

SANKHYA Tools Collection

User Guide and Reference Manual

THIS DOCUMENT CONTAINS PROPRIETARY INFORMATION OF SANKHYA TECHNOLOGIES PRIVATE LIMITED. Use, duplication and disclosure are subject to license restrictions.

(C) Copyright 2004 Sankhya Technologies Private Limited

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means viz., electronic, mechanical, photo-copying, recording, or otherwise, without the prior consent of the publisher.

SANKHYA, SANKHYA TECHNOLOGIES, SANKHYA Tools Collection, Dynamically Targetable Tools Framework, SANKHYA Software are Trademarks, Service Marks or Registered Trademarks of Sankhya Technologies Private Limited. All other brands and names are the property of their respective owners.

Part No. 10020101-006

SANKHYA Tools Collection

User Guide and Reference Manual

Sankhya Technologies Private Limited

Part No. 10020101-006

Table 1: Revision History

Revision number	Revision History	Date
001	Updated for STC 1.0 Beta-1	04 Oct 2002
002	Merged MIPS and ARM	11 Feb 2003
003	Updated Sankhya Librarian	20 Mar 2003
004	Updated Icode specification	01 Apr 2003
005	Fixed the defects for Beta-1	28 Apr 2003
006	Updated for STC 1.0 Beta-2	07 Jul 2003

Preface

SANKHYA Tools Collection (STC) is a collection of ready-to-use software development tools built using Sankhya's DTF Technology (Refer Appendix B)

The document consists of two parts :

Part-1 *User Guide* provides the user with the description on how to invoke the code generator, assembler, linker, simulator librarian and dumper. It also provides the Intermediate code specification.

Part-2 *Reference Manual* provides the user with the features of the STC tools, Assembler syntax and directives and the Simulator commands.

Notational Conventions

The guide follows the following conventions

- % - The 'percentage' sign denotes a Unix environment.
- > - The 'greater than' symbol represent a DOS/Windows environment.
- Italic* - The words given in italics represent an option.
- `code` - The guide differentiates the normal text from a program code through this color. Any code or part of a code/command line statements in this document will therefore be represented using this color.
- ... - Indicates that some portion of the material has been removed to simplify the description.
- [] - Indicates an optional argument that can be used in the command line.

Contents

Preface

Part 1 - User Guide	1
1.1 Introduction	1
1.1.1 Host platforms supported.....	3
1.1.2 Setting up the environment variables.....	3
1.2 SANKHYA Code Generator	5
1.2.1 Introduction.....	5
1.2.2 Invoking SANKHYA Code Generator for MIPS (scgmips)	5
1.2.3 Invoking SANKHYA Code Generator for ARM (scgarm).....	8
1.3 SANKHYA Assembler	12
1.3.1 Introduction.....	12
1.3.2 Invoking SANKHYA Assembler for MIPS (sasmips).....	12
1.3.3 Invoking SANKHYA Assembler for ARM (sasmarm).....	16
1.4 SANKHYA Linker	20
1.4.1 Introduction.....	20
1.4.2 Invoking SANKHYA Linker for MIPS (sldmips).....	20
1.4.3 Invoking SANKHYA Linker for ARM (sldarm).....	23
1.5 SANKHYA Simulator	27
1.5.1 Introduction.....	27
1.5.2 Invoking SANKHYA Simulator for MIPS (ssimmips).....	27
1.5.3 Invoking SANKHYA Simulator for ARM (ssimarm).....	31
1.6 SANKHYA Librarian	35
1.6.1 Introduction.....	35

1.6.2	Invoking SANKHYA Librarian (slib)	35
1.7	SANKHYA Dumper	42
1.7.1	Introduction.....	42
1.7.2	Invoking SANKHYA Dumper	42
1.7	Using SANKHYA Tools Collection	50
1.7.1	Usage Example - ‘swap’	50
 Part 2 - Reference Manual		53
2.1	SANKHYA Code Generator	53
2.1.1	Overview.....	53
2.1.2	SANKHYA Code Generator features.....	53
2.1.3	SANKHYA Intermediate code	53
2.1.3.1	Tokens.....	54
2.1.3.2	Variables	57
2.1.3.3	Types qualifier	59
2.1.3.4	Attributes.....	60
2.1.3.5	Pointer reference and dereference.....	67
2.1.3.6	Labels.....	68
2.1.3.7	Operands	68
2.1.3.8	Operators.....	70
2.1.3.9	Instructions.....	90
2.1.3.10	Expressions	96
2.1.3.11	Functions.....	98
2.2	SANKHYA Assembler	105
2.2.1	Overview.....	105
2.2.2	SANKHYA Assembler features	105
2.2.3	Assembler syntax	105
2.2.4	Statements	106
2.2.5	SANKHYA Assembler Directives	107
2.2.5.1	.text.....	107

2.2.5.2	.data	107
2.2.5.3	.align	108
2.2.5.4	.space	108
2.2.5.5	.word	108
2.2.5.6	.half	109
2.2.5.7	.4byte	109
2.2.5.8	.byte	109
2.2.5.9	.2byte	110
2.2.5.10	.size	110
2.2.5.11	.comm	110
2.2.5.12	.extern	111
2.2.5.13	.type	111
2.2.5.14	Description:	111
2.2.5.15	.ascii/.asciz	111
2.2.5.16	.section	112
2.2.5.17	.file	112
2.2.5.18	.equiv	113
2.2.5.19	.lcomm	113
2.2.5.20	.p2align	113
2.2.5.21	.skip	114
2.2.5.22	.short	114
2.2.5.23	.hword	114
2.2.5.24	.err	115
2.2.5.25	.equ	115
2.2.5.26	.abort	115
2.2.5.27	.globl	116
2.2.5.28	.end	116
2.2.5.29	.ent	116
2.2.5.30	.gpword	117
2.2.5.31	.sdata	117
2.2.5.32	.rdata	117
2.2.5.33	.global	118
2.2.5.34	.code	118
2.2.5.35	.thumb	119
2.2.5.36	.arm	119

	2.2.5.37 .balign.....	119
	2.2.5.38 .set.....	120
2.3	SANKHYA Linker	121
2.3.1	Overview.....	121
2.3.2	SANKHYA Linker features.....	121
2.4	SANKHYA Simulator	122
2.4.1	Overview.....	122
2.4.2	SANKHYA Simulator features.....	122
2.4.3	Simulator Commands	122
2.5	SANKHYA Librarian	126
2.5.1	Overview.....	126
2.5.2	SANKHYA Librarian features	127
2.6	SANKHYA Dumper	128
2.6.1	Overview.....	128
2.6.2	SANKHYA Dumper features	128
 Part 3 - Appendix A - SANKHYA Code Generator Error Codes		130
 Part 4 - Appendix B - DTF Technology		132
 Part 5 - Appendix C - SANKHYA Librarian Error Messages.....		133
 Index		

USER GUIDE

Sankhya Technologies Private Limited

Part 1 - User Guide

1.1 Introduction

In the current market scenario, embedded developers need a cost effective, single-stop solution for all their development needs to achieve increased efficiency and to reduce the overall product development life cycle.

Sankhya Tools Collection is a package of Software Development Tools that accelerates the development of embedded enterprise applications by providing a set of user-friendly and quickly configurable tools.

Sankhya Tools Collection comes with the following tools needed to build and debug a complete embedded application

- Front-end** - Transforms a program written in high level language (such as 'C') to an intermediate representation of the code
- Code Generator** - Transforms the intermediate representation to assembly code
- Assembler** - Translates assembly source files to the respective object files
- Linker** - Links the various object files and libraries to build an application
- Simulator** - Simulates the hardware or processor thereby enabling to debug an application.

- Librarian** - groups any number of input object files in ELF format and generates a single archive file.
- Dumper** - displays information about the input object file or archive files.

SANKHYA Tools Collection is primarily targeted towards embedded systems developers, CPU designers and vendors and System-On-Chip developers.

This release is for MIPS (MIPS32-ISA) target. This release includes *scgmips*, *sasmmips*, *sldmips*, *ssimmips* and *sdumpmips*

- scgmips** - SANKHYA Code generator for MIPS
- sasmmips** - SANKHYA Assembler for MIPS
- sldmips** - SANKHYA Linker for MIPS
- ssimmips** - SANKHYA Simulator for MIPS
- sdumpmips** - SANKHYA Dumper for MIPS

This release is for ARM (ARMv4-ISA) target. This release includes *scgarm*, *sasmarm*, *sldarm*, *ssimarm* and *sdumparm*.

- scgarm** - SANKHYA Code generator for ARM
- sasmarm** - SANKHYA Assembler for ARM
- sldarm** - SANKHYA Linker for ARM
- ssimarm** - SANKHYA Simulator for ARM
- sdumparm** - SANKHYA Dumper for ARM

This release also includes Librarian which is target independent.

- slib** - SANKHYA Librarian

1.1.1 Host platforms supported

The following host platforms are supported

- Solaris 2.7
- Red Hat Linux 7.2
- Windows NT/2000

1.1.2 Setting up the environment variables

To work with STC tools, the following environment variables need to be set.

- `STC_HOME` - Directory path pointing to the root of installation

For Unix hosts:

`STC_HOME` is `<INSTALL_DIR>`

For Windows hosts:

`STC_HOME` is `%INSTALL_DIR%`

where `<INSTALL_DIR>` is the directory where STC tools are installed.

By default STC, is installed under `/opt/sankhya/stc` in Unix and `C:\sankhya\stc` in Windows NT/2000.

In order to set the environment variable and PATH variable on Unix, source the Unix shell script `stc.csh` in the STC tools installation root directory.

```
% source <STC_HOME>/stc.csh
```

On Windows host, run *stc.cmd*, in the STC tools installation directory. In a Windows Command Prompt, type

```
> %STC_HOME%\stc.cmd
```

1.2 SANKHYA Code Generator

1.2.1 Introduction

SANKHYA Code Generator (*scg*) is the code generator built on Sankhya's DTF technology (Refer Appendix B), which generates GNU style assembly file.

1.2.2 Invoking SANKHYA Code Generator for MIPS (*scgmips*)

SANKHYA Code Generator for MIPS (*scgmips*) can be invoked from the command line as mentioned below. The Code generator takes the intermediate code `<file>.i`, as the input, generated either by an ANSI C or C++ front-end.

Syntax :

On Unix hosts :

```
% scgmips <infile> [-h] [-V] [-s syntax] [-p proc] [-o outfile]
  [-m mdfile] [-ma abi_mdfile] [-mx tx_mdfile] [-ms syn_mdfile]
  [-mo opt_mdfile]
```

On Windows hosts:

```
> scgmips.exe <infile> [-h] [-V] [-s syntax] [-p proc] [-o outfile]
  [-m mdfile] [-ma abi_mdfile] [-mx tx_mdfile] [-ms syn_mdfile]
  [-mo opt_mdfile]
```

Description :

The following is the description of the syntax.

SYNTAX	DESCRIPTION
scgmips	Invokes the code generator
<infile>	Input intermediate file
-o <outfile>	Specifies the output file [default : <i>infile</i> with default extension]
-s <syntax>	Specifies assembler syntax [default: gcc]
-V	Displays version information
-h	Displays help message
-p proc	Specifies the Processor variant
-m <mdfile>	Input processor description file.
-ma <abi_mdfile>	Input processor ABI model file.
-mx <tx_mdfile>	Input transformation model file.
-ms <syn_mdfile>	Input Syntax model file
-mo <opt_mdfile>	Input optimization model file.

