Alpha Microprocessors
Motherboard Software Design Tools
User’s Guide

Order Number: EC–QHUWD–TE

Revision/Update Information: This is a revised document. It supersedes the Alpha Microprocessors Motherboard Software Design Tools User’s Guide, EC–QHUWC–TE.
While DIGITAL believes the information included in this publication is correct as of the date of publication, it is subject to change without notice.

Digital Equipment Corporation makes no representations that the use of its products in the manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

©Digital Equipment Corporation 1997. All rights reserved.
Printed in U.S.A.

AlphaPC, DECchip, DIGITAL, DIGITAL Semiconductor, DIGITAL UNIX, OpenVMS, and the DIGITAL logo are trademarks of Digital Equipment Corporation.

DIGITAL Semiconductor is a Digital Equipment Corporation business.

Intel is a registered trademark of Intel Corporation.
Linux is a registered trademark of Croce, William R. Della, Jr.
Motorola is a registered trademark of Motorola, Inc.
UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Ltd.
VxWorks is a registered trademark of Wind River Systems, Inc.
Windows NT is a trademark of Microsoft Corporation.
Xilinx is a trademark of Xilinx Incorporated.

All other trademarks and registered trademarks are the property of their respective owners.
Contents

Preface

1 Introduction
   1.1 Overview ................................................................. 1–1
   1.2 Software Design Tools Summary .................................. 1–1

2 Installation and Setup
   2.1 Overview ................................................................. 2–1
   2.2 Tools Created or Modified .......................................... 2–1
   2.3 Sample Files ............................................................ 2–2

3 ALIST
   3.1 Overview ................................................................. 3–1
   3.2 Command Format ...................................................... 3–1

4 ASTRIP
   4.1 Overview ................................................................. 4–1
   4.2 Command Format ...................................................... 4–1

5 CLIST
   5.1 Overview ................................................................. 5–1
   5.2 Command Format ...................................................... 5–1
6 CSTRIP
   6.1 Overview .......................................................... 6–1
   6.2 Command Format .................................................. 6–1

7 GNU Assembler
   7.1 Overview .......................................................... 7–1
   7.2 Command Format .................................................. 7–1
   7.3 PALcode Assembler Instructions Added to GAS ..................... 7–2
   7.4 GAS and GLD Programming Considerations ....................... 7–5

8 HEX32
   8.1 Overview .......................................................... 8–1
   8.2 Command Format .................................................. 8–1

9 HEXPAD
   9.1 Overview .......................................................... 9–1
   9.2 Command Format .................................................. 9–1

10 HFCOMP
   10.1 Overview .......................................................... 10–1
   10.2 Command Format .................................................. 10–1

11 MAKEROM
   11.1 Overview .......................................................... 11–1
   11.2 ROM Header Information Fields .................................... 11–1
   11.3 Command Format .................................................. 11–4

12 PALcode Violation Checker
   12.1 Overview .......................................................... 12–1
   12.2 PVC Input Files .................................................... 12–1
   12.2.1 Executable PALcode Image File ................................. 12–2
   12.2.2 Entry Points File ................................................. 12–2
15  SROM Packer
   15.1  Overview ................................................................. 15–1
   15.2  Command Format ....................................................... 15–1

16  SYSGEN
   16.1  Overview ................................................................. 16–1
   16.2  Command Format ....................................................... 16–1

17  ULOAD
   17.1  Overview ................................................................. 17–1
   17.2  Command Format ....................................................... 17–1

A  Support, Products, and Documentation

Index
Figures

11–1  MAKEROM Fields. ................................................................. 11–2
12–1  PVC Tool Map. ................................................................. 12–2
### Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–1</td>
<td>Software Design Tools Summary</td>
<td>1–1</td>
</tr>
<tr>
<td>2–1</td>
<td>Tools and Supported Operating System</td>
<td>2–1</td>
</tr>
<tr>
<td>12–1</td>
<td>PVC Label Format</td>
<td>12–4</td>
</tr>
<tr>
<td>12–2</td>
<td>PVC Label Type</td>
<td>12–5</td>
</tr>
</tbody>
</table>
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:

- 21164PC
- 21164
- 21068
- 21066A
- 21064A
- 21066
- 21064

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 17 describe the tools created or modified for the Alpha Microprocessors Software Design Tools Kit.
• Appendix A contains information about technical support services and associated documentation.

Conventions

The following conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A percent sign (%)</td>
<td>Indicates the DIGITAL UNIX operating system command prompt.</td>
</tr>
<tr>
<td>A greater than sign (&gt;)</td>
<td>Indicates the Windows NT operating system command prompt.</td>
</tr>
<tr>
<td>A greater than sign and a percent sign (&gt;%)</td>
<td>Indicates that a command is supported in Windows NT and the DIGITAL UNIX operating systems.</td>
</tr>
<tr>
<td>Square brackets ([[]])</td>
<td>Denote optional syntax.</td>
</tr>
<tr>
<td><strong>Boldface type</strong></td>
<td>Indicates Debug Monitor firmware command text.</td>
</tr>
<tr>
<td><em>Italic type</em></td>
<td>Emphasizes important information, indicates variables in command syntax, and indicates complete titles of manuals.</td>
</tr>
<tr>
<td><strong>Monospaced type</strong></td>
<td>Indicates an operating system command, a file name, or directory pathname.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Tool Name</th>
<th>Purpose</th>
<th>Input</th>
<th>Output</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIST</td>
<td>Produces a listing of disassembled code plus symbolic information</td>
<td>a.out object file</td>
<td>List file (default), -e entry point file, -m PVC map file</td>
<td>-v, -h, -f</td>
</tr>
<tr>
<td>ASTRIP</td>
<td>Strips header</td>
<td>a.out object file</td>
<td>Stripped object file (executable)</td>
<td>-a, -v, -h, -n, -r</td>
</tr>
<tr>
<td>CLIST</td>
<td>Produces a listing of disassembled code plus symbolic information</td>
<td>coff format object file</td>
<td>List file (default), -e entry point file, -m PVC map file</td>
<td>-v, -h, -f</td>
</tr>
<tr>
<td>CSTRIP</td>
<td>Strips header</td>
<td>coff format file</td>
<td>Stripped object file (executable)</td>
<td>-a, -v, -h, -n, -r</td>
</tr>
<tr>
<td>GAS</td>
<td>GNU-based assembler</td>
<td>Source</td>
<td>a.out (default)</td>
<td>-P, -o, -l, -v, -21164</td>
</tr>
<tr>
<td>HEX32</td>
<td>Generates Intel Hex32 output</td>
<td>Executable file</td>
<td>Intel Hex32 file (.hex)</td>
<td>-v, -o</td>
</tr>
<tr>
<td>HEXPAD</td>
<td>Adds padding to a Hex file</td>
<td>a.out object file</td>
<td>a.out (default)</td>
<td>-v, -h, -x, -b</td>
</tr>
</tbody>
</table>
# Software Design Tools Summary

