SDevTC Development Environment

Publication date: August 1998

Sony Computer Entertainment America
919 E. Hillsdale Blvd., 2nd floor
Foster City, CA 94404

Sony Computer Entertainment Europe
Waverley House
7-12 Noel Street
London W1V 4HH, England

Table of Contents

List of Figures

List of Tables

About This Manual

This manual is the latest release of instructional material relating to the Standard Development Tool Chain (SDevTC) for the PlayStation® as of Run-Time Library release 4.3. The purpose of this manual is to describe the SDevTC (formerly "Psy-Q") Development Environment for PlayStation software development.

Changes Since Last Release

This manual was originally written in 1995 as a guide to the first development boards for the PlayStation console. It was updated in 1996. Since that time, the hardware has changed significantly, and the software interface has also changed. Please note the following changes to this manual:

Updated Section	Description
Ch. 1: PC Installation	This chapter contains legacy instructions for the installation of discontinued hardware (the Psy-Q development board) and software (PSYBIOS.COM). You can ignore this chapter and use the following documents instead: • *DTL-H2500 Installation Manual***.** If you are installing the hardware and software for the DTL-H2500, use this manual, which is supplied with your DTL-H2500 board. A copy of the manual can be found on the Technical Reference CD in \DEVREFS\DTLH2500. • *Readme First*. If you own a DTL-H2700, refer to the "Readme First" document supplied with your DTL-H2700 board.
Ch. 2: Using CCPSX	In the section "Environmental Variables", the variable SN_PATH has been added. The installation manuals mentioned above contain the correct set of environment variables.

Related Documentation

This manual should be read in conjunction with the PlayStation *Developer Reference Series*.

Note: The Developer Support Website posts current developments regarding the Libraries and also provides notice of future documentation releases and upgrades.

Contents of Issue Diskette

File	Description
ASMPSX.EXE	R3000 assembler (SDevTC syntax)
ASPSX.EXE	R3000 assembler (implements a subset of MIPS compatible syntax, as output by ‘C’ compilers)
CCPSX.EXE	A generic control program capable of controlling the compiler also supplied with the system
DBUGPSX.EXE	R3000 Debugger
PSYLINK.EXE	SDevTC Linker
PSYLIB.EXE	SDevTC Librarian
RUN.EXE	Standalone executable/binary downloader
PSYMAKE.COM	SDevTC Make Utility
PSYBIOS.COM	SDevTC TSR BIOS extensions for PC host

Depending on the issue and version, the following files may also be included:

File	Description
MAKEFILE.MAK	Sample makefile
PSYQ.CB	Brief macros source
PSYQ.CM	Compiled Brief macros
xxxx.ICO	ICON Files, to aid installation of SDevTC under Windows

Developer Reference Series

This manual is part of the *Developer Reference Series*, a series of technical reference volumes covering all aspects of PlayStation development. The complete series is listed below:

Manual	Description
PlayStation Hardware	Describes the PlayStation hardware architecture and overviews its subsystems.
PlayStation Operating System	Describes the PlayStation operating system and related programming fundamentals.
Run-Time Library Overview	Describes the structure and purpose of the run-time libraries provided for PlayStation software development.
Run-Time Library Reference	Defines all available PlayStation run-time library functions, macros and structures.
Inline Programming Reference	Describes in-line programming using DMPSX, GTE inline macro and GTE register information.
SDevTC Development Environment	Describes the SDevTC (formerly "Psy-Q") Development Environment for PlayStation software development.
3D Graphics Tools	Describes how to use the PlayStation 3D Graphics Tools, including the animation and material editors.
Sprite Editor	Describes the Sprite Editor tool for creating sprite data and background picture components.
Sound Artist Tool	Provides installation and operation instructions for the DTL-H800 Sound Artist Board and explains how to use the Sound Artist Tool software.
File Formats	Describes all native PlayStation data formats.
Data Conversion Utilities	Describes all available PlayStation data conversion utilities, including both stand-alone and plug-in programs.
CD Emulator	Provides installation and operation instructions for the CD Emulator subsystem and related software.
CD-ROM Generator	Describes how to use the CD-ROM Generator software to write CD-R discs.
Performance Analyzer User Guide	Provides general instructions for using the Performance Analyzer software.

Performance Analyzer Technical Reference	Describes how to measure software performance and interpret the results using the Performance Analyzer.
DTL-H2000 Installation and Operation	Provides installation and operation instructions for the DTL-H2000 Development System.
DTL-H2500/2700 Installation and Operation	Provides installation and operation instructions for the DTL-H2500/H2700 Development Systems.

Typographic Conventions

Certain Typographic Conventions are used through out this manual to clarify the meaning of the text. The following conventions apply to all narrative text except for structure and function descriptions:

Convention	*Meaning*
`courier`	Indicates literal program code.
Bold	Indicates a document, chapter or section title.

The following conventions apply within structure and function descriptions only:

Convention	*Meaning*
Medium Bold	Denotes structure or function types and names.
Italic	Denotes function arguments and structure members.

Developer Support

Sony Computer Entertainment America (SCEA)

SCEA developer support is available to licensees in North America only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
In North America	*In North America*
Attn: Developer Tools Coordinator Sony Computer Entertainment America 919 East Hillsdale Blvd., 2nd floor Foster City, CA 94404 Tel: (650) 655-8000	E-mail: DevTech_Support@playstation.sony.com Web: http://www.scea.sony.com/dev Developer Support Hotline: (650) 655-8181 (Call Monday through Friday, 8 a.m. to 5 p.m., PST/PDT)

Sony Computer Entertainment Europe (SCEE)

SCEE developer support is available to licensees in Europe only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
In Europe	*In Europe*
Attn: Production Coordinator Sony Computer Entertainment Europe Waverley House 7-12 Noel Street London W1V 4HH Tel: +44 (0) 171 447 1600	E-mail: dev_support@playstation.co.uk Web: https://www-s.playstation.co.uk Developer Support Hotline: +44 (0) 171 390 1680 (Call Monday through Friday, 9 a.m. to 6 p.m., GMT or BST/BDT)

Chapter 1:
PC Installation

Overview

The SDevTC development system consists of the following physical components:

- PC board
- Security dongle
- Connecting cable
- PC driver and BIOS extensions
- SDevTC executable files

Accompanying this are:

- A ‘C’ and ‘C’++ compiler
- The associated library and header files

Installation is therefore a relatively straightforward procedure and is described in this chapter under the following headings:

- Installation check list
- Installing the PC interface
- Installing the PC software
- PSYBIOS.COM
- Checking installation

Installation Check List

Check the configuration of the SDevTC PC board and install it in the host PC (see the *Installing the PC Interface* section in this chapter).

Connect the PC to the PS-X target machine with the SCSI cable provided.

Install the software provided with this version of SDevTC and alter the host PC’s start up files (see the *Installing the PC Software* section in this chapter).

Insert the security dongle into the parallel (printer) port on the host PC.

Note: The security dongle must NOT be inserted into the connector on the SDevTC board provided or serious damage will result.

If required some simple checks may be performed to verify installation (see the *Checking Installation* section in this chapter).

Installing the PC Interface

The SDevTC PC interface board should be fitted into an empty 16-bit slot in the host PC. The host must be an IBM PC-AT or compatible, running MSDOS version 3.1 or higher.

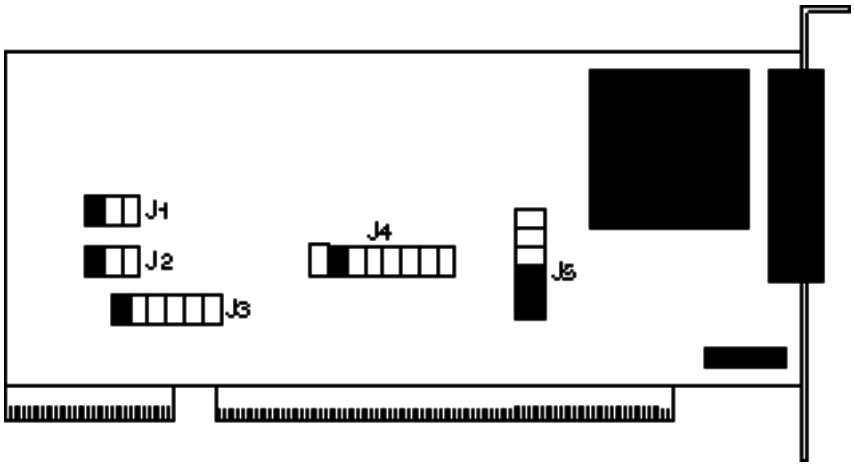
If no 16-bit slot is available, the board can be fitted into an 8-bit slot however this causes some degradation in performance.

Caution must be taken when handling the board. Touch the metal casing of the computer immediately before first touching the board and do not touch any connections on the board.

Prior to fitting, the 5 sets of jumpers on the board (see the following diagram of the SDevTC interface board) should be checked and configured as required. It is likely that the default (factory) settings will suffice. The default settings have been chosen so that the possibility of contention with other internal

boards is minimized. Nevertheless, care should be taken that settings on the SDevTC board do not conflict with any other card in the system. The meaning of the on-board jumpers is given below (the jumper names match those in the diagram).

Figure 1-1: SDevTC Interface Board



The On-Board Jumpers

J1 Select DRQ channel.

J2 Select DACK channel.

For J1 and J2, left to right jumper positions represent channels: 7, 6, 5. The factory setting is 7, although Adaptec SCSI defaults to 5. Both J1 and J2 must be set to the same channel.

J3 Selects IRQ number.

From left to right, these are: 15,12,11,10,7,5.

The factory setting is 15.

J4 Selects the base port address.

From left to right, these are: 300, 308, 310, 318, 380, 388, 390, 398 (hex).

The factory setting is 308.

J5 The bottom 3 jumpers are the SCSI ID.

The factory setting is 7.

The top three jumpers are reserved.

Installing the PC Software

The SDevTC issue diskette contains the following:

- The Assembler
- The Linker
- The Debugger
- The PC Driver
- Other target-specific BIOS extensions
- Windows accessories

The GNU ‘C’ and GNU ‘C++’ issue disks currently provided with SDevTC contain the following:

- The ‘C’ source code compiler
- The ‘C++’ source code compiler
- The GNU compiler manuals
- The Free Software Foundation public license agreement

The PSX libraries diskette contains:

- The ‘C’ libraries
- Their associated header files

n.b. C++ libraries are not currently supplied

Installing the SDevTC Development Software and the Compilers

To install the PSX system software onto your hard disk, execute the following commands.

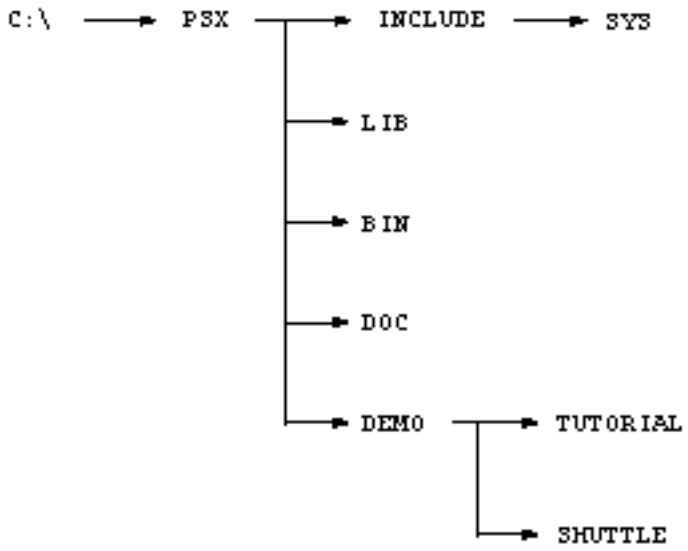
Assuming that your floppy disk drive is drive A:, and you are installing onto drive C:, your hard disk.

```
c:
cd\
a:install a: c:
```

Then follow the on-screen prompts for instructions.

This will create the following directory structure on your PC hard disk:

Figure 1-2: Directory Structure of the PSX System Software



Please adjust the PSYBIOS line added to your `autoexec.bat` if you have your SDevTC PC board configured differently (refer to manual for more details).

See `CCPSX.TXT` in the PSYQ directory for further details of environment variables.

You will then have to reboot your PC for these modifications to become effective.

You must have the security dongle plugged into the PC parallel port in order to run the SDevTC Assembler or debugger.

Note: Do not plug the security dongle into the SDevTC SCSI port. This will cause damage to the SDevTC system and your PC.

The chapter on the CCPSX control program gives limited information on using a compiler. The full documentation for use of the GNU ‘C’ compiler can be found on the GNU ‘C’ documentation disk.

If you are intending to use the 'C' compiler add the following line to your AUTOEXEC.BAT file.

```
set GO32TMP=c:/
```

This will cause the gnu compiler to store any temporary files it creates in the root of your drive C. If this is not appropriate for your machine then modify it as necessary.

If your PC does not have a floating point coprocessor then add the following line as well.

```
set GO32=emu c:/compiler/emu387
```

Running from Windows

The SDevTC Development system can readily be launched from within Windows, by performing the following:

- Create a new program group using the NEW option in the FILE menu of PROGRAM MANAGER.
- Create .PIF files, using the PIF Editor, to allow programs to run efficiently in a DOS window.
- Create new program items, again using the NEW option, for each SDevTC facility to be run under Windows.

PSYBIOS.COM

Description

PSYBIOS.COM is a TSR (Terminate and Stay Resident) program, that acts as a driver for the SDevTC interface board installed in the host PC.

Syntax

PSYBIOS *[options]*

where each option is preceded by a forward slash (/) and separated by spaces.

Options

/a Card address	Set card address: 300, 308, 310, 318, 380, 388, 390, 398.
/b Size	Specify file transfer buffer size: 2 to 32 (in kilobytes).
/d Channel	Specify DMA channel: 5,6,7;0=off.
/i Intnum	Specify IRQ number: 5,7,10,11,12,15;0=off.
/s Id	Specify SCSI: 0 to 7.
/8	Run in 8 bit slot mode.

Remarks

Normally PSYBIOS.COM is loaded in the AUTOEXEC.BAT. It can safely be loaded high to free conventional memory.

If PSYBIOS is run again with no options, the current image will be removed from memory. This is useful if you wish to change the options without rebooting the PC.

If the DMA number is not specified the BIOS will work without DMA however it will be slower.

The BIOS can drive the interface in 8-bit mode however this is the slowest mode of operation for the interface.

The buffer size option (/b) sets the size of the buffer used when the target machine accesses files on PC. A larger buffer will increase the speed of these accesses however more PC memory will be consumed.

Examples

```
PSYBIOS /a308 /d7 /i15
```

Start the driver, using card address 308, DMA channel 7, interrupt vector 15.

This command would be correct for the default jumper settings.

Checking Installation

Checking the Host-Target Connection

To check the operation of host-target connection complete the following:

1. Load the PC board driver by typing, typically:

   ```
   PSYBIOS /a308 /d7 /i15
   ```

   Note: This command will work only for the default factory board settings and should not be entered if PSYBIOS has been loaded in `AUTOEXEC.BAT`. (see the *PSBIOS.COM* section in this chapter).

2. Switch on the target console.

3. Run the program `RUN.EXE`, without parameters, to verify the link to the target adapter (see *Download Utility RUN.EXE* on page 3–6).

   If RUN correctly identifies the target, the basic setup of your SDevTC system is correct.

Checking the Operation of the Security Dongle

Note: Do not plug the security dongle into the SDevTC SCSI port.

You must have the security dongle plugged into the PC parallel port in order to run the SDevTC Assembler or debugger or the compiler. To test its operation, type the command:

```
DBUGPSX ? <return>
```

This should print a list of the valid switch. If you see the message:

```
Software Data Key not present
```

your dongle has not been installed properly. If you see the message:

```
Printer Port Loaded, or BadNetwork Communications, Check Setup
```

your particular PC is not supplying enough power to your printer port to operate the security dongle. To resolve this problem you should contact your support representative.

Chapter 2:
Using CCPSX

Overview

CCPSX is a generic control program that accepts a list of source files and option switches, then calls a C++ pre-processor, C pre-processor, C3, assembler and/or linker, as appropriate, to produce the required output. It is discussed in the following sections:

- *Command Line Syntax*
- *Environment Variables*
- *Source Files*
- *Option Switches*

For full information on the compiler supplied with your system please see the files on the compiler documentation disk.

To download and run .CPE files on the PSX target use RUN.EXE. (see *Download Utility RUN.EXE* section on page 3–6).

Command Line Syntax

The CCPSX command line consists of a sequence of source files and control switches. Control switches are prefixed with a '`-`' sign. For example:

```
ccpsx -c main.c
```

Note: The case of alphabetical switches is important. -O does not have the same effect as -o.

Long command lines can be stored in control files. By using an '`@`' sign in front of the control file name the contents of the control file can be embedded in the command line. For example:

```
ccpsx @main.cf -o main
```

This will embed the contents of `main.cf` in the command line before the `-o` option. (Makefiles are a more elegant way of organizing the whole compilation process (see *Chapter 15, The PSYMAKE Utility*).

The contents of the control file can be split across several lines without the need to use any special characters. An end of line character in a control file is treated as a space. You can specify as many control files as you want on the command line and a control file can even reference another control file.

Environmental Variables

CCPSX will search for various environment variables to specify where the files it requires can be found. These environment variables should be set up by your `AUTOEXEC.BAT` so that the required variables are always set up each time you boot your computer (see the *Installing the PC Software* section on page 1–4).

The environment variables are:

`COMPILER_PATH`	This specifies the path to the directory where the compiler executable files are stored. CCPSX will look for the files `CPPPSX.EXE`, `CC1PSX.EXE` and `CPLUSPSX.EXE` in this directory.
`C_INCLUDE_PATH`	This specifies the path to the directory where the standard 'C' and 'C++' header files are stored. This is the directory that is searched when a include statement specifies the file name in angle brackets (e.g., include <stdio.h>)
`LIBRARY_PATH`	This specifies the path to the directory where the standard 'C' and 'C++' library files are stored.

`TMPDIR`	This specifies the path to a directory where temporary files are created during compilation.
`SN_PATH`	This specifies the path to the directory where the SDevTC executable files are stored (PSYLINK.EXE, etc).

For example the environment variables could be set as follows:

```
set COMPILER_PATH=c:/compiler
set PSYQ_PATH=c:/psyq
set C_INCLUDE_PATH=c:/psx/include
set LIBRARY_PATH=c:/psx/lib
set TMPDIR=c:/
set SN_PATH=c:/pssn/bin
```

Note: The slashes in the paths may be either forwards or backwards (UNIX or DOS syntax).

Source Files

The specified source files can be either C or assembler source files or object files. CCPSX decides how to deal with a source file based on the files extension. The following table describes how each file extension is processed:

`.C`	Pass through C pre-processor, C compiler, Assembler, Linker
`.I`	Pass through C compiler, Assembler, Linker
`.CC`	Pass through C pre-processor, C++ compiler, Assembler, Linker
`.CPP`	Pass through C pre-processor, C++ compiler, Assembler, Linker
`.II`	Pass through C++ compiler, Assembler, Linker
`.IPP`	Pass through C++ compiler, Assembler, Linker
`.ASM`	Pass through C pre-processor, Assembler, Linker
`.S`	Pass through Assembler, Linker
`other`	Pass through Linker

Remarks

The DOS file system is not case sensitive and so the case of the extension has no effect.

Various command line switches can stop processing at any stage eliminating linking, assembling or compiling.

The `-x` option can be used to override the automatic selection of action based on file extension, see below for more details.

Files with an extension that is not recognized are treated as object files and passed to the linker. This obviously includes `.OBJ` files, the standard object file extension.

Several different source files, which may have different extensions, may be placed on one command line.

Option Switches

The following are the most common command line options, for further details on the compiler options see the documentation on the compiler disk.

Note that case is important. `-O` is not the same as `-o`.

Options controlling the type of output

`-E` Pre-process only. If no output file is specified then output is sent to the screen (standard output).

`-S` Compile to assembly language. If no output file is specified then `.C` files are compiled to a file with the same name but with a `.S` extension. ASM files are preprocessed as specified in the `-E` option above.

`-c` Compile to object files. C files are compiled and assembled to .OBJ files. Assembler files are just assembled. If an output file is specified then all output is sent to this file, otherwise it is sent to a file with the same name as the original source file but with a `.OBJ` extension.

`-Ipath` Specify extra include path for pre-compiler.

If none of the options listed above are used then the linker will be called. If an output file name is specified then the linkers output will go to this file. If no output is specified then the linkers output is written to a file called `A.OUT.`

Generating Debug Info

To force the C compiler to generate the information required for debugging the command line switch `-g` should be used. For example:

```
ccpsx -g main.c -o main
```

Optimization

Optimization is controlled by use of the `-O` switch. Various levels of optimization are possible:

`-O0` No optimization (default)

`-O` or `-O1` Standard level of optimization

`-O2` Full optimization

`-O3` Full optimization and function inlining

Other types of optimization may be controlled by other compiler switches. See the your compiler's documentation for more details.

Note: Compiling with `-g` and `-O` simultaneously can lead to strange effects during debugging as substantial changes to the code order and variable storage can be made during optimization.

General

`-W` Suppress compiler and pre-processor warnings.

`-DNAME`

`-DNAME=VALUE` These options are passed to the pre-processor and predefine the symbol NAME (to VALUE if specified) before processing begins.

`-UNAME` Undefines the predefined name NAME before pre-processing starts.

`-v` This option will cause ccpsx to print every command it is about to execute, before executing it.

`-f` Compiler option

`-m` Machine specific option

`-kanji` This option will cause a special stage to be run that will allow kanji 2-character sequences to be correctly dealt with by C compilers.

Linker

The linker used is the standard SDevTC linker. The following switches are relevant to the action of the linker:

(Options starting `-X` correspond to the option of the same name in the SDevTC standard linker documentation).

`-nostdlib`	This will suppress the automatic linking with the PSX standard libraries (libgs, libgte, libgpu, libetc, libapi and libsn).
`-l libname`	Include the specified library when linking.
`-Xo$xxxx`	ORGs the linker's output to address xxxx (hexadecimal). The default org address is 0
`-Xd`	When downloading the linkers output directly to the PSX, this option will prevent the program from starting to execute until the user starts it from the debugger.
`-Xb`	May be required when linking a particularly large program. This allows the program to be linked but will slow down linking if it need not have been specified.
`-o`	Specifies the destination

To send output direct to the target use. For example:

```
-o t0:
```

(The 0 in t0 is the scsi id of the target you wish to communicate to. This is normally 0 but may be any number in the range 0 to 7)

To write output to a file:

```
-o filename
```

The file produced will normally be a `.CPE` file. To produce a file which is just pure binary specify the option `-Xp` on the command line. The binary file produced will be based at the org address specified.

