NEC

User's Manual

IMAPCAR Series Processor

1DC GUI Debugger Console Commands

Tools

Document No. U19917EE1V0UM00
Date Published: July 2009

Legal Notes

The information in this document is current as of July 2009. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.

- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".
  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.
  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.
  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).
  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

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

(Note)

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

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

Table of Contents

1 OVERVIEW

This document describes the commands that can be used in the console view of 1DC-SDB, which is a 1DC source-level debugger for IMAP series processors that is implemented as an Eclipse plug-in.

1DC-SDB is a GUI-based tool used to debug source code line-by-line while running one of the following in the background and communicating with it:

1. dbimap5, a terminal-version debugger that interfaces with the actual machine
2. simimap5, a simulator that simulates the operation of the actual machine

To simplify the following discussion, dbimap is used to refer to background tasks.

1DC-SDB does not issue commands to dbimap directly. Instead, it issues commands via sdbimap, a tool that communicates with 1DC-SDB while running. sdbimap internally interprets commands passed by 1DC-SDB and then issues a lower level command to dbimap, which communicates with sdbimap while running in the background, only if necessary. Therefore, all commands issued in the console view are passed to sdbimap once.

For example, to inform 1DC-SDB of the value of the variable X, the print X command is first issued to sdbimap. sdbimap receives this command and then uses the debugging information in the executable file to find the address Y of X. Next, sdbimap issues the lower level command showd Y to dbimap. As a result, dbimap passes the value stored at the address Y returned by sdbimap to 1DC-SDB. Note that the debugging information for an executable file can be embedded in that file by using the 1DC compiler cc1dc5 to compile the 1DC source file with the –db* option specifiedNote.

Because sdbimap was developed using Tcl/Tk, not only can it execute newly prepared commands for IMAP series processors (hereafter referred to as sdbimap commands), but it can also execute Tcl/Tk commands and scripts. This means that 1DC-SDB can be used for both GUI-based debugging and for debugging that uses batch files including Tcl/Tk scripts.

Note

When an executable file is compiled with the –db option (not –db1) specified, most optimization is suppressed when generating the code. To get the best performance possible out of an IMAP series processor, recompile and link 1DC source files without the –db1 or –db option specified after debugging them.

2 Tcl/Tk AND sdbimap

This chapter provides an overview of Tcl (the tool command language), which is used by sdbimap.

2.1 Tcl/Tk Functions Usable from sdbimap

By using sdbimap with Tcl scripts, complicated operations can be inspected. Because Tcl is an interpreted language, trial and error can be used during development for complex sequence control. In addition, if a required command is not provided as an sdbimap command, Tcl can be used to generate a user-specified command.

sdbimap can use almost all of the Tcl/Tk functions, including the following:

1. Command history (like the C shell)
2. Complete evaluation of mathematical expressions and support for all C operators
3. Broad range of math functions
4. Lists and arrays
5. Regular expression pattern matching
6. Procedures
7. User-defined commands
8. Command conversion
9. GUI creation (using Tk)

For details about Tcl and Tk (the Tcl tool kit), see the following online references:

- The Tcl manual pages can be obtained from http://www.tcl.tk/man/.
- For the Japanese Tcl/Tk manual pages, visit http://www.sra.co.jp/tb/tclman/.

2.2 Handling sdbimap Commands in Tcl/Tk Scripts

The sdbimap commands described in this document can be specified in Tcl/Tk scripts just like Tcl/Tk commands.

When an sdbimap command is called from within a Tcl/Tk script, of the results of executing the command, the character string returned to 1DC-SDB is added to the Tcl variable sdb::Results (a list). This makes it possible for other commands in a Tcl/Tk script to receive the character string in the results of executing an sdbimap command as data.

3 MAIN WINDOWS GENERATED BY sdbimap COMMANDS

This chapter introduces the functions and features of the main windows generated by `sdbimap` commands

ASM window

The asm window is generated using the asm command. The asm command displays the disassembled code of the function where the current program counter is in the asm window.

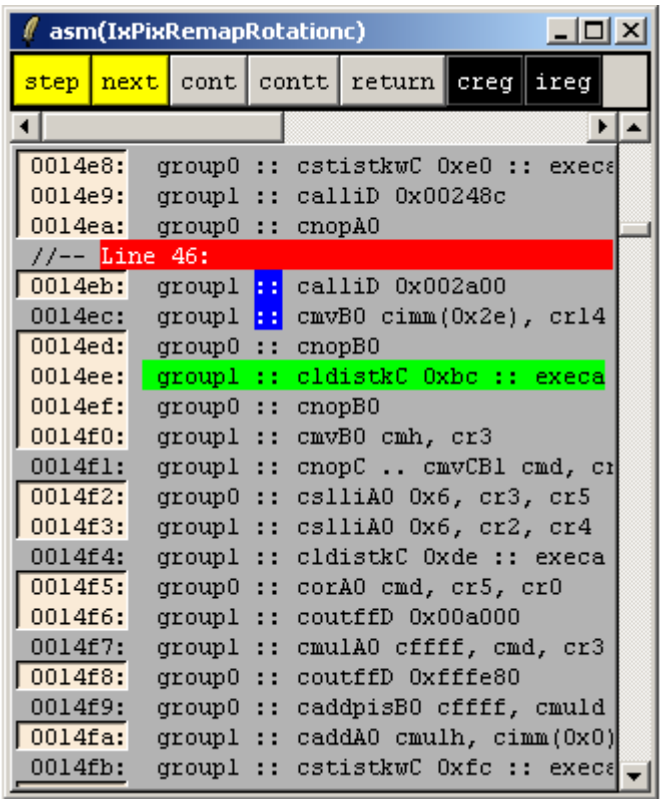


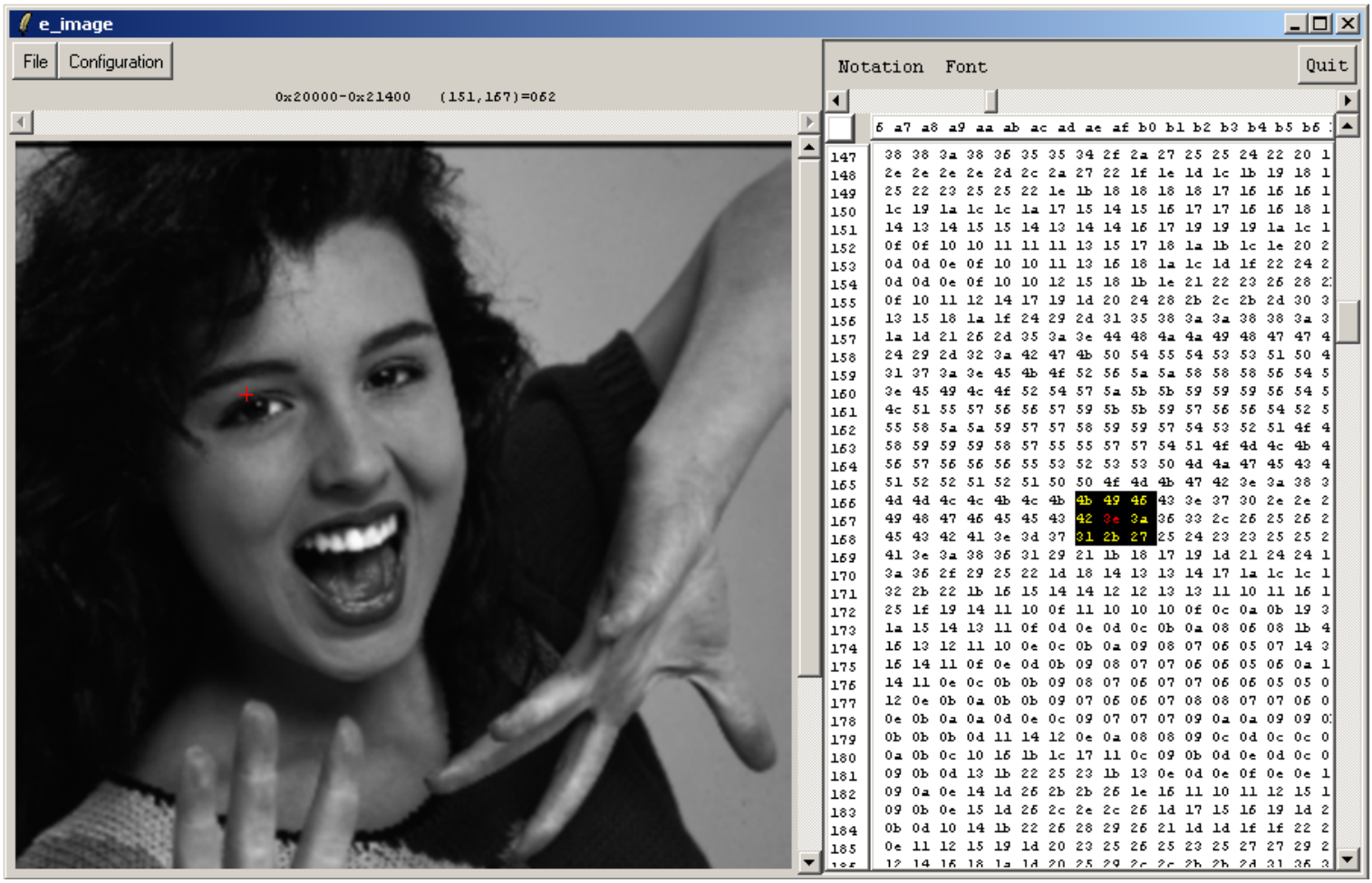
Image window

The image window is generated using the win command.

The image window contains window-manipulation menu buttons at its upper left. These buttons can be used to select the image display magnification, resolution, whether the image is in color or monochrome, the refresh mode, the starting address, the number of displayed lines, and the increment amount. The image window can be used to increment the starting display address by the specified amount, and provides a decrement button, return button, and increment button in that order at its upper right for doing so (when the compact display mode is canceled).

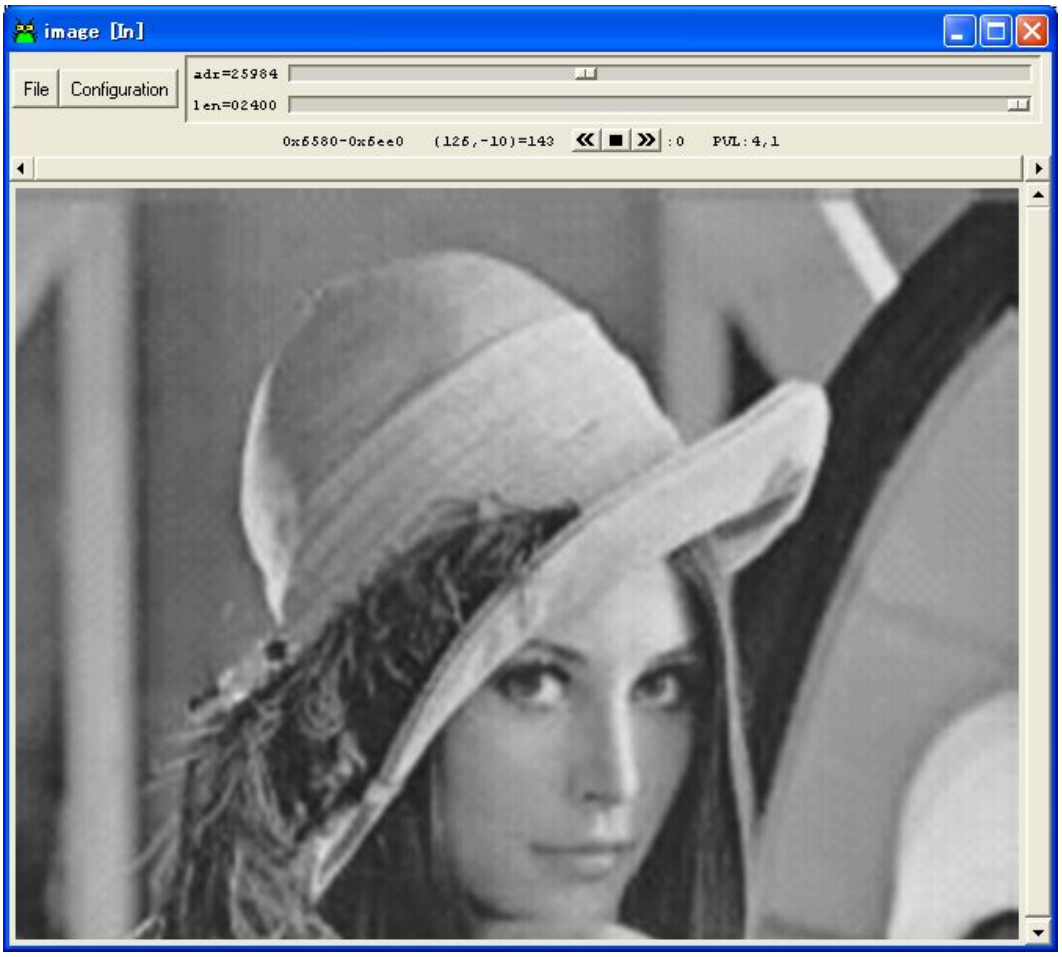
The pixel value at a specific location on an image can be displayed above the image in the format of (0xx,0yy)=value by left-clicking that location. The displayed value is a single value for a monochrome image or R, G, and B values for a color image. The values to the left are the starting address and ending address of the image displayed in the current window. If you move the mouse while holding down the right mouse button, the values for the clicked pixel and the other pixels in the 3x3 grid centered on that pixel are displayed in a callout. If you press Shift at this time, the grid expands to a 5x5 grid, and, if you press Ctrl, the grid expands to an 11x11 grid.

The File menu button at the top of the image window includes a Character window entry. By clicking this entry, a numerical sub-window that displays the currently displayed image as characters can be generated, which is useful when examining the values of each pixel in an image.



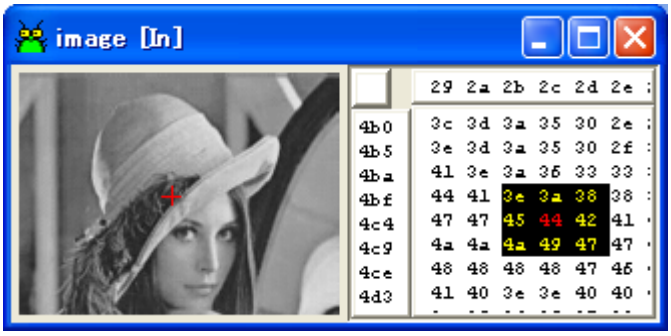
When a large image, such as a 640x480 image, is displayed in the image window, the window sometimes takes up most of the screen. In this case, it can be useful to reduce the size of the displayed image. To reduce the size, you can either use the sub-menu for specifying enlargement or reduction found by clicking the **Configuration** menu button or right double-click the image. When double-clicked, all portions of the image window are reduced to one-fourth the normal size, displaying the window as a thumbnail. To restore the window to its normal size, right double-click the image again. The following shows a window containing a full-size 640x480 image and two windows displayed as thumbnails.