## Table 1–1 Software Design Tools Summary

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Purpose</th>
<th>Input</th>
<th>Output</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HFCOMP</strong></td>
<td>Compresses an input file</td>
<td>System ROM file</td>
<td>Compressed file</td>
<td>-v, -h, -t, -11164PC, -21164, -21066, -21064</td>
</tr>
<tr>
<td><strong>MAKEROM</strong></td>
<td>Builds a ROM image</td>
<td>ROM image files</td>
<td>-o output file</td>
<td>-l, -c, -x, -s, -f, -i, -v, -h, -r</td>
</tr>
<tr>
<td><strong>PVC</strong></td>
<td>Checks for PALcode violations</td>
<td>Executable file, entry point file, map file</td>
<td>Log</td>
<td>Not applicable</td>
</tr>
<tr>
<td><strong>RCSV</strong></td>
<td>Generates an output file that can be used as an include file</td>
<td>Source file</td>
<td>Include file</td>
<td>-h, -v</td>
</tr>
<tr>
<td><strong>SREC</strong></td>
<td>Generates S-record format code</td>
<td>-a a.out object file, -i executable file</td>
<td>Motorola S-record format (.sr)</td>
<td>-v, -h, -o</td>
</tr>
<tr>
<td><strong>SROM</strong></td>
<td>Generates SROM code</td>
<td>Executable file</td>
<td>Intel Hex format (.hex)</td>
<td>-v, -h, -11164PC, -21164, -21064</td>
</tr>
<tr>
<td><strong>SYSGEN</strong></td>
<td>Builds an image</td>
<td>-a a.out, -c coff format, -s stripped format</td>
<td>-o executable image file</td>
<td>-v, -h, -e, -p</td>
</tr>
<tr>
<td><strong>UPLOAD</strong></td>
<td>Downloads a file through the serial port</td>
<td>ROM image files</td>
<td>—</td>
<td>-load_address, -serial_port, -baud_rate, -xb</td>
</tr>
</tbody>
</table>
2
Installation and Setup

2.1 Overview

The Alpha Microprocessors Motherboard Software Design Tools are supported for Alpha microprocessor-based hardware that runs the DIGITAL UNIX or Windows NT operating system. To install the tools, see the Read Me First document.

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 *(Sheet 1 of 2)*

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Description</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIST</td>
<td>Generates a listing file from C source and its associated assembler</td>
<td>DIGITAL UNIX, Windows NT</td>
</tr>
<tr>
<td>ASTRIP</td>
<td>Strips header information from an a.out format executable file</td>
<td>DIGITAL UNIX, Windows NT</td>
</tr>
<tr>
<td>CLIST</td>
<td>Produces a listing from coff format</td>
<td>DIGITAL UNIX</td>
</tr>
<tr>
<td>CSTRIP</td>
<td>Strips header information from a coff format executable file</td>
<td>DIGITAL UNIX</td>
</tr>
<tr>
<td>GAS</td>
<td>GNU-based assembler</td>
<td>DIGITAL UNIX, Windows NT</td>
</tr>
<tr>
<td>HEX32</td>
<td>Generates Intel Hex32 output</td>
<td>DIGITAL UNIX, Windows NT</td>
</tr>
<tr>
<td>HEXPAD</td>
<td>Adds padding to a Hex file</td>
<td>DIGITAL UNIX, Windows NT</td>
</tr>
</tbody>
</table>
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.

### Table 2–1 Tools and Supported Operating System

<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Description</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>HFCOMP</td>
<td>Compresses the specified input file using a Huffman encoding algorithm</td>
<td>DIGITAL UNIX, Windows NT</td>
</tr>
<tr>
<td>MAKEROM</td>
<td>Builds a ROM image by adding header information and then concatenates the files</td>
<td>DIGITAL UNIX, Windows NT</td>
</tr>
<tr>
<td>PVC</td>
<td>Checks for PALcode violations</td>
<td>DIGITAL UNIX, Windows NT</td>
</tr>
<tr>
<td>RCSV</td>
<td>Generates an output file that can be used as an include file</td>
<td>DIGITAL UNIX, Windows NT</td>
</tr>
<tr>
<td>SREC</td>
<td>Takes an arbitrary image and converts it to Motorola S-record format</td>
<td>DIGITAL UNIX, Windows NT</td>
</tr>
<tr>
<td>SROM</td>
<td>Embeds instruction cache initialization into the executable data and generates Intel Hex format</td>
<td>DIGITAL UNIX, Windows NT</td>
</tr>
<tr>
<td>SYSGEN</td>
<td>Concatenates the specified input files into one contiguous image</td>
<td>DIGITAL UNIX, Windows NT</td>
</tr>
<tr>
<td>ULOAD</td>
<td>Downloads a file through the SROM serial port</td>
<td>DIGITAL UNIX, Windows NT</td>
</tr>
</tbody>
</table>
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:

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

The following table lists the options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>verbose</td>
<td>Gives more information than usual</td>
</tr>
<tr>
<td>h</td>
<td>help</td>
<td>Prints information about how to use ALIST</td>
</tr>
<tr>
<td>e</td>
<td>entry points</td>
<td>Produces entry point output for PVC</td>
</tr>
<tr>
<td>m</td>
<td>map</td>
<td>Outputs PVC symbols from object file</td>
</tr>
<tr>
<td>f</td>
<td>full information</td>
<td>Does not skip the zero location</td>
</tr>
</tbody>
</table>
Command Format