If you do not wish to use the default output then the output file should be specified with the `-o` option. For example:

```
ccpsx main.c -o main
```

Will compile main.c and link the program to produce a file called MAIN as the final output.

It is possible to compile several separate programs in one command by specifying all the program names. For example:

```
ccpsx -c file1.c file2.c file3.c
```

will compile `file1.c`, `file2.c` and `file3.c` to `file1.obj, file2.obj` and `file3.obj` respectively. If an output file is specified then the output from each separate compilation will overwrite this file each time and so only the final program to be compiled will produce any output.

To produce a symbol file for debugging purposes the name of the symbol file should follow the destination file name, separated by a comma (and no spaces). For example,

```
ccpsx main.c -o main.cpe,main.sym
```

will send the program to `MAIN.CPE` and symbols to `MAIN.SYM`.

To produce a map file the destination map file name should follow the symbol file name, separated from it by a comma (and no spaces). For example:

```
ccpsx main.c -o main.cpe,main.sym,main.map
```

will write a map file to `MAIN.MAP`.

Other standard SDevTC linker options can be specified by using `-X` option in the same way that `/option` would be used when calling the linker directly.

The linker will always set the entry point of the program to

```
__SN_ENTRY_POINT
```

This symbol is defined in the standard library file `LIBSN.LIB`.

Chapter 3:
The ASMPSX Assembler

Overview

The ASMPSX assembler can assemble R3000 source code at over 1 million lines per minute. Executable image or binary object code can be downloaded by the assembler itself, to run in the target machine immediately, or later, using the RUN utility.

This chapter discusses how to run an assembly session in the following sections:

- The Assembler Command Line Syntax
- Running With Brief
- Assembler and Target Errors
- The Download Utility RUN.EXE

Note: The `ASMPSX.EXE` assembler program recognizes the standard R3000 register names such as zero, at, v0, a0, s0, t0, k0, ra etc. It will also recognize register names r0, r1, r2.... r31 however, register names preceded with a '$' are not supported because this clashes with the hexadecimal number notation. The `ASPSX.EXE` assembler does support the '$' syntax for register names but does not provide SDevTC's extensive macro facilities or hexadecimal numbers. `ASPSX.EXE` is intended only for assembling code generated by a 'C' compiler.

The Assembler Command Line Syntax

During the normal development cycle ASMPSX may be:

- Run in stand alone mode.
- Launched from an editing environment, such as Brief.
- Invoked as part of the Make utility (see Chapter 15, The PSYMAKE Utility)

When run independently the Assembler command line must obey the following syntax:

ASMPSX */switchlist source,object,symbols,listings,tempdata*
or
ASMPSX *@commandfile*

If the first character on the command line following the ASMPSX command is an `@` sign, the string slash (`/`) following it signifies a SDevTC command file containing a list of Assembler commands. The components of the command line are discussed below.

Command Line Switches

The assembly is controlled by inclusion of a set of command line switches, each preceded by a forward slash (`/`). The `/o` switch introduces a string of assembler options (see *Assembler Options* on page 10–4) as can be defined in the source code using the OPT directive. The available command line switches are listed below:

`/c`	If the listing option is on, then any code not assembled because of a conditional assembly will be listed.
`/d`	Sets debug mode. If the object code is sent to the target machine, do not start it.
`/e` n=x	Assigns the value x to the symbol n.
`/g`	Non-global symbols will be output directly to the linker object file.
`/i`	Invokes the information window while assembling.
`/j` pathname	Nominates a search path for Include files.

/k	Permits the inclusion of predefined foreign conditionals, such as IFND (see *The MACROS Directive* on page 6–8).
/l	Outputs a file for the Psylink Linker.
/m	Expands all macros encountered.
/o options	Specifies Assembler options. (see *Assembler Options* on page 10–4).
/p	Outputs pure binary object code instead of an executable image in .CPE format (see *Download Utility RUN.EXE* on page 3–6).
/w	Outputs EQUATE statements to the Psylink file.
/z	Outputs line numbers to the Psylink file.
/zd	Generates source level debug information.

Further Components

Source	The file containing the source code. If an extension is not specified, the default is .S. If this parameter is omitted, the Assembler outputs help in the form of a list of switches.
Object	The file to which object code is written. If the object code is to be sent directly to the target machine specify a filename of Tn:, where n signifies the SCSI device number of the target. If this parameter is omitted object code will not be produced.
Symbols	The file to which symbol information is written, for use by the DBUGPSX Debugger.
Listings	The file to contain listings generated by assembly.
Tempdata	This parameter nominates a file to be placed on the RAM disk for faster access. If the name is omitted the default is ASM.TMP. Note that the temporary file is always deleted after assembly is complete.

Remarks

If any of the above parameters are omitted, the dividing comma must still be included on the command line unless it follows the last parameter.

The Assembler run may be prematurely terminated by pressing:

Control-C	
Control-Break	Recognized quicker because it does not require a DOS operation to spot it.
Esc	Only if the /i option has been specified to invoke the Info box. This may be advisable with some earlier versions of Brief that are erratic following Control-C or Control-Break.

The Assembler checks for an environment variable called ASMPSX. This can contain default options, switches and file specifications, in the form of a command line, including terminating commas for unspecified parameters. Defaults can be overridden in the runtime command line.

Examples

```
ASMPSX /zd /o w- scode,t0:,scode.sym
```

This command will initiate the assembly of the R3000 source code contained in a file called SCODE.S, with the following active options:

- Source level debug information to be generated.
- Warning messages to be suppressed.
- The resultant object code to be transferred directly to the target machine, SCSI device 0.
- Symbol information to be output to a file called SCODE.SYM.
- Assembly listing to be suppressed because the listing file is missing.

```
ASMPSX @game.pcf
```

This command will recognize the preceding @ sign and take its command line from a SDevTC command file called `GAME.PCF`.

Running with Brief

Most programmers prefer to develop programs completely within a single, enabling environment. Future versions of SDevTC will provide a self-contained superstructure with a built-in editor, tailored to the requirements of the assembly and debug subsystems. For the time being however it is recommended that programmers seeking such facilities should use Borland's Brief editor, which is already supported by SDevTC.

Installation in Brief

Copy the file `PSYQ.CM`, containing macros, into the `\BRIEF\MACROS` directory, or create it from source file `PSYQ.CB`.

Set the BCxxx environment variable.

Set the BFLAGS environment variable, with -mpsyq appended, to force the SDevTC macro file to be loaded on start-up.

On the next run of the Assembler or Brief, a set-up dialogue box is output, which allows defaults to be specified for output destination, options and switches.

Because of problems in some versions of Brief following a `Control-C` or `Control-Break`, it is recommended that the `/i` option is used on assembly. This will enable the Info Box, allowing `Esc` to be used for premature assembly termination.

Note: CCPSX may also be run from Brief but the appropriate macros are not presently provided.

Assembly and Target Errors

During the assembly process, errors may be generated as follows:

- By the assembler itself, as it encounters error conditions in the source code.
- By a failure during downloading of the object code.

Remarks

Appendix A gives a full list of Assembler error messages.

Errors during the download normally produce an error message, followed by an option to Retry, Bus Reset or Abort, such as:

```
Target not Available
Bus not Available

Abort, Retry or Bus Reset
```

Download Utility RUN.EXE

Description

This program downloads runable object code to the target machine.

Syntax

RUN *[switches] file name [[switches] filename] ..*

where switches are preceded by a forward slash (/).

Remarks

If RUN is executed without any runtime parameters the program will simply attempt to communicate with the target adapter hardware. If successful, RUN displays the target identification, if the attempt fails an appropriate error message is displayed.

Only CPE format files, as output by the development system, may be downloaded. Up to 8 CPE files may be specified. If no extension is specified, `.CPE` is assumed. For an executable file, execution will begin as indicated in the source code.

Multiple executable files may be specified. However, only the last executable address will apply (specified files are read from left to right).

The following switches are available:

`/h` Halt target (download but do not run)

`/t` Use target SCSI id number

`/u` Use target unit number

Example

```
RUN SOLDIER.CPE
```

Downloads the executable file `SOLDIER.CPE` to the target machine and begins execution as indicated in the original source code.

Chapter 4:
Syntax of Assembler Statements

Overview

In order to control the running of an Assembler, source code traditionally contains a number of additional statements and functions. These allow the programmer to direct the flow and operation of the Assembler as each section of code is analysed and translated into a machine-readable format. Normally, the format of Assembler statements will mirror the format of the host language, and ASMPSX follows this convention.

This chapter discusses the presentation and syntax of the ASMPSX statements as follows:

- Statement format
- Name and label format
- Assembler constants
- Assembler functions
- Assembler operators
- The RADIX directive
- The ALIAS and DISABLE directives

Statement Format

ASMPSX statements are formatted as follows:

```
Name or Label   Directive  Operand
```

The following syntactical rules apply:

- Individual fields are delimited by spaces or tabs
- Overlong lines can be split by adding an ampersand (&). The next line is then taken as a continuation
- Lines with an equals (=) sign as the first character are considered to be the options of a CASE directive (see *The CASE and ENDCASE Directives* on page 5–33).
- Comment lines:
  - Comments normally follow the operand, and start with a semi-colon.
  - Lines which consist of space or tab characters are treated as comments.
  - A complete line containing characters other than space or tabs is treated as a comment if it starts with a semi-colon or asterisk.

Note: If a character is preceded by a backslash and up arrow (\^), the corresponding control character is substituted.

Name and Label Format

Names and labels consist of standard alpha-numeric symbols, including upper-case letters, lower-case letters and numeric digits, that is in the ranges:

A to Z, a to z, 0 to 9

In addition, the following characters can occur:

`Colon (:)`	Can be used at the end of a name or label when defined, but not when referenced.
`Question Mark (?)`	
`Underscore (_)`	These three characters are often used to improve the overall readability.
`Dot (.)`	

`At sign (@)` Indicates the start of a local label (see *Chapter 8, Local Labels*). Note that by using the Assembler option `/ln`, the local label symbol can be changed to a character other than `@`.

The following usage rules apply throughout:

- Numeric digits and question marks must not be the first character of a name
- Labels normally start in column 1 however if they start elsewhere there must be no characters preceding the name, except space or tab and the last character must be a colon
- If a problem in interpretation is caused by the inclusion of a non-alphanumeric character in a name or label, that character can be replaced by a backslash or the entire name or label surrounded by brackets

Assembler Constants

The ASMPSX Assembler supports the following constant types:

Character Constants

A character string enclosed in quote marks is a character constant and is evaluated as its ASCII value. Character constants may contain up to 4 characters, to give a 32-bit value thus:

“A”	=	65
“AB”	=	(65*256)+66 = 16706
“ABC”	=	(65*65536) + (66*256)+67 = 4276803
“ABCD”	=	(65*16777216) + (66*65536) + (67*256)+68 = 1094861636

Integer Constants

Integer constants are normally evaluated as decimal, the default base, unless one of the following pertains:

- The RADIX directive changes the base.
- `$`, as the first character of an integer signifies a Hex number, `%` signifies a Binary number.
- If a character is preceded by a backslasha nd up arrow (\^), the corresponding control character is substituted.
- The AN Assembler option allows numbers to be defined as Intel and Zilog integers, that is, the number must start with a numeric character and end with either `D` for Decimal, `H` for Hexadecimal or `B` for Binary.

Special Constants

The following predefined constants are available in ASMPSX:

`_year`	As a four digit number, for example 1995
`_month`	1=January, 12 = December
`_day`	for example 1 = 1st day of month
`_weekday`	0 = Sunday, 6 = Saturday
`_hours`	00-23
`_minutes`	00-59
`_seconds`	00-59

`*`	Contains the current value of the Location Counter
`@`	Contains the actual PC value at which the current value will be stored (see example below)
`narg`	Contains the number of parameters in the current macro argument
`__rs`	Contains the current value of RS counter
`_filename`	A predefined string containing the name of the primary file undergoing assembly, that is, the file specified on the ASMPSX command line

Time and date constants are set to the start of assembly. They are not updated during the assembly process.

```
RunTime      db     “\_hours:\_minutes:&
                    \_seconds”
```

This expands to the form hh:mm:ss, as follows:

```
RunTime      db     “21:8:49”
```

Note: This example uses the special macro parameter `\`, which is described in *Chapter 6, Macros*.

The current value of the Program Counter can be used as a constant. To substitute the value of the Location Counter at the current position an asterisk (`*`) is used:

```
                          section    Bss, g_bss
Firstbss     equ          *
```

Since `*` gives the address of the start of the line,

```
                    org    $100
                    dw     *,*,*
```

defines $100 three times.

An `@`, when used on its own as a constant, substitutes the value of the location counter, pointing to an address at which the current value will be stored.

```
                    org    $100
                    dw     @,@,@
```

defines $100,$104,$108.

Assembler Functions

ASMPSX offers a large number functions to ease the programmer’s task. Several are common to other assemblers. These are listed below. In addition there is a group of specialized functions, that are described on the following pages.

`def (a)`	Returns true if a has been defined
`ref (a)`	Returns true if a has been referenced
`type (a)`	Returns the data type of a
`sqrt (a)`	Returns the square root of a
`strlen (text)`	Returns the length of string in characters
`strcmp (texta,textb)`	Returns true if strings match
`instr ([start,]txa,txb)`	Locate substring a in string b
`substr ([start],[end],string)`	Assigns the portion of string to a label
`sect (a)`	Returns the base address of section a
`offset (a)`	Returns the offset into section a
`sectoff (a)`	Equivalent to offset

`group (a)`	Returns the base address of group a
`groupoff (a)`	Returns the offset into group a

Special Functions

`filesize ("filename")`	Returns the length of the specified file or -1 if it does not exist
`groupsize (X)`	Returns the current (not final) size of group X
`grouporg (X)`	Returns the ORG address of group X or the group in which X is defined if X is a symbol or section name
`groupend (X)`	Returns the end address of group X
`sectend (X)`	Returns the end address of section X
`sectsize (X)`	Returns the current (not final) size of section X
`alignment (X)`	Gives the alignment of previously defined symbol X. This value depends upon the base alignment of the section in which X is defined as follows: Word aligned - Value in range 0-3 Halford aligned - Value in range 0-1 Byte aligned - Value always 0

Assembler Operators

The ASMPSX Assembler makes use of the following expression operators:

Table 4-1: ASMPSX Assembler Expression Operators

Symbol	Type	Usage	Action
( )	Primary	(a)	Brackets of parenthesis
+	Unary	+a	a is positive
-	Unary	-a	a is negative (see Note 1)
+	Binary	a+b	Increment a by b
-	Binary	a-b	Decrement a by b
*	Binary	a*b	Multiply a by b
/	Binary	a/b	Divide a by b, giving the quotient
%	Binary	a%b	Divide a by b, giving the modulus
<<	Binary	a<<b	Shift a to the left, b times
>>	Binary	a>>b	Shift a to the right, b times
~	Unary	~a	Logical compliment or NOT a
&	Binary	a&b	a is logically ANDed with b
^	Binary	a^b	a is exclusively ORed with b
!	Binary	a!b	a is inclusively ORed with b
|	Binary	a|b	Acts the same as !
=	Binary	a=b	a is equal to b
<>	Binary	a<>b	a is unequal to b (see Note 2)
<	Binary	a<b	a is less than b (see Note 2)
>	Binary	a>b	a is greater than b (see Note 2)
<=	Binary	a<=b	a is less than or equals b (see Note 2)
>=	Binary	a>=b	a is greater than or equals b (see Note 2)

Note1: Since the ASMPSX Assembler will evaluate 32-bit expressions, the negation bit is Bit 31. Therefore $FFFFFF and $FFFFFFF are positive hex numbers, $FFFFFFFF is a negative number.

Note2: If a comparison evaluates as true the result is returned as -1, if it evaluates as false 0 is returned.

Hierarchy of Operators

Expressions in ASMPSX are evaluated using the following precedence rules:

- Parentheses form the primary level of hierarchy and force precedence, their contents are performed first.
- Without the aid of parentheses operators are performed in the order dictated by the hierarchy table.
- Operators with similar precedence are performed in the direction of their associativity, normally from left to right, except unary operators.

Hierarchy Table

Table 4-2: Hierarchy Table

Operator	Direction	Description
( )	←	Primary
+ , -, ~	→	Unary
<<, >>	→	Shift
&, !, ^	→	Logical
*, /, %	→	Multiplicative
+, -	→	Additive
>, <, <=, >=	→	Relational
=, <>	→	Equality

The RADIX Directive

Description

The ASMPSX Assembler defaults to a base of 10 for integers. This may be changed by preceding individual numbers by the characters % or $, to change the base for that integer to binary or hex alternatively the RADIX directive can be used to change the default base.

Syntax

RADIX *newbase*

Remarks

Acceptable values for the new base are in the range of 2 to 16.

Whatever the current default, the operand of the RADIX directive is evaluated to a decimal base.

The AN assembler option (see *Chapter 10, Options, Listings, and Errors*) will not be put into effect if the default RADIX is greater than 10, since the signifiers B and D are used as digits in hexadecimal notation.

Example

```
        radix  8
```

Sets the default base to OCTAL.

The ALIAS and DISABLE Directives

Description

These directives allow the programmer to avoid a conflict between the reserved system names of constants and functions and the programmer's own symbols. Symbols can be renamed by the ALIAS directive and the original names DISABLEd, rendering them usable by the programmer.

Syntax

newname	**ALIAS**	*name*
	DISABLE	*name*

Remarks

Symbolic names currently known to the ASMPSX Assembler may be ALIASed and DISABLEd however these directives must not be used to disable ASMPSX directives.

Example

```
_Offset      alias      offset
                        disable  offset
                        ...
_Offset      dh         _Offset (Lab)
offset       dh         *-pointer
```

Chapter 5:
General Assembler Directives

Overview

The ASMPSX assembler provides a variety of functions and directives to control assembly of the source code and its layout in the target machine.

This chapter documents the Assembler directives which allow the programmer to control the processes of assembly, grouped as follows:

- Assignment directives
- Data definition
- Controlling program execution
- Controlling assembly
- Target-related directives

Assignment Directives

The directives in this section are used to assign a value to a symbolic name. The value may be a constant, variable or string.

EQU
SET (and =)
EQUS
EQUR
Rsize
RSSET
RSRESET

The EQU Directive

Description

Assigns the result of the expression, as a constant value, to the preceding symbolic name.

Syntax

[symbol name] **EQU** *expression*

Remarks

The ASMPSX Assembler allows the assigned expression to contain forward references. If an EQU cannot be evaluated as it is currently defined, the expression will be saved and substituted in any future references to the equate (see NOTE below).

It is possible to include an equate at assembly time, on the Assembler command line. This is useful for specifying major options of conditional assembly, such as test mode (see *The Assembler Command Line Syntax* section on page 3–3).

Assigning a value to a symbol with EQU is absolute; an attempt at secondary assignment will produce an error. However, it is permissible to reassign the current value to an existing symbol. Typically this occurs when subsidiary code redefines constants already used by the master segment.

Example

```
Length     equ      4
Width      equ      8
Depth      equ      12
Volume     equ      Length*Width*Depth
DmaHigh    equ      $ffff8609
DmaMid     equ      DmaHigh+2
```

Note: `List  equ  Lastentry-Firstentry`

if Firstentry and Lastentry have not yet been defined, then

```
dw     List+2
```

will be treated as

```
dw     (Lastentry-Firstentry)+2
```

the equated expression is implicitly bracketed.

See also: SET and EQUS.

The SET Directive

Description

Assigns the result of the expression, as a variable, to the preceding symbolic name.

Syntax

[symbol name]	**SET**	*expression*
[symbol name]	**=**	*expression*

Remarks

SET and equals (=) are interchangeable

Values assigned by a SET directive may be reassigned at any time.

The ASMPSX Assembler does not allow the assigned expression in a SET directive to contain forward references. If a SET cannot be evaluated as it is currently defined an error is generated.

If the symbol itself is used before it is defined, ASMPSX generates a warning and assigns it the value determined by the preliminary pass of the Assembler.

The symbol in a SET directive does not assume the type of the operand, it is therefore better suited to setting local values, such as in macros, rather than in code with a relative start position, such as a SECTION construct, which may cause an error (see examples).

Examples

```
Loopcount    set    0
GrandTotal   =      SubTotalA+SubTotalB
xdim         set    Bsize<<SC
```

The following example will encrypt the string passed as the macro parameter (see

The MACRO, ENDM and MEXIT Directives on page 6–6):

```
cbb     macro       string
lc      =           0
        rept        strlen (\string)
cc      substr      lc+1,lc+1,\string   ;extract one character into label cc
        db          '\cc'^ ($A5+lc)     ;encrypt the character stored in cc
                                        and define in memory
lc      =           lc+1                ;increment counter
        endr                            ;do for all chars in string
        endm
```

See also: EQU

The EQUS Directive

Description

Assigns a text string or a string variable to a symbol.

Syntax

[symbol name] **EQUS** *“text”*
[symbol name] **EQUS** *“text”*
[symbol name] **EQUS** *symbol name*

Remarks

Textual operands are delimited by double or single quotes. If it is required to include a double quote in the text string, delimit with single quotes or two double quotes. Similarly, to include a single quote in the text, delimit with double quotes or two single quotes (see examples below).

If delimiters are omitted, the Assembler assumes the operand to be the symbol name of a previously defined string variable, the value of which is assigned to the new symbol name.

Point brackets, `{` and `}`, are special delimiters used in Macros (see Chapter 6, *Macros*).

Symbols equated with the EQUS directive can appear at any point in the code, including as part of another text string. If there is the possibility of confusion with the surrounding text, a backslash (\) may be used before and, if necessary, after the symbol name to ensure the expression is expanded correctly (see below).