Full-size display (640 × 480)



Thumbnail display (all areas reduced to 1/4)

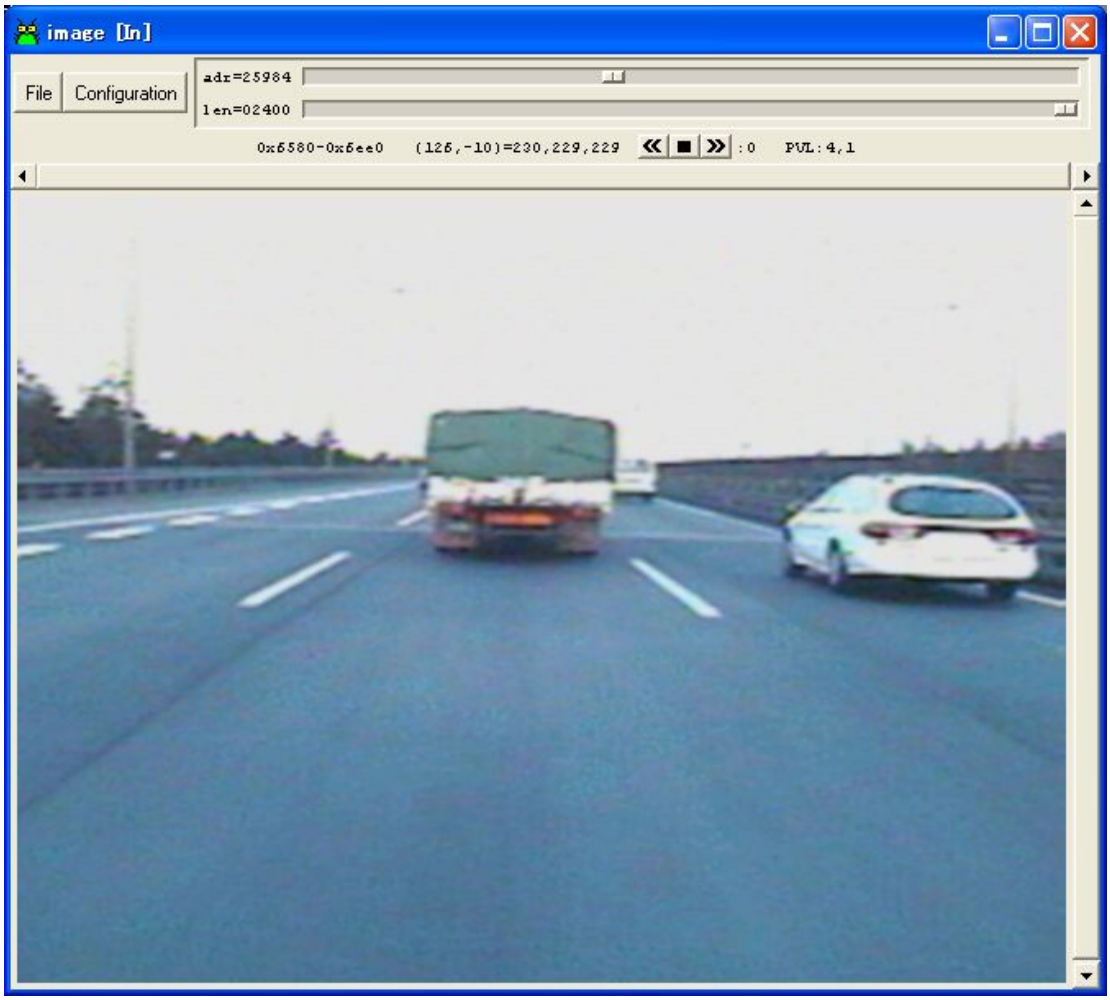
Thumbnail + numerical window display



Color images can also be displayed. If a Y/C image that includes starting luminance components and the corresponding color components is stored at sequential addresses in memory, the image can be displayed in the image window by specifying **Color (yc)** instead of **Grayscale** in the **Color** sub-menu found by clicking the **Configuration** menu button in the image window.

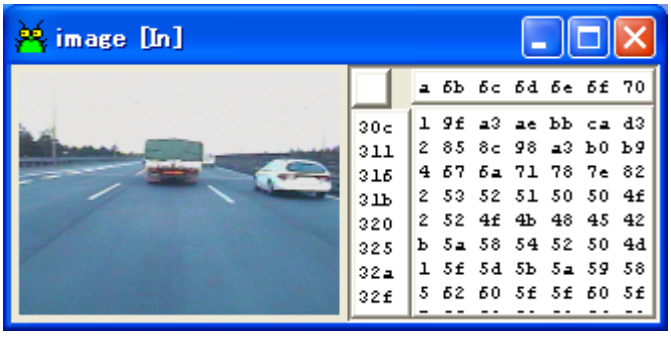
If an image that has the `.ppm` extension is loaded using the **File** menu, the image window automatically changes to the color display mode.

Full-size display (640 × 480)



Thumbnail display
(all areas reduced to 1/4)

Thumbnail + numerical window display

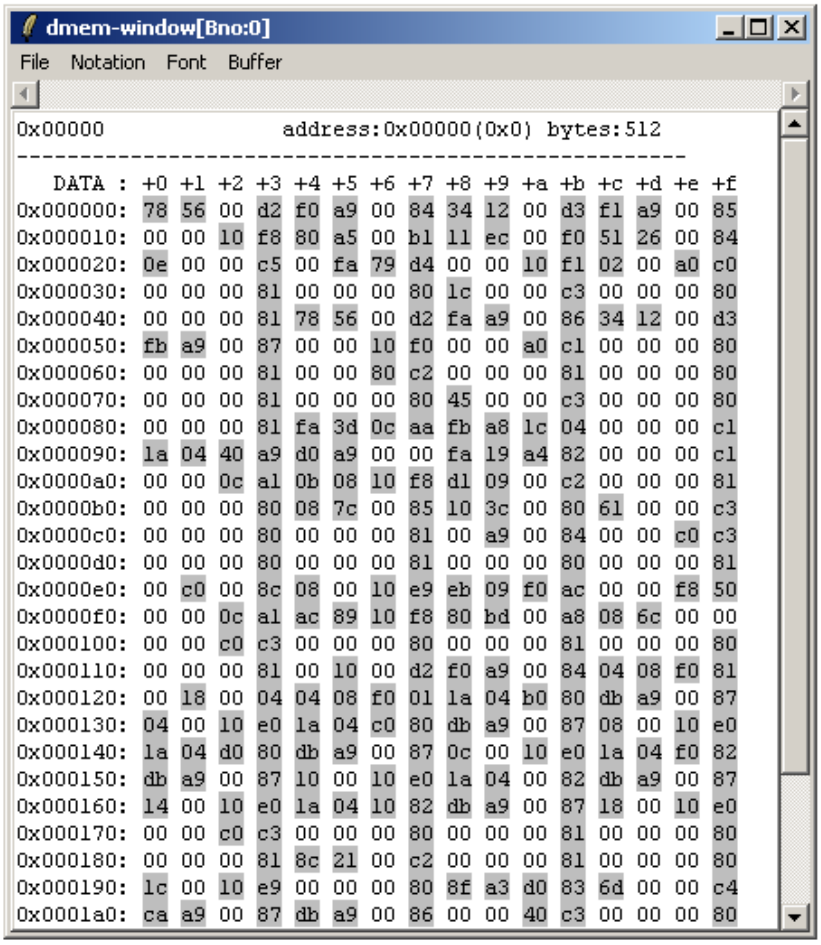


Memory window

Memory windows display the contents of memory as text and are generated using the `winop` command. The following types of memory windows can be generated:

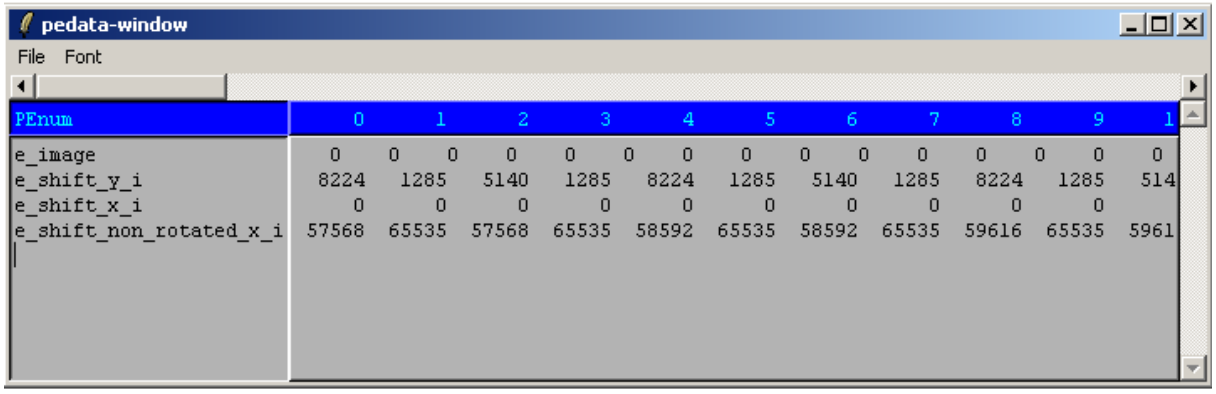
1. Dmem window (displays the data memory)
2. Imem window (displays the PE array memory)
3. Emem window (displays the external image memory)
4. Pmem window (displays the program memory)
5. Creg window (displays controller registers)
6. Ireg window (displays PE array registers)

An example of the dmem window, which displays the contents of the data memory, is shown below. The first line contains information about the starting address of the displayed contents and the number of displayed bytes. The data memory actually consists of data in external memory and data in the cache, but, if addresses are cached and the contents have been overwritten, the dmem window displays the addresses in red and the corresponding data in the cache. In addition, non-zero data in the window is highlighted in gray for emphasis.



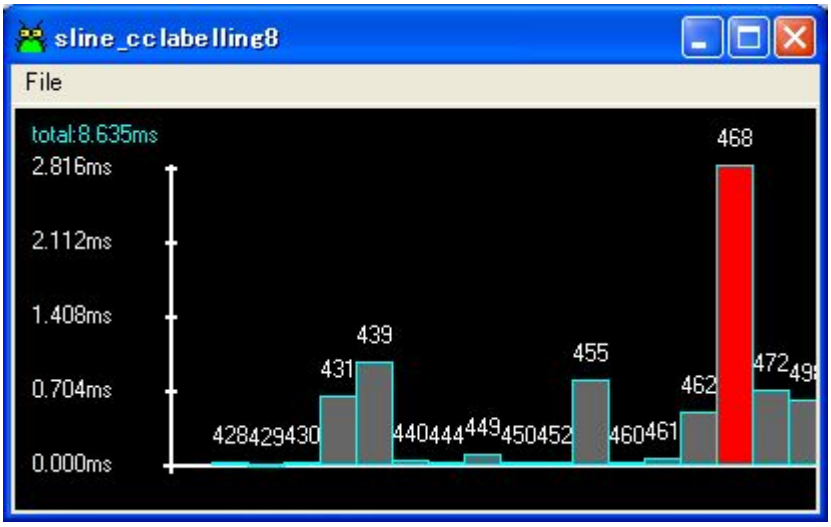
PEdata window

The pedata window displays the values of (sep-type) variables assigned to PE array memory. sep-type variables consist of a number of elements equal to the number of PEs, and displaying their values requires a wide space. Therefore, although the values of normal (non-sep-type) variables are displayed in the console view when the `print` command is used, when the `print` command is used to display the value of a sep-type variable, the pedata window is generated and displays the value.



Timing window

The timing window displays the results of measuring the processing time on a line or function basis. This window is generated by executing the `sline` command with a function name or line number specified. The measured processing time for each line of code in the specified function is displayed in the console view, and, at the same time, a timing window is generated that displays a bar graph in which the bar for the line of code that had the longest processing time is displayed in red and the line numbers are displayed over the bars.



View window

The view window (editing window) displays information generated by `sdbimap` commands and is a simple internal editor for editing 1DC programs. This window is generated by executing the `view` command with a file name specified, and the contents of the file are displayed in the window.

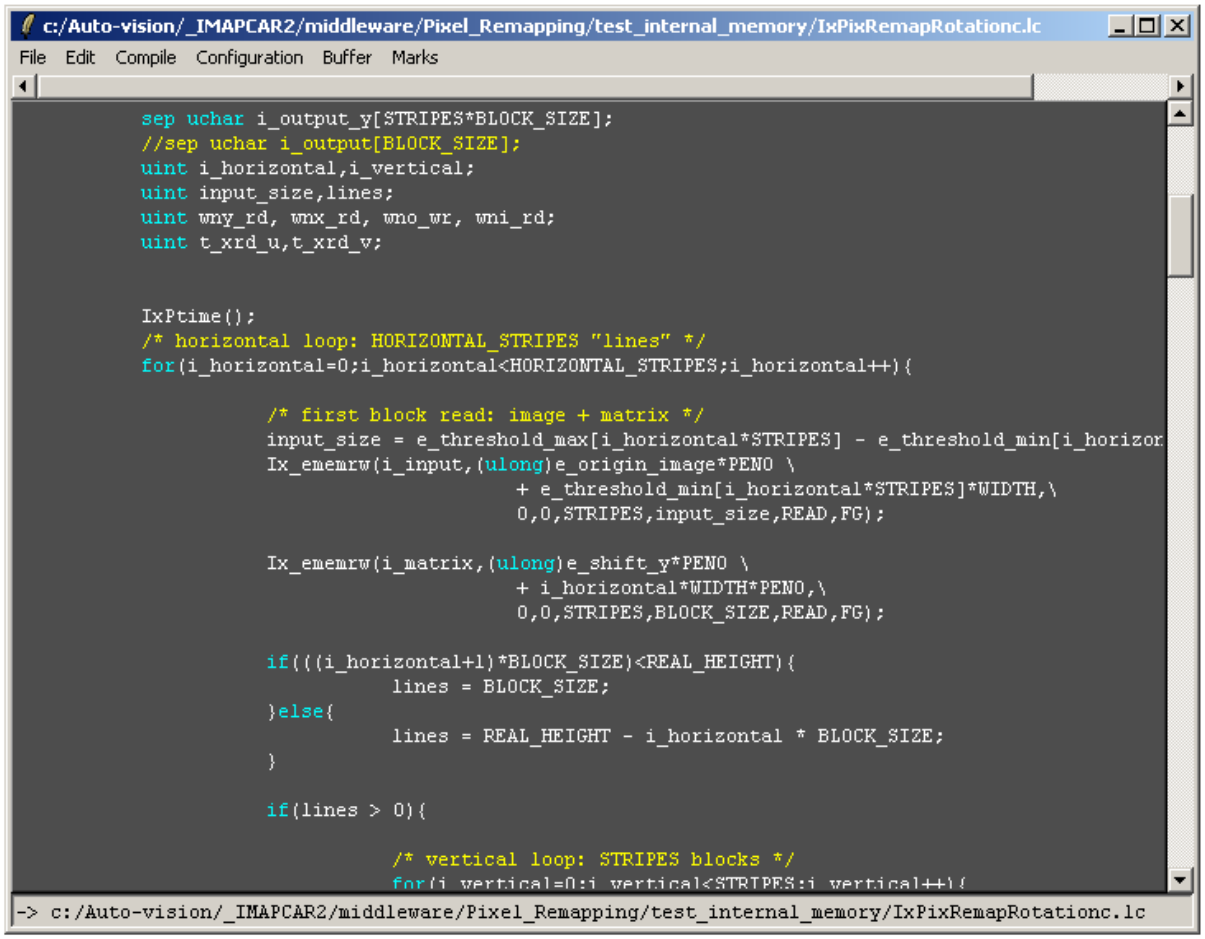
The view window is intended to be used for the following purposes:

1. Editing, compiling, and linking 1DC programs

Text displayed in the view window can be edited. After editing this text, `cc1dc` (the 1DC compiler) can be directly called from the Compile menu at the top of the window to compile, assemble, and link programs, and, if warning or error messages are output by the compiler, entries are automatically created in the Marks menu at the top of the window to enable movement to the locations that caused the warnings or errors. However, executable programs generated by using this compiler are stored in the current directory, regardless of the 1DC-SDB settings. Therefore, we recommend normally using the project builder provided by 1DC-SDB to compile source code.

2. Displaying profile results

A view window is generated to display the results of using the `prof` command to obtain information about the processing time.



Variable control windows

1D sliders, check boxes, a 2D scroll map, and a 2D pointer map can be generated as a useful GUI for interactively adjusting global variables while looking at the video output result of image recognition processing. These windows are collectively referred to as *variable control windows*. These windows include the scale window, sel window, scale2d window, and scale2dx window, and are generated by executing the `scale`, `sel`, `scale2d`, and `scale2dx` commands, respectively, with variables (or sets of variables) specified.

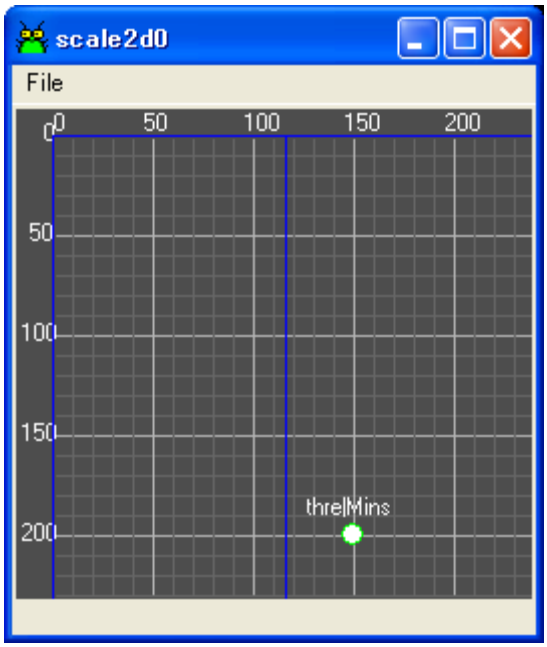
By performing one of the four operations below, values (the x and y coordinates of the cursor in the case of the 2D scroll map or 2D pointer map) are written to the memory for their global variables (or sets of global variables in the case of the 2D scroll map or 2D pointer map). In particular, real time positional information for the mouse in the 2D pointer map is written to its set of variables as required (without clicking the mouse).