Default filename extension :

SANKHYA Code Generator takes the following as the default extensions. In case, the output file is not specified, the output filename is the same as the input filename.

TABLE 1. Code generator Filename extensions

FILE	UNIX	WINDOWS
Input Intermediate file	.i	.i
Output Assembly file	.s	.s

Usage examples :

On Unix hosts:

```
% scgmips test.i
```

On Windows hosts:

```
> scgmips.exe test.i
```

On running the above command, the output file test.s is generated. Here, test.i is the input intermediate file.

```
% scgmips test.i -p MIPS32 -o test.s
```

On running the above command, the output file test.s is generated for the 'MIPS32' variant.

```
% scgmips test.i -s gcc -o test.s
```

On running the above command, the output file test.s is generated for ‘gcc’ assembler syntax.

```
% scgmips -h
```

On running the above command, the banner, the version information and the syntax for using *scgmips* along with all the options, is displayed.

```
% scgmips -V
```

On running the above command, the banner and the version information for *scgmips* is displayed.

1.2.3 Invoking SANKHYA Code Generator for ARM (*scgarm*)

SANKHYA Code Generator for ARM (*scgarm*) can be invoked from the command line as mentioned below. The Code generator takes the intermediate code <file>.i, as the input, generated either by an ANSI C or C++ front-end.

Syntax :

On Unix hosts :

```
% scgarm <infile> [-h] [-V] [-s syntax] [-p proc] [-o outfile]  
[-m mdfile] [-ma abi_mdfile] [-mx tx_mdfile] [-ms syn_mdfile]  
[-mo opt_mdfile]
```

On Windows hosts:

```
> scgarm.exe <infile> [-h] [-V] [-s syntax] [-p proc] [-o outfile]
  [-m mdfile] [-ma abi_mdfile] [-mx tx_mdfile] [-ms syn_mdfile]
  [-mo opt_mdfile]
```

Description :

The following is the description of the syntax.

SYNTAX	DESCRIPTION
scgarm	Invokes the code generator
<infile>	Input intermediate file
-o <outfile>	Specifies the output file [default : <i>infile</i> with default extension]
-s <syntax>	Specifies assembler syntax [default: gcc]
-V	Displays version information
-h	Displays help message
-p proc	Specifies processor variant
-m <mdfile>	Input processor description file.

- ma <abi_mdfile> Input processor ABI model file.
- mx <tx_mdfile> Input transformation model file.
- ms <syn_mdfile> Input syntax model file
- mo <opt_mdfile> Input optimization model file

Default filename extension :

SANKHYA Code Generator takes the following as the default extensions. In case, the output file is not specified, the output filename is the same as the input filename.

TABLE 1. Code generator Filename extensions

FILE	UNIX	WINDOWS
Input Intermediate file	.i	.i
Output Assembly file	.s	.s

Usage examples :

On Unix hosts:

```
% scgarm test.i
```

On Windows hosts:

```
> scgarm.exe test.i
```

On running the above command, the output file test.s is generated. Here, test.i is the input intermediate file.

```
% scgarm test.i -p ARMv4 -o test.s
```

On running the above command, the output file test.s is generated for the 'ARMv4' variant.

```
% scgarm test.i -s gcc -o test.s
```

On running the above command, the output file test.s is generated for 'gcc' assembler syntax.

```
% scgarm -h
```

On running the above command, the banner, the version information and the syntax for using *scgarm* along with all the options, is displayed.

```
% scgarm -V
```

On running the above command, the banner and the version information for *scgarm* is displayed.

1.3 SANKHYA Assembler

1.3.1 Introduction

An assembler is a program that takes a file containing assembly instructions and assembler directives as the input and produces an object file as the output.

SANKHYA Assembler for (*sasm*) is a two-pass assembler. *sasmmips* assembles GNU style assembler files into relocatable ELF object files.

1.3.2 Invoking SANKHYA Assembler for MIPS (*sasmmips*)

SANKHYA Assembler for MIPS (*sasmmips*) can be invoked from the command line as mentioned below. It takes the assembly file *infile*, generated either by the SANKHYA Code generator for MIPS target (*scgmips*) or by the gcc compiler, as the input.

Usage :

On Unix hosts:

```
% sasmmips <infile> [-h] [-V] [-s syntax] [-E endian ] [-p proc]
[-o outfile] [-m mdfile] [-ms syntax_mdfile] [-mr reloc_mdfile]
[-md file] [--nodelayslot]
```

On Windows hosts:

```
> sasmmips.exe <infile> [-h] [-V] [-s syntax] [-E endian] [-p proc]
[-o outfile] [-m mdfile] [-ms syntax_mdfile] [-mr reloc_mdfile]
[-md file] [--nodelayslot]
```

Description :

The following is the description of the syntax.

SYNTAX	DESCRIPTION
sasmmips	Invokes the Assmebler
<infile>	Input Assembly file
-o <outfile>	Specifies output file name [default : <i>infile</i> with default extension]
-E <endian>	Specifies endianness. Takes either 'little' or 'big' as its value. [default : Big endian]
-s <syntax>	Specifies assembler syntax. [default : gcc]
-p <proc>	Specifies target variant
-V	Displays version information
-h	Displays help message
-m <mdfile>	Input processor description file
-ms <syntax_mdfile>	Input Syntax mdfile

-mr <relocmdfile>	Input processor relocation model file
-md <file>	Input delay slot description file
--nodelay_slot	Do not place nops in delay slots

Default filename extension :

SANKHYA Assembler takes the following as the default extensions. In case, the output filename is not specified, the output filename is the same as the assembly input filename.

TABLE 2. Assembler Filename extensions

FILE	UNIX	WINDOWS
Assembly source file	.s	.s
Relocatable object file	.o	.o

Usage example :

On Unix hosts:

```
% sasm MIPS test.s
```

On Windows hosts:

```
> sasm MIPS.exe test.s
```

On running the above command, the output file test.o is generated. Here, test.s is the input source file.

```
% sasmmps -s gcc -E little -o test1.o test.s
```

where,

- test.s - assembly file generated by the code generator
- E <little> - target endianness will be set to 'little'
- s <gcc> - 'gcc' syntax will be used
- o test1.o - Object file will be 'test1.o'

On running the above command, the output file test1.o is generated for 'gcc' assembler syntax for little endian target.

```
% sasmmps test.s -p MIPS32 -o test.o
```

On running the above command, the output file test.o is generated for the 'MIPS32' variant.

```
% sasmmps -h
```

On running the above command, the banner, the version information and the syntax for using *sasmmps* along with all the options, is displayed.

```
% sasmmps -V
```

On running the above command, the banner and the version information for *sasmmps* is displayed.

1.3.3 Invoking SANKHYA Assembler for ARM (*sasmarm*)

SANKHYA Assembler for ARM (*sasmarm*) can be invoked from the command line as mentioned below. It takes the assembly file *infile*, generated either by the SANKHYA Code generator for ARM target (*scgarm*) or by the gcc compiler, as the input.

Usage :

On Unix hosts:

```
% sasmarm <infile> [-h] [-V] [-s syntax] [-E endian ] [-p proc]
[-o outfile] [-m mdfile] [-ms syntax_mdfile] [-mr reloc_mdfile]
[-md file] [--nodelayslot]
```

On Windows hosts:

```
> sasmarm.exe <infile> [-h] [-V] [-s syntax] [-E endian] [-p proc]
[-o outfile] [-m mdfile] [-ms syntax_mdfile ] [-mr reloc_mdfile]
[-md file] [--nodelayslot]
```

Description :

The following is the description of the syntax

SYNTAX	DESCRIPTION
sasmarm	Invokes the Assmebler
<infile>	Input Assembly file

-o <outfile>	Specifies output file name [default : <i>infile</i> with default extension]
-E <endian>	Specify endianness. Takes either 'little' or 'big' as its value. [default : Big endian]
-s <syntax>	Specify assembler syntax. [default : gcc]
-p <proc>	Specify target variant
-V	Display version information
-h	Display help message
-m <mdfile>	Input processor description file
-ms <syntax_mdfile>	Input Syntax mdfile
-mr <relocmdfile>	Input processor relocation model file
-md <file>	Input delay slot description file
--nodelay_slot	Do not place nops in delay slots

Default filename extension :

SANKHYA Assembler takes the following as the default extensions. In case, the output filename is not specified, the output filename is the same as the assembly input filename.

TABLE 2. Assembler Filename extensions

FILE	UNIX	WINDOWS
Assembly source file	.s	.s
Relocatable object file	.o	.o

Usage example :

On Unix hosts:

```
% sasmarm test.s
```

On Windows hosts:

```
> sasmarm.exe test.s
```

On running the above command, the output file test.o is generated. Here, test.s is the input source file

```
% sasmarm -s gcc -E little -o test1.o test.s
```

where,

- test.s - assembly file generated by the code generator
- E <little> - target endianness will be set to 'little'

-s <gcc> - ‘gcc’ syntax will be used
-o test1.o - Object file will be ‘test1.o’

On running the above command, the output file test1.o is generated for ‘gcc’ assembler syntax for little endian target.

```
% sasmarm test.s -p ARMv4 -o test.o
```

On running the above command, the output file test.o is generated for the ‘ARMv4’ variant.

```
% sasmarm -h
```

On running the above command, the banner, the version information and the syntax for using *sasmarm* along with all the options, is displayed.

```
% sasmarm -V
```

On running the above command, banner and the version information for *sasmarm* is displayed.

1.4 SANKHYA Linker

1.4.1 Introduction

SANKHYA Linker generates executable by combining one or more ELF object files and archives. It replaces the symbolic addresses in the relocatable object file with the real addresses.

1.4.2 Invoking SANKHYA Linker for MIPS (*sldmips*)

SANKHYA Linker for MIPS (*sldmips*) can be invoked from the command line as mentioned below. It takes the relocatable ELF object file `<file>.o`, generated either by the SANKHYA Assembler for MIPS target (*sasmmips*) or by the gcc compiler, as the input.

Syntax :

On Unix hosts :

```
% sldmips <infile> [-h] [-V] [-E endian] [-e addr] [-o outfile]  
[-l archive] [-L libpath] [-defsym symbol=value] [-mr relocmdfile]
```

On Windows hosts:

```
> sldmips.exe <infile> [-h] [-V] [-E endian] [-e addr] [-o outfile]  
[-l archive] [-L libpath] [-defsym symbol=value] [-mr relocmdfile]
```

Description :

The following is the description of the syntax.

SYNTAX	DESCRIPTION
sldmips	Invokes the Linker
<infile>	Input object file
-o <outfile>	Sets the output file name [default : a.out]
-E <endian>	Specifies the target endianness. Takes either “little” or “big” as its value. [default : Big endian]
-e <addr>	Specifies Program start address
-V	Displays version information
-h	Displays help message
-l <archive>	Specifies archive file to be linked
-L <libpath>	Specifies library path to be searched
-defsym <symbol=value>	Defines symbol value(hex value)

`-mr <relocmdfle>` Input processor relocation model file

Default filename extension :

SANKHYA Linker takes the following as the default extensions. By default, the output file is a.out.

TABLE 3. Linker Filename extensions

FILE	UNIX	WINDOWS
Input object file	.o	.o
Output file	a.out	a.out

Usage example :

On Unix hosts:

```
% sldmips test.o
```

On Windows hosts:

```
> sldmips.exe test.o
```

On running this command, linker generates the default output file a.out for the object file test.o

```
% sldmips test.o -o test.x
```

On running the above command, the input object file, test.o produces test.x as the output absolute file

```
% sldmips test.o -E little -e 00400030 -o test.x
```

On running the above command, the input object file, test.o produces test.x, the output absolute file for little endian target, program the start address being the one mentioned on the command line.

```
% sldmips -h
```

On running the above command, the banner, the version information and the syntax for using *sldmips* along with all the options, is displayed.

```
% sldmips -V
```

On running the above command, the banner and the version information for *sldmips* is displayed.

