets a shared data-section start address for the specified files.
-ubss <address/section> { <files>}	Sets a shared bss-section start address for the specified files.
-objsym	Creates section symbols.
-section <section>	Sets a virtual/shared section name.
-section <section> = <address>	Sets a output section name and the start address.

Examples of Relocation

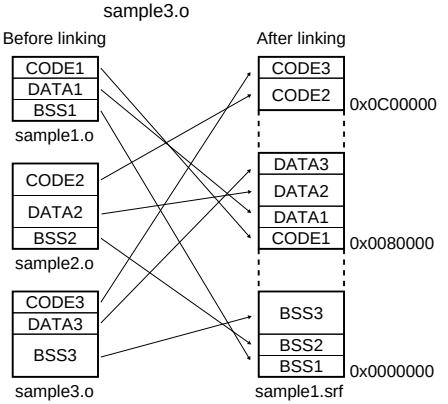
Standard relocation

Command file: sample1.o
sample2.o



Relocation of specific files

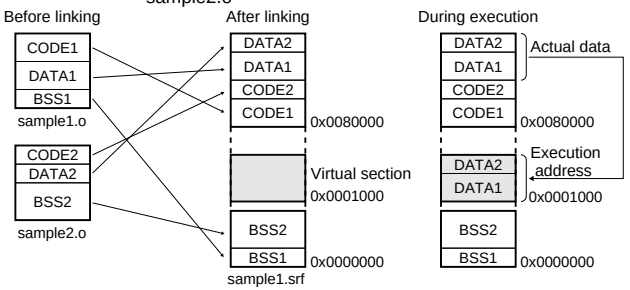
Command file: -code 0x0080000
-bss 0x0000000
-code 0x0c00000 {sample2.o sample3.0}
sample1.o
sample2.o
sample3.o



Examples of Relocation

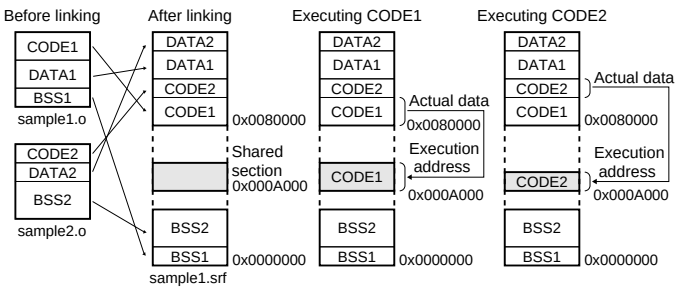
Relocation of virtual section

Command file: -udata 0x01000
sample1.o
sample2.o



Relocation of shared section

Command file: -ucode 0x0a000 {sample1.o sample2.o}
sample1.o
sample2.o



Error Messages

Cannot open input file.	The input file cannot be opened.
Cannot open output file.	The output file cannot be opened.
Cannot open working file.	The temporary file cannot be opened.
Cannot allocate memory.	There is not enough memory.
Cannot read a file.	The file cannot be read.
Unknown option near line = #.	The linker command in the linker command file is illegal.
Invalid parameter near line = #.	The command parameter is illegal.
Not define section name near line = #.	The section name has not been defined.
Uninitialized section name near line = #.	The section name has no defined address.
Code section map out of range.	The mapping address of the code section is out of the range.
Data section map out of range.	The mapping address of the data section is out of the range.
Bss section map out of range.	The mapping address of the bss section is out of the range.
Section mapping conflict, Address = <address>.	The address of the section is duplicated.
Too many object files.	The number of object files has exceeded the limit. (Max. 4000)
Too many output sections.	The number of output sections has exceeded the limit. (Max. 256)
Too many input sections.	The numbers of sections to be allocated in the default section has exceeded the limit. (Max. 4000)
Too many library files.	The number of library files has exceeded the limit. (Max. 256)
Too many global bss symbols.	The number of global sysbols in bss sections has exceeded the limit. (Max. 1024)
Too many object symbols.	The number of object sysbols has exceeded the limit. (Max. 36000)
Too many U sections.	The number of U sections has exceeded the limit. (Max. 256)
Too many section symbols.	The number of section symbols has exceeded the limit. (Max. 256)
Too many U section symbols.	The number of U section symbols has exceeded the limit. (Max. 256)
No object files.	Input object files have not been specified.
Not srf33 object file or library file.	The input file is not an object file or a library file in the srf33 format.
Chain information size is greater than file size.	The srf33 object file is illegal.
Chain seek address is greater than file size.	The srf33 object file is illegal.
Undefined symbol type.	The srf33 object file is illegal.
Undefined relocation type.	The srf33 object file is illegal.

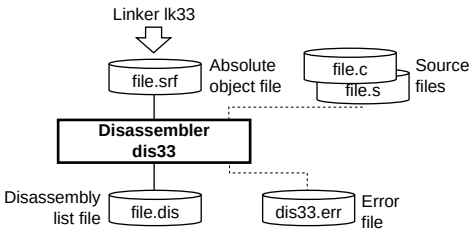
Warning Messages

'<symbol>' already defined in '<file name2>'.	The symbol was defined duplicatedly.
unresolved external symbol '<symbol>'.	Reference was made to an undefined symbol.
Out of relocation range, Address = <address>.	The instruction exceeded the relocatable range.
Cannot mapping to USection '<file name>(<section>)'.	The section cannot be allocated in the U section.
Cannot find relocatable section '<file name>(<section>)'.	The relocatable section cannot be found.
Alignment <address 1> -> <address 2>.	The address is aligned to 4-byte boundary.
Cannot access section information.	The srf33 object file is illegal.

Outline

Disassembles the object file delivered in the srf33 format by the linker, and creates a list file that can be referred to with mnemonic and source codes. You will find this function effective when you need to view the correspondence between source files and absolute addresses after linking them.

Flowchart



Start-up Command Usage

```
Disassembler 33 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x.
Usage:
    dis33 [options] <file name>
Options:
    -e : produce log file (dis33.err)
    -m : generate disassemble code with source mix
    -c : generate disassemble code only
    -d : generate data dump only
    -a address1 address2
       : specify disassemble area
         address1 - start address, hexadecimal number
         address2 - end address, hexadecimal number
Output:
    Disassemble file (.dis)
    Log file (dis33.err)
Example:
    dis33 -e -m -a 0x80000 0x8ffff sample.srf
```

Error Messages

Cannot open file, <file name>.	The file cannot be opened.
Cannot close file, <file name>.	The file cannot be closed.
Cannot write to file, <file name>.	Data cannot be written to the file.
Cannot close SROFF file.	The object file cannot be closed.
Cannot load data, memory allocation failure.	The section information cannot be loaded due to a memory allocation error.
Cannot load debug information, memory allocation failure.	The debugging information cannot be loaded due to a memory allocation error.
Cannot load data, file read failure, <file name>.	The section information cannot be loaded due to a file read error.
Cannot load debug information, file read failure, <file name>.	The debugging information cannot be loaded due to a file read error.
Invalid file name, <file name>.	".dis" is specified for the input file extension.
Too many sections.	The srf33 object file is illegal.
Cannot load data, please check SROFF file.	The srf33 object file is illegal.
Cannot load debug information, please check SROFF file.	The srf33 object file is illegal.

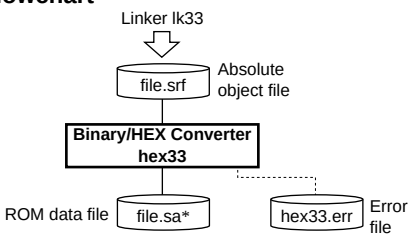
Warning Messages

No debug information.	There is no debugging information in the input file.
Cannot open file, <file name>.	The source file cannot be opened.
Line number of source file is invalid.	The source lines are insufficient. The source might be modified.

Outline

Converts the object file delivered in the srf33 format by the linker into a Motorola S3 format HEX file for writing to the ROM or creating the mask data.

Flowchart



Start-up Command Usage

```
HEX Data Converter 33 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
Usage:
    hex33 [options] <start address> <end address> <file name>
Options:
    -e : produce log file (hex33.err)
    -x : add start and end addresses to the file name extension
    -z : make converted address absolute
    -r : check all data within start and end area
Output:
    hex file  (.sa, .sa_<start address>_<end address>)
    log file  (hex33.err)
Example:
    hex33 -x -z 80000 80fff sample.srf
```

Error Messages

File open error.	The file cannot be opened.
File write error.	Data cannot be written to the file.
File read error.	The file cannot be read.
Memory allocation error.	There is not enough memory.
"<file name>" file could not be opened.	The file cannot be opend.
Input file is not SRF33 file.	The input file is not a file in srf33 format.
Start address error.	The conversion start address is illegal.
End address error.	The conversion end address is illegal.
Out of area in address <address>.	The conversion data exceeded the specified address range.
Chain information size is greater than file size.	The srf33 object file is illegal.
Chain seek address is greater than file size.	The srf33 object file is illegal.
File control flag error.	The srf33 object file is illegal.
Section address error.	The srf33 object file is illegal.
Section ID error.	The srf33 object file is illegal.
Data conflicted at <address>.	The srf33 object file is illegal.

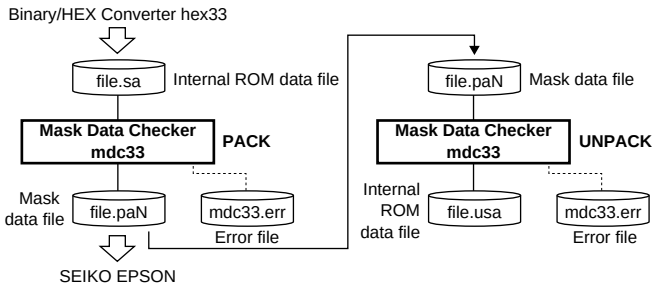
Warning Message

Section information chain is not found.	The srf33 object file is illegal.

Outline

Converts the internal ROM HEX data file delivered by the binary/HEX converter into the mask data.

Flowchart



Start-up Command Usage

```
Mask Data Checker 33 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
Usage:
    mdc33 [options] <ROM size> <file name>
Options:
    -e : produce log file (mdc33.err)
    -p : pack to mask data file
    -u : unpack from mask data file
Output:
    in pack   : pack 33XXXXXX.sa to 33XXXXXX.pa0
    in unpack : unpack 33XXXXXX.paN to 33XXXXXX.usa
Example:
    mdc33 -p 4 331040a0.sa
    mdc33 -u 4 331040a0.pa0
    mdc33 -e -p 4 331040a0.sa
```

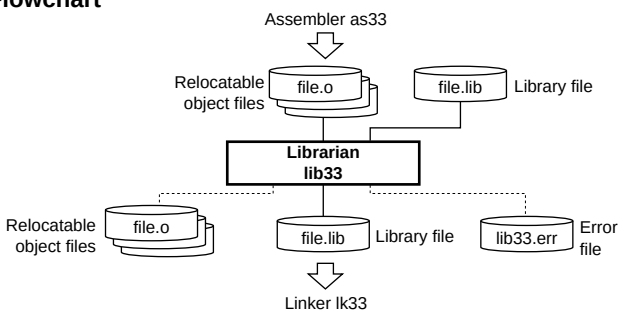
Error Messages

Invalid ROM Size	The specified ROM size is out of the range (1KB–512KB).
Pack file name <file> should be 33XXXXXX.sa	The pack file name is illegal.
UnPack file name <file> should be 33XXXXXX.paN	The unpack file name is illegal.
<file name> file is not found	The file cannot be found.
<file name> file can not open	The file cannot be opened.
mdc33.err file can not open	The mdc33.err file cannot be opened.
Hex data error : Invalid S3 file	The input file is not a file in Motorola S3 format (when packing).
Hex data error : Invalid S3 record	The S3 record is illegal (when packing).
Hex data error : Invalid S3 address	The S3 address is ilegal (when packing).
Hex data error : Invalid S3 data	The data is illegal (when packing).
Hex data error : S3 Check sum error	The check sum is incorrect (when packing).
Hex data error : Invalid S7 record	The end record is illegal (when packing).
Hex data error : Invalid comment	The header or foder is illegal (when unpaking).
Hex data error : Invalid S2 file	The input file is not a file in Motorola S2 format (when unpacking).
Hex data error : Invalid S2 record	The S2 record is illegal (when unpacking).
Hex data error : Invalid S2 address	The S2 address is ilegal (when unpacking).
Hex data error : Invalid S2 data	The data is illegal (when unpacking).
Hex data error : S2 Check sum error	The check sum is incorrect (when unpacking).
Hex data error : Invalid S8 record	The end record is illegal (when unpacking).

Outline

The lib33 can register relocatable object files to libraries, delete objects from libraries and extract library modules to object files.

Flowchart



Start-up Command Usage

```
Librarian 33 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
Usage:
    lib33 [option] <library-file> <files>
Options:
    -e : produce log file (lib33.err)
    -a : add new files after an existing member of the library
    -d : delete members from the library
    -l : display a table listing the contents of library
    -x : extract members from the library
Output:
    srf33 library file (.lib)
Example:
    lib33 -a libc.lib putc.o getc.o
```

Error Messages

Cannot open file	The file cannot be opened.
Not srf33 library file	The library is not a file in srf33 library format.
Max object files	The number of objects exceeded the limit (Max. 256).

Warning Messages

Not srf33 object file	The object file is not a file in srf33 format.
Multiple object file '<file name>'	The object file has already been registered.
Cannot find '<file name>' in library	The object name cannot be found in the library.

Outline

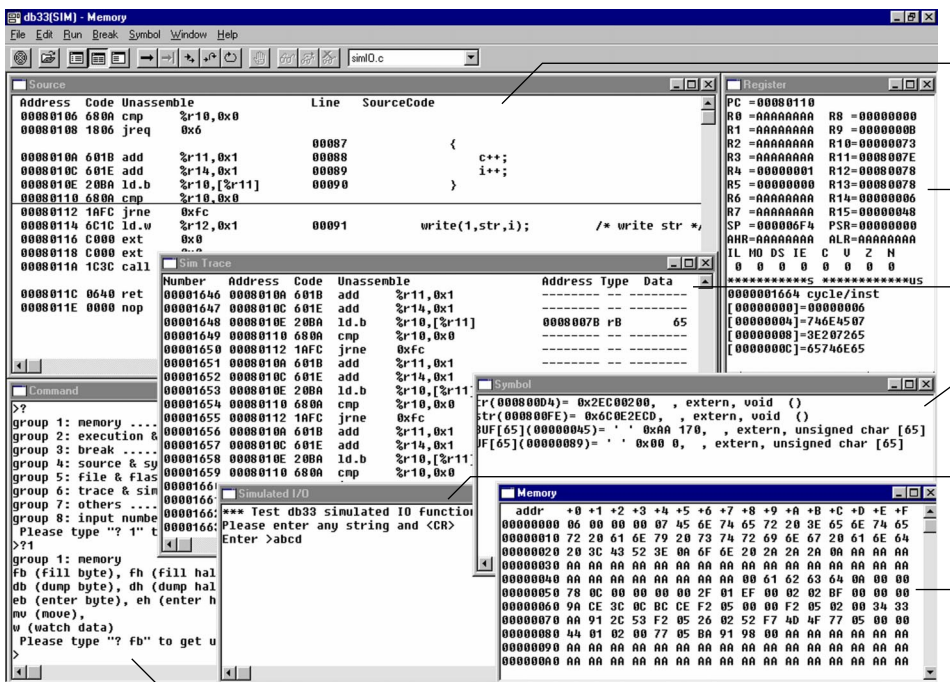
The db33 serves to perform debugging by controlling the ICE33 hardware tool. It also comes with a simulating function that lets you perform debugging on a personal computer without using the ICE33. Commands that are used frequently, such as break and step, are registered on the tool bar, minimizing the necessary keyboard operations. Moreover, sources, symbols, registers, and command execution results can be displayed in multi windows, with resultant increased efficiency in the debugging tasks.

