COMPAQ

Alpha Microprocessors Motherboard Software Design Tools

User's Guide

Order Number: EC–QHUWE–TE

Revision/Update Information: This is a revised document. It supersedes the *Alpha Microprocessors Motherboard Software Design Tools User's Guide*, EC–QHUWD–TE.

Compaq Computer Corporation

April 1999

Contents

Preface

1 Introduction

2 Installation and Setup

3 ALIST

4 ASTRIP

5 CLIST

6 CSTRIP

7 GNU Assembler

8 HAL Assembler

9 HEX32

10 HEXPAD

11 HFCOMP

12 IC4MAT

13 MAKEROM

14 PALcode Violation Checker

15 RCSV

16 SREC

17 SROM Packer

18 SYSGEN

19 ULOAD

20 XLOAD

A Support

Index

Figures

Tables

Preface

Introduction

This document describes the toolset used to develop Alpha microprocessor motherboard firmware.

Audience

The Alpha Microprocessors Motherboard Software Design Tools are for tool developers and designers who use the following Alpha microprocessors:

- 21264 (AlphaPC 264DP)
- 21164PC (AlphaPC 164SX)
- 21164 (AlphaPC 164LX)

Content Overview

The information in this document is organized as follows:

- Chapter 1 is a general overview of the software design tools.
- Chapter 2 is an overview of the tools, and it provides information about installation and sample files.
- Chapter 3 through Chapter 20 describe the tools created or modified for the Alpha Microprocessors Software Design Tools Kit.
- Appendix A contains information about customer support and associated documentation.

Conventions

The following conventions are used in this document:

Convention	**Definition**
A percent sign (%)	Indicates the Tru64 UNIX operating system command prompt.
A greater than sign (>)	Indicates the Windows NT operating system command prompt.
A greater than sign and a percent sign (>%)	Indicates that a command is supported in Windows NT and the Tru64 UNIX operating systems.

Convention	Definition
Square brackets ([])	In a command format, denote optional syntax. In bit fields, denote extents when used with pairs of numbers separated by a colon. For example, [7:3] specifies bits 7, 6, 5, 4, and 3.
Parentheses ( )	In command formats, indicate that if you choose more than one option, you must enclose the choices in parentheses.
Boldface type	Indicates commands and examples of user input.
Italic type	Emphasizes important information, indicates variables in command syntax, and indicates complete titles of manuals.
`Monospaced type`	Indicates an operating system command, a file name, or directory pathname.

1
Introduction

1.1 Overview

This document describes tools that have been modified or created for designers who develop firmware for an Alpha microprocessor. With these tools, you can verify your PALcode and produce data to program SROMs in Intel Hex and Motorola S-record formats.

1.2 Software Design Tools Summary

Table 1–1 summarizes the tools developed or modified for the software design tools.

Table 1–1 Software Design Tools Summary

Tool Name	Purpose	Input	Output	Options
ALIST	Produces a listing of disassembled code plus symbolic information	a.out object file	List file (default), -e entry point file, -m PVC map file	-v, -h, -f
ASTRIP	Strips header	a.out object file	Stripped object file (executable)	-a, -v, -h, -n, -r
CLIST	Produces a listing of disassembled code plus symbolic information	coff format object file	List file (default), -e entry point file, -m PVC map file	-v, -h, -f
CSTRIP	Strips header	coff format file	Stripped object file (executable)	-a, -v, -h, -n, -r
GAS	GNU-based assembler (for 21164 and SROM code for 21264)	Source	a.out (default)	-P, -o, -l, -v, -21164, -21264
HAL	Hudson assembler linker (21264)	Source	Binary file and optional list file	-o, -O, -l, -d, -c, -p#, -D, -q, -i, -e, -e#, -C, -s#, -h, -wa, -nt
HEX32	Generates Intel Hex32 output	Executable file	Intel Hex32 file (.hex)	-v, -o
HEXPAD	Adds padding to a Hex file	a.out object file	a.out (default)	-v, -h, -x, -b
HFCOMP	Compresses an input file	System ROM file	Compressed file	-v, -h, -t, -21264, -21164, -21066, -21064

Table 1–1 Software Design Tools Summary (Continued)

Tool Name	Purpose	Input	Output	Options
IC4MAT	Generates an Icache image file and attaches SROM and write-once chain output	Stripped binary executable file	Image file	-21264, -21164, -21164PC, -v, -s, -l, -a, -d, -b, -m, -p, -h
MAKEROM	Builds a ROM image	ROM image files	-o output file	-l, -c, -x, -s, -f, -i, -v, -h, -r
PVC	Checks for PALcode violations	Executable file, entry point file, map file	Log	Not applicable
RCSV	Generates an output file that can be used as an include file	Source file	Include file	-h, -v
SREC	Generates S-record format code	-a a.out object file, -i executable file	Motorola S-record format (.sr)	-v, -h, -o
SROM	For the 21164, generates SROM code	Executable file	Intel Hex format (.hex)	-v, -h, -21164PC -21164, -21064
SYSGEN	Builds an image	-a a.out, -c coff format, -s stripped format	-o executable image file	-v, -h, -e, -p
ULOAD	On Tru64 UNIX, downloads a file through the serial port	ROM image files	—	-load_address, -serial_port, -baud_rate, -xb
XLOAD	On Windows NT, downloads a file through the serial port	ROM image files	—	fast

2

Installation and Setup

2.1 Overview

The Alpha Microprocessors Motherboard Software Design Tools are supported on Alpha systems running the Tru64 UNIX or Windows NT operating system. To install the tools, refer to the *Alpha Motherboards Software Developer's Kit Read Me First.*

2.2 Tools Created or Modified

Table 2–1 lists the tools that have been created or modified for the software design tools and the operating systems that currently support them.

Table 2–1 Tools and Supported Operating System

Tool Name	Description	Operating System
ALIST	Generates a listing file from C source and its associated assembler	Tru64 UNIX, Windows NT
ASTRIP	Strips header information from an a.out format executable file	Tru64 UNIX, Windows NT
CLIST	Produces a listing from coff format	Tru64 UNIX
CSTRIP	Strips header information from a coff format executable file	Tru64 UNIX
GAS	GNU-based assembler	Tru64 UNIX, Windows NT
HAL	Hudson assembler linker	Tru64 UNIX, Windows NT
HEX32	Generates Intel Hex32 output	Tru64 UNIX, Windows NT
HEXPAD	Adds padding to a Hex file	Tru64 UNIX, Windows NT
HFCOMP	Compresses the specified input file using a Huffman encoding algorithm	Tru64 UNIX, Windows NT
IC4MAT	Converts a stripped binary executable file into an image file suitable for loading into the Icache	Tru64 UNIX, Windows NT
MAKEROM	Builds a ROM image by adding header information and then concatenates the files	Tru64 UNIX, Windows NT

Table 2–1 Tools and Supported Operating System (Continued)

Tool Name	Description	Operating System
PVC	Checks for PALcode violations	Tru64 UNIX, Windows NT
RCSV	Generates an output file that can be used as an include file	Tru64 UNIX, Windows NT
SREC	Takes an arbitrary image and converts it to Motorola S-record format	Tru64 UNIX, Windows NT
SROM	Embeds instruction cache initialization into the executable data and generates Intel Hex format	Tru64 UNIX, Windows NT
SYSGEN	Concatenates the specified input files into one contiguous image	Tru64 UNIX, Windows NT
ULOAD	Downloads a file through the SROM serial port	Tru64 UNIX
XLOAD	Downloads a file through the SROM serial port	Windows NT

2.3 Sample Files

The software design tools include sample files. These files allow users to start up and perform sample runs on the provided tools. For more details, see the Read Me First document supplied with your motherboard.

3

ALIST

3.1 Overview

The ALIST tool produces a listing of disassembled object code and symbolic information from an a.out style object file generated by GAS. ALIST is also used to generate the entry point and map file for PVC.

3.2 Command Format

The basic ALIST command format is:

```
>% alist [-options] [input_file] [> output_file]
```

The following table lists the options:

Option	Designation	Description
v	verbose	Gives more information than usual
h	help	Prints information about how to use ALIST
e	entry points	Produces entry point output for PVC
m	map	Outputs PVC symbols from object file
f	full information	Does not skip the zero location

If ALIST is specified with no options or file information, then ALIST searches the current default directory for an a.out file, generates a listing of that object file, and sends the output to stdout. The list output may be piped to an output file. For example:

```
%  alist osfpal.o > osfpal.lis
```

To produce an entry points file for PVC, enter this command:

```
%  alist -e osfpal.o > osfpal.ent
```

To produce a PVC symbols (.map) file, enter this command:

```
%  alist -m osfpal.o > osfpal.map
```

4

ASTRIP

4.1 Overview

The ASTRIP tool postprocesses the object file produced by GAS for input into PVC, SROM, and SREC. This tool is used to strip header information from the object file.

4.2 Command Format

The basic ASTRIP command format is:

```
>% astrip [-options] input_file [> output_file]
```

The following table lists the options:

Option	Designation	Description
v	verbose	Prints more information than usual.
h	help	Prints information about how to use ASTRIP.
a	—	Strips all sections, data as well as text, from the object file.
n *number*	number	Strips a specified number of bytes from the front of the file; a number must be supplied.
r	round	Rounds the stripped file to an 8-byte boundary. (For example, if the stripped file is 257 bytes long, then the file is rounded to 264 bytes.)

If an output file name is not specified, then the default for the Tru64 UNIX operating system is the input file name with a .strip extension. For the Windows NT operating system, the default extension is .stp.

For example, to produce an executable file format for PVC, enter this command:

```
% astrip osfpal.o > osfpal.nh
```

5

CLIST

5.1 Overview

The CLIST tool produces a listing from the coff format object file.

5.2 Command Format

The basic CLIST command format is:

```
>% clist [-options] [input_file] [> output_file]
```

The following table lists the options:

Option	Designation	Description
v	verbose	Gives more information than usual
h	help	Prints information about how to use CLIST
e	entry points	Produces entry point output for PVC
m	map	Produces PVC symbols from object file
f	full information	Does not skip the zero location

If CLIST is specified with no options or file information, it searches the current default directory for an a.out file, generates a listing of that object file, and sends the output to stdout. The list output may be piped to an output file. For example:

```
% clist sample.o > sample.lis
```

6

CSTRIP

6.1 Overview

The CSTRIP tool postprocesses a coff format object file. This tool strips header and trailer information and leaves the code and initialized data in the output file. The output file can then be loaded onto the motherboard.