1. Dragging a 1D slider
2. Selecting any of the check boxes
3. Changing the position of a mark in the 2D scroll map
4. Moving a mouse cursor that is in the 2D pointer map

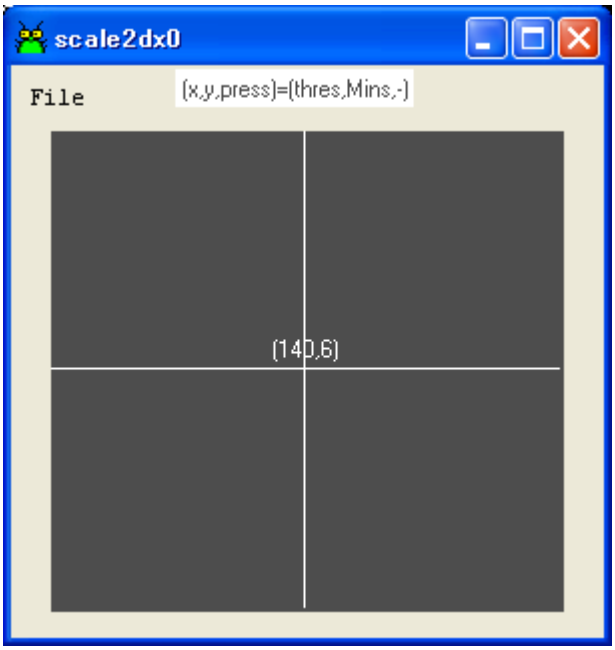
1D slider

Check box

2D scroll map



2D pointer map



4 COMMANDS

4.1 Command Format

The general format of `sdbimap` commands is as follows:

<*command name*> [<*parameter*>...] [; <*command name*> [<*parameter*>...]]

Commands and parameters are delimited using non-newline whitespace characters (spaces or tabs). The execution of multiple commands can be specified on one line by inserting semicolons (`;`) between the commands.

In the description of how to specify commands below, portions enclosed by < and > are individual parameters, and portions enclosed by [and] are optional parameters. If … appears after a parameter, an optional number of parameters can be specified after that parameter.

When numerical parameters are specified, they are normally handled as decimal values. However, if such parameters start with 0x, they are handled as hexadecimal values. The following shows example command formats and how many parameters are specified for each.

Format	Number of parameters
<*command name*>	0
<*command name*> <*parameter*> <*parameter*>	2
<*command name*> [<*parameter*>]	0 or 1
<*command name*> [<*parameter*> [<*parameter*>]]	0, 1, or 2
<*command name*> <*parameter*>...	1 or more
<*command name*> [<*parameter*>...]	0 or more

4.2 IMAP Executable Search Paths

`sdbimap` retains the paths to search for executable IMAP files (the search paths). These paths are implicitly used when commands such as `runv` are used to specify executable IMAP files (whose extension is `IX`).

The `path` command can be used to add, remove, or clear search paths. When `sdbimap` is started, the current directory is default as a search path.

4.3 Specifying Command Parameters

Among the commands described in this chapter, many have one of the following formats:

Format: *command name....* [<*var-name*>] [<*line-number-or-function-name*>] [<*other-parameters*>]
Format: *command name....* [<*break-line-number*>] ...
Format: *command name....* [<*executable-file-name*>] ...

The following describes how to specify <*var-name*>,<*line-number-or-function-name*>, and <*break-line-number*>, which are common to all commands.

4.3.1 Specifying <var-name>

Commands that have the <*var-name*> parameter obtain the following five types of information based on the specified <*var-name*> and then reference and modify the current <*var-name*> value or simply return an address:

1. The absolute address
2. The memory type
3. The number of accessed bytes
4. The data type
5. The PE position (if the memory type is IMEM or EMEM)

For example, based on the <*var-name*> value specified as its first parameter, the `print` command displays the number of accessed bytes of data starting at an absolute address in the specified type of memory in the command window or pedata window, in accordance with the specified data type.

Either of the formats below can be used for <*var-name*>, but the first is normally used. Note that `[` is an explicitly specified square bracket. Also, if `{ }` is followed by `?`, the portion enclosed by `{ }` is repeated 0 or more times, and, if the expression enclosed by `{ }` is of the form *A*|*B*, either *A* or *B* is specified.

1. `{*} ?` <variable A>`{\[`<uni-type variable B>|<value B>`] }? {: \[`<uni-type variable C>|<value C>`:]}? }`

    a) Specify variables currently in the scope of execution for variable A, variable B, and variable C. Note that variable B and variable C must be of the `uni` type.

    b) The address for variable A (or the result of adding an offset obtained by multiplying any specified uni-type variable B or value B with the number of bytes in each dimension relevant to variable A) specifies an absolute address.

    c) A * prefix means that a specification is intermediate and that the previously mentioned absolute address specifies the absolute address. If there are multiple `*`s, the above operation is repeated that many times. The number of `*`s that can be specified depends on the declared data type of variable A.

    d) The memory type, number of accessed bytes, and data type are automatically calculated based on the specification for `{*}?`<variable A>`{[`<uni-type variable B>|<value B>`]}?`.

    e) If the memory type is IMAP memory or external memory, *uni-type variable B* or *value B* specifies the PE position.

    f) If no PE position is specified, all PEs are assumed to be specified.

For example, for the following 1DC code:

```
sep uchar a[240];
int b,c;
```

Specifications for <*var-name*> such as the following are valid:

a	Address of a
`a\[b]`	All elements of the `sep` data `a[b]`
`a\[1]`	All elements of the `sep` data `a[1]`
`a\[b]:\[c:]`	Element c of the `sep` data `a[b]`
`a\[b]:\[3:]`	The third element of the `sep` data `a[b]`

2. {<value A>{:<letter A>}?{:<value B>}?{:<value C>}?}

a) value A specifies an absolute address. Any one of the following letters can be specified for letter A as the memory type: d, which indicates the data memory, i, which indicates the IMAP memory, e, which indicates the external memory, or p, which indicates the program memory. If this letter is omitted, the data memory is assumed.

b) value B is optional and specifies the number of accessed bytes as 1, 2, 4, or 8. If this value is omitted, 1 is assumed.

c) The data type cannot be specified and is always assumed to be unsigned.

d) If the specified memory type is IMEM (PE array memory) or EMEM (external memory), value C is assumed to specify the PE position, and, if this value is omitted, all PEs are assumed to be specified.

The following are example <var-name> specifications.

<var-name>	**Description**
`2:i:2`	Specifies the two lines (2 bytes per PE) starting at the IMEM address 2 and is assumed to be one item of `unsigned int` data.
`12:e:1:3`	Specifies the value of the line (one byte per PE) starting at the EMEM address 12 of PE 3 and is assumed to be one item of `unsigned char` data.

Example 1DC code and the absolute addresses, memory types, numbers of accessed bytes, data types, and PE positions acquired for several example <*var-name*> specifications are shown below. Note that the absolute memory address `G`*x* is assigned to the global variable `x` during linkage, and the local variable `y` corresponds to the absolute memory address `L`*y* obtained by adding the offset corresponding to `y` to the stack pointer when execution processing enters the scope of `y`. In addition, `-` indicates unspecified information, *?* indicates an error (due to an invalid specification), and * addresses indicate the corresponding address.

```
int a[10];
static unsigned int b[3][4][5][6];
char *c;
outside sep unsigned char *d;

int func(x,y)
separate long x;
separate unsigned int y;
{
      int *i();
      static int (*j)();
      int k;
      ...........
}
```

<var-name>	Address	Memory type	Number of accessed bytes	Data type	PE Position
a	Ga	-	-	pointer	-
a[2]	Ga + 4	dmem	2	signed	-
*a[2]	?	?	?	?	-
b	Gb	-	-	-	-
b[1]	Gb + 240	-	-	pointer	-
b[1][2]	Gb + 420	-	-	pointer	-
b[1][2][1]	Gb + 480	-	-	pointer	-
b[1][2][1][1]	Gb + 4820	dmem	2	unsigned	-
c	Gc	dmem	4	pointer	-
*c	*Gc	dmem	1	signed	-
**c	?	?	?	?	-
d	Gd	dmem	4	pointer	
d[1]	Gd + 4	dmem	4	pointer	
*d[2]	* (Gd + 2)	emem	1	unsigned	
*d	* Gd	emem	1	unsigned	
*d :[127 :]	* Gd	emem	1	unsigned	127
*d :[:c]	* Gd	emem	1	unsigned	Gc
x	Lx	imem	4	signed	-
x[4]	Lx + 16	imem	4	signed	-
y	Ly	imem	2	unsigned	-
y:[2:]	Ly	imem	2	unsigned	2
y[3] :[k:]	Ly + 6	imem	2	unsigned	Gk
y[k] :[k:]	Ly + 2*k	imem	2	unsigned	Gk
i	Gi	-	-	pointer	-
*i	?	?	?	?	-
j	Gj	dmem	2	pointer	-
*j	?	?	?	?	-
k	Lk	dmem	2	signed	-
k:[2:]	Lk + 6	dmem	2	signed	-
k[3] :[k:]	?	?	?	?	-
k[3] :[k:]	?	?	?	?	-

4.3.2 Specifying <line-number-or-function-name>

Commands that have the <*line-number-or-function-name*> parameter obtain scope information based on <*line-number-or-function-name*>. The `print` command uses this scope information, for example, to determine whether specified variables are local or global.

When both <*var-name*> and <*line-number-or-function-name*> have to be specified in the format for one command, the scope of <*var-name*> is determined based on the following rules:

1. If <*line-number-or-function-name*> is omitted, <*var-name*> is first searched for within the scope of the stop position and then globally.
2. If <*line-number-or-function-name*> is `-`, <*var-name*> is assumed to be a global variable.

The format for <*line-number-or-function-name*> is as follows:

Format: {<*file-name*>`:`} ? {<*function-name*>`:`}? {<*value*>}

If <*file-name*> is omitted, the currently loaded source file is assumed to be specified, and, if <*function-name*> is omitted, the function that includes line <*value*> in the currently loaded source file is assumed to be specified. Commands that include <*line-number-or-function-name*> among their parameters first search for <*var-name*> in the specified scope. <*var-name*> is assumed to be a local variable if it is found or a global variable if not.

For commands for which parameters can be specified after <*line-number-or-function-name*>, specify `0` for <*line-number-or-function-name*> to omit it.

4.3.3 Specifying <break-line-number>

<*break-line-number*> is specified using one of the following formats:

<*line-number*>

<*file-name*>`:`<*line-number*>

<*function-name*>

If only a line number is specified, the number is assumed to refer to the source file currently loaded in the source window. If a function name is specified, the starting address of that function is assumed.

4.3.4 Specifying <*executable-file-name*>

<*executable-file-name*> refers to an executable IMAP file that has the `.ix` extension, and, if the file is not in the current directory, the search paths are searched for it.

4.3.5 Specifying <*foreground-color*> or <*background-color*>

For commands that have the <*foreground-color*> or <*background-color*> parameter, specify one of the following colors:

```
AliceBlue          AntiqueWhite      AntiqueWhite1    AntiqueWhite2
AntiqueWhite3      AntiqueWhite4     BlanchedAlmond   BlueViolet
CadetBlue          CadetBlue1        CadetBlue2       CadetBlue3
CadetBlue4         CornflowerBlue    DarkBlue         DarkCyan
DarkGoldenrod      arkGoldenrod1     DarkGoldenrod2   DarkGoldenrod3
DarkGoldenrod4     DarkGray          DarkGreen        DarkGrey
DarkKhaki          DarkMagenta       DarkOliveGreen   DarkOliveGreen1
DarkOliveGreen2    DarkOliveGreen3   DarkOliveGreen4  DarkOrange
DarkOrange1        DarkOrange2       DarkOrange3      DarkOrange4
DarkOrchid         DarkOrchid1       DarkOrchid2      DarkOrchid3
DarkOrchid4        DarkRed           DarkSalmon       DarkSeaGreen
DarkSeaGreen1      DarkSeaGreen2     DarkSeaGreen3    DarkSeaGreen4
DarkSlateBlue      DarkSlateGray     DarkSlateGray1   DarkSlateGray2
DarkSlateGray3     DarkSlateGray4    DarkSlateGrey    DarkTurquoise
DarkViolet         DeepPink          DeepPink1        DeepPink2
DeepPink3          DeepPink4         DeepSkyBlue      DeepSkyBlue1
DeepSkyBlue2       DeepSkyBlue3      DeepSkyBlue4     DimGray
DimGrey            DodgerBlue        DodgerBlue1      DodgerBlue2
DodgerBlue3        DodgerBlue4       FloralWhite      ForestGreen
GhostWhite         GreenYellow       HotPink          HotPink1
HotPink2           HotPink3          HotPink4         IndianRed
IndianRed1         IndianRed2        IndianRed3       IndianRed4
LavenderBlush      LavenderBlush1    LavenderBlush2   LavenderBlush3
LavenderBlush4     LawnGreen         LemonChiffon     LemonChiffon1
LemonChiffon2      LemonChiffon3     LemonChiffon4    LightBlue
LightBlue1         LightBlue2        LightBlue3       LightBlue4
LightCoral         LightCyan         LightCyan1       LightCyan2
LightCyan3         LightCyan4        LightGoldenrod   LightGoldenrod1
LightGoldenrod2    LightGoldenrod3   LightGoldenrod4  LightGoldenrodYellow
LightGray          LightGreen        LightGrey        LightPink
LightPink1         LightPink2        LightPink3       LightPink4
LightSalmon        LightSalmon1      LightSalmon2     LightSalmon3
LightSalmon4       LightSeaGreen     LightSkyBlue     LightSkyBlue1
LightSkyBlue2      aquamarine        aquamarine1      aquamarine2
aquamarine3        aquamarine4       azure            azure1
azure2             azure3            azure4           beige
bisque             bisque1           bisque2          bisque3
bisque4            black             blue             blue1
blue2              blue3             blue4            brown
brown1             brown2            brown3           brown4
burlywood          burlywood1        burlywood2       burlywood3
burlywood4         chartreuse        chartreuse1      chartreuse2
chartreuse3        chartreuse4       chocolate        chocolate1
chocolate2         chocolate3        chocolate4       coral
coral1             coral2            coral3           coral4
cornsilk           cornsilk1         cornsilk2        cornsilk3
cornsilk4          cyan              cyan1            cyan2
cyan3              cyan4             firebrick        firebrick1
firebrick2         firebrick3        firebrick4       gainsboro
gold               gold1             gold2            gold3
gold4              goldenrod         goldenrod1       goldenrod2
goldenrod3         goldenrod4        gray             gray0-gray99
grey               grey0-grey99      green            green1
```

```
green2          green3          green4          honeydew
honeydew1       honeydew2       honeydew3       honeydew4
ivory           ivory1          ivory2          ivory3
ivory4          khaki           khaki1          khaki2
khaki3          khaki4          lavender        linen
magenta         magenta1        magenta2        magenta3
magenta4        maroon          maroon1         maroon2
maroon3         maroon4         moccasin           navy
orange          orange1         orange2         orange3
orange4         orchid          orchid1         orchid2
orchid3         orchid4         peru            pink
pink1           pink2           pink3           pink4
plum            plum1           plum2           plum3
plum4           purple          purple1         purple2
purple3         purple4         red             red1
red2            red3            red4            salmon
salmon1         salmon2         salmon3         salmon4
seashell        seashell1       seashell2       seashell3
seashell4       sienna          sienna1         sienna2
sienna3         sienna4         snow            snow1
snow2           snow3           snow4           tan
tan1            tan2            tan3            tan4
thistle         thistle1        thistle2        thistle3
thistle4        tomato          tomato1         tomato2
tomato3         tomato4         turquoise       turquoise1
turquoise2      turquoise3      turquoise4         violet
wheat           wheat1          wheat2          wheat3
wheat4          white           yellow          yellow1
yellow2         yellow3         yellow4
```

Remarks

1. *gray0-gray99* refers to any value in the following sequence: `gray0`, `gray1`, `gray2`, ... `gray98`, `gray99`.
2. *grey0-grey99* refers to any value in the following sequence: `grey0`, `grey1`, `grey2`, ... `grey98`, `grey99`.

4.4 Commands According to Function

The following table lists the commands that can be executed from the console view according to their function.