Start-up Command Usage

```
Debugger for 33 Ver x.xx
Copyright (C)  SEIKO EPSON CORP. 199x

 - usage -
db33.exe -p parameter_file [-sim] [-c command_file] [-w] [-comX] [-b baud_rate] [-sf] [lptX]
    -p            ... parameter file
    -sim          ... simulator mode
    -c            ... command file
    -w            ... open only command window
    -comX (X:1-4) ... com port, default com1
    -b            ... baud rate, 4800, 9600, 19200, 38400(default), 57600, 115200 [bps]
    -sf           ... display with small font
    -lptX (X:1-2) ... parallel port
```

Windows



[Source] window
Displays programs with disassemble codes, source codes or disassemble and source codes.

[Register] window
Displays register values, watched memory and execution counter values.

[Trace] window
Displays traced data.

[Symbol] window
Displays monitored symbol information.

[Simulated I/O] window
Used to input and display data for simulated I/O function.

[Memory] window
Displays memory data.

[Command] window
Used to enter debug commands and display the execution results.

Tool Bar

Escape break Breaks the program being executed forcibly.

Load file Loads an srf33 absolute object file.

Source Switches the [Source] window to the source display mode.

Mix Switches the [Source] window to the mix display mode.

Unassemble Switches the [Source] window to the disassemble display mode.

Go Executes the program continuously from the current PC address.

Go to *1 Executes the program from the current PC address to the address pointed with the cursor in the [Source] window.

Step Executes the program 1 step at the current PC address.

Next Executes the program 1 step at the current PC address. Functions, subroutines and software interrupt routines are executed as 1 step.

Reset cold Cold-resets the CPU.

Software break *1 Sets the the cursor position in the [Source] window to a software break point.

Symbol watch *1 Displays the information of the symbol pointed with the cursor in the [Source] window to the [Command] window.

Symbol add *1 Registers the symbol pointed with the cursor in the [Source] window to the symbol monitor list and displays the information to the [Symbol] or [Command] window.

Symbol delete *2 Deletes the symbol pointed with the cursor in the [Symbol] window from the symbol monitor list.

Tool Bar

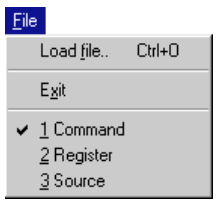
[Select source] combo box
This box is used to select the source file name of the program to be displayed in the [Source] window. The source file names listed in this box are obtained from the debugging information in the loaded object file.
Source files can be selected even if the [Source] window is set in the unassemble display mode. In this case, the [Source] window displays the codes corresponding to the selected source.

Function key

[F3] (Command redisplay)
Redisplays the command previously entered in the [Command] window at the prompt position. The redisplayed command can be executed again by pressing the [Enter] key.

*1: Can be used when the [Source] window is activated.
*2: Can be used when the [Symbol] window is activated.

Menus



[File] menu

Load File... Loads an srf33 absolute object file.
Exit Terminates the db33.

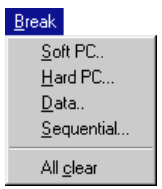
When selecting a window name from the list, the window becomes active.

[Edit] menu

Paste *1 Pastes the copied text (command) in the [Command] window.

[Run] menu

Go Executes the program continuously from the current PC address.
Go to *2 Executes the program from the current PC address to the address pointed with the cursor in the [Source] window.
Step Executes the program 1 step at the current PC address.
Next Executes the program 1 step at the current PC address. Functions, subroutines and software interrupt routines are executed as 1 step.
Reset cold Cold-resets the CPU.
Reset hot Hot-resets the CPU.



[Break] menu

Soft PC... Sets software PC break points.
Hard PC... Sets a hardware PC break points.
Data... Sets a data break condition.
Sequential... Sets sequential break conditions.
All clear Clears all break conditions.

Menus

[Symbol] menu

Watch *2 Displays the information of the symbol pointed with the cursor in the [Source] window to the [Command] window.
Add *2 Registers the symbol pointed with the cursor in the [Source] window to the symbol monitor list and displays the information to the [Symbol] or [Command] window.
Delete *3 Deletes the symbol pointed with the cursor in the [Symbol] window from the symbol monitor list.

[Window] menu

Opens or activates the selected window.

[Help] menu

About db33... Displays the version of the db33.

*1: Valid only for the [Command] window.
*2: Can be used when the [Source] window is activated.
*3: Can be used when the [Symbol] window is activated.

Debug Commands

Memory operation

fb [addr1 addr2 data]	Fills a memory area (addr1–addr2) with byte data.
fh [addr1 addr2 data]	Fills a memory area (addr1–addr2) with half word data.
fw [addr1 addr2 data]	Fills a memory area (addr1–addr2) with word data.
db [addr1 [addr2]]	Displays memory (addr1–(addr2)) data in byte units.
dh [addr1 [addr2]]	Displays memory (addr1–(addr2)) data in half word units.
dw [addr1 [addr2]]	Displays memory (addr1–(addr2)) data in word units.
eb [addr1]	Rewrites memory (addr1–) data in byte units.
eh [addr1]	Rewrites memory (addr1–) data in half word units.
ew [addr1]	Rewrites memory (addr1–) data in word units.
mv [addr1 addr2 addr3]	Copies memory area (addr1–addr2→addr3–).
w	Sets watch data addresses.

Register operation

rd	Displays register data.
rs	Sets register data.

Program execution

g [addr]	Executes the program from current PC (to addr).
s [step]	Executes the program 1 step or the step count specified.
n [step]	Executes the next operation 1 step or the step count specified.

CPU reset

rstc	Cold-resets the CPU.
rsth	Hot-resets the CPU.

Interrupt

int [type level]	Generates an interrupt. (Simulator mode only)

Break

bp [no. status [addr]]	Sets/clears software PC break points.
bs [addr]	Sets a software PC break point.
bc addr	Clears the software PC break point set at addr.
bh [addr]	Displays or sets hardware PC break point.
bhc	Clears the hardware PC break point.
bd [mode addr {r | w | *}]	Sets data break condition.
bsq [mode [c1 [c2 [c3]]]]	Sets sequential break condition. (ICE mode only)
bl	Displays all the set break conditions.
bac	Clears all the set break conditions.

Program display

u [addr]	Displays the program in disassembly mode from PC or addr.
sc [addr]	Displays the program in source mode from PC or addr.
m [addr]	Displays the program in mix mode from PC or addr.
ss string	Displays the program from the line that contains the string. (source mode only)

Symbol information

sy [option]	Displays symbol information list.
sa [symbol [-switch]]	Registers the symbol to the symbol monitor list.
sd no.	Deletes the symbol from the symbol monitor list.
sw symbol [-switch]	Displays the symbol information.
sw scope	Displays the symbol information list.

File

lf [file]	Loads an srf33 absolute object file.
lh [file offset]	Loads a Motorola S3 format file.
lo [file]	Loads an option file. (ICE mode only)
maf	Displays the flash memory map information.

Flash memory

ifl	Reads from the flash memory. (ICE mode only)
sfl	Writes to the flash memory. (ICE mode only)
efl	Erases the flash memory. (ICE mode only)

Trace

tm	Sets trace mode.
td [no.]	Displays trace data. (ICE mode only)
ts	Searches trace data. (ICE mode only)
tf	Saves trace data to a file. (ICE mode only)

Simulated I/O

stdin	Sets the simulated input condition.
stdout	Sets the simulated output condition.

Others

com [file]	Executes commands in a command file.
cmw [file]	Executes commands in a command file with interval.
log [file]	Output the log to a file.
od	Displays option data. (ICE mode only)
ct [[option] value]	Converts data into other types and displays the results.
ext [addr]	Converts a instruction into an extended instruction format.
ma	Displays the map information.
md	Sets debugger operating modes.
q	Terminates the debugger.
?	Displays the command usage.

Parameters

s/n (step=0–65535), **int** (type=0–127, level=0–15), **bp** (no.=0–15, status=1:set/2:enable/3:disable/4:clear), **bd/bsq** (mode=1:set/2:clear), **bsq** (hit = 1: c1/2:c1&c2/3:c1–c3, c1–3=<addr><16-bit pattern><bus type>, bus type=0:All/1:Inst/2:VecR/3:DatR/4DatW/5:StkR/6:StkW/7:DmaR/8:DmaW), **sy** (option=#:file list, /:function list,@[string]:tag list, [file]/[function]/[string]:symbol list)
sa/sw (-switch:display format=-b#, -d#, -u#, -h#, -c, -f, -df), **sd** (no.=1–99)
sw (scope: [file]/[function]/), td (no.=0–32767), **ct** (option=::binary, .:hex→double, ':hex→char string, ":char string→hex)

Command Parameter Entry Format

Number

Usable letters	Default:	[0–9]
	Decimal number:	+[0–9]
	Hexadecimal number:	0x[0–9, a–f, A–F]
Default format	Decimal number ("+" can be omitted.)	
Effective range	Depending on the command	

Data

Usable letters	Default:	[0–9, a–f, A–F]
	Decimal number:	+[0–9]
	Hexadecimal number:	0x[0–9, a–f, A–F]
Default format	Hexadecimal number ("0x" can be omitted.)	
Effective range	Depending on the command	

Address

Usable letters	Default:	[0–9, a–f, A–F]
	Decimal number:	+[0–9]
	Hexadecimal number:	0x[0–9, a–f, A–F]
	Symbol:	See Symbol
	Line number:	See Line number
Default format	Hexadecimal number ("0x" can be omitted.)	
Effective range	0x0–0x0fffffff	

Line number

Format	#<line number>
	<file name>#<line number>
Usable letters	[0–9] (only a decimal number without sign.)
Effective range	1–Number of lines in the source file

Symbol

Usable letters	*, _, a–z, A–Z, 0–9, ., ->, [,]	
	eg.) i *fp ar[2] st->mem st.mem	
Symbol serch path	current block→current function→current file→extern→hexadecimal	
Scope specification	[file]/[function]/symbol ("." can be used for the current file or function.)	
	//symbol	extern symbol.
	/./symbol	static or auto variable in the current function.
	.//symbol	static variable in the current file.
	file//symbol	static variable in the file.
	/function/symbol	static or auto variable in the function.
	./function/symbol	static or auto variable in the function of the current file.
	file/function/symbol	static or auto variable in the function of the file.

Parameter File

Parameters

CHIP	<name>	Chip name (33XXX)
IROM	<size>	Internal ROM size (0–20000)
FOPT	<size>	Option size (0–3fff)
PRC VER	<ver1> <ver2>	PRC board version range (0–ff)
PRC STATUS	**********	PRC board status (16 bits), [0, 1, *]
{MCU | MPU}		Boot address (MCU=0x80000, MPU=0xc00000)
VER	<ver>	Parameter file version for user (0–ff)

; Emulation memory allocation (max 8 areas, 1MB/area, 1MB boundary)

EMRAM	#00000 #fffff	RAM emulation area (#=0–ff), W only
EMROM	#00000 #fffff	ROM emulation area (#=0–ff), R/W

; Map allocation (max 31 areas, 256-byte boundary)

RAM	<start addr> <end addr>	Internal RAM or target board RAM map, R/W
IO	<start addr> <end addr>	Internal I/O or target board I/O map, R/W
ROM	<start addr> <end addr>	Target board ROM map, W only
ERAM	<start addr> <end addr>	Emulation memory map (for RAM), R/W
EROM	<start addr> <end addr>	Emulation memory map (for ROM), W only
EIO	<start addr> <end addr>	Emulation memory map (for I/O), R/W

; Stack area except internal RAM area (max 8 areas, 256-byte boundary)

STACK	<bottom addr> <top addr>	Stack map
END		End mark

Status/Error/Warning Messages

Break status messages

Break by software PC break	Break caused by software PC breakpoint
Break by hardware PC break	Break caused by hardware PC breakpoint
Break by temporary break	Break caused by temporary breakpoint
Break by data break	Break caused by data break condition
Break by read memory	Data break caused by reading memory
Break by write memory	Data break caused by writing to memory
Break by sequential break	Break caused by sequential break condition
Key Break	Break caused by [Escape break] button
Break by accessing no map area	Break caused by accessing no map area
Break by writing ROM area	Break caused by writing to ROM area
Break by out of SP area	Break caused by accessing outside stack area
Break by external break	Break caused by signal input to ICE33 BRKIN pin
Break by illegal instruction	Break caused by executing illegal instruction in simulator mode

Error messages

address1 > address2	The beginning address is larger than end address.
Address is in no map area.	The specified address (symbol) is out of the mapped area.
Address is not 2 byte boundary.	The program code address is not a 2-byte boundary address.
Address range (0-0xFFFFFFF).	The address is out of the range.
Already exist input address.	The address has been set to a break point.
Aymbol not in scope.	The symbol cannot be found in the scope.
Break number (1-16).	The software PC break point number is out of the range.
Cannot add symbol any more.	99 symbols have been registered.
Cannot allocate memory.	Memory cannot be allocated.
Cannot close file.	The file cannot be closed.
Cannot get file status.	The file information is incorrect.
Cannot get input, please check the system.	An error has occurred during input process.
Cannot get memory.	Memory allocation has failed.
Cannot load data, file open failure.	The srf33 file load has failed; the file cannot be opened.
Cannot load data, file read failure.	The srf33 file load has failed; the file cannot be read.
Cannot load data, memory allocation failure.	The srf33 file load has failed; memory cannot be allocated.
Cannot load data, please check SRF33 file.	The srf33 file load has failed; some file other than srf33 executable format is specified.
Cannot load debug information, debug information is wrong.	The debug information load has failed; the debug information is illegal. (Program/ data is loaded successfully.)
Cannot load debug information, file open failure.	The debug information load has failed; the source file cannot be opened. (Program/ data is loaded successfully.)
Cannot load debug information, file read failure.	The debug information load has failed; the source file cannot be read. (Program/ data is loaded successfully.)

Error messages

Cannot load debug information, memory allocation failure.	The debug information load has failed; memory cannot be allocated. (Program/ data is loaded successfully.)
Cannot load debug information, please check SRF33 file.	The debug information load has failed; the srf33 format is illegal. (Program/ data is loaded successfully.)
Cannot load debug information, too many lines.	The debug information load has failed; too many source lines are included. (Program/ data is loaded successfully.)
Cannot open com port.	The COM port or baud rate cannot be set.
Cannot open file any more.	The number of files exceeds the limit.
Cannot open file.	The file cannot be opened.
Cannot open parallel port.	The parallel port cannot be opened.
Cannot open parameter file.	The parameter file cannot be opened.
Cannot open stdio file.	The stdio file cannot be opened.
Cannot read chip name.	Chip name information (.par file) cannot be read.
Cannot read emulation memory map.	Emulation memory map information (.par file) cannot be read.
Cannot read FO area size.	FOPT size information (.par file) cannot be read.
Cannot read IROM size.	Internal ROM size information (.par file) cannot be read.
Cannot read MCU/MPU information.	MCU/MPU mode information (.par file) cannot be read.
Cannot read parameter file version.	Parameter file version information (.par file) cannot be read.
Cannot read PRC board status.	PRC board status information (.par file) cannot be read.
Cannot read PRC board version.	PRC board version information (.par file) cannot be read.
Cannot read stdin file.	The stdin file cannot be read.
Cannot set address any more.	16 software PC break points have been set.
Cannot set hardware PC break.	The hardware PC break point cannot be set.
Cannot set temporary break.	Temporary break point cannot be set.
Cannot write log file.	Log data cannot be written to the file.
Cannot write stdout file.	The output data cannot be written to the stdout file.
Cannot write trace file.	Trace data cannot be written to the file.
Chip name should be 5 characters.	The chip name length is not 5 characters.
Chip name should be start with "33".	The chip name must begin with 33.
Command is not supported at present mode.	A command not supported for the current mode (ICE or simulator) is executed.
Communication data size error.	The communication data size is incorrect.
Communication error.	Overrun, framing, or BCC error has occurred during transmission from/to the ICE33.
CPU down.	The PRC board operates erratically.
CPU is not running.	The ICE33 CPU has stopped operating.
CPU is running.	The ICE33 CPU is executing.
Current mode is not source mode.	String search ia only available in the source display mode.