6.2 Command Format

The basic CSTRIP command format is:

```
>% cstrip [-options] input_file [> output_file]
```

The following table lists the options:

Option	Designation	Description
v	verbose	Prints more information than usual.
h	help	Prints information about how to use CSTRIP.
a	—	Strips all sections, data as well as text, from the object file.
n *number*	number	Strips a specified number of bytes from the front of the file; a number must be supplied.
r	round	Rounds the stripped file to an 8-byte boundary. (For example, if the stripped file is 257 bytes long, then the file is rounded to 264 bytes.)

If an output file name is not specified, then the default is the input file name with a .strip extension.

7

GNU Assembler

7.1 Overview

The Free Software Foundation GNU assembler (GAS) takes source files as input and assembles them into a.out format object files. GAS has been modified to include support for the PALcode extensions described in the following documents:

- *21264 Alpha Microprocessor Hardware Reference Manual*
- *Alpha 21164PC Microprocessor Hardware Reference Manual*
- *Alpha 21164 Microprocessor Hardware Reference Manual*

Note that GAS is the assembler for the Alpha 21164 and can also be used for the Alpha 21264. The other assembler, HAL, can be used only for the 21264. For information about the HAL assembler, see Chapter 8.

More detailed documentation about GAS is available from the Free Software Foundation.

7.2 Command Format

The basic GAS command format is:

```
>% gas [-options] input_file_list
```

The following table describes the options:

Option	Description
P	Automatically runs the C preprocessor standard with the operating system. This gives support for C macros, defines, and so on.
o *filename*	Specifies the name of the output object file. The default output file name is a.out.
l	Creates a list output. By default, the list output is sent to stdout; however, this output can be piped to a file.
v	Prints the version number.
21164	Generates code for the Alpha 21164 microprocessor family.

The input_file_list element is one or more input file names separated by spaces.

The following example generates an object file for PVC:

```
% gas -P -o osfpal.o osfpal.s
```

The following example generates a list output and pipes it to a file called hwrpb.lis:

```
% gas -l hwrpb.s > hwrpb.lis
```

7.3 PALcode Assembler Instructions Added to GAS

This section contains PALcode assembler instructions that have been added to GAS for the Alpha microprocessors:

- `hw_ld`
- `hw_st`
- `hw_ret`
- `hw_mfpr`
- `hw_mtpr`
- `hw_rei`
- `hw_rei_stall`

GAS also contains additional 21264 instructions in the form of FIX and MVI extensions to the Alpha architecture. See the *Alpha Architecture Handbook* for more information about these extensions.

7.3.1 hw_ld

Hardware load instruction.

```
hw_ld/[options] ra,disp(rb)
```

You can use one or more of the following options:

Option	Description
p	Specifies that the effective address is physical
a	Uses current mode bits in ALT_MODE IPR
r	Read-with-write check on virtual HW_LD instructions
q	Quadword data length
v	Flags a virtual PTE fetch
l	Physical/Lock – The effective address for the **hw_ld** instruction is physical. It is the load lock version of **hw_ld**.
vv (21264 only)	Virtual/VPTE – Flags a virtual VPTE fetch. Used by trap logic to distinguish a single translation buffer (TB) miss from a double TB miss. Kernel mode access checks are performed.
vw (21264 only)	Virtual/WrChk – The effective address for the **hw_ld** instruction is virtual. Access checks for fault-on-read (FOR), fault-on-write (FOW), read and write protection.
vwa (21264 only)	Virtual/WrChk/Alt – The effective address for the **hw_ld** instruction is virtual. Access checks for FOR, FOW, read and write protection. Access checks use DB_ALT_MODE IPR.

The options, if used, must be specified in the order listed in the previous table. For example, it is illegal to list the **q** before the **p**, as shown in the following example.

Incorrect example:

```
hw_ld/qp $3,42($4)
```

Correct example:

```
hw_ld/pq $3,42($4)
```

There are two variants of the **hw_ld** instruction:

```
hw_ldq/[p][a][r][v][l] ra,disp(rb)
hw_ldl/[p][a][r][v][l] ra,disp(rb)
```

hw_ldq is an abbreviation for **hw_ld/q** (quadword), and **hw_ldl** is a variant for the default (longword) condition.

7.3.2 hw_st

Hardware store instruction.

```
hw_st/[options] ra,disp(rb)
```

You can omit options, or use one or more of the following options:

Option	Description
p	Specifies that the effective address is physical
a	Use current mode bits in ALT_MODE IPR
q	Quadword data length
c	Store conditional version of HW_ST
pc (21264 only)	Physical/Cond – The effective address for the **hw_st** instruction is physical. Store the conditional version of the **hw_st** instruction. The lock flag is returned in RA.
v (21264 only)	Virtual – The effective address for the **hw_st** instruction is virtual.
wa (21264 only)	Virtual/Alt – The effective address for the **hw_st** instruction is virtual. Access checks use DTB_ALT_MODE IPR.

Note that RWC is always set to zero for the write and is not listed as an option. Again, the options, if used, must be specified in the order listed in the previous table.

There are two variants of the **hw_st** instruction:

```
hw_stq/[p][a][c] ra,disp(rb)
hw_stl/[p][a][c] ra,disp(rb)
```

hw_stq is an abbreviation for **hw_st/q** (quadword), and **hw_stl** is a variant for the default (longword) condition.

7.3.3 hw_ret

Hardware return instruction. The different types affect stack prediction.

`hw_ret/jmp` `(`*`register)`*

`hw_ret/jsr` `(`*`register)`*

`hw_ret/ret` `(`*`register)`*

`hw_ret/co` `(`*`register)`*

The stall option is set by adding **s** to the **hw_ret** instruction; for example:

`hw_rets/`*`option`* `(`*`register)`*

If stall is set, the fetcher is stalled until the **hw_ret** instruction is retired or aborted. The 21264 will:

- Force a mispredict
- Kill instructions that were fetched beyond the **hw_ret** instruction
- Refetch the target of the **hw_ret** instruction
- Stall until the **hw_ret** instruction is retired or aborted

If instructions beyond the **hw_ret** have been issued out of order, they will be killed and refetched.

You can use one of the following options:

Option	Description
jmp	Specifies to not push the PC onto the prediction stack. The predicted target is PC + (4*DISP[12:0]).
jsr	Specifies to push the PC onto the prediction stack. The predicted target is PC + (4*DISP[12:0]).
ret	Pops the prediction off the stack and uses it as a target.
co	Pops the prediction off the stack and uses it as a target. The PC is pushed back onto the stack.

The following table describes the argument for this instruction:

Argument	Description
register	Specifies the register that contains the return address. You must choose R31 as RA.

7.3.4 hw_mfpr

`hw_mfpr/[options] ra,rc`

You can use one of the following options:

Option	Field	Description
p	PAL	References a PAL_TEMP register
a	ABX	References a register in the Abox (load and store unit)
i	IBX	References a register in the Ibox (instruction fetch and decode unit)

The Alpha 21164 microprocessor family does not support any options for this instruction.

The following table describes the arguments:

Argument	Description
ra	Destination
rc	Index into the appropriate internal processor register set, or, for the 21164 microprocessor family, an index of the desired IPR

For example, to read PAL_TEMP(15) into register 3, enter this instruction:

```
hw_mfpr/p $3,$15
```

7.3.5 hw_mtpr

This instruction is similar in form to **hw_mfpr** except that it is writing.

`hw_mtpr/[options] ra,rc`

You can use one or more of the following options:

Option	Field	Description
p	PAL	References a PAL_TEMP register
a	ABX	References an Abox register
i	IBX	References an Ibox register

The Alpha 21164 microprocessor family does not support any options for this instruction.

The following table describes the arguments:

Argument	Description
ra	Source
rc	Index into the appropriate internal processor register set, or, for the 21164 microprocessor family, an index of the desired IPR

7.3.6 hw_rei

`hw_rei`

This instruction generates a return from PALmode through the exception address IPR.

7.3.7 hw_rei_stall

`hw_rei_stall`

This instruction is the same as **hw_rei** except that it inhibits Istream fetch until the **hw_rei** itself is issued.

This command applies only to the Alpha 21164 microprocessor family.

7.4 GAS and GLD Programming Considerations

If you create multiple object files that need to be linked together to build your image, you want to avoid certain pitfalls.

The role of the linker (GLD) is to concatenate object files and resolve references across object files. Thus, if you have multiple files that require explicit placement of their code, you must perform a monolithic assembly of those object files.

Because GAS aligns code within segments, you must be careful about how you use the .= directive to alter the location counter. For example, to start data at address 2000:

```
.text
 code
.=0x2000
.data
 data
```

If the .= directive is given in the second segment (.data), then you would get the code followed by 0x2000 bytes of space followed by the data. This causes the data to be offset rather than assigned to the specific address (see the following example). This problem is independent of the segment type, so that, if .text and .data were replaced with .text 0 and .text 1, then the results would be the same.

```
.text
 code
.data
.=0x2000
 data
```

Do not rely on the `.align` directive to align code to a page. It is more reliable to use zeros to align code within a page. See the *Alpha Architecture Reference Manual* for more details about pages and page frame numbers (PFNs).

8

HAL Assembler

8.1 Overview

The Hudson Assembler Linker (HAL) is an assembly language for programming the Alpha 21264. Source programs written in HAL MACRO are translated into binary code by the HAL MACRO assembler, which produces a binary file and, optionally, a listing file. HAL MACRO source programs contain a sequence of source statements. The source statements may be any one of the kind shown in Table 8–1.

Table 8–1 HAL MACRO Source Statements

Source Statement	Description
Alpha native-mode instructions	Manipulates data and performs such functions as addition, data conversion, and transfer of control. Instructions are usually followed in the source statement by operands, which can be any kind of data needed for completion. The Alpha instruction set is described in detail in the *Alpha Architecture Reference Manual*.
Direct assignment statements	Equates symbols to values.

Table 8–1 HAL MACRO Source Statements (Continued)

Source Statement	Description
Assembler directives	Guides the assembly process and provides tools for using the instructions. There are two classes of assembler directives: General Assembler Directives • Store data or reserve memory for data storage • Control the alignment of program parts in memory • Specify the methods of accessing memory sections in which a program will be stored • Specify the entry point of the program or its parts • Specify the way in which symbols are referenced • Control the format and content of the listing file • Display informational messages • Control the assembler options that are used to interpret the source program • Call other operating system commands to retrieve more MACRO code MACRO Directives • Repeat identical or similar sequences of source statements throughout a program without rewriting those sequences • Use string operators to manipulate and test the contents of source statements

More detailed documentation about HAL is available in the *HAL V5.00 Reference Manual*.