Function	Command name
sdbimap commands	
Reloading IMAP executable files (IX files)	reload
Compiling 1DC programs	change compilec compiled compilem compilev fchange
General debugging	brk cont delete deleteall fill fstack getv mem next print ret run setv step symbols where stop
Debugging assembly code	creg ireg map mreg whereis crun
Adjusting parameters	rscale rscale2d refresh rsel scale1d scale2d scale2dx scalecolor sel selcolor touch wscale wscale2d wsel
Measuring processing time	measure prof sfunc sline cfunc
Setting up, starting, and stopping timer interrupts	timerint
Loading image data	loadd loade loadi loadpe loadpeblk loadpeblkrgb loadpeblkyc loadpergb loadpeyc loadpi loadpirgb loadpiyc loadse loadsi loadt
Saving image data	saved savee savei savepe savepeblk savepeblkrgb savepeblkyc savepergb savepeyc savepi savepirgb savepiyc savese savesi savet
Displaying the contents of images, data arrays, or registers	charwin thumbnail win winop
Other	addr alias geo help home ix kill path popd pushd tofile unalias wait
Tcl commands (examples)	
append array binary break case catch cd clock close concat continue dde encoding eof error eval exec exit expr fblocked fconfigure fcopy file fileevent flush for foreach format gets glob global history if incr info interp join lappend lindex linsert list llength lrange lreplace lsearch lsort namespace open package pid pkg::create pkg::mkIndex proc puts pwd read regexp registry regsub rename resource return scan seek set socket source split string subst switch tell time trace unknown unset update uplevel upvar variable vwait while	
Tk commands (examples)	
bell bind bindtags bitmap button canvas checkbutton clipboard destroy entry event focus font frame grab grid image label listbox lower menu menubutton message option options pack photo place radiobutton raise scale scrollbar selection send text tk tk_bindForTraversal tk_bisque tk_chooseColor tk_chooseDirectory tk_dialog tk_focusFollowsMouse tk_focusNext tk_focusPrev tk_getOpenFile tk_getSaveFile tk_menuBar tk_messageBox tk_popup tk_setPalette tkerror tkvars tkwait toplevel winfo wm	

4.5 Using sdbimap Commands from within Tcl/Tk Scripts

All sdbimap commands can be freely called from within Tcl/Tk scripts. However, except for the getv command, most are only specified to display their execution results in the console view, and their results cannot be directly passed to standard Tcl/Tk commands. However, the tofile command can be used to send the results of executing sdbimap commands to a file instead of displaying them in the console view. For details, see the `getv` and `tofile` commands.

Script example

```
load testbin.ix ;# Load the IX file.
loadpe testimage/trui128.pgm In ;# Load the image to process.
win In ;# Try displaying the loading results.
setv 100 thres ;# Try changing the value of a global variable.
setv lines 100 ;# Try changing the value of a global variable.
display i
display thres
brk testbin.lc:27 ;# Set up a breakpoint.
set limit 0
while {[getv i]!= 20 && $limit<30} {cont; incr limit}
delete testbin.lc:27 ;# Remove the breakpoint.
setv 100 i ;# Try changing the operation by changing the value of an
           # automatic variable.
# Save the display window contents to a file. (This uses an undocumented
# command.)
sdb::Savedisplay testbin_display.txt
# Execute processing to the end.
cont
win Out ;# Display the processing results.
savepe result.pgm Out ;# Save the processing results.
# Display addresses, values, and other data.
puts "\n value of thres is [getv thres]"
puts "\t address of thres is [addr thres]"
puts "\n address of i In is [addr In]"
puts "\t value of i In\[0\] is [getv In\[0\]]"
# Generate GUI buttons.
toplevel .mybutton
button .mybutton.runv -text "runv" -command "runv"
button .mybutton.doit -text "do it" -command "setv 1 DoIt"
pack .mybutton.runv ;# Generate a button that executes the runv command
                     # when clicked.
pack .mybutton.doit ;# Generate a button that sets the 1DC variable DoIt
                     # to 1 when clicked.
```

4.6 sdbimap Commands (in alphabetical order)

4.6.1 Addr

Format addr <var-name> [<line-number-or-function-name>]

This command displays the address of the variable specified for <var-name> in the scope specified for <line-number-or-function-name>. If the variable <var-name> does not exist in the scope of <line-number-or-function-name>, -1 is displayed.

Example > addr thres [CR]

1044000: d

(Displays the address of the variable thres: Here, a data memory address is displayed.)

> addr thresA [CR]

-1

(-1 is displayed because the variable thresA does not exist in the currently loaded executable program.)

> addr thresA test.lc: 22 [CR]

1042970: d

(The address of the local variable thres, which is in the function that includes line 22 of test.lc, is displayed.)

4.6.2 asm

Format asm [<func-name>]

This command opens a window that displays the assembly code for the function specified by <func-name>. If <func-name> is omitted, a window that displays the assembly code for the function at the current position is opened.

4.6.3 brk

Format brk <break-line-number>

This command sets up a breakpoint at line <break-line-number> of a 1DC source program. The line specified for <break-line-number> must be specifiable as a breakpoint. This command can also be executed by left-clicking the numbers of lines displayed in the source window that are specifiable as breakpoints. Clicked lines are set up as break points. Breakpoints are not set up if lines for which breakpoints are not specifiable are clicked.

Examples > brk 34 [CR]

(A breakpoint is set up at line 34.)

> brk main.lc:34 [CR]

(A breakpoint is set up at line 34 of the main.lc source file included in the executable file.)

4.6.4 cfunc

Format func <function-name>

This command executes the processing from the stopped position in the source to the first completion of the function specified by <function-name> in the run mode (in which video interrupts are prohibited). Based on the results of executing this command, the processing time from the beginning of the function until just before the last line is displayed in the command window. Unlike sfunc and other commands, the processing time displayed by this command includes the execution time for background processing and the number of processing steps, as well as the number of processing steps due to program cache misses and data cache misses.

4.6.5 change

Format change <option>*

This command changes the user-specified compiling options compilev, compilem, and compiled, which are used for all files, to <option>. If <option> is omitted, the dialog box is displayed, and entering new options is requested. If <option> is -, the current user-specified compiling options are cleared. If <option> is ?, the currently specified compiling options for each command related to compiling are displayed. Note that, for a file for which the fchange command was used to specify compiling options, the options specified using that command are used, not those specified using the change command.

Examples > change -v -Kdontraise [CR]

[Compilec] uses "-PENO=128 -O -c"

[Compiled] uses "-PENO=128 -O -db"

[Compilem] uses "-PENO=128 -O -m"

[Compilev] uses "-PENO=128 -O"

> change [CR]

(The dialog box is displayed, and entering new options is requested.)

> change - [CR]

User Compile Option is cleared.

4.6.6 charwin

Format charwin <var-name>

This command generates the numerical window for the image window that displays the contents of <var-name>. However, if the image window that displays these contents does not yet exist, the following command is used to generate the image window, and then the numerical window is generated:

charwin <var-name>

4.6.7 cont

Format cont

This command resumes execution of the interrupted program currently being debugged, and runs the program to the next assembly breakpoint specified, for example, using the `brk` command.

4.6.8 creg

Format creg

This command opens the window that displays the CP registers (the creg window).

4.6.9 cpprof

Format cpprof

This command calculates information about how many times all the functions in the current source file have been executed on a CP (and in the PE array) and the percentage of the total execution time used to execute these functions and for the total number of execution steps, and then displays the results in the view window. However, note that this command does not count the execution time for background processing, such as the `IxEmemrd` and `IxEmemwr` functions, or the number of processing steps due to program cache misses and data cache misses.

This command can be used regardless of the compiling options used when the executable file is created. For example, a profile can be obtained even of an executable file that does not include source information.

Because, unlike the `prof` command, this command does not collect information about functions executed in PUs, it can obtain a profile much faster than `prof`.

4.6.10 delete

Format delete <break-line-number>

This command removes the breakpoint specified for line <break-line-number> of a 1DC source program. The command does nothing if the specified line does not have a breakpoint or temporary breakpoint.

Examples > delete tst.lc: 34 [CR]

(The breakpoint or temporary breakpoint specified for line 34 of tst.lc is removed.)

4.6.11 deleteall

Format deleteall

This command removes all breakpoints and temporary breakpoints that are currently set up.

Examples > deleteall [CR]

(All breakpoints and temporary breakpoints that are set up are removed.)

4.6.12 display

Format `display` [<var-name>] [<line-number-or-function-name>] [<from> <to>]

This command displays the value of the variable specified by <*var-name*> that is in the scope specified by <*line-number-or-function-name*> in the display window. Note that this command cannot display an indexed element of a `sep`-type array.

Executing this command generates the display window if it does not exist. If <*var-name*> is a local variable, its value is only displayed in the display window when execution processing enters the scope of the variable. If <*from*> and <*to*> are specified, a <*var-name*> value at an address from the offset <*from*> to the offset <*to*> is displayed. The default values of both <*from*> and <*to*> are `0`.

If <*var-name*> is omitted, all variables in the current scope are displayed in the display window.

Examples Below, `MX` is a `uni`-type scalar variable, `sptr` is a pointer to `sep` data, and `sary` is a `sep`-type array.

```
char MX;
sep int *sptr:
```

`> display MX - [CR]` (The display window is generated and the value of `MX` is displayed.)

`> display sptr [CR]` (The value of `sptr` (a `uni`-type value) is displayed in the display window.)

`> display *sptr [CR]` (The value pointed to by `sptr` (a `sep`-type value) is displayed in the display window.)

`> display sptr[2] [CR]` (The value of `sptr[2]` (a `sep`-type value) is displayed in the display window.)

`> display sptr[2]:[3:] [CR]` (The value of `sptr[2]:[3:]` (a `uni`-type value) is displayed in the display window.)

`> display *sptr:[3:] [CR]` (The value of `*sptr:[3:]` (a `uni`-type value) is displayed in the display window.)

4.6.13 fill

Format fill <value> <fill-length> <variable-name> [<line-number-or-function-name>]

Examples `> fill 0 240 In [CR]` (This fills the 240 lines starting at the global sep-type `In` with 0s.) This command fills the <*fill-length*> byte area starting at the address of <*variable-name*> with <*value*>.

4.6.14 fstack

Format fstack

This command displays the contents of the current function stack. Note that this command only works correctly if all the 1DC source code is compiled with `-db` or `-db3` specified.

Examples > fstack [CR]

1) called from: do_fft() in binarize.lc: 12

2) called from: fft() in fft.lc: 34

3) current: check() in main.lc: 15

4.6.15 geo

Format geo <window-type> <x-position> <y-position> [<x-size> <y-size>] [<image-name>]

This command moves the window specified by <window-type> to the coordinates specified by <x-position> and <y-position>. If the <window-type> window does not exist, this command does nothing. Note that, if both <x-size> and <y-size> are specified, the window is also resized to <x-size> by <y-size> pixels.

Any one of the following character strings can be specified for <window-type>: display, image, scale1d (or scale), scale2d, sel, source, or view. If there is more than one window of the <window-type> type, the windows are moved to the specified position and tiled in order. However, if <window-type> is image and <image-name> is specified, only the image window corresponding to <image-name> is moved or resized. <image-name> indicates information such as the name of the sep-type array specified when the win command is executed.

4.6.16 getv

Format getv <var-name>

This command returns the value of the uni-type variable <var-name>.

4.6.17 help

Format help [<command>]

This command displays the format for <command>. If <command> is omitted, all sdbimap commands and dbimap commands, which use the ix command to run as background tasks, are displayed. To display all commands that can be executed from the command window, including Tcl/Tk commands, use the Tcl/Tk info command.

Examples > help load [CR]

Command format: load [<*file-name*>]

> info commands [CR]

(All valid commands are displayed.)

4.6.18 ireg

Format ireg

This command opens the window that displays the PE array registers (the ireg window).

4.6.19 ix

Format ix <dbimap-command-call>

The character string starting after ix and ending at a newline character is passed to dbimap, which handles background tasks. The dbimap execution results are displayed in the command window.

4.6.20 kill

Format kill <window-type>

This command closes the type of window specified by <*window-type*>. Any of the following can be specified for <*window-type*>: `display`, `regwin`, `scale1d` (or `scale`), `scale2d`, `scale2dx`, `sel`, or `view`.

4.6.21 loadd

Format loadd [<file-name>] [<var-name>] [<length>]

This command loads the file <file-name> into the <length> line area of the data memory starting at the address of the variable <var-name>. If a value is specified for <var-name>, it is assumed to be an absolute address in the data memory.

Specify the name of the binary file containing the data to be loaded for <file-name>. If <file-name> is omitted, or if <file-name> is -, the file browser starts, and the selection of a file is requested.

If other parameters are omitted, their default values are as follows:

<*var-name*>	`0:d`
<*length*>	File size

Examples > loadd ~/data/data.256 a

(The ~/data/data.256 file is loaded into the data memory starting at the address of the global variable a.)

> loadd - 10 [CR]

(The file browser starts, and the selection of a binary file is requested. The selected file is loaded into the data memory starting at the address 10.)

> loadd [CR]

(The file browser starts, and the selection of a binary file is requested. The selected file is loaded into the data memory starting at the address 0.)

4.6.22 loade

Format `loade` [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

This command is the same as the `loadi` command, except `loade` loads the file into the external memory (EMEM).

4.6.23 loadi

Format loadi [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

This command loads the file <file-name> into the <length> line area that is <width> bytes wide and starts at the position found by adding the offset in the PE direction <offset> to the first line of the sep-type array <var-name>. If a value is specified for <var-name>, it is assumed to be an absolute address in the IMAP memory.

Specify the name of the binary file containing the data to be loaded for <file-name>. If <file-name> is omitted, or if <file-name> is -, the file browser starts, and the selection of a file is requested.

If other parameters are omitted, their default values are as follows:

<*var-name*>	`0:i`
<*length*>	File size / <*width*>
<*offset*>	`0`
<*width*>	Number of PEs - <*offset*>

Examples > loadi ~/image/sample.raw In [CR]

(The ~/image/sample.raw file is loaded into the IMAP memory starting at the address of the sep-type array In.)

> loadi - 0x100: i [CR]

(The file browser starts, and the selection of a binary file is requested. The selected file is loaded into the IMAP memory starting at the address 256.)

> loadi [CR]

(The file browser starts, and the selection of a binary file is requested. The selected file is loaded into the IMAP memory starting at the address 0.)

4.6.24 loadpe, loadpergb, loadpeyc

Format loadpe [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

Format loadpergb [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

Format loadpeyc [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

The `loadpergb` command is the same as the `loadpe` command, except `loadpergb` loads the RGB data in the file into the area starting at the address of the first line of the `outside sep`-type array specified by <*var-name*> in the order R, G, and then B, loads a `ppm` binary file, and actually loads *either the number of lines of data specified by <length> multiplied by 3 or the number of lines in the <var-name> array, whichever is smaller*.

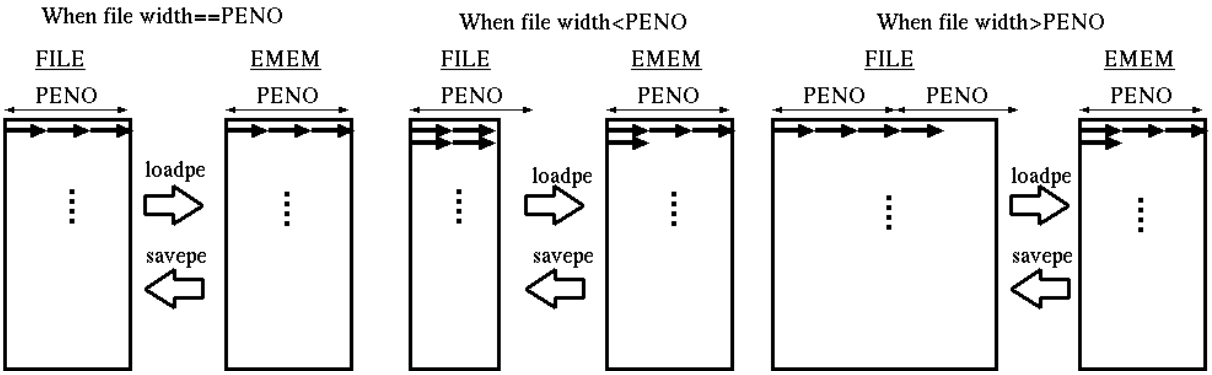
The `loadpeyc` command is the same as the `loadpe` command, except `loadpeyc` converts the RGB data in the file to the luminance data *y* and the color data *c* and loads this data into the area starting at the address of the first line of the `outside sep`-type array specified by <*var-name*> in the order *y* and then *c*, loads a `ppm` binary file, and actually loads *either the number of lines of data specified by <length> multiplied by 2 or the number of lines in the <var-name> array, whichever is smaller*.