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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>verbose</td>
<td>Prints more information than usual.</td>
</tr>
<tr>
<td>h</td>
<td>help</td>
<td>Prints information about how to use ASTRIP.</td>
</tr>
<tr>
<td>a</td>
<td>—</td>
<td>Strips all sections, data as well as text, from the object file.</td>
</tr>
<tr>
<td>n number</td>
<td>number</td>
<td>Strips a specified number of bytes from the front of the file; a number must be supplied.</td>
</tr>
<tr>
<td>r</td>
<td>round</td>
<td>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.)</td>
</tr>
</tbody>
</table>
Command Format

If an output file name is not specified, then the default for the DIGITAL 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:

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

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

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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>verbose</td>
<td>Prints more information than usual.</td>
</tr>
<tr>
<td>h</td>
<td>help</td>
<td>Prints information about how to use CSTRIP.</td>
</tr>
<tr>
<td>a</td>
<td>—</td>
<td>Strips all sections, data as well as text, from the object file.</td>
</tr>
<tr>
<td>n number</td>
<td>number</td>
<td>Strips a specified number of bytes from the front of the file; a number must be supplied.</td>
</tr>
<tr>
<td>r</td>
<td>round</td>
<td>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.)</td>
</tr>
</tbody>
</table>

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:

- DIGITAL Semiconductor Alpha 21164PC Microprocessor Hardware Reference Manual
- DIGITAL Semiconductor Alpha 21164 Microprocessor Hardware Reference Manual
- DIGITAL Semiconductor Alpha 21064 and Alpha 21064A Microprocessors Hardware Reference Manual

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

7.2 Command Format

The basic GAS command format is:

```bash
> % gas [-options] input_file_list
```
The following table describes the options:

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

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

The following example generates an object file for PVC:

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

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

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

### 7.3 PALcode Assembler Instructions Added to GAS

The following PALcode assembler instructions have been added to GAS for the Alpha microprocessors:

- **hw_ld**

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

You can use one or more of the following options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>PHY</td>
<td>Specifies that the effective address is physical</td>
</tr>
<tr>
<td>a</td>
<td>ALT</td>
<td>Uses current mode bits in ALT_MODE IPR</td>
</tr>
<tr>
<td>r</td>
<td>RWC or WRTCK</td>
<td>Read-with-write check on virtual HW_LD instructions</td>
</tr>
</tbody>
</table>
PALcode Assembler Instructions Added to GAS

<table>
<thead>
<tr>
<th>Option</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>QW</td>
<td>Quadword data length</td>
</tr>
<tr>
<td>v</td>
<td>VPTE</td>
<td>Flags a virtual PTE fetch (21164 microprocessor family only)</td>
</tr>
<tr>
<td>l</td>
<td>LOCK</td>
<td>Load lock version of HW_LD (21164 microprocessor family only)</td>
</tr>
</tbody>
</table>

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:

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

The correct example is:

```
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.

The v and l options apply only to the Alpha 21164 microprocessor family.

- **hw_st**

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

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>PHY</td>
<td>Specifies that the effective address is physical</td>
</tr>
<tr>
<td>a</td>
<td>ALT</td>
<td>Use current mode bits in ALT_MODE IPR</td>
</tr>
<tr>
<td>q</td>
<td>QW</td>
<td>Quadword data length</td>
</tr>
<tr>
<td>c</td>
<td>COND</td>
<td>Store conditional version of HW_ST (21164 microprocessor family only)</td>
</tr>
</tbody>
</table>

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.
PALcode Assembler Instructions Added to GAS

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.

The `c` option applies only to the Alpha 21164 microprocessor family.

- `hw_mfpr`  
  
  `hw_mfpr/[options] ra,rc`

You can use one of the following options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>PAL</td>
<td>References a PAL_TEMP register</td>
</tr>
<tr>
<td>a</td>
<td>ABX</td>
<td>References a register in the Abox (load and store unit)</td>
</tr>
<tr>
<td>i</td>
<td>IBX</td>
<td>References a register in the Ibox (instruction fetch and decode unit)</td>
</tr>
</tbody>
</table>

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

The following table describes the arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>Destination</td>
</tr>
<tr>
<td>rc</td>
<td>Index into the appropriate internal processor register set, or, for the 21164 microprocessor family, an index of the desired IPR</td>
</tr>
</tbody>
</table>

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

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

- `hw_mtpr`  
  
  This instruction is similar in form to `hw_mfpr` except that it is writing.
  
  `hw_mtpr/[options] ra,rc`
GAS and GLD Programming Considerations

You can use one or more of the following options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>PAL</td>
<td>References a PAL_TEMP register</td>
</tr>
<tr>
<td>a</td>
<td>ABX</td>
<td>References an Abox register</td>
</tr>
<tr>
<td>i</td>
<td>IBX</td>
<td>References an Ibox register</td>
</tr>
</tbody>
</table>

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

The following table describes the arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>Source</td>
</tr>
<tr>
<td>rc</td>
<td>Index into the appropriate internal processor register set, or, for the 21164 microprocessor family, an index of the desired IPR</td>
</tr>
</tbody>
</table>

- **hw_rei**

  **hw_rei**

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

- **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.
GAS and GLD Programming Considerations

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 AXP Architecture Reference Manual for more details about pages and page frame numbers (PFNs).
8.1 Overview

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

8.2 Command Format

The basic HEX32 command format is:

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

The following table lists the options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>verbose</td>
<td>Prints more information than usual</td>
</tr>
<tr>
<td>o</td>
<td>offset</td>
<td>Specifies image offset</td>
</tr>
</tbody>
</table>

If input and output files are not specified, then stdin and stdout are used.
9
HEXPAD

9.1 Overview
The HEXPAD tool uses an Intel Hex file format (see SROM Packer tool) 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.

9.2 Command Format
The basic HEXPAD command format is:

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

The following table lists the options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>verbose</td>
<td>Prints more information than usual</td>
</tr>
<tr>
<td>h</td>
<td>help</td>
<td>Prints information about how to use HEXPAD</td>
</tr>
<tr>
<td>x</td>
<td>padding size</td>
<td>Specifies padded data size in a hexadecimal format</td>
</tr>
<tr>
<td>b</td>
<td>byte</td>
<td>Specifies padding byte</td>
</tr>
</tbody>
</table>
10
HFCOMP

10.1 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.

10.2 Command Format

The basic HFCOMP command format is:

```
> hfcomp [-options] input_file output_file
```
Command Format

The following table lists the options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>verbose</td>
<td>Gives more information than usual</td>
</tr>
<tr>
<td>h</td>
<td>help</td>
<td>Prints information about how to use HFCOMP</td>
</tr>
<tr>
<td>t</td>
<td>target</td>
<td>Target location where decompressed image should go</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(default = 0)</td>
</tr>
<tr>
<td>21164PC</td>
<td>21164PC code</td>
<td>Generate code for Alpha 21164PC</td>
</tr>
<tr>
<td>21164</td>
<td>21164 code</td>
<td>Generate code for Alpha 21164</td>
</tr>
<tr>
<td>21066</td>
<td>21066/68 code</td>
<td>Generate code for Alpha 21066/68</td>
</tr>
<tr>
<td>21064</td>
<td>21064 code</td>
<td>Generate code for Alpha 21064 (default)</td>
</tr>
</tbody>
</table>
11 MAKEROM

11.1 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.

11.2 ROM Header Information Fields

The ROM header information placed at the beginning of each ROM image contains the fields shown in Figure 11–1.
ROM Header Information Fields

**Figure 11–1 MAKEROM Fields**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Header Revisions Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>x00</td>
<td>all</td>
</tr>
<tr>
<td>x04</td>
<td>all</td>
</tr>
<tr>
<td>x08</td>
<td>all</td>
</tr>
<tr>
<td>x0C</td>
<td>all</td>
</tr>
<tr>
<td>x10</td>
<td>all</td>
</tr>
<tr>
<td>x14</td>
<td>all</td>
</tr>
<tr>
<td>x18</td>
<td>all</td>
</tr>
<tr>
<td>x1C</td>
<td>all</td>
</tr>
<tr>
<td>x20</td>
<td>1+</td>
</tr>
<tr>
<td>x24</td>
<td>1+</td>
</tr>
<tr>
<td>x28</td>
<td>1+</td>
</tr>
<tr>
<td>x2C</td>
<td>1+</td>
</tr>
<tr>
<td>x30</td>
<td>2+</td>
</tr>
<tr>
<td>x34</td>
<td>1+</td>
</tr>
</tbody>
</table>

- **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 0x5A5AC3C3C3C.

- **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.
ROM Header Information Fields

- **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.

<table>
<thead>
<tr>
<th>Firmware ID</th>
<th>Firmware Type (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBM</td>
<td>0</td>
<td>Alpha Motherboards Debug Monitor firmware</td>
</tr>
<tr>
<td>WNT</td>
<td>1</td>
<td>Windows NT firmware</td>
</tr>
<tr>
<td>SRM</td>
<td>2</td>
<td>Alpha System Reference Manual Console</td>
</tr>
<tr>
<td>FSB</td>
<td>6</td>
<td>Alpha Motherboards Fail-Safe Booter</td>
</tr>
<tr>
<td>Milo</td>
<td>7</td>
<td>Linux Miniloader</td>
</tr>
<tr>
<td>VxWorks</td>
<td>8</td>
<td>VxWorks Real-Time Operating System</td>
</tr>
<tr>
<td>SROM</td>
<td>10</td>
<td>Serial ROM</td>
</tr>
</tbody>
</table>
Command Format

- **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.

### 11.3 Command Format

The basic MAKEROM command format is:

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

The following table lists the options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>verbose</td>
<td>Gives more information than usual</td>
</tr>
<tr>
<td>h</td>
<td>help</td>
<td>Prints information about how to use MAKEROM</td>
</tr>
<tr>
<td>r</td>
<td>offset</td>
<td>Provides optional offset into the ROM where image is located</td>
</tr>
<tr>
<td>o</td>
<td>output file</td>
<td>Specifies output file</td>
</tr>
</tbody>
</table>

The following table lists input_file_options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>fw_id</td>
<td>Specifies firmware type_number or type_name</td>
</tr>
<tr>
<td>l</td>
<td>address</td>
<td>Specifies destination address</td>
</tr>
<tr>
<td>c</td>
<td>compress</td>
<td>Compresses this file. Default is no compression</td>
</tr>
<tr>
<td>x</td>
<td>value</td>
<td>Sets the optional firmware ID field to the specified hexadecimal value</td>
</tr>
<tr>
<td>s</td>
<td>string</td>
<td>Sets the optional firmware ID field to the specified string</td>
</tr>
<tr>
<td>f</td>
<td>file</td>
<td>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</td>
</tr>
</tbody>
</table>

The following example shows the predefined firmware types:

```bash
% 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..... 0xb8d
```
Command Format

% cat timestamp.fw
0104009504181217
Version: 1.4 950418.1217
12

PALcode Violation Checker

12.1 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.

12.2 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 these files, you need to take the PALcode source and generate an object file. To generate an object file, preprocess the PALcode source file with the C preprocessor, 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.

Figure 12–1 shows the PVC tool map.
12.2.1 Executable PALcode Image File

The executable PALcode image file contains machine code instructions. This file is normally generated from the GAS object file. ASTRIP postprocesses the GAS object file to extract the machine code instructions and strip header information. The following example generates an executable file for PVC:

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

12.2.2 Entry Points File

The entry points file contains a list of entry points that you want PVC to check. The legal PAL entry points are defined in the following documents:

- *DIGITAL Semiconductor Alpha 21164PC Microprocessor Hardware Reference Manual*
- *DIGITAL Semiconductor Alpha 21164 Microprocessor Hardware Reference Manual*
- *DIGITAL Semiconductor Alpha 21064 and Alpha 21064A Microprocessors Hardware Reference Manual*
PVC Input Files

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

This file is normally generated from a GAS object file with the ALIST tool. For example:

% alist -e filename.o > filename.ent

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

12.2.3 Description of PVC Symbols

A .map file is used to describe the special PVC symbols derived from labels in the PAL source code file. The .map file is also generated using the ALIST tool. The file name for the .map file should match the file name for the .exe 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$osf115000 00004298  
pvc$osf285000.1 00004430  
pvc$osf295000.2 000044B8  
pvc$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$3000.6 0000540C  
pvc$osf31884 000056F0  
```

### 12.3 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 12–1 describes the parts of a PVC label.

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

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 12–2.