1.4.3 Invoking SANKHYA Linker for ARM (*sldarm*)

SANKHYA Linker for ARM (*sldarm*) can be invoked from the command line as mentioned below. It takes the relocatable ELF object file <file>.o, generated either by the SANKHYA Assembler for ARM target (*sasmarm*) or by the gcc compiler, as the input.

Syntax :

On Unix hosts :

```
% sldarm <infile> [-h] [-V] [-E endian] [-e addr] [-o outfile]  
[-l archive] [-L libpath] [-defsym symbol=value] [-mr relocmdfile]
```

On Windows hosts:

```
> sldarm.exe <infile> [-h] [-V] [-E endian ] [-e addr ] [-o outfile]
[-l archive] [-L libpath] [-defsym symbol=value] [-mr relocmdfile]
```

Description :

The following is the description of the syntax.

SYNTAX	DESCRIPTION
sldarm	Invokes the Linker
<infile>	Input object file
-o <outfile>	Sets the output file name [default : a.out]
-E <endian>	Specifies the target endianness. Takes either “little” or “big” as its value. [default : Big endian]
-e <addr>	Specifies Program start address
-V	Displays version information
-h	Displays help message
-l <archive>	Specifies archive file to be linked

- L <libpath> Specifies library path to be searched
- defsym <symbol=value> Defines symbol value(hex value)
- mr <reloc_mdfile> Input processor relocation model file

Default filename extension :

SANKHYA Linker takes the following as the default extensions. By default, the output file is a.out.

TABLE 3. Assembler Filename extensions

FILE	UNIX	WINDOWS
Input Object file	.o	.o
Output Absolute file	a.out	a.out

Usage example :

On Unix hosts:

```
% sldarm test.o
```

On Windows hosts:

```
> sldarm.exe test.o
```

On running this command, linker generates the default output file a.out for

the object file test.o

```
% sldarm test.o -o test.x
```

On running the above command, the input object file, test.o produces test.x as the output absolute file

```
% sldarm test.o -E little -e 00400030 -o test.x
```

On running the above command, the input object file, test.o produces test.x, the output absolute file for little endian target, program the start address being the one mentioned on the command line.

```
% sldarm -h
```

On running the above command, the banner, version information and the syntax for using *sldarm* along with all the options, is displayed.

```
% sldarm -V
```

On running the above command, the banner and the version information for *sldarm* is displayed.

1.5 SANKHYA Simulator

1.5.1 Introduction

SANKHYA Simulator (*ssim*) is an instruction set simulator

1.5.2 Invoking SANKHYA Simulator for MIPS (*ssimmips*)

SANKHYA Simulator for MIPS (*ssimmips*) can be executed from the command line as mentioned below. It takes an absolute file *<file>.x*, generated either by the SANKHYA Linker for MIPS target (*sldmips*) or by the gcc compiler, as the input.

Syntax :

On Unix hosts:

```
% ssimmips [-h] [-V] [-b] [-i cmd_file] [-l log_file] [-e error_file]  
[-m mdfile]
```

On Windows hosts:

```
> ssimmips.exe [-h] [-V] [-b] [-i cmd_file] [-l log_file] [-e error_file]  
[-m mdfile]
```

Description :

The following is the description of the syntax.

SYNTAX	DESCRIPTION
ssimmips	Invokes the Simulator
-i <cmd_file>	Simulator include command file
-V	Displays version information
-h	Displays help message
-b	Run in non-interactive mode
-m <mdfile>	Input processor description file
-l <log_file>	Log simulator output to <log_file>
-e <error_file>	Log simulator error to <error_file>

Default filename extension :

Simulator takes the following as the default extensions.

TABLE 4. Simulator Filename extensions

FILE	UNIX	WINDOWS
ELF executable file	.x	.x

Usage example :

On Unix hosts:

The following is the simulator include file 'test.cmd':

```
trace on
cpu c MIPS32
set cpu c
mblock m 0x20000 rw
mregion c m 0x400030 rw
set r29 0x410000
set r31 0xffffffff
load test.x
step 3
memget 0x400030
memget 0x400040
quit
```

where test.x is the executable.

The Simulator can be invoked as follows:

```
% ssimmips -i test.cmd
cpu c MIPS32
set cpu c
mblock m 0x20000 rw
mregion c m 0x400030 rw
set r29 0x410000
```

```
set r31 0xffffffff
load test.x
...916 bytes loaded
step 3
400030 addiu r29 r29 16
400034 sw r30 8 ( r29 )
400038 addu r30 r29 r0
memget 0x400030
Memory

[0x400030] = 0x27bd0010
memget 0x400040
Memory

[0x400040] = 0xafc50014
quit

% ssimmips -h
```

On running the above command, the syntax for using *ssimmips* along with all the options, is displayed.

```
% ssimmips -V
```

On running the above command, the version information for *ssimmips* is displayed.

1.5.3 Invoking SANKHYA Simulator for ARM (*ssimarm*)

SANKHYA Simulator for ARM (*ssimarm*) can be executed from the command line as mentioned below. It takes an absolute file *<file>.x*, generated either by the SANKHYA Linker for ARM target (*sldarm*) or by the gcc compiler, as the input.

Syntax :

On Unix hosts:

```
% ssimarm [-h] [-V] [-i cmd_file] [-m mdfile]
```

On Windows hosts:

```
> ssimarm.exe [-h] [-V] [-i cmd_file] [-m mdfile]
```

Description :

The following is the description of the syntax.

SYNTAX	DESCRIPTION
ssimarm	Invokes the Simulator
-i <i><cmd_file></i>	Simulator include command file
-V	Displays version information
-h	Displays help message

<abs_file> Absolute file to be loaded into simulator

-m <mdfile> Input processor description file

Default filename extension :

Simulator takes the following as the default extensions.

TABLE 4. Simulator Filename extensions

FILE	UNIX	WINDOWS
ELF executable file	.x	.x

Usage example :

On Unix hosts:

The following is the simulator include file 'test.cmd':

```

trace on
cpu c ARMv4
set cpu c
mblock m 0x20000 rw
mregion c m 0x400000 rw
set r13 0x410000
set r15 0xffffffff
load test.x
set r0 0x412000
set r1 0x412004

```

```
step 3
memget 0x400030
memget 0x400034
quit
```

where test.x is the executable.

The Simulator can be invoked as follows:

```
% ssimarm -i test.cmd
cpu c ARMv4
set cpu c
mblock m 0x20000 rw
mregion c m 0x400000 rw
set r13 0x410000
set r15 0xffffffff
load test.x
...904 bytes loaded
set r0 0x412000
set r1 0x412004
step 3
400030 mov r12 << r13 0
400034 stmdb r13 ! [ r15 r14 r12 r11 ]
400038 sub r11 r12 4
memget 0x400030
Memory

[0x400030] = 0xe1a0c00d
memget 0x400034
```

Memory

```
[0x400034] = 0xe92dd800  
quit
```

```
% ssimarm -h
```

On running the above command, the syntax for using *ssimarm* along with all the options, is displayed.

```
% ssimarm -V
```

On running the above command, the version information for *ssimarm* is displayed.

1.6 SANKHYA Librarian

1.6.1 Introduction

SANKHYA Librarian (*slib*) groups any number of input object files in ELF format and generates a single archive file. The main function of a librarian is to generate and update library files.

1.6.2 Invoking SANKHYA Librarian (*slib*)

SANKHYA Librarian (*slib*) can be invoked from the command line as mentioned below.

Usage :

On Unix hosts:

```
% slib [-V] [-h] [-q] [-d] [-r] [-t] [-u] [-x] <file>.a [<file>.o ... ]
```

On Windows hosts:

```
> slib.exe [-V] [-h] [-q] [-d] [-r] [-t] [-u] [-x] <file>.a [<file>.o ... ]
```

Description :

The following is the description of the syntax.

SYNTAX	DESCRIPTION
slib	Invokes the librarian
-V	Displays version information
-h	Displays help message
-q	Appends files to the end of archive
-d	Deletes one or more files from archive
-r	Replaces or adds files in archive
-t	Prints the list of archive member names
-u	Updates older files in archive
-x	Extracts files from archive

Usage example :

On Unix hosts:

```
% slib -q test.a test1.o
```

On Windows hosts:

```
> slib.exe -q test.a test1.o
```

where,

test.a - archive file

test1.o - input object file

On executing the above command, the object file is appended to the archive file. Both [file.a] and [file.o] arguments should be mentioned.

To append another object file,

```
% slib -q test.a test2.o
```

where,

test2.o - input object file

The -t option lists the object files in an archive file. It takes only the [file.a] argument.

```
% slib -t test.a
```

The following output is obtained on executing the above command.

```
test1.o
test2.o
```

The `-d` option deletes the specified object files from the archive file. Both `[file.a]` and `[file.o]` arguments should be mentioned.

```
% slib -d test.a test1.o
```

where,

test1.o - object file to be deleted

On executing the command given below,

```
% slib -t test.a
```

the following output is obtained

```
test2.o
```

The `-r` option either replaces an existing file with the specified file or adds a non-existing file to the archive file. Both `[file.a]` and `[file.o]` arguments should be mentioned.

```
%slib -r test.a test2.o test1.o
```

where,

test2.o-the file to be replaced

test1.o-the file to be replaced with

On executing the above command, the test2.o gets replaced by test1.o.

With,

```
%slib -t test.a
```

the following output is obtained

```
test1.o
```

On executing the command mentioned below, the test3.o, gets appended to the archive file.

```
%slib -r test.a test2.o test3.o
```

where,

test3.o - the object file which was not already present in the archive.

With,

```
%slib -t test.a
```

the following output is obtained

```
test1.o  
test3.o
```

The -u option updates the already existing object file in the archive file with

any updates. Both [file.a] and [file.o] arguments need to be mentioned.

```
% slib -u test.a test1.o test2.o
```

where,

- test1.o - the object file that need to be updated
- test2.o - the object file that has to be replaced with

With,

```
%slib -t test.a
```

the following output is obtained

```
test2.o  
test3.o
```

The -x option extracts the files from the archives. Here, only the [file.a] argument needs to be mentioned.

```
% slib -x test.a
```

On executing the above command, the files in the archive files are extracted.

```
% slib -h
```

On executing the above command, the banner, the version information and the syntax for using *slib* along with all the options, is displayed.

Sankhya Librarian Version 1.0 (Beta2)
Copyright (c) 2002 Sankhya Technologies Private Limited
All Rights Reserved.

Usage : slib <options ...> <file>.a [<file>.o ...]

Options:

- V --Displays version information
- h -- Displays help message
- q -- Quickly appends files at end of archive
- d -- Deletes one or more files from archive
- r -- Replaces or adds files in archive
- t -- Prints a list of archive member names
- u -- Updates older files in archive
- x -- Extracts files from archive

On executing the following command, the banner and the version information for *slib* is displayed.

```
% slib -V
```

```
Sankhya Librarian Version 1.0 (Beta2)  
Copyright (c) 2002 Sankhya Technologies Private Limited  
All Rights Reserved.
```

1.7 SANKHYA Dumper

1.7.1 Introduction

SANKHYA Dumper (*sdump*) displays information about the input ELF object file or ELF archive files.