Examples

To include single quotes in a string delimited by single quotes, either change the delimiters to double quotes, or double-up the internal single quote. Similarly, this syntax applies to double quotes as follows:

```
Sinquote    equs   ‘What’s the point?’
Sinquot2    equs   “What’s the point?”

Doubquote   equs   “Say “”Hello”” and go”
Doubquote   equs   ‘Say “Hello” and go’
```

```
Program     equs   “SDevTC v 1.2”
Qtex        equs   “What’s the score?”
            db     “Remember to assemble & \_filename”,0
```

```
z           equs   “123”
            ...
            dw     z+4
```

converts to

```
            dw     123+4
```

whereas the following expression needs backslashes to be expanded correctly:

```
            dw     number\z\a
```

converts to

```
                   dw     number123a
```

```
SA                 equs   ‘StartAddress’
                   ...
                   dw     \SA\4
```

converts to

```
dw     StartAddress4
```

See also: EQU and SET

The EQUR Directive

Description

Defines a symbol as an alternative for a register.

Syntax

[symbol name] **EQUR** *register name*

Remarks

The major use of the EQUR directive is to improve the overall readability of the source code.

In order that the Assembler can evaluate the expression correctly, dots are not allowed as part of the symbol name within EQUR.

Example

```
                lw      t1,RGBinds (a2)
```

This could be rewritten using EQURs, as follows:

```
          Red    equr   t1
          Green  equr   a2
                 ...
                 lw     Red,RGBinds (Green)
```

The R*size* Directive

Description

Assigns the value of the __RS variable to the symbol, and advances the rs counter by the number of bytes, half words or words, specified in count.

Syntax

[symbol name]	**R***size*	*count*
where *size* is:	b	Byte (8 bits)
	h	Word (16 bits)
	w	Long word (32 bits)

Remarks

This directive, together with the following two associated directives, operate on or with the ASMPSX variable __RS, which contains the current offset.

Example

```
                rsreset
Icon_no         rb      1
Dropcode        rh      1
Actcode         rh      1
Actname         rb      10
Objpos          rw      1
Artlen          rb      0
```

After each of the first five Rsize equates, the __RS pointer is advanced. The values for each equate are as follows:

```
Icon_no         0       (set to zero by RSRESET)
Dropcode        1
Actcode         4       (Automatic Even set, advances the pointer to even
                         boundary)
Actname         6
Objpos          16
Artlen          20
```

The last rb does not advance the __RS pointer since a count of zero is equivalent to an EQUATE to the __RS variable.

See also: RSSET and RSRESET

The RSSET Directive

Description

Assigns the specified value to the __RS variable.

Syntax

RSSET *value*

Remarks

This directive is normally used when the offsets are to start at a value other than zero.

See also: R*size* and RSRESET

The RSRESET Directive

Description

Sets the __RS variable to zero.

Syntax

RSRESET *[value]*

Remarks

Using this directive is the normal way to initialize the __RS counter at the start of a new data structure.

The optional parameter is provided for compatibility with other assemblers. If present, RSRESET behaves like the RSSET directive.

Examples

(See the R*size* directive example.)

See also: R*size* and RSSET

Data Definition

The directives in this section are used to define data and reserve space.

D*size*
DC*size*
DS*size*
HEX
DATASIZE and DATA
IEEE32 and IEEE64

The D*size* Directive

Description

This directive evaluates the expressions in the operand field, and assigns the results to the preceding symbol, in the format specified by the *size* parameter. Argument expressions may be numeric values, strings or symbols.

Syntax

symbol name	**D***size*	expression,..,expression
where *size* is:	b	Byte (8 bits)
	h	Half word (16 bits)
	w	Word (32 bits)

Remarks

Textual operands are delimited by double or single quotes. If it is required to include a double quote in the text string, delimit with single quotes or two double quotes, similarly, to include a single quote in the text, delimit with double quotes or two single quotes (see examples below). If delimiters are omitted, the Assembler assumes the operand to be the symbol name of a previously defined string variable, the value of which is assigned to the new symbol name.

If the Assembler encounters a parameter that is out-of-range, ASMPSX flags an error. The following statements will produce errors:

```
db     257
db     -129
dh     66000
dh     -33000
```

Examples

```
Hexvals      dh     $80d,$a08,0,$80d,0
Coords       dh     -15,46
Pointers     dw     StartMarker,EndMarker
ErrorMes     db     “File Error”,0
```

See also: DC*size*

The DC*size* Directive

Description

This directive generates a block of memory, of the specified length, containing the specified value throughout.

Syntax

	DC*size*	*length, value*
where *size* is:	b	Byte (8 bits)
	h	Half word (16 bits)
	w	Word (32 bits)

Examples

```
        dcb     256,$7F
```

Generates 256 bytes containing $7F.

```
        dch     64,$FF
```

Generates 64 half words containing $FF.

See also: D*size*

The DS*size* Directive

Description

Allocates memory to the symbol, of the specified length, and initializes it to zero.

Syntax

[symbol name]	**DS***size*	length
where *size* is:	b	Byte (8 bits)
	h	Half word (16 bits)
	w	Word (32 bits)

Remarks

If this directive is used to allocate memory in a Group/Section with the BSS attribute, the reserved area will not be initialized (see *Chapter 9, Structuring the Program*).

Examples

```
        List    dsh     64
```

Reserves an area 64 half words long, and sets it to zero.

```
        Buffer  dsb     1024
```

Reserves a 1k byte area, and sets it to zero.

See also: D*size* and DC*size*

The HEX Directive

Description

This directive takes a list of unsigned hex nibble pairs as an argument, which are concatenated to give bytes. It is intended as a quick way of inputting small hex expressions.

Syntax

[symbol name] **HEX** *hexlist*

Remarks

Data stored as HEX is difficult to read, less memory-efficient and causes more work for the Assembler therefore it is suggested that the HEX statement is only used for comparatively minor data definitions. To load larger quantities of data, it is recommended that the data is stored in a file to be INCLUDEd as a binary file at runtime.

Example

```
HexString    hex    100204FF0128
```

is another way of writing

```
HexString    db     $10,$02,$04,$FF,$01,$28
```

See also: INCBIN

The DATASIZE and DATA Directives

Description

Together, these directives allow the programmer to define values between 1 and 256 bytes long (8 to 2048 bits). The size of the DATA items must first be defined by a DATASIZE directive.

Syntax

DATASIZE *size*
DATA *value[, value...]*

where value is a numeric string, in hex or decimal, optionally preceded by a minus sign.

Remarks

If a value specified in the DATA directive converts to a value greater than can be held in size specified by DATASIZE, the ASMPSX assembler flags an error.

Example

```
        datasize    8
        ...
        data        $123456789ABCDEF0
        data        -1, $FFFFFFFFFFFF
```

See also: IEEE32, IEEE64

The IEEE32 and IEEE64 Directives

Description

These directives allow 32- and 64-bit floating point numbers to be defined in IEEE format.

Syntax

IEEE32 *fp_value*
IEEE64 *fp_value*

Examples

```
ieee32      1.23,34e10;defining two floating point numbers
ieee64      123456.7654321e-2
```

See also: DATA and DATASIZE

Controlling Program Execution

The directives in this section are used to alter the state of the program counter and control the execution of the Assembler.

ORG
CNOP
OBJ and OBJEND

The ORG Directive

Description

The ORG directive informs the Assembler of the location of the code in the target machine.

Syntax

ORG *address*

where address is a previously-defined symbol, or a hex or decimal value, optionally preceded by a question mark (?).

Remarks

If a link file is output the ORG directive must not be used (see *Chapter 9, Structuring the Program*).

If the program contains SECTIONs, a single ORG is allowed, and it must precede all SECTION directives. If the program does not utilize the SECTION construct it may contain multiple ORGs.

The ORG operand can be preceded by a question mark to indicates the amount of RAM required by the program however the ORG ? function only works on machines with operating systems to allocate the memory; for instance, it will work on the Amiga but not the Sega Mega Drive.

Example

```
                org     $100
Begin           lw      r2,8 (a0)

Program         equ     $4000
                ...
                org     Program
```

See also: OBJ, OBJEND, GROUP and SECTION

The CNOP Directive

Description

Resets the program counter to a specified offset from the specified size boundary.

Syntax

CNOP *offset,size boundary*

Remarks

As in code containing SECTIONs, the ASMPSX Assembler does not allow the program counter to be reset to a size boundary greater than the alignment already set for that section. Therefore, a CNOP statement, with a size boundary of 2, is not allowed in a section that is byte-aligned.

Example

```
                section  prime
Firstoff        =        512
Firstsize       =        2
                ...
                cnop     Firstoff,Firstsize
```

Sets the program counter to 512 bytes above the next half word boundary.

The OBJ and OBJEND Directives

Description

OBJ forces the code following it to be assembled as if it were at the specified address, although it will still appear following on from the previous code. OBJEND terminates this process and returns to the ORG'd address value.

Syntax

OBJ *address*
...
OBJEND

Remarks

The OBJ-OBJEND construct is useful for code that must be assembled at one address (for instance in a ROM cartridge) but will be run at a different address, after being copied there.

Code blocks delimited by OBJ-OBJEND cannot be nested.

Example

```
org    $100
dw     *
dw     *

obj    $200
dw     *
dw     *
objend

dw     *
dw     *
```

The above code will generate the following sequence of words, starting at address $100:

```
address   Value
$100      $100
$104      $104
$108      $200
$10c      $204
$110      $110
$114      $114
```

See also: ORG

Include Files

The source code for most non-trivial programs is too large to be handled as a single file. It is normal for a program to be constructed of subsidiary files, which are called together during the assembly process. The directives in this section are used to collect together the separate source files and control their usage also discussed are operators to aid the control of code to be assembled from INCLUDEd files.

INCLUDE
INCBIN
REF
DEF

The INCLUDE Directive

Description

Informs the ASMPSX assembler to draw in and process another source file before resuming the processing of the current file.

Syntax

INCLUDE *filename*

where filename is the name of the source file to be processed, including drive and path identifiers (see Note below). The filename may be surrounded by quotes but they will be ignored.

Remarks

Traditionally there will be one main file of source code which contains INCLUDEs for all the other files.

INCLUDEd files can be nested.

The /j switch can be used to specify a search path for INCLUDEd files (see *The Assembler Command Line Syntax* section on page 3–3).

Example

A typical start to a program would be:

```
                section     short1
codestart       j           entrypoint

                db          _hours,_minutes
                db          _day,_month
                dh          _year

                include     vars1.s

                section     short2

                include     vars2.s

                section     code

                include     graph1.s
                include     graph2.s
                include     maths.s
                include     trees.s
                include     tactics.s

entrypoint      li          t0,8
                            lw    a0,fred
                            ...
```

Note: Since a path name contains backslashes, the text in the operand of an INCLUDE statement may be confused with the usage of text previously defined by an EQUS directive. To avoid this, a second backslash may be used or the backslash may be replaced by a forward slash (solidus).

Thus,

```
include      d:\source\levels.s
```

may be rewritten as

```
include      d:\\source\\levels.s
```

or

```
include      d:/source/levels.s
```

See also: INCBIN

The INCBIN Directive

Description

Informs the ASMPSX Assembler to draw in and process binary data held in another source file before resuming the processing of the current file.

Syntax

symbol **INCBIN** *filename[,start,length]*

filename is the name of the source file to be processed, including drive and path identifiers (see Note1). Optionally, the filename may be surrounded by quotes which will be ignored. *start* and *length* are optional values, allowing selected portions of the specified file to be included (see Note2).

Remarks

This directive allows quantities of binary data to be maintained in a separate file and pulled into the main program at assembly time. Typically, such data might be character movement strings or location co-ordinates. The Assembler is passed no information concerning the type and layout of the incoming data therefore labeling and modifying the INCBINed data are the responsibility of the programmer.

The /j switch can be used to specify a search path for INCLUDEd files (see *The Assembler Command Line Syntax* section on page 3–3).

Example

```
Charmove    incbin   "d:\source\charmov.bin"
```

Note1: Since a path name contains backslashes, the text in the operand of an INCBIN statement may be confused with the usage of text previously defined by an EQUS directive. To avoid this a second backslash may be used or the backslash may be replaced by a forward slash (solidus).

Thus,

```
Charmove    incbin   d:\source\charmov.bin
```

may be rewritten as

```
Charmove    incbin   d:\\source\\charmov.bin
```

or

```
Charmove    incbin   d:/source/charmov.bin
```

Note2: The nominated file may be accessed selectively, by specifying a position in the file from which to start reading and a length. Note that:

- If start is omitted, the INCBIN commences at the beginning of the file.
- If the length is omitted, the INCBIN continues to the end of the file.
- If both start and length are omitted, the entire file is INCBINed.

See also: INCLUDE and HEX

The REF Directive

Description

REF is a special operator to allow the programmer to determine which segments of code are to be INCLUDEd.

Syntax

[~]REF *(symbol)*

The optional preceding tilde [~] is synonymous with NOT.

Remarks

REF is *true* if a reference has previously been encountered for the *symbol* in the brackets.

Example

```
        if      ref (Links)
Links   lw      r1,8 (a0)
        ...
        jr      ra
        endif
```

The Links routine will be assembled if a reference to it has already been encountered.

The DEF Directive

Description

Like the REF operator, DEF is a special function. It allows the programmer to determine which segments of code have already been INCLUDEd.

Syntax

[~]DEF *(symbol)*

The optional preceding tilde [~] is synonymous with NOT.

Remarks

DEF is true if the symbol in the brackets has previously been defined.

Example

```
                if      ~def (load_addr)
load_addr       equ     $1000
exec_addr       equ     $1000
reloc_addr      equ     $80000-$300
                endif
```

The address equates will be assembled if load_addr has not already been defined.

Controlling Assembly

The following directives give instructions to the ASMPSX assembler during the assembly process. They allow the programmer to select and repeat sections of code:

END
IF, ELSE, ELSEIF, ENDIF and ENDC
CASE and ENDCASE
REPT and ENDR
WHILE and ENDW
DO and UNTIL

The END Directive

Description

The END directive informs the Assembler to cease its assembly of the source code.

Syntax

END *[address]*

Remarks

The inclusion of this directive is mostly cosmetic since the Assembler will cease processing when the input source code is exhausted.

The optional parameter specifies an initial address for the program (see also *The REGS Directive* section on page 5–37).

Example

```
startrel    lw      t0, (a0)
            addu    r3,r2,r1
            ...
            jr      ra
            end
```

See also: REGS

The IF, ELSE, ELSEIF, ENDIF and ENDC Directives

Description

These conditional directives allow the programmer to select the code for assembly.

Syntax

IF *[~]expression*
ELSE
ELSEIF *[~]expression*
ENDIF
ENDC

Remarks

The ENDC and ENDIF directives are interchangeable.

If the ELSEIF directive is used without a following expression, it acts the same as an ELSE directive.

The optional tilde [~] preceding the operand expression is synonymous with NOT. Its use normally necessitates the prudent use of brackets to preserve the sense of the expression.

Examples

```
                if          Sony1
sec_dir         equ         2
                elseif      Target
sec_dir         equ         1
                else
sec_dir         equ         3
                endif
_____________________________________
                if          ~usesquare
round           macro
                addu        \1,\2,\3
                endm
                elseif
round           macro
                endm
                endc
_____________________________________
ldimm           macro       dest,imm
                if          (\imm>-32768)& (\imm<32768)
                addiu       \dest,r0,\imm
                else
                            lui\dest, (\imm)>>16
                if          (\imm)&$ffff
                ori         \dest,\dest, (\imm)&$ffff
                endif
                endif
                endm
```

See also: CASE

The CASE and ENDCASE Directives

Description

The CASE directive is used to select code in a multiple-choice situation. The CASE argument defines the expression to be evaluated. If the argument (s) after the equals sign are true, the code that follows is assembled. The equals-question mark case is selected if no previous case is true.

Syntax

CASE *expression*
=expression[,expression]
=?
ENDCASE

Remarks

In the absence of an equals-question mark (=?) case, if the existing cases are unsuccessful the case-defined code is not assembled.

Example

The following example is similar to the first example listed under the IF directive:

```
Target      equ         Sony1
            ...
            case        Target
=Sony1
sec_dir     equ         2

=Sony2
sec_dir     equ         1

=?          db          “New Version”,0
sec_dir     equ         3

            endcase
```

See also: IF conditionals

The REPT and ENDR Directives

Description

These directives allow the programmer to repeat the code between the REPT and ENDR statements. The number of repetitions is determined by the value of count.

Syntax

REPT *count*
...
ENDR

Remarks

When used in a macro, REPT is frequently associated with the NARG function.

Examples

```
        rept            12
        dh              0,0,0,0
        endr
_____________________________________
cbb     macro           string
lc      =               0
        rept            strlen (\string)
cc      substr          lc+1,lc+1,\string
        db              "\cc"^ ($A5+lc)
lc      =               lc+1
        endr
        endm
```

See also: DO and WHILE

The WHILE and ENDW Directives

Description

These directives allow the programmer to repeat the code between the WHILE and ENDW statements, as long as the expression in the operand holds true.

Syntax

WHILE *expression*
...
ENDW

Remarks

Currently, any string equate substitutions in the WHILE expression take place once only, when the WHILE loop is first encountered (see NOTE below for the ramifications of this).

Example

```
MultP     equ                 16
          ...
Indic     =                   MultP
          while               Indic>1
          lw                  r1,term (a0)
          ...
Indic     =                   Indic-1
          endw
```

Note: Because string equates are only evaluated at the start of the WHILE loop, the following will not work.

```
s         equs                “x”
          while               strlen (“\s”) < 4
          db                  “\s”,0
s         equs                “\s\x”
          endw
```

To avoid this, set a variable each time round the loop to indicate that looping should continue.

```
s         equs                “x”
looping   =                   -1
          while               looping
          db                  “\s”,0
s         equs                “\s\x”
looping   =                   strlen (“\s”) < 4
          endw
```

See also: REPT and DO

The DO and UNTIL Directives

Description

These directives allow the programmer to repeat the code between the DO and UNTIL statements, until the specified expression becomes true.

Syntax

DO
...
UNTIL *expression*

Remarks

Unlike the WHILE directive, string equates in an UNTIL expression will be reevaluated each time round the loop.

Example

```
MultP     equ         16
          ...
Indic     =           MultP
          do
          lw          r1,term (a0)
          ...
Indic     =           Indic-1
          until       Indic<=1
```

See also: REPT and WHILE

Target-Related Directive

The following directive allows the programmer to specify certain initial parameters in the target machine:

REGS

The REGS Directive

Description

If a CPE file is produced, or object code output is directed to the target, the REGS directive specifies the contents of the registers, at the start of code execution.

Syntax

REGS regcode=expression[,regcode=expression]

where regcode is the mnemonic name of a register, such as r1,pc

Remarks

For relocatable code, which is specific to the target or pure binary code, this directive is not available.

Examples

```
        regs   t0=$2700
        regs   pc=entrypoint
```

In these examples, the register assigns could have been declared on one line, separated by commas.

Chapter 6:
Macros

Overview

The ASMPSX assembler provides the programmer with extensive macro facilities. Macros allow the programmer to assign names to complete code sequences. They may then be used in the main program like existing assembler directives.

This chapter discusses the following topics, directives and functions:

- Macro parameters
- MACRO, ENDM and MEXIT
- SHIFT and NARG
- MACROS
- PUSHP and POPP
- PURGE
- TYPE

Macro Parameters

Parameters

Macro parameters obey the following rules:

The parameters listed on the macro invocation line may appear at any point in the code declared between the MACRO and ENDM statements. Each parameter is introduced by a backslash (\). Where this may be confused with text from an EQUS, a backslash may also follow the parameter.

Up to thirty two different parameters are allowed, numbered \0 to \31. \0 is a special parameter which gives the contents of the field following a '.' character with the macro name when it was invoked, that is, the text after the point symbol (.). This includes not only .b, .h or .w, but also any text. For example:

```
zed         macro
            \0
            endm
            ...
            zed.nop
```

will generate a NOP instruction.

Instead of the \0 to \31 format, parameters can be given symbolic names by their inclusion as operands to the MACRO directive. The preceding backslash (\) is not mandatory however if there is the possibility of confusion with the surrounding text, a backslash may be used before the symbol name and if necessary, after it, to ensure the expression is expanded correctly. For example:

```
Position    macro       A,B,C,Pos,Time
            dh          \Time* (\A*\Pos+\B*\Pos+\C*\Pos)
            endm
```

Surrounding the operand of an invoked macro with greater than and less than signs (<...>), allows the use of comma and space characters. For example:

```
Credits     macro
            dh          \1,\2
            db          \3
            db          0
            even
            endm
            ...
```

Continuation Lines: when invoking a macro, it is possible that the parameter list will become overlong. As with any directive statement, to improve readability the line can be terminated by an ampersand (&) and continued on the next line. For example:

```
chstr         macro
              rept          narg
              db            k_\1
              shift
              endr
              db            0
              even
              endm
              ...
cheatstring   chstr         i,c,a,n,b,a,r,e,l,y,&
                            s,t,a,n,d,i,t
```

This example will generate a labeled list of define bytes:

```
              db     k_i
              db     k_c
              ...
              db     k_i
              db     k_t
              db     0
```

Special Parameters

There are a number of special parameter formats available in macros:

Converting integers to text: The parameters \ and \$ replace the decimal ( ) or hex ($) value of the symbol following them with its character representation. This technique is commonly used to access Run Date and Time. For example:

```
              org           $1006
RunTime       db            "\_hours:\_minutes:&
                            \_seconds"
```

This expands to the form hh:mm:ss, as follows

```
RunTime       db            "21:8:49"
```

Generating Unique Labels

The parameter \@ can be used as the last characters of a label name in a macro. When the macro is invoked this will be expanded to an underscore followed by a decimal number, this number is increased on each subsequent invocation to give a unique label. For example:

```
Slots         macro
              add           r1,r0,r0
              lw            r1,\1
              beq           r1,r0,dun\@
next\@        addiu         f3,r0,1
              bgt           r3,r0,next\@
dun\@
              \endm
              ...
              Slots         freeob1,numslot1
```

Each time the Slots macro is used new labels in the form next_001 and dun_001 will be generated.

Entire Parameter

If the special parameter _ (backslash underscore) is encountered in a macro it is expanded to the complete argument specified on the macro invocation statement. For example:

```
All             macro
                db              \_
                endm
                ...
                All             1,2,3,4

will generate
                db              1,2,3,4
```

Control Characters

The parameter `\^x`, where x denotes a control character, will generate the specified control character.

Using the Macro Label

The label heading the invocation line can be used in the macro, by specifying the first name in the symbol list of the MACRO directive to be an asterisk (*), and substituting `\*` for the label itself. However, the resultant label is not defined at the current program location therefore the label remains undefined unless the programmer gives it a value.