<table>
<thead>
<tr>
<th><code>&lt;num&gt;</code> Field</th>
<th>Label Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–1007</td>
<td>Error</td>
</tr>
<tr>
<td>1008</td>
<td>No branch</td>
</tr>
<tr>
<td>2000–3999</td>
<td>Computed goto</td>
</tr>
<tr>
<td>4000 →</td>
<td>Subroutine branch</td>
</tr>
</tbody>
</table>

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

PVC$osf123$2000.1

### 12.3.1 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:

```plaintext
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.
12–6  PALcode Violation Checker

12.3.2 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.

12.3.2.1 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)
```

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)
```
Labels

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.

12.3.2.2 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
```

This RET goes back to the correct address both times.

12.3.2.3 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.
Starting and Running PVC

12.4 Starting and Running PVC

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

```
% pvc
PALcode Violation Checker V3.26
Default Cpu set to DECchip 21164 family.
PVC> set code osfpal_pc164.nh
PVC> set entry osfpal_pc164.ent
PVC> set map osfpal_pc164.map
PVC> go
Initializing Alpha dependent tables..
Initializing 21164 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 12.6 for other commands you can use during your PVC session.)

12.5 Creating a PVC Environment

To automatically load PVC input files when you begin your PVC session, set up the following environment variables through your .login file (if you are using DIGITAL 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
PVC Commands

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

```bash
% 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:

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

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

```bash
% setenv PVC_ENTRY ~/user/pvc/osfpal_pc164.ent
% setenv PVC_PAL ~/user/pvc/osfpal_pc164.nh
% setenv PVC_MAP ~/user/pvc/osfpal_pc164.map
% setenv PVC_CPU 21164
% pvc
```

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

PALcode Violation Checker V3.26
Default Cpu set to DECchip 21164 family.

PVC> show files
The executable file is /disks/users4/user/pvc/osfpal_pc164.nh
The map file is /disks/users4/user/pvc/osfpal_pc164.map
The entry point file is /disks/users4/user/pvc/osfpal_pc164.ent
There is no log file specified.

PVC> exit

12.6 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.
PVC Commands

add

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 V3.26
Default Cpu set to DECchip 21164 family.

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

PVC> show entries
# 1:  500  PAL$ARITH

PVC> exit
```
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.

ersors
Specifies that errors are not reported.

memory_usage
Specifies that node and cycle usage are set to zero.

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.
PVC Commands

clear flag

Description

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

Example

```plaintext
% pvc
PALcode Violation Checker V3.26
Default Cpu set to DEChip 21164 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
```
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.
PVC Commands
delete

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
PVC Commands

**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 V3.26
Default Cpu set to DECchip 21164 family.
PVC> set code osfpal_pc164.nh
PVC> do 600
Initializing Alpha dependent tables..
Initializing 21164 dependent tables..
Disassembling executable...
Searching through .map file for violation exceptions...
Beginning PALcode check...
Checking the UNNAMED routine, entry point 600:
(PVC #1003) Permutation 0 completed **ABNORMALLY**
          due to a HALT instruction
Address of HALT: 600
End of PALcode check...
PVC> exit
```
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
% exit
```
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```

```bash
% pvc
PALcode Violation Checker V3.26
Default Cpu set to DECchip 21164 family.
PVC> set code osfpal_pc164.nh
PVC> set entry osfpal_pc164.ent
PVC> set map osfpal_pc164.map
PVC> go
Initializing Alpha dependent tables..
Initializing 21164 dependent tables..
Disassembling executable...
Searching through .map file for violation exceptions...
Beginning PALcode check...
End of PALcode check...
PVC> quit
```
PVC Commands

help

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

```plaintext
% pvc

PALcode Violation Checker V3.26

Default Cpu set to DECchip 21164 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 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 Commands

help

PVC> help commands

set cpu 21164 Check DECchip 21164 family.
set cpu 21064 Check DECchip 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
PVC Commands

help

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
PVC Commands

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

quit

PVC>
PVC Commands
set code_file

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. This file is normally generated from the GAS object file and is postprocessed with the ASTRIP tool.

Example
PVC> set code_file arith.exe
set cpu

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

Format

```
set cpu cpu_name
```

Parameters

- **21064**
  
  Specifies the PALcode restrictions for the Alpha 21064 microprocessor. (This includes the 21064A, 21066, 21066A and 21068.)

- **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 21164.

Example

```
PVC> set cpu 21164
```
PVC Commands

set delay

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 CPU families. It can still be issued, but it will not be used.
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. This file is normally generated from the GAS object file and is postprocessed with the ALIST tool.

**Example**

```
PVC> set entry_file arith.ent
```
### PVC Commands

**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.

- **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
PVC> do pal$reset
Beginning PALcode check...

Checking the pal$reset routine, entry point 0:
(PVC #1003) Permutation 0 completed **ABNORMALLY**.
Address of HALT: 4000
End of PALcode check...

PVC> set flag trace_code
PVC> do pal$reset
Beginning PALcode check...

Checking the pal$reset routine, entry point 0:
0   HW_MTPR   R31, NOP
4   BR   R1, 4000
4000  HALT
Checking the pal$reset routine, entry point 0:
(PVC #1003) Permutation 0 completed **ABNORMALLY**.
Address of HALT: 4000
Permutation 0 completed abnormally via HALT.
A total of 1 permutations were traced
End of PALcode check...