Status/Error/Warning Messages

Error messages	
Data alignment error.	Alignment in the srf file is incorrect.
Data incomplete.	The file structure is illegal.
Data range (0-0xFF).	The input data is out of the range.
Data range (0-0xFFFF).	The input data is out of the range.
Data range (0-0xFFFFFFFF).	The input data is out of the range.
Debug data failure.	The debugging data is illegal.
Diagnostic test failure.	The ICE33 self-diagnosis resulted in error.
Duplicate input address.	Same break address is set twice.
Duplicate input break number.	Same break point number is set twice.
Empty file.	The file does not contain data.
File end during guidance input.	The command file has ended in the middle of the parameters of the guidance format.
File not found.	The file cannot be found.
Flash memory error.	Error in writing or erasing flash memory.
Flash memory is not mapped.	The ICE flash memory is not mapped.
Flash ROM is protected.	Flash memory is protected against access.
Fo address range (0-0x3FFF).	The option dump address is out of the range.
Format error.	The format is illegal.
Function not found.	The function cannot be found.
ICE is busy.	The ICE33 is busy processing a job.
ICE is free run mode.	The ICE33 is operating in free-run mode.
ICE is maintenance mode.	The ICE33 is placed in maintenance mode.
ICE is not mapped.	The ICE built-in memory is not mapped.
ICE system error.	ICE system error has occurred.
Interrupt level (0-15).	The interrupt level is out of the range.
Interrupt type (0-215).	The interrupt type is out of range.
Invalid break address.	The break address has not been set.
Invalid break number.	The break point number has not been set.
Invalid command or parameter.	The specified command or parameter is invalid.
Invalid emulation memory map.	The emulation memory map information (.par file) is invalid.
Invalid file name.	The file name is invalid.
Invalid group number.	The command group number is invalid.
Invalid map command or invalid sequence.	The map file contains an illegal character or incorrect sequence.
Invalid memory map.	The memory map information (.par file) is invalid.
Invalid parameter.	The parameter is incorrect.
Invalid stack map.	The stack map information (.par file) is invalid.
Invalid value.	The input value is illegal.
IROM size is too long.	The IROM size is too large.
No "END" in parameter file.	There is no end mark (END) in the parameter file.

Error messages	
No map area.	The input address is out of the mapped area.
No symbol at the number.	Symbol is not registered in the specified number.
Not ASCII character.	The string contains some other ASCII character.
Not defined ID.	ICE33's response ID is invalid.
Not found input strings.	The string cannot be found.
Number of emulation memory is wrong.	The number of emulation memory block in the parameter file is invalid.
Number of parameter.	The number of parameters in the command is invalid.
On tracing.	The ICE33 is tracing execution data.
Over max include file number.	The number of include files exceeds the limit.
Parallel interface time out.	The file cannot be loaded through the parallel interface within the predefined time.
Parallel port is busy.	The parallel port is in busy status.
Pointer pointed no map area.	The pointer variable has pointed out of the mapped area.
Post line range (0-256).	The number of post-display line in the trace search is out of the range.
Pre line range (0-256).	The number of pre- display line in the trace search is out of the range.
Register variable cannot be changed to address.	Addresses cannot be specified with a register variables.
Reset timeout.	The ICE33 CPU cannot be reset.
Sequential break format error.	The sequential break condition is invalid.
Shared RAM is busy.	An ICE33 internal error has occurred.
Source window not opened.	The [Source] window is closed.
Start cycle number < End cycle number.	The end cycle number is greater than the start cycle number.
Stdout data size.	The output data size in the output buffer is illegal.
Step range (1-65535).	The step count is out of the range.
Symbol is too long.	The symbol name is too long.
Symbol not found.	The symbol cannot be found.
Target down.	The PRC board does not operate correctly or remains reset.
Target instruction cannot be extended.	The instruction cannot be extended by ext.
Time out.	Communication time-out.
Too many include.	Number of included files exceeds the limit.
Too many source file.	The source file is too large.
Trace range (0-32767).	The trace cycle number is invalid.
Verify error.	Verify error when writing to flash memory.
Wrong data.	Data in the file is incorrect.
Wrong header.	The file header is incorrect.

Status/Error/Warning Messages

Warning messages

Emulation memory address is not 1M byte boundary.	The emulation memory map address in the parameter file is not a 1MB boundary address.
FO size (0-0x4000), map as 0x4000.	The FO size is incorrect, so it is mapped as 0x4000.
FO size should be an even number, map as 0xXXXXXXXX.	The FO size must be an even number, so it is mapped as 0xXXXXXXXX.
Invalid line, move to next valid line.	The source line has no address. The next effective address is used.
IROM size (0-0x80000), map as 0x80000.	The IROM size is incorrect, so it is mapped as 0x80000.
Line number of source file is invalid.	The line number is not included in the source file.
Memory map is not 256 byte boundary.	The memory map (.par file) must be specified in 256-byte units.
No debug information.	The srf33 file does not have the debug information.
No source, display on mix mode.	There is no source information. The program is displayed in mix mode.
Number of source line exceeded 65535.	The line number is out of the range.
PRC status does not match.	The PRC board status is different from the parameter file.
PRC version does not match.	The PRC board version is different from the parameter file.
Round down to multiple of 2.	The input address is adjusted to a 2-byte boundary.
Round down to multiple of 4.	The input address is adjusted to a 4-byte boundary.
Stack map is not 256 byte boundary.	The stack map (.par file) must be specified in 256-byte units.

Outline

Executes the command lines in the make file according to the conditions that are specified in the dependency list. The compiler to the linker can be automatically executed.

Start-up Command Usage

```
Make for 33/63 Ver x.xx
Copyright (C) SEIKO EPSON CORP. 199x
Usage:
    make [options] [target]
Options:
    -f <file name> : makefile name
    -h  : output usage
    -n  : no command execution
Example:
    make -f test.mak
    make -f test.mak CLEAN
```

Error Messages

Cannot open XXXXXXXX	The make file cannot be opened.
Cannot open tmp file	The temporary file cannot be opened.
Invalid syntax near line #	There is a syntax error in the statement.
Invalid suffix .XXX	The suffix has not been defined.
Invalid macro name XXXX	The macro name has not been defined.
Abnormal termination in XXXX	An error occurred while processing the module.
Not enough memory	There is not enough memory.
Too long string	The string has exceeded 1000 characters.
No target found	The target cannot be found.
Don't know how to make XXXX	There is no target file.
Command exit with X	The command has been terminated abnormally.

Warning Message

XXXX is up-to-date	The target has been updated.

Syntax in Make File

Dependency list 1

<target file>: <dependent file 1> [<dependent file 2> ....]
[TAB <command 1>
TAB <command 2>
:]

The commands are executed in the following cases:
- when the target file has not been created
- when a dependent file is newer than the target file

When command is omitted, the corresponding suffix list will be executed.

Dependency list 2

<target name>:
[TAB <command 1>
TAB <command 2>
:]

The target name is used as a label.

Suffix definition

.SUFFIXES: .xxx .yyy .zzz ....

Suffix list

.<suffix of dependent file 1>.<suffix of target file>:
[TAB <command 1>
TAB <command 2>
:]

Macro definition

<macro name> = <macro body>

Macro reference

$(<macro name>)

Symbols

@<command>	Switches the command echo off.
-<command>	Ignores the execution results (error code).
$*	Replaced withthe target file name (excluding the suffix).
$@	Replaced withthe target file name (including the suffix).

Floating-point Emulation Library fp.lib

Double-type operation

__adddf3	Addition	(%r11, %r10) ← (%r13, %r12) + (%r15, %r14)
__subdf3	Subtraction	(%r11, %r10) ← (%r13, %r12) - (%r15, %r14)
__muldf3	Multiplication	(%r11, %r10) ← (%r13, %r12) * (%r15, %r14)
__divdf3	Division	(%r11, %r10) ← (%r13, %r12) / (%r15, %r14)
__negdf2	Sign change	(%r11, %r10) ← -(%r13, %r12)

Float-type operation

__addsf3	Addition	%r10 ← %r12 + %r13
__subsf3	Subtraction	%r10 ← %r12 - %r13
__mulsf3	Multiplication	%r10 ← %r12 * %r13
__divsf3	Division	%r10 ← %r12 / %r13
__negsf2	Sign change	%r10 ← -%r12

Type conversion

__fixunsdfsi	double → unsigned int	%r10 ← (unsigned int) (%r13, %r12)
__fixdfsi	double → int	%r10 ← (int) (%r13, %r12)
__floatsidf	int → double	(%r11, %r10) ← (double) %r12
__fixunssfsi	float → unsigned int	%r10 ← (unsigned int) %r12
__fixsfsi	float → int	%r10 ← (int) %r12
__floatsisf	int → float	%r10 ← (float) %r12
__truncdfsf2	double → float	%r10 ← (float) (%r13, %r12)
__extendsfdf2	float → double	(%r11, %r10) ← (double) %r12

Comparison

__fcmpd	double type	Changes %psr by (%r13, %r12) - (%r15, %r14)
__fcmps	float type	Chenges %psr by %r12 - %13

Integer Division Library idiv.lib

Integer division

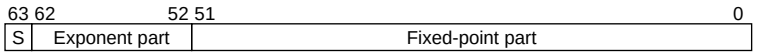
__divsi3	Signed division	%r10 ← %r12 / %r13
__udivsi3	Unsigned division	%r10 ← %r12 / %r13

Modulo operation

__modsi3	Signed operation	%r10 ← %r12 % %r13
__umodsi3	Unsigned operation	%r10 ← %r12 % %r13

Floating-point Data Format

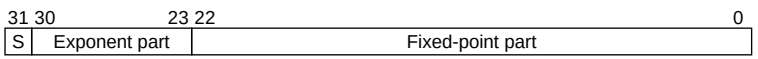
Double-type data format



Double-type effective range

+0:	0.0e+0 0x00000000 00000000
-0:	-0.0e+0 0x80000000 00000000
Maximum normalized number:	1.79769e+308 0x7fefffff ffffffff
Minimum normalized number:	2.22507e-308 0x00100000 00000000
Maximum unnormalized number:	2.22507e-308 0x000fffff ffffffff
Minimum unnormalized number:	4.94065e-324 0x00000000 00000001
Infinity:	0x7ff00000 00000000
-Infinity:	0xfff00000 00000000

Float-type data format



Float-type effective range

+0:	0.0e+0f 0x00000000
-0:	-0.0e+0f 0x80000000
Maximum normalized number:	3.40282e+38f 0x7f7fffff
Minimum normalized number:	1.17549e-38f 0x00800000
Maximum unnormalized number:	1.17549e-38f 0x007fffff
Minimum unnormalized number:	1.40129e-45f 0x00000001
Infinity:	0x7f800000
-Infinity:	0xff800000

io.lib (header file: stdio.h)

fopen( )	FILE *fopen(char *filename, char *mode);	(dummy)∗1
freopen( )	FILE *freopen(char *filename, char *mode, FILE *stream);	(dummy)∗1
fclose( )	int fclose(FILE *stream);	(dummy)
fflush( )	int fflush(FILE *stream);	(dummy)
fseek( )	int fseek(FILE *stream, long int offset, int orign);	(dummy)∗1
ftell( )	long int ftell(FILE *stream);	(dummy)
rewind( )	void rewind(FILE *stream);	(dummy)
fgetpos( )	int fgetpos(FILE *stream, fpos_t *ptr);	(dummy)
fsetpos( )	int fsetpos(FILE *stream, fpos_t *ptr);	(dummy)∗1
fread( )	size_t fread(void *ptr, size_t size, size_t count, FILE *stream);	∗1, ∗2
fwrite( )	size_t fwrite(void *ptr, size_t size, size_t count, FILE *stream);	∗1, ∗2
fgetc( )	int fgetc(FILE *stream);	∗2
getc( )	int getc(FILE *stream);	∗1, ∗2
getchar( )	int getchar( );	∗1, ∗2
ungetc( )	int ungetc(int c, FILE *stream);	∗1
fgets( )	char *fgets(char *s, int n, FILE *stream);	∗1, ∗2
gets( )	char *gets(char *s);	∗1, ∗2
fputc( )	int fputc(int c, FILE *stream);	∗2
putc( )	int putc(int c, FILE *stream);	∗1, ∗2
putchar( )	int putchar(int c);	∗1, ∗2
fputs( )	int fputs(char *s, FILE *stream);	∗1, ∗2
puts( )	int puts(char *s);	∗1, ∗2
remove( )	int remove(char *filename);	(dummy)∗1
rename( )	int rename(char *oldname, char *newname);	(dummy)∗1
setbuf( )	void setbuf(FILE *stream, char *buf);	(dummy)
setvbuf( )	int setvbuf(FILE *stream, char *buf, int type, size_t size);	(dummy)
tmpfile( )	FILE *tmpfile( );	(dummy)∗1
tmpnam( )	char *tmpnam(char *buf);	(dummy)∗1
feof( )	int feof(FILE *stream);	(dummy)
ferror( )	int ferror(FILE *stream);	(dummy)
clearerr( )	void clearerr(FILE *stream);	(dummy)
perror( )	void perror(char *s);	∗1, ∗2
fscanf( )	int fscanf(FILE *stream, char *format, ...);	∗1, ∗2
scanf( )	int scanf(char *format, ...);	∗1, ∗2
sscanf( )	int sscanf(char *s, char *format, ...);	∗1, ∗2
fprintf( )	int fprintf(FILE *stream, char *format, ...);	∗1, ∗2
printf( )	int printf(char *format, ...);	∗1, ∗2
sprintf( )	int sprintf(char *s, char *format, ...);	∗1, ∗2
vfprintf( )	int vfprintf(FILE *stream, char *format, va_list arg);	∗1, ∗2
vprintf( )	int vprintf(FILE *stream, char *format, va_list arg);	∗1, ∗2
vsprintf( )	int vsprintf(char *s, char *format, va_list arg);	

lib.lib (header file: stdlib.h, time.h)

malloc( )	void *malloc(size_t size);	∗1
calloc( )	void *calloc(size_t elt_count, size_t elt_size);	∗1
free( )	void free(void *ptr);	∗1
realloc( )	void *realloc(void *ptr, size_t size);	∗1
system( )	int system(char *command);	
exit( )	void exit(int status);	
abort( )	void abort( );	
atexit( )	int atexit(void (*func)(void));	
getenv( )	char *getenv(char *str);	
bsearch( )	void *bsearch(void *key, void *base, size_t count, size_t size, int (*compare)(void *, void *));	
qsort( )	void qsort(void *base, size_t count, size_t size, int (*compare)(void *, void *));	
abs( )	int abs(int x);	
labs( )	long int labs(long int x);	
div( )	div_t div(int n, int d);	∗1
ldiv( )	ldiv_t ldiv(int n, int d);	∗1
rand( )	int rand( );	
srand( )	void srand(unsigned int seed);	
atol( )	long int atol(char *str);	
atoi( )	int atoi(char *str);	∗1
atof( )	double atof(char *str);	∗1
strtod( )	double strtod(char *str, char **ptr);	∗1
strtol( )	long int strtol(char *str, char **ptr, int base);	∗1
strtoul( )	unsigned long int strtoul(char *str, char **ptr, int base);	∗1
clock( )	clock_t clock( );	(dummy)
asctime( )	char *asctime(struct tm *ts);	(dummy)
ctime( )	char *ctime(time_t *timeptr);	(dummy)
difftime( )	double difftime(time_t ti, time_t t2);	(dummy)
gmtime( )	struct tm *gmtime(time_t *t);	
localtime( )	struct tm *localtime(time_t *t);	(dummy)
mktime( )	time_t mktime(struct tm *tmptr);	
time( )	time_t time(time_t *tptr);	∗1