Extended Parameters

The ASMPSX Assembler accepts a set of elements, enclosed in curly brackets (`{ }`), to be passed to a macro parameter. The NARG function and SHIFT directive can then be used to handle the list (see page 6-7. For example:

```
cmd             macro
cc              equs            {\1}
                rept            narg (cc)
                \cc
                shift           cc
                endr
                endm
```

The line

```
cmd             <nop,jr ra, nop>
```

for instance, will generate the following code

```
nop
jr              ra
nop
```

The MACRO, ENDM and MEXIT Directives

Description

A macro consists of the source lines and parameter place markers between the MACRO directive and the ENDM. The label field is the symbolic name by which the macro is invoked. The operand allows the entry of a string of parameter data names. When the assembler encounters a directive consisting of the label and optional parameters, the source lines are pulled into the main program and expanded by substituting the place markers with the invocation parameters. The expansion of the macro is stopped immediately if the assembler encounters a MEXIT directive.

Syntax

Label **MACRO** *[symbol,..symbol]*
...
MEXIT
...
ENDM

Remarks

The invocation parameter string effectively starts at the character after the macro name, that is, the dot (`.`) character. Text strings, as well as `.b`, `.w` and `.l` are permissible parameters (see previous section).

Control structures within macros must be complete. Structures started in the macro must finish before the ENDM. Similarly, a structure started externally must not be terminated within the macro. To imitate a simple control structure from another assembler a short macro might be used (see *The MACROS Directive* section on page 6–8).

Examples

```
remove      macro
            dh          -2,0,0
            endm
```

The following example uses the parameter functions (see previous section)

```
Form        macro
            if          strcmp (‘\1’,’0’)
            dh          0
            else
            dh          \1-FormBase
            endif
            endm
```

See also: MACROS

The SHIFT Directive and NARG

Description

These directives cater for a macro having a variable parameter list as its operand. The NARG symbol is the number of arguments on the macro invocation line; the SHIFT directive shifts all the arguments one place to the left, losing the leftmost argument.

Syntax

directive **NARG**
...
SHIFT
where NARG is a reserved, predefined symbol.

Example

```
routes      macro
            rept            narg
            if              strcmp (‘\1’,’0’)
            dh              0
            else
            dh \1-routebase
            endif
            shift
            endr
            endm
            ...
            routes          0,gosouth_1
```

This example goes through the list of parameters given to the macro and define a half word of $0000 if the argument is zero or a 16 bit offset into the 'routebase' table of the given label.

See also: Extended parameters

The MACROS Directive

Description

The MACROS directive allows the entry of a single line of code as a macro, with no associated ENDM directive. The single line of code can be a control structure directive.

Syntax

Label **MACROS**
[symbol,..symbol]

Remarks

The MACROS directive may be used to stand in for a single, complex, code line. Often the short macro allows the programmer to synthesize a directive from another assembler. Including the `/k` option on the ASMPSX command line will cause several macros which emulate foreign directives to be generated.

Example

```
a           =           0      ;a=0 do explosion
                                ;a=1 do offset calculation
            if          0
boom        macros
            jal         explos\1
            else
boom        macros
            lw          r1,blowup-tactbase (\1)
            endif
```

See also: MACRO

The PUSHIP and POPP Directives

Description

These directives allow text to be pushed into, and then popped from, a string variable.

Syntax

PUSHP *string*
POPP *string*

Remarks

There is no requirement for the PUSHP and corresponding POPP directives to appear in the same macro.

Example

```
makeframe       macro       stksize
                pushp       "\stksize"
                sub         sp,\stksize
                endm
freeframe       macro
                popp        stksize
                add         sp,\stksize
                endm
```

This means the user does not have to specify stksize when using freeframe. The user must ensure that calls to makeframe and freeframe are balanced.

The PURGE Directive

Description

The PURGE directive removes an expanded macro from the internal tables and releases the memory it occupied.

Syntax

PURGE *macroname*

Remarks

If it is required to redefine a macro it is not necessary to purge it first. If an existing macro is redefined, the original macro is purged by the Assembler.

Example

```
HugeM           macro
                dh          \1
                dh          \2
                ...
                dh          \31
                endm

                HugeM       para1,103,faultlevel,&
                ...
40,50,para31

                purge       HugeM
```

The TYPE Function

Description

TYPE is a function to provide information about a symbol. It is frequently used with a macro to determine the nature of its parameters. The value is returned as a word; the meanings of the bit settings are given below.

Syntax

TYPE *(symbol)*

The reply word can be interpreted as follows:

Bit 0 *Symbol has an absolute value*
Bit 1 *Symbol is relative to the start of the section*
Bit 2 *Symbol was defined using* SET
Bit 3 *Symbol is a macro*
Bit 4 *Symbol is a string equate* (EQUS)
Bit 5 *Symbol was defined using* EQU
Bit 6 *Symbol appeared in an* XREF *statement*
Bit 7 *Symbol appeared in an* XDEF *statement*
Bit 8 *Symbol is a function*
Bit 9 *Symbol is a group name*
Bit 10 *Symbol is a macro parameter*
Bit 11 *Symbol is a short macro* (MACROS)
Bit 12 *Symbol is a section name*
Bit 14 *Symbol is a register equate* (EQUR)

Chapter 7:
String Manipulation Functions

Overview

To enhance the Macro structure, the ASMPSX Assembler includes powerful functions for string manipulation. These enable the programmer to compare strings, examine text and prepare subsets.

This chapter covers the following string handling functions and directives:

- STRLEN
- STRCMP
- INSTR
- SUBSTR

The STRLEN Function

Description

A function which returns the length of the text specified in the brackets.

Syntax

STRLEN *(string)*

Remarks

The STRLEN function is available at any point in the operand.

Example

```
Nummov      macro
            rept        strlen (\1)
            lw          r3, (a0)
            add         a0,4
            sw          r3, (a1)
            add         a1,4
            endr
            endm
            `...
            Nummov      “12345”
```

The number of characters in the string is used as the extent of the loop.

See also: STRCMP

The STRCMP Function

Description

A function which compares two text strings in the brackets and returns true if they match. Otherwise it returns false.

Syntax

STRCMP *(string1,string2)*

Remarks

When comparing two text strings the STRCMP function starts numbering the characters in the target texts from one.

Example

```
        Vers        equ         “Acs”
        ...
                    if          strcmp (“\Vers”,”Sales”)
                    lh          r3,SalInd (a0)
                    else
                    if          strcmp (“\Vers”,”Acs”)
                    lh          r3,AcInd (a0)
                    else
                    if          strcmp (“\Vers”,”Test”)
                    lh          r3,TstInd (a0)
                    endif
                    endif
                    endif
```

See also: STRLEN

The INSTR Function

Description

This function searches a text string for a specified sub-string. If the string does not contain the sub-string zero is returned. If the sub-string is present, the result is the location of the sub-string from the start of the target text. There is an optional parameter specifying an alternate start point within the string.

Syntax

INSTR *([start,]string, sub-string)*

Remarks

When returning the offset of a located sub-string, the INSTR function starts numbering the characters in the target text from one.

Example

```
        Mess        equs        “Demo for Sales Dept”
                    ...
                    if          instr (“\Mess”,”Sales”)
                    lh          r3,SalInd (a0)
                    else
                    lh          r3,AcInd (a0)
                    endif
```

See also: SUBSTR

The SUBSTR Directive

Description

This directive assigns a value to a symbol. The value is a sub-string of a previously specified text string, defined by the start and end parameters. The start and end parameters will default to the start and end of the string if omitted.

Syntax

symbol **SUBSTR** *[start],[end],string*

Remarks

When assigning a sub-string to a symbol the SUBSTR directive starts numbering the characters in the source text from one.

Examples

```
Message      equs         “A short Sample String”
Part1        substr       9,14,”\Message”
Part2        substr       16,,”\Message”
Part3        substr       ,7,”\Message”
Part4        substr       ,,”\Message”
```

where: Part1 equals Sample

Part2 equals String

Part3 equals A short

The last statement is equivalent to an EQUS assigning the whole of the original string to Part4.

```
cbb    macro    string
cc     =        0
       rept     strlen (\string)
cc     substr   lc+1,lc+1,\string
       db       ‘\cc’^ ($A5+lc)
       lc+1
       endr
       endm
```

This is an example of encryption of a string.

See also: INSTR and EQUS

Chapter 8:
Local Labels

Overview

As a program develops, finding label names that are both unique and definitive becomes increasingly difficult. Using local labels eases this situation by allowing meaningful label names to be reused.

This chapter covers the following topics and directives:

- Local label syntax and scope
- MODULE and MODEND
- LOCAL

Local Label Syntax and Scope

Syntax

Local labels are preceded by a local label signifier. By default this is an @ sign however any other character may declared by using the l option in an OPT directive or on the Assembler command line (see *The Assembler Command Line Syntax* section on page 3–3).

Local label names follow the general label rules specified in Chapter 4.

Local labels are not descoped by the expansion of a macro.

Scope

The region of code within which a Local label is effective is called its scope. Outside this area the label name can be reused. There are three methods of defining the scope of a Local Label:

- The scope of a local label is implicitly defined between two non-local labels. Setting a variable, defining an equate or RS value does not de-scope current local labels unless the d option has been used in an OPT directive or on the Assembler command line (see *The Assembler Command Line Syntax* section on page 3–3).
- The scope of a Local Label can also and more normally, be defined by the directives MODULE and MODEND (see Chapter 8).
- To define labels (or any other symbol type) for local use in a macro the LOCAL directive can be used (see Chapter 8).

Examples

```
plot2       lb          r3,loc (a0)
            ...
            subiu       r3,r0,-83
            bge         r3,@chk1
            addiu       r3,r0,168
            j           @ret
@chk1       subiu       r3,r0,83
            ...
            jal         lcolour
            addiu       r2,r0,r3
SetX        set         x+1
@ret        jr          ra
            plot3
            ...
@ret        jr          ra
```

The code above shows a typical use for Local Labels, as “place markers” within a self-contained subroutine. The scope is defined by the non-local labels, `Plot2` and `Plot3`; the SET statement does not descope the routine. The labels `@chk1` and `@ret` are reusable.

```
plot2         lb            r3,loc (a0)
              subiu         r3,r0,-83
              bge           r3,@chk1
              jal           lcolour
              bra           setplot

@chk1

setplot       set           x+1

Plot3
              ...
              j             @chk1
```

In the example above, the final branch will cause an error since it is outside the scope of `@chk1`.

The MODULE and MODEND Statements

Description

Code occurring after a MODULE statement and up to and including the MODEND statement is considered to be a module. Local labels defined in a module can be reused but cannot be referenced outside the module's scope. A Local label defined elsewhere cannot be referenced within the current module.

Syntax

MODULE
...
MODEND

Remarks

Modules can be nested.

The MODULE statement itself is effectively a non-local label and will descope any currently active default scoping.

Macros can contain modules or be contained in a module. A local label occurring in a module can be referred to by a macro residing anywhere within the module. A module contained within a macro can effectively provide labels local to that macro.

Example

```
Strat       module
            addi        r1,r0,-1
            j           @Lab1
            ...
@Lab1       addi        r4,r0,-1
            beq         r4,r0,@SetTact
            ...
            jr          ra
@SetTact    module
            lh          r5,Tactic2 (a0)
                        ...
@Lab1       lw          r6,8 (a0)
            ...
            beq         r5,r0,@Lab1
            jr          ra
            modend

            modend
```

See also: LOCAL

The LOCAL Directive

Description

The LOCAL directive is used to declare a set of macro-specific labels.

Syntax

LOCAL *symbol,..,symbol*

Remarks

The scope of symbols declared using the LOCAL directive is restricted to the host macro.

The LOCAL directive does not force a type on the symbol set that makes up its operand. In practice therefore such symbols can be used as equates, string equates or any other type, as well as labels.

Example

```
doorpos     macro
            local       m_door1,m_door2,doorw
            jal         doorw
            jal         doorw

m_door1     equ         door_start*\1
m_door2     equ         door_fin*\2

doorw       ...
            endm
```

See also: MODULE

Chapter 9:
Structuring the Program

Overview

Normally, the organization of the memory of the target machine does not match the layout of the source files. To create a structured target memory, as well as to create relocatable program sections, the ASMPSX assembler uses the concept of Groups and Sections.

This chapter covers the following topics and directives:

- GROUP
- SECTION
- PUSHS and POPS
- SECT and OFFSET

The GROUP Directive

Description

This directive declares a group with up to seven group attributes.

Syntax

GroupName **GROUP** *[Attribute,..Attribute]*
where an attribute is one of the following:
WORD
BSS
ORG *(address)*
FILE *(filename)*
OBJ *(address)*
SIZE *(size)*
OVER *(GroupName)*

Attribute Descriptions

WORD-the group may be addressed using absolute word addressing. Note that this will have an effect only if the ow+ parameter has been used to allow optimization to occur.

```
Group1      group       word
```

BSS-no initialized data to be declared in this group. For example:

```
Group1      group       bss
```

ORG-sets the ORG address of the group, without reference to the other group addresses. If this attribute is omitted, the group will be placed in memory following on from the end of the previous group. For example:

```
            org         $100
G1          group
G2          group       org ($400)
G3          group
```

will place the groups in the sequence G1,G2,G3

FILE-outputs a group, such as an overlay, to the specified binary file, other groups will be output to the declared file. For example:

```
Group1      group       org ($400),file (“charov.bin”)
```

OBJ-sets the group’s OBJ address. Code is assembled as if it is running at the OBJ address but is placed at the group’s ORG address. If no address is specified then the OBJ value is the same as the group’s ORG address. For example:

```
Group1      group       org ($400),obj ($1000)
Group2      group       org ($800),obj ()
```

SIZE-specifies the maximum allowable size of the group. If the size exceeds the specified size, the assembler reports an error. For example:

```
Group1      group       size (32768)
```

OVER-overlays this group on the specified group. Code at the start of the second group is assembled at the same address as the start of the first group. The largest of the overlayed groups' sizes is used as the size of each group. Note that it is necessary to use the FILE attribute to force different overlays to be written to different output files. For example:

```
Group2       group        over (Group1)
```

See also: SECTION

The SECTION Directive

Description

This directive declares a logical code section.

Syntax

	SECTION*size*	*SectionName[,Group]*
SectionName	**SECTION***size*	*[Attribute,..Attribute]*

The second format is a special case, designed to allow definition of a section with group attributes (see below for a description).

Remarks

Unless the section has been previously assigned, if the GROUP name is omitted the section will be placed in an unnamed default group.

It is possible to define a section with group attributes. The assembler will automatically create a group with the section name preceded by a tilde (~) and place the section in it. For example:

```
Sect1        section      bss
```

defines Sect1, with the BSS attribute, in a group called `~Sect1`.

The size parameter can be b, h or w. If the parameter is omitted, the default size is word. When a size is specified on a section directive, then alignment to that size is forced at that point. The start of the section is aligned on a boundary based on the largest size on any of the entries to that section, in all modules in the case of linked code.

Note that if a section is sized as byte, the EVEN directive is not allowed in the section. Note also that it is not permitted to use the CNOP directive to realign the Program Counter to a value greater than the alignment of the host section (see Chapter 5).

If sections are used to structure application code, only a single ORG directive, which must precede all section definitions, can be used. Groups and Sections may have ORG attributes to position them. When producing linkable output, no ORG directives or attributes are permitted. Sections are ordered within a group in the sequence that the Linker encounters the section definitions.

Example

```
Sectionb     one
db           1,2,3

section      two
db           10,11,12

sectionb     one
db           4,5

section      two
db           13,14
```

Will produce the following sections and bytes

```
one    1, 2, 3, 4, 5
two    10, 11, 12, 0, 13, 14
```

See also: GROUP

The PUSHS and POPS Directives

Description

These directives allow the programmer to open a new, temporary section, then return to the original section. PUSHS saves the current section, POPS restores it.

Syntax

PUSHS
POPS

Example

```
        plotcomp    lw          r3,8 (a0)
                    ...
        passdl      equ         *
                    pushs
                    section     dolight
                    dw          passdl
                    ...
                    pops
```

This example shows PUSHS and POPS being used to pass system information, in the form of the location counter, between sections.

The SECT and OFFSET Functions

Description

The SECT function returns the address of the section in which the symbol in the brackets is defined. The OFFSET function returns the location in the host section of the symbol in the brackets.

Syntax

SECT *(expression)*
OFFSET *(expression)*

Remarks

If a link is being performed, the SECT function is evaluated when it is linked. If there is no link it will be evaluated when the second pass has finished.

Likewise, if a link is being performed, the OFFSET function is evaluated when it is linked however if there is no link, the OFFSET will be evaluated during the first pass.

Example

```
        dh     sect (Table1)
        dh     sect (Table2)
        dh     offset (*)
```

Chapter 10:
Options, Listings and Errors

Overview

This chapter completes the discussion of the ASMPSX Assembler and its facilities. It covers methods of determining run-time Assembler options, producing listings and error-handling, as well as passing information to the Linker:

- OPT
- Assembler options
- PUSHO and POPO
- LIST and NOLIST
- INFORM and FAIL
- XREF, XDEF and PUBLIC
- GLOBAL

The OPT Directive

Description

This directive allows Assembler options to be enabled or disabled, in the application code (see next page for a full list of options).

Syntax

OPT *option,..option*

Remarks

An option is turned on and off by the character following the option code:

+ (plus sign) = On
- (minus sign) = Off

Options may also be enabled or disabled by using the `/O` switch on the Assembler command line (see *The Assembler Command Line Syntax* section on page 3–3).

Example

```
        opt    an+,l+,e-
```

See also: PUSHO and POPO

Assembler Options

The following reference list shows the various options and optimizations available during assembly, with the default settings. The options are later described in detail.

Assembler options (first of +/- specified is default)

`ae+/-`	Enable automatic even mode
`an-/+`	Allow alternate number format
`at+/-`	Allow the assembler to use the temporary register
`c-/+`	Case sensitivity
`d-/+`	De-scope local labels on equ, set, etc.
`e+/-`	Print lines containing errors
`h-/+`	Automatic hazard removal (insert NOP)
`l-/+`	Use '`.`' as leading character for local labels (default '`@`')
`l<char>`	Make <char> the local label character (Note: you cannot use + or - as a local label)
`m+/-`	Enable/disable macro instructions
`n-/+`	Insert NOP in branch delay slots
`s-/+`	Treat equated symbols as labels
`t-/+`	Automatically truncate values in db/dh/dw statements
`w+/-`	Print warnings
`ws-/+`	Allow white space in operands
`v-/+`	Write local labels to symbol file
`x-/+`	Assume XREF's are in the section they are declared in

Note that the Assembler optimization options are not valid for forward references.

Option Descriptions

AE-Automatic Even

When using the word and long word froms of DC, DCB, DS and RS, enabling this option forces the program counter to the following word boundary prior to execution.

AN-Alternate Numeric

Setting this option allows the inclusion of numeric constants in Zilog or Intel format, that is, followed by H, D or B to signify Hex, Decimal or Binary. (See also *The RADIX Directive* on page 4–7).

AT-Allow the Assembler to Use the Temporary Register

Within MIPS standard format code, some instructions are not actually instructions but are macros. For example:

```
sw     t0,fred
```

when assembled will produce the following:

```
lui    at,fred>>16
sw     t0,fred&$ffff (at)
```

which trashes the AT register. If AT- then errors will be generated when instructions like those above are used. If AT+ then warnings are generated if the user uses the AT register.

C-Case Sensitivity

When the C option is set, the case of the letters in a label's name is significant, for instance, SHOWSTATS, ShowStats and showstats would all be discrete.

D-Descope Local Labels

If this option is enabled, local labels will be descoped if an EQU or SET directive is encountered.

E-Error Text Printing

If this option is enabled, the text of the line that caused an Assembler error will be printed, as well as the host file name and the line number.

H-Automatic Hazard Removal

If there is a pipeline hazard the assembler will warn the user. If H+ then the assembler will insert a NOP into the code. Note this does not apply to instructions following returns at the end of subroutines.

L-Local Label Signifier

In ASMPSX, local labels are signified by a preceding at sign (@). This option allows the use of the character following the option letter as the signifier. Thus, `L:` would change the local label character to a colon (`:`). L+ and L- are special formats that toggle the character between a dot (`.`) and an at sign (@) respectively.

M-Enable/Disable Macro Instructions

If m- then errors will be given on macro instructions. For example the following code

```
li     r2,$587329
```

will expand to the following sequence

```
lui    r2,$58
ori    r2,$7329
```

If m+ then

```
li     r2,$587329
```

will generate an error.

N-Generate a NOP in Branch Delay Slots

Will automatically insert a NOP after all branch instructions.

S-Treat Equated Symbols as Labels

If enabled then disassembly will treat equates as labels rather than just values.

T-Automatic Truncation

If enabled this will cause truncation of values to the size of the defined data. For example:

```
db -1
```

will be assembled as

```
db $ff
```

W-Print Warning Messages

The Assembler identifies various instances where a warning message would be printed but assembly is allowed to continue. Disabling the W option will suppress the reporting of warning messages.

WS-Allow White Spaces

If this option is set on, operands may contain white spaces thus the statement:

```
dw     1 + 2
```

defines a longword of value 1 with WS set off and a longword of value 3 with WS set on.

X-XREFs in Defined Section

This option set to on specifies that XREFs are assumed to be in the section in which they are defined. This allows optimization to absolute word addressing to be performed if the section is defined with the WORD attribute or is in a Group with the WORD attribute.

The PUSHO and POPO Directives

Description

The PUSHO directive saves the current state of all the assembler options, POPO restores the options to their previous state. They are used to make a temporary alteration to the state of one or more options.

Syntax

PUSHO
POPO

Example

```
                pusho                   ;save options state
                opt     ws+, c+         ;change options state
SetAlts         =       height * time
SETALTS         dh      256 * SetAlts

                popo                     ;restore previous state
```

See also: OPT

The LIST and NOLIST Directives

Description

The NOLIST directive turns off listing generation; the LIST directive may be used to turn listing on or off.

Syntax

NOLIST
LIST *indicator*

where *indicator* is a plus sign (+) or a minus sign (-).

Remarks

If a list file is nominated, either by its inclusion on the Assembler command line or in the ASMPSX environment variable, a listing will be produced during the first pass.

(The default assembler options are set with the `SET ASMPSX=` command on the command line or in the `AUTOEXEC.BAT` file.)