Therefore, the following describes only `loadpe`.

Specify the name of the `pgm` binary file (or the `ppm` file for `loadpeyc` or `loadpergb`) containing the data to be loaded for <*file-name*>. If <*file-name*> is omitted, or if <*file-name*> is `-`, the file browser starts, and the selection of a file is requested.

The `loadpe` command loads the <*width*> by <*length*> rectangular area *R* starting at the beginning of <*file-name*> to pixel <*offse*t> of each line, starting at the first line of the outside `sep`-type array specified by <*var-name*>. If the total number of bytes in *R* exceeds the size of the <*var-name*> array, only the portion that does not exceed this size is loaded, and then a warning message is output. Note that, if `r` is specified for <*width*>, <*file-name*> is resized to a width equal to the number of PEs, and then the <*width of the number of PEs*> by <*length*> rectangular area *R* is loaded.

If a value such as `0:e` is specified for <*var-name*>, it is assumed to be an absolute address in EMEM (and all data in *R* is loaded because information about the array size does not exist). However, if there are less lines in <*file-name*> than the number of lines specified for <*length*>, `Bad memory length` is displayed and data is not loaded. Note that, if <*offset*> is specified, it is ignored. (For details, see the following figure.)



If parameters are omitted, their default values are as follows:

<*file-name*>	The file selection window is generated.
<*var-name*>	`0:e`
<*length*>	The value specified in the file
<*offset*>	`0`
<*width*>	The value specified in the file

4.6.25 loadpi, loadpirgb, loadpiyc

Format loadpi [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

Format loadpirgb [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

Format loadpiyc [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

The `loadpirgb` command is the same as the `loadpi` command, except `loadpirgb` loads the RGB data in the file to pixel <*offset*> of each line, starting at the address of the first line of the `sep`-type array specified by <*var-name*>, in the order R, G, and then B, loads a `ppm` binary file, and actually loads *either the number of lines of data specified by <length> multiplied by 3 or the number of lines in the <var-name> array, whichever is smaller*.

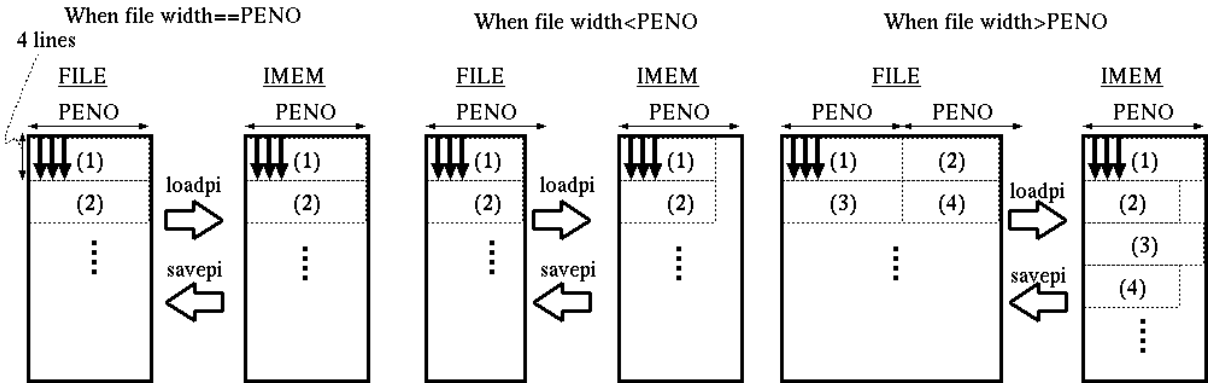
The `loadpiyc` command is the same as the `loadpi` command, except `loadpiyc` converts the RGB data in the file to the luminance data *y* and the color data *c* and loads this data into the area starting at the address of the first line of the `sep`-type array specified by <*var-name*> in the order *y* and then *c*, loads a `ppm` binary file, and actually loads *either the number of lines of data specified by <length> multiplied by 2 or the number of lines in the <var-name> array, whichever is smaller*.

Therefore, the following describes only `loadpi`.

Specify the name of the `pgm` binary file (or the `ppm` file for `loadpiyc` or `loadpirgb`) containing the data to be loaded for <*file-name*>. If <*file-name*> is omitted, or if <*file-name*> is `-`, the file browser starts, and the selection of a file is requested.

The `loadpi` command loads the <*width*> by <*length*> rectangular area *R* starting at the beginning of <*file-name*> to pixel <*offset*> of each line, starting at the first line of the `sep`-type array specified by <*var-name*>. If the total number of bytes in *R* exceeds the size of the <*var-name*> array, only the portion that does not exceed this size is loaded, and then a warning message is output. Note that, if `r` is specified for <*width*>, <*file-name*> is resized to a width equal to the number of PEs, and then the <*width of the number of PEs*> by <*length*> rectangular area *R* is loaded.

If a value is specified for <*var-name*>, it is assumed to be an absolute address in IMEM (and all data in *R* is loaded because information about the array size does not exist). However, if there are less lines in <*file-name*> than the number of lines specified for <*length*>, `Bad memory length` is displayed and data is not loaded. If <*offset + width*> exceeds the number of PEs, four lines of data are loaded at a time (which contain (number of PEs × 4 bytes) each), a newline character is added to the end of the fourth line, and then loading continues. (For details, see the following figure.)



If parameters are omitted, their default values are as follows:

<*file-name*>	The file selection window is generated.
<*var-name*>	`0:i`
<*length*>	The value specified in the file
<*offset*>	`0`
<*width*>	The value specified in the file

4.6.26 loadpeblk, loadpeblkyc, loadpeblkrbg

Format loadpeblk [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

Format loadpeblkyc [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

Format loadpeblkrbg [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

The `loadpeblkyc` command is the same as the `loadpeblkrgb` command, except `loadpeblkyc` converts the RGB data in the file to the luminance data *y* and the color data *c* and loads this data into the area starting at the address of the first line of the `sep`-type array specified by <*var-name*> in the order *y* and then *c*, loads a `ppm` binary file, and actually loads *either the number of lines of data specified by <length> multiplied by 2 or the number of lines in the <var-name> array, whichever is smaller*.

The `loadpeblkrgb` command is the same as the `loadpeblk` command, except `loadpeblkrgb` loads the RGB data in the file to pixel <*offse*t> of each line, starting at the address of the first line of the `sep`-type array specified by <*var-name*>, in the order R, G, and then B, loads a `ppm` binary file, and actually loads *either the number of lines of data specified by <length> multiplied by 3 or the number of lines in the <var-name> array, whichever is smaller*.

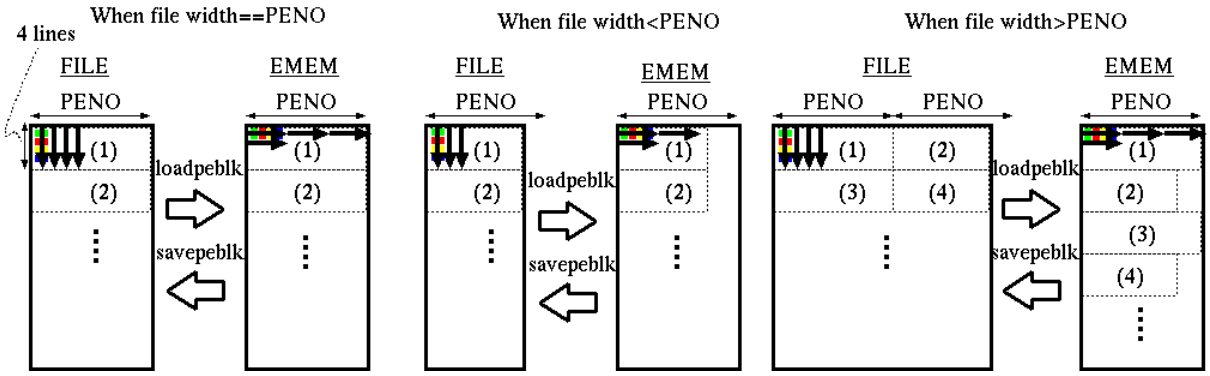
Therefore, the following describes only `loadpeblk`.

Specify the name of the `pgm` binary file (or the `ppm` file for `loadpeblkyc` or `loadpeblkrgb`) containing the data to be loaded for <*file-name*>. If <*file-name*> is omitted, or if <*file-name*> is `-`, the file browser starts, and the selection of a file is requested.

The `loadpeblk` command rotates every 4 bytes of the <*width*> by <*length*> rectangular area *R* starting at the beginning of <*file-name*> 90 degrees clockwise and loads the area to pixel <*offse*t> of each line, starting at the first line of the `sep`-type array specified by <*var-name*>. If `r` is specified for <*width*>, <*file-name*> is resized to a width equal to the number of PEs, and then the <*width-of-the-number-of-PEs*> by <*length*> rectangular area *R* is loaded.

If the total number of bytes in *R* exceeds the size of the <*var-name*> array, only the portion that does not exceed this size is loaded, and then a warning message is output. If a value is specified for <*var-name*>, it is assumed to be an absolute address in IMEM (and all data in *R* is loaded because information about the array size does not exist). However, if there are less lines in <*file-name*> than the number of lines specified for <*length*>, `Bad memory length` is displayed and data is not loaded.

If <*offset + width*> exceeds the number of PEs, 4 bytes from each PE are loaded at a time, a newline character is added to the end of the fourth line, and then loading continues. (For details, see the following figure.)



If parameters are omitted, their default values are as follows:

<*file-name*>	The file selection window is generated.
<*var-name*>	`0:e`
<*length*>	The value specified in the file
<*offset*>	`0`
<*width*>	The value specified in the file

Example > loadpeblk ~/image/sample.pgm In [CR]

(The ~/image/sample.pgm file is loaded into the EMEM area starting at the sep-type array In, with every 4 bytes rotated 90 degrees counter-clockwise.)

4.6.27 loadse

Format `loadse` [<file-name>] [<var-name>] [<length>]

This command is the same as the `loadsi` command, except `loadse` loads the file into an `outside sep`-type array.

4.6.28 loadsi

Format `loadsi` [<*file-name*>] [<*var-name*>] [<*length*>]

This command loads the file <*file-name*> into the <*length*> line area that starts at the first line of the `sep`-type array <*var-name*>. If a value is specified for <*var-name*>, it is assumed to be an absolute address in the IMAP memory. The loading address <*var-name*> and data length <*length*> can be specified by a file header, but the parameters specified on the command line are prioritized.

Specify the IMAP text file containing the data to be loaded for <*file-name*>.

If <*file-name*> is omitted, or if <*file-name*> is `-`, the file browser starts, and the selection of a file is requested.

An IMAP text file has the following format:

Loading-address number-of-lines `[CR]`

data data .... `[CR]` •

data data .... `[CR]` │ Number of lines of data

..... │

data data .... `[CR]` •

If other parameters are omitted, their default values are as follows:

<*var-name*>	`0:i`
<*length*>	The value specified in the file

Examples > loadsi ~/image/sample.256.imap In [CR]

(The ~/image/sample.256 file is loaded into the IMAP memory starting at the address of the sep-type array In.)

> loadsi . 256: i [CR]

(The file browser starts, and the selection of an IMAP text file is requested. The selected file is loaded into the area starting at the address 256.)

> loadsi [CR]

(The file browser starts, and the selection of an IMAP text file is requested. The selected file is loaded according to the loading address specified by the file-internal header.)

4.6.29 loadt

Format loadt [<file-name>] [<function-name>] [<length>]

This command loads the file <*file-name*> into the <*length*> line area in the program memory that starts at the position offset <*offset*> from the starting address of the function <*function-name*> and is <*width*> words wide. If a value is specified for <*function-name*>, it is assumed to be an absolute address in the program memory.

Specify the binary file that contains the program code to be loaded for <*file-name*>. If <*file-name*> is omitted, or if <*file-name*> is `-`, the file browser starts, and the selection of a file is requested.

This command is used to perform lower level operations. Use the `load` command to load normal IMAP executables to the program memory.

If other parameters are omitted, their default values are as follows:

<*function-name*>	`0`
<*length*>	*file-size* / (<*width*> * 2)
<*offset*>	`0`
<*width*>	5 - <*offset*>

Examples > loadt ~/sample/func.io main [CR]

(~/sample/func.io is loaded into the area starting at the address of the main function.)

> loadt - 0x100 [CR]

(The file browser starts, and the selection of a binary file is requested. The selected file is loaded into the program memory starting at the address 256.)

> loadt [CR]

(The file browser starts, and the selection of a binary file is requested. The selected file is loaded into the program memory starting at the address 0.)

4.6.30 map

Format map

This command displays information about the memory map of the currently loaded IMAP executable file.

Examples > cd /lpa/testprog [CR]

> load testbin.ix [CR]

> map [CR]

```
TEXT     IDATA    IBSS     DDATA      DBSS       EDATA     EBSS     PROGRAM
0000-0fea 0000-0000 --------- 20000-2052d 20530-2053d --------- 0800-09df * testbin.ix
```

4.6.31 measure

Format measure

When this command is executed, a dialog box is displayed that asks which function in the current source file to measure the processing time for. When a function is entered and **OK** is clicked, the result of measuring the processing time of the function is displayed.

4.6.32 mem

Format mem [p | d | i | e | pa | pm]

This command displays memory usage information for the currently loaded executable object file. The displayed information or display method differs depending on the specified parameter as follows:

`p`	Displays the usage status of the program memory (in order of the function names).
`pa`	Displays the usage status of the program memory (in order of the function addresses).
`pm`	Displays the usage status of the program memory (starting with functions that are using less memory).
`d`	Displays the usage status of the data memory (per variable).
`i`	Displays the usage status of the IMAP memory (per variable).
`e`	Displays the usage status of the IMAP external memory (per variable).

Note that, when displaying how much memory is used by functions, information about stack usage is displayed, but this is the amount of the stack explicitly used within functions, and the actual amount of the stack being used might also include the amount of the stack used by other functions called within the functions for which information is displayed.

4.6.33 mreg

Format mreg

This command opens the window that displays information about all CP and PU registers (the mreg window).

4.6.34 next

Format next

This command executes the processing from the position where processing was stopped to the next line where a breakpoint can be set up, with video interrupts prohibited (in the still mode). The execution of any function called during this processing is finished.

Note that this command cannot be used if the current position is in a function compiled using `-db1` or `-m`.

Also note that the next line where a breakpoint can be set up used by this command is the next such line based on the current line of code in the source window, not the next such line reached by executing the program (such as by stepping into functions). If the next such line cannot be reached by continuing program execution, the program continues to the next breakpoint or to the end.

Examples > next [CR]

... Stopped at line: tst.lc:51

> next [CR]

... Stopped at line: tst.lc:52

> next [CR]

... Stopped at line: tst.lc:53

4.6.35 pmem2file

Format pmem2file <filename>

This command writes out the result of disassembling the PMEM contents of all currently loaded executable files to the file <file-name>. This command can be used to determine which instructions are used in executable files.

4.6.36 print

Format print <var-name> [<line-number-or-function-name>] [<from> <to>] [u | d | x]

This command displays the value of the variable specified by <var-name>, which is in the scope specified by <line-number-or-function-name>. For a sep variable, the values of all elements are displayed in the pedata window, and information such as the total of the non-zero element values in each PE, the number of the PE that has the maximum element value, and the number of the PE that has the minimum element value is displayed in the command window.

If <from> and <to> are specified, a <var-name> value at an address from the offset <from> to the offset <to> is displayed. The default value of both <from> and <to> is 0.

Specifying u, d, or x for the fifth parameter changes the value to an unsigned decimal value, a signed decimal value, or a hexadecimal value, respectively. The default setting is to display a decimal value, and whether the value is signed depends on the data type of <var-name>.

4.6.37 prof

Format prof [<function-name>]

This command calculates information about the specified function in the current source file, including how many times the function is executed, the total number of execution steps, the total execution time for the function, and the percentage of the total processing time used for the function, and then displays the information in the view window. However, note that this command does not count the execution time for background processing, such as the ememrd and ememwr functions, or the number of processing steps due to program cache misses or data cache misses.

This command can be used regardless of the compiling options used when the executable file is created. For example, a profile can be obtained even of an executable file that does not include source information. Any function or functions called by the currently loaded executable file can be specified for <function-name>.