∗1 These functions need to declare and initialize the global variables.
∗2 These functions need to define the low-level functions and I/O buffers.

math.lib (header file: math.h, errno.h, float.h, limits.h)

fabs( )	double fabs(double x);	∗1
ceil( )	double ceil(double x);	∗1
floor( )	double floor(double x);	∗1
fmod( )	double fmod(double x, double y);	∗1
exp( )	double exp(double x);	∗1
log( )	double log(double x);	∗1
log10( )	double log10(double x);	∗1
frexp( )	double frexp(double x, int *nptr);	∗1
ldexp( )	double ldexp(double x, int n);	∗1
modf( )	double modf(double x, double *nptr);	∗1
pow( )	double pow(double x, double y);	∗1
sqrt( )	double sqrt(double x);	∗1
sin( )	double sin(double x);	∗1
cos( )	double cos(double x);	∗1
tan( )	double tan(double x);	∗1
asin( )	double asin(double x);	∗1
acos( )	double acos(double x);	∗1
atan( )	double atan(double x);	
atan2( )	double atan2(double y, double x);	∗1
sinh( )	double sinh(double x);	∗1
cosh( )	double cosh(double x);	∗1
tanh( )	double tanh(double x);	

ctype.lib (header file: ctype.h)

isalnum( )	int isalnum(char c);
isalpha( )	int isalpha(char c);
iscntrl( )	int iscntrl(char c);
isdigit( )	int isdigit(char c);
isgraph( )	int isgraph(char c);
islower( )	int islower(char c);
isprint( )	int isprint(char c);
ispunct( )	int ispunct(char c);
isspace( )	int isspace(char c);
isupper( )	int isupper(char c);
isxdigit( )	int isxdigit(char c);
tolower( )	int tolower(char c);
toupper( )	int toupper(char c);

stdarg.h

va_atart( )	void va_start(va_list ap, type lastarg);
va_arg( )	type va_arg(va_list ap, type);
va_end( )	void va_end(va_list ap);

∗1 These functions need to declare and initialize the global variables.

string.lib (header file: string.h)

memchr( )	char *memchr(char *s, int c, int n);
memcmp( )	int memcmp(char *s1, char *s2, int n);
memcry( )	char *memcpy(char *s1, char *s2, int n);
memmove( )	char *memmove(char *s1, char *s2, int n);
memset( )	char *memset(char *s, int c, int n);
strcat( )	char *strcat(char *s1, char *s2);
strchr( )	char *strchr(char *s, int c);
strcmp( )	int strcmp(char *s1, char *s2);
strcpy( )	char *strcpy(char *s1, char *s2);
strcspn( )	size_t *strcspn(char *s1, char *s2);
strerror( )	char *strerror(int code);
strlen( )	size_t strlen(char *s);
strncat( )	size_t strncat(char *s1, char *s2, int n);
strncmp( )	int strncmp(char *s1, char *s2, int n);
strncpy( )	char *strncpy(char *s1, char *s2, int n);
strpbrk( )	char *strpbrk(char *s1, char *s2);
strrchr( )	char *strrchr(char *s, int c);
strspn( )	size_t strspn(char *s1, char *s2);
strstr( )	char *strstr(char *s1, char *s2);
strtok( )	char *strtok(char *s1, char *s2);

∗1 Global Variable Declarations and Initialization

FILE _iob[FOPEN_MAX+1];	_iob[N]._flg=_UGETN; _iob[N]._buf=0; _iob[N]._fd=N; (N=0: for stdin, N=1: for stdout, N=2: for stderr)
FILE *stdin;	stdin=&_iob[0];
FILE *stdout;	stdout=&_iob[1];
FILE *stderr;	stderr=&_iob[2];
int errno;	errno=0;
unsigned int seed;	seed=1;
unsigned char *_STACK_TOP	_STACK_TOP=(unsigned char *)<stack top address>
unsigned char *_STACK_BOTTOM	_STACK_BOTTOM=(unsigned char *)<stack bottom address>
unsigned char *ucNxtAlcp;	ucNxtAlcp=(unsigned char *)<stack bottom address>
unsigned char *ucBefAlcp;	ucBefAlcp=(unsigned char *)NULL;
unsigned char *end_alloc;	end_alloc=(unsigned char *)<stack top address>
time_t gm_sec;	gm_sec=1;

∗2 Low-level Function Definitions

read( )	int read(int fd, char *buf, int nbytes); unsigned char READ_BUF[65]; (another variable name can be used.) unsigned char READ_EOF;
write( )	int write(int fd, char *buf, int nbytes); unsigned char WRITE_BUF[65]; (another variable name can be used.)

Symbols in the Instruction List

Registers/Register Data

%rd, rd: A general-purpose register (R0–R15) used as the destination register or the contents of the register.
%rs, rs: A general-purpose register (R0–R15) used as the source register or the contents of the register.
%rb, rb: A general-purpose register (R0–R15) that has stored a base address accessed in the register indirect addressing mode or the contents of the register.
%sd, sd: A special register (PSR, SP, ALR, AHR) used as the destination register or the contents of the register.
%ss, ss: A special register (PSR, SP, ALR, AHR) used as the source register or the contents of the register.
%sp, sp: Stack pointer or the contents of the stack pointer.

Memory/Addresses/Memory Data

[%rb]: Specification for register indirect addressing.
[%rb]+: Specification for register indirect addressing with post-increment.
[%sp+immX], [%rb+immX], [%rb+symbol±immX]:
Specification for register indirect addressing with a displacement.
[immX], [symbol±immX]:
Specification for absolute addressing.
B[XXX]: The address specified with XXX, or the byte data stored in the address.
H[XXX]: The half-word space in which the base address is specified with XXX, or the half-word data stored in the space.
W[XXX]: The word space in which the base address is specified with XXX, or the word data stored in the space.

Immediate

immX: A X-bit unsigned immediate data.
signX: A X-bit signed immediate data.

Symbol/Label

Symbol: A symbol that points an address.
Label: A branch destination label.

Bit Field

(X): Bit X of data.
(X:Y): A bit field from bit X to bit Y.
{X, Y···}: Indicates a bit (data) configuration.

Functions

←: Indicates that the right item is loaded or set to the left item.
+: Addition
-: Subtraction
&: AND
|: OR
^: XOR
!: NOT
×: Multiplication

Flags

MO: MAC overflow flag
DS: Dividend sign flag
Z: Zero flag
N: Negative flag
C: Carry flag
V: Overflow flag
–: Not changed
↔: Set (1), reset (0) or not changed
0: Reset (0)

R8

–: Basic instruction (not expanded)
○: Indicates that the R8 register (gp) may be used for expansion.
×: Indicates that the R8 register is not used for expansion.

R9

–: Basic instruction (not expanded)
○: Indicates that the R9 register is used under the most conditions for expansion.
△: Indicates that the R9 register may be used depending on the condition.
×: Indicates that the R9 register is not used for expansion.

D

○: Indicates that the instruction can be used as a delayed instruction.
–: Indicates that the instruction cannot be used as a delayed instruction.

Notes

- The instruction list contains the basic instructions in the E0C33000 instruction set and the extended instructions (x..., except for xor).
  "Italic basic instructions" indicate that the upper compatible extended instructions are provided.
  The extended instruction must be expanded with the ext33.
- Some extended instructions use the R9 register as a scratch register. Do not save a value to the R9 register before using an extended instruction.
- When the global pointer is specified, the R8 register is used to keep the global pointer address. Therefore do not change the R8 register value after setting the address.

Classification	Mnemonic		Function	Flags						R8	R9	D
	Opcode	Operand		MO	DS	C	V	Z	N			
Signed byte data transfer	ld.b	%rd, %rs	rd(7:0)←rs(7:0), rd(31:8)←rs(7)	–	–	–	–	–	–	–	–	–
		%rd, [%rb]	rd(7:0)←B[rb], rd(31:8)←B[rb](7)	–	–	–	–	–	–	–	–	–
		%rd, [%rb]+	rd(7:0)←B[rb], rd(31:8)←B[rb](7), rb←rb+1	–	–	–	–	–	–	–	–	–
		%rd, [%sp+imm6]	rd(7:0)←B[sp+imm6], rd(31:8)←B[sp+imm6](7)	–	–	–	–	–	–	–	–	–
		[%rb], %rs	B[rb]←rs(7:0)	–	–	–	–	–	–	–	–	–
		[%rb]+, %rs	B[rb]←rs(7:0), rb←rb+1	–	–	–	–	–	–	–	–	–
		[%sp+imm6], %rs	B[sp+imm6]←rs(7:0)	–	–	–	–	–	–	–	–	–
	xld.b	%rd, [imm32]	rd(7:0)←B[imm32], rd(31:8)←B[imm32](7)	–	–	–	–	–	–	×	○	–
		%rd, [symbol+imm32]	rd(7:0)←B[symbol+imm32], rd(31:8)←B[symbol+imm32](7)	–	–	–	–	–	–	○	○	–
		%rd, [symbol-imm32]	rd(7:0)←B[symbol-imm32], rd(31:8)←B[symbol-imm32](7)	–	–	–	–	–	–	×	○	–
		%rd, [%rb+imm32] *1	rd(7:0)←B[rb+imm32], rd(31:8)←B[rb+imm32](7)	–	–	↔	↔	↔	↔	×	△	–
		%rd, [%rb+symbol+imm32] *1	rd(7:0)←B[rb+symbol+imm32], rd(31:8)←B[rb+symbol+imm32](7)	–	–	↔	↔	↔	↔	×	△	–
		%rd, [%rb+symbol-imm32] *1	rd(7:0)←B[rb+symbol-imm32], rd(31:8)←B[rb+symbol-imm32](7)	–	–	↔	↔	↔	↔	×	○	–
		%rd, [%sp+imm32]	rd(7:0)←B[sp+imm32], rd(31:8)←B[sp+imm32](7)	–	–	–	–	–	–	×	×	–
		[imm32], %rs	B[imm32]←rs(7:0)	–	–	–	–	–	–	×	○	–
		[symbol+imm32], %rs	B[symbol+imm32]←rs(7:0)	–	–	–	–	–	–	○	○	–
		[symbol-imm32], %rs	B[symbol-imm32]←rs(7:0)	–	–	–	–	–	–	×	○	–
		[%rb+imm32], %rs *1	B[rb+imm32]←rs(7:0)	–	–	↔	↔	↔	↔	×	△	–
		[%rb+symbol+imm32], %rs *1	B[rb+symbol+imm32]←rs(7:0)	–	–	↔	↔	↔	↔	×	△	–
		[%rb+symbol-imm32], %rs *1	B[rb+symbol-imm32]←rs(7:0)	–	–	↔	↔	↔	↔	×	○	–
		[%sp+imm32], %rs	B[sp+imm32]←rs(7:0)	–	–	–	–	–	–	×	×	–
Unsigned byte data transfer	ld.ub	%rd, %rs	rd(7:0)←rs(7:0), rd(31:8)←0	–	–	–	–	–	–	–	–	–
		%rd, [%rb]	rd(7:0)←B[rb], rd(31:8)←0	–	–	–	–	–	–	–	–	–
		%rd, [%rb]+	rd(7:0)←B[rb], rd(31:8)←0, rb←rb+1	–	–	–	–	–	–	–	–	–
		%rd, [%sp+imm6]	rd(7:0)←B[sp+imm6], rd(31:8)←0	–	–	–	–	–	–	–	–	–
	xld.ub	%rd, [imm32]	rd(7:0)←B[imm32], rd(31:8)←0	–	–	–	–	–	–	×	○	–
		%rd, [symbol+imm32]	rd(7:0)←B[symbol+imm32], rd(31:8)←0	–	–	–	–	–	–	○	○	–
		%rd, [symbol-imm32]	rd(7:0)←B[symbol-imm32], rd(31:8)←0	–	–	–	–	–	–	×	○	–
		%rd, [%rb+imm32] *1	rd(7:0)←B[rb+imm32], rd(31:8)←0	–	–	↔	↔	↔	↔	×	△	–
		%rd, [%rb+symbol+imm32] *1	rd(7:0)←B[rb+symbol+imm32], rd(31:8)←0	–	–	↔	↔	↔	↔	×	△	–
		%rd, [%rb+symbol-imm32] *1	rd(7:0)←B[rb+symbol-imm32], rd(31:8)←0	–	–	↔	↔	↔	↔	×	○	–
		%rd, [%sp+imm32]	rd(7:0)←B[sp+imm32], rd(31:8)←0	–	–	–	–	–	–	×	×	–

Remarks

∗1) These extended instructins may affect the flags because they use the "add" instruction for expansion according to the condition.