The Assembler maintains a current listing status variable which is initially set to zero. List output is only generated when this variable is zero or positive. The listing directives affect the listing variable as follows:

`NOLIST`	Sets it to -1
`LIST`	With no parameter, sets it to zero
`LIST +`	Adds 1
`LIST -`	Subtracts 1

Example

Directive	Status	Listing produced
Nolist	-1	No
List -	-2	No
List	0	Yes
List -	-1	No
List -	-2	No
List +	-1	No
List +	0	Yes

Note: The Assembler automatically suppresses production of listings in the following circumstances:

- During macro expansion
- For unassembled code because of a failed conditional

These actions can be overridden by:

- Including the /M option on the Assembler command line to list expanding macros
- Including the /C option on the Assembler command line to list conditionally ignored code (see *The Assembler Command Line Syntax section* on page 3–3).

The INFORM and FAIL Directives

Description

The INFORM directive displays an error message contained in text, which may optionally contain parameters to be substituted by the contents of expressions after evaluation. Further Assembler action is based upon the state of severity. The FAIL directive is a predefined statement, included for compatibility with other assemblers. It generates an “Assembly Failed” message, and halts assembly.

Syntax

INFORM *severity,text[,expressions]*
FAIL

Remarks

These directives allow the programmer to display an appropriate message if an error condition which the Assembler does not recognize is encountered.

Severity is in the range 0 to 3, with the following effects:

0: the Assembler simply displays the text

1: the Assembler displays the text and issues a warning

2: the Assembler displays the text and raises an error

3: the Assembler displays the text, raises a fatal error and halts the assembly

Text may contain the parameters `%d`, `%h` and `%s`. They will be substituted by the decimal, hex or string values of those expressions that follow.

Example

```
TableSize     equ             TableEnd-TableStart
MaxTable      equ             512
              if              TableSize>>MaxTable
              inform          0,”Table starts at %h and&
                              is %h bytes long”,&
                              TableStart,TableSize
              inform          3,”Table Limit Violation”
              endif
```

The XDEF, XREF and PUBLIC Directives

Description

If several subprograms are being linked, to refer to symbols in a subprogram which are defined in another subprogram, use the XDEF, XREF and PUBLIC directives.

Syntax

XDEF *symbol[,symbol]*
XREF *symbol[,symbol]*

PUBLIC *on*
PUBLIC *off*

Remarks

In the subprogram where symbols are initially defined, the XDEF directive is used to declare them as externals.

In the subprogram that refers to the symbols, the XREF directive is used to indicate that the symbols are in a another subprogram.

The Assembler does not completely evaluate an expression containing an XREFed symbol however resolution will be effected by the linker.

The PUBLIC directive allows the programmer to declare a number of symbols as externals. With a parameter of on, it tells the Assembler that all further symbols should be automatically XDEFed until a PUBLIC off is encountered.

Examples

Subprogram A contains the following declarations:

```
        xdef   Scores,Scorers
        ...
```

The corresponding declarations in subprogram B are:

```
        xdef         PointsTable
        xref         Scores,Scorers
        ...
        ____________________________________________

                     public       on
        Origin       =            MainChar
        Force        dh           speed*origin
        Rebound      dh           45*angle
                     public       off
```

See also: XREF, XDEF and PUBLIC (page 10–10).

The GLOBAL Directive

Description

The GLOBAL directive allows a symbol to be defined which will be treated as either an XDEF or an XREF. If a symbol is defined as GLOBAL and is later defined as a label, it will be treated as an XDEF. If the symbol is never defined, it will be treated as an XREF.

Syntax

GLOBAL *symbol[,symbol]*

Remarks

This is useful in header files because it allows all separately assembled subprograms to share one header file, defining all global symbols. Any of these symbols later defined in a subprogram will be XDEFed, the others will be treated as XREFs.

See also: XREF, XDEF and PUBLIC

Chapter 11:
The Debugger DBUGPSX

Overview

DBUGPSX is a full source level debugger, as well as a traditional symbolic debugger. This allows source code to be viewed, run and traced, stepped-over and breakpoints set and cleared.

The original symbolic debug facilities are all still available. A source level display will revert to a symbolic disassembly when no source level information is available.

The following Debugger topics are discussed in this chapter:

- Command line syntax
- Configuration files
- Activity windows
- General debugger usage
- Keyboard options
- Menu options

For full operation of the debugger the pollhost function/macro should be included in the source code of the program to be debugged (see Chapter 12).

Command Line Syntax

Syntax

DBUGPSX */switches filename filename*

or

DBUGPSX ? *which displays a help message.*

Remarks

Filename specifies the name of a file containing symbols, produced by the using the /zd option during assembly. If no extension is shown, a default extension of .SYM will be added. Multiple filenames are allowed and must be separated by a space, the symbol files will then be loaded in the order specified. If CCPSX is being used to drive the compile-assemble-link sequence then the -g option must be used to produce symbols (see readme files on issue disks for latest options etc.). Generally when debugging, compiler optimizations such as instruction re-ordering should not be used as this makes source level debugging virtually impossible.

Valid Switches

`/h`	Halts target machine when Debugger starts
`/s`*file*	Overrides default configuration filename
`/v`*exprtext*	Evaluates expression text and put result to standard output device
`/e`*file[,file,file]*	Loads target machine with CPE file (s)
`/r`	Specifies data screen rows in video bios
`/c-`	Turns case sensitivity off
`/c+`	Turns case sensitivity on
`/d`	Disables automatic run-to-main at C program start-up
`/f$xxxxxxxx`	Finds line number and file for address $xxxxxxxx
`/i`	Specifies update interval (in 1/18ths sec)

`/l`	Sets Debug label level
`/t`	Sets target SCSI device number (default is 0)
`/u-`	Turns continual update mode off
`/u+`	Turns continual update mode on
`/&`*expr,.. expr*	Lists of parameter expressions separated by commas
`/m`	Sets the Debugger mouse sensitivity. is a number between 1 and 4, 3 is the default. DBUGPSX drives the mouse itself. This overcomes some shortcomings exhibited by the Microsoft mouse driver, particularly in 132 column mode
`/m+`	Uses the current system mouse driver; later versions of the Microsoft drivers (8 upwards) allow the mouse to be used in a DOS window
`/m-`	Reverts to the DBUGPSX mouse driver
`/R`	Uses alterante mnemonics in register window

Source level mode can be used if a symbol file is specified on the command line. This file contains symbols and additional source level information produced by the `/zd` option in the ASMPSX assembler.

Expressions passed to the Debugger using the `/&` switch can be referred to in the form &0, &1, etc., where 0 means the first expression on the command line, 1 means the second.

Configuration Files

When DBUGPSX is loaded it accesses a configuration file containing information about the current Debugger environment. The current configuration can be saved at any time during an active Debugger session. The default filename can be overridden with an option on the command line (`/s`) or at run-time, so that the most frequently used configurations are always readily available.

Configuration File Names

The normal configuration file name is `DBUGPSX.C`, where the first number is the target SCSI id number, and the second number is the virtual screen. Typically therefore the configuration loaded at start-up is `DBUGPSX.C00`.

If this file is not located in the current directory, the Debugger looks for a file called `DBUGPSX.DF` (the default configuration file).

If that file is not located, the Debugger home directory will be searched for a file built from the target name string. For example:

```
SONY_PSX.CFG
```

Contents of Configuration Files

A configuration file can include the following information:

- Read memory ranges
- Write memory ranges
- Video type and usage
- Label level
- Colour and mono attributes
- Tab settings
- Current window type and display position
- Breakpoints
- History details

Activity Windows

The Debugger display consists of one or more activity windows. The number of windows, the contents of each window and the window size can all be specified at run-time. The default display consists of two windows. The upper window normally contains a display of the registers, the lower window shows the disassembly of the code at the current pointer.

The Debugger can run up to 10 virtual screens; each screen has its own configuration file (see previous page). Alternate screens can be accessed by pressing `Alt-n`, where n is the screen number 0-9, where 0 means screen 10.

Window Types

The Register Window provides a complete view of the selected processor's registers. Register contents can be changed:

- By typing directly at the current cursor location
- Entering an expression (pressing the Enter key displays an expression input window)

The Disassembly Window shows the contents of the target memory as disassembled code. If a symbol file has been loaded into the Debugger, symbol names are substituted as appropriate, according to the label level. Breakpoints, conditions and counts can be added to any line and the code run, traced or stepped.

If the PC of the target machine is pointing at a line in the current disassembly display it is indicated by a greater than sign (>). If a line contains a breakpoint, that line is displayed in a different colour; the breakpoint count and expression details are shown at the end of the line.

The Hex Window displays memory in hexadecimal, either in byte, word or long word form. Like the Register Window contents can be changed:

- By typing directly at the current cursor location
- Entering an expression for evaluation (pressing the Enter key displays an expression input window)
- Pressing + or - will increment or decrement the value at the cursor position

The Watch and Variable windows allow variables, tables and code locations to be monitored as your program is running.

The Variable window automatically tracks the scope of your C program. As you trace though your program and the variable scope changes, this window will always display the current local variables.

The Watch window performs a similar function for user specified expressions and is typically used to display global variable data. You can enter all your global variables in this window by pressing Alt-G. Specific C or Assembler expressions (global and local) can be entered at the cursor position. Entries can also be deleted. All entries in a Watch window are saved when you exit and are restored the next time you run the Debugger.

In both the Variable and Watch windows, pointers and arrays can be de-referenced and structures, unions and enums can be opened up for closer examination, by placing the cursor over the relevant entry and pressing '+' from the numeric keypad. They can be subsequently closed by pressing '-'. This will even work for local register structures within unions within structures etc.

The Text or File Window allows a text file to be viewed directly. Note that pressing the Enter key, while the cursor is in the File Window, allows entry of a further file name for display.

The Source Level window is an extension of the File window. Most source level key commands are the same as for a Disassembly window.

To enter source mode, tell the File window to display program source at a particular address. The easiest way to do this is to hit the TAB key. As in a Disassembly window, this causes the window to locate to the current program counter address. If the Debugger has source level information for that part of your program, it will display corresponding source code.

Alternatively, you can enter source mode by typing Alt-G (for the Goto location) and then entering the address you wish to locate to (any expression, or Assembly language label name, or C function name, such as *main*, will do fine).

Note that in Source mode, line numbers are added to the left side of the window display and the PC line is indicated with a '>' after the line number, similar to a Disassembly window.

If you wish to view text which is truncated off the right side of the window then the window can be scrolled to the left and right using the left and right cursor keys.

You can step, trace, run to cursor and set breakpoints in the source code in much the same way as a for Disassembly window. The cursor in the currently active text window will track the PC during a trace. Note, however, that unlike tracing in a Disassembly window, a trace at Source Level may trace more than one instruction as it will trace the entire source line, which, if it is a macro or a 'C' source line, may correspond to the execution of one or more instructions. Similarly F8 (Stepover) will step-over the entire source line, which could be equivalent to stepping over several subroutine calls.

If you are unsure of how a Source Level operation will behave, a Disassembly window can be viewed at the same time to determine how the operations correspond to actual processor instructions. If you attempt to step into a C function or Assembly language subroutine for which the Debugger does not have any source level information, then the Debugger will attempt to perform a step-over operation instead. If this is not possible (e.g. if the code without source level information is jumped to rather than called), then the window display will switch to Disassembly mode. The trace can be continued and when the PC returns to a region for which there is Source information. The window will switch back to the Text display.

In order to use any of the Source Level features, you must have the necessary extra debugging information in your Symbol File(s). If this is not present, then the Debugger will be unable to switch to Source mode and Source Level operations will produce appropriate error messages. This information is added to the Symbol Files by the C compiler if you add the -g switch to your CCPSX command line, or by the Assembler if you specify the /zd switch on the ASMPSX command line.

The Symbol File normally contains the full original pathnames of all files used to build your project. When Source Level debugging, the Debugger will attempt to load those files from the same locations. In some cases this may not be convenient, e.g. if part of the project was built by another developer on a different PC or on a network drive. Even if you have a copy of the appropriate Source Files you may not have them at the same location.

To get around this you can provide the Debugger with a search patch for Source Files. Just select a Source File window and type Alt-P. You will be prompted to enter a normal DOS search path. This search path can contain many entries and be as long as you wish.

For example,

```
c:\;c:\temp\myfiles;c:\gnumips\src\common
```

The Debugger will first look for the file in the directory specified in the Symbol File (determined when the project was built). If it cannot locate the file at the original location then it will search for it by name at the following locations:

```
c:\filename.sym
c:\temp\myfiles\filename.sym
c:\gnumips\src\common\filename.sym
```

The first filename match located will be assumed to be the correct Source File.

The search path will be saved in the Debugger Configuration File when you exit the Debugger. To remove an existing search path just specify a blank search path string.

Additional Debugger Features

Automatic Overlay Support allows the Debugger to dynamically track overlays as they are loaded into your program and work with the Source Files and variables specific to that overlay. This requires no modification

to your Source Code. You need only to tell the Linker which files will overlay in memory. Any number of concurrent overlays are supported over multiple memory areas. A simple overlay is included with the SDevTC software.

Big Text Screens allow you to view as much information as possible by working in a higher text resolution. This is achieved by putting the text screen in the required resolution before running the Debugger. At least 80x50 is recommended, but the Debugger will happily work in higher text resolutions up to 132x66. Most modern VGA cards are capable of 132 column text modes and come with a utility to set such resolutions. SDevTC also includes the BV.exe program which will take any screen mode and adapt it to 30, 32, 60 or 64 lines. Use the /r50 debugger command line switch if you prefer to edit at 80-25 but debug at 80x50.

Multiple Text Screens are available when a single 132x64 screen is not enough to display all your debug information. Up to 10 virtual screens can be used. Switch between them by pressing Alt-*n* (where *n* is a digit 1,2,3,4,..0). The configuration at each screen is saved when you switch, and is restored when you switch back. These configurations are also saved when you exit the Debugger.

Name Completion is provided by all prompts where C or Assembler labels are entered. Type the first few letters of the name and press Ctrl-N. If the required label does not appear, repeat Ctrl-N to 'toggle' through the alternatives. This facility also works for a name in the middle of an expression.

Prompt Histories are provided to allow you to select from your history of prior entries. This information is saved when you exit the Debugger and restored for your next debugging session.

CPU Hardware Breaks can be used to cause the PlayStation CPU to stop when a specific memory location is read from or written to. Hardware breaks can be accessed by typing Alt-B.

Note that syntax from either C or Assembler can be used with this facility.

In Assembler mode you will be prompted for a mask value. This mask has 1 bit to enable, therefore a mask value of -1 ($FFFFFFFF) will be the usual value to trap one particular address.

In C mode you only need to enter the name of your C variable. The Debugger will automatically calculate the correct address and mask value.

General Debugger Usage

The following key strokes and mouse actions allow the programmer to exercise control over the Debugger display. A complete list of all key and menu options is given later in the chapter.

Moving between Windows

Use one of the following methods to move between Debugger windows:

- Press F1, followed by a cursor (up, down, left or right) key to point to the required window
- Press Shift, plus a cursor key
- Point at the required window with the mouse and click

Selecting the Window Type

Do one of the following to change the type of the currently selected window:

- Use the mouse to select the Set Type option from the Window menu
- Press Shift and F1

In each case, a selection window is presented. Use the mouse or the cursor keys, plus ENTER, to choose the new type.

Resizing Windows

To change the size of a Debugger window:

- Position the cursor in the required window
- Press F2
- Use the cursor keys to move the selected window edge to the desired size
- Press the Enter key to confirm

Note that the currently selected window may be zoomed to fill the screen by pressing Control-Z. Press again to re-present the original display

Splitting an Existing Window

To add another window to the display:

- Position the cursor in the required window
- Press F3

The new window is the same type as the source window.

Joining Two Windows

To remove a Debugger window:

- Position the cursor in the required window
- Press F4
- Use the cursor keys to select the window edge to be removed
- Press Enter to confirm

Moving the Cursor within a Window

The cursor control keys allow the repositioning of the cursor in the selected window, as follows:

Window Specific Control

Register Window: Use the four arrow keys to move between register values. The Home key positions the cursor in the top left register field.

Watch Window: Use the four arrow keys to move between adjacent lines and characters. The Home key positions the cursor in the top left character position.

Disassembly and Text Window: Use the up and down arrows to move the highlight bar. The Home key moves the line under the cursor to the top of the window.

Hex Window: Use the four arrow keys to move between adjacent lines and bytes/words. The Home key moves the byte/word under the cursor to the top left of the window.

Locking a Window

A window can be locked into displaying a specific memory region as follows:

- Pressing Alt-L, and entering an address, or an expression which evaluates to an address, in the input box.
- Selecting the Lock option from the Window menu.
- Pressing Control-L turns the lock on and off.

A display can be locked to the expression &0; this allows the Debugger to be started with a window pointing to an address or label specified on the command line.

If a lock expression is set but deactivated by Control-L the Debugger will start-up with the display initially positioned at the lock address but the window start can subsequently be changed with the cursor keys, etc., as normal.

General Mouse Usage

Clicking the left mouse button repositions the cursor to the site of the click. If the new position is in another window, it will become the active window.

Clicking the right mouse button on a register in the Register window will open an expression input box.

Clicking the right mouse button on a memory field in the Hex window will open an expression input box.

Clicking the right mouse button on a line in the Disassembly window toggles a breakpoint.

A window can be re-sized by clicking the left mouse button on a window edge and dragging it to the new position.

Dragging a window border to the edge of the window deletes the window.

Keyboard Options

The following table is a complete list of keyboard options, categorized by function. Many of these functions are duplicated by menu options however such functions are shown in both lists for reference purposes.

Expressions

At many points in the session the Debugger will prompt for input, this can often take the form of an expression for evaluation. Expressions in the Debugger follow the same rules as the Assembler (see Chapter 4) with the following exceptions:

- Expressions may contain processor registers
- The Debugger assumes a radix of hexadecimal; to indicate a number is decimal, it should be preceded by a sign
- Indirect addresses are indicated by square brackets [ ]
- When the Debugger gets an indirect datum it assumes a long word. This can be overridden by using the @ sign in place of the dot, together with b or w, following the square bracket

Prompts

Each time the Debugger requests input, the reply is stored. These stored prompts form a history, which can be accessed (and then edited) at data entry time by using the up and down arrow keys. Note that, when the Debugger closes the last four historic entries in each class are stored on the configuration file and restored the next time that the Debugger is loaded.

Table of Options

Table 11-1: DBUGPSX Options

Key(s)	Effect
Leaving the Debugger	
`Ctrl-X`	Exit Debugger without saving the current configuration
`Alt-X`	Exit Debugger and save the current configuration
Window handling	
`F1`	Move to next window

F2	Resize Window
F3	Divide Window into two
F4	Delete Window
Shift-Arrows	Move to selected Window
Ctrl-Z	Zoom current Window; again to restore original display
Shift-F1	Select Window type
Debug control	
Ctrl-F2	Re-start paused Debugging session if using a CPE file
Esc	Halt the target at first opportunity
Shift-Esc	Halt the target turning off interrupts
Alt-R	Restore registers from previous save
Alt-I	Set the update interval: the interval is input in 18ths of a second. Therefore 18 means once a second, 9 means twice a second, etc.
Alt-U	Turn update mode on or off
F6	Run target code until the instruction under the cursor is reached
F7	Single step (steps into subroutine)
F8	Single step (steps over subroutine)

Key(s)	Effect
F9	Run target code from current program location
Shift-F7	Trace traps and break instructions
Shift-F9	Run to address specified in input window
Alt-F4	Backtrace: this function provides an UNDO of the updates effected by the latest trace (the Debugger keeps a record of about 200 instructions depending on their content). Note that updates to certain write registers in the target machine and memory areas designated as write only, cannot be undone.
File accessing	
<	Upload specified data from the target to a named file on the PC
>	Download a file to the target
Shift-F10	Load a new configuration file
Alt-S	Send specified section of disassembly to a PC file
Miscellaneous	
F10	Select a menu option
Alt-H	Hex calculator: enter an expression to be evaluated
Alt-*n*	Switch to virtual screen n (1-9 plus 0=10)
Alt-N	Label continuation: enter first few letters from a label and press Alt-N to find first match. Repeated Alt-N cycles through all matches
Disassembly window	
Up/Down Arrows	Move highlight bar
Left/Right Arrows	Move display by one word
PgUp/Dn	Move display by a page
Home	Move display so that the highlighted line is at the top
Alt-G	Go to address specified in input window
Tab	Move the highlight bar to the Program Counter
Shift-Tab	Make the Program Counter the same as the currently highlighted address
Alt-L	Lock the display to a specified address
Ctrl-L	Turn lock on or off
Ctrl-D	Disassembly memory to a PC text file

`Ctrl-S`	Search for a particular instruction fragment (such as text, space as separator, etc.)
`Ctrl-N`	Continue search
`Enter`	Mini-Assembler: displays an input box to enter a single line of source code to be assembled and inserted at the location under the cursor
File and source windows	
`Tab`	Load appropriate source file. When in source level mode use keys as in Disassembly window
`Alt-G`	Locate source display to specified address
`Alt-T`	Override default tab settings for this window
	(prompts for a list of tab positions)
`Ctrl-S`	Search for a text string
`Ctrl-N`	Continue search
`Alt-P`	Specify source-file search path
Breakpoints	
`Alt-C`	Enter condition for the highlighted breakpoint
`Ctrl-C`	Enter count for the highlighted breakpoint

Key(s)	Effect
`F5`	Turn highlighted breakpoint on or off
`Shift-F5`	Clear all current breakpoints
`Shift-F6`	Reset all current breakpoint counts
Hex Window	
`Arrow`	Move to adjacent byte/half word/word
`PgUp/Dn`	Move display by one page
`Home`	Move display so that currently highlighted byte/half word/word is at the top
`Alt-W`	Switch display between byte, half word and word
`Enter`	Change contents of current location to the result of an expression entered in an input window
`0-9 A-F`	Directly change contents of highlighted location
`+`	Increment contents of highlighted location
`-`	Decrement contents of highlighted location
`Alt-G`	Go to address specified in input window
`Alt-F`	Move display to address contained in highlighted location
`Ctrl-S`	Search for a hex (or text if started with " character) string
	(prompts for a space-separated list of bytes/words/longs)
`Ctrl-N`	Continue search
Register Window	
`Arrows`	Move to next register
`Home`	Move to top left register
`Enter`	Change contents of current register to the result of an expression entered in an input window
`0-9 A-F`	Directly change contents of highlighted register
Watch Window	
`Arrows`	Move to next watch expression
`Home`	Move to top watch expression
`Ins`	Add a new watch expression
`Del`	Delete the highlighted watch expression
`+`	Opens information on the data under the cursor (structure, array etc.) or

	de-references the data if it is a pointer
`-`	Closes expanded information
`Tab`	Changes the *result display* format of a C expression under the cursor
`Right/Left`	Increments/Decrements array index under the cursor (currently only if not expanded with +)
Var Window	
`Up/Down`	Move to next watch expression
`Home`	Move to top watch expression
`Ins`	Add a new watch expression
`Del`	Delete the highlighted watch expression
`+`	Opens information on the data under the cursor (structure, array etc.), or de-references the data if it is a pointer
`-`	Closes expanded information
`Tab`	Changes the *result display* format of a C expression under the cursor
`Right/Left`	Increments/Decrements array index under the cursor (currently only if not expanded with +)

Key(s)	Effect
`< >`	(Also comma and dot) Crawls up and down the stack, adjusting the scope of current debugger display to that of calling functions. Top level is a 'C' callstack display
'C' Callstack Display	
`>`	(Also dot) Return to current scope
`Enter`	Display the variables of the scope under the cursor.