1.7.2 Invoking SANKHYA Dumper

SANKHYA Dumper for MIPS (*sdumpmips*) and ARM (*sdumparm*) can be invoked from the command line as mentioned below.

Usage :

On Unix hosts:

```
% sdump{mips/arm} [-V] [-H] [-f] [-g] [-a] [-h] [-sh] [-ph] [-r] [-s]
  [-x <number>] [-m <file>] [-mr <file>] <file.a / file.o>
```

On Windows hosts:

```
> sdump{mips/arm}.exe [-V] [-H] [-f] [-g] [-a] [-h] [-sh] [-ph] [-r] [-s]
  [-x <number>] [-m <file>] [-mr <file>] <file.a / file.o>
```

Description :

The following is the description of the syntax.

SYNTAX	DESCRIPTION
<code>sdump{mips/arm}</code>	Invokes the dumper
<code>-V</code>	Displays version information
<code>-H</code>	Displays help message
<code>-f</code>	Dumps each file header of an archive
<code>-g</code>	Dumps global symbols of an archive symbol table
<code>-a</code>	Dumps all the required informations about object file (equivalent to <code>-sh -ph -r -s</code>)
<code>-h</code>	Dumps the object file headers
<code>-sh</code>	Dumps the section headers
<code>-ph</code>	Dumps the program headers
<code>-r</code>	Dumps relocation informations
<code>-s</code>	Dumps symbol table entries

- x<number> Dumps the content of section <number> in hexadecimal

- m <file> Obtain processor machine description from <file>

- mr <file> Obtain relocation description from <file>

Usage example :

-H option displays the banner, the version information and the syntax for using *sdumpmips* along with all the options

```
> sdumpmips -H
Sankhya Objectdumper Version 1.0 (Beta2) for MIPS
Copyright (c) 2003 Sankhya Technologies Private Limited
All Rights Reserved.
```

```
Usage : sdumpmips <options ...> [<file>.o/.a ...]
```

```
Supported Object File Format: elf32-big, elf32-little
```

Options:

- V -- Displays version information
- H -- Displays help message
- f -- Dumps each file header of an archive
- g -- Dumps global symbols of an archive symbol table
- a -- Dumps all the required informations about object file (equivalent to -sh -ph -r -s)
- h -- Dumps the object file headers

-sh	--	Dumps the section headers
-ph	--	Dumps the program headers
-r	--	Dumps relocation informations
-s	--	Dumps symbol table entries
-x <number>	--	Dumps the content of section <number> in hexadecimal
-m <file>	--	Obtain processor machine description from <file>
-mr <file>	--	Obtain relocation description from <file>

-V option displays the banner and the version information of dumper.

```
>sdumpmips -V
Sankhya Objectdumper Version 1.0 (Beta2) for MIPS
Copyright (c) 2003 Sankhya Technologies Private Limited
All Rights Reserved.
```

-f option lists the object files in the archive file in the following format.
permissions size date of creation file name

```
>sdumpmips -f test.a

File: test.a

test.o: ELF
33188 5068/100 772 Wed Jun 11 14:13:17 2003 test.o

min.o: ELF
33188 5068/100 772 Wed Jun 11 14:08:20 2003 min.o
```

```
demo.o: ELF
33188 5068/100 772 Wed Jun 11 14:13:01 2003 demo.o
```

-g option displays the global symbol table of an archive file.

```
>sdumpmips -g test.a
```

```
File: test.a
```

```
Archive Global Symbol Table:
```

Symbol Name	File Name	Offset
minminmin	test.o	0x00000064
	min.o	0x000003a4
	demo.o	0x000006e4

-h option display the object file header information.

```
File: test.o
```

```
ELF Header:
```

```
Magic: 7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00
```

```
...
```

```
...
```

```
Section Headers:
```

[No]	Name	Type	Off	Size	Esize	Flag	Align
[0]		0	000000	000000	00	00	0
[1]	.shstrtab	3	000034	0000c1	00	00	1

```
...
```

...

There are 8 section headers

There are no program headers in this file

-sh option displays the section header information of object file.

```
> sdumpmips -sh test.o
```

```
File: test.o
```

```
Section Headers:
```

[No]	Name	Type	Off	Size	Esize	Flag	Align
[0]		0	000000	000000	00	00	0
[1]	.shstrtab	3	000034	0000c1	00	00	1

...

...

There are 8 section headers

-ph option displays the program header information of executable file.

```
>sdumpmips -ph test.o
```

```
File: test.o
```

```
There are no program headers in this file
```

-r option displays target relocation information along with the target relocation name.

```
>sdumpmips -r test.o
```

File: test.o

Relocation section '.rel.text':

Offset	Info	Symbol's Value	Symbol's Name	Addend	Type
00000030	00104	00000000	.text	0	R_MIPS_26a
00000040	00104	00000000	.text	0	R_MIPS_26a

-s option displays symbol table information of the object file.

```
>sdumpmips -s test.o
```

File: test.o

Symbol table:

No:	Value	Size	Type	Bind	Shindx	Name
0:	00000000	0	0	0	0	
1:	00000000	0	3	0	2	
	...					
	...					

-x option displays the content of section <number> in hexadecimal format. It displays 'disassembly' information also along with code section content display.

```
> sdumpmips -x 3 test.o
```

File: test.o

Hex dump of section '.symtab':

```
0x00000000 00000000 00000000 00000000 00000000
0x00000010 00000000 00000000 00000000 03000002
```

```
0x00000020 00000000 00000000 00000000 03000005  
0x00000030 00000000 00000000 00000000 03000006  
0x00000040 00000001 00000000 0000005c 12000002  
0x00000050
```

-a option displays all the required informations about object file. Generally, it displays file header, section header, program headers, section contents and symbol table information. It is equivalent to -sh,-ph, -r and -s options taken together.

-m option provides processor machine description information to perform disassembly action.

-mr provides relocation information of the target to display target dependedent relocation names.

1.7 Using SANKHYA Tools Collection

1.7.1 Usage Example - 'swap'

SANKHYA Tools Collection samples for the target 'tgt' (arm/mips) are found in the directory `$STC_HOME/samples/{tgt}`. The swap sample explained below swaps the values between two memory locations.

Here is the complete overview of the steps to be followed.

Step 1. Create an icode file

Step 2. Invoke the code generator to generate the assembly file

Step 3. Invoke the assembler to assemble the generated assembly file

Step 4. Invoke the linker to generate the executable

Step 5. Create the command file for the simulator

Step 6. Invoke the simulator to execute the application

The following section explains in detail of the above specified steps.

Step 1. Create the icode file

Contents of icode file 'swap.i'

```
function swap
param {type=int:ptr} i
param {type=int:ptr} j
begin
    local {type=int} tmp
    = tmp {arity=1}* i
```

```
= {arity=1}* i {arity=1}* j  
= {arity=1}* j tmp  
end
```

Step 2. Invoke the code generator as follows.

```
%scgmips swap.i
```

Step 3. Invoke the assembler as follows.

```
%sasmips swap.s
```

Step 4. Invoke the linker as follows.

```
%sldmips swap.o -o swap.x
```

Step 5. Create the command file as below.

Here is the contents of swap.cmd

```
cpu c MIPS32  
set cpu c  
mblock m 0x20000 rw  
mregion c m 0x400030 rw  
set r29 0x410000  
set r31 0xffffffff  
load swap.x  
memset 0x412000 10  
memset 0x412004 20
```

```
set r4 0x412000
set r5 0x412004
run 0xffffffff
memget 0x412000
memget 0x412004
quit
```

Step 6. Invoke the simulator as follows.

```
%ssimmips -i swap.cmd
```

The following output will be displayed in the terminal.

```
...916 bytes loaded
Memory

[0x412000] = 0x00000014
Memory

[0x412004] = 0x0000000a
```

Note:

For more information on running the sample for ARM, please refer the samples directory provided as part of the release.

REFERENCE MANUAL

Sankhya Technologies Private Limited

Part 2- Reference Manual

2.1 SANKHYA Code Generator

2.1.1 Overview

SANKHYA Code Generator (*scg*) supports the SANKHYA intermediate code specification and generates an appropriate assembly code.

2.1.2 SANKHYA Code Generator features

The following are some of the significant features of SANKHYA Code Generator:

- Input is simple and extremely powerful intermediate code (human readable)
- Interoperates with GCC binutils for MIPS.
- Supports little-endian and big-endian processor variants.
- Supports different assembly language formats.

2.1.3 SANKHYA Intermediate code

SANKHYA Code Generator, takes intermediate code (icode) as input and converts it to an assembly code. The following sections describe the syntax and the specifications for SANKHYA intermediate code.

2.1.3.1 Tokens

Variables

The variables are declared as

```
var_type var
```

where,

var_type denotes the type of the variable

var denotes the variable name

Example :

```
int tmp
```

The default variable type is integer.

Data Types

The following is the list of data types supported by SANKHYA Code generator and the memory occupied by each.

- bool - Boolean type (1 byte)
- char - Character type (1 byte)
- unsigned char - Unsigned Character type
- short int - Short Integer type
- unsigned short int - Unsigned Short Integer type
- int - Integer type (2 bytes)

- unsigned int - Unsigned Integer type
- long - Long Integer type (4 bytes)
- unsigned long - Unsigned Long Integer type
- float - Single Precision floating-point (4 bytes)
- double - Double Precision floating-point (4 bytes)
- long double - Long Double Precision floating-point (4 bytes)

Assigning Variables

The variables can be assigned by using the “=” operator

Syntax:

```
= <lvalue> <expr>
```

Description:

The value of the <expr> is assigned to <lvalue>. The <lvalue> can be either variables or registers or memory.

Example:

```
=a+bc
```

where a,b,c are variables. The above instruction adds the variables ‘b’ and ‘c’ and places the result in ‘a’.

Note: Prefix notation is followed for writing icode.

Keywords

The following are the keywords used in an icode program

begin	-	to specify begin of the function's body.
end	-	to specify end of the function's body.
function	-	to specify function's definition.
global	-	to declare global variable(s).
local	-	to declare function's local variable(s).
param	-	to declare function's formal parameter(s).

Punctuations

The following punctuations are allowed in SANKHYA icode.

{ } - used to denote an attribute list. Attributes for any operators or any operands are enclosed within "{" and "}".

Example: {attrib_name=val,attrib_name=val,..}

,

- used to separate the attributes

Example: {attrib_name=val,attrib_name=val,..}

:

- used to link an attribute value

Example: {attrib_name=val1:val2:val3:...,attrib_name=val,..}

// - used to denote a comment statement

Example: // This is a demo

% - used to denote special operator

Example: {attrib_name=val1:val{ “{width=32”, rd, “%plus”, rt, rs” } }

2.1.3.2 Variables

Every variable name must start with an alphabet, the rest of the name can consist of alphanumeric and underscore character. A variable must be declared before it can be used.

Local variable

Local variables are declared within the body of the function, and can only be used within that function.

Local variables are declared as

```
local {type=<data_type>}<var>          - Ordinary variable declaration
local {type=<data_type>:ptr}<var>      - Pointer Variable declaration
```

where,

<data_type> denotes the data type of the variable. Default value is integer

<var> denotes the variable name.

Example:

```
local a
local {type=int}a
local {type=char:ptr}str
```

Global variables

Global variables are declared outside the body of any functions, and can be used within any functions.

Global variables are declared as

```
global {type=<data_type>}<var>      - Ordinary variable declaration
global {type=<data_type>:ptr}<var>  - Pointer Variable declaration
```

where,

<data_type> denotes the data type of the variable. Default value is integer
<var> denotes the variable name.