PVC> clear flag trace
PVC> set flag scheduled
PVC> do 4
Beginning PALcode check...

Checking the UNNAMED routine, entry point 4:
(PVC #1003) Permutation 0 completed **ABNORMALLY**.
Address of HALT: 4000
Cycle:  0   Addr:  4   BR   R1, 4000
Cycle:  1   Addr: 4000   HALT
```
PVC Commands
set flag

Cycle:  0    Addr:    4 BR        R1, 4000
Cycle: These stalls simulate a cache miss
Cycle:  1    **Stall**
Cycle:  2    **Stall**
Cycle:  3    **Stall**
Cycle:  4    **Stall**
Cycle:  5    **Stall**
Cycle: These stalls simulate a branch bubble
Cycle:  6    **Stall**
Cycle:  7    Addr: 4000    HALT

End of PALcode check...

PVC> clear flag scheduled

PVC> set flag permutations

PVC> do 4
Disassembling executable...
Searching through .map file for violation exceptions...
Beginning PALcode check...
Checking the UNNAMED routine, entry point 4: (PVC #1003) Permutation 0 completed **ABNORMALLY**.
Address of HALT: 4000
There are 1 permutations to the UNNAMED entry point.
End of PALcode check...

PVC> clear flag permutations

PVC> set flag cycle_count

PVC> do 4
Disassembling executable...
Searching through .map file for violation exceptions...
Beginning PALcode check...
Checking the UNNAMED routine, entry point 4: (PVC #1003) Permutation 0 completed **ABNORMALLY**.
Address of HALT: 4000
Permutation 1 was 2 cycles long (not counting latencies).
Permutation 1 was 8 cycles long (taking latencies into account).
End of PALcode check...

PVC> clear flag cycle_count

PVC> set flag memory_usage

PVC> do 4
Beginning PALcode check...
Checking the UNNAMED routine, entry point 4: (PVC #1003) Permutation 0 completed **ABNORMALLY**.
Address of HALT: 4000
PVC Commands
set flag

Node usage: 1.
Cycle usage: 8.
End of PALcode check...
PVC Commands

set freq_file

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 V3.26
Default Cpu set to DECchip 21164 family.
PVC> set cpu 21164
Cpu set to DECchip 21164 family that first shipped in 1994.
PVC> set code osfpal_pcl64.nh
PVC> set freq_file freq.log
PVC> do 500
```
PVC Commands

set freq_file

Initializing Alpha dependent tables..
Initializing 21164 dependent tables..
Disassembling executable...
Searching through .map file for violation exceptions...
Beginning PALcode check...

PVC> exit
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 arith.log
```
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. This file is normally generated from the GAS object file and is postprocessed with the ALIST tool. See Section 12.2.3 for more information about using the ALIST tool.

**Example**

```
PVC> set map_file arith.map
```
PVC Commands

set pal_base

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 V3.26
Default Cpu set to DECchip 21164 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 21164.
The warnings flag is set.
The PAL base is 10000.
The PAL end is FFFFFFF.
PVC> exit
```
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 FFFFFFFF.

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 V3.26
Default Cpu set to DECchip 21164 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 21164.
The warnings flag is set.
The errors flag is set.
The PAL base is 0.
The PAL end is f10000.
PVC> exit
```
PVC Commands

show

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 V3.26
Default Cpu set to DECchip 21164 family.

PVC> show all
There is no log file specified.
The CPU is set to 21164.
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 21164.

PVC> set entry_file osfpal_pc164.ent

PVC> show entries
#  1:     0     PAL_RESET
#  2:    80     PAL_IACCVIO
#  3:   100     PAL_INTERRUPT

PVC> show files
The entry point file is osfpal_pc164.ent.
There is no log file specified.

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

PVC> exit
13
RCSV

13.1 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.

13.2 Command Format

The RCSV utility command format is:

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

The following table describes the options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>help</td>
<td>Prints information about how to use SYSGEN</td>
</tr>
<tr>
<td>v</td>
<td>verbose</td>
<td>Prints more information than usual</td>
</tr>
</tbody>
</table>

An example of the RCSV utility command follows:

```bash
% rcsv -v srom.s rcsv.h
```
14 SREC

14.1 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.

14.2 Command Format

The SREC command format is:

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

The following table lists the options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>verbose</td>
<td>Prints more information than usual.</td>
</tr>
<tr>
<td>h</td>
<td>help</td>
<td>Prints information about how to use SREC.</td>
</tr>
<tr>
<td>a</td>
<td>—</td>
<td>Input file is a.out format (output of GAS).</td>
</tr>
<tr>
<td>i</td>
<td>image</td>
<td>Input file is image format (output of ASTRIP).</td>
</tr>
<tr>
<td>o number</td>
<td>—</td>
<td>Places object at specified number offset in output file.</td>
</tr>
</tbody>
</table>
Command Format

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
15

SROM Packer

15.1 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.

15.2 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:

<table>
<thead>
<tr>
<th>Option</th>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>verbose</td>
<td>Prints more information than usual.</td>
</tr>
<tr>
<td>h</td>
<td>help</td>
<td>Prints information about how to use SROM Packer.</td>
</tr>
<tr>
<td>21164PC</td>
<td>21164PC</td>
<td>Generates instruction cache image for Alpha 21164PC.</td>
</tr>
<tr>
<td>21164</td>
<td>21164</td>
<td>Generates instruction cache image for Alpha 21164.</td>
</tr>
<tr>
<td>21064</td>
<td>21068, 21066, and 21064</td>
<td>Generates instruction cache image for Alpha 21068, 21066, and 21064. This is the default.</td>
</tr>
</tbody>
</table>
Command Format

If an output file name is not specified, then the default output name on a host system that runs the DIGITAL 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
16
SYSGEN

16.1 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.

16.2 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:

<table>
<thead>
<tr>
<th>File Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Specifies a.out file produced by GAS. This is the default.</td>
</tr>
<tr>
<td>c</td>
<td>Specifies DIGITAL UNIX coff object file.</td>
</tr>
<tr>
<td>ennn</td>
<td>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.</td>
</tr>
<tr>
<td>o</td>
<td>Specifies output file. If not supplied, defaults to stdout.</td>
</tr>
<tr>
<td>p</td>
<td>Specifies the byte used for padding between images. The default is 0x00.</td>
</tr>
<tr>
<td>s</td>
<td>Specifies stripped format file (no header).</td>
</tr>
</tbody>
</table>
Command Format

The following table describes the options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>help</td>
<td>Prints information about how to use SYSGEN</td>
</tr>
<tr>
<td>v</td>
<td>verbose</td>
<td>Prints more information than usual</td>
</tr>
</tbody>
</table>

For example:

```
% sygenc -v -a -e0 dbmpal -a -e10000 eb66_rom.nh -o eb66_rom.img
```

This example concatenates two images, dbmpal and eb66_rom.nh, into a single image eb66_rom.img. The file options supplied with the dbmpal image indicate that it is an a.out format file based at address 0. The file options specified with the eb66_rom.nh image indicate that it is also an a.out format file based at address 10000 hexadecimal.
17

17.1 Overview

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

17.2 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:

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

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 timing and bytes remain to be sent.
If you need technical support, a DIGITAL Semiconductor Product Catalog, or help deciding which documentation best meets your needs, visit the DIGITAL Semiconductor World Wide Web Internet site:

http://www.digital.com/semiconductor

You can also call the DIGITAL Semiconductor Information Line or the DIGITAL Semiconductor Customer Technology Center. Please use the following information lines for support.

For documentation and general information:

DIGITAL Semiconductor Information Line

United States and Canada: 1–800–332–2717
Outside North America: 1–510–490–4753
Electronic mail address: semiconductor@digital.com

For technical support:

DIGITAL Semiconductor Customer Technology Center

Phone (U.S. and international): 1–978–568–7474
Fax: 1–978–568–6698
Electronic mail address: ctc@hlo.mts.dec.com

DIGITAL Semiconductor Products

Note: The following products and order numbers might have been revised. For the latest versions, contact your local distributor.
To order Alpha microprocessors and motherboards, contact your local distributor. The following tables list some of the semiconductor products available from DIGITAL Semiconductor.

### Chips

<table>
<thead>
<tr>
<th>Chips</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIGITAL Semiconductor Alpha 21164 600 MHz Microprocessor</td>
<td>21164–MB</td>
</tr>
<tr>
<td>DIGITAL Semiconductor Alpha 21164 533 MHz Microprocessor</td>
<td>21164–P8</td>
</tr>
<tr>
<td>DIGITAL Semiconductor Alpha 21164 466 MHz Microprocessor</td>
<td>21164–IB</td>
</tr>
</tbody>
</table>

Motherboard kits include the motherboard and motherboard user’s manual.

### Motherboard Kits

<table>
<thead>
<tr>
<th>Motherboard Kits</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIGITAL Semiconductor AlphaPC 164SX Motherboard Windows NT</td>
<td>21A05–A0</td>
</tr>
<tr>
<td>DIGITAL Semiconductor AlphaPC 164SX Motherboard DIGITAL UNIX</td>
<td>21A05–A1</td>
</tr>
<tr>
<td>DIGITAL Semiconductor AlphaPC 164LX Motherboard Windows NT</td>
<td>21A04–C0</td>
</tr>
<tr>
<td>DIGITAL Semiconductor AlphaPC 164LX Motherboard DIGITAL UNIX</td>
<td>21A04–C1</td>
</tr>
<tr>
<td>DIGITAL Semiconductor AlphaPC 164 Motherboard Windows NT</td>
<td>21A04–B0</td>
</tr>
<tr>
<td>DIGITAL Semiconductor AlphaPC 164 Motherboard DIGITAL UNIX</td>
<td>21A04–B2</td>
</tr>
</tbody>
</table>

Design kits include full documentation and schematics. They do not include motherboards or related hardware.

### Design Kits

<table>
<thead>
<tr>
<th>Design Kits</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIGITAL Semiconductor AlphaPC 164 Motherboard Design Kit</td>
<td>QR–21A04–12</td>
</tr>
</tbody>
</table>
**DIGITAL Semiconductor Documentation**

The following table lists some of the available DIGITAL Semiconductor documentation.

<table>
<thead>
<tr>
<th>Title</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha Architecture Handbook²</td>
<td>EC–QD2KB–TE</td>
</tr>
<tr>
<td>DIGITAL Semiconductor Alpha 21164PC Microprocessor Hardware Reference Manual</td>
<td>EC–R2W0A–TE</td>
</tr>
<tr>
<td>DIGITAL Semiconductor AlphaPC 164SX Motherboard Product Brief</td>
<td>EC–R57CA–TE</td>
</tr>
<tr>
<td>DIGITAL Semiconductor AlphaPC 164LX Motherboard Product Brief</td>
<td>EC–R2RZA–TE</td>
</tr>
<tr>
<td>AlphaPC 164SX Motherboard Windows NT User’s Manual</td>
<td>EC–R57DA–TE</td>
</tr>
<tr>
<td>AlphaPC 164LX Motherboard Windows NT User’s Manual</td>
<td>EC–R2ZQD–TE</td>
</tr>
<tr>
<td>DIGITAL Semiconductor AlphaPC 164 Motherboard Product Brief</td>
<td>EC–QUQKC–TE</td>
</tr>
<tr>
<td>AlphaPC 164 Motherboard User’s Manual</td>
<td>EC–QPG0B–TE</td>
</tr>
<tr>
<td>DIGITAL Semiconductor AlphaPC 164 Motherboard Design Kit Read Me First</td>
<td>EC–QPFZA–TE</td>
</tr>
<tr>
<td>DIGITAL Semiconductor AlphaPC 164 Motherboard DIGITAL UNIX Product Brief</td>
<td>EC–QZT6B–TE</td>
</tr>
<tr>
<td>AlphaPC 164 Motherboard DIGITAL UNIX User’s Manual</td>
<td>EC–QZT5B–TE</td>
</tr>
<tr>
<td>DIGITAL Semiconductor Alpha Motherboards Software Developer’s Kit and Firmware Update V3.1 Product Brief</td>