Menu Options

The DBUGPSX menu affords easy mouse access to the commonest Debugger functions. Note that, if no mouse is available, the Menu can still be accessed by pressing F10.

Table 11-2: DBugPSX Menu Options

Option	Effect
FILE menu	
Reload	Reload the last executable file
Download	Download a file to the target
Upload	Upload specified data from the target to a named file on the PC
Disassemble	Send specified section of disassembly to a named PC file
Exit to DOS	Exit to DOS shell, type EXIT to return to the Debugger
Exit Debugger	Exit the Debugger and save the current configuration
RUN menu	
Go	Run target code from the current Program Counter
Stop	Halt the target machine, turning off all interrupts
To Address	Run to address, specified in input window
Backtrace	This function provides an UNDO of the updates effected by the latest trace (the Debugger keeps a record of about 200 instructions, depending on their content). Note that updates to certain write registers in the target machine and memory areas designated as write only, cannot be undone
WINDOW menu	
Set Type	Select window type
Lock	Lock the display to the address entered in an input window

Print	Output screen to system printer
Set Tabs	Enter up to 8 tab positions, in decimal, separated by spaces. Note this function is only relevant to File and Source Disassembly windows
CONFIG menu	
Load	Load a new configuration file
Save	Save the current configuration to the specified file
CPU menu	
Save Regs	Save the current state of the registers
Reset Regs	Reload the previously saved register state
Reset	Reset the target processor
STEP menu	
	Trace Mode: traps and break instructions are stepped over
STEPOVER menu	
	Stepover mode: subroutine call instructions are stepped over

Chapter 12:
The Debug Stub Functions

Overview

If the debugger stub on the target machine has been initialized (by setting the dip switches on the PSX appropriately, see the PSX target box documentation) then debugger stub services are available from assembly language using the BREAK opcodes. These services include fileserver functions which allow the PSX to access files on the PC hard disk as well as some specific debugger functions. These fileserver functions are also accessible as 'C' callable functions and the pollhost function as a 'C' macro, provided by the default library `LIBSN.LIB`. The facilities provided are discussed in two sections:

- Assembly language facilities
- The 'C' library functions

Note: The Pollhost function should be included in user source code for proper operation of the debugger. It is responsible for transferring data to the PC debugger so should be put in the main loop or the vertical blank interrupt of the program to be debugged. This call takes time to transfer the data however if updates are turned off (see Chapter 11) or the debugger is exited, then the call will return immediately.

Assembly Language Facilities

To call a particular function, load any parameters into the specified registers then issue the BREAK command followed by the appropriate code.

Table 12-1: Function Codes

Code	Function	Meaning and Arguments
`0x0000`		Used by debugger as breakpoint for step/trace etc. but can also be used to halt user code for examination by debugger.
`0x0101`	PCinit()	Initializes PC remote filing system Passed: nothing Returns: v0 = error code (0 if successful)
`0x0102`	PCcreate()	Creates a file on the PC Passed: a1 = pointer to ascii file name (zero terminated) a2 = file attributes Return: v0 = PC error code (0 if successful) v1 = file handle
`0x0103`	PCopen()	Opens a file on the PC Passed: a1 = pointer to ascii file name (zero terminated) a2 = file open mode (0 is read/write, see PC DOS info) Return: v0 = PC error code (0 if successful) v1 = file handle
`0x0104`	PCclose()	Closes a file on the PC Passed: a1 = file handle Return: v0 = error code (0 if successful)
`0x0105`	PCread()	Reads a file on the PC Passed: a1 = file handle a2 = count of number of bytes to read a3 = pointer to buffer Return: v0 = error code (0 if successful) v1 = amount of data actually read

Code	Function	Meaning and Arguments (Cont.)
`0x0106`	PCwrite()	Writes a file on the PC Passed: a1 = file handle a2 = count of number of bytes to write a3 = pointer to buffer Return: v0 = error code (0 if successful) v1 = amount of data actually written
`0x0107`	PCseek()	Seeks (moves file pointer) position in a PC file Passed: a1 = file handle a2 = seek offset a3 = file seek mode (0 = relative to start of file) (1 = relative to current file pointer) (2 = relative to end of file)
`0x0400`		Pollhost PC. (To service PC data request etc) Preserves all registers This function polls the host PC and allows it access to the PSX memory, necessary, for example, during real time debugging (see the first page of this chapter)
`0x0401`		Cold Start debugger stub. Does not return any values
`0x0402`		Warm Start debugger stub. Does not return any values
`0x0403`		Enables/Disables interrupts while in downloader Default at cold-start is interrupts-disabled while debugger stub is running Passed: a0 = value to be ANDed with status reg on entry to debugger a1 = value to be ORed with status reg on entry to debugger Default values at debugger startup are 0xFFFFFFFC and 0: i.e., interrupts off during debugger functions
`0x0404`		Enables/Disables CPU cache Passed: a0=0 to disable a0=1 to enable cache
`0x0405`		Unhooks debugger from interrupt vectors This will restore the interrupt vector at 0x00000080 to its previous state (as it was before the debugger stub was installed)
`0x0406`		Flags which interrupts are to be handled by debugger Passed: a0 = bit mask (1 bit per interrupt source) Each bit masks an interrupt source (as per each possible value of CAUSE register) Bit No. Corresponding interrupt source 00 Interrupt 01 Not on PSX (TLB modified exception on R3000) 02 Not on PSX (TLB load exception) 03 Not on PSX (TLB store) 04 Address Error (load or instr fetch) 05 Address Error (Store) 06 Bus Error (instr fetch) 07 Bus Error (Data load or store) 08 Syscall 09 Break 10 Reserved Instruction 11 Copro Unusable 12 Arithmetic Overflow Bit=1: Interrupt to be trapped by debugger Bit=0: Interrupt to be passed to old handler (for example OS) Default value for this flag word at debugger stub cold start is bit 0 not set (0), bits 1 to 12 set (1)

The ‘C’ Library Functions

The following are provided by LIBSN.LIB and declared in LIBSN.H. The majority are fileserver functions but there is also one macro to provided feedback for the debugger:

Pollhost
PCinit
PCopen
PClseek
PCread
PCwrite
PCclose

The Pollhost Macro

Description

Causes the target box to poll the host PC allowing the debugger access to the PSX memory while it is running code, that is, not single stepping.

Syntax

pollhost is defined as follows:

define **pollhost ()** *asm ("break 1024") /* Ox0400 */*

so its syntax is obvious:

pollhost ()

Remarks

A macro is used so the call is inline preserving the variable scope.

This function should be included in user source code for proper operation of the debugger. It is responsible for transferring data to the PC debugger therefore it should be put in the main loop of the program to be debugged or in a vertical blank interrupt. This call will obviously take time to transfer the data however if you turn updates off (see Chapter 11) or exit the debugger then the call will return immediately.

The PCinit Function

Description

This function reinitializes the PC filing system, closes open files etc..

Prototype

int **PCinit** (void);

passed: void

return: error code (0 if no error)

The PCopen Function

Description

This function opens a file on the PC host.

Prototype

int **PCopen** (char *name, int flags, int perms);

passed: PC file pathname
open mode (0 = read access, 1 = write access, 2 = read/write access)
permission flags (this is included only for UNIX compatibility and should be set to 0)

return: file-handle or -1 if error

The PClseek Function

Description

This function seeks the file pointer to the specified position in the file.

Prototype

int	**PClseek** (int fd, int offset, int mode);
passed:	file-handle seek offset seek mode (mode 0 = rel to start, mode 1 = rel to current fp, mode 2 = rel to end)
return:	absolute value of new file pointer position

Remarks

To find the length of a file which is to be read into memory perform:

```
len = PClseek ( fd, 0, 2);
```

This will set len to the length of the file and can then be passed to PCread ().

The PCread Function

Description

This function reads a specified number of bytes from a file on the PC.

Prototype

int	**PCread** (int fd, char *buff, int len);
passed:	file-handle buffer address count
return:	count of number of bytes actually read

Remarks

Unlike the assembler function this provides for a full 32-bit count

PCread should not be passed extreme values of count, as could be done on a UNIX system, as this will cause the full amount specified to be transferred, not just to the end of the file. To find the length of a file which is to be read into memory perform:

```
len = PClseek ( fd, 0, 2);
```

This will set len to the length of the file and can then be passed to PCread ()

The PCwrite Function

Description

This function writes the specified number of bytes to a file on the PC

Prototype

int	**PCwrite** (int fd, char *buff, int len);
passed:	file-handle buffer address count
return:	count of number of bytes actually written

Remarks

Unlike assembler function this provides for full 32-bit count

The PCclose Function

Description

This function closes an open file on PC

Prototype

int	**PCclose** (int fd);
passed:	file-handle
return:	negative if error

Chapter 13:
The PSYLINK Linker

Overview

The SDevTC Linker, PSYLINK, is a fully-featured linker which works with all processor types and is compatible with other popular cross-compilers, such as Sierra and Aztec C. It facilitates the splitting of complex programs into separate, manageable subprograms, which can be recombined by PSYLINK into a final, single application.

This chapter discusses the linker, together with the Librarian utility, under the following headings:

- Command line syntax
- Linker command files
- XDEF, XREF and PUBLIC
- GLOBAL

The Linker-associated Assembler directives are repeated here for ease of reference.

Command Line Syntax

Description

The PSYLINK link process is controlled by a series of parameters on the command line and by the contents of a Linker command file. The syntax for the command line is as follows:

Syntax

PSYLINK *[switches] sourcefiles,outputfile,symbolfile,mapfile,libraries*

If an argument is omitted the separating comma must still appear, unless it is the last argument specified on the line.

Linker Switches

Switches are preceded by a forward slash and separated by commas. The following switches are available:

Table 13-1: PSYLINK Switches

Switch	Description
`/a`	Sets C variable alignment to 2 bytes
`/b`	Specifies that the linker should run in 'big' mode. This allows the linker to link larger programs but with a link-time penalty
`/c`	Tells the linker to link case sensitive; if it is omitted, all names are converted to upper case
`/d`	Debug Mode, performs link only
`/e` *symb=value*	Assigns value to symbol
`/i`	Invokes a window containing Link details
`/l` *path*	Specifies path to search for library files
`/m`	Outputs all external symbols to the map file
`/n` *maximum*	Sets the maximum number of object files or library modules that can be linked from 1 to 32768; default is 256; higher values require larger amounts of memory
`/o` *address*	Sets an address for an ORG statement
`/o ?`*address*	Requests target to assign memory for ORG

/p	Outputs padded pure binary object code; ORGed sections of code are separated with random data
/ps	Outputs ASCII representation of binary file in Motorola s-record format
/r *format*	Creates machine specific relocatable output
/s	All sections must be in defined groups
/u *number*	Specifies the unit number in a multi-processor target
/x *address*	Sets address for the program to commence execution
/z	Clears all requested BSS memory sections

Arguments

Sourcefile(s)	A list of code source files, output by the ASMPSX assembler. File names are separated by spaces or plus (+) signs; if the file starts with an @ sign, it signifies the name of a Linker command file (see next page for a description of the format).
Outputfile	The destination file for the output object code, if omitted, no object code is produced. If the output file name is in the format Tn:, the object code is directly sent to the target machine, n specifies the SCSI device number.
Symbolfile	The destination file for the symbol table information for use by the Debugger.
Mapfile	The destination file for map information.
Libraryfiles	Library files available (see Chapter 14).

Linker Command Files

Command files contain instructions for the Linker about source files and how to organize them. The Linker command file syntax is much like the Assembler syntax, with the following commands available:

Commands

INCLUDE filename	Specify name of object file to be read
INCLIB filename	Specify library file to use
ORG address	Specify ORG address for output
WORKSPACE address	Specify new target workspace address
name EQU value	Equate name to value
REGS pc=address	Set initial PC value
name GROUP attributes	Declare group
name SECTION attributes	Declare section with attributes
SECTION name[,group]	Declare section, and optionally specify its group
name ALIAS oldname	Specify an ALIAS for a symbol name
UNIT unitnum	Specify destination unit number

Group Attributes

BSS	Group is uninitialized data
ORG (*address)*	Specify group's org address
OBJ *(address)*	Specify group's obj address

OBJ *( )*	Group's obj address follows on from previous group
OVER *(group)*	Overlay specified group
FILE *("filename")*	Write group's contents to specified file
SIZE *(maxsize)*	Specify maximum allowable size

Remarks

Sections within a group are in the order that section definitions are encountered in the command file or object/library files.

Any sections that are not placed in a specified group will be grouped together at the beginning of the output.

Groups are output in the order in which they are declared in the Linker command file or the order in which they are encountered in the object and library files.

Sections which are declared with attributes, that is, not in a group, in either the object or library files, may be put into a specified group by the appropriate declaration in the Linker command file.

Example

```
                include     "inp.obj"
                include     "sort.obj "
                include     "out.obj"

                org         1024
                regs        pc=progstart

comdata         group
code            group
bssdata         group       bss

                section     datal,comdata
                section     data2,comdata

                section     codel,code
                section     code2,code

                section     tables,bssdata
                section     buffers,bssdata
```

XDEF, XREF and PUBLIC Directives

Description

If several subprograms are being linked; to refer to symbols in a subprogram which are defined in another subprogram, use XDEF, XREF and PUBLIC.

Syntax

XDEF *symbol[,symbol]*
XREF *symbol[,symbol]*
PUBLIC *on*
PUBLIC *off*

Remarks

In the subprogram where symbols are initially defined, the XDEF directive is used to declare them as externals.

In the subprogram which refers to the symbols, the XREF directive is used to indicate that the symbols are in a another subprogram.

The Assembler does not completely evaluate an expression containing an XREFed symbol however resolution will be effected by the linker.

Specifying a size of w on the XREF directive indicates that the symbol can be accessed using absolute word addressing.

The PUBLIC directive allows the programmer to declare a number of symbols as externals. With a parameter of on it tells the Assembler that all further symbols should be automatically XDEFed until a PUBLIC off is encountered.

Examples

Subprogram A contains the following declarations:

```
        xdef        Scores,Scorers
        xref.w      PointsTable
        ...
```

The corresponding declarations in subprogram B are:

```
        xdef        PointsTable
        xref        Scores,Scorers
        ...
```

```
                    public              on
        Origin      =                   MainChar
        Force       dh                  speed*origin
        Rebound     dh                  45*angle
                    public              off
```

See also: The XDEF, XREF and PUBLIC Directives on page 10-10

GLOBAL

Description

The GLOBAL directive allows a symbol to be defined which will be treated as either an XDEF or an XREF. If a symbol is defined as GLOBAL and is later defined as a label, it will be treated as an XDEF. If the symbol is never defined, it will be treated as an XREF.

Syntax

GLOBAL *symbol[,symbol]*

Remarks

This is useful in header files because it allows all separately assembled modules to share one header file, defining all global symbols. Any of these symbols later defined in a module will be XDEFed, the others will be treated as XREFs.

See also: XREF, XDEF and PUBLIC

Chapter 14:
The Librarian

Overview

If the Linker cannot find a symbol in the files produced by the Assembler, it can be instructed, by a Linker command line option to search one or more object module Library files.

This chapter discusses Library usage and the PSYLIB library maintenance program in the following sections:

- PSYLIB Command Line Syntax
- Using the Library Feature

PSYLIB Command Line Syntax

Description

The Library program, `PSYLIB.EXE`, adds to, deletes from, lists and updates libraries of object modules.

Syntax

PSYLIB */switches library module...module*

where switches are preceded by a forward slash (/), and separated by commas.

Switches

`/a`	Add the specified modules to the library
`/d`	Delete the specified module from the library
`/l`	List the modules contained in the library
`/u`	Update the specified modules in the library
`/x`	Extract the specified modules from the library

Arguments

Library	The name of the file to contain the object module library
Module list	The object modules involved in the library maintenance

See also: PSYLINK

Using the Library Feature

To incorporate a Library at link time, specify a library file on the Linker command line (see Chapter 13).

If the Linker locates the required external symbol in a nominated library file, the module is extracted and linked with the object code output by the Assembler.

When using a C compiler, the `-c` option (Note lowercase c) on the CCPSX command line will force the compiler to output only `.OBJ` files, these can then be added to a library.

Chapter 15:
The PSYMAKE Utility

Overview

PSYMAKE is a make utility for MS-DOS which automates the building and rebuilding of computer programs. It is general purpose and not limited to use with the SDevTC system. The utility is discussed under the following headings:

- PSYMAKE Command Line Syntax
- Format of the Makefile

PSYMAKE Command Line Syntax

Description

PSYMAKE rebuilds only those components of a system that need rebuilding. Whether a program needs rebuilding is determined by the file date stamps of the target file and the source files that it depends on. Generally, if any of the source files are newer than the target file the target file will be rebuilt.

Syntax

PSYMAKE *[options] [target]*

Remarks

Valid options are:

`/b`	Build all, ignoring dates
`/d` *name=string*	Define name as string
`/f` *filename*	Specify the MAKE file
`/i`	Always ignore error status
`/q`	Quiet mode: do not print commands before executing them
`/x`	Do not execute commands, just print them

If no `/f` option is specified, the default makefile is `MAKEFILE.MAK`. If no extension is specified on the makefile name, `.MAK` will be assumed.

If no target is specified the first target defined in the makefile will be built.

Format of the Makefile

The Makefile consists of a series of commands, governed by explicit rules known as dependencies, and implicit rules. When a target file needs to be built, PSYMAKE will first search for a dependency rule for that specific file. If none can be found, PSYMAKE will use an implicit rule to build the target file.

Dependencies

A dependency is constructed as follows:

```
targetfile:                [sourcefiles]
            [command
            ...
            command]
```

The first line instructs PSYMAKE that the file “targetfile” depends on the files listed as “sourcefiles”.

If any of the source files are dated later than the target file, or the target file does not exist, PSYMAKE will issue the commands that follow in order to rebuild the target file.

If no source files are specified, the target file will always be rebuilt.

If any of the source files do not exist, PSYMAKE will attempt to build them first, before issuing the commands to build the current target file. If PSYMAKE cannot find any rules defining how to build a required file, it will stop and report an error.

The target file name must start in the left hand column. The commands to be executed in order to build the target must all be preceded by white space (either space or tab characters). The list of commands ends at the next line encountered with a character in the leftmost column.

Examples

```
main.cpe:     main.s inc1.h inc2.h
       ASMPSX main,main
```

This tells PSYMAKE that main.cpe depends on the files `main.s`, `inc1.h` and `inc2.h`. If any of these files are dated later than `main.cpe`, or `main.cpe` does not exist, the command “ASMPSX main,main” will be executed in order to create or update `main.cpe`.

```
main.cpe:     main.s inc1.h inc2.h
       ASMPSX /l main,main,main
       psylink main,main
```

Here, two commands are required in order to rebuild main.cpe.

Implicit Rules

If no commands are specified, PSYMAKE will search for an implicit rule to determine how to build the target file. An implicit rule is a general rule stating how to derive files of one type from another type, for instance, how to convert `.ASM` files into `.EXE` files.

Implicit rules take the form:

```
.<source extension>.<target extension>:
       command
       [...
       command ]
```

Each <extension> is a 1, 2 or 3 character sequence specifying the DOS file extension for a particular class of files.

At least one command must be specified.

Example

```
.s.bin:
       asmpsx /p $*,$*
```

This states that to create a file of type .bin from a file of type .s, the ASMPSX command should be executed. (See below for an explanation of the `$*` substitutions.)

Executing Commands

Once the commands to execute have been determined, PSYMAKE will search for and invoke the command. The search order is current directory then directories in the path.

If the command cannot be found as an `A.EXE` or `A.COM file` or the command is `A.BAT` file, PSYMAKE will invoke `COMMAND.COM` to execute the command/batch file. This enables commands like CD and DEL to be used.

Command Prefixes

The commands in a dependency or implicit rule command list may optionally be prefixed with the following qualifiers:

`@`	Suppress printing of command before execution
`-level`	Abort if exit status exceeds specified level
`-`	(Without number) ignore exit status (never abort)

Normally, unless `/q` is specified on the command line, PSYMAKE will print a command before executing it. If the command is prefixed by `@`, it will not be printed.

If a command is prefixed with a hyphen followed by a number, PSYMAKE will abort if the command returns an error code greater than the specified number.

If a command is prefixed with a hyphen without a number, PSYMAKE will not abort if the command returns an error code.

If neither a hyphen or a hyphen and number is specified and `/i` is not specified on the command line, PSYMAKE will abort if the command returns an error code other than 0.

Macros

A macro is a symbolic name which is equated to a piece of text. A reference to that name can then be made and will be expanded to the assigned text. Macros take the form:

name = text

The text of the macro starts at the first non-blank character after the equals sign (=) and ends at the end of the line. Note that:

- Case is significant in macro names
- Macro names may be redefined at any point
- If a macro definition refers to another macro, expansion takes place at time of usage
- A macro used in a rule is expanded immediately
- To invoke a macro its name must be enclosed within $ (and). For example $ (flags) for the variable flags

Example

```
FLAGS       =           /p /s
...
.s.bin:
       ASMPSX $ (FLAGS) $*,$*
```

The $ (FLAGS) in the ASMPSX command will be replaced with `/p /s`.