Example:

```
global a
global {type=int}a
global {type=char:ptr}str
```

2.1.3.3 Types qualifier

Constant

The Code Generator can optimize the usage of "constants". Constants can be identified using the attribute value "const".

Syntax:

```
local/global {type=<data_type>:const}<const_name> = <const_value>
```

Example:

```
local {type=float:const}pi = 3.14
```

Volatile

The Code Generator cannot optimize the usage of "volatile" variables. The volatile variables can be identified using the attribute value "volatile".

Syntax:

```
local/global {type=<data_type>:volatile}<volatile_var>
```

Example:

```
global {type=int:volatile}semaphore
```

2.1.3.4 Attributes

Attributes determine the properties of a name. Attributes can be represented by attribute name and its value(s). Multiple values can be assigned for an attribute, using the ":" separator. The attributes list starts with '{' and ends with '}'. Attributes in a list, are separated by ','.

Example:

```
= {sign=u,width=32}+ r2 r3
```

In the above icode, the operator "+" has the attributes "sign", "width" and its values are "u", "32" respectively.

By default, all attributes are used for getting (not used in matching) and setting purpose. If some attributes need to be used for getting purpose only, that attribute name should be prefixed with '!'.

Example:

```
{!delayslot=1}j {eltype=symbol}L3
```

In the above icode, the attribute 'delayslot' is used only for getting purpose (i.e 'delayslot' attribute is not used in the attribute match process).

Attributes can be classified into two types: One is a set of attributes which are used for getting and setting. Other is, a set of attributes which are used for getting purpose only.

The most important attribute of a name is its *type*, which determines the values it might take. The following are the attributes that can be set in an

expression.

- Access Attribute
- Cycle Attribute

Access Attribute

Access attributes refer to the read/write permissions that are set by the user in an icode expression.

For example, for an icode expression,

$$= a + bc$$

where a,b and c are operands , the access attributes can be given as follows

$$= \{access=wo\}a + \{access=ro\} b \{access=ro\} c$$

where,

‘access’ is the attribute to access the operand

ro - Read Only

wo - Write Only

rw - Read and Write

If the attributes are not declared in an icode expression, it means there is no restriction on the ownership. The default value for access attribute is ‘**rw**’.

Cycle Attribute

SANKHYA icode also supports cycle attributes to be set for icode expressions to take care of real time requirements.

Syntax:

```
cycle=<number>  
read-cycle=<number>  
write-cycle=<number>
```

For example, for the icode expression,

```
++r1
```

the cycle attribute can be represented as

```
{cycle=2}++ {read-cycle=1, write-cycle=3} r1
```

Delay slot attribute

Syntax:

```
{delayslot=<numeric>}
```

Description:

Branch and some load instructions scheduling involves delay slot which is used to execute delay slot instructions which immediately follows the

current instruction in memory.

Compiler tools generate code accordingly to care of the delay slots by swapping the preceding instruction or using null instruction (nop)

In order to specify delay slot requirements at icode expression level, it is more suitable to add a new attribute for the icode element which triggers the delay slot.

Example:

For unconditional jump,

```
{!delayslot=1}j r1
```

For conditional jump (say, beq r1 r0 L3)

```
{!delayslot=1}jt == r1 r0 L3
```

For load operation (say, lw r1 10(r30))

```
r1 {arity=1,!delayslot=1}* + r30 10
```

Nullify attribute

Syntax:

```
{nullify=<numeric>}
```

Description:

For conditional jump instructions, the delay slot instructions are scheduled before checking the conditions. If the condition fails, then the delay slot instructions are nullified. This property of scheduling in jump instruction can be processed using the new attribute "nullify".

Example:

For branch likely conditional jump instruction (say, beql r1 r0 L3).

```
{!delayslot=1,nullify=1}jt == r1 r0 L3
```

TABLE 5. Literals

Attribute name	Attribute values	Description
sign	s u	Signed Unsigned
width	<numeric value>	To specify width of the operand or width of the operator result
type	char int long float double ptr	Character data type Integer data type Long data type Float data type Double data type Pointer
arity	<numeric value>	Specify number of operands required for a operator
eltype	symbol	Element type is symbol

Attribute name	Attribute values	Description
etype	aconst	Expression type is assembler constant
	lconst	Expression type is Linker constant
cycle	<numeric value>	Specify number of cycle
read-cycle	<numeric value>	Specify number of read cycle
write-cycle	<numeric value>	Specify number of write cycle
delayslot	<numericvalue>	Specify number of delay slots
nullify	<numericvalue>	Specify number of instructions to be nullified

Multiple Registers Representation

Syntax:

$$[\langle r1 \rangle \langle r2 \rangle \dots \langle rn \rangle]$$

Example:

$$\{type=ia\} = [r1 r2 r3] \{arity=1\} * r12$$

This instruction loads registers r1, r2 and r3 from the memory location pointed by the register "r12"

$$\{type=ia\} = \{arity=1\} * r12 [r1 r2 r3]$$

This instruction stores registers r1, r2 and r3 in the memory location pointed by the register "r12"

where,

type=ia - represents that multiple load/store operation is of type "incremental after".

Other types are,

- ib - increment before
- da - decrement before
- db - decrement before

Register Pair Representation

Syntax:

```
<r1>:<r2>
```

Example:

```
= r1:r2 {width=64}+ {width=64}* r3 r4 r1:r2
```

This instruction places 64-bit results in 32-bit register pair (r1 and r2),

2.1.3.5 Pointer reference and dereference

Pointer Variable

A pointer variable can be declared as

```
{type=var_type:ptr} i
```

Example:

```
{type=int:ptr} i
```

The above statement specifies that the variable "i" is a pointer to an integer data type.

Pointer Dereference

Syntax:

```
{type=int:ptr} i  
{type=int} j  
= j {arity=1}* i
```

where,

i is a pointer to integer

j is the integer type variable

{arity=1}* represents dereference to the pointer variable

2.1.3.6 Labels

A label is an identifier or a symbol identifying a particular line in an assembly language program.

It is a sequence of nops (no-operation instructions), used to modify the instructions that precedes them. Typically, they are used to reference another point in the code where the complement label is located.

Syntax

```
label <label_name>
```

2.1.3.7 Operands

Operands can be either Literals, Symbols, Registers or Memory.

Literals

Literals can be used as a prefix to attributes.

TABLE 5. Literals

Type	Example
Integer Constants	1, 2, -4, +6, 4, 10, 0xabcd, 0123
Character Constants	'a' '\0123'
Strings	"abc"
Real	1.3, 3.4E1, -3.4E-22, +5.6E+30

Symbols

Labels can be declared in terms of element type (eltype) and expression type (etype).

Example:

```
{eltype=symbol}function_name
```

The above example denotes that the label 'function_name' is an element of type 'symbol'.

```
{etype=aconst}+ {eltype=symbol}function_name 4
```

The above example denotes that for the expression 'function_name+4', the element type is symbol and the expression type is assembler constant.

Register

Registers are represented as *r<register_no>*

Example:

r2 denotes the second register.

Note: Register operand cannot be used in input intermediate for Code Generator.

Memory

Memory is also an operand for instruction.

Example:

Load *r3* <addr>

Here, <addr> refers to the memory address

2.1.3.8 Operators

An operator is a token that represents a particular operation to be performed on one or two operands. The combination of operands and operator is called an expression.

a) Arithmetic Operators

SANKHYA Intermediate code supports the following arithmetic operators

Addition ('+')

Syntax:

+ <expr1> <expr2>

The above instruction returns the result of "<expr1> + <expr2>".

Example:

```
+ a b  
+ r1 r2
```

Subtraction ('-')

Syntax:

- <expr1> <expr2>

The above instruction returns the result of "<expr1> - <expr2>".

Example:

```
- a b  
- r1 r2
```

Multiplication ('*')

Syntax:

`* <expr1> <expr2>`

The above instruction returns the product of <expr1> and <expr2>.

Example:

`* a b`
`* r1 r2`

Division ('/')

Syntax:

`/ <expr1> <expr2>`

The above instruction returns the result of "<expr1> / <expr2>".

Example:

`/ a b`
`/ r1 r2`

Modulo ('%')**Syntax:**

```
% <expr1> <expr2>
```

The above instruction returns the remainder of "<expr1> / <expr2>".

Example:

```
% a b  
% r1 r2
```

b) Logical Operators**AND ('&&')****Syntax:**

```
&& <expr1> <expr2>
```

The above instruction returns true, if <expr1> and <expr2> is non-zero. If <expr1> or <expr2> is zero, then it returns false.

Example:

```
&& a b
```

OR ('||')**Syntax:**

```
|| <expr1> <expr2>
```

The above instruction returns true, if <expr1> or <expr2> is non-zero. If <expr1> and <expr2> is zero, then it returns false.

Example:

```
|| a b
```

NOT ('!')**Syntax:**

```
! <expr1>
```

The above instruction returns true, if <expr1> is zero. Otherwise, returns false.

Example:

```
! a
```

c) Relational Operators

Equal ('==')

Syntax:

`== <expr1> <expr2>`

The above instruction returns true if `<expr1> == <expr2>`. Otherwise, returns false.

Example:

`== a b`

Not Equal ('!=')

Syntax:

`!= <expr1> <expr2>`

The above instruction returns true if `<expr1> != <expr2>`. Otherwise, returns false.

Example:

`!= a b`

Less than ('<')**Syntax:**

< <expr1> <expr2>

The above instruction returns true if <expr1> < <expr2>. Otherwise, returns false.

Example:

< a b

Less than or Equal ('<=')**Syntax:**

<= <expr1> <expr2>

The above instruction returns true if <expr1> <= <expr2>. Otherwise, returns false.

Example:

<= a b

Greater than ('>')

Syntax:

`> <expr1> <expr2>`

The above instruction returns true if `<expr1> > <expr2>`. Otherwise, returns false.

Example:

`> a b`

Greater than or Equal ('>=')

Syntax:

`>= <expr1> <expr2>`

The above instruction returns true if `<expr1> >= <expr2>`. Otherwise, returns false.

Example:

`>= a b`

d) Bitwise operators

Bitwise AND ('&')

Syntax:

`& <expr1> <expr2>`

The above instruction returns the result of bit-wise AND operation "<expr1> & <expr2>".

Example:

`& a b`

Bitwise OR ('|')

Syntax:

`| <expr1> <expr2>`

The above instruction returns the result of bit-wise OR operation "<expr1> | <expr2>".

Example:

`| a b`

Bitwise NOT ('~')**Syntax:** $\sim <expr1>$

The above instruction returns one's complement of $<expr1>$.

Example: $\sim a$ **Bitwise Exclusive OR ('^')****Syntax:** $\wedge <expr1> <expr2>$

The above instruction returns the result of bit-wise exclusive-OR operation " $<expr1> \wedge <expr2>$ ".

Example: $\wedge a b$

Logical Shift Left ('<<')**Syntax:**

```
<< <expr1> <expr2>
```

The above instruction returns the result of <expr1> left shifted by <expr2> amount of bits. The emptied bits will be filled with zero.

Example:

```
<< a b
```

Logical Shift Right ('{sign=u}>>')**Syntax:**

```
{sign=u}>> <expr1> <expr2>
```

The above instruction returns the result of <expr1> right shifted by <expr2> amount of bits. The emptied bits will be filled with zero.

Example:

```
{sign=u}>> a b
```

Arithmetic Shift Right ('>>')

Syntax:

```
>> <expr1> <expr2>
```

The above instruction returns the result of <expr1> right shifted by <expr2> amount of bits. The emptied bits will be filled with sign-bit (i.e most significant bit).