If <function-name> is omitted, this command displays information in the view window about all functions in the currently loaded executable file, including how many times the functions are executed, the total number of execution steps, the total execution time for the functions, and the percentage of the total processing time used for the functions.

4.6.38 refresh

Format refresh

This command executes the `rscale`, `rsel`, and `rscale2d` commands in a row. This command is useful when performing a reload after revising and recompiling a program for applying all the values in the corresponding locations in memory to the values displayed for 1D sliders and other value adjustment GUI tools before the reload.

If the image window, register window, memory window, or another window exists, this command also applies the values at the corresponding locations in memory to that window.

4.6.39 reload

Format reload

This command reloads an executable object file previously loaded using the `load` command from the disk. This command is used after the source file is recompiled.

Unlike when the `load` command is re-executed, when the `reload` command is executed and an executable object file is reloaded, the previously specified breakpoint information and the various windows generated for controlling parameters remain unchanged.

4.6.40 ret

Format ret

This command finishes executing the function at the current position and then returns to the function that called it.

4.6.41 rscale

Format rscale [<number>]

This command matches the position of the 1D slider specified by the number <number> with the current value of the global variable assigned to that slider in memory. If <number> is omitted, the above processing is performed for all 1D sliders.

Examples > rscale 0 [CR]

(The position of the 1D slider that has the ID number 0 is matched with the current value of the corresponding global variable.)

> rscale [CR]

(The positions of all 1D sliders are matched with the current value of the corresponding global variable.)

4.6.42 rscale2d

Format rscale2d [<number>]

This command matches the positions of each symbol in the 2D scroll map specified by the number <number> with the current value of the corresponding global variable (set). If <number> is omitted, the above processing is performed for the symbols of all 2D scroll maps.

Examples > rscale2d 0 [CR]

(The position of each symbol in the 2D scroll map that has the ID number 0 is matched with the current value of the corresponding global variable (set).)

> rscale2d [CR]

(The position of each symbol in all 2D scroll maps is matched with the current value of the corresponding global variable (set).)

4.6.43 rsel

Format rsel [<number>]

This command matches the value selected using the check box specified by the number <number> with the current value of the global variable assigned to that button in memory. If <number> is omitted, the above processing is performed for all check boxes.

Examples > rsel 0 [CR]

(The value selected using the check box that has the ID number 0 is matched with the current value of the corresponding global variable.)

> rsel [CR]

(The values selected using all check boxes are matched with the current values of the corresponding global variables.)

4.6.44 run

Format run [<executable-file-name>] [<var-name>=<value>]

If the first parameter is omitted, this command executes the currently loaded executable object file in the still mode (in which video interrupts are prohibited). If a breakpoint is set up, execution stops there. If there are no breakpoints, execution stops at the end of the main function.

If an <*executable-file-name*> with the extension `.ix` is specified for the first parameter, the corresponding executable object file is executed in the still mode (in which video interrupts are prohibited).

For <*var-name*>, specify the name of a global variable in either a previously loaded executable object file or in the executable object file specified as the first parameter (making sure to specify a variable that has the `long` data type or a larger one, that is, a variable that uses at least 4 bytes, for the second type of file). For <*value*>, specify the number for that variable. The required number of global variables and values can be sequentially specified in the format <*var-name*>=<*value*>. This command writes the corresponding initial value <*value*> to the memory for each specified global variable <*var-name*> immediately before executing the executable object file.

Examples :> brk 54 [CR]

> run [CR]

... Stopped at line: tst.lc:54

> run Threshold=30[CR]

... Stopped at line: tst.lc:34

(30 is specified for the global variable Threshold, and then the already loaded executable object file is executed.)

4.6.45 runv

Format runv [<executable-file-name>] [<var-name>=<value>]

This command executes the IMAP executable file specified by <*executable-file-name*> in the video mode, using the video I/O mode value shown below. If <*executable-file-name*> is not in the current directory, the search path is searched for the file, and, if <*executable-file-name*> is omitted, the currently loaded executable object file is executed.

For <*var-name*>, specify the name of a global variable in the specified executable object file. For <*value*>, specify the number for that variable. The required number of global variables and values can be sequentially specified in the format <*var-name*>=<*value*>. The `runv` command writes the corresponding initial value <*value*> to the memory for each specified global variable <*var-name*> immediately before executing the executable object file.

Examples > runv ~/sample.ix [CR]

(~/sample.ix is executed in the video mode.)

> runv tst fstart=20 fno=30[CR]

(20 and 30 are specified for the global variables fstart and fno in the object file tst.ix, respectively, and then test.ix is executed in the video mode.)

4.6.46 save

Format save [<file-name>]

This command saves the text in the source window into the file <*file-name*>. If <*file-name*> is omitted, the file browser is displayed, and inputting <*file-name*> is requested.

4.6.47 saved

Format saved [<file-name>] [<var-name>] [<length>]

This command saves the contents of the <*length*> line area of the data memory starting at the address of the variable <*var-name*> as the binary file <*file-name*>. If a value is specified for <*var-name*>, it is assumed to be an absolute address in the data memory. If <*file-name*> is omitted, or if <*file-name*> is `-`, the file name entry window is displayed, and the entry of a file name is requested.

If other parameters are omitted, their default values are as follows:

<*var-name*>	`0:d`
<*length*>	`1`

Examples > saved ~/data/data.256 a

(The contents of the data memory starting at the address of the variable a are saved to the file ~/data/data.256 in binary format.)

> saved - 10 [CR]

(The file name entry window is displayed, and the entry of a file name is requested. The contents of the data memory starting at the address 10 are saved to the entered file in binary format.)

> saved [CR]

(The file name entry window is displayed, and the entry of a file name is requested. The contents of the data memory starting at the address 0 are saved to the entered file in binary format.)

4.6.48 savee

Format savee [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

This command is the same as the `savei` command, except `savee` saves the data in an `outside sep` type array.

4.6.49 savei

Format savei [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

This command saves the <length> line area that is <width> bytes wide and starts at the position found by adding the offset in the PE direction <offset> to the first line of the sep-type array <var-name> to the file <file-name> in binary format. If a value is specified for <var-name>, it is assumed to be an absolute address in the IMAP memory. Specify the name of the file to which to save the data for <file-name>. If <file-name> is omitted, or if <file-name> is -, the file name entry window is displayed, and the entry of a file name is requested.

If other parameters are omitted, their default values are as follows:

<*var-name*>	`0:i`
<*length*>	`VLINES`
<*offset*>	`0`
<*width*>	Number of PEs - <*offset*>

Examples > savei ~/image/sample.raw In [CR]

(The contents of the IMAP memory starting at the address of the sep-type array In are saved to the file ~/image/sample.raw in binary format.)

> savei - 0x100:i [CR]

(The file name entry window is displayed, and the entry of a file name is requested. The contents starting at the IMAP memory address 256 are saved to the entered file in binary format.)

> savei [CR]

(The file name entry window is displayed, and the entry of a file name is requested. The contents starting at the IMAP memory address 0 are saved to the entered file in binary format.)

4.6.50 savepe, savepergb, savepeyc

Format savepe [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

savepergb [<file-name>] [<var-name>] [<length>] [<offset>] [<width>] [<color-offset>]

savepeyc [<file-name>] [<var-name>] [<length>] [<offset>] [<width>] [<color-offset>]

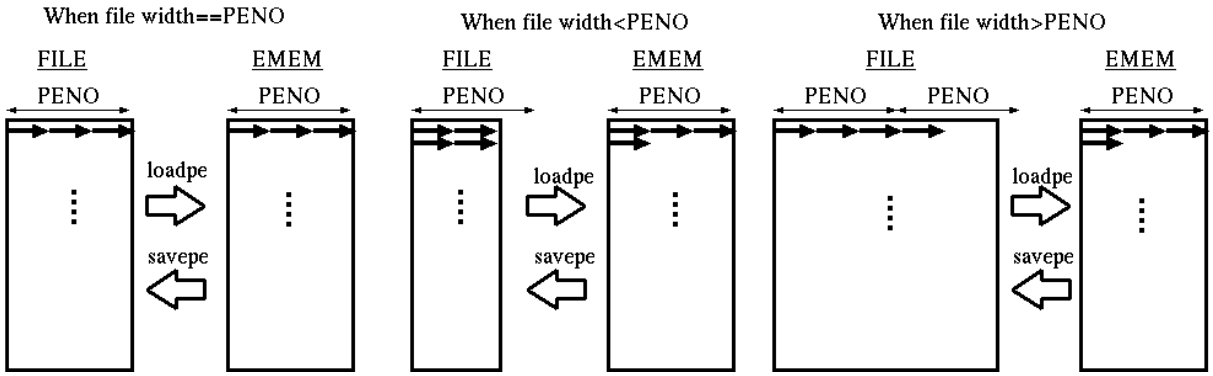
The `savepergb` command is the same as the `savepe` command, except, if <*color-offset*> is not specified for `savepergb`, it assumes the first <*length*> lines in the area starting at the first line of the `outside sep`-type array <*var-name*> to be R data, the next <*length*> lines to be G data, and the <*length*> lines following those to be B data, and saves data to a file in the `ppm` binary format. The `savepergb` command is also the same as the `savepe` command, except, if <*color-offset*> is specified, `savepergb` assumes the first <*color-offset*> lines in the area starting at the first line of the `outside sep`-type array <*var-name*> to be R data, the next <*color-offset*> lines to be G data, and the <*color-offset*> lines following those to be B data, and saves data to a file in the `ppm` binary format.

The `savepeyc` command is the same as the `savepe` command, except, if <*color-offset*> is not specified for `savepeyc`, it assumes the first <*length*> lines in the area starting at the first line of the `sep`-type array <*var-name*> to be y data and the next <*length*> lines to be c data, and saves data to a file in the `ppm` binary format. The `savepeyc` command is also the same as the `savepe` command if <*color-offset*> is specified, except `savepeyc` assumes the first <*color-offset*> lines in the area starting at the first line of the `sep`-type array <*var-name*> to be y data and the next <*color-offset*> lines to be c data, and saves data to a file in the `ppm` binary format. Note that, when a `ppm` binary file saved using the `savepeyc` command is reloaded to a `sep`-type array using a command such as `loadpeyc`, there are arithmetic errors during the conversion of the data from YC data, to RGB data, and then back to YC data that result in slight differences from the data originally stored in the `outside sep`-type array.

The following describes only `savepe`.

Specify the name of the file to which to save the data for <*file-name*>. If <*file-name*> is omitted, or if <*file-name*> is `-`, the file browser starts, and the selection of a file is requested.

The data in the area starting at the first line of the `outside sep`-type array <*var-name*> is saved to the <*width*> × <*length*> file <*file-name*> in the `pgm` binary format (or the `ppm` binary format for `savepeyc` or `savepergb`). If a value such as `0:e` is specified for <*var-name*>, it is assumed to be an absolute address in EMEM. Note that, if <*offset*> is specified, it is ignored. (For details, see the following figure.)



If parameters are omitted, their default values are as follows:

<*file-name*>	The file selection window is generated.
<*var-name*>	`0:e`
<*length*>	The number of lines in the `outside sep`-type array <*var-name*>
<*offset*>	`0`
<*width*>	Number of PEs
<*color-offset*>	`VLINES`

4.6.51 savepi, savepirgb, savepiyc

Format savepi [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

savepirgb [<file-name>] [<var-name>] [<length>] [<offset>] [<width>] [<color-offset>]

savepiyc [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

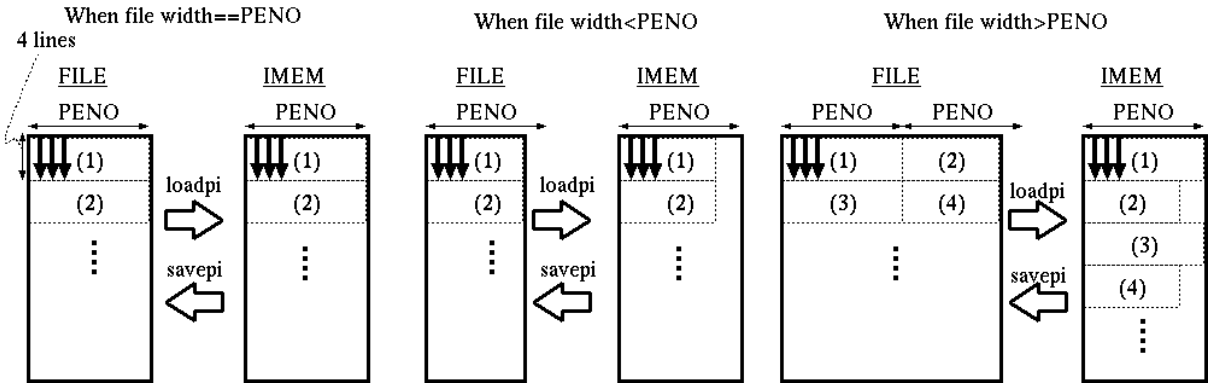
The `savepirgb` command is the same as the `savepi` command, except, if <*color-offset*> is not specified for `savepirgb`, it assumes the pixels starting at pixel <*offset*> on the first <*length*> lines in the area starting at the first line of the `sep`-type array <*var-name*> to be R data, the same pixels on the next <*length*> lines to be G data, and the same pixels on the <*length*> lines following those to be B data, and saves data to a file in the `ppm` binary format. The `savepirgb` command is also the same as the `savepi` command if <*color-offset*> is specified, except `savepirgb` assumes the first <*color-offset*> lines in the area starting at the first line of the `sep`-type array <*var-name*> to be R data, the next <*color-offset*> lines to be G data, and the <*color-offset*> lines following those to be B data, and saves data to a file in the `ppm` binary format.

The `savepiyc` command is the same as the `savepi` command, except, if <*color-offset*> is not specified for `savepiyc`, it assumes the pixels starting at pixel <*offset*> on the first <*length*> lines in the area starting at the first line of the `sep`-type array <*var-name*> to be y data and the same pixels on the next <*length*> lines to be c data, and saves data to a file in the `ppm` binary format. The `savepiyc` command is also the same as the `savepi` command if <*color-offset*> is specified, except `savepiyc` assumes the pixels starting at pixel <*offset*> on the first <*color-offset*> lines in the area starting at the first line of the `sep`-type array <*var-name*> to be y data and the same pixels on the next <*color-offset*> lines to be c data, and saves data to a file in the `ppm` binary format. Note that, when a `ppm` binary file saved using the `savepiyc` command is reloaded to a `sep`-type array using a command such as `loadpiyc`, there are arithmetic errors during the conversion of the data from YC data, to RGB data, and then back to YC data that result in slight differences from the data originally stored in the `sep`-type array.

The following describes only `savepi`.

Specify the name of the file to which to save the data for <*file-name*>. If <*file-name*> is omitted, or if <*file-name*> is `-`, the file browser starts, and the selection of a file is requested.

The data starting at pixel <*offset*> on each line in the area starting at the first line of the `sep`-type array <*var-name*> is saved to the <*width*> × <*length*> file <*file-name*> in the `pgm` binary format (or the `ppm` binary format for `savepiyc` or `savepirgb`). If a value such as `0:e` is specified for <*var-name*>, it is assumed to be an absolute address in IMEM. If <*offset + width*> exceeds the number of PEs, four lines of data are saved at a time (which contain (*number of PEs* × 4 bytes) each), a newline character is added to the end of the fourth line, and then saving continues. (For details, see the following figure.)



If parameters are omitted, their default values are as follows:

<*file-name*>	The file selection window is generated.
<*var-name*>	`0:i`
<*length*>	The number of lines in the `sep`-type array <*var-name*>
<*offset*>	`0`
<*width*>	Number of PEs - <*offset*>
<*color-offset*>	`VLINES`

Examples > savepi ~/image/sample.pgm In [CR]

(The contents of the IMEM memory starting at the address of the sep-type array In are saved to the file ~/image/sample.pgm in the pgm binary format.)

> savepi - 0x100:i [CR]

(The file name entry window is displayed, and the entry of a file name is requested. The contents starting at the IMEM memory address 256 are saved to the entered file in the pgm binary format.)

> savepi [CR]

(The file name entry window is displayed, and the entry of a file name is requested. The contents starting at the IMEM memory address 0 are saved to the entered file in the pgm binary format.)