Predefined Macros

The following predefined macros all begin with a dollar sign and are intended to aid file usage:

`$d` Defined test macro, e.g.:
```
!if $d (MODEL)
```
if MODEL is defined

`$*` Base file name with path, e.g.:
```
C:\PSYQ\TEST
```

`$<<` Full file name with path, e.g.:
```
C:\PSYQ\TEST.S
```

`$:` Path only, e.g.:
`C:\PSYQ`

`$.` Full file name, no path, e.g.:
`TEST.S`

`$&` Base file name, no path, e.g.:
`TEST`

The filename predefined macros can only be used in command lists of dependency and implicit rules.

Directives

The following directives are available:

`!if` *expression*

`!elseif` *expression*

`!else`

`!endif`

These directives allow conditional processing of the text between the if, elseif, else and the closing endif. Any non-zero expression is True, zero is False.

`!error` *message* Print the message and stop.

`!undef` *macroname* Undefines a macro name.

Expressions

Expressions are evaluated to 32 bits, and consist of the following components:

Decimal Constants	e.g. 1 10 1234
Hexadecimal	e.g. $FF00 $123abc
Monadics	- ~ !
Dyadics	+ - * / % > < & | ^ && || > < >= <= == (or =) != (or <>)

The operators have the same meaning as they do in the C language, except for = and <>, which have been added for convenience.

Value Assignment

Macro names can be assigned a calculated value; for instance:

```
NUMFILES == $ (NUMFILES)+1
```

(Note that there are two equals signs in the value assignment)

This evaluates the right hand side, converts it to a decimal ascii string and assigns the result to the name on the left.

In the above example, if NUMFILES was currently “42”, it will now be “43”.

Note that:

```
$ (NUMFILES)+1
```

would have resulted in NUMFILES becoming “42+1”.

Undefined macro names convert to ‘0’ in expressions and null string elsewhere.

Comments

Comments are introduced by a hash mark (#):

```
main.exe:    main.asm     # main.exe only depends
                          # on main.asm
# whole line comment
```

Line Continuation

A command too long to fit on one line may be continued on the next by making '\' the last character on the line, with no following spaces or tabs:

```
main.exe:    main.asm i1.h i2.h \
       i3.h i4.h
```

Chapter 16:
SDevTC Debugger for Windows 95

Overview

The SDevTC Debugger for Windows 95 takes advantage of the new range of 32-bit operating systems available for PCs; providing full source level as well as traditional symbolic debugging and supporting and enhancing all the power of the DOS-based version plus the advantages of a multi-tasking GUI environment.

It helps you to detect, diagnose and correct errors in your programs via the step and trace facilities, with which you can examine local and global variables, registers and memory.

Breakpoints can be set wherever you need them at C and Assembler level. If required, these breaks can be made conditional on an expression. Additionally, selected breakpoints can be disabled for particular runs.

The Debugger employs drop-down menus, tool buttons, keyboard shortcuts and pop-up menus to help you debug quickly and intuitively.

Projects

The Debugger uses Projects to group together details of Files, Targets, Units, Views and other settings and preferences. All this information is saved and made available for your next debugging session.

Views

The Debugger offers the functionality of splitting the screen into a number of Panes, each displaying discrete or linked information. This information is available within a View, or document window (MDI Child). Each View can be split horizontally or vertically into the number of Panes you require and each Pane can be set to show a specific type of information.

You can have as many combinations of either tiled Panes or overlapping Views as you choose.

Your choice of Views depends on the level at which you are debugging. For example, it is appropriate to use a Register Pane for assembler debugging and a Local Pane when debugging in C.

Individual Views can be saved on disk for subsequent use in other Projects. However, when you close the Debugger and then re-start a session, your previous screen set-up will initially be displayed automatically.

Color Schemes

To aid identification, a **separate** color scheme can be allocated to the Views used by each Unit that you reference. Alternatively, the same color can be allocated to **all** Views.

Files

The Symbol Files you require are located and loaded by the Debugger and the relevant CPE and Binary Files are downloaded to the Target. Where a multi-unit system is in use you must also specify the Unit where Symbol and Binary Files are to be loaded.

Dynamic Update

Changes in memory are highlighted on each display update, showing which areas of memory are being altered as the Target is being run and you are stepping and tracing your code.

Chapter Contents

The following topics are discussed in this chapter:

- On-line Help
- Installing the Debugger
- Launching the Debugger
- The SDevTC File Server
- Connecting the Target and Unit
- SdevTC Project Management
- SDevTC Debugger Productivity Features
- SdevTC Views
- Working with Panes
- Debugging Your Program
- Closing the Debugger

On-line Help Available For The Debugger

Help text describing the features covered in this chapter can also be accessed on-line via the Help menu on the main menu.

Selecting these options will result in the following:

- **Contents** will display the Contents page of the help system in the left-hand side of the screen. Clicking any of the underlined topics will provide further information about the relevant subject.
- **Pane Types** and the required **Pane** will directly access relevant text for the chosen pane.
- **Installation** will display installation procedures.
- **About** will provide the version number.

Within the on-line help system, clicking text with a **dotted** underline will display a pop-up description while double-clicking text with a **solid** underline will display another (linked) help page.

The buttons at the top of the help text window can be used to facilitate the following:

- Search and/or Find to locate a particular word or topic.
- Back to re-display the previous page.
- << and >> to display the previous and next page in the browse sequence, as outlined in the Table Of Contents. (See below).
- Glossary to display an alphabetic listing of terms found in the help system. Click on any topic to obtain a pop-up definition.

As well as accessing information via the Contents page, on-line help can also be located via the Table Of Contents in the right-hand area of the screen. This represents the subject areas of the help system as book icons. Double-click any icon to display titles of the individual pages which compose each ‘book’. Double-click any of these pages and the text will be displayed in the left-hand side of the screen.

Installing The Debugger

A Set-up program is used to install the Debugger. This is distributed via either of the following methods:

- Full Release Files
- Maintenance Patch Files

Both methods are described in more detail below the Directory Structure.

Directory Structure

All the Files relating to the Windows software live in one directory tree. This tree can reside anywhere but it is probably easier to locate it on the root of a local drive.

The default directory name is:

```
C:\PsyQ_Win\
```

It is *recommended* that you follow this convention.

Set-up also installs several Files in the Windows System directory and adds two keys to the Registry.These keys are:

- [HKEY_LOCAL_MACHINE\SOFTWARE\SN Systems] (hardware settings)
- [HKEY_CURRENT_USER\Software\SN Systems] (configuration information)

Set-up also registers the File types **.psy** (SDevTC Project), **.pqp** (SDevTC patch) and **.cpe,** and adds some programs to the Start menu.

IMPORTANT: Do not install the program on a server and execute it across a network. For un-installation advice, please contact SN Systems.

Obtaining Releases and Patches

Releases and patches are available directly from SN Systems' BBS and ftp sites. In order to access these sites you will need an account with the necessary permissions.

To apply for an account, telephone SN Systems or contact them via Support@snsys.com. Patches and releases can also be obtained via email in MIME, provided that you are a member of the Windows-Users mailing list. (See below).

Note: Members of the Windows-Users mailing list will be notified of releases and patches as they become available.

Determining the Latest Releases and Patches

This is achieved via any of the following methods:

- Contact John@snsys.com.
- Look in one of the File sites for the latest Files and information.
- Send mail to the auto-responder maildrop - Versions@snsys.com.

Mailing Lists

SN Systems maintains the following mailing lists:

- Announce@snsys.com - For all announcements regarding SDevTC
- Windows-Users@snsys.com - For up-to-date information
- Windows-Discuss@snsys.com - For all Debugger users (discussion)

The first two are read-only and provide details about revisions and other information. The third is an open read/write list which hosts any Debugger related discussions, problems, suggestions or comments.

For more information on these lists, send a HELP message to Norman@snsys.com (the Robot List Manager) or John@snsys.com.

Addresses for SN Systems' ftp, web and BBS sites

- ftp://ftp.snsys.com
- http://www.snsys.com
- BBS - +44 (0)117 9299 796 and +44 (0)117 9299 798

Beta Test Scheme

SN Systems maintains a separate scheme for beta testing new versions of the Debugger. The benefits of this are as follows:

- You will receive new versions of the Debugger before any other user.
- You will have a prioritized chance to supply feedback to the Debugger's authors.

If you are a member of this scheme, you don't need to install release versions of the Debugger.

For more information, contact John@snsys.com.

Installing a Full Release

A Full Release File contains an archive of several Files and a Set-up program that can be used to install the Debugger automatically.

To install the release:

1. Obtain the latest full release from SN Systems.
2. Read **Readme.txt** which contains last-minute installation instructions.
3. If the release is on a floppy, launch **Setup.exe** straight-away. If however, the release is in a zip File, you must unzip the File into a temporary directory and then launch Setup from that temporary directory.
4. If this is the first full installation of the Debugger, confirm the displayed license conditions.
5. Specify or confirm the directory in which you wish to install the Debugger.
6. The Files will be installed and the Registry will be updated.
7. Depending on the type of installation, specify the settings for the DEX Board or SCSI Card. (See **Configuring Your Dex Board/SCSI Card** below).

Once the dialog has been completed the installation is complete.

Note: This method can be used for the first installation of the Debugger and also for subsequent upgrades if you do not wish to use Maintenance Patches. See **Upgrading Your System** below.

Upgrading Your System

From time to time, SN Systems will provide updates to the Debugger that introduce bug fixes and new features. For your convenience, updates are supplied as full installations **and** as maintenance patches.

A Maintenance Patch contains only the difference between Files so it is much smaller. This makes it quicker to download and apply. However, patches can only be applied over certain previous versions.

To apply a Maintenance Patch:

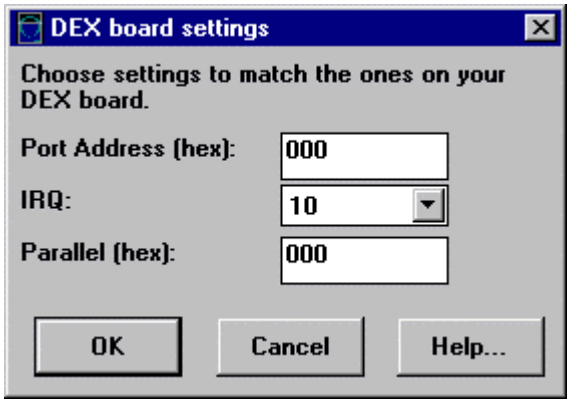
1. Determine your current release by reading the **About** box for the Debugger.
2. Obtain the Maintenance Patch from SN Systems. Instructions will be provided so you can determine which patch must be applied.
3. Apply the patch by using the PQSetup program. This is available on the Start menu or by double-clicking a patch file (.pqp). Follow the on-line instructions.

Configuring Your Dex Boards

If you are installing a Full Release for the PlayStation (DEX only), you must specify the settings for these DEX Boards.

Enter appropriate values to the dialog box displayed during the Set-up program.

Figure 16-1: DEX Board Settings Dialog Box



1. Enter a 3 or 4-digit hexadecimal number to the Port Address and Parallel boxes and specify an IRQ value by clicking on the down arrow and selecting as appropriate.
2. Click OK.

The installation is now complete.

IMPORTANT: Port Address and IRQ values must be correct for the Debugger to work. If they are incorrect or another device is configured to use similar settings, the programs will not work.

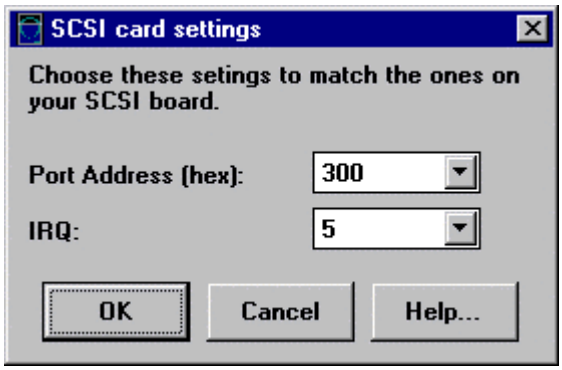
Note: The Parallel setting should be set to the I/O address of the port to which your dongle is connected. Most PCs use 378 for 1pt1. If your dongle is on a different parallel port or your PC uses a non-standard port address, change this value. See FAQ.DOC for more information.

Configuring Your SCSI Card

If you are installing a Full Release for a SCSI Target, you must specify the settings for the SCSI Card.

Enter appropriate values to the dialog box displayed during the Set-up program.

Figure 16-2: SCSI Card Settings Dialog Box



1. Specify a Port Address and IRQ value by clicking on the down arrows and selecting as appropriate.
2. Click OK.

The installation is now complete.

IMPORTANT: Port Address and IRQ values must be correct for the Debugger to work. If they are incorrect or another device is configured to use similar settings, the programs will not work.

Note: The IRQ value can be set to 0 to run without interrupts. However, this is only recommended for troubleshooting since running without interrupts will seriously impair the performance of the system.

Testing the Installation

Once the Debugger has been installed, you should now run PsyServe.exe in order to test that the configuration is working correctly.

A message similar to the following should be displayed:

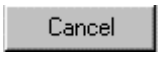
```
Psy-Q File and Message Server, Copyright 1995, SN Systems Ltd,
Version: 1.00 (December 1995)

Target: Sony PlayStation Plug-In
Resources:  Port=0x390, Interrupt=12

Loading SCSI Drivers...
Connected to SONY_PSX5.15
Ready to Serve
```

If no message appears at all, your Port and IRQ settings may be incorrect or there may be a resource conflict with some other device.

However, you can change the Port/IRQ settings by re-running the Set-up program as follows:

1. Click 

on the Open File dialog box.
2. Select the appropriate Card from the Cards menu.

You will then be presented with the same dialog box as was displayed during installation.

See FAQ.DOC if you continue to have problems.

Documentation

If you experience problems during installation, the following documents provide useful information:

- README.DOC details how to obtain and apply maintenance releases
- FAQ.DOC contains Frequently Asked Questions and should be consulted if you experience any problems
- BUG.TXT describes how to report bugs
- TODO.TXT lists known bugs, problems and features that are not yet incorporated into the Debugger

Launching The Debugger

There are several ways of launching the SDevTC Debugger under Windows 95.

A simple way is as follows:

1. Select the Start menu from Windows 95.
2. Choose the Programs option from the list displayed.
3. Select the Psy-Q folder from the list of programs.
4. Select Psy-Q Debugger from the folder.

You can also launch the Debugger from the desktop or folders or through Explorer in Windows 95.

With the drag and drop facility you can drop an SDevTC Debugger Project File (extension .PSY) onto the icon of the Debugger and the selected Project is launched.

Alternatively, as file type .PSY has been registered with the Windows 95 shell, you can right-click on a Project File, select Debug from the menu and the Debugger will be launched with the selected Project.

Note: While the Debugger is still running, you can open a new Project by following the procedure described in the previous paragraph.

When you launch the Debugger it scans for recognized Units. If none are found, a dialog box prompts you to either Repoll or Quit. If the latest downloader has not been installed, you are prompted to download this. The SDevTC File server is automatically launched with the Debugger.

See Also: Launching the File Server without the Debugger

The SDevTC File Server

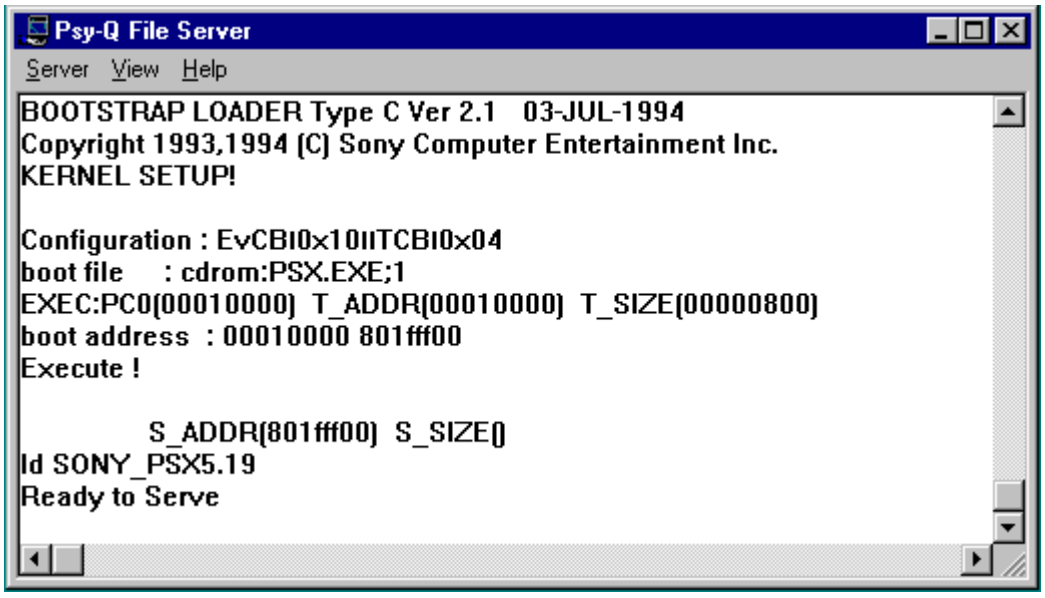
The primary function of the SDevTC File Server is to provide the PC Open and PC Read functions for your program.

It is always launched when the Debugger is launched and must always be running in the background while the Debugger is being used, both file serving and collecting messages.

When the File Server is running, the icon and name of the application appear on the Task bar of Windows 95.

You can view the messages appended into the message window of the File Server during debugging by clicking on this icon.

Figure 16-3: File Server Message Window



If you wish the message window to be permanently displayed on top of other windows, select Always on Top from the View menu.

When the Debugger or File server experiences a communication error, a dialog box will display a relevant error code and a Retry and Cancel button.

Figure 16-4: Communication Error Dialog Box

Press Retry and the attempted connection will be repeated. Press Cancel and the system will try to carry on and will attempt to recover from the error.

Note: If the Debugger comes up with this message, the File Server can still be used to reset the Target.

File Server Menu Commands

In addition, seven menu commands can be used with the File server:

- Run Project loads all Files (except Symbol Files) that are set to 'download on project startup' and runs the Project without loading the Debugger.
- Download CPE downloads the cpe file to the Target.
- Run CPE runs the Target after the cpe file has been downloaded.
- Ping determines the current status of the Target.
- Halt provides the option to stop the Target if it is running.
- Clear Window removes any File Server messages.
- Reset Target

Note: Resetting the Target while the Debugger is running may cause unpredictable results.

Note: The Reset option is also available from the System menu of the File server.

IMPORTANT: The SDevTC File Server must always be running when the Debugger is running. You will not be permitted to stop the server until you close the Debugger. However, the File Server can be run without the Debugger.

Launching the File Server without the Debugger

If you wish to launch the File Server independently of the Debugger, for example, to run your Project without loading the Debugger:

1. Select the Start menu of Windows 95.
2. Choose the Programs option from the list displayed.
3. Select the Psy-Q folder from the list of programs.
4. Choose the Psy-Q File Server option from the folder.

Note: When you launch the File server the Target is automatically reset, whether the Debugger is running or not.

See Also: Launching the Debugger

Connecting The Target and Unit

The SDevTC Debugger automatically checks your system when you launch it, identifies any Targets that are connected and, according to whether you are running a single or multi-unit system, automatically connects to the relevant unit(s).

The Unit toolbar appears at the bottom of the Debugger window directly above the Status line. The first icon in the toolbar has a pictogram of the Target known as the Unit button.

Figure 16-5: Unit Button

There will be a Unit toolbar and unique button for each unit identified. Click on the button to display the unit menu. This menu allows you to download and load (as relevant) foreign CPE and Symbol Files and non-foreign CPE and Binary Files. The menu also allows you to see and edit breakpoints.

The menu options are:

- Download CPE
- Download Binary
- Load Symbols
- Breakpoints

Each toolbar contains a set of debugging icons which represent:

- Starting programs
- Stopping a program running
- Stepping into a subroutine
- Stepping over a subroutine

Note: These actions operate only with respect to the relevant unit. Therefore, when a multi-unit system is in use, they will not necessarily operate with respect to the Active View.

Note: The SDevTC File server is automatically launched when the SDevTC Debugger is started. The Server window displays any output from the Target while it is running.

SDevTC Project Management

An SDevTC Project is a combination of the elements and settings associated with a specific development project. It consists of any or all of the following:

- Units to be debugged
- Screen layout
- CPE Files
- Symbol Files
- Binary Files
- Breakpoints
- Other settings and preferences.

This set of information is used by the Debugger to track the debugging process. When you save a Project this includes all the Views, color schemes and breakpoints already specified for it. These settings are reinstated when the Project is next opened.

Setting up and Managing Projects

To create a new Project you can either:

1. Open the default SDevTC Project by selecting New from the Project menu.
2. Save and name the Project.

or

1. As 1) above.
2. Select files for the Project and add them to the file list.
3. Set file properties for executable files.
4. Save and name the Project
5. Re-open the Project with the files in the file list.

Selecting Files for Your Project

The SDevTC Debugger uses files that are output from the build process. Three types of file may be included in the Project. These are:

- CPE Executable Files
- Symbol Files
- Binary Files

Adding Files to The List of Project Files

This is achieved as follows:

1. Select the Project menu from the Menu bar.
2. Choose Files from the menu; the Files dialog appears.
3. Click Add to insert them into your file list.
4. Select CPE, Binary or Symbol Files from the 'Files of Type' drop-down list.
5. Locate the file and click OK.
6. When you add a file to the file list, a relevant dialog box requests you to set the file properties. For CPE and Binary Files these will determine the downloading of files to the Target. Additionally, for Binary and Symbol Files, they determine the unit to which they will be loaded. See the 'Understanding File Properties' sections below.

   Note: It is not necessary to specify the unit to which a CPE File should be loaded as this information is held within the file itself.
7. Repeat the operation until all the files you require appear in the list. To remove a file from the list, highlight it and click Remove.
8. Click Close when you have added all the files you require.

The CPE and Binary Files will be downloaded in the order shown in the file list.

Note: As file type .CPE has been registered with the Windows 95 shell, you can run a program directly from the shell by double-clicking on the relevant CPE File. Alternatively, if you wish to download the file to the Target without running it, right-click the relevant file and select Download from the menu.

Note: When you add Binary and Symbol Files to a Project, they are not loaded until the Project is saved and re-opened.

Changing the Order of Files in the File List

If you have multiple CPE and Binary Files within your Project, the order in which they are loaded during debugging is determined by the positions you placed them in the File list.