8.2 Command Format

The basic HAL command format is:

```
>% hal [-options] output_files [-options] input_files
```

The following table describes HAL command line options.

Option	Description
-o *filename*	Specifies the name of the output binary file.
-O *filename*	Also specifies the name of the output binary file; however, the format will be in Tru64 UNIX.
-l *filename*	Creates a list output in the specified file.
-d *filename*	Specifies that after parsing all the MACROS in the input files, output a MACRO library file that contains only the MACRO definitions.
-c *filename*	Reads back in a MACRO library file the definitions created earlier using the d option.
-p#	Specifies the page size to use. The default is 8192 bytes.

Option	Description
-D [bft|all] [file]	Turns on debugging. The flags are: • **b** turns on bison debug information. • **f** turns on flex debug information. • **t** turns on internal symbol debug information. • **all** turns on all debug information. The file argument redirects the internal symbol debug information (**t**) only. Other debug information is always sent to STTDERR.
-q	Runs HAL in quiet mode.
-i *filename*	Can be used before input file names, but not needed by HAL.
-e[#] *string*	Specifies a line of code to be parsed by HAL before any input files are parsed. Enclose the string in double-quotes if it contains spaces. Note that HAL interprets this line as regular MACRO code if you use just "-e". If you use "-e#" (where # is an integer number of 0 or higher), HAL interprets the line as a .CMD_INPUT and places it into command buffer #.
-C	Allows HAL to core/stack dump. Normally, HAL catches this and just prints an error message.
-s#	Specifies the starting address for the link stage. This value can be up to 64 bits and, if not in decimal, you must give the MACRO code radix operator first. If you do not use this option, the default value is 0.
-h *name*	Enables you to pick which Alpha CPU to assemble for. For this version of HAL, the only choice is EV6. Note that you can accomplish the same result by using the .ARCHITECTURE directive.
-wa	Enables HAL to print warning messages if, while evaluating an absolute expression, it proves to be relocatable. Normally, HAL just files those equations and checks their values after link time to see if they differ from earlier ones. If they are the same, no error message is printed.
-nt	Turns off the stack trace printing normally done for each message generated by the .PRINT directive. When this option is not used, the default is EV5.

The following example generates the output file *pal.exe* and the list output *pal.lis* for EV6. The two source files are *file1.mar* and *file2.mar.*

```
% hal -o pal.exe -l pal.lis -h EV6 -i file1.mar file2.mar
```

In the previous example, the `-i` option is used to indicate the input files, but it is not necessary.

8.3 21264 PALcode Assembler Instructions

This section contains PALcode assembler instructions for the 21264 Alpha microprocessor.

8.3.1 hw_ld

Hardware load instruction.

`hw_ld``[`*`size]/[type] register, [displacement] (register)`*

The following table describes the arguments.

Arguments	Description
size	Specifies the size of the memory access. Can be “l” for longword or “q” for quadword.
type (optional)	Specifies the type of memory access to make. If not used, the default is a normal virtual access, unless you use the following choices: p Physical access pl Physical access and load lock type v Virtual and VPTE access w Virtual access with read and write protection checks wa Virtual access with read and write using alt-mode protection checks
register	Specifies the register to write for the memory access. Like other registers in HAL, this can be an absolute expression.
displacement	Specifies an expression whose bottom 12 bits are used as the displacement value. The expression must not contain any undefined symbols and must be an absolute expression.

The following example shows a hardware load that specifies quadword, physical access to register 1, with a displacement value of 20 decimal:

```
hw_ldq/p r1, 20(r2)
```

8.3.2 hw_m*x*pr

Move from/to an internal process register instruction.

`hw_mfpr` *`register, expression [,scoreboard]`*

`hw_mtpr` *`register, expression [,scoreboard]`*

The following table describes the arguments.

Arguments	Description
register	Specifies the register to write/read for the IPR access. Like other registers in HAL, this can be an absolute expression.
expression	Specifies the IPR number that identifies which IPR to access. The expression must contain no undefined symbols and must be an absolute expression. This expression should contain both the index and the scoreboard (16) bits.
scoreboard (optional)	Specifies the scoreboard bit for the IPR access. The expression must contain no undefined symbols and must be an absolute expression. The expression argument already contains the normal scoreboard bits. This field is used whenever other scoreboard bits should be OR'ed in.

The following example shows a move to internal process register 1, specifying the IPR number and the scoreboard class:

```
hw_mtpr r1 EV6__l_CTL
```

The previous example employs a PALcode definition file, which defines EV6__l_CTL as **`<^x11 @8>!^x10>`**, which specifies **`^x11`** as the IPR number and **`^x10`** (class 4) as the scoreboard. This instruction is therefore equivalent to:

```
hw_mtpr r1 ^x11@8, ^x10
```

You can OR in additional scoreboard bits, as shown in the example. Also, because the register is a 16-bit field that contains both the IPR number and the index, **`^x11@8`** places the register number in the correct position in the 16-bit field.

8.3.3 hw_ret*x*

Hardware return instruction. The different types affect stack prediction.

hw_ret (*register)*

hw_ret_stall (*register)*

hw_jmp (*register)*

hw_jmp_stall (*register)*

hw_jsr (*register)*

hw_jsr_stall (*register)*

hw_jcr (*register)*

hw_jcr_stall (*register)*

hw_xxx *destregister, (register), [hint]* The following table describes the arguments.

Arguments	Description
destregister	Specifies the register in which to place the next PC. Like other registers in HAL, this can be an absolute expression. This register is needed only if you use this alternate format. If not, register 31 will be the destination.
register	Specifies the register that contains the return address. Like other registers in HAL, this can be an absolute expression.
hint (optional)	Specifies the hint value to use with the instruction. If not specified, zero will be used. You must specify destregister if you use hint.

The following example shows a hardware return to register 23, with the stall option specified:

```
hw_ret_stall (r23)
```

The *21264 Specifications* contain a complete description of stall behavior.

8.3.4 hw_st*x*

Hardware store instruction.

`hw_st``[`*`size]/[type] register, [displacement] (register)`*

The following table describes the arguments.

Arguments	Description
size	Specifies the size of the memory access. Can be “l” for longword or “q” for quadword.
type (optional)	Specifies the type of memory access to make. If not used, the default is a normal virtual access, unless you use the following choices: p Physical access pc Physical access and store conditional type a Virtual and protection checks done using alt-mode
register	Specifies the register to read for the memory access. Like other registers in HAL, this can be an absolute expression.
displacement	Specifies an expression whose bottom 12 bits are used as the displacement value. The expression must not contain any undefined symbols and must be an absolute expression.

The following example shows a hardware store that specifies quadword, physical access to register 1, with a displacement value of 20 decimal:

```
hw_ldq/p r1, 20(r2)
```

8.4 MAPCVT

The MAPCVT tool processes a HAL output listing file into a .map file that the PALcode Violation Checker (PVC) can read.

```
mapcvt [-pvc] input_file output_file
```

The `-pvc` option creates a .map file that consists only of PVC labels, which is the typical use.

9

HEX32

9.1 Overview

The HEX32 tool generates an Intel Hex32 (MCS86) file from a stripped executable.

9.2 Command Format

The basic HEX32 command format is:

```
>% hex32 [-options] [input_file] [output_file]
```

The following table lists the options:

Option	Designation	Description
v	verbose	Prints more information than usual
o	offset	Specifies image offset

If input and output files are not specified, then stdin and stdout are used.

10

HEXPAD

10.1 Overview

The HEXPAD tool uses an Intel Hex file format (see SROM Packer tool in Chapter 17) to add a specific amount of padding to a file. This tool can be used to fill all unused bytes in an SROM with a known value.

10.2 Command Format

The basic HEXPAD command format is:

```
>% hexpad [-options] input_file [> output_file]
```

The following table lists the options:

Option	Designation	Description
v	verbose	Prints more information than usual
h	help	Prints information about how to use HEXPAD
x	padding size	Specifies padded data size in a hexadecimal format
b	byte	Specifies padding byte

11

HFCOMP

11.1 Overview

The HFCOMP tool compresses the specified input file using a Huffman encoding algorithm to produce a compressed, executable image that will automatically decompress itself to the proper memory location when executed. This tool is intended to allow for more optimal usage of ROM space by reducing the size of ROM images.

When you execute the hfcomp command, the compressed files automatically decompress to the location specified by the -t option. If the compressed files are not loaded at their proper addresses, the decompressed files will relocate to the proper address in memory when the compressed image is executed.

To use the hfcomp command, the EB_TOOLBOX environment variable must be defined to indicate the path to the decompression library files, decmp64.img or decmp164.img. These library files contain the decompression and relocation code that will ensure that the compressed image is in the correct location before it is decompressed.

HFCOMP will automatically append the proper library file to the front of the compressed image based on the -21xxx option specified on the command line. The compressed code will then be located at offset 0x4000 from the beginning of the image. For example, if the Debug Monitor firmware (rom.cmp) is loaded at address 0x300000, then the compressed code begins at 0x304000.

11.2 Command Format

The basic HFCOMP command format is:

```
>% hfcomp [-options] input_file output_file
```

The following table lists the options:

Option	Designation	Description
v	verbose	Gives more information than usual
h	help	Prints information about how to use HFCOMP
t	target	Target location where decompressed image should go (default = 0)
21064 (default)	21064 code	Generate code for the DECchip 21064

Command Format

Option	Designation	Description
21064A	21064A code	Generate code for the DECchip 21064A
21066	21066/21068 code	Generate code for the DECchip 21066/21068
21164	21164 code	Generate code for DECchip 21164
21164A	21164A code	Generate code for DECchip 21164
21164PC	21164PC code	Generate code for DECchip 21164PC
21264	21264 code	Generate code for DECchip 21264

12

IC4MAT

12.1 Overview

The IC4MAT tool is used for the 21264 and performs the same function that the SROM Packer performs for the 21164: it converts a stripped binary executable file into an image file suitable for loading into the Icache. Additionally, this command appends the Cbox information to the front of the ROM. This image typically is loaded into the CPU's SROM serial port upon CPU reset.

12.2 Command Format

IC4MAT has the following command format:

```
>% ic4mat [options] cbox_file exe_file [output_file]
```

If no options are specified, the default condition is to generate an instruction cache image for the Alpha 21264 with a maximum cache size of 8KB with no SROM padding.