Classification	Mnemonic		Function	Flags						R8	R9	D
	Opcode	Operand		MO	DS	C	V	Z	N			
Signed half word data transfer	ld.h	%rd, %rs	rd(15:0)←rs(15:0), rd(31:16)←rs(15)	–	–	–	–	–	–	–	–	–
		%rd, [%rb]	rd(15:0)←H[rb], rd(31:16)←H[rb](15)	–	–	–	–	–	–	–	–	–
		%rd, [%rb]+	rd(15:0)←H[rb], rd(31:16)←H[rb](15), rb←rb+2	–	–	–	–	–	–	–	–	–
		%rd, [%sp+imm6]	rd(15:0)←H[sp+imm7], rd(31:16)←H[sp+imm7](15); imm7={imm6,0}	–	–	–	–	–	–	–	–	–
		[%rb], %rs	H[rb]←rs(15:0)	–	–	–	–	–	–	–	–	–
		[%rb]+, %rs	H[rb]←rs(15:0), rb←rb+2	–	–	–	–	–	–	–	–	–
		[%sp+imm6], %rs	H[sp+imm7]←rs(15:0); imm7={imm6,0}	–	–	–	–	–	–	–	–	–
	xld.h	%rd, [imm32]	rd(15:0)←H[imm32], rd(31:16)←H[imm32](15)	–	–	–	–	–	–	×	○	–
		%rd, [symbol+imm32]	rd(15:0)←H[symbol+imm32], rd(31:16)←H[symbol+imm32](15)	–	–	–	–	–	–	○	○	–
		%rd, [symbol-imm32]	rd(15:0)←H[symbol-imm32], rd(31:16)←H[symbol-imm32](15)	–	–	–	–	–	–	×	○	–
		%rd, [%rb+imm32] *1	rd(15:0)←H[rb+imm32], rd(31:16)←H[rb+imm32](15)	–	–	↔	↔	↔	↔	×	△	–
		%rd, [%rb+symbol+imm32] *1	rd(15:0)←H[rb+symbol+imm32], rd(31:16)←H[rb+symbol+imm32](15)	–	–	↔	↔	↔	↔	×	△	–
		%rd, [%rb+symbol-imm32] *1	rd(15:0)←H[rb+symbol-imm32], rd(31:16)←H[rb+symbol-imm32](15)	–	–	↔	↔	↔	↔	×	○	–
		%rd, [%sp+imm32]	rd(15:0)←H[sp+imm32], rd(31:16)←H[sp+imm32](15)	–	–	–	–	–	–	×	×	–
		[imm32], %rs	H[imm32]←rs(15:0)	–	–	–	–	–	–	×	○	–
		[symbol+imm32], %rs	H[symbol+imm32]←rs(15:0)	–	–	–	–	–	–	○	○	–
		[symbol-imm32], %rs	H[symbol-imm32]←rs(15:0)	–	–	–	–	–	–	×	○	–
		[%rb+imm32], %rs *1	H[rb+imm32]←rs(15:0)	–	–	↔	↔	↔	↔	×	△	–
		[%rb+symbol+imm32], %rs *1	H[rb+symbol+imm32]←rs(15:0)	–	–	↔	↔	↔	↔	×	△	–
		[%rb+symbol-imm32], %rs *1	H[rb+symbol-imm32]←rs(15:0)	–	–	↔	↔	↔	↔	×	○	–
		[%sp+imm32], %rs	H[sp+imm32]←rs(15:0)	–	–	–	–	–	–	×	×	–
Unsigned half word data transfer	ld.uh	%rd, %rs	rd(15:0)←rs(15:0), rd(31:16)←0	–	–	–	–	–	–	–	–	–
		%rd, [%rb]	rd(15:0)←H[rb], rd(31:16)←0	–	–	–	–	–	–	–	–	–
		%rd, [%rb]+	rd(15:0)←H[rb], rd(31:16)←0, rb←rb+2	–	–	–	–	–	–	–	–	–
		%rd, [%sp+imm6]	rd(15:0)←H[sp+imm7], rd(31:16)←0; imm7={imm6,0}	–	–	–	–	–	–	–	–	–
	xld.uh	%rd, [imm32]	rd(15:0)←H[imm32], rd(31:16)←0	–	–	–	–	–	–	×	○	–
		%rd, [symbol+imm32]	rd(15:0)←H[symbol+imm32], rd(31:16)←0	–	–	–	–	–	–	○	○	–
		%rd, [symbol-imm32]	rd(15:0)←H[symbol-imm32], rd(31:16)←0	–	–	–	–	–	–	×	○	–
		%rd, [%rb+imm32] *1	rd(15:0)←H[rb+imm32], rd(31:16)←0	–	–	↔	↔	↔	↔	×	△	–
		%rd, [%rb+symbol+imm32] *1	rd(15:0)←H[rb+symbol+imm32], rd(31:16)←0	–	–	↔	↔	↔	↔	×	△	–
		%rd, [%rb+symbol-imm32] *1	rd(15:0)←H[rb+symbol-imm32], rd(31:16)←0	–	–	↔	↔	↔	↔	×	○	–
		%rd, [%sp+imm32]	rd(15:0)←H[sp+imm32], rd(31:16)←0	–	–	–	–	–	–	×	×	–

Remarks

∗1) These extended instructins may affect the flags because they use the "add" instruction for expansion according to the condition.

Classification	Mnemonic		Function	Flags						R8	R9	D
	Opcode	Operand		MO	DS	C	V	Z	N			
Word data transfer	ld.w	%rd, %rs	rd←rs	–	–	–	–	–	–	–	–	○
		%sd, %rs	sd←rs	–	–	–	–	–	–	–	–	–
		%rd, %ss	rd←ss	–	–	–	–	–	–	–	–	–
		%rd, sign6	rd(5:0)←sign6(5:0), rd(31:6)←sign6(5)	–	–	–	–	–	–	–	–	○
		%rd, [%rb]	rd←W[rb]	–	–	–	–	–	–	–	–	–
		%rd, [%rb]+	rd←W[rb], rb←rb+4	–	–	–	–	–	–	–	–	–
		%rd, [%sp+imm6]	rd←W[sp+imm8]; imm8={imm6,00}	–	–	–	–	–	–	–	–	–
		[%rb], %rs	W[rb]←rs	–	–	–	–	–	–	–	–	–
		[%rb]+, %rs	W[rb]←rs, rb←rb+4	–	–	–	–	–	–	–	–	–
		[%sp+imm6], %rs	W[sp+imm8]←rs; imm8={imm6,00}	–	–	–	–	–	–	–	–	–
	xld.w	%rd, sign32	rd←sign32	–	–	–	–	–	–	×	×	–
		%rd, symbol+imm32	rd←symbol+imm32	–	–	–	–	–	–	×	×	–
		%rd, symbol-imm32	rd←symbol-imm32	–	–	–	–	–	–	×	×	–
		%rd, [imm32]	rd←W[imm32]	–	–	–	–	–	–	×	○	–
		%rd, [symbol+imm32]	rd←W[symbol+imm32]	–	–	–	–	–	–	○	○	–
		%rd, [symbol-imm32]	rd←W[symbol-imm32]	–	–	–	–	–	–	×	○	–
		%rd, [%rb+imm32] *1	rd←W[rb+imm32]	–	–	↔	↔	↔	↔	×	△	–
		%rd, [%rb+symbol+imm32] *1	rd←W[rb+symbol+imm32]	–	–	↔	↔	↔	↔	×	△	–
		%rd, [%rb+symbol-imm32] *1	rd←W[rb+symbol-imm32]	–	–	↔	↔	↔	↔	×	○	–
		%rd, [%sp+imm32]	rd←W[sp+imm32]	–	–	–	–	–	–	×	×	–
		[imm32], %rs	W[imm32]←rs	–	–	–	–	–	–	×	○	–
		[imm32], %sp	W[imm32]←sp	–	–	–	–	–	–	×	○	–
		[symbol+imm32], %rs	W[symbol+imm32]←rs	–	–	–	–	–	–	○	○	–
		[symbol-imm32], %rs	W[symbol-imm32]←rs	–	–	–	–	–	–	×	○	–
		[symbol+imm32], %sp	W[symbol+imm32]←sp	–	–	–	–	–	–	○	○	–
		[symbol-imm32], %sp	W[symbol-imm32]←sp	–	–	–	–	–	–	×	○	–
		[%rb+imm32], %rs *1	W[rb+imm32]←rs	–	–	↔	↔	↔	↔	×	△	–
		[%rb+imm32], %sp *1	W[rb+imm32]←sp	–	–	↔	↔	↔	↔	×	○	–
		[%rb+symbol+imm32], %rs *1	W[rb+symbol+imm32]←rs	–	–	↔	↔	↔	↔	×	△	–
		[%rb+symbol-imm32], %rs *1	W[rb+symbol-imm32]←rs	–	–	↔	↔	↔	↔	×	○	–
		[%rb+symbol+imm32], %sp *1	W[rb+symbol+imm32]←sp	–	–	↔	↔	↔	↔	×	○	–
		[%rb+symbol-imm32], %sp *1	W[rb+symbol-imm32]←sp	–	–	↔	↔	↔	↔	×	○	–
		[%sp+imm32], %rs	W[sp+imm32]←rs	–	–	–	–	–	–	×	×	–
		[%sp+imm32], %sp	W[sp+imm32]←sp	–	–	–	–	–	–	×	○	–

Remarks

∗1) These extended instructins may affect the flags because they use the "add" instruction for expansion according to the condition.

Classification	Mnemonic		Function	Flags						R8	R9	D
	Opcode	Operand		MO	DS	C	V	Z	N			
Logic operation	and	%rd, %rs	rd←rd & rs	–	–	–	–	↔	↔	–	–	○
		%rd, sign6	rd←rd & sign6(with sign extension)	–	–	–	–	↔	↔	–	–	○
	xand	%rd, %rd, sign32	rd←rd & sign32	–	–	–	–	↔	↔	×	×	–
		%rd, %rs, sign32	rd←rs & sign32	–	–	–	–	↔	↔	×	×	–
	or	%rd, %rs	rd←rd | rs	–	–	–	–	↔	↔	–	–	○
		%rd, sign6	rd←rd | sign6(with sign extension)	–	–	–	–	↔	↔	–	–	○
	xoor	%rd, %rd, sign32	rd←rd | sign32	–	–	–	–	↔	↔	×	×	–
		%rd, %rs, sign32	rd←rs | sign32	–	–	–	–	↔	↔	×	×	–
	xor	%rd, %rs	rd←rd ^ rs	–	–	–	–	↔	↔	–	–	○
		%rd, sign6	rd←rd ^ sign6(with sign extension)	–	–	–	–	↔	↔	–	–	○
	xxor	%rd, %rd, sign32	rd←rd ^ sign32	–	–	–	–	↔	↔	×	×	–
		%rd, %rs, sign32	rd←rs ^ sign32	–	–	–	–	↔	↔	×	×	–
	not	%rd, %rs	rd←!rs	–	–	–	–	↔	↔	–	–	○
		%rd, sign6	rd←!sign6(with sign extension)	–	–	–	–	↔	↔	–	–	○
	xnot	%rd, sign32	rd←!sign32	–	–	–	–	↔	↔	×	×	–
Arithmetic operation	add	%rd, %rs	rd←rd + rs	–	–	↔	↔	↔	↔	–	–	○
		%rd, imm6	rd←rd + imm6(with zero extension)	–	–	↔	↔	↔	↔	–	–	○
		%sp, imm10	sp←sp + imm12(with zero extension); imm12={imm10,00}	–	–	–	–	–	–	–	–	○
	xadd	%rd, %rd, imm32	rd←rd + imm32	–	–	↔	↔	↔	↔	×	×	–
		%sp, %sp, imm32	sp←sp + imm32	–	–	↔	↔	↔	↔	×	○	–
		%rd, %rs, imm32	rd←rs + imm32	–	–	↔	↔	↔	↔	×	×	–
		%rd, %sp, imm32	rd←sp + imm32	–	–	↔	↔	↔	↔	×	×	–
		%rd, %rd, %sp	rd←rd + sp	–	–	↔	↔	↔	↔	×	○	–
		%sp, %sp, %rs	sp←sp + rs	–	–	↔	↔	↔	↔	×	○	–
	adc	%rd, %rs	rd←rd + rs + C	–	–	↔	↔	↔	↔	–	–	○
	sub	%rd, %rs	rd←rd - rs	–	–	↔	↔	↔	↔	–	–	○
		%rd, imm6	rd←rd - imm6(with zero extension)	–	–	↔	↔	↔	↔	–	–	○
		%sp, imm10	sp←sp - imm12(with zero extension); imm12={imm10,00}	–	–	–	–	–	–	–	–	○
	xsub	%rd, %rd, imm32	rd←rd - imm32	–	–	↔	↔	↔	↔	×	×	–
		%sp, %sp, imm32	sp←sp - imm32	–	–	↔	↔	↔	↔	×	○	–
		%rd, %rs, imm32	rd←rs - imm32	–	–	↔	↔	↔	↔	×	×	–
		%rd, %sp, imm32	rd←sp - imm32	–	–	↔	↔	↔	↔	×	×	–
		%rd, %rd, %sp	rd←rd - sp	–	–	↔	↔	↔	↔	×	○	–
		%sp, %sp, %rs	sp←sp - rs	–	–	↔	↔	↔	↔	×	○	–
	sbc	%rd, %rs	rd←rd - rs - C	–	–	↔	↔	↔	↔	–	–	–

Remarks

Classification	Mnemonic		Function	Flags						R8	R9	D
	Opcode	Operand		MO	DS	C	V	Z	N			
Arithmetic operation	cmp	%rd, %rs	rd - rs	–	–	↔	↔	↔	↔	–	–	○
		%rd, sign6	rd - sign6(with sign extension)	–	–	↔	↔	↔	↔	–	–	○
	xcmp	%rd, sign32	rd - sign32	–	–	↔	↔	↔	↔	×	×	–
		%rd, %sp	rd - sp	–	–	↔	↔	↔	↔	×	○	–
		%sp, %rs	sp - rs	–	–	↔	↔	↔	↔	×	○	–
	mlt.h	%rd, %rs	alr←rd(15:0) × rs(15:0); calculated with sign	–	–	–	–	–	–	–	–	○
	mltu.h	%rd, %rs	alr←rd(15:0) × rs(15:0); calculated without sign	–	–	–	–	–	–	–	–	○
	mlt.w	%rd, %rs	{ahr, alr}←rd × rs; calculated with sign	–	–	–	–	–	–	–	–	–
	mltu.w	%rd, %rs	{ahr, alr}←rd × rs; calculated without sign	–	–	–	–	–	–	–	–	–
	div0s	%rs	Setup for signed division; alr = dividend, rs = divisor	–	↔	–	–	–	↔	–	–	–
	div0u	%rs	Setup for unsigned division; alr = dividend, rs = divisor	–	0	–	–	–	0	–	–	–
	div1	%rs	Step division for one bit (∗1); alr←quotient, ahr←remainder (unsigned)	–	–	–	–	–	–	–	–	–
	div2s	%rs	Correction step 1 for signed division (∗2)	–	–	–	–	–	–	–	–	–
	div3s		Correction step 2 for signed division (∗2); alr←quotient, ahr←remainder	–	–	–	–	–	–	–	–	–
Shift & rotation	*srl*	*%rd, imm4*	Logical shift to right imm4 bits; imm4=0–8, zero enters to MSB	–	–	–	–	↔	↔	–	–	○
		%rd, %rs	Logical shift to right rs bits; rs=0–8, zero enters to MSB	–	–	–	–	↔	↔	–	–	○
	xsrl	%rd, imm5	Logical shift to right imm5 bits; imm5=0–31, zero enters to MSB	–	–	–	–	↔	↔	×	×	–
		%rd, %rs	Logical shift to right rs bits; rs=0–31, zero enters to MSB	–	–	↔	↔	↔	↔	×	○	–
	sll	*%rd, imm4*	Logical shift to left imm4 bits; imm4=0–8, zero enters to LSB	–	–	–	–	↔	↔	–	–	○
		%rd, %rs	Logical shift to left rs bits; rs=0–8, zero enters to LSB	–	–	–	–	↔	↔	–	–	○
	xsll	%rd, imm5	Logical shift to left imm5 bits; imm5=0–31, zero enters to LSB	–	–	–	–	↔	↔	×	×	–
		%rd, %rs	Logical shift to left rs bits; rs=0–31, zero enters to LSB	–	–	↔	↔	↔	↔	×	○	–
	sra	*%rd, imm4*	Arithmetical shift to right imm4 bits; imm4=0–8, sign copied to MSB	–	–	–	–	↔	↔	–	–	○
		%rd, %rs	Arithmetical shift to right rs bits; rs=0–8, sign copied to MSB	–	–	–	–	↔	↔	–	–	○
	xsra	%rd, imm5	Arithmetical shift to right imm5 bits; imm5=0–31, sign copied to MSB	–	–	–	–	↔	↔	×	×	–
		%rd, %rs	Arithmetical shift to right rs bits; rs=0–31, sign copied to MSB	–	–	↔	↔	↔	↔	×	○	–
	sla	*%rd, imm4*	Arithmetical shift to left imm4 bits; imm4=0–8, zero enters to LSB	–	–	–	–	↔	↔	–	–	○
		%rd, %rs	Arithmetical shift to left rs bits; rs=0–8, zero enters to LSB	–	–	–	–	↔	↔	–	–	○
	xsla	%rd, imm5	Arithmetical shift to left imm5 bits; imm5=0–31, zero enters to LSB	–	–	–	–	↔	↔	×	×	–
		%rd, %rs	Arithmetical shift to left rs bits; rs=0–31, zero enters to LSB	–	–	↔	↔	↔	↔	×	○	–
	rr	*%rd, imm4*	Rotation to right imm4 bits; imm4=0–8, LSB goes to MSB	–	–	–	–	↔	↔	–	–	○
		%rd, %rs	Rotation to right rs bits; rs=0–8, LSB goes to MSB	–	–	–	–	↔	↔	–	–	○
	xrr	%rd, imm5	Rotation to right imm5 bits; imm5=0–31, LSB goes to MSB	–	–	–	–	↔	↔	×	×	–
		%rd, %rs	Rotation to right rs bits; rs=0–31, LSB goes to MSB	–	–	↔	↔	↔	↔	×	○	–

Remarks

∗1) The div1 instruction must be executed 32 times when performing 32-bit data ÷ 32-bit data. In unsigned division, the division result is loaded to the alr and ahr registers.