Example:

```
>> a b  
{sign=s}>> a b
```

Rotate Left ('rol')

Syntax:

```
rol <expr1> <expr2>
```

The above instruction returns the result of <expr1> rotated left by <expr2> amount of bits. The emptied bit will be filled with most significant bit for every rotation.

Example:

```
rol r2 2
```

Rotate Right ('ror')

Syntax:

```
ror <expr1> <expr2>
```

The above instruction returns the result of <expr1> rotated right by <expr2> amount of bits. The emptied bit will be filled with least significant bit for every rotation.

Example:

```
ror r2 2
```

e) Other Operators

NOR (NOT OR) operator

Syntax:

```
~| <op1> <op2>
```

Example:

```
~| r1 r2
```

This instruction represents "r1 NOR r2".

Add with carry**Syntax:**

```
++ <op1> <op2> <carry_bit>
```

Example:

```
++ r1 r2 bf 29 1 cpsr
```

The above instruction adds 'r1' and 'r2' with carry, for ARM processor.

Subtract with carry**Syntax:**

```
-- <op1> <op1> ! <carry_bit>
```

Example:

```
-- r1 r2 ! bf 29 1 cpsr
```

This instruction subtracts 'r2' from 'r1' with borrow, for ARM processor.

Counting Leading/Trailing Zero/One

Syntax:

```
bc <occurrence> <value> <expr>
```

where,

<occurrence> can be either "lead" or "trail"(i.e "left" or "right")

<value> can be either "0" or "1"

<expr> can be any expression/operand.

Example:

```
bc lead 0 r1
```

This instruction counts the leading zero in 'r1'

Left/Right Most Bit Detection

Syntax:

```
bp <occurrence> <value> <expr>
```

where,

<occurrence> can be either "lead" or "trail" (i.e "left" or "right").

<value> can be either "0" or "1"

<expr> can be any expression/operand

Example:

```
bp trail 1 r1
```

This instruction finds the rightmost '1', in r1

Pre increment**Syntax:**

```
= <op1> + <op1> <expr>
```

where,

<expr> - represents sub-expression (i.e an operand in main expression).

Example:

```
= r3 + = r1 + r1 4 r2
```

In the above instruction, "= r1 + r1 4" represents pre-incrementing 'r1' by 4.

Post increment**Syntax:**

```
%rcomma <op1> = <op1> + <op1> <exp>
```

where,

<expr> - represents sub-expression

Example:

```
= r3 + %rcomma r1 = r1 + r1 4 r2
```

In the above instruction, "%rcomma r1 = r1 + r1 4" represents post-incrementing 'r1' by '4'.

Setting bit pattern in an operand**Syntax:**

```
= bf <start_bit> <width> <operand> <expr>
```

Example:

```
= bf 4 4 r1 10
```

The above instruction sets bit 4 to 7 as "1010" (i.e 10), in the register 'r1'.

Unary Plus ('+')**Syntax:**

```
{arity=1}+ <expr1>
```

The above instruction just returns <expr1>.

Example:

`{arity=1}+ a`

Unary Minus ('-')**Syntax:**

`{arity=1}- <expr>`

The above instruction returns negated value of <expr1> (i.e -<expr1>).

Example:

`{arity=1}- a`

Address Operator ('&')**Syntax:**

`{arity=1}& <lvalue>`

The above instruction returns address of <lvalue>.

Example:

`{arity=1}& a`

Dereference operator ('')****Syntax:**

```
{arity=1}* <addr_expr>
```

The above instruction returns contents of memory location pointed to by <addr_expr>. The <addr_expr> can be any pointer variable or valid address expression.

Example:

The memory contents pointed to by the pointer i_ptr is represented as

```
{arity=1}* i_ptr
```

Swap function can be represented in icode as follows,

```
function swap
param {type=int:ptr} i      // int * i;
param {type=int:ptr} j      // int * j;
begin
  local {type=int} tmp      // int tmp;
  = tmp {arity=1}* i        // tmp = *i;
  = {arity=1}* i {arity=1}* j // *i = *j;
  = {arity=1}* j tmp        // *j = tmp;
end
```

Bit Field operator ('bf')

Syntax:

```
% bf <start_bit> <width> <expr1>
```

where,

- bf - operator used to extract the bit field from the <expr>
- index - start bit to extract from <expr>
- width - number of bits to be extracted from <expr>.
- exp - <address-expression>

The above instruction extracts the bits between <start_bit> and "<start_bit> + <width>" from <expr1>, and returns the extracted bits value. This operator can be used for both getting and setting (if it is used as lvalue) range of bits in an operand.

Example:

To get the second bit from r1, the icode can be written as

```
= % bf 2 1 r1
```

To get the second, third and fourth bits from r1,

```
= % bf 2 3 r1
```

Push and Pop Operators

Syntax:

```
% PUSH <argument>
% POP  <argument>
```

where,

- <argument> - address-exp or registers_list
- PUSH - pushes the <argument> into the stack
- POP - pops the <argument> from the stack

Example:

```
push [ r1, r2, r3 ] - push the registers r1,r2, r3 into the stack.
pop  [ r1, r2, r3 ] - pop the registers r1,r2, r3 from the stack.
```

2.1.3.9 Instructions

An icode instruction can be represented in the following ways:

(i)“{attrib_name=val,attrib_name=val,..}operator", "operand1", “operand2”

Example:

```
{ "{sign=s,width=32}=" rd, a }
```

(ii) “{attrib_name=val1:val2:val3:...,attrib_name=val,..}operator”, "operand"

Example:

```
{ "{width=32,type=int:ptr}"= " rd, a }
```

(iii) “{attrib_name=val..}operator", "operand", "%operator", "operand"

In the above notation, the operator is specified with leading "%".

Example:

(a) { "{width=32}"=, rd, "%+", rt, rs }

or

```
{ "{width=32}"=, rd, "%plus", rt, rs }
```

(b) { "{width=32}"=, rd, "%*", rt, rs }

or

```
{ "{width=32}"=, rd, "%multi", rt, rs }
```

where,

rd, rt and rs	-	Register
a	-	Address

Branch Instructions

SANKHYA icode supports both conditional and unconditional branch instructions that could be used in a program.

(i) Jump

A jump statement forces the flow of execution to jump unconditionally or conditionally to another location in the script.

Syntax

```
jcc <cc> <cereg-f> <address-expr>
```

This instruction checks the condition. Once the condition ‘cc’ gets satisfied, the statement with the address ‘address-expr’ is executed next. The cereg flags are set.

A return statement is a jump statement. The syntax is as follows:

```
return <var_name>
```

where var_name is a variable.

Example

return res - returns the value of ‘res’ to the caller.

(ii) Compare and Set

This instruction compares operand1 with operand2 and sets the appropriate register.

Syntax :

```
cmp op1 op2 <ccreg-f>
```

This instruction compares operand1 with operand2 and sets the condition code register flags (<ccreg-f>)

```
cmp.cc cc op1 op2 <ccreg-b>
```

This instruction compares operand1 with operand2 based on the condition 'cc' and sets the condition code register bit (<ccreg-b>), if the condition is satisfied.

where,

<ccreg-f> - {ccflags=zero:negative} <ccreg>
 <ccreg-b> - {ccflags=bool} <ccreg>
 <cc> - operand having the following values,

eq	-	equal
ne	-	not equal
lt	-	less than
le	-	less than or equal
gt	-	greater than
ge	-	greater than or equal
cs	-	carry set
cc	-	carry clear (i.e no carry)
vs	-	overflow set
vc	-	overflow clear (i.e no overflow)

Example :

The intermediate code for comparing register 'r1' with 'r2' and set the program status register for ARM target is,

```
cmp r1 r2 cpsr
```

(iii) Compare and Jump

```
cmp.jt <cond-expr> <psr> <label>
```

where,

psr - Program Status Register

This instruction branches to <label> if the <cond-expr> is true.

```
cmp.jf <cond-expr> <psr> <label>
```

where,

psr - Program Status Register

This instruction branches to <label> if the <cond-expr> is false.

(iv) Unconditional jump**Syntax :**

```
j <addr-expr>
```

On executing the above instruction, the next statement to be executed is the statement with the address represented by <addr-expr>.

Example :

```
j L3  
j + L3 10
```

(v) Conditional Jump

Syntax :

```
jt <cond-expr> <addr-expr>
```

This instruction jumps to address <addr-expr>, if the <cond-expr> is true.

```
jf <cond-expr> <addr-expr>
```

This instruction jumps to address <addr-expr>, if the <cond-expr> is false.

Example :

```
jt == r1 r2 L3 - jumps to L3, if r1 == r2  
jf > r1 r2 L4 - jumps to L4, if r1 <= r2
```

Return Statement

Syntax :

```
return <var_name>
```

The above instruction returns the value of <var_name> to the calling function.

Example :

```
return res
```

This instruction returns the value of 'res' to the calling function.

Label Statement

Syntax :

```
label <label_name>
```

The above instruction defines <label_name> as a label.

Example :

```
label L3
```

2.1.3.10 Expressions

An expression consists of an operand on its own, or a combination of operators and operands. Prefix notation is used to write expressions. Arity is defined as the number of operands that can be used with an operator.

TABLE 6. Operators and arity values

OPERATOR	NAME	ARITY	EXAMPLE
+	plus	2	+ exp exp
-	sub	2	- exp exp
*	mul	2	* exp exp
/	div	2	/ exp exp
%	modulo	2	% exp exp
<<	shift_left	2	<< exp exp
>>	shift_right	2	>> exp exp
<	less_than	2	< exp exp
<=	less_than_equal	2	<= exp exp
>	greater_than	2	> exp exp
>=	greater_than_equal	2	>= exp exp
==	equal	2	== exp exp
!=	not equal	2	!= exp exp
&&	AND	2	&& exp exp
	OR	2	exp exp
&	bitwise_and	2	& exp exp
	bitwise_or	2	exp exp
+	unary-plus	1	+ exp
-	unary-minus	1	- exp
*	dereference	1	* ptr-exp
&	address	1	& lvalue
j	Jump	1	j <label>
jt	Jump if true	2	j <cond_expr> <label>
jf	Jump if false	2	j <cond_expr> <label>
bf	bit field	3	bf <start bit> <width> <expr>
?	execute conditionally	3	? cc ccreg exp
,	comma	2	, <exp> <exp>
^	exclusive OR	2	^ exp exp
~	bitwise not	1	~ exp
push	push	-	push <arg_list>
pop	pop	-	pop <arg_list>
nop	no operation	0	nop
rcomma	reverse comma	2	rcomma <expr> <expr>

where,

exp - <operand> or <operator> <operand-list>

Conditional Expression (Predicated Instruction)

Syntax :

? <cond-expr> <expr>

It evaluates the expression <expr> only when the <cond-expr> is true.

Example :

? == r1 r2 = r3 + r4 r5

The above instruction, evaluates the expression ‘(r3=r4+r5)’ only if the condition ‘(r1==r2)’ is satisfied.

2.1.3.11 Functions

Declaration

A function can be declared as

function <return_type> <function_name>

The default function return type is “int”

Example :

```
function min
```

where ‘**min**’ is the function name with no return type.