The following table lists the options:

Option	Designation	Description
v	verbose	Prints more information than usual.
h	help	Prints help text.
s	simulation	Generates simulation output.
21264	21264	Generates instruction cache image for DC21264.
l *filename* (21264 only)	list	Generates a list file based on the Cbox file read.
a (21264 only)	alias	Places Cbox register alias names instead of the standard register names into the list file.
d (21264 only)	–	Generates a text version of the output file. By default, a binary file is produced.
b (21264 only)	base address	Icache image base address.
m (21264 only)	maximum address	Icache image maximum address.
p (21264 only)	CPU pass	EV6 pass number (default = 3).

Command Format

Option	Designation	Description
21164PC	21164PC	Generates instruction cache image for DC21164PC.
21164	21164	Generates instruction cache image for DC21164.
21064	21064	Generates instruction cache image for DC21064.
21066	21066	Generates instruction cache image for DC21066.
21068	21068	Generates instsruction cache image for DC21068.

Example:

```
% ic4mat -21264 -v -l test.list test.cbox test.exe test.img
```

13
MAKEROM

13.1 Overview

The MAKEROM tool builds a ROM image by adding header information to the input files. Each input file generates one header plus the image, which is then concatenated and written to the output file. These headers are used by the SROM and other software to identify an image contained in the ROM. MAKEROM can also compress these input files using a simple repeating byte compression algorithm. The decompression code is provided in the SROM. Other improved compression techniques that embed appropriate decompression code can also be used, such as the HFCOMP tool.

13.2 ROM Header Information Fields

The ROM header information placed at the beginning of each ROM image contains the fields shown in Figure 13–1.

Figure 13–1 MAKEROM Fields

31	0	Offset	Header Revisions Supported
Validation Pattern 0x5A5AC3C3		0x00	all
Inverse Validation Pattern 0xA5A53C3C		0x04	all
Header Size (Bytes)		0x08	all
Image Checksum		0x0C	all
Image Size (Memory Footprint)		0x10	all
Decompression Flag		0x14	all
Destination Address Lower Longword		0x18	all
Destination Address Upper Longword		0x1C	all
Firmware ID [15:8] Reserved [31:24]	Header Rev [7:0] Header Rev Ext [23:16]	0x20	1+
ROM Image Size		0x24	1+
Optional Firmware ID [31:0]		0x28	1+
Optional Firmware ID [63:32]		0x2C	1+
ROM Offset [31:2]	ROM Offset Valid [0]	0x30	2+
Header Checksum (excluding this field)		0x34	1+

FM-05103.AI4

ROM Header Information Fields

- Validation Pattern

  The first quadword contains a special signature pattern that is used to verify that this “special” ROM header has been located. The validation pattern is 0x5A5AC3C3A5A53C3C.

- Header Size (Bytes)

  The header size is the next longword. This is provided to allow for some backward compatibility in the event that the header is extended in the future. When the header is located, current versions of SROM code determine where the image begins based on the header size. Additional data added to the header in the future will simply be ignored by current SROM code. Additionally, the header size = 0x20 implies Version 0 of this header specification. For any other size, see Header Rev to determine header version.

- Image Checksum

  The next longword contains the image checksum. This is used to verify the integrity of the ROM. Checksum is computed in the same fashion as the header checksum. Although this field was provided with Version 0 of this header specification, the checksum was not really computed until Version 1.

- Image Size

  The image size is used by the SROM code to determine how much of the system ROM should be loaded.

- Decompression Flag

  The decompression flag tells the SROM code if the MAKEROM tool was used to compress the ROM image with a “trivial repeating byte algorithm.” The SROM code contains routines that perform this decompression algorithm. Other compression/decompression schemes may be employed that work independently from this one.

- Destination Address

  This quadword contains the destination address for the image. The SROM code will begin loading the image at this address and subsequently begin its execution.

- Header Rev

  The revision of the header specifications used in this header. This is necessary to provide compatibility to future changes to this header specification. Version 0 headers are identified by the size of the header. See Header Size. For Version 1 or greater headers, this field must be set to a value of 1. The header revision for Version 1 or greater headers is determined by the sum of this field and the Header Rev Ext field. See Header Rev Ext.

- Firmware ID

  The firmware ID is a byte that specifies the firmware type. This information facilitates image boot options necessary to boot different operating systems.

Firmware ID	Firmware Type (decimal)	Description
DBM	0	Alpha Motherboards Debug Monitor firmware
WNT	1	Windows NT firmware
SRM	2	Alpha System Reference Manual Console
FSB	6	Alpha Motherboards Fail-Safe Booter
Milo	7	Linux Miniloader
VxWorks	8	VxWorks Real-Time Operating System
SROM	10	Serial ROM

- Header Rev Ext

  The header revision for Version 1 or greater headers is determined by the sum of this field and the Header Rev field. See Header Rev.

- ROM Image Size

  The ROM image size reflects the size of the image as it is contained in the flash ROM. See Image Size.

- Optional Firmware ID

  This optional field can be used to provide additional firmware information such as firmware revision or a character descriptive string of up to 8 characters.

- ROM Offset

  This field specifies the default ROM offset to be used when programming the image into the ROM.

- ROM Offset Valid

  The lower bit of the ROM Offset Valid must be set when the ROM Offset field is specified. When no ROM Offset is specified, the ROM Offset and ROM Offset Valid fields will contain zero.

- Header Checksum

  The checksum of the header is used to validate the presence of a header beyond the validation provided by the validation pattern. See Validation Pattern. The header checksum is computed from the beginning of the header up to but excluding the header checksum field itself. If there are future versions of this header, the header checksum should always be the last field defined in the header. The checksum algorithm used is compatible with the standard BSD4.3 algorithm provided on most implementations of UNIX.

13.3 Command Format

The basic MAKEROM command format is:

```
>% makerom [-options][-input_file_options] input_file -o output_file
```

The following table lists the options:

Option	Designation	Description
v	verbose	Gives more information than usual
h	help	Prints information about how to use MAKEROM
r	offset	Provides optional offset into the ROM where image is located
o	output file	Specifies output file

The following table lists input_file_options:

Option	Designation	Description
l*address*	load	Specifies destination address.
c	compress	Compresses this file. Default is no compression.
x*value*	—	Sets the optional firmware ID field to the specified hexadecimal value.
s*string*	—	Sets the optional firmware ID field to the specified string.
f*file*	file	Sets the optional firmware ID field from information supplied in the specified file. The file must contain either a hexadecimal value or a quoted ASCII string.
i*fw_id*	—	Specifies the firmware type_number or type_name.

The following example shows the predefined firmware types:

```
%  makerom -v -iDBM -ftimestmp.fw -l300000 rom.cmp -o rom.rom
makerom [V2.0]
...Output file is rom.rom
...processing input file rom.cmp
Image padded by 3 bytes
Header Size......... 52 bytes
Image Checksum...... 0x1c7d (7293)
Image Size (Uncomp). 122032 (119 KB)
Compression Type.... 0
Image Destination... 0x0000000000300000
Header Version...... 1
Firmware ID......... 0 - Alpha Motherboard Debug Monitor
ROM Image Size...... 122032 (119 KB)
Firmware ID (Opt.).. 0104009504181217  .......
Header Checksum..... 0x0b8d
% cat timestmp.fw
0104009504181217
Version: 1.4 950418.1217
```

14

PALcode Violation Checker

14.1 Overview

The PALcode Violation Checker (PVC) tool checks assembly language code for instruction sequences that could cause unexpected results, and produces warning messages that describe the violation.

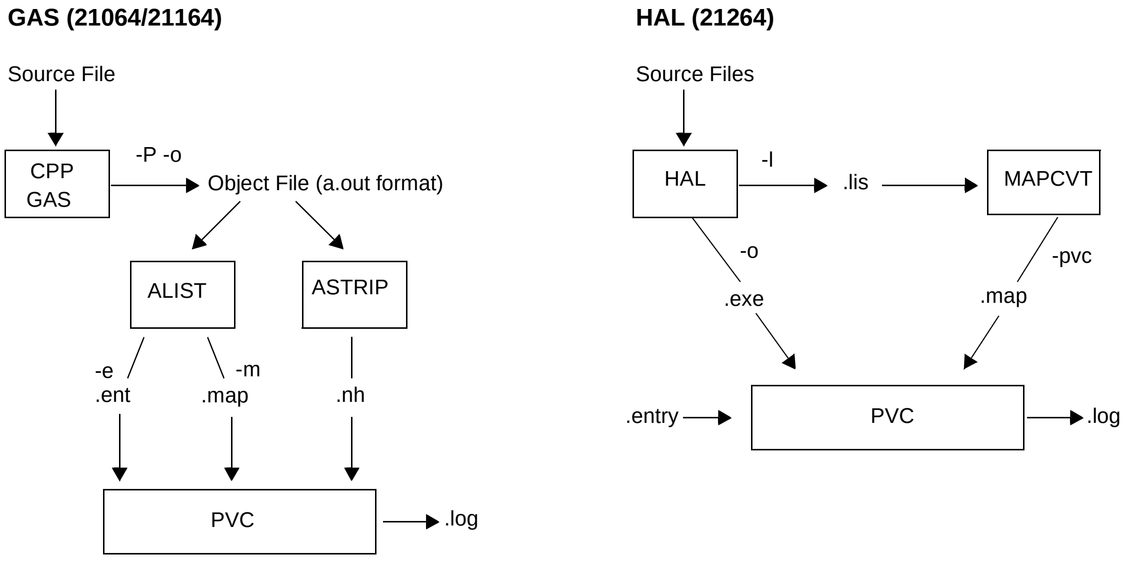
14.2 PVC Input Files

Three input files are required by PVC:

- An executable PALcode image (.exe or .nh)
- A set of PALcode entry points (.ent or .entry)
- A description of PVC symbols (.map)

To generate input files for the Alpha 21064 and Alpha 21164, you need to take the PALcode source and generate an object file with GAS. To generate input files for the Alpha 21264, you need to take the PALcode sources and generate .lis and .exe files with HAL. For more information about GAS, see Chapter 7. For more information about HAL, see Chapter 8. Figure 14–1 shows the PVC tool map for both assemblers.

Figure 14–1 PVC Tool Map for GAS and HAL



14.3 Generating PVC Input Files with GAS

Use the following steps to generate PVC input files with GAS, as shown in Figure 14–1.

1. To generate an object file, preprocess the PALcode source file with the C preprocessor (CPP), and then run GAS. Or, combine these two steps by using the GAS -P option. For example:

   ```
   % gas -P -o filename.o filename.s
   ```

   This produces an object file used as input for the ALIST and ASTRIP tools to produce the PVC input files.