∗2) It is not necessary to execute the div2s and div3s instructions for unsigned division.

Classification	Mnemonic		Function	Flags						R8	R9	D
	Opcode	Operand		MO	DS	C	V	Z	N			
Shift & rotation	*rl*	*%rd, imm4*	Rotation to left imm4 bits; imm4=0–8, MSB goes to LSB	–	–	–	–	↔	↔	–	–	○
		%rd, %rs	Rotation to left rs bits; rs=0–8, MSB goes to LSB	–	–	–	–	↔	↔	–	–	○
	xrl	%rd, imm5	Rotation to left imm5 bits; imm5=0–31, MSB goes to LSB	–	–	–	–	↔	↔	×	×	–
		%rd, %rs	Rotation to left rs bits; rs=0–31, MSB goes to LSB	–	–	↔	↔	↔	↔	×	○	–
Bit operation	*btst*	*[%rb], imm3*	Z flag←1 if B[rb](imm3)=0	–	–	–	–	↔	–	–	–	–
	xbtst	[imm32], imm3	Z flag←1 if B[imm32](imm3)=0	–	–	–	–	↔	–	×	○	–
		[symbol+imm32], imm3	Z flag←1 if B[symbol+imm32](imm3)=0	–	–	–	–	↔	–	○	○	–
		[symbol-imm32], imm3	Z flag←1 if B[symbol-imm32](imm3)=0	–	–	–	–	↔	–	×	○	–
		[%rb+imm32], imm3 *1	Z flag←1 if B[rb+imm32](imm3)=0	–	–	↔	↔	↔	↔	×	○	–
		[%rb+symbol+imm32], imm3 *1	Z flag←1 if B[rb+symbol+imm32](imm3)=0	–	–	↔	↔	↔	↔	×	○	–
		[%rb+symbol-imm32], imm3 *1	Z flag←1 if B[rb+symbol-imm32](imm3)=0	–	–	↔	↔	↔	↔	×	○	–
		[%sp+imm32], imm3 *1	Z flag←1 if B[sp+imm32](imm3)=0	–	–	↔	↔	↔	↔	×	○	–
	bclr	*[%rb], imm3*	B[imm32](imm3)←0	–	–	–	–	–	–	–	–	–
	xbclr	[imm32], imm3	B[imm32](imm3)←0	–	–	–	–	–	–	×	○	–
		[symbol+imm32], imm3	B[symbol+imm32](imm3)←0	–	–	–	–	–	–	○	○	–
		[symbol-imm32], imm3	B[symbol-imm32](imm3)←0	–	–	–	–	–	–	×	○	–
		[%rb+imm32], imm3 *1	B[rb+imm32](imm3)←0	–	–	↔	↔	↔	↔	×	○	–
		[%rb+symbol+imm32], imm3 *1	B[rb+symbol+imm32](imm3)←0	–	–	↔	↔	↔	↔	×	○	–
		[%rb+symbol-imm32], imm3 *1	B[rb+symbol-imm32](imm3)←0	–	–	↔	↔	↔	↔	×	○	–
		[%sp+imm32], imm3 *1	B[sp+imm32](imm3)←0	–	–	↔	↔	↔	↔	×	○	–
	bset	*[%rb], imm3*	B[rb](imm3)←1	–	–	–	–	–	–	–	–	–
	xbset	[imm32], imm3	B[imm32](imm3)←1	–	–	–	–	–	–	×	○	–
		[symbol+imm32], imm3	B[symbol+imm32](imm3)←1	–	–	–	–	–	–	○	○	–
		[symbol-imm32], imm3	B[symbol-imm32](imm3)←1	–	–	–	–	–	–	×	○	–
		[%rb+imm32], imm3 *1	B[rb+imm32](imm3)←1	–	–	↔	↔	↔	↔	×	○	–
		[%rb+symbol+imm32], imm3 *1	B[rb+symbol+imm32](imm3)←1	–	–	↔	↔	↔	↔	×	○	–
		[%rb+symbol-imm32], imm3 *1	B[rb+symbol-imm32](imm3)←1	–	–	↔	↔	↔	↔	×	○	–
		[%sp+imm32], imm3 *1	B[sp+imm32](imm3)←1	–	–	↔	↔	↔	↔	×	○	–
	bnot	*[%rb], imm3*	B[rb](imm3)←!B[rb](imm3)	–	–	–	–	–	–	–	–	–
	xbnot	[imm32], imm3	B[imm32](imm3)←!B[imm32](imm3)	–	–	–	–	–	–	×	○	–
		[symbol+imm32], imm3	B[symbol+imm32](imm3)←!B[symbol+imm32](imm3)	–	–	–	–	–	–	○	○	–
		[symbol-imm32], imm3	B[symbol-imm32](imm3)←!B[symbol-imm32](imm3)	–	–	–	–	–	–	×	○	–
		[%rb+imm32], imm3 *1	B[rb+imm32](imm3)←!B[rb+imm32](imm3)	–	–	↔	↔	↔	↔	×	○	–
		[%rb+symbol+imm32], imm3 *1	B[rb+symbol+imm32](imm3)←!B[rb+symbol+imm32](imm3)	–	–	↔	↔	↔	↔	×	○	–
		[%rb+symbol-imm32], imm3 *1	B[rb+symbol-imm32](imm3)←!B[rb+symbol-imm32](imm3)	–	–	↔	↔	↔	↔	×	○	–
		[%sp+imm32], imm3 *1	B[sp+imm32](imm3)←!B[sp+imm32](imm3)	–	–	↔	↔	↔	↔	×	○	–

Remarks

∗1) These extended instructins may affect the flags because they use the "add" instruction for expansion according to the condition.

Classification	Mnemonic		Function	Flags						R8	R9	D
	Opcode	Operand		MO	DS	C	V	Z	N			
Branch	*jrgt* *jrgt.d*	*sign8*	pc←pc+sign9 if !Z&!(N^V) is true; sign9={sign8,0}	–	–	–	–	–	–	–	–	–
	xjrgt	label+imm32	pc←label+imm32 if !Z&!(N^V) is true	–	–	–	–	–	–	×	×	–
	xjrgt.d	sign32	pc←pc+sign32 if !Z&!(N^V) is true	–	–	–	–	–	–	×	×	–
	jrge *irge.d*	*sign8*	pc←pc+sign9 if !(N^V) is true; sign9={sign8,0}	–	–	–	–	–	–	–	–	–
	xjrge	label+imm32	pc←label+imm32 if !(N^V) is true	–	–	–	–	–	–	×	×	–
	xirge.d	sign32	pc←pc+sign32 if !(N^V) is true	–	–	–	–	–	–	×	×	–
	jrlt *jrlt.d*	*sign8*	pc←pc+sign9 if N^V is true; sign9={sign8,0}	–	–	–	–	–	–	–	–	–
	xjrlt	label+imm32	pc←label+imm32 if N^V is true	–	–	–	–	–	–	×	×	–
	xjrlt.d	sign32	pc←pc+sign32 if N^V is true	–	–	–	–	–	–	×	×	–
	jrle *jrle.d*	*sign8*	pc←pc+sign9 if Z | (N^V) is true; sign9={sign8,0}	–	–	–	–	–	–	–	–	–
	xjrle	label+imm32	pc←label+imm32 if Z | (N^V) is true	–	–	–	–	–	–	×	×	–
	xjrle.d	sign32	pc←pc+sign32 if Z | (N^V) is true	–	–	–	–	–	–	×	×	–
	jrugt *jrugt.d*	*sign8*	pc←pc+sign9 if !Z&!C is true; sign9={sign8,0}	–	–	–	–	–	–	–	–	–
	xjrugt	label+imm32	pc←label+imm32 if !Z&!C is true	–	–	–	–	–	–	×	×	–
	xjrugt.d	sign32	pc←pc+sign32 if !Z&!C is true	–	–	–	–	–	–	×	×	–
	jruge *jruge.d*	*sign8*	pc←pc+sign9 if !C is true; sign9={sign8,0}	–	–	–	–	–	–	–	–	–
	xjruge	label+imm32	pc←label+imm32 if !C is true	–	–	–	–	–	–	×	×	–
	xjruge.d	sign32	pc←pc+sign32 if !C is true	–	–	–	–	–	–	×	×	–
	jrult *jrult.d*	*sign8*	pc←pc+sign9 if C is true; sign9={sign8,0}	–	–	–	–	–	–	–	–	–
	xjrult	label+imm32	pc←label+imm32 if C is true	–	–	–	–	–	–	×	×	–
	xjrult.d	sign32	pc←pc+sign32 if C is true	–	–	–	–	–	–	×	×	–
	jrule *jrule.d*	*sign8*	pc←pc+sign9 if Z | C is true; sign9={sign8,0}	–	–	–	–	–	–	–	–	–
	xjrule	label+imm32	pc←label+imm32 if Z | C is true	–	–	–	–	–	–	×	×	–
	xjrule.d	sign32	pc←pc+sign32 if Z | C is true	–	–	–	–	–	–	×	×	–
	jreq *jreq.d*	*sign8*	pc←pc+sign9 if Z is true; sign9={sign8,0}	–	–	–	–	–	–	–	–	–
	xjreq	label+imm32	pc←label+imm32 if Z is true	–	–	–	–	–	–	×	×	–
	xjreq.d	sign32	pc←pc+sign32 if Z is true	–	–	–	–	–	–	×	×	–

Remarks

Classification	Mnemonic		Function	Flags						R8	R9	D
	Opcode	Operand		MO	DS	C	V	Z	N			
Branch	*jrne* *jrne.d*	*sign8*	pc←pc+sign9 if !Z is true; sign9={sign8,0}	–	–	–	–	–	–	–	–	–
	xjrne	label+imm32	pc←label+imm32 if !Z is true	–	–	–	–	–	–	×	×	–
	xjrne.d	sign32	pc←pc+sign32 if !Z is true	–	–	–	–	–	–	×	×	–
	call	*sign8*	sp←sp-4, W[sp]←pc+2, pc←pc+sign9; sign9={sign8,0}	–	–	–	–	–	–	–	–	–
	call.d	%rb	sp←sp-4, W[sp]←pc+2, pc←rb	–	–	–	–	–	–	–	–	–
	xcall	label+imm32	sp←sp-4, W[sp]←pc+2, pc←label+imm32	–	–	–	–	–	–	×	×	–
	xcall.d	sign32	sp←sp-4, W[sp]←pc+2, pc←pc+sign32	–	–	–	–	–	–	×	×	–
	jp	*sign8*	pc←pc+sign9; sign9={sign8,0}	–	–	–	–	–	–	–	–	–
	jp.d	%rb	pc←rb	–	–	–	–	–	–	–	–	–
	xjp	abel+imm32	pc←pc+label+imm32	–	–	–	–	–	–	×	×	–
	xjp.d	sign32	pc←pc+sign32	–	–	–	–	–	–	×	×	–
	ret ret.d		pc←W[sp], sp←sp+4	–	–	–	–	–	–	–	–	–
	reti		psr←W[sp], sp←sp+4, pc←W[sp], sp←sp+4	↔	↔	↔	↔	↔	↔	–	–	–
	retd		Returns from debugging routine (for ICE software)	–	–	–	–	–	–	–	–	–
	int	imm2	sp←sp-4, W[sp]←pc+2, sp←sp-4, W[sp]←psr, pc←software exception vector	–	–	–	–	–	–	–	–	–
	brk		Interrupt for debugging (for ICE software)	–	–	–	–	–	–	–	–	–
Extension	*ext*	*imm13*	Extends the immediate or operand of the following instruction.	–	–	–	–	–	–	–	–	–
Push & pop	pushn	%rs	Repeats "sp←sp-4, W[sp]←rn"; rn=rs to r0	–	–	–	–	–	–	–	–	–
	popn	%rd	Repeats "rn←W[sp], sp←sp+4"; rn=r0 to rd	–	–	–	–	–	–	–	–	–
MAC	mac	%rs	Repeats "{ahr, alr}←{ahr, alr} + H[<rs+1>]+ × H[<rs+2>]+" rs times (∗1)	↔	–	–	–	–	–	–	–	–
System control	nop		No operation; pc←pc+2	–	–	–	–	–	–	–	–	–
	halt		Sets Halt mode	–	–	–	–	–	–	–	–	–
	slp		Sets Sleep mode	–	–	–	–	–	–	–	–	–
Others	scan0	%rd, %rs	Scan 0 bit from 1 byte from MSB in rs, rd←offset from MSB of found bit	–	–	↔	0	↔	0	–	–	○
	scan1	%rd, %rs	Scan 1 bit from 1 byte from MSB in rs, rd←offset from MSB of found bit	–	–	↔	0	↔	0	–	–	○
	swap	%rd, %rs	rd(31:24)←rs(7:0), rd(23:16)←rs(15:8), rd(15:8)←rs(23:16), rd(7:0)←rs(31:24)	–	–	–	–	–	–	–	–	○
	mirror	%rd, %rs	rd(31:24)←rs(24:31), rd(23:16)←rs(16:23), rd(15:8)←rs(8:15), rd(7:0)←rs(0:7)	–	–	–	–	–	–	–	–	○

Remarks

∗1) <rs+1>, <rs+2>: contents of the registers that follow rs. (eg. rs=r0: <rs+1>=r1, <rs+2>=r2; rs=r15: <rs+1>=r0, <rs+2>=r1); They are incremented (+2) after each operation. The mac instruction can be executed only in the models that have an optional multiplier.