4.6.52 savepeblk, savepeblkyc, savepeblkrbg

Format savepeblk [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

savepeblkyc [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

savepeblkrbg [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

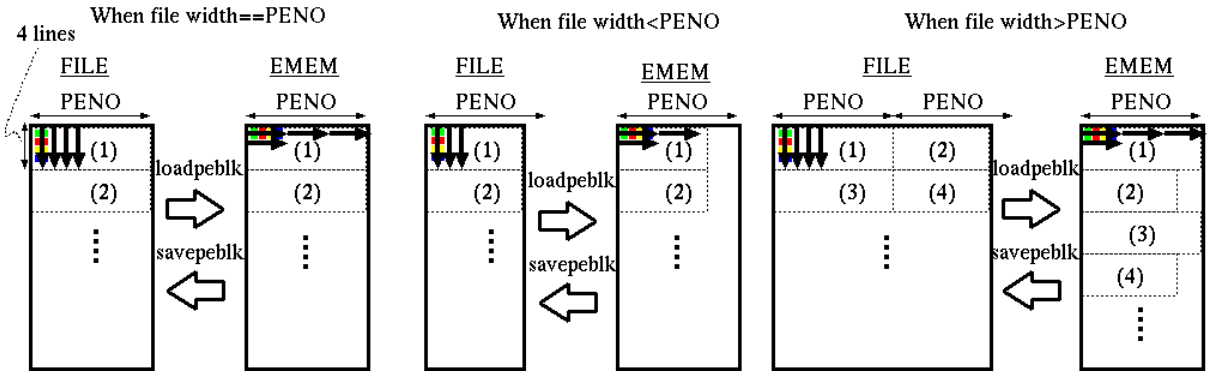
The `savepeblkyc` command is the same as the `savepeblkrgb` command, except `savepeblkyc` converts the luminance data y and the color data c starting at pixel <*offset*> on each line in the area starting at the first line of the `sep`-type array specified by <*var-name*> to RGB data, and then saves this data to <*file-name*> in the `ppm` binary format in the order R, G, and then B.

The `savepeblkrgb` command is the same as the `savepeblk` command, except `savepeblkrgb` saves the RGB data starting at pixel <*offset*> on each line in the area starting at the first line of the `sep`-type array specified by <*var-name*> to <*file-name*> in the `ppm` binary format in the order R, G, and then B.

Therefore, the following describes only `savepeblk`.

Specify the name of the file to which to save the data for <*file-name*>. If <*file-name*> is omitted, or if <*file-name*> is `-`, the file browser starts, and the selection of a file is requested.

Every 4 bytes of the data starting at pixel <*offset*> on each line in the area starting at the first line of the `sep`-type array <*var-name*> are rotated 90 degrees clockwise and then saved to the <*width*> × <*length*> file <*file-name*> in the `pgm` binary format (or the `ppm` binary format for `savep*blkyc` or `savep*blkrgb`). If a value such as `0:e` is specified for <*var-name*>, it is assumed to be an absolute address in EMEM. If <*offset + width*> exceeds the number of PEs, four lines of data are saved at a time (which contain (*number of PEs* × 4 bytes) each), a newline character is added to the end of the fourth line, and then saving continues. (For details, see the following figure.)



If parameters are omitted, their default values are as follows:

<*file-name*>	The file selection window is generated.
<*var-name*>	`0:e`
<*length*>	The value specified in the file
<*offset*>	`0`
<*width*>	The value specified in the file

Examples > savepeblk ~/image/sample.pgm In [CR]

(The data in EMEM starting at the address of the sep-type array In is saved to the ~/image/sample.pgm file, with every 4 bytes rotated 90 degrees clockwise.)

4.6.53 savese

Format savese [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

This command is the same as the savesi command, except the data is saved to an outside sep-type array.

4.6.54 savesi

Format savesi [<file-name>] [<var-name>] [<length>] [<offset>] [<width>]

This command saves the contents of the <*length*> lines starting at the first line of the `sep`-type array <*var-name*> to the file <*file-name*> in the IMAP text format. If a value such as `0:i` is specified for <*var-name*>, it is assumed to be an absolute address in the IMAP memory.

Specify the name of the file to which to save the data for <*file-name*>. If <*file-name*> is omitted, or if <*file-name*> is `-`, the file name entry window is displayed, and the entry of a file name is requested.

If other parameters are omitted, their default values are as follows:

<*var-name*>	`0:i`
<*length*>	`VLINES`
<*offset*>	`0`
<*width*>	Number of PEs - <*offset*>

Examples > savesi ~/image/sample.256.imap In [CR]

(The contents of the IMAP memory starting at the address of the sep-type array In are saved to the ~/image/sample.256 file in the IMAP text file format.)

> savesi . 256:i [CR]

(The file name entry window is displayed, and the entry of a file name is requested. The contents starting at the IMAP memory address 256 are saved to the entered file in the IMAP text format.)

> savesi [CR]

(The file name entry window is displayed, and the entry of a file name is requested. The contents starting at the IMAP memory address 0 are saved to the entered file in the IMAP text format.)

4.6.55 savet

Format savet [<file-name>] [<function-name>] [<length>] [<offset>] [<width>]

This command saves the <*length*> line area in the program memory that starts at the position offset <*offset*> from the starting address of the function <*function-name*> and is <*width*> words wide to the file <*file-name*> in binary format. If a value such as `0:p` is specified for <*function-name*>, it is assumed to be an absolute address in the program memory.

Specify the name of the file to which to save the data for <*file-name*>. If <*file-name*> is omitted, or if <*file-name*> is `-`, the file name entry window is displayed, and the entry of a file name is requested.

This command is used to perform lower level operations. It is not normally necessary to save the contents of program memory to files for executable object files.

If parameters are omitted, their default values are as follows:

<*function-name*>	`0`
<*length*>	`1`
<*offset*>	`0`
<*width*>	`4`

Examples > savet ~/sample/func.io main 100 [CR]

(The 100 lines (of code) starting at the address of the main function are saved to the file ~/sample/func.io in binary format.)

> savet - 0x100 [CR]

(The file name entry window is displayed, and the entry of a file name is requested. The contents starting at the program memory address 256 are saved to the entered file in binary format.)

> savet [CR]

(The file name entry window is displayed, and the entry of a file name is requested. The contents starting at the program memory address 0 are saved to the entered file in binary format.)

4.6.56 scale1d

Format scale1d <number> <var-name> [<start-number>] [<end-number>] [<alias-var-name>][<foreground-color>] [<background-color>]

This command assigns the dynamically specified value of the global variable <*var-name*> to the slider whose ID is <*number*>. If <*number*> is -, the correct ID is automatically assigned in order, starting at 0. If a sep-type global variable is specified, the elements of the corresponding variable in each PE are set to the same value.

If <*alias-var-name*> is specified, it is used as the slider label. If <*alias-var-name*> is not specified, <*var-name*> is displayed as the slider label.

<*start-number*> and <*end-number*> specify the minimum and maximum values of the slider, respectively. The values specified for <*start-number*> and <*end-number*> are assumed to be 32-bit signed integers. <*start-number*> must be less than <*end-number*>, and, if omitted, the values of these parameters are 0 and 255, respectively.

<*foreground-color*> and <*background-color*> can be used to specify the foreground and background colors of the label portion of the slider, respectively.

The scale window is generated by the first execution of this command. If an ID that is already assigned to a different variable is specified for <*number*>, the previous assignment is discarded, a slider that has that ID is newly assigned for use by <*var-name*>, and the label in the scale window is updated.

Examples > scale1d - s 0 4 [CR]

(A slider is generated for which the value of the global variable MY is specified (and to which an ID is automatically assigned).)

(For this slider, the only values that can be specified for s are 0, 1, 2, 3, and 4.)

> scale1d - Thres [CR]

(A slider is generated for which the value of the global variable Thres is specified.)

4.6.57 scale2d

Format scale2d <number> <var-name> [<other-var-name>] [<clik-var-name>] [<alias-name>] [<foreground-color-name>] [<background-color-name>] [<offset-variable-name>]

Examples > scale2d - X Y [CR]

(A 2D scroll map is generated for which the values of the variables X and Y are specified.)

> scale2d 10 X Y Z [CR]

(A symbol for which the values of the variables X and Y are specified is generated in the 2D scroll map whose ID is 10, and the symbol attributes are specified such that 1 is written to the variable Z when the button is clicked, and 0 is written to Z when the button is released.)

> scale2d - X Y - Adjust black orange [CR]

(A symbol for which the values of the variables X and Y are specified that has the label name Adjust is generated in the most recently created 2D scroll map. If the scroll map does not yet exist, a scroll map that has a black foreground and orange background is generated.)

> scale2d 0 X Y - Adjust black orange [CR]

(In addition to the above, a symbol is generated in the 2D scroll map whose ID is 0. If a scroll map whose ID is 0 does not yet exist, a scroll map that has a black foreground and orange background is generated.)

> scale2d - X Y - - - Z [CR]

(A symbol is generated in the 2D scroll map in order to specify values for element Z of the arrays X and Y.)

The operation of this command differs depending on whether <*other-var-name*> is specified.

1. If <*other-var-name*> is specified

This command assigns the values of the global variable <*var-name*> and the global variable <*other-var-name*> to the 2D scroll map whose ID is <*number*>. As a result, a symbol representing both variables is displayed on the scroll map, and, each time the symbol is left-clicked and dragged, the x coordinate of the symbol position is written to <*var-name*> and the y coordinate of the symbol position is written to <*other-var-name*>. The written values are displayed at the upper part of the symbol at the same time. Note that, if `-` is specified for <*number*>, the symbol is generated in the most recently created 2D scroll map.

2. If <*other-var-name*> is omitted

This command assigns the value of the global variable <*var-name*> to the 2D scroll map whose ID is <*number*>. <*var-name*> must have a data type that is 2 bytes or longer (`int` or `long`). As a result of the assignment, a symbol representing the variable is displayed on the scroll map, and, each time the symbol is left-clicked and dragged, a value whose higher byte is the x coordinate of the symbol position and whose lower byte is the y coordinate of the symbol position is written to <*var-name*>. The written values are also displayed at the upper part of the symbol at the same time. Note that, if `-` is specified for <*number*>, the symbol is generated in the most recently created 2D scroll map.

The meanings of the other parameters are as follows:

1. If <*alias-name*> is specified, the name of the symbol displayed on the 2D scroll map is <*alias-name*>. Any character string consisting of English characters can be specified for <*alias-name*>.
2. If <*foreground-color-name*> and <*background-color-name*> are specified and the 2D scroll map whose ID is <*number*> does not yet exist, a 2D scroll map that has the foreground color <*foreground-color-name*> and the background color <*background-color-name*> is generated.
3. If the global variable <*clik-var-name*> is specified, `1` is written to <*clik-var-name*> while the left mouse button is held down over the symbol, and `0` is written to <*clik-var-name*> when the left mouse button is released while over the symbol.
4. If the global variable <*offset-variable-name*> is specified, <*var-name*> (and <*other-var-name*>) are assumed to be starting addresses, the value of <*offset-variable-name*> is assumed to be the offset, and the area addressed by these values is the target for writing by the 2D scroll map. This can be used to specify a value according to the scroll map for a suitable element in an array by specifying the name of the array for <*var-name*> (and <*other-var-name*>) and the name of a global variable containing the index for <*offset-variable-name*>.

4.6.58 scale2dx

Format scale2dx <number> <var-name> [<other-var-name>] [<clik-var-name>] [<foreground-color-name>] [<background-color-name>] [<width>] [<height>] [<offset-variable-name>]

Examples > scale2dx - X Y [CR]

(A 2D pointer map is generated for which the values of the variables X and Y are specifed.)

The operation of this command differs depending on whether <*other-var-name*> is specified.

1. If <*other-var-name*> is specified

This command assigns the values of the global variable <*var-name*> and the global variable <*other-var-name*> to the 2D pointer map whose ID is <*number*>. If `-` is specified for <*number*>, the ID is automatically assigned. If the mouse is moved within the 2D pointer map, the x coordinate of the mouse position is written to <*var-name*> and the y coordinate of the mouse position is written to <*other-var-name*>. The written x and y values are displayed in the center of the pointer map.

2. If <*other-var-name*> is omitted

This command assigns the value of the global variable <*var-name*> to the 2D pointer map whose ID is <*number*>. If `-` is specified for <*number*>, the ID is automatically assigned. <*var-name*> must have a data type that is 2 bytes or longer (`int` or `long`). If the mouse is moved within the 2D pointer map, a 2-byte value whose higher byte is the x coordinate of the mouse position and whose lower byte is the y coordinate of the mouse position is written to <*var-name*>. The written x and y values are displayed in the center of the pointer map.

The meanings of the other parameters are as follows:

1. If <*foreground-color-name*> and <*background-color-name*> are specified, a 2D pointer map that has the ID <*number*>, the foreground color <*foreground-color-name*>, and the background color <*background-color-name*> is generated. If a 2D pointer map that already has the same ID exists, that map is deleted, and then a new map is created.
2. If the global variable <*clik-var-name*> is specified, `1` is written to <*clik-var-name*> while the left mouse button is held down within the map, and `0` is written to <*clik-var-name*> when the left mouse button is released while over the map.
3. If <*width*> and <*height*> are specified, they are assumed to be the width and height of the map to be created.
4. If the global variable <*offset-variable-name*> is specified, <*var-name*> (and <*other-var-name*>) are assumed to be starting addresses, the value of <*offset-variable-name*> is assumed to be the offset, and the area addressed by these values is the target for writing by the 2D pointer map. In other words, a value can be specified according to the pointer map for a suitable element in an array by specifying the name of the array for <*var-name*> (and <*other-var-name*>) and the name of a global variable containing the index for <*offset-variable-name*>.

4.6.59 scalecolor

Format scalecolor [<foreground-color-name>|-] [<background-color-name>]

After executing this command, the previously specified foreground and background colors for the label portion of a 1D slider generated using the scale1d command are changed to <foreground-color> and <background-color>, respectively.

If <foreground-color> and <background-color> are omitted, the previously specified foreground and background colors for the label portion of the 1D slider are displayed.

Examples > scalecolor cyan purple [CR]

(After executing this command, the previously specified foreground and background colors for the label portion of a 1D slider are assumed to be cyan and purple, respectively.)

> scalecolor [CR]

(The previously specified foreground and background colors for the label portion of a 1D slider are displayed.)

4.6.60 sel

Format sel <number> <var-name> [<candidate-number>+] [<foreground-color>] [<background-color>]

This command assigns the dynamically specified value of the global variable <*var-name*> to the check box whose ID is <*number*>. If <*number*> is -, the correct ID is automatically assigned in order, starting at 0. If a sep-type global variable is specified, the elements of the corresponding variable in each PE are set to the same value.

<*candidate-number*> is a candidate value that can be assigned to <*var-name*>, and one or more such candidates can be specified. If <*candidate-number*> is omitted, it is assumed that the generation of a check box has been specified. To specify <*foreground-color*> and <*background-color*> while omitting <*candidate-number*>, just omit <*candidate-number*>.

<*foreground-color*> and <*background-color*> can be used to specify the foreground and background colors of the label portion of the check box, respectively.

The sel window is generated by the first execution of this command. If an ID that is already assigned to a different variable is specified for <*number*>, the previous assignment is discarded, a check box that has that ID is newly assigned for use by <*var-name*>, and the label in the sel window is updated.

Examples > sel - s 0 1 2 14 [CR]

(A check box is generated for which the value of the global variable s is specified (and to which an ID is automatically assigned).)

(For this check box, the only values that can be specified for s are 0, 1, 2, and 14)

> sel - Thres [CR]

(A check box is generated for which only 0 or 1 can be specified for the global variable Thres.)

4.6.61 selcolor

Format selcolor [<foreground-color-name>|-] [<background-color-name>]

After executing this command, the previously specified foreground and background colors for the label portion of a check box generated using the `sel` command are changed to <*foreground-color*> and <*background-color*>, respectively.

If <*foreground-color*> and <*background-color*> are omitted, the previously specified foreground and background colors for the label portion of the check box are displayed.

Examples > selcolor cyan purple [CR]

(After executing this command, the previously specified foreground and background colors for the label portion of a check box are assumed to be cyan and purple, respectively.)

> selcolor [CR]

(The previously specified foreground and background colors for the label portion of a check box are displayed.)

4.6.62 setv

Format setv [<value>] [<var-name>] [<line-number-or-function-name>]

This command writes the value <value> to the memory where the variable <var-name> exists. For a `sep`-type variable, the same value <value> is specified for the elements in all PEs.

Specify a number or character string for <value>. If a character string is specified, each character code in the string is inserted into <var-name>, in bytes, starting at the first address of <var-name>, regardless of the number of bytes in each element of <var-name>, and a `0` is inserted at the end.