2. To generate the executable PALcode image file, use ASTRIP to postprocess the GAS object file. This extracts the machine code instructions and strips header information. The following example generates an executable file with no header (.nh) for PVC:

   ```
   % astrip filename.o > filename.nh
   ```

3. To generate an entry points file, use ALIST to postprocess the GAS object file. The following example generates an entry points file for PVC:

   ```
   % alist -e filename.o > filename.ent
   ```

   Note: An entry points file generated by ALIST may require some editing to remove entries that are not legal PAL entry points (for example, local labels).

   The legal PAL entry points are defined in the Alpha Hardware Reference Manuals (specific to your CPU) listed in Appendix A.

   The file format is:

   ```
   offset_value(hex)     pal_entry_point_label
   ```

   Note that offset_value is the offset from the base of the executable code. For example:

   ```
   0000  PAL$RESET
   0020  PAL$MCHK
   0060  PAL$ARITH
   00e0  PAL$INTERRUPT
   ```

4. To generate a description of PVC symbols derived from labels in the PAL source code file, use the ALIST tool again. The file name for the .map file should match the file name for the .nh file so that it can be called in automatically with the executable file. For example:

   ```
   % alist -m  filename.o  >  filename.map
   ```

   The format of the output .map file generated by the ALIST tool is:

   ```
   label    address
   ```

   For example:

   ```
   pvc$osf11$5000 00004298
   pvc$osf28$5000.1 00004430
   pvc$osf29$5000.2 000044B8
   pvc$osf0$3000 000053BC
   pvc$osf1$3000.1 000053C0
   pvc$osf2$3000.2 000053D0
   pvc$osf3$3000.3 000053E0
   ```

```
pvc$osf4$3000.4 000053F0
pvc$osf5$3000.5 00005400
pvc$osf6$3000.6 0000540C
pvc$osf31$84 000056F0
```

14.4 Generating PVC Input Files with HAL

Use the following steps to generate PVC input files with HAL, as shown in Figure 14–1. For complete information about HAL, see the *HAL V5.00 Reference Manual*, included in the documentation directory of the SDK CD-ROM.

1. Generate a .exe and a .lis file, as shown in the following example:

   ```
   % hal -o filename.exe -l filename.lis filename1.mar filename2.mar
   ```

   The executable PALcode image file contains machine code instructions. The .lis file is used by the MAPCVT tool.

2. To generate an entry points file, use any editor you choose and edit this file. An example entry points file is included on the SDK CD-ROM as `ebfw/palcode/dp264/osfpal.entry`.

   The legal PAL entry points are defined in the Alpha Hardware Reference Manuals (specific to your CPU) listed in Appendix A.

   The file format is:

   ```
   offset_value(hex)     pal_entry_point_label
   ```

   Note that offset_value is the offset from the base of the executable code. For example:

   ```
   0000  PAL$RESET
   0020  PAL$MCHK
   0060  PAL$ARITH
   00e0  PAL$INTERRUPT
   ```

3. To generate a .map file, use the MAPCVT tool, as shown in the following example:

   ```
   % mapcvt -pvc filename.lis filename.map
   ```

   The -pvc option creates a .map file that consists only of PVC labels, which is the typical use. The .map file is a description of PVC symbols derived from labels in the PAL source code file.

   The format of the output .map file generated by the MAPCVT tool is:

   ```
   label    address
   ```

   For example:

   ```
   pvc$osf11$5000 00004298
   pvc$osf28$5000.1 00004430
   pvc$osf29$5000.2 000044B8
   pvc$osf0$3000 000053BC
   pvc$osf1$3000.1 000053C0
   pvc$osf2$3000.2 000053D0
   pvc$osf3$3000.3 000053E0
   pvc$osf4$3000.4 000053F0
   pvc$osf5$3000.5 00005400
   pvc$osf6$3000.6 0000540C
   pvc$osf31$84 000056F0
   ```

14.5 Labels

Labels are defined in the PALcode source file to allow you to specify additional information to PVC. Labels serve the following two functions in PVC:

- To suppress error messages, disabling a specific PALcode restriction for a specific instruction
- To specify how PVC follows a computed goto or subroutine branch

The label format is:

```
PVC<$><label_name><$><num>[.<dest>]
```

Table 14–1 describes the parts of a PVC label.

Table 14–1 PVC Label Format

Label Part	Description
PVC	Specifies that the label is a PVC label. It must appear in all uppercase or all lowercase letters.
<$>	Specifies single character delimiter. It must be a dollar sign ($).
<label_name>	Provides a unique name for the label. This field is ignored by PVC.
<num>	Specifies the label type (error, computed goto, or a subroutine branch).
<dest>	Specifies that this label is the destination of a computed goto or a subroutine branch.

All label examples in this document use a dollar sign ($) as the delimiter.

The <num> field can be used to give you more detailed information about the type of label, as shown in Table 14–2.

Table 14–2 PVC Label Type

<num> Field	Label Type
0–1007	Error
1008	No branch
2000–3999	Computed goto
4000→	Subroutine branch

For example, this label specifies a PVC label for a computed goto destination:

```
PVC$osf123$2000.1
```

14.5.1 Suppressing Error Messages for a Given Instruction

In some cases, you may decide that your PALcode can violate a PALcode restriction without harming your code. For these cases, you should use labels to shut off the normal PVC error checking by following these steps:

1. Place a label at the address of the instruction that causes the message you want to suppress.

2. Place the label with the <num> field set to the error number associated with the message.

For example, during a PVC session, the following message is reported:

```
Checking the CODE routine, entry point 0:

***

Error executing instruction HW_MFPR   R6, ICCSR at address 4 on cycle 1!!

(PVC #77) You can’t read back from the ICCSR until 3 bubbles after writing it.

***
```

You determine that, for this case, the HW_MFPR will not harm your code, so you specify the following label at address 4 in your PALcode source file:

```
PVC$123$77:
```

The `123` string between the delimiters is the label_name and is ignored by PVC. The `77` is the <num> field and specifies to PVC that, if error type 77 occurs at this label address, then the error is not displayed.

14.5.2 Handling Computed Gotos and Subroutine Branches

Another use of labels is to specify how PVC follows a computed goto or a subroutine branch. This information cannot be extracted statically; therefore, labels are required for instructions such as jump to subroutine (JSR) and return from subroutine (RET). You can also instruct PVC to ignore a certain branch to optimize your PVC run.

14.5.2.1 Computed Gotos

When creating a label for a computed goto, you need one label that designates an origin, and one or more labels that designate a destination target. All origin and target pairs must have the same integer between 2000 and 3999 in the <num> field. The <destination> field of the label is used to designate a target for the goto.

For example, in the .map file, the following is a goto origin:

```
pvc$osf0$3000 000053BC
```

The following is an example of target labels for the specified origin:

```
pvc$osf1$3000.1 000053C0
pvc$osf2$3000.2 000053D0
pvc$osf3$3000.3 000053E0
pvc$osf4$3000.4 000053F0
pvc$osf5$3000.5 00005400
pvc$osf6$3000.6 0000540C
```

In the following example, register 3 (r3) can have either of two target addresses, 10$ or 20$:

```
jsr r0, (r3)
halt
```

Target addresses and code are:

```
10$: subq r4, r5, r7
20$: subq r4, r6, r7
   ret  r31,  (r0)
```

Labels

The following are examples of the appropriate use of labels:

```
pvc$x$2000:
   jsr r0, (r3)
pvc$x$2001.1
pvc$x$2002.1:
   halt

pvc$x$2000.1:
10$: subq r4, r5, r7
pvc$x$2001:
   ret r31, (r0)

pvc$x$2000.2:
20$:  subq r4, r6, r7
pvc$x$2002:
   ret r31, (r0)
```

Note that the returns are treated just like the initial jsr subroutines.

14.5.2.2 Subroutine Branches

To specify a label for a branch to subroutine (BSR), set the <num> field value to 4000 or higher. To associate all BSRs that go to the same subroutine as well as the RET at the end of that subroutine, assign the same integer to this field. Use the <destination> field to specify a RET. For example:

```
pvc$osf11$5000 00004298
pvc$osf28$5000.1 00004430
pvc$osf29$5000.2 000044B8
```

Every time PVC finds a BSR marked this way, PVC pushes PC + 4 onto a stack. Then, when PVC hits a RET that also has a label, it checks the stack to make sure the top entry matches where it is and goes to that address. For example:

```
pvc$r$4000:
   bsr r10, subr
   bis r31,r31,r31
   bis r31,r31,r31
   bis r31,r31,r31
pvc$s$4000:
   bsr r10, subr
   halt

subr:
   mulq r1,#256,r2
pvc$t$4000.1:
   ret r31, (r10)
```

This RET goes back to the correct address both times.

14.5.2.3 Ignoring a Branch

To tell PVC not to follow a certain branch, put a label with the <num> field set to 1008 at the appropriate address. For example, if all the CALL_PAL slots jump to a routine that checks for OPCDEC, and then branch to other flows, and so on, you are repeatedly checking OPCDEC. Skipping this branch could improve execution time; however, because of the reduced checking, this feature should only be used if it dramatically improves PVC execution time.

14.6 Starting and Running PVC

After you have prepared the input files, you can begin your PVC session. For example:

```
% pvc
PALcode Violation Checker V3.34
Default Cpu set to Alpha chip 21264 family.
PVC> set code osfpal_21264.exe
PVC> set entry osfpal_21264.entry
PVC> set map osfpal_21264.map
PVC> go
PALcode Violation Checker V3.34
Default Cpu set to Alpha chip 21264 family.
Initializing Alpha dependent tables..
Initializing 21264 dependent tables..
Disassembling executable...
Searching through map file for violation exceptions...
Beginning PALcode check...
End of PALcode check...
PVC> quit
```

PVC messages, errors, and warnings are sent to stdout (in most cases the terminal screen). The following example sets up a PVC log file to collect this information:

```
PVC> set log_file filename.log
```

If the run is successful, a Run Completed message is displayed. (See Section 14.8 for other commands you can use during your PVC session.)