Extended instruction		Expansion format				
Opcode	Operand	Condition 1	Condition 2	Condition 3	Condition 4	Condition 5
xld.b **xld.ub** **xld.h** **xld.uh** **xld.w**	**%rd, [imm32]**	imm32≤0x1f	0x1f<imm32≤0x3ffff	0x3ffff<imm32	–	–
	[imm32], %rs *1 <when gp is not used> eg.) xld.w %rd,[imm32]	ld.w %r9,imm32(5:0) ld.w %rd,[%r9]	ext imm32(18:6) ld.w %r9,imm32(5:0) ld.w %rd,[%r9]	ext imm32(31:19) ext imm32(18:6) ld.w %r9,imm32(5:0) ld.w %rd,[%r9]		
	%rd, [imm32]	sign32=0x0	0x0<sign32≤0x1fff	0x1fff<sign32≤0x3ffffff	0x3ffffff<sign32	imm32<gp
	[imm32], %rs *1 <when gp is used> eg.) xld.w %rd,[imm32] (sign32=-gp+imm32)	ld.w %rd,[%r8]	ext sign32(12:0) ld.w %rd,[%r8]	ext sign32(25:13) ext sign32(12:0) ld.w %rd,[%r8]	ext imm32(31:19) ext imm32(18:6) ld.w %r9,imm32(5:0) ld.w %rd, [%r9]	Expanded into the format without gp specification according to the imm32 value.
	%rd, [symbol+imm32]	symbol+imm32≤0x1f	0x1f<symbol+imm32≤0x3ffff	0x3ffff<symbol+imm32 *2	–	–
	[symbol+imm32], %rs *1 <when gp is not used> eg.) xld.w %rd,[symbol+imm32]	ld.w %r9,symbol+imm32@l ld.w %rd,[%r9]	ext symbol+imm32@m ld.w %r9,symbol+imm32@l ld.w %rd,[%r9]	ext symbol+imm32@h ext symbol+imm32@m ld.w %r9,symbol+imm32@l ld.w %rd, [%r9]		
	%rd, [symbol+imm32]	symbol+sign32=0x0	0x0<symbol+sign32≤0x1fff	0x1fff<symbol+sign32≤0x3ffffff *2	0x3ffffff<symbol+sign32	symbol+imm32<gp
	[symbol+imm32], %rs *1 <when gp is used> eg.) xld.w %rd,[symbol+imm32] (sign32=-gp+imm32)	ld.w %rd,[%r8]	ext symbol+sign32@al ld.w %rd,[%r8]	ext symbol+sign32@ah ext symbol+sign32@al ld.w %rd,[%r8]	ext symbol+imm32@h ext symbol+imm32@m ld.w %r9,symbol+imm32@l ld.w %rd, [%r9]	Expanded into the format without gp specification according to the symbol+imm32 value.
	%rd, [symbol-imm32]	Fixed	–	–	–	–
	[symbol-imm32], %rs *1 eg.) xld.w %rd,[symbol-imm32]	ext symbol-imm32@h ext symbol-imm32@m ld.w %r9,symbol-imm32@l ld.w %rd, [%r9]				
	%rd, [%rb+imm32]	imm32=0x0	0x0<imm32≤0x1fff	0x1fff<imm32≤0x3ffffff	0x3ffffff<imm32	–
	[%rb+imm32], %rs *1 eg.) xld.w %rd,[%rb+imm32]	ld.w %rd,[%rb]	ext imm32(12:0) ld.w %rd,[%rb]	ext imm32(25:13) ext imm32(12:0) ld.w %rd,[%rb]	ext imm32(31:19) ext imm32(18:6) ld.w %r9,imm32(5:0) add %r9,%rb ld.w %rd,[%r9]	
	%rd, [%rb+symbol+imm32]	symbol+imm32=0x0	0x0<symbol+imm32≤0x1fff	0x1fff<symbol+imm32≤0x3ffffff	0x3ffffff<symbol+imm32 *2	–
	[%rb+symbol+imm32], %rs *1 eg.) xld.w %rd,[%rb+symbol+imm32]	ld.w %rd,[%rb]	ext symbol+imm32al ld.w %rd,[%rb]	ext symbol+imm32@ah ext symbol+imm32@al ld.w %rd,[%rb]	ext symbol+imm32@h ext symbol+imm32@m ld.w %r9,symbol+imm32@l add %r9,%rb ld.w %rd, [%r9]	

Remarks

*1) These operands are available only for xld.b, xld.h, xld.w instructions. *2) This extension format is used when the sysmpl is undefined.

Expansion Format of Extended Instructions (2)

Extended instruction		Expansion format				
Opcode	Operand	Condition 1	Condition 2	Condition 3	Condition 4	Condition 5
xld.b **xld.ub** **xld.h** **xld.uh** **xld.w**	**%rd, [%rb+symbol-imm32]** **[%rb+symbol-imm32], %rs** *1 eg.) xld.w %rd,[%rb+symbol-imm32]	Fixed ext symbol-imm32@h ext symbol-imm32@m ld.w %r9,symbol-imm32@l add %r9,%rb ld.w %rd, [%r9]	–	–	–	–
xld.b **xld.ub**	**%rd, [%sp+imm32]** **[%sp+imm32], %rs** *1 eg.) xld.b %rd,[%sp+imm32]	imm32≤0x3f ld.b %rd,[%sp+imm32(5:0)]	0x3f<imm32≤0x7ffff ext imm32(18:6) ld.b %rd,[%sp+imm32(5:0)]	0x7ffff<imm32 ext imm32(31:19) ext imm32(18:6) ld.b %rd,[%sp+imm32(5:0)]	–	–
xld.h **xld.uh**	**%rd, [%sp+imm32]** **[%sp+imm32], %rs** *1 eg.) xld.h %rd,[%sp+imm32]	imm32≤0x7f ld.h %rd,[%sp+imm32(6:1)]	0x7f<imm32≤0x7ffff ext imm32(18:6) ld.h %rd,[%sp+imm32(5:0)]	0x7ffff<imm32 ext imm32(31:19) ext imm32(18:6) ld.h %rd,[%sp+imm32(5:0)]	–	–
xld.w	**%rd, [%sp+imm32]** **[%sp+imm32], %rs** *1 eg.) xld.w %rd,[%sp+imm32]	imm32≤0xff ld.w %rd,[%sp+imm32(7:2)]	0xff<imm32≤0x7ffff ext imm32(18:6) ld.w %rd,[%sp+imm32(5:0)]	0x7ffff<imm32 ext imm32(31:19) ext imm32(18:6) ld.w %rd,[%sp+imm32(5:0)]	–	–
	[imm32], %sp <when gp is not used> eg.) xld.w [imm32],%sp	imm32≤0x1f ld.w %r9,%sp pushn %r0 ld.w %r0,imm32(5:0) ld.w [%r0],%r9 popn %r0	0x1f<imm32≤0x3ffff ld.w %r9,%sp pushn %r0 ext imm32(18:6) ld.w %r0,imm32(5:0) ld.w [%r0],%r9 popn %r0	0x3ffff<imm32 ld.w %r9,%sp pushn %r0 ext imm32(31:19) ext imm32(18:6) ld.w %r0,imm32(5:0) ld.w [%r0],%r9 popn %r0	–	–
	[imm32], %sp <when gp is used> eg.) xld.w [imm32],%sp (sign32=-gp+imm32)	sign32=0x0 ld.w %r9,%sp ld.w [%r8],%r9	0x0<sign32≤0x1fff ld.w %r9,%sp ext sign32(12:0) ld.w [%r8],%r9	0x1fff<sign32≤0x3ffffff ld.w %r9,%sp ext sign32(25:13) ext sign32(12:0) ld.w [%r8],%r9	0x3ffffff<sign32 ld.w %r9,%sp pushn %r0 ext imm32(31:19) ext imm32(18:6) ld.w %r0,imm32(5:0) ld.w [%r0],%r9 popn %r0	imm32<gp Expanded into the format without gp specification according to the imm32 value.

Remarks

*1) These operands are available only for xld.b, xld.h, xld.w instructions.

Expansion Format of Extended Instructions (3)

Extended instruction		Expansion format				
Opcode	Operand	Condition 1	Condition 2	Condition 3	Condition 4	Condition 5
xld.w	**[symbol+imm32], %sp**	symbol+imm32≤0x1f	0x1f<symbol+imm32≤0x3ffff	0x3ffff<symbol+imm32 *1	–	–
	<when gp is not used> eg.) xld.w [symbol+imm32],%sp	ld.w %r9,%sp pushn %r0 ld.w %r0,symbol+imm32@l ld.w [%r0],%r9 popn %r0	ld.w %r9,%sp pushn %r0 ext symbol+imm32@m ld.w %r0,symbol+imm32@l ld.w [%r0],%r9 popn %r0	ld.w %r9,%sp pushn %r0 ext symbol+imm32@h ext symbol+imm32@m ld.w %r0,symbol+imm32@l ld.w [%r0],%r9 popn %r0		
	[symbol+imm32], %sp	symbol+sign32=0x0	0x0<symbol+sign32≤0x1fff	0x1fff<symbol+sign32≤0x3ffffff	0x3ffffff<symbol+sign32 *1	symbol+imm32<gp
	<when gp is used> eg.) xld.w [symbol+imm32],%sp (sign32=-gp+imm32)	ld.w %r9,%sp ld.w [%r8],%r9	ld.w %r9,%sp ext symbol+sign32@al ld.w [%r8],%r9	ld.w %r9,%sp ext symbol+sign32@ah ext symbol+sign32@al ld.w [%r8],%r9	ld.w %r9,%sp pushn %r0 ext symbol+imm32@h ext symbol+imm32@m ld.w %r0,symbol+imm32@l ld.w [%r0],%r9 popn %r0	Expanded into the format without gp specification according to the symbol+imm32 value.
	[symbol-imm32], %sp	Fixed	–	–	–	–
	eg.) xld.w [symbol-imm32],%sp	ld.w %r9,%sp pushn %r0 ext symbol-imm32@h ext symbol-imm32@m ld.w %r0,symbol-imm32@l ld.w [%r0],%r9 popn %r0				
	[%rb+imm32], %sp	imm32=0x0	0x0<imm32≤0x1fff	0x1fff<imm32≤0x3ffffff	0x3ffffff<imm32, %rb≠%r0	0x3ffffff<imm32, %rb=%r0
	eg.) xld.w [%rb+imm32],%sp	ld.w %r9,%sp ld.w [%rb],%r9	ld.w %r9,%sp ext imm32(12:0) ld.w [%rb],%r9	ld.w %r9,%sp ext imm32(25:13) ext imm32(12:0) ld.w [%rb],%r9	ld.w %r9,%sp pushn %r0 ext imm32(31:19) ext imm32(18:6) ld.w %r0,imm32(5:0) add %r0,%rb ld.w [%r0],%r9 popn %r0	ld.w %r9,%sp pushn %r1 ext imm32(31:19) ext imm32(18:6) ld.w %r1,imm32(5:0) add %r1,%rb ld.w [%r1],%r9 popn %r1

Remarks

*1) This extension format is used when the sysmpl is undefined.

Expansion Format of Extended Instructions (4)

Extended instruction		Expansion format				
Opcode	Operand	Condition 1	Condition 2	Condition 3	Condition 4	Condition 5
xld.w	**[%rb+symbol+imm32], %sp**	symbol+imm32=0x0	0x0<symbol+imm32≤0x1fff	0x1fff<symbol+imm32≤0x3fffffff	0x3fffffff<symbol+imm32,%rb≠%r0 *1	0x3fffffff<symbol+imm32,%rb=%r0 *2
	eg.) xld.w [%rb+symbol+imm32],%sp	ld.w %r9,%sp ld.w [%rb],%r9	ld.w %r9,%sp ext symbol+imm32al ld.w [%rb],%r9	ld.w %r9,%sp ext symbol+imm32@ah ext symbol+imm32@al ld.w [%rb],%r9	ld.w %r9,%sp pushn %r0 ext symbol+imm32@h ext symbol+imm32@m ld.w %r0,symbol+imm32@l add %r0,%rb ld.w [%r0],%r9 popn %r0	ld.w %r9,%sp pushn %r1 ext symbol+imm32@h ext symbol+imm32@m ld.w %r1,symbol+imm32@l add %r1,%rb ld.w [%r1],%r9 popn %r1
	[%rb+symbol-imm32], %sp	%rb≠%r0	%rb=%r0	–	–	–
	eg.) xld.w [%rb+symbol-imm32],%sp	ld.w %r9,%sp pushn %r0 ext symbol-imm32@h ext symbol-imm32@m ld.w %r0,symbol-imm32@l add %r0,%rb ld.w [%r0],%r9 popn %r0	ld.w %r9,%sp pushn %r1 ext symbol-imm32@h ext symbol-imm32@m ld.w %r1,symbol-imm32@l add %r1,%rb ld.w [%r1],%r9 popn %r1			
	[%sp+imm32], %sp	imm32≤0xff	0xff<imm32≤0x7ffff	0x7ffff<imm32	–	–
	eg.) xld.w [%sp+imm32],%sp	ld.w %r9,%sp ld.w [%sp+imm32(7:2)],%r9	ld.w %r9,%sp ext imm32(18:6) ld.w [%sp+imm32(5:0)],%r9	ld.w %r9,%sp ext imm32(31:19) ext imm32(18:6) ld.w [%sp+imm32(5:0)],%r9		
	%rd, sign32	-32≤sign32≤31	-262144≤sign32<-32 or 31<sign32≤262143	sign32<-262144 or 262143<sign32	–	–
	eg.) xld.w %rd,sign32	ld.w %rd,sign32(5:0)	ext sign32(18:6) ld.w %rd,sign32(5:0)	ext sign32(31:19) ext sign32(18:6) ld.w %rd,sign32(5:0)		
	%rd, symbol+imm32	symbol+imm32≤0x1f	0x1f<symbol+imm32≤0x3ffff	0x3ffff<symbol+imm32 *3	–	–
	eg.) xld.w %rd,sign32	ld.w %rd,symbol+imm32	ext symbol+imm32@m ld.w %rd,symbol+imm32@l	ext symbol+imm32@h ext symbol+imm32@m ld.w %rd,symbol+imm32@l		
	%rd, symbol-imm32	Fixed	–	–	–	–
	eg.) xld.w %rd,sign32	ext symbol-imm32@h ext symbol-imm32@m ld.w %rd,symbol-imm32@l				

Remarks

∗1) This extension format is used when the sysmpl is undefined and %rb≠%r0. ∗2) This extension format is used when the sysmpl is undefined and %rb=%r0.

∗3) This extension format is used when the sysmpl is undefined.