<td>EC–QXQKC–TE</td>
</tr>
<tr>
<td>Alpha Motherboards Software Developer’s Kit and Firmware Update Read Me First</td>
<td>EC–QERSH–TE</td>
</tr>
<tr>
<td>Title</td>
<td>Order Number</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Alpha Microprocessors Motherboard Windows NT 3.51 and 4.0 Installation Guide</td>
<td>EC–QLUAH–TE</td>
</tr>
<tr>
<td>PALcode for Alpha Microprocessors System Design Guide</td>
<td>EC–QFGLC–TE</td>
</tr>
<tr>
<td>Alpha SRM Console for Alpha Microprocessor Motherboards User’s Guide</td>
<td>EC–QK8DF–TE</td>
</tr>
</tbody>
</table>

1 To purchase the *Alpha AXP Architecture Reference Manual*, contact your local distributor or call Butterworth-Heinemann (Digital Press) at 1-800-366-2665.

2 This handbook provides information subsequent to the *Alpha AXP Architecture Reference Manual*. 
# Index

<table>
<thead>
<tr>
<th>A</th>
<th>add command, 12–10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ALIST</td>
</tr>
<tr>
<td></td>
<td>command format, 3–1</td>
</tr>
<tr>
<td></td>
<td>entry point format, 12–3</td>
</tr>
<tr>
<td></td>
<td>input, 12–1</td>
</tr>
<tr>
<td></td>
<td>map file format, 12–3</td>
</tr>
<tr>
<td>Alpha Microprocessor Motherboard Debug Monitor, 14–1</td>
<td></td>
</tr>
<tr>
<td>Assembler instructions added to GAS, 7–2</td>
<td></td>
</tr>
<tr>
<td>Assembler programming considerations for GAS, 7–5</td>
<td></td>
</tr>
<tr>
<td>ASTRIP</td>
<td>command format, 4–1</td>
</tr>
<tr>
<td></td>
<td>default extension, 4–2</td>
</tr>
<tr>
<td></td>
<td>input, 12–1, 12–2</td>
</tr>
<tr>
<td>Audience, ix</td>
<td></td>
</tr>
</tbody>
</table>

| C | clear flag command, 12–11 |
|   | clear log file command, 12–13 |
|   | CLIST             |
|   | command format, 5–1 |
|   | Computed goto labels, 12–6 |
|   | Concatenate object files, 7–5 |
|   | Content overview, ix |

Conventions
- of document, x

| CSTRIP | command format, 6–1 |
|        | default extension, 6–1 |

| D | delete command, 12–14 |
|   | do command, 12–15 |
| Document | audience, ix |
|          | conventions, x |
|          | introduction, ix |
|          | structure of, ix |

| E | ent file for PVC, 12–2 |
|   | Entry points file, 12–2 |
|   | Error messages for PVC, 12–5 |
|   | exe file for PVC, 12–2 |
|   | Executable PALcode, 12–2 |
|   | exit command, 12–16 |
G

GAS
  .= directive, 7–6
  .align directive, 7–6
  assembler considerations, 7–5
  assembler instructions, 7–2
  command format, 7–1
GLD, 7–5
GNU assembler
  See GAS
go command, 12–17
Goto labels, 12–6

H

help command, 12–18
HEX32
  command format, 8–1
HEXPAD
  command format, 9–1
HFCOMP
  command format, 10–1
Huffman algorithm, 10–1
hw_ld, 7–2
hw_mfpr, 7–4
hwktop, 7–4
hw_rei, 7–5
hw_rei_stall, 7–5
hw_st, 7–3

I

Ignoring a branch with labels, 12–7
Installing the tools, 2–1
Introduction
  to document, ix

L

Labels for PVC, 12–4
Linker, 7–5

M

MAKEROM, 11–1
  command format, 11–4
map file, 12–3
Motorola S-record, 14–1

N

nh file for PVC, 12–2

O

Overview, 1–1

P

PALcode assembler instructions, 7–2
PALcode Violation Checker
  See PVC
PVC
  command format, 12–8
  entry points file, 12–2
  environment, 12–8
  executable files, 12–2
  input files, 12–1
  labels, 12–4
  map file, 12–3
  running of, 12–8
  tool map, 12–2
PVC commands, 12–9
  add, 12–10
  clear flag, 12–11
  clear log_file, 12–13
  delete, 12–14
  do, 12–15
  exit, 12–16
  go, 12–17
  help, 12–18
  quit, 12–21
  set code_file, 12–22
  set cpu, 12–23
  set delay, 12–24
  set entry_file, 12–25
  set flag, 12–26
  set freq_file, 12–30
  set log_file, 12–32
  set map_file, 12–33
  set pal_base, 12–34
  set pal_end, 12–35
  show, 12–36
  symbols, 12–3

PVC labels
  computed goto, 12–6
  format, 12–4
  ignoring a branch, 12–7
  subroutine branches, 12–7
  suppressing error messages, 12–5
  type, 12–5

PVC symbols, 12–3
PVC_CPU, 12–8
PVC_ENTRY, 12–8
PVC_LOG, 12–8
PVC_MAP, 12–8
PVC_PAL, 12–8

Q
  quit command, 12–21

R
  RCSV
    command format, 13–1
  ROM header, 11–1
  ROM image, 11–1

S
  set code_file command, 12–22
  set cpu command, 12–23
  set delay command, 12–24
  set entry_file command, 12–25
  set flag command, 12–26
  set freq_file command, 12–30
  set log_file command, 12–32
  set map_file command, 12–33
  set pal_base command, 12–34
  set pal_end command, 12–35
  show command, 12–36

SREC
  command format, 14–1

SROM format
  Motorola S-record, 14–1

SROM Packer
  command format, 15–1
  default extension, 15–2

Structure
  of document, ix

Subroutine branch labels, 12–7
Suppressing error messages for PVC, 12–5
Symbols for PVC, 12–3
SYSGEN
  command format, 16–1

T
  Tool installation, 2–1
  Tool list, 2–1

U
  ULOAD command format, 17–1