14.7 Creating a PVC Environment

To automatically load PVC input files when you begin your PVC session, set up the following environment variables though your .login file (if you are using Tru64 UNIX with a C shell) or the Control Panel (if you are using the Windows NT operating system):

- PVC_PAL — for the executable file
- PVC_ENTRY — for the entry points file
- PVC_MAP — for the .map file
- PVC_CPU — for the CPU type
- PVC_LOG — for the log file

For the Tru64 UNIX operating system with a C shell, the environment variable command format is as follows:

```
% setenv PVC_ENTRY ~/user_area/subdir/filename.ent
% setenv PVC_PAL ~/user_area/subdir/filename.exe
```

For the Windows NT operating system, the environment variable command format is as follows:

```
> set PVC_ENTRY=drive:\user_area\subdir\filename.ent
```

```
> set PVC_PAL=drive:\user_area\subdir\filename.exe
```

An example of the Tru64 UNIX with a C shell environment variable command format follows:

```
% setenv PVC_ENTRY ~/user/pvc/osfpal_21264.entry
% setenv PVC_PAL ~/user/pvc/osfpal_21264.exe
% setenv PVC_MAP ~/user/pvc/osfpal_21264.map
% setenv PVC_CPU 21264
% pvc
```

When you issue the PVC command, the files load automatically. For example:

```
PALcode Violation Checker V3.34

Default Cpu set to Alpha chip 21264 family.

PVC> show files

The executable file is  /disks/users4/user/pvc/osfpal_21264.exe
The map file is         /disks/users4/user/pvc/osfpal_21264.map
The entry point file is /disks/users4/user/pvc/osfpal_21264.entry
There is no log file specified.

PVC> exit
```

14.8 PVC Commands

This section describes the PALcode Violation Checker (PVC) commands. The commands are listed in alphabetical order. All PVC commands can be abbreviated to the first three characters.

14.8.1 add

The **add** command adds an entry point to the entry point list.

Format

add

_address

_name

Parameters

_address

Specifies the address.

_name

Specifies the entry point name.

Description

The **add** command allows you to add an entry point for the current PVC session. All additions are reflected with the **show entries** command. However, the entry file is not modified.

Example

```
% pvc

PALcode Violation Checker V3.34

Default Cpu set to Alpha chip 21264 family.

PVC> add
_address (in hex): 500
_name: pal$arith

PVC> show entries
#  1:    500     PAL$ARITH

PVC> exit
```

14.8.2 clear flag

The **clear flag** command clears the specified flag_type parameter.

Format

clear flag flag_type

Parameters

all

Specifies that all flags are turned off or set to zero.

cycle_count

Specifies that the cycle count is set to zero.

dead_code

Specifies that code never branched to is ignored.

errors

Specifies that errors are not reported.

memory_usage

Specifies that node and cycle usage are set to zero. This flag is not used in the 21264; however, you can use the cycle_count flag to accomplish the same effect.

permutations

Specifies that the number of code paths is not displayed.

scheduled_code

Specifies that the scheduled output is not displayed.

trace_code

Specifies that code is not displayed while checked.

warnings

Specifies that warnings are not reported.

Description

The **clear flag** command sets the specified flag_type off or sets the value to zero.

Example

```
% pvc

PALcode Violation Checker V3.34

Default Cpu set to Alpha chip 21264 family.

PVC> show flags
The warnings flag is set.
The errors flag is set.
PVC> clear flag warnings
PVC> show flags
The errors flag is set.

PVC> exit
```

14.8.3 clear log_file

The **clear log_file** command closes any open log file set for your PVC session.

Format

clear log_file

Parameters

None.

Description

The **clear log_file** command closes the log file. All messages and output are reported to stdout (the terminal screen).

Example

```
PVC> clear log_file
Log file closed.
```

14.8.4 delete

The **delete** command causes PVC to ignore the specified entry points.

Format

delete start_entry_id [- end_entry_id]

Parameters

start_entry_id - end_entry_id

Specifies a range of entry points.

Description

The **delete** command causes PVC to ignore all entry points specified at or between the specified start_entry_id and end_entry_id for the rest of the current PVC session. The remaining entry points are renumbered.

Example

```
PVC> delete 100 – 119
```

14.8.5 do

The **do** command executes a single entry point.

Format

do entry_point

Parameters

entry_point

Specifies the entry_id or the entry point name as displayed when you enter the **show entries** command.

Description

The **do** command executes a single entry point.

Example

```
% pvc

PALcode Violation Checker V3.34

Default Cpu set to Alpha chip 21264 family.

PVC> set code osfpal_21264.exe

PVC> do 600

Initializing Alpha dependent tables..
Initializing 21264 dependent tables..
Disassembling executable...
Searching through .map file for violation exceptions...

Beginning PALcode check...

End of PALcode check...

PVC> exit
```

14.8.6 exit

The **exit** command terminates a PVC session.

Format

exit

Parameters

None.

Description

The **exit** command terminates a PVC session; it has no effect on input files. The **exit** and **quit** commands have the same function.

Example

```
PVC> exit
%
```

14.8.7 go

The **go** command executes all entry points.

Format

go

Parameters

None.

Description

The **go** command allows PVC to begin checking your code. It executes all entry points. If you have created a log file, informational messages from your PVC run are sent to that file; otherwise, they display on the screen. When all entry points have been executed, you receive a message that the file has completed, and the PVC> prompt appears.

Example

```
% pvc

PALcode Violation Checker V3.34

Default Cpu set to Alpha chip 21264 family.

PVC> set code osfpal_21264.exe

PVC> set entry osfpal_21264.entry

PVC> set map osfpal_21264.map

PVC> go

Initializing Alpha dependent tables..
Initializing 21264 dependent tables..
Disassembling executable...
Searching through .map file for violation exceptions...

Beginning PALcode check...

End of PALcode check...

PVC> quit
```

14.8.8 help

The **help** command displays basic PVC command information.

Format

help

Parameters

None.

Description

The **help** command displays a list of commands implemented in the current version of PVC.

Example

```
% pvc

PALcode Violation Checker V3.34

Default Cpu set to Alpha chip 21264 family.

PVC> help
PVC is primarily used to check for Alpha PALcode violations. It can also
be used to disassemble executable code (set flag trace) and display code
as the CPU would execute it (set flag scheduled_code).
Here is a sample PVC run:
PVC> set cpu 21264
PVC> set code_file pal.exe
PVC> set entry_file pal.entry
PVC> set log_file pal_pvc.log
PVC> go
PVC> exit
For more help enter:
HELP Commands
HELP Flags
HELP Environment_variables
PVC> help commands

set cpu 21264                            Check Alpha chip 21264 family.
set cpu 21164                            Check Alpha chip 21164 family.
set cpu 21064                            Check Alpha chip 21064 family.



set code_file      pal.exe      PALcode executable.
set map_file       pal.map      PALcode map file.
set entry_file     pal.entry    PALcode entry point addresses and names.
set log_file       pal.log      Optional Log file. Use Clear log_file to
                                close.
set freq_file      pal.freq     Optional address usage count file.
go                              Check all PAL addresses in entry_file.
```

```
do n                              Check PAL entry point at address n.
exit                              Terminal PVC session.
set pal_base n                    Offset all PAL addresses by n. The
                                  default is 0.
set flag x                        Set PVC flag x, enter HELP FLAGS for a
                                  list.
Show all                          Show files, cpu type, and flags set.
```

```
PVC> help flags
```

```
No flag commands are required for a typical PVC run.

The errors and warnings flags are set by default.

set flag          all               Set all flags.
                  errors           Display restriction errors.
                  warnings         Display restriction warnings and
                                   guidelines.
                  permutations     Report number of code paths.
                  scheduled_code   Display instructions as CPU would
                                   execute them.
                  dead_code        Report code that is not reached.
                  memory_usage     Report address and cycle usage.
                  cycle_count      Report permutation cycle counts.
                  trace_code       Disassemble instructions for each
                                   permutation.

There is a clear flag command for each set flag command.
The show flags command will display flags currently set.
```

```
PVC> help env
```

```
PVC environment variables.

PVC_PAL             Executable file (pal.exe)
PVC_MAP             Map file (pal.map)
PVC_ENTRY           PALcode entry point file (pal.entry)
PVC_LOG             Log file (pal.log)
PVC_CPU             CPU type

Example command to set a variable under UNIX:

> setenv PVC_PAL ~fred/pvc/pal.exe

Example command to set a variable under Windows NT:
> set PVC_PAL = a:pal.exe

Example command to set a variable under OpenVMS:
> define PVC_PAL sys$login_device:[.pvc]pal.exe
```

14.8.9 quit

The **quit** command terminates a PVC session.

Format

quit

Parameters

None.

Description

The **quit** command terminates a PVC session; it has no effect on input files. The **quit** and **exit** commands have the same function.

Example

```
PVC> quit
%
```

14.8.10 set code_file

The **set code_file** command specifies the executable PALcode file.

Format

set code_file filename

Parameters

filename

Specifies a file name that contains machine code instructions.

Description

The **set code_file** command reads an executable PALcode file into PVC.

For the 21164 and earlier CPUs, this file is normally generated from the GAS object file and is postprocessed with the ASTRIP tool.

For the 21264, this file is generated from the HAL assembler and is postprocessed with the MAPCVT tool. See Section 8.4 for more information about MAPCVT.

Example

```
PVC> set code_file pal.exe
```

14.8.11 set cpu

The **set cpu** command determines which set of restrictions is used for the current PVC session.

Format

set cpu cpu_name

Parameters

21264

Specifies the PALcode restrictions for the Alpha 21264 microprocessor family.

21164

Specifies the PALcode restrictions for the Alpha 21164 microprocessor family.

Description

The **set cpu** command determines which set of PALcode restrictions is used for the current PVC session. This command should be set before any **go** or **do** commands are given. The default CPU is the 21264.

Example

```
PVC> set cpu 21264
```

14.8.12 set delay

The **set delay** command determines the cache latency.

Format

set delay delay_value

Parameters

delay_value

Specifies the latency for bubbles and cache misses. The default is 5; the maximum value is FFFFFFFF.

Description

The **set delay** command determines the cache latency for cache misses.

Example

```
PVC> set delay 6
Cache latency noted.
```

Note: The **set delay** command is not supported for the 21164 and 21264 CPU families. It can still be issued, but it will not be used.

14.8.13 set entry_file

The **set entry_file** command specifies the entry list file.

Format

set entry_file filename

Parameters

filename

Specifies a file name that contains a list of entry points.

Description

The **set entry_file** command reads a file containing a list of entry points into PVC.

For the 21164 and earlier CPUs, this file is normally generated from the GAS object file and is postprocessed with the ALIST tool.

For the 21264, this file is generated from the HAL assembler and is postprocessed with the MAPCVT tool. See Section 8.4 for more information about MAPCVT.