Expansion Format of Extended Instructions (5)

Extended instruction		Expansion format				
Opcode	Operand	Condition 1	Condition 2	Condition 3	Condition 4	Condition 5
xand **xoor** **xxor**	**%rd, %rd, sign32** eg.) xand %rd,%rd,sign32	-32≤sign32≤31 and %rd,sign32(5:0)	-262144≤sign32<-32 or 31<sign32≤262143 ext sign32(18:6) and %rd,sign32(5:0)	sign32<-262144 or 262143<sign32 ext sign32(31:19) ext sign32(18:6) and %rd,sign32(5:0)	–	–
	%rd, %rs, sign32 eg.) xand %rd,%rs,sign32	0x0≤sign32≤0x1fff (within 13 bits) ext sign32(12:0) and %rd,%rs	0x1fff<sign32≤0x3fffffff (within 26 bits) ext sign32(25:13) ext sign32(12:0) and %rd,%rs	0x3fffffff<sign32<0xfffc0000 (26 bits<sign32<-262144) ld.w %rd,%rs ext sign32(31:19) ext sign32(18:6) and %rd,sign32(5:0)	0xfffc0000≤sign32<0xfffffe0 (-262144≤sign32<-32) ld.w %rd,%rs ext sign32(18:6) and %rd,sign32(5:0)	0xfffffe0≤sign32≤0xffffffff (-32≤sign32≤-1) ld.w %rd,%rs and %rd,sign32(5:0)
xnot	**%rd, sign32** eg.) xnot %rd,%rd,sign32	-32≤sign32≤31 not %rd,sign32(5:0)	-262144≤sign32<-32 or 31<sign32≤262143 ext sign32(18:6) not %rd,sign32(5:0)	sign32<-262144 or 262143<sign32 ext sign32(31:19) ext sign32(18:6) not %rd,sign32(5:0)	–	–
xadd **xsub**	**%rd, %rd, imm32** eg.) xadd %rd,%rd,imm32	imm32≤0x3f add %rd,imm32(5:0)	0x3f<imm32≤0x7ffff ext imm32(18:6) add %rd,imm32(5:0)	0x7ffff<imm32 ext imm32(31:19) ext imm32(18:6) add %rd,imm32(5:0)	–	–
	%rd, %rs, imm32 eg.) xadd %rd,%rs,imm32	imm32≤0x1fff ext imm32(12:0) add %rd,%rs	0x1fff<imm32≤0x3fffffff ext imm32(25:13) ext imm32(12:0) add %rd,%rs	0x3fffffff<imm32 ld.w %rd,%rs ext imm32(31:19) ext imm32(18:6) add %rd,imm32(5:0)	–	–
	%rd, %sp, imm32 eg.) xadd %rd,%sp,imm32	imm32≤0x3f ld.w %rd,%sp add %rd,imm32(5:0)	0x3f<imm32≤0x7ffff ld.w %rd,%sp ext imm32(18:6) add %rd,imm32(5:0)	0x7ffff<imm32 ld.w %rd,%sp ext imm32(31:19) ext imm32(18:6) add %rd,imm32(5:0)	–	–
	%sp, %sp, imm32 eg.) xadd %rd,%sp,imm32	imm32≤0xfff add %sp,imm32(11:2)	0xfff<imm32≤0x7ffff ld.w %r9,%sp ext imm32(18:6) add %r9,imm32(5:0) ld.w %sp,%r9	0x7ffff<imm32 ld.w %r9,%sp ext imm32(31:19) ext imm32(18:6) add %r9,imm32(5:0) ld.w %sp,%r9	–	–

Remarks

Expansion Format of Extended Instructions (6)

Extended instruction		Expansion format				
Opcode	Operand	Condition 1	Condition 2	Condition 3	Condition 4	Condition 5
xadd **xsub**	**%rd, %rd, %sp**	Fixed	–	–	–	–
	eg.) xadd %rd,%rd,%sp	ld.w %r9,%sp add %rd,%r9				
	%sp, %sp, %rs	Fixed	–	–	–	–
	eg.) xadd %rd,%rs,imm32	ld.w %r9,%sp add %r9,%rs ld.w %sp,%r9				
xcmp	**%rd, sign32**	-32≤sign32≤31	-262144≤sign32<-32 or 31<sign32≤262143	sign32<-262144 or 262143<sign32	–	–
	eg.) xcmp %rd,sign32	cmp %rd,sign32(5:0)	ext sign32(18:6) cmp %rd,sign32(5:0)	ext sign32(31:19) ext sign32(18:6) cmp %rd,sign32(5:0)		
	%rd, %sp	Fixed	–	–	–	–
	eg.) xcmp %rd,%sp	ld.w %r9,%sp cmp %rd,%r9				
	%sp, %rs	Fixed	–	–	–	–
	eg.) xcmp %rd,%sp	ld.w %r9,%sp cmp %r9,%rs				
xsrl **xsll** **xsra** **xsla** **xrr** **xrl**	**%rd, %rs**	Fixed	–	–	–	–

	eg.) xsrl %rd,%rs	ld.w %r9,%rs
		and %r9,0x1f
		cmp %r9,0x8
		jrle 4
		srl %rd,0x8
		jp.d -3
		sub %r9,0x8
		srl %rd,%r9				

	%rd, imm5	imm5≤8	8<imm5<16	imm5=16	16<imm5≤24	24<imm5
	eg.) xsrl %rd,imm5	srl %rd,imm5(3:0)	srl %rd,0x8 srl %rd,imm5(2:0)	srl %rd,0x8 srl %rd,0x8	srl %rd,0x8 srl %rd,0x8 srl %rd,imm5(3:0)	srl %rd,0x8 srl %rd,0x8 srl %rd,0x8 srl %rd,imm5(2:0)

Remarks

Expansion Format of Extended Instructions (7)

Extended instruction		Expansion format				
Opcode	Operand	Condition 1	Condition 2	Condition 3	Condition 4	Condition 5
xbtst **xbclr** **xbset** **xbnot**	**[imm32], imm3** <when gp is not used> eg.) xbtst.w [imm32],imm3	imm32≤0x1f ld.w %r9,imm32(5:0) btst [%r9],imm3	0x1f<imm32≤0x3ffff ext imm32(18:6) ld.w %r9,imm32(5:0) btst [%r9],imm3	0x3ffff<imm32 ext imm32(31:19) ext imm32(18:6) ld.w %r9,imm32(5:0) btst [%r9],imm3	–	–
	[imm32], imm3 <when gp is used> eg.) xbtst.w [imm32],imm3 (sign32=-gp+imm32)	sign32=0x0 btst [%r8],imm3	0x0<sign32≤0x1fff ext sign32(12:0) btst [%r8],imm3	0x1fff<sign32≤0x3ffffff ext sign32(25:13) ext sign32(12:0) btst [%r8],imm3	0x3ffffff<sign32 ext imm32(31:19) ext imm32(18:6) ld.w %r9,imm32(5:0) btst [%r9],imm3	imm32<gp Expanded into the format without gp specification according to the imm32 value.
	[symbol+imm32], imm3 <when gp is not used> eg.) xbtst [symbol+imm32],imm3	symbol+imm32≤0x1f ld.w %r9,symbol+imm32@l btst [%r9],imm3	0x1f<symbol+imm32≤0x3ffff ext symbol+imm32@m ld.w %r9,symbol+imm32@l btst [%r9],imm3	0x3ffff<symbol+imm32 *1 ext symbol+imm32@h ext symbol+imm32@m ld.w %r9,symbol+imm32@l btst [%r9],imm3	–	–
	[symbol+imm32], imm3 <when gp is used> eg.) xbtst [symbol+imm32],imm3 (sign32=-gp+imm32)	symbol+sign32=0x0 btst [%r8],imm3	0x0<symbol+sign32≤0x1fff ext symbol+sign32@al btst [%r8],imm3	0x1fff<symbol+sign32≤0x3ffffff *1 ext symbol+sign32@ah ext symbol+sign32@al btst [%r8],imm3	0x3ffffff<symbol+sign32 ext symbol+imm32@h ext symbol+imm32@m ld.w %r9,symbol+imm32@l btst [%r9],imm3	symbol+imm32<gp Expanded into the format without gp specification according to the symbol+imm32 value.
	[symbol-imm32], imm3 eg.) xbtst [symbol-imm32],imm3	Fixed ext symbol-imm32@h ext symbol-imm32@m ld.w %r9,symbol-imm32@l btst [%r9],imm3	–	–	–	–
	[%rb+imm32], imm3 eg.) xbtst [%rb+imm32],imm3	imm32=0x0 btst [%rb],imm3	0x0<imm32≤0x1fff ext imm32(12:0) btst [%rb],imm3	0x1fff<imm32≤0x3ffffff ext imm32(25:13) ext imm32(12:0) btst [%rb],imm3	0x3ffffff<imm32 ext imm32(31:19) ext imm32(18:6) ld.w %r9,imm32(5:0) add %r9,%rb btst [%r9],imm3	–
	[%rb+symbol+imm32], imm3 eg.) xbtst [%rb+symbol-imm32],imm3	symbol+imm32=0x0 btst [%rb],imm3	0x0<symbol+imm32≤0x1fff ext symbol+imm32@al btst [%rb],imm3	0x1fff<symbol+imm32≤0x3ffffff ext symbol+imm32@ah ext symbol+imm32@al btst [%rb],imm3	0x3ffffff<symbol+imm32 ext symbol+imm32@h ext symbol+imm32@m ld.w %r9,symbol+imm32@l add %r9,%rb btst [%r9],imm3	–

Remarks

∗1) This extension format is used when the sysmpl is undefined.

Expansion Format of Extended Instructions (8)

Extended instruction		Expansion format				
Opcode	Operand	Condition 1	Condition 2	Condition 3	Condition 4	Condition 5
xbtst **xbclr** **xbset** **xbnot**	**[%rb+symbol-imm32], imm3** eg.) xbtst [%rb+symbol-imm32],imm3	Fixed ext symbol-imm32@h ext symbol-imm32@m ld.w %r9,symbol-imm32@l add %r9,%rb btst [%r9],imm3	–	–	–	–
	[%sp+imm32], imm3 eg.) xbtst [%sp+imm32],imm3	imm32=0x0 ld.w %r9,%sp btst [%r9],imm3	imm32≤0x3f ld.w %r9,%sp add %r9,imm32(5:0) btst [%r9],imm3	0x3f<imm32≤0x7ffff ld.w %r9,%sp ext imm32(18:6) add %r9,imm32(5:0) btst [%r9],imm3	0x7ffff<imm32 ld.w %r9,%sp ext imm32(31:19) ext imm32(18:6) add %r9,imm32(5:0) btst [%r9],imm3	–
xcall **xcall.d** **xjp** **xjp.d**	**label+imm32** eg.) xcall label+imm32	Expanded into 1 instruction ∗2 call label+imm32	Expanded into 2 instructions ∗2 ext label+imm32@rm call label+imm32@rl	Expanded into 3 instructions ∗2 ext label+imm32@rh ext label+imm32@rm call label+imm32@rl	–	–
xjr∗1 **xjr**∗1**.d**	**sign32** eg.) xcall,sign32	-256≤sign32≤254 call sign32(8:1)	-2097152≤sign32<-256 or 254<sign32≤2097150 ext sign32(21:9) call sign32(8:1)	sign32<-2097152 or 2097150<sign32 ext sign32(31:22)<<0x3 ext sign32(21:9) call sign32(8:1)	–	–

Remarks

∗1) xjreq, xjreq.d, xjrne, xjrne.d, xjrgt, xjrgt.d, xjrge, xjrge.d, xjrlt, xjrlt.d, xjrle, xjrle.d, xjrugt, xjrugt.d, xjruge, xjruge.d, xjrult, xjrult.d, xjrule, xjrule.d

∗2) Number of expanded instructions

Relative distance (absolute value) #A	Label location	-near flag	Number of expanded instructions
0 to 126 #B	Same file as the instruction	–	1
	Another file	–	2
To threshold value #C	–	–	2
To 0x7fffffff #D	–	Spesified	2
	–	Not specified	3
Unknown relative distance	–	Specified	2
	–	Not specified	3

#A: The value indicates the number of instructions from the extended branch instruction to the branch destination label.

#B: Up to 125 when branching toward to a higher address.

#C: Up to the threshold value - 1 when branching toward to a higher address in the same file.

#D: Up to 0x7ffffffe when branching toward to a higher address in the same file.

The threshold value is half of the value specified using -j option. When using the -j option's default value of 0x180000, the threshold value will be 0xc0000. Values in ( ) apply when branching to a lower address. The threshold value may be decreased due to distance judgment when branching toward to a lower address.

pp33 pseudo-instructions

#include	<file name>	Inserts other file in the source file.
#define	<define name> [<string>]	Defines a character string with a define name.
#macro	<macro name> [$1] [,$2] ... [,$32]	Defines a statement string with a macro name.
	<statements>	Branch label in the macro is specified
#endm		with $$1 to $$64.
#ifdef	<name>	Conditional assembling
	<statements 1>	<name> defined: <statements 1> is assembled.
[#else		<name> undefined: <statements 2> is assembled.
	<statements 2>]	
#endif		
#ifndef	<name>	Conditional assembling
	<statements 1>	<name> undefined: <statements 1> is assembled.
[#else		<name> defined: <statements 2> is assembled.
	<statements 2>]	
#endif		

as33 pseudo-instructions

.abs		Specifies absolute assembling. (Described at file top)
.code		Declares the start of a code section.
.data		Declares the start of a data section.
.comm	<global symbol> <size>	Declares a bss section and secures memory area.
.lcomm	<local symbol> <size>	Declares a bss section and secures memory area.
.org	<address>	Specifies an absolute address. (Only for absolute source)
.set	<symbol> <address>	Defines an absolute address for a symbol. (Only for absolute source)
.word	<data>[<data> ... <data>]	Defines word data in a code or dara section.
.word	<symbol>	Defines word data in a code or data section.
.half	<data>[<data> ... <data>]	Defines half word data in a code or dara section.
.byte	<data>[<data> ... <data>]	Defines byte data in a code or dara section.
.ascii	"<string>"	Defines an ascii character string in a code or dara section.
.space	<size>	Fills the specified size of area in a code or dara section with 0x0.
.align	<N>	Aligns location at 2^N boundary address.
.global	<symbol>	Declares the symbol as global.
.list		Turns output ON(.list)/OFF(.nolist) in the assembly list file.
.nolist		(Effective only when the -l option is specified)
.file	"<file name>"	Outputs source information for debugging.
.endfile		
.loc	<line number>	
.def	<symbol>, <parameters>, endef	Outputs symbol information for debugging.

Operators (provided by pp33)

		(Priority)	Examples
( )	Parenthesis	(1)	1+(1+2*5)
+	Plus sign	(2)	+0xff
-	Minus sign	(2)	-123
~	Negation	(2)	~0x1234
^H	Acquires bit field (31:19)	(2)	0x1234^H
^M	Acquires bit field (18:6)	(2)	0x1234^M
^L	Acquires bit field (5:0)	(2)	0x1234^L
^AH	Acquires bit field (25:13)	(2)	0x1234^AH
^AL	Acquires bit field (12:0)	(2)	0x1234^AL
*	Multiplication	(3)	0xf*5
/	Division	(3)	0x123/0x56
%%	Residue	(3)	0x123%%0x56
+	Addition	(4)	1+2
-	Subtraction	(4)	0xff-0b111
<<	Shifting to left	(5)	0x113<<3
>>	Shifting to right	(5)	1>>2
&	AND	(6)	0b1101&0b111
^	XOR	(7)	12^35
|	OR	(8)	0x123|0xff

Numbers and defnum names can be used as terms in expressions.
The expression is calculated as a signed 32-bit data.
Do not put any space or TAB between operator and number.

Symbols with displacement (provided by as33)

The as33 enables to describe symbols with displacement as shown below.
LABEL+imm32 LABEL+sign32

Symbol masks (provided by as33)

@rh (or @RH)	ext	LABEL@rh	LABEL@rh=<LABEL-PC>(31:22)
@rm (or @RM)	ext	LABEL@m	LABEL@m=<LABEL-PC>(21:9)
@rl (or @RL)	<op>	LABEL@rl	LABEL@rl=<LABEL-PC>(8:0)
@h (or @H)	ext	LABEL@h	LABEL@h=LABEL(31:19)
@m (or @M)	ext	LABEL@h	LABEL@m=LABEL(18:6)
@l (or @L)	ld.w	%rd, LABEL@l	LABEL@l=LABEL(5:0)
@ah (or @AH)	ext	LABEL@ah	LABEL@ah=LABEL(25:13)
@al (or @AL)	ext	LABEL@al	LABEL@al=LABEL(12:0)

<op> call/call.d, jp/ip.d, jrgt/jrgt.d, jrge/jrge.d, jrlt/ jrlt.d, jrle/jrle.d, jrugt/jrugt.d, jruge/jruge.d, jrult/ jrult.d, jrule/jrule.d, jreq/jreq.d, jrne/jrne.d