Function Definition

The format of a function definition in intermediate code, is as follows

```
function <return_type> <function_name>
param {attrib_type=val,attrib_type=val,..} <parameter_name>
param {attrib_type=val,attrib_type=val,..} <parameter_name>
.
.
param {attrib_type=val,attrib_type=val,..} <parameter_name>
begin
  local {attrib_type=val,attrib_type=val,..} <var_name>
  local {attrib_type=val,attrib_type=val,..} <var_name>
  .
  .
  local {attrib_type=val,attrib_type=val,..} <var_name>
  <<ICODE EXPRESSIONS>

end
```

where,

- <return_type> - function’s return type (i.e void, int, char,).
- <function_name> - function name.

- <param_name> - formal parameter name.
- <var_name> - local variable name.

For functions, icode supports attributes, used for icode value matching. Multiple attributes are also supported on an icode value as shown below.

Param

Syntax

```
param {attrib_name=val,attrib_name=val,..} <parameter_name>
```

The “param” in the above icode statement refers to the function paramters

Local

The keyword ‘local’ is used to declare the local variables

Syntax

```
local {attrib_name=val,attrib_name=val,..} <var_name>
```

Begin

The function body in an icode program should always start with a ‘begin’ statement

Syntax

```
begin
```

End

The 'end' statement in a function signifies the end of the function body.

Syntax

```
end
```

Usage Example:

The following example shows a function that finds the least of two numbers.

```
function min
param a
param b
begin
    cmp.jt < a b L3
    return b
    label L3
    return a
end
```

In the above program, the parameters a and b are declared in the function min. As seen from the program, the smaller of the two (a and b) is returned after comparison using the cmp.jt statement. The keyword 'cmp.jt' compares and jumps to the mentioned label, if the condition is found to be true.

Function call

Syntax :

```
call <addr-expr>
```

The above instruction transfers the execution control to a function (sub-routine) which is represented by <addr-expr>. After executing all statements in the called function, the execution resumes from the statement next to the call statement.

Example :

```
call min
```

The format of a function call inside a function is given with the following example.

A function "fun" calls a function "min" which has two actual parameters, is defined as below;

```
function fun
param a
param b
begin
    ...
    ...
    param_arg a
    param_arg b
    call min
```

```

    ....
    ....
end

```

The following processor ABI attributes can be used to control the function call ABI.

```

pcstack = {stack,lr}
    // represents where is the return pc stored.
sp = <reg>
lr = <reg>
sp_adj = {pre, post}
sp_incr = <value>
    //represents how many bytes to be added or subtracted to/
    from stack pointer, for the function call.

```

Argument Passing

The actual arguments can be passed to a function as below,

```
param_arg <argument>
```

The following processor ABI attributes can be used to control the argument passing.

```

argstack = {stack,regs:stack}
    //represents where are the arguments stored.
asp = <reg>

```

```
regs = <reg-list>  
asp_adj = {pre, post}  
ksp_incr = <value>
```

Where,
<reg-list> is a list of registers used for passing arguments.

Attributes for registers in reg-list.

type	-	Type for which this register can be used.
width	-	List of widths for which this register can be used.
nomatch	-	If this is not a valid match, action to be taken.
stackc	-	Continue using stack for rest of arguments.
stacku	-	Use stack for this argument alone.
regc	-	Continue trying from next register.
regt	-	Try next register for this argument alone.

Each register in the list can be specified with attributes to decide on whether the register can be used for passing the next argument.

2.2 SANKHYA Assembler

2.2.1 Overview

SANKHYA Assembler (*sasm*) performs two passes on the input assembly file. During the first pass, the symbol table is constructed, with the value of the symbols being the labels used in the assembly program.

During the second pass, the assembler produces the binary codes for each assembly statement. It also creates the relocation table that has information relating to symbols, instructions and the address to patch. The assembler produces the relocatable object file.

2.2.2 SANKHYA Assembler features

The following are the significant features of SANKHYA Assembler.

- Interoperability with GNU gcc compiler for MIPS32
- Command-line based operation and support for command-line options.
- Support for little-endian and big-endian processor variants
- Support for different assembly language formats
- Generates ELF object files.

2.2.3 Assembler syntax

An Assembler syntax can be any of the following:

Character set

Assembler recognizes alphabetical characters from A-Z and a-z, numeric characters from 0-9 and special characters like comma, semicolon, # and

%. Alphabetic characters are case-sensitive.

Symbols

Symbols are used for naming. It is written as a label name immediately followed by a colon (':'). The symbol then represents the current value of the active location counter. Symbol names begin with an alphabet.

Example: func:

Operators

Operators are special keywords which conduct an operation on operands.

Example : {+, -, /,}

Expression

An expression is a sequence of one or more symbols, constants, variables, array elements separated by the operators. Prefix notation is used for representing expressions. Example: =a + b c

2.2.4 Statements

An Assembler statement can be any of the following:

Instruction statements

This actually represents assembly instructions

Example: add r1,r2,r3

Directives

Directives instruct the assembler how to output code but are not assembly instructions. They begin with a period ('.') followed by characters in lower case.

Example : '.data' tells the assembler that the subsequent statements are to be assembled at the end of data section.

Comments

A comment statement is not processed by the assembler. A comment statement begins with a `'//'`.

2.2.5 SANKHYA Assembler Directives

Following are the directives supported by the SANKHYA Assembler for MIPS and ARM.

2.2.5.1 .text

Syntax : `.text`

Description:

Indicates that the code after this directive should be added to the `.text` section.

Usage: `.text`

2.2.5.2 .data

Syntax : `.data`

Description:

Indicates that the code after this directive should be added to the `.data` section.

Usage: `.data`

2.2.5.3 .align

Syntax : .align expression

Description:

Aligns the current location to a specified boundary. The extra space (if any) created due to this alignment is filled with zeroes.

Usage: .align 4

2.2.5.4 .space

Syntax : .space expression

Description:

Reserves the number of memory bytes specified by the expression. The memory thus reserved is set to zeroes.

Usage: .space 255

2.2.5.5 .word

Syntax : .word expr1 [,expr2]...[,exprn]

Description:

Aligns the expression with 2 byte alignment. This is similar to .align 2 .

Usage: .word 5
.word L1

2.2.5.6 .half

Syntax : .half expr [,expr2],...[,exprn]

Description:

The expression value in 'expr [,expr2]...[,exprn]' is truncated to a 16-bit value and assembled in successive locations.

Usage: .half 4
.half L2

2.2.5.7 .4byte

Syntax : .4byte expr1 [,expr2]...[,exprn]

Description:

The expression value in 'expr1 [,expr2]...[,exprn]' is truncated to a 32-bit value and assembled in successive location.

Usage: .4byte 5
.4byte L1-L2

2.2.5.8 .byte

Syntax : .byte expr1 [,expr2]...[,exprn]

Description:

The expression value in 'expr1 [,expr2]...[,exprn]' is truncated to a 8-bit value and assembled in successive location.

Usage: `.byte 5`
`.byte L1-L2`

2.2.5.9 `.2byte`

Syntax : `.2byte expr1 [,expr2]...[,exprn]`

Description:

The expression value in 'expr1 [,expr2]...[,exprn]' is truncated to a 16-bit value and assembled in successive location.

Usage: `.2byte 5`
`.2byte L1-L2`

2.2.5.10 `.size`

Syntax : `.size name, expression`

Description:

Sets the size of the object specified as 'name' to the value specified in 'expression'.

Usage: `.size i,4`

2.2.5.11 `.comm`

Syntax : `.comm name, expression`

Description:

Specifies that the 'name' is a global common symbol with size specified by 'expression'.

Usage: .comm i,4

2.2.5.12 .extern

Syntax : .extern name [, <expression>]

Description:

Instructs the assembler that the symbolic name is not defined in the current assembly. Mips Assembler takes one more argument <expression> to .extern directive that specifies the size of the symbolic name in bytes.

Usage: .extern i,5
.extern j

2.2.5.13 .type

Syntax : .type name, value

2.2.5.14 Description:

Specifies the type of the object 'name' as 'value'.

Usage: .type add, function

2.2.5.15 .ascii/.asciz

Syntax : .ascii/.asciz string [,string]...

Description:

Assembles each string specified as arguments in consecutive locations. Each string must be specified within quotes. The difference between `.ascii` and `.asciz` directive is `.asciz` null pads the string whereas `.ascii` does not null pad the string.

Usage: `.ascii "Hello\n"`
`.asciz "Hai"`

2.2.5.16 .section

Syntax : `.section name`

Description:

Creates a section with the specified 'name'.

Usage: `.section Y`

2.2.5.17 .file

Syntax: `.file <file name>`

Description:

Specifies the source file used for generating the assembly file.

Usage: `.file "add.i"`

2.2.5.18 .equiv

Syntax: .equiv <name> <value>

Description:

Assigns the value specified by 'expr' to the symbolic name. The value can be a register-relative address, program-relative address, absolute address, constant and an expression. This directive will issue an error message if the symbolic name is already assigned to a value.

Usage: .equiv L1 10

2.2.5.19 .lcomm

Syntax : .lcomm name, expression

Description:

The .lcomm directive instructs the assembler to allocate the symbol 'name' of length 'expression' in the bss area.

Usage: .lcomm i,8

2.2.5.20 .p2align

Syntax: .p2align <expr1> [<expr2> <expr3>]

Description:

Instructs the assembler to pad the location to a storage boundary specified by 'expr1' as a power of 2. 'expr2' specifies the padding bytes whereas the 'expr3' specifies the maximum number of bytes that must be skipped.

Usage: .p2align 5

2.2.5.21 .skip

Syntax: .skip <expr1> [<expr2>]

Description:

This directive instructs the assembler to set bytes of size specified by 'expr1' to the value 'expr2'.

Usage: .skip 8

2.2.5.22 .short

Syntax: .short expr1 [, expr2, ..., exprN]

Description:

Aligns the expressions with 2 byte alignment. Equivalent to .word directive.

Usage: .short 4

2.2.5.23 .hword

Syntax: .hword <expr>

Description:

The .hword directive emits 16 bit number for the expression specified in 'expr'.

Usage: .hword

2.2.5.24 .err

Syntax: .err

Description:

.err instructs the assembler that an error has occurred. Assembler stops assembling, when .err directive is encountered.

Usage: .err

2.2.5.25 .equ

Syntax: .equ name expr {type}

Description:

Assigns the value specified by 'expr' to the symbolic name. The value can be a register-relative address, program-relative address, absolute address, constant and a expression

Usage: equ a 10
 equ abc L3+10

2.2.5.26 .abort

Syntax: .abort

Description:

Instructs the assembler to stop the assembly.

Usage: .abort

MIPS Specific Directives:**2.2.5.27 .globl**

Syntax : .globl <symbol>

Description:

<symbol> after the .globl directive is defined as an external symbol.

Usage: .globl F1

2.2.5.28 .end

Syntax : .end proc_name

Description:

Specifies the end of a procedure

Usage: .end add

2.2.5.29 .ent

Syntax : .ent proc_name

Description:

Specifies the beginning of a procedure.

Usage: `.ent add`

2.2.5.30 .gpword

Syntax : `.gpword local-sym`

Description:

This directive aligns the symbol specified by 'local-sym' in 2 byte alignment. It sets the relocation entry for the local-sym as of type R_MIPS_GPREL32.

Usage: `.gpword 4`
`.gpword L2`

2.2.5.31 .sdata

Syntax : `.sdata`

Description:

Assembler adds the data after the `.sdata` directive to the `.sdata` section.

Usage: `.sdata`

2.2.5.32 .rdata

Syntax : `.rdata`

Description:

Assembler adds the data after the `.rdata` directive to the `.rdata` section.

Usage: `.rdata`

ARM Specific Directives**2.2.5.33 .global**

Syntax : `.global <symbol>`

Description:

`<symbol>` after the `.global` directive is defined as an external symbol which can be used by the linker for resolving external references.