Example

```
PVC> set entry_file pal.entry
```

14.8.14 set flag

The **set flag** command sets the specified flag type.

Format

set flag flag_type

Parameters

all

Specifies that all flags are set.

cycle_count

Displays the number of CPU cycles per permutation.

dead_code

Displays code that has not been executed. This command can be used in conjunction with the **set pal_base** and **set pal_end** commands to set the boundaries for this display. Specifies code never branched to.

errors

Displays error messages. This is the default.

memory_usage

Displays node and cycle usage. This flag is not used in the 21264; however, you can display equivalent information by using the cycle_count flag.

permutations

Displays the number of code paths through the code. For example, a single if-then-else style construct gives two paths through the code or two permutations.

scheduled_code

Displays the following information per cycle: address being executed, disassembly of the code being executed, and the stalled cycles waiting for memory.

trace_code

Displays code as it is checked.

warnings

Displays warning messages. This is the default.

Description

The **set flag** command sets the specified flag_type. By default, errors and warnings are set and reported. To display flags, see the **show flag** command. To cancel a flag, see the **clear flag** command.

Example

```
$ PVC

PALcode Violation Checker V3.34
Default CPU set to Alpha chip 21264 family.

PVC> set code osfpal_21264.exe
PVC> set entry osfpal_21264.entry
PVC> do powerup

Initializing Alpha dependent tables...
Initializing 21264 dependent tables...
Disassembling executable...
Searching through map file for violation exceptions...

Beginning PALcode check...
PVC>
PVC>
PVC> set flag trace_code
PVC>
PVC> do powerup


Beginning PALcode check...

 Checking the POWERUP routine, entry point 0:

 0        SRL       R22, #62, R1

 4        BR        R31, 10

 10       BLBC      R1, 6640

 Branch not taken from 10:

 14       BR        R31, 4000

 4000     BR        R1, 4004

          .

          .

          .

 6758     BNE       R31, 675c

 Branch not taken from 6758:

 675C  HW_RET_STALL (R23)

 Permutation 2 completed normally.


 Branch not taken from 6758 to 675C:

 675C  HW_RET_STALL (R23)

 Permutation 3 completed normally.


 A total of 4 permutations were traced.

 End of PALcode check...


 Checking the POWERUP routine, entry point 0:

   NOTE:


   The PVC scheduler is a much simplified model of the 21264.
   It does partially model the Retire, Reg, Queue, map, slot, and
   ic stages, but assumes zero latency memories and caches.
   It models iq, fp, ipr, map, and register dependent stalls, but
   does not model br prediction stalls, mb stalls, or inim stalls.
   PVC can be used to check for excessive IPR or register dependency
   stalls.
```

```
======> Scheduling PAL entry address: 0 PERMUTATION (0)

Cycle:   0  Addr   0:    SRL       R22,    #62, R1  Addr   4:   BR     R31,  10
Cycle:   0  ..pipes u0, u1 not allowed
Cycle:   1  Addr   10:   BLBC      R1,     6640     Addr   14:  BR     R31,  4000
Cycle:   1  ..pipes u0, u1 not allowed
Cycle:   2  Addr   4000: BR        R1,     4004     ..register dependency (R1,)
Cycle:   2
Cycle:   3  ..Possible ebox Stall (hw_mtpr or hw_mfpr ipr dependency)
Cycle:   3  ..Possible ebox Stall (register dependency R1,)
Cycle:   3  ..Possible ebox Stall (register file write port busy)
Cycle:   4  ..Possible ebox Stall (hw_mtpr or hw_mfpr ipr dependency)
Cycle:   4  ..Possible ebox Stall (register dependency R1,)
Cycle:   4  ..Possible ebox Stall (register file write port busy)
Cycle:   5  Addr   4004: LDAH      R1,     0(R1)    ..hw_mfpr or hw_mtpr ipr
                                                    dependency
        .
        .
        .
        Permutation (0)
           cycle       count: 124
           ebox stall  count: 72
           ebox busy   count: 116
           instruction count: 98
           ebox issued count: 86
           fbox issued count: 0
              squashed count: 12

        Statistics for POWERUP routine at pal entry address: 0
         Highest cycle       count is 206 in Permutation (2)
         Highest ebox stall  count is  73 in Permutation (2)
         Highest ebox busy   count is 171 in Permutation (2)
        End of Palcode check...

        PVC> set flag cycle_count
        PVC> do powerup

        Initializing Alpha dependent tables..
        Initializing 21264 dependent tables..
        Disassembling executable....
        Searching through map file for violation exceptions...

        Beginning PALcode check...
        Permutation 0 was 124 cycles long (not counting latencies).
        Permutation 1 was 124 cycles long (not counting latencies).
        Permutation 2 was 206 cycles long (not counting latencies).
        Permutation 3 was 206 cycles long (not counting latencies).
```

```
Statistics for powerup routine at PAL entry address: 0
  Highest cycle        count is 206 in Permutation (2)
  Highest ebox stall   count is  73 in Permutation (2)
  Highest ebox busy    count is 171 in Permutation (2)
End of Palcode check...

PVC>
```

14.8.15 set freq_file

The **set freq_file** command specifies a file to contain address usage data from PVC.

Format

set freq_file filename

Parameters

filename

Specifies an output file name.

Description

The **set freq_file** command opens the specified file name to collect address usage data. Each line contains address usage information for one address in the following format:

Addr: xxx   Freq: n   inst_decode

where: *Addr: xxx* is the PALcode address.

Freq: n is the number of code paths (permutations) to this address.

inst_decode is the disassembled instruction.

Example

```
% pvc

PALcode Violation Checker V3.34

Default Cpu set to Alpha chip 21264 family.

PVC> set cpu 21264

Cpu set to Alpha chip 21264 family.

PVC> set freq_file freq.log

PVC> do 500

Initializing 21264 dependent tables..
Beginning PALcode check...
  Permutation 0 was 165 cycles long (not counting latencies).
  Permutation 1 was 148 cycles long (not counting latencies).
  Permutation 2 was 299 cycles long (not counting latencies).
  Permutation 3 was 283 cycles long (not counting latencies).
      .
      .
      .

  Permutation 190 was 276 cycles long (not counting latencies).
  Permutation 191 was 276 cycles long (not counting latencies).
  Statistics for UNNAMED routine at PAL entry address: 500
   Highest cycle        count is 299 in Permutation (2)
   Highest ebox stall   count is 117 in Permutation (2)
   Highest ebox busy    count is 232 in Permutation (40)
End of PALcode check...
```

```
       \sample output from freq.log\

Addr:   300    Freq:   1     HW_MFPR       R23, EV6_EXC_ADDR;    scbd<7:0>=0000
Addr:   304    Freq:   1     HW_MFPR       R4, EV6_VA_FORM;      scbd<7:0>=1111
Addr:   310    Freq:   1     HW_MFPR       R6, EV6_VA;           scbd<7:0>=1111
                       .
                       .
                       .
Addr:   6754   Freq:   24    HW_MTPR       R31, EV6_IC_FLUSH;    scbd<7:0>=0001
Addr:   6758   Freq:   24    BNE           R31, 675c
Addr:   675C   Freq:   48    HW_RET_STALL (R23)
```

```
              PVC> exit
```

14.8.16 set log_file

The **set log_file** command specifies a file to contain error, warning, and informational messages from PVC.

Format

set log_file filename

Parameters

filename

Specifies a file name to collect output from PVC. If not specified, this information is displayed on the terminal screen.

Description

The **set log_file** command opens the specified file name to collect message information from the PVC session.

Example

```
PVC> set log_file pal.log
```

14.8.17 set map_file

The **set map_file** command specifies the PALcode .map file.

Format

set map_file filename

Parameters

filename

Specifies a file name that contains PVC symbol values. If not specified, PVC assumes the .map file name is identical to the code_file name.

Description

The **set map_file** command reads the PALcode .map file into PVC.

For the 21164 and earlier CPUs, this file is normally generated from the GAS object file and is postprocessed with the ALIST tool. See Section 4 for more information about using the ALIST tool.

For the 21264, this file is generated from the HAL assembler and is postprocessed with the MAPCVT tool. See Section 8.4 for more information about MAPCVT.

Example

```
PVC> set map_file pal.map
```

14.8.18 set pal_base

The **set pal_base** command determines the base from which the PAL entry points are offset.

Format

set pal_base address

Parameters

address

Specifies the new PAL base address; the default is 0.

Description

The **set pal_base** command determines the base from which the PAL entry points are offset. For example, if you specify that the pal_base is 10000 and your entry file specifies that pal$arith is 42, then PVC looks 10042 bytes into the file for the code associated with pal$arith. Thus, you could use the offset to the text (the code) given by ALIST as the pal_base, rather than strip the object produced by GAS.

Example

```
% pvc

PALcode Violation Checker V3.34

Default Cpu set to Alpha chip 21264 family.

PVC> set pal_base 10000
PAL base noted. All entry points will be displaced from that offset.

PVC> show all

There is no log file specified.
The CPU is set to 21264.
The warnings flag is set.
The errors flag is set.
The PAL base is 10000.
The PAL end is FFFFFFF.

PVC> exit
```

14.8.19 set pal_end

The **set pal_end** command specifies the offset to the end of code in the executable file.

Format

set pal_end end_address

Parameters

end_address

Specifies the end of code to be checked; the default is FFFFFFF**.**

Description

The **set pal_end** command is the offset in the code file to the end of the code. This allows PVC to predetermine where it looks for dead code (code never branched to). It never looks beyond pal_end bytes into the code.

Example

```
% pvc

PALcode Violation Checker V3.34

Default Cpu set to Alpha chip 21264 family.

PVC> set pal_end f10000
PAL end noted.  PVC won’t look for dead code past that address.

PVC> show all

There is no log file specified.
The CPU is set to 21264.
The warnings flag is set.
The errors flag is set.
The PAL base is 0.
The PAL end is f10000.

PVC> exit
```

14.8.20 show

The **show** command displays the status or value, or both, of the specified show_type parameter.

Format

show show_type

Parameters

all

Displays file names for all selected files, the current CPU type, pal_base, pal_end, and any flags selected.

cpu

Displays the currently selected CPU.

entries

Displays all entry points from the entry file (.ent or .entry) last set with the **set entry_file** command. The first field on each output line is an entry_id, followed by the address and entry point name.

files

Displays all input and output files defined (such as executable, entry, map, and log files).

flags

Displays all flags previously set.

Description

The **show** command displays the status or value, or both, of the files, flags, and CPU you have selected. You can also display entry points valid for the current PVC session.