Examples `> setv 10 thres [CR]`

`10` is specified for the local variable `thres` in the current function if such a variable exists, or for the global variable `thres` otherwise.

`> setv cloudy-scene 0: d [CR]`

12 bytes of character codes followed by a `0` are assigned starting at the DMEM address 0.

4.6.63 sfunc

Format sfunc <function-name> [<start-line-number>] [<end-line-number>]

First, starting at the stop position in the current source code, the program is executed in the step mode either until the end of the first execution of the function <function-name> or, if <end-line-number> is specified, until line <end-line-number>. Based on the execution results, either the processing time from line <start-line-number> of the function to the line before line <end-line-number> if <start-line-number> and <end-line-number> are specified or the processing time from the first line of the function to the line before the last line of the function if <start-line-number> and <end-line-number> are not specified is displayed in the command window when the program stops. However, note that this command does not count the execution time or the number of processing steps for background processing, such as the ememrd and ememwr functions, or the number of processing steps due to program cache misses or data cache misses.

Examples > sfunc labeling [CR]

labeling() Line 42...52: 229987steps, 5.750msec (24076-254063)

...Stopped at labeling.lc:52, [6.35(+0.00)ms] 254063(+0)steps

> load testbin [CR]

> sfunc [CR]

(The view window is opened, and the processing time for each function called in testbin.ix is displayed.)

4.6.64 sline

Format sline <function-name> [<start-line-number>] [<end-line-number>]

First, starting at the stop position in the current source code, the program is executed in the still mode (in which video interrupts are prohibited) either until the end of the first execution of the function <function-name> or, if <end-line-number> is specified, until line <end-line-number>. Based on the execution results, either the processing time from line <start-line-number> of the function to the line before line <end-line-number> if <start-line-number> and <end-line-number> are specified or the processing time from the first line of the function to the line before the last line of the function if <start-line-number> and <end-line-number> are not specified is displayed in the command window when the program stops, and information such as the processing time for each line and the proportion of the time used by all such processing is displayed in the view window. A timing window is created at the same time, which displays the above execution results as a bar graph in which the bar for the line of code that had the longest processing time is displayed in red. However, note that this command does not count the execution time or the number of processing steps for background processing, such as the ememrd and ememwr functions, or the number of processing steps due to program cache misses or data cache misses.

Note that, for an executable file compiled with the –m or –O option specified, the processing time for each line displayed by the sline command might not correctly correspond to the each line in the program because the code is optimized.

4.6.65 step

Format step [<number>]

Starting at the current stop position, this command executes the next <number> lines at which breakpoints can be specified in the still mode (in which video interrupts are prohibited). If there are function calls on those lines and source code for the functions exists, the debugger steps into those functions. If <number> is omitted, the debugger proceeds only to the next line at which a breakpoint can be specified is reached.

Note the following exception: if the current position is in a file compiled using the –db3 or –db option, the debugger does not step into functions compiled using options such as –db1 or –m. To step into such a function in this case, execute the sstep command from the command prompt, not the step command.

If the current position is in a function compiled using options such as –db1 or –m, the step command cannot be used.

Examples

```
> step [CR]
... Stopped at line: tst.lc: 51
> step 3 [CR]
... Stopped at line: tst.lc: 54
> step [CR]
... Stopped at line: tst.lc: 55
```

4.6.66 stop

Format stop

This command stops the video I/O and executes a command reset of the board. This command can be used to stop programs run using runv.

4.6.67 symbols

Format symbols

This command displays the addresses of all the functions and global variables in the currently loaded executable object file in the view window.

4.6.68 thumbnail

Format thumbnail <var-name> [<line-number-or-function-name>] [<number-of-lines>] [<color>] [<offset>] [<width>] [<blk>]

This command uses the win command to generate the image window used to display the <var-name> array as a thumbnail. An image window displayed as a thumbnail can be restored to its original size by double-clicking the displayed image.

<blk> is valid only if the target display area is in EMEM and is used to specify the window display mode as shown in the following table.

Specified value of `blk`	Displayed window width
0	Displays the image using a width equivalent to the number of PEs (default)
-	Displays the image using the width specified by <*width*>
1	Displays the image using the width specified by <*width*> after rotating every 4 bytes of data in the memory 90 degrees clockwise

4.6.69 timerint

Format timerint [<interval_in_usec>|on|off] [<interrupt vector No.>] [<interrupt function name>]

This command sets up, starts, or stops timer interrupts.

If the first parameter is a number, it is assumed to be the interrupt interval in μs and parameters 2 and 3 are required. For parameter 2 (<interrupt vector No.>), specify the interrupt vector number, and, for parameter 3 (<interrupt function name>), specify the name of the interrupt function. If the first parameter is `on` or `off`, timer interrupts are enabled or disabled, respectively. If a value other than `on` or `off` is specified for the first parameter, the current timer interrupt settings are displayed.

Note that simply setting up timer interrupts does not start them. To use timer interrupts, be sure to start them by using this command again after using this command to set them up.

Examples > timerint 50 15 int_func [CR]

(Timer interrupts are set up to occur every 50 μs at interrupt vector 15, and int_func is registered as the interrupt function for that vector.)

> timerint on [CR]

(Timer interrupts are started.)

4.6.70 tofile

Format tofile <command> [<file-name> <view-or-not>]

Normally, the results of executing sdbimap commands are output to the standard output (stdout or the command window for commands executed at the command window prompt), but this command can be used to specify that the results normally output to the standard output while the sdbimap command specified by <command> executes be written to the file specified by <file-name> instead. (If the specified file already exists, the results are appended to the file.) If <file-name> is omitted, the file tofile_sdbimap.txt is created in the temporary directory, and the execution results are output to it.

The default value of <view-or-not> is `0`, but, if `1` is specified for it, the contents of the file <file-name> are displayed in the view window after <command> executes. To omit the <file-name> specification and specify only <view-or-not>, specify `.` for <file-name>.

Examples > tofile “help” help_display.txt [CR]

(The results of executing the help command are written to the file help_display.txt.)

4.6.71 touch

Format touch

This command is executed following the wscale, wsel, and wscale2d commands. This command is useful when performing a reload after revising and recompiling a program for applying all the values in the corresponding locations in memory to the values displayed for 1D sliders and other value adjustment GUI tools before the reload.

4.6.72 until

Format until <var-name> <compare-symbol> <var-name or immediate>

This command is used to continue processing until the variable <var-name> has the relationship with the variable or immediate value <var-name or immediate> specified by <compare-symbol>. Note that, if a breakpoint is reached during such processing, processing stops regardless of whether the relationship exists. Only the symbols <=, >=, <, >, ==, and != can be specified for <compare-symbol>.

Examples > until i > 23 [CR]

(Processing continues until the value of the variable i is greater than 23.)

> until a <= b [CR]

(Processing continues until the value of the variable a is less than or equal to b.)

4.6.73 view

Format view [<file-name-or-func-name>]

If the text file specified by <file-name-or-func-name> exists, this command reads the file and displays it in the view window. If <file-name-or-func-name> does not seem to exist, this command searches the current source code for the file or function specified by <file-name-or-func-name>, and, if found, displays the source file or the source file containing the function in the view window. If <file-name-or-func-name> is omitted, the file browser is displayed, and the selection of a file is requested.

Examples > view ~/tmp/sample.lc [CR]

(The view window is generated, and the contents of ~/tmp/sample.lc are displayed.)

> view [CR]

(The file browser is displayed, and the selection of a file is displayed.)

4.6.74 viewer

Format viewer [<editor-or-viewer-name-and-path>]

This command is used to specify whether to use the sdbimap-specific viewer or a user-specified viewer (an editor) when the view window is generated, such as by view commands.

If – is specified for <editor-or-viewer-name-and-path>, the specification of the sdbimap-specific view window is assumed. If <editor-or-viewer-name-and-path> is omitted, the currently specified viewer information is displayed.

Examples > viewer [CR]

Current viewer / editor is "the default sdbimap viewer".

> viewer c: /windows/notepad.exe [CR]

(The use of c: /windows/notepad.exe for the view window is specified.)

> viewer - [CR]

(The use of the sdbimap-specific viewer for the view window is specified.)

4.6.75 where

Format where

This command displays information such as the current program counter value and the name of the function in the corresponding source code.

Examples :> where [CR]

In main() after 1231 steps [0x9d6c->0x0ea5]

In the above example, the first number in the square brackets is the starting address of the main function, and the following number is the current program counter value.

4.6.76 whereis

Format whereis

This command displays the position of the current assembly instruction.

Examples > whereis [CR]

4.6.77 win

Format win [<var-name>] [<line-number-or-function-name>] [<number-of-lines>] [<color>] [<offset>] [<width>] [<blk>]

If <var-name> is a `sep`-type variable or array, this command generates the image window used to display the memory for <var-name>. If <number-of-lines> is specified, the image window is initialized so as to display <number-of-lines> lines of the image.

If <var-name> is a `uni`-type variable or array, this command generates the dmem window used to display the memory for <var-name>. In this case, <number-of-lines> indicates the number of elements to display, and <color> and <offset> cannot be specified.

For <color>, `grey` or `yc` can be specified, and the image described below is displayed in each case. Note that, if `yc` or `rgb` is specified for <color>, the value specified for <number-of-lines> cannot be greater than <offset>. Furthermore, a specification for <offset> is only meaningful if `yc` is specified.

1. If `grey` is specified, the <*number-of-lines*> line 8-bit grayscale image from <*var-name*> is displayed.
2. If `yc` is specified, the first <*offset*> lines from <*var-name*> are assumed to be luminance information (y), the next <*offset*> lines are assumed to be color information (CrCb), and a <*number-of-lines*> line 24-bit color image is displayed.
3. If `rgb` is specified, the first <*offset*> lines from <*var-name*> are assumed to be R signal data, the next <*offset*> lines are assumed to be G signal data, the following <*offset*> lines are assumed to be B signal data, and a <*number-of-lines*> line 24-bit color image is displayed. (However, this specification is not currently supported.)

<*width*> specifies the width of the image to be displayed in pixels.

<*blk*> is valid only if the target display area is in EMEM and is used to specify the window display mode as shown in the following table.

Specified value of `blk`	Displayed window width
0	Displays the image using a width equivalent to the number of PEs (default)
-	Displays the image using the width specified by <*width*>
1	Displays the image using the width specified by <*width*> after rotating every 4 bytes of data in the memory 90 degrees clockwise

If <*number-of-lines*> is omitted and <*var-name*> is an array, the number of lines initially displayed in the image window is specified depending on both the array size and the value specified for <*color*> as shown below. Below, *N* represents the total array size.

1. If `grey` is specified, either *N* lines or <*offset*> × 6 lines are displayed, whichever is less.
2. If `yc` is specified, either *N*/2 or <*offset*> lines are displayed, whichever is less.
3. If `rgb` is specified, either *N*/3 or <*offset*> lines are displayed, whichever is less. (However, this specification is not currently supported.)

Menus are also displayed at the top of the generated image window that enable you to do the following:

1. **File** menu

a) Save a displayed image to a file.

b) Load an image file into the displayed area.

c) Open a character window corresponding to the displayed image.

2. **Mode** menu

a) Specify whether to normally perform refreshes.

b) Specify the magnification for a displayed image.

c) Specify the PE multiplexing rate (multiple pixels per PE).

c) Specify the displayed colors.

e) Change the number of displayed lines.

g) Change the starting display address.

h) Change the number of lines to display.

i) Specify the display method (such as a thumbnail, normal display, or aligning the window size with an image that has a size of 0).

There is a shortcut button for the `win` command (labeled **win**) at the top of the source window. To call this command using the button, use the mouse to select an array, and then click the button.

If parameters are omitted, their default values are as follows:

<var-name>	`0:e`
<line-number-or-function-name>	`0`
<number-of-lines>	Number of lines in the array
<color>	`grey`
<offset>	`VLINES`
<width>	Number of PEs
<blk>	`0`

Examples > win csum [CR]

(An image window is opened that displays a grayscale image of the external or IMAP memory indicated by the csum pointer that points to the sep-type area. If csum is a pointer, 240 lines are initially displayed.)

> win 120: i - 100 [CR]

(An image window is opened that displays a grayscale image of the addresses in the IMAP memory from 120 to 220.)

> win 120: e - 50 yc 100 [CR]

(An image window is opened that displays a color image of only the first 50 lines of the color image whose luminance signal y is stored at addresses 120 to 220 and whose color signal CrCb is stored at addresses 220 to 320.)

4.6.78 winop

Format winop <op-name (c | u | d | D | n | r | R)> [<memory-type (c | r | p | d | i | e)>] [<start-address>] [<length>] [o <offset>] [w <width>]

This command performs the <*op-name*> operation on the memory window of the type specified by <*memory-type*> (a window that displays memory contents as text).

As shown below, the English character `c`, `u`, `d`, `D`, `a`, `n`, `r`, or `R` can be specified for <*op-name*>, and the English character `c`, `r`, `p`, `d`, `i`, or `e` can be specified for <*memory-type*>. If <*memory-type*> is omitted, the default value is `i`.

<*op-name*>:

`c | create`	Generates the window.
`u | popup`	Restores the window.
`d | popdown`	Minimizes the window.
`D | destroy`	Closes the window.
`a | auto`	Starts automatic updating.
`n | no auto`	Stops automatic updating.
`r | draw`	Displays the window.
`R | refresh`	Refreshes the window.

<*memory-type*>:

`c | creg`	RVSC register window
`r | ireg`	IMAP register window
`p | pmem`	Program memory window
`d | dmem`	Data memory window
`i | imem`	IMAP memory window
`e | emem`	External memory window

If <*op-name*> is `c`, <*start-address*> and <*length*> can be specified. In this case, a memory window used to display the <*length*> byte area starting at the address <*start-address*> (whose default value is `0`) is generated. If a variable is specified for <*start-address*>, the default value for <*length*> is the number of bytes from the address of that variable to the end of the memory for that variable, and, if an immediate value is specified for <*start-address*>, the default value is one byte (or one line for imem or emem). Note that a specification for <*start-address*> or <*length*> is valid only if <*memory-type*> is `p`, `d`, `i`, or `e`. Similarly, a specification for <*offset*> or <*width*> is valid only if <*memory-type*> is `i` or `e`.

Examples : > winop c d [CR]

(A memory window that displays the data memory is opened.)

> winop c d 256 1024 [CR]

(A memory window that displays the 1,024 bytes starting at the data memory address 256 is opened.)

> winop R d [CR]

(A memory window that displays the data memory is refreshed.)

> winop u i [CR]

(All currently open memory windows that display the IMAP memory are restored.)

> winop c i src 10

(A memory window that displays the first 10 elements of the global sep-type array src is opened.)

> winop c i src[10] 10

(A memory window that displays the 10 elements starting at the eleventh element of the global sep-type array src (src[10]) is opened.)

4.6.79 wscale

Format wscale [<number>]

This command writes the value equivalent to the slider position of the 1D slider that has the number specified by <number> to the memory for the global variable currently assigned to that slider. If <number> is omitted, the above processing is performed for all 1D sliders that exist.

Examples :> wscale 0 [CR]

(The value equivalent to the slider position of the 1D slider that has the ID 0 is written to the memory for the corresponding global variable.)

> wscale [CR]

(The values equivalent to the slider positions of all 1D sliders are written to the memory for the corresponding global variables.)

4.6.80 wscale2d

Format wscale2d [<number>]

This command writes the values equivalent to the positions of each symbol in the 2D scroll map that has the number specified by <number> to the memory for the corresponding (set of) global variables. If <number> is omitted, the above processing is performed for the symbols in all 2D scroll maps that exist.

Examples :> wscale2d 0 [CR]

(The values equivalent to the positions of each symbol in the 2D scroll map that has the ID 0 are written to the memory for the corresponding (set of) global variables.)

> wscale2d [CR]

(The values equivalent to the positions of each symbol in all 2D scroll maps are written to the memory for the corresponding (sets of) global variables.)

4.6.81 wsel

Format wsel [<number>]

This command writes the value equivalent to the value selected using the check box that has the number specified by <number> to the memory for the global variable currently assigned to that check box. If <number> is omitted, the above processing is performed for all check boxes that exist.

Examples : > wsel 0 [CR]

(The value equivalent to the value selected using the check box that has the ID 0 is written to the memory for the corresponding global variable.)

> wsel [CR]

(The values equivalent to the values selected using all check boxes are written to the memory for the corresponding global variables.)

5 Revision history

Version	Date	Document Number	Description
1.0	July 2009	U19917EE1V0UM00	First version

The following revision list shows all functional changes compared to the previous version.

Chapter	Page	Description