Usage: `.global F1`

2.2.5.34 .code

Syntax: `.code 16|32`

Description:

This directive instructs the assembler to treat the subsequent instructions as 16 bit THUMB instructions (`.code 16`) or 32 bit ARM instructions (`.code 32`) depending upon the option specified.

Usage: `.code 16`

`.code 32`

2.2.5.35 `.thumb`

Syntax: `.thumb`

Description:

This directive instructs the assembler to treat the subsequent instructions as 16 bit THUMB instructions. It is equivalent to `.code 16`.

Usage: `.thumb`

2.2.5.36 `.arm`

Syntax: `.arm`

Description:

This directive instructs the assembler to treat the subsequent instructions as 32 bit ARM instructions. It is equivalent to `.code 32`.

Usage: `.arm`

2.2.5.37 `.balign`

Syntax: `.balign <expr1> [, <expr2>] [, <expr3>]`

Description:

Instructs the assembler to pad the location to a storage boundary specified by 'expr1' in bytes. 'expr2' specifies the padding bytes whereas the 'expr3' specifies the maximum number of bytes that must be skipped.

Usage: `.balign 16`

2.2.5.38 .set

Syntax: .set <option>

Description:

This directive enables the 'option' for the subsequent assembly instructions.

Usage: .set macro

2.3 SANKHYA Linker

2.3.1 Overview

SANKHYA Linker for (*sld*) reads and combines several ELF object files into a single executable (or absolute) file.

sld can combine any number of relocatable ELF object files and can produce a single executable file (ELF or S-record). The linker relocates based on the relocation table information.

2.3.2 SANKHYA Linker features

Linker supports the following script based operations:

- specify starting address for sections.
- specify input and output files.
- specify standard include directory path.
- assign values to symbol.
- specify the layout of output object file.
- specify the entry point of program.
- specify input/output object file format.
- specify memory allocation.

2.4 SANKHYA Simulator

2.4.1 Overview

SANKHYA Simulator (ssim) is an instruction set simulator that can load ELF absolute files and executes the instructions.

2.4.2 SANKHYA Simulator features

The following are the significant features of SANKHYA Simulator

- Instruction set simulation
- Command-line operation
- Support for all common simulator commands like load, memset, show register value, disassemble and support for system commands like: list model names, set current cpu, simulate a new cpu.
- Support for execution of commands from a script/command file
- Support for little-endian and big-endian processor variants.

2.4.3 Simulator Commands

The following Simulator commands are supported

```
> cpu <label> <Processor Model>
```

- to simulate a cpu of 'Processor Model' with the name 'label'.

Example: `cpu c MIPS32`

```
> disasm <saddr> <no-of-lines>
```

- prints 'no-of-lines' disassembly of instructions starting from

the address 'saddr'

Example: `disasm 0x400030 2`

> `load <exefile>`

- to load a executable 'exefile'

Example: `load test.x`

> `mblock <id> <size> <access mode>`

- to create a memory block of the specified name 'id' with the specified 'size' and 'access_mode'.

Example: `mblock m 0x20000 rw`

> `memget <addr>`

- to get the memory value at the specified address 'addr'.

Example: `memget 0x400030`

> `memset <addr> <value>`

- to set the 'value' to the specified address 'addr'

Example: `memset 0x400030 0x27de0010`

> `model <model file>`

- to load specific processor model

Example: `model test.md`

> `mregion <id> <size> <access mode>`

- to set memory region property.

Example: `mregion c m 0x400000 rw`

> show <regname>

- to show the specified register value

Example: show r2

> set <regname> <val>

- to set register value

Example: set r4 0x10

> set <cpu label>

- to set current cpu

Example: set cpu c

>step [<count>]

- to step through each machine instruction

- performs multiline step according to step count option

Example: step 8

> run [<address>]

- to execute the program from PC until exception or
breakpoint is reached

- to execute the program from PC until the address is reached

Example: run 0xffffffff

> inc <simulator command file>

- to execute commands from the given command file

Example: inc test.cmd

> quit

- to exit the simulator.

> help

- to show help for simulator commands.

> trace <mode>

- to display the instructions/commands executed in the simulator if the mode is set as 'on'. By default, trace mode is not enabled.

Example: trace on

2.5 SANKHYA Librarian

2.5.1 Overview

SANKHYA Librarian (slib) groups any number of input object files in ELF format and generates a single archive file. Its main function is to create and update archive files with printable and portable headers.

The archive file has the following structure,

```
archive magic string
archive symbol table member header
archive symbol table file member
archive string table member header(if present)
archive string table file member (if present)
archive file member1 header
archive file member1
:
:
:
:
archive file memberN
```

Each archive begins with archive magic string,

```
#define ARMAG "!<arch>\n" /* magic string */
#define SARMAG 8 /* length of magic string */
```

The symbol table has all the common, global symbols in the input object file. Using the symbol table, linker identified the undefined symbols while creating a archive.

It has the following format

<no of symbols> <array of offsets into archive file> <array of symbols>

2.5.2 SANKHYA Librarian features

- Generates and updates archives and libraries.
- Interoperates with GCC archiver 'ar'.

2.6 SANKHYA Dumper

2.6.1 Overview

SANKHYA Dumper '*sdump*' displays information about the input object file or archive files.

SANKHYA Dumper supports 'elf32' file formats for both 'big' and 'little' endianness values.

It displays information on all file headers, global symbol table archive files. For an object file, it displays informations like file header, section headers, section contents, segment list, program headers, string table, symbol table and relocation.

It also displays 'disassembly' of the instructions along with code section contents.

2.6.2 SANKHYA Dumper features

The following are the features of SANKHYA Dumper

- Lists the object files in the archive file and object file header information.
- Displays the global symbol table of an archive file.
- Displays all the required informations about object file, like file header, section header, program headers, section contents and symbol table information

- Displays the section header information of object file.
- Displays the program header information of executable file.
- Displays target relocation informations and provides target relocation name.
- Displays symbol table information of the object file.
- Displays the content of section <number> in hexadecimal format. It also displays 'disassembly' information along with code section content display.
- Provides processor machine description information to perform disassembly
- Provides relocation information of target to display target dependedent relocation names.

Appendix A - SANKHYA Code Generator Error Codes

ERROR NUMBER	ERROR INFORMATION	ERROR CODE SEVERITY
01	Processor Stack Pointer Register is NOT available.	ECS_FATAL_ERROR
02	Processor Frame Pointer Register is NOT available.	ECS_FATAL_ERROR
03	Processor Return Register is NOT available.	ECS_FATAL_ERROR
04	Processor Parameter Registers is NOT available.	ECS_WARNING
05	Processor Temporary Registers is NOT available.	ECS_WARNING
06	Processor Register Allocation Failed.	ECS_FATAL_ERROR
07	Processor ABI info is NOT available	ECS_FATAL_ERROR
08	No such file or directory	ECS_WARNING
09	No input file	ECS_ERROR

ERROR NUMBER	ERROR INFORMATION	ERROR CODE SEVERITY
10	Not valid input file	ECS_ERROR
11	Processor Prolog and Epilog info is NOT available	ECS_WARNING
12	Function Return type mis-matched	ECS_ERROR

Appendix B - DTF Technology

DTF Technology consists of the SANKHYA Machine Description Language (SMDL), SMDL parser and C++ libraries, that can be used to build various tools including

- Code Generator
- Assembler
- Linker
- Instruction and Cycle-Accurate CPU Simulator

All of these use the exact same machine description of a processor expressed using SMDL.

DTF Technology is ideal for Processor Architects and designers. As SMDL files can be modified for incremental changes to a processor in a matter of minutes, CPU designers can experiment with SMDL and optimize processor designs. DTF based tools can support different kinds of processors including VLIW, DSP, RISC and CISC.

DTF Technology can be used to quickly build software tools like Binary Translators, Dis-assemblers, De-compilers and object format converters.

Appendix C - SANKHYA Librarian Error Messages

ERROR NUMBER	ERROR INFORMATION	DESCRIPTION
01	arname is not an archive.	The archive doesn't have either .a or .lib extestion or archive magic string.
02	insufficient argument, object file not found.	The arguments in the command line are insufficient.
03	fname file not found.	The given file is not present in the specified directory.
04	arname, archive not found.	The specified archive is not present in the directory.
05	No input file specified.	Archive name or object file names are not specified in the command line except -h and -V
06	cannot create archive file.	The specified directory may not have write permissions.

Index

A

Access Attribute 61
Arithmetic operators 71
Assembler 1
assembler 12
Assembler syntax 105
Attributes 59, 60
attributes 100

B

Bit field extraction 89
Bitwise operator 78
Branch Instructions 91

C

Character set 105
Code Generator 1
Comments 107
Compare and jump 94
Compare and jump operators 94
Compare and Set 92
Compare and set 92
Compare and set operator 92
Conditional expression 98
Conditional jump 95
Constant 59
Cycle Attribute 62

D

default extensions 7, 10, 14, 18
Delay slot attribute 62
Directives 106
DWARF 105

E

ELF object file 105

Expressions 96, 106

F

Function call 102

Functions 98

G

Global variable 58

Global variables 58

I

Instruction statements 106

Instructions 90

Intermediate code 5, 8

Intermediate code Specification 53

J

Jump 92

Jump operator 92

L

Label 96

Labels 68

Linker 1, 2, 20, 23

linker 121

Literal 69

Local variable 57

Logical operators 73

M

Memory 70

N

Nullify attribute 63

O

Operands 68

Operators 70, 106

Operators and arity values 97

P

Parser 1

Pointer 67

Pointer dereference 68

Pop 90

PUSH 90

Push 90

R

Register 69, 70

Register pair 67

Relational operator 75

Relational operators 75

relocation table 105

Return 96

S

SANKHYA Assembler 12, 105

SANKHYA CODE GENERATOR 5

SANKHYA Code Generator 5, 53

SANKHYA Linker 20, 121

SANKHYA SIMULATOR 27, 35, 42, 50, 122, 126, 128

SANKHYA Simulator 27, 35, 42, 50, 122, 126, 128

Sankhya Tools Collection 1

sasmmips 2, 12, 27, 31, 105

scgmips 2, 5, 12, 16, 53

Simulator 1, 27

Simulator Commands 122

sldmips 2, 121

ssimmips 2, 27, 122

Statements 106

stc.cmd 4

stc.csh 3

symbol table 105

Symbols 69, 106

T

Types qualifier 59

U

Unconditional jump 94

V

Variable 57

Volatile 59

For More Information

STC Download	http://www.sankhya.com/info/products/tools/download.html
STC Documentation	http://www.sankhya.com/info/products/tools/docs.html
STC Sales & Support	sales@sankhya.com

SANKHYA

Sankhya Technologies Private Limited
#13/2, "JayaShree", Third Floor, First Street, Jayalakshmpuram,
Nungambakkam,
Chennai 600 034, INDIA
Tel: +91 44 2822 7358
Fax: +91 44 2822 7357

Sankhya Technologies India Operations Private Limited
#30-15-58,"Silver Willow",Third Floor,
Dabagardens
Visakhapatnam 530 020, INDIA
Tel:+91 891 554 2666
Email: sales@sankhya.com
<http://www.sankhya.com>

SANKHYA, SANKHYA TECHNOLOGIES, SANKHYA Tools Collection, Dynamically Targetable Tools Framework, SANKHYA Software are trademarks, servicemarks or registered trademarks of Sankhya Technologies Private Limited. All other brands and names are the property of their respective owners.