Example

```
% pvc

PALcode Violation Checker V3.34

Default Cpu set to Alpha chip 21264 family.

PVC> show all
There is no log file specified.
The CPU is set to 21264.
The warnings flag is set.
The errors flag is set.
The PAL base is 0.
The PAL end is FFFFFFF.

PVC> show cpu
The CPU is set to 21264.

PVC> set entry_file osfpal_21264.entry

PVC> show entries
#  1:     0     POWERUP
#  2:   100     DTBM_DOUBLE_3
#  3:   180     DTBM_DOUBLE_4
#  4:   200     FEN
        .
        .
        .
#142:   3F80    PAL_3F80
#143:   3FC0    PAL_3FC0

PVC> show files
The entry point file is osfpal_21264.entry
There is no log file specified.

PVC> show flags
The warnings flag is set.
The errors flag is set.

PVC> exit
```

15

RCSV

15.1 Overview

The RCSV tool takes the RCS version of an input file and generates an output file that can be used as an include file. The include file contains definitions that describe the RCS version of the input file. The RCS version is used when building the SROM code.

15.2 Command Format

The RCSV utility command format is:

```
>% rcsv [-options] [[-file_options] input_file]...[[-file_options] output_file]
```

The following table describes the options:

Option	Designation	Description
h	help	Prints information about how to use SYSGEN
v	verbose	Prints more information than usual

An example of the RCSV utility command follows:

```
% rcsv -v srom.s rcsv.h
```

16
SREC

16.1 Overview

The S-record tool (SREC) produces an input file for programming SROMs with device programmers. SREC generates Motorola S-record output from either an executable file (such as a file produced by ASTRIP), or an a.out format object file produced by GAS. The Motorola S-record file can also be loaded through the serial port of a motherboard with the Alpha Microprocessor Motherboard Debug Monitor **load** or **boot** commands.

16.2 Command Format

The SREC command format is:

```
>%  srec [-options] [input_file] [output_file]
```

The following table lists the options:

Option	Designation	Description
v	verbose	Prints more information than usual.
h	help	Prints information about how to use SREC.
a	—	Input file is a.out format (output of GAS).
i	image	Input file is image format (output of ASTRIP).
o *number*	—	Places object at specified number offset in output file.

Both the input_file and output_file elements are optional, and if none are supplied, then stdin and stdout, respectively, are used.

For example:

```
% srec -a artest.o artest.sr
% srec -i artest.exe artest.sr
```

17

SROM Packer

17.1 Overview

The SROM Packer (SROM) tool processes an executable file (such as one produced by ASTRIP) and packs the bits into an image using the SROM file format required by the CPU. The resultant image is provided in an Intel Hex file format for programming ROMs (see HEXPAD) with a device programmer.

The SROM Packer cannot be used to generate images for the Alpha 21264.

17.2 Command Format

The SROM Packer has the following command format:

```
>% srom [-options] input_file [output_file]
```

If no options are specified, the default condition is to generate an instruction cache image for the Alpha 21064 with a maximum cache size of 8KB with no SROM padding.

The following table lists the options:

Option	Designation	Description
v	verbose	Prints more information than usual.
h	help	Prints information about how to use SROM Packer.
21164PC	21164PC	Generates instruction cache image for Alpha 21164PC.
21164	21164	Generates instruction cache image for Alpha 21164.
21064 (default)	21068, 21066, and 21064	Generates instruction cache image for Alpha 21068, 21066, and 21064.

If an output file name is not specified, then the default output name on a host system that runs the Tru64 UNIX operating system is the name of the input file with an .srom extension. For the Windows NT operating system, the default extension is .srm.

For example:

```
% srom artest.o artest.srom
```

18

SYSGEN

18.1 Overview

The SYSGEN tool concatenates the parts of an image. SYSGEN arranges the specified input files into one contiguous image based on information in the file header or supplied on the command line.

SYSGEN also provides padding between the end of one input file and the next so that the output is what you expect without regard for the size of the input files.

18.2 Command Format

The SYSGEN utility command format is:

```
>% sysgen [-options] [[-file_options] input_file]...
[[-file_options] output_file]
```

The following table describes the file options:

File Option	Description
a	Specifies a.out file produced by GAS. This is the default.
c	Specifies Tru64 UNIX coff object file.
e*nnn*	Overrides or supplies entry point or base address of image. The number supplied is a hexadecimal number. This is required if there is no header information in the file.
o	Specifies output file. If not supplied, defaults to stdout.
p	Specifies the byte used for padding between images. The default is 0x00.
s	Specifies stripped format file (no header).

The following table describes the options:

Option	Designation	Description
h	help	Prints information about how to use SYSGEN
v	verbose	Prints more information than usual

Command Format

For example:

```
% sysgen -v -e8000 -s osfpal_dbm.exe -e10000 -s dbm.nh -o dbm.img
     sysgen, system builder [V3.1]
     Padding byte: (0x00)
     Files are:
     osfpal_dbm.exe: (stripped), entry = 0x00008000,0 text, 0 data
     fsb.nh: (stripped), entry = 0x00010000, 0 text, 0 data
     fsb.img: (output), entry = 0x00000000,0 text, 0 data

     00000000 00008000 00006d40    osfpal_dbm.exe pad, base/entry, size
     000012c0 00010000 0000d1b0    fsb.nh pad, base/entry, size
      --- Data sum = 0059576E  Data size = 86448 (0x151B0, 84.42 KB) ---
```

This example concatenates PALcode and Debug Monitor images, osfpal_dbm.exe and dbm.nh, into a single image dbm.img. The file options supplied with the osfpal_dbm image indicate that it is based at address 8000. The file options specified with the dbm.nh image indicate that it is based at address 10000 hexadecimal.

19
ULOAD

19.1 Overview

The ULOAD tool is used on Tru64 UNIX to download a file through the serial port of your host system to the motherboard running the Alpha Microprocessors Mini-Debugger.

19.2 Command Format

The ULOAD has the following command format:

```
>% uload input_file.ext [options]
```

The full file name and the extension must be specified for the input file. No extensions are implied.

The following table lists the options:

Option	Designation	Description
load_address	Load Address	Specifies the HEX physical address in the target memory, where the image will be loaded.
serial_port	Serial Port	Specifies the name of the serial line/port to which the remote terminal is connected.
baud_rate	Baud Rate	Specifies one of two possible baud rates that may be specified: 9600 and 19200. The default is 19200.
xb	XB	Executes the XB command after loading the image.

To load the file name pc64fsb.cmp into the motherboard's memory at address 0x300000, at 19200 baud rate, type the following command:

```
% uload pc64fsb.cmp 300000 /dev/tty01
```

The ULOAD tool will perform the necessary initialization of the Mini-Debugger, wait for the Mini-Debugger prompt (SROM), and send the file with the XM command. A timer displays how much time and how many bytes remain to be sent.

20

XLOAD

20.1 Overview

The XLOAD tool is used on Windows NT to download a file through the serial port of your host system to the motherboard running the Alpha Microprocessors Mini-Debugger.

20.2 Command Format

The XLOAD command has the following format:

```
DP264> xload input_file load_address console_line [option]
```

The full file name and the extension must be specified for the input file. No extensions are implied.

The load address is the HEX physical address in the target memory, where the image will be loaded.

The console line is the name of the serial line to which the target console is connected.

The following table explains the option:

Option	Designation	Description
fast	Fast	Execute this command at 19200 baud. (The default is 9600 baud.)

To load the file name blast.exe into the motherboard's memory at address 0x4000, at 19200 baud rate, type the following command:

```
DP264> xload blast.exe 4000 com1 fast
```

The XLOAD tool will perform the necessary initialization of the Mini-Debugger, wait for the Mini-Debugger prompt (SROM), and send the file with the XM command. A timer displays how much time and how many bytes remain to be sent.

A
Support

A.1 Customer Support

The Alpha OEM website provides the following information for customer support.

URL	Description
http://www.digital.com/alphaoem	Contains the following links: • **Developers' Area:** Development tools, code examples, driver developers' information, and technical white papers • **Motherboard Products:** Motherboard details and performance information • **Microprocessor Products:** Microprocessor details and performance information • **News:** Press releases • **Technical Information:** Motherboard firmware and drivers, hardware compatibility lists, and product documentation library • **Customer Support:** Feedback form

A.2 Alpha Documentation

The following table lists some of the available Alpha documentation. You can download Alpha documentation from the Alpha OEM World Wide Web Internet site:

http://www.digital.com/alphaoem

Click on **Technical Information**.
Then click on **Documentation Library**.

Title	Order Number
Alpha Architecture Reference Manual[1]	EY–W938E–DP
Alpha Architecture Handbook	EC–QD2KC–TE
Alpha 21164 Microprocessor Hardware Reference Manual	EC–QP99C–TE
Alpha 21164 Microprocessor Data Sheet	EC–QP98C–TE

Title	**Order Number**
Alpha 21164PC Microprocessor Hardware Reference Manual	EC–R2W0A–TE
AlphaPC 264DP Product Brief	EC–RBD0A–TE
AlphaPC 264DP User’s Manual	EC–RB0BA–TE
AlphaPC 264DP Technical Reference Manual	EC–RB0DA–TE
AlphaPC 164SX Motherboard Product Brief	EC–R57CA–TE
AlphaPC 164SX Motherboard Windows NT User’s Manual	EC–R57DB–TE
AlphaPC 164SX Motherboard DIGITAL UNIX User’s Manual	EC–R8P7B–TE
AlphaPC 164SX Motherboard Technical Reference Manual	EC–R57EB–TE
AlphaPC 164LX Motherboard Product Brief	EC–R2RZA–TE
AlphaPC 164LX Motherboard Windows NT User’s Manual	EC–R2ZQF–TE
AlphaPC 164LX Motherboard Tru64 UNIX User’s Manual	EC–R2ZPC–TE
AlphaPC 164LX Motherboard Technical Reference Manual	EC–R46WC–TE
Alpha Motherboards Software Developer’s Kit Product Brief	EC–QXQKD–TE
Alpha Motherboards Software Developer’s Kit Read Me First	EC–QERSJ–TE
Alpha Microprocessors Motherboard Debug Monitor User’s Guide	EC–QHUVG–TE
Alpha Microprocessors SROM Mini-Debugger User’s Guide	EC–QHUXD–TE

[1] Not available on website. To purchase the *Alpha Architecture Reference Manual*, contact your local sales office or call Butterworth-Heinemann (DIGITAL Press) at 1–800–366–2665.

Index