To change the file sequence:

1. Select the Project menu from the Menu bar.
2. Choose Files from the menu.
3. Highlight a file.
4. Use Promote or Demote to alter the position of the file in the list.

Repeat the process until the files are in the required order.

Note: This option is only useful if you have multiple CPE and Binary Files in your Project and the load order is important.

Specifying CPE File Properties

When you select a CPE File to include in your Project, a dialog box requests that you set the properties for this file.

These properties allow you to control the downloading of files to the Target. The options are:

- Download when Project starts . This causes the CPE File to be downloaded when the Project is opened or reopened.
- Run after CPE has been downloaded . This causes the Unit to start running the code after downloading the file.

You may select either or both of these properties for any CPE File in the Project.

If you do not set the properties of at least one CPE File, the Debugger will not download any files to the Target when the Project is opened.

To change CPE File properties:

1. Select the Project menu from the Menu bar.
2. Choose Files from the menu.
3. Select the CPE File to change.
4. Click Properties....
5. Use the check boxes to apply the properties.
6. Click Close.

Specifying Symbol File Properties

When you select a Symbol File to include in your Project, a dialog box requests that you confirm or specify the unit to which the file should be loaded.

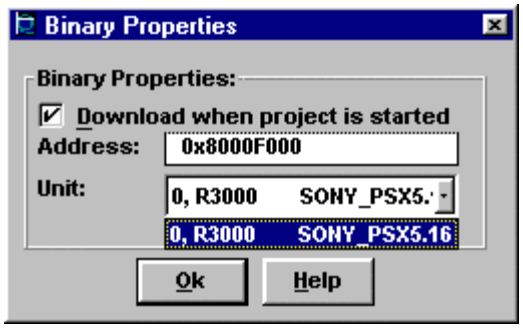
To change Symbol File properties:

1. If the required unit is not already displayed, click the down arrow until it appears.
2. Highlight the required unit.
3. Click OK.
4. Click Close.

Specifying Binary File Properties

When you select a Binary File to include in your Project, you must complete the following dialog box:

Figure 16-6: Binary File Properties Dialog Box



These properties allow you to control the downloading of files to the Target:

- Download when Project starts. If this is selected the Binary File will be downloaded when the Project is opened or reopened.
- Downloaded to a specified address. The files will be downloaded to the address specified. This should be in OX notation for hexadecimal numbers. The default address will be zero.

Specify the Unit where the File is to be loaded. Click on the down arrow to display further units.

If you do not set the first option for at least one Binary File, the File Server will not download any Binary Files to the Target when the Project is opened. However, all Binary Files in the Project will be available on the relevant unit menu.

To change Binary File properties:

1. Select the Project menu from the Menu bar.
2. Choose Files from the menu.
3. Select the Binary File to change and click 

.
4. Select the Download when Project starts option if required and/or enter a relevant address .
5. Confirm or specify the unit where the file is to be loaded.
6. Click 

.

Saving Your Project

Once the files have been selected, the new Project must be saved and re-loaded before debugging can begin.

This is achieved as follows:

1. Select the Project menu from the Menu bar.
2. Choose the Save option from the menu.
3. Give a name and path to your Project.
   File names in Windows 95 are up to 250 characters long and can contain spaces.
   SDevTC Debugger Project Files must be saved with the default file extension of .PSY.
4. Click 

.

Note: For a new Project you can choose the Restore rather than the Save option. Restore prompts you to save the Project before reloading it.

Note: The Save or Save As options can be used to save an existing Project.

Re-opening A Project

After saving a new Project you must re-open it before working with the files which have been added to the file list.

This is achieved as follows:

1. Select the Project menu from the Menu bar.
2. Choose the Re-open option from the menu.

Note: The Re-open icon on the toolbar  can also be used to re-open a Project.

Saving a Project under a New Name

The Save As option on the Project menu is used to save changes made to an existing Project, under a new name.

The default file extension for an SDevTC Debugger Project is .PSY. When you save Project Files you must use this extension.

To save a Project under a new name:

1. Select the Project menu from the Menu bar.
2. Choose the Save As option from the menu.
3. Give a name and path to the renamed Project.
4. Click Save.

Restoring a Project

The Restore option on the Project menu is used to re-load a Project in the state in which it was last saved, abandoning any changes made since the last save.

To restore a Project:

1. Select the Project menu from the Menu bar.
2. Choose the Restore option from the menu.

Opening an Existing Project

When you launch the SDevTC Debugger, the last Project you worked on will be loaded automatically.

To open a different Project:

1. Select the Project menu from the Menu bar.
2. Choose the Open option from the menu.
3. Select the Project (.PSY) you require.
4. Click Open.

Note: An existing Project can also be opened via the Open Project icon  found on the toolbar.

Note: As file type .PSY has been registered with the Windows 95 shell, you can run a Project by double-clicking on the relevant .PSY file within the shell. Alternatively, if you only wish to load the Project into the Debugger, right-click the relevant file and select Debug from the menu.

Manually Loading Files into a Project

External Files can be downloaded at any time. They are not saved with the Project.

External CPE Files are downloaded to the Target as follows:

1. Click on the Unit menu at the base of the Debugger screen.
2. Choose the Download CPE option from the menu.
3. Choose the External File option.
4. Browse and select the required CPE File.
5. Click OK.

Note: You can also download a CPE File by double/clicking it within the shell.

Symbol Files can be loaded into the Debugger as follows:

1. Click on the relevant Unit menu at the base of the Debugger window.
2. Choose the Load Symbols option from the menu.
3. Browse and select the required Symbol File.
4. Click OK.

SDevTC Debugger Productivity Features

To enable you to work faster and more efficiently when using the SDevTC Debugger, the following two features speed up your control of the debugging runs.

- Toolbar Icons
- Hot Keys

Toolbar Icons

The toolbar contains the group of icons shown above. Icons provide a quicker means of activating commands and setting properties.

From left to right they represent the following actions:

- Open a Project File
- Save and then reopen the current Project
- Open a new View
- Switch to the next View
- Split the Active Pane horizontally
- Split the Active Pane vertically
- Delete the Active Pane
- Set the default color scheme

The Show Toolbar option on the Project menu is used to toggle the menu bar on and off. When the option is ticked the toolbar is displayed.

To toggle the toolbar:

1. Select the Project menu from the Menu bar.
2. Choose the Show Toolbar / Hide Toolbar option from the menu.

Note: Every Pane type has its own, additional toolbar which is appended to the main toolbar when that Pane is made Active.

Hot Keys

The following Hot Keys can be used instead of the Debugger menu options:

Table 16-1: Debugger Hot Keys

F2	Split horizontal
F3	Split vertical
F4	Delete current pane
F5	Toggle breakpoint on and off
F6	Run to cursor
F7	Step into a subroutine
F8	Step over a subroutine

F9	Run a program
Esc	Halt a program
Ctl+Shft+D	Change pane to Disassembly Pane
Ctl+Shft+L	Change pane to Local Pane
Ctl+Shft+M	Change pane to Memory Pane
Ctl+Shft+R	Change pane to Register Pane
Ctl+Shft+S	Change pane to Source Pane
Ctl+Shft+W	Change pane to Watch Pane
Shft+Arrow Keys	Activates adjacent pane in the specified direction. Where more than one, the current caret position determines the pane to be made active.
Ins	New view

Note: These keys will all operate within the Active Pane.

SDevTC Views

A View appears in the main window of the SDevTC Debugger. It is used to display debugging information according to your requirements and to control step and trace actions during debugging.

When an SDevTC Project is first created it has a default pane layout.

Views can be split into as many panes as you wish. These can be of the same or different types.

Only one pane is Active at any time; it will be displayed in a color scheme different from the others.

Note: Having created a View of different panes you can save this as a View File either in, or independent of, the Project. Further information about panes can be found in Working With Panes and Selecting A Pane Type .

Creating an SDevTC View

Within an SDevTC Project you can create as many Views as required. In turn, each View can be split into as many panes as you need.

When you open a new Project, one View is displayed for each unit connected.

Figure 16-7: Default View

To create a new View:

1. Select the View menu from the Menu bar.
2. Choose the New option from the menu.
3. From the Choose Unit box specify the unit for which you wish to create a new View.

Note: The Choose Unit box will not appear when you are connected to a single unit.

You can also use the New View icon on the toolbar  to create a new view or, you can use the Hot Key Insert .

Alternatively, you can open a new view from the relevant unit button, in which case you won't be prompted for the required unit.

Note: A new view is supplied with the title 'Default View'. The View Name option in the View menu should be used to give it a title.

Note: Views can be saved either inside or outside of SDevTC Projects.

Cycling between Views

If you have more than one view open within a Project you can cycle between them as follows:

1. Select the View menu from the Menu bar.
2. Choose the Next View option.

The views are cycled around until you see the one you require. All views appear on the View list regardless of the units for which they have been specified.

Note: This is not the name of the File. See the note in Saving Your Views above for further details.

Working With Panes

When an SDevTC Project is first set up, the default view contains a default pane layout for each unit connected. However, this view can be split into as many panes as you wish. These can be of the same or different types. Only one of the panes is Active; it will be displayed in a color scheme different from the others.

A pane can be made Active via any of the following methods:

- Clicking on it.

F4 can be used to delete the active pane.

Changing Fonts in Panes

If required, the SetFont command can be used to change the display of text within a pane, as follows:

1. Make the required pane active.
2. Select the View menu from the Menu Bar.
3. Choose Set Font from the menu.

A standard Windows dialog box allows you to select from the available fonts.

Note: When you split a Pane, the new Pane will be displayed in the same font as the original one.

IMPORTANT: You will only be able to use non proportional fonts, e.g. Courier, New Courier, Fixed Sys, Terminal…

See Also: Changing Color Schemes In Views

Scrolling within a Pane

Many panes are unable to display the full set of information that is available to the Debugger in the small screen area shown. Therefore, the Debugger puts scroll bars onto panes where there is more information than can be displayed on that part of the screen.

To see this additional information drag the thumb within the scroll bar or click on the arrows at either end of the scroll bar.

You can also scroll to the region you want by clicking on the desired pane to make it active and then clicking and holding the left mouse button before dragging it to the top or bottom of the pane.

See Also: Changing Pane Sizes

Selecting a Pane Type

There are six types of panes and you may display any number and combination.

A menu that allows you to change pane properties is accessed via the Pane menu on the Menu bar or by right clicking the mouse on a relevant pane. These menus are unique to the type of pane that is active, but all the menus have the option Change Pane that allows you to switch between the different types.

Additionally, icons representing each type of pane appear adjacent to the main toolbar.

- Memory Pane - Displays areas of memory within the Target
- Registers Pane - Displays the registers of the relevant CPU
- Disassembly Pane - Displays the code that the CPU is running
- Source Pane - Displays Source Files associated with program that CPU is running
- Local Pane - Displays local variables
- Watch Pane - Displays ‘watches’ or expressions

Click on the relevant icon to change the active pane.

Note: You can also use the hot keys to switch between pane types.

Icons representing menu options for the selected pane are dynamically appended to the far right of the main tool bar. For example, if a Disassembly Pane is active, Disassembly Pane options will be displayed.

Further details about the options for each type of Pane are below.

Ctl+Shft+R

Using the Pane Type option from the Pane menu

3. Using the hot keyCtl+Shft+D

See Also:

- Anchoring Panes To The PC
- Moving To A Known Address Or Label
- Setting Breakpoints
- Editing Breakpoints

Ctl+Shft+S

See Also:

- Anchoring Panes To The PC
- Moving To A Known Address Or Label
- Setting Breakpoints
- Editing Breakpo ints

When you select the Watch Pane menu or click the right hand mouse button over a Watch Pane, the following menu is displayed:

- Add Watch
- Edit Watch
- Delete Watch
- Clear All Watches
- Expand/Collapse - to view/hide the components of a structure or an array
- Increase Index - to view higher indexed values within an array
- Decrease Index - to view lower indexed values within an array

These options can also be activated by the following methods:

- Using the appropriate hot keys
- Clicking on the appropriate icons

Structures, pointers and arrays can be opened in a Watch Pane.

- If you open a structure the members of that structure are displayed.
- If you open a pointer it is dereferenced.
- If you open an array the first element of the array is displayed.

The contents of the Watch Pane are saved within the View when the Project is saved.

Variables can be viewed in hexadecimal or decimal modes by right-clicking within the Pane and 'toggling' between Hexadecimal/Decimal (on the displayed menu) as required. A tick will appear alongside Hexadecimal when this mode is selected.

Any Watch variable that evaluates to a 'C', l-type expression, can be assigned a new value.

0x80002000 to the specified pointer.

The de-referenced structure changes to reflect the amended value.

IMPORTANT: The expression that you are assigning and the new value, must have compatible types.

Note: Variables can be assigned while the Target is running.

Expanding or Collapsing a Variable

Pointers, structures and arrays are variables which can be expanded or collapsed in Local or Watch Panes when you place the caret over them.

If you expand a pointer , a line will be added below the pointer for the dereferenced pointer. For example, if the pointer is to an integer, the dereferenced pointer will display that integer.

An expanded structure will display all the elements of that structure below it.

For an expanded array the second line of the display will display the first element of the array.

To expand or collapse a variable:

1. Select the Pane menu for the Local or Watch Panes.
2. Choose the Expand or Collapse option from the menu.

When shown in the Watch Pane, expressions which can be expanded or collapsed will be prefixed as follows:

- \+ this indicates an expression that can be expanded
- \- this indicates that the expression is expanded and can be closed.

This is followed by the expression's type and value.

To edit an expression highlight it and press Return.

Multiple Selection below for further details. The status line will display ‘no matches found’ where relevant.

Advanced Symbol Matching

In addition to basic name completion which always completes the symbol at the end of the specified expression, extended name completion can be used to complete a symbol anywhere in the expression, as follows:

1. Enter a partial expression.
2. Place the caret (insertion point) on or at the end of the symbol you wish to complete, and according to the group you wish to search press one of the following key combinations:

   Alt-N - All symbols (Normal)

   Alt-G - Global (Static & external variables)

   Alt-L - Local (Automatic variables in scope)

Forcing an Update

During continual update, the information you see in the Debugger windows won't be updated until the next connection. Therefore, the slower the update rate, the longer it will be before exceptions can be spotted. However, it is possible to force an update by pressing Ctl+U or selecting the Update option from Debug on the main menu.

Setting Breakpoints

Breakpoints can be set in the Source and Disassembly Panes. They can be absolute (i.e. always break) or conditional upon an expression.

They are displayed in the pane as a different colored bar.

To set a breakpoint:

1. Make a Source or Disassembly Pane active.
2. Click on the instruction or line at which you want to set the break.
3. Select the Debug menu from the Menu bar.
4. Choose the Toggle Breakpoints option from the menu.

When you select a breakpoint from the list displayed on the Unit menu, the Edit Breakpoint dialog box shown above appears.

The enabled check box allows you to enable and disable breakpoints. When the check box is set, the breakpoint is enabled. Only enabled breakpoints will be included in a debugging run.

There are two types of breaks:

- Break at Point is a standard break.
- Break if expression is true is a conditional break.

4. Select the type of break you require from the Type box pull-down list.

   Both options display the breakpoint address in the Location box. The location can be overtyped to move a break.

   When the Break if expression is true option is enabled to create a conditional breakpoint, enter a C-like expression or a label in the Expression box.

Esc.

Moving the Program Counter

The program counter (PC) can be set via the Set PC command.

This command moves the program counter to the current caret position.

It is found on the Pane menus for Source and Disassembly Panes and is set as follows:

1. Make a Source or Disassembly Pane active.
2. Place the caret where you wish the PC to move to.
3. Click the right hand mouse button to call the Pane menu.
4. Select the Set PC option from the menu.

With this command, no instructions are executed between the previous and new PC position.

value of the specified expression, note the consequences when you enter a name containing C debug information as well as an Assembler label.

For example, if _ramsize is specified you will be taken to the value of _ramsize, not to where it is defined. This is because the C expression evaluator sees the C definition of ramsize first and then evaluates it. To Goto this address, you must enter either &_ramsize or :_ramsize.

Ctl G.

Expression Evaluation Features

Register Names

Register names can be specified in any dialog box where expressions can be entered. By default, the evaluator looks for C symbols first, so any variables which are the same as register names will be shown instead. If a name is being interpreted as a register it will be prefixed by a '$'.

It is recommended that you use this '$' prefix when entering register names to explicitly tell the evaluator that it is looking at a register.

Note: Registers have a C type of 'int'.

Typecasts and Typedefs

Typecasts can be entered into an expression via the usual C syntax.

If you entered '(int*)$fp' to a Watch Pane you would see the following:

```
+int*(int*)$fp = 0x8000ff00
```

Typecasting also works for structure tags. However, you are not required to enter the keyword 'struct' when casting to a structure tag.

```
-Tester* (Tester*)$fp = 0x807ff88
  -Tester
     +unsigned char* m_pName = 0x00000645
     +unsigned char* mpLongName = 0xFFFFFFFF
```

You can also cast to typedefs; for example, entering '(daddr_t)p' will produce:

```
long (daddr_t)p = 0x00003024
```

Labels

Labels can also be included in a C expression. The evaluator looks for C level information first and then label information. If it finds a label, it will prefix it with a ':'.

It is recommended that you use this prefix when entering labels to explicitly tell the evaluator that it is looking at a label.

Note: Labels have a C type of 'int'.

Functions

If you include a function name in an expression, its value will be the same as its address. It will appear in a Watch window as follows:

```
int ( ) main = (...) (0x80010BFC)
```

Identifying Changed Information

Any changes to variables since the last debugging step are displayed in a color of your choice on all panes except for Disassembly and Source.

This color is set via the Set Default Colour option from the View menu.

See Also:

Changing Color Schemes in Views

Closing the Debugger

Closing the Debugger without Saving Your Changes

The Quit option on the Project menu stops the Debugger running but does NOT save the current Project.

To close the Debugger without saving your changes:

1. Select the Project menu from the Menu bar.
2. Choose the Quit option from the menu.

Closing the Debugger and Saving Your Changes

The Exit option on the Project menu saves the current Project at the latest state and stops the Debugger running.

To close the Debugger and save your changes:

1. Select the Project menu from the Menu bar.
2. Choose the Exit option from the menu.

Note: It is also possible to close the Project by clicking on the X icon on the system menu shown in the top right corner of the Debugger window.

Note: Next time you open the Debugger, the last project you saved will be launched automatically.

See Also:

Saving Your Project

Appendix A:
Error Messages

Overview

This appendix documents SDevTC error messages and is divided into the following sections:

Assembler Error Messages
PSYLINK Error Messages
PSYLIB Error Messages

Error Message Format

In the list below, `%x` represents the variable part of the error message, as follows:

`%s` is replaced by a string
`%c` is replaced by a single character
`%d` is replaced by a 16-bit decimal number
`%l` is replaced by a 32-bit decimal number
`%h` is replaced by a 16-bit hexadecimal number
`%n` is replaced by a symbol name
`%t` is replaced by a symbol type, e.g. section, symbol or group.

Assembler Error Messages

`‘%n’ cannot be used in an expression`

%n will be the name of something like a macro or register

`‘%n’ is not a group`

Group name required

`‘%n’ is not a section`

Section name expected but name %n was found

`Alignment cannot be guaranteed`

Warning of attempt to align that cannot be guaranteed due to the base alignment of the current section

`Alignment’s parameter must be a defined name`

In call to alignment ( ) function

`Assembly failed`

Text of the FAIL statement

`Bad size on opcode`

E.g. attempt to use .b when only .w is allowed

`Branch (%l bytes) is out of range`

Branch too far

`Branch to odd address`

Warning of branch to an odd address

```
Cannot POPP to a local label
```

E.g. POPP @x

```
Cannot purge - name was never defined
Case choice expression cannot be evaluated
```

On case statement

```
Code generated before first section directive
```

Code generating statements appeared before first section directive

```
Could not evaluate XDEF'd symbol
```

XDEF'd symbol was equated to something that could not be evaluated

```
Could not open file '%s'
Datasize has not been specified
```

Must have a DATASIZE before DATA statement

```
Datasize value must be in range 1 to 256
```

DATASIZE statement

```
Decimal number ille radix
```

Specified decimal digit not legal in current radix

```
DEF's parameter must be a name
```

Error in DEF ( ) function reference

```
Division by zero
End of file reached without completion of %s construct
```

E.g. REPT with no ENDR

```
ENDM is illegal outside a macro definition
Error closing file
```

DOS close file call returned an error status

```
Error creating output file
```

Could not open the output file

```
Error creating temporary file
```

Could not create specified temporary file

```
Error in assembler options
Error in expression
```

Similar to syntax error

```
Error in floating point number
```

In IEEE32 / IEEE64 statement

```
Error in register list
```

Error in specification of register list for MOVEM / REG

`Error opening list file`

DOS open returned an error status

`Error reading file`

DOS read call returned an error status

`Error writing list file`

DOS write returned an error status or disk full

`Error writing object file`

DOS write call returned an error or disk is full

`Error writing temporary file`

Disk write error, probably disk full

`Errors during pass 1 - pass 2 aborted`

If pass 1 has errors then pass 2 is not performed

`Expanded input line too long`

1024 chars<M=>

`Expected comma after << >>`

<<...>> bracketed parameter in MACRO call parameter list

```
Expected comma after operand
Expected comma between operands
Expected comma between options
```

In an OPT statement

`Expecting ‘%s’ at this point`

Expecting one of ENDIF/ENDCASE etc. but found another directive

`Expecting ‘+’ or ‘-’ after list command`

In a LIST statement

`Expecting ‘+’ or ‘-’ after option`

In an OPT statement

`Expecting a number after /b option`

On Command line

```
Expecting comma between operands in INSTR
Expecting comma between operands in SUBSTR
Expecting comma or end of line after list
```

In { ... } list

`Expecting ON or OFF after directive`

In PUBLIC statement

`Expecting options after /O`

On Command line

```
Expecting quoted string as operand
Expression must evaluate
```

Must be evaluated now, not on pass 2

```
Fatal error - macro exited with unterminated %s loop
```

End of macro with unterminated WHILE/REPT/DO loop.

Due to the way the assembler works, this must be treated as a fatal error

```
Fatal error - stack underflow - PANIC
Assembler internal error - should never occur!
File name must be quoted
Files may only be specified when producing CPE or pure binary output
```

In FILE attribute of group

```
Forward reference to redefinable symbol statements.
```

The value used in the forward reference was the last value the symbol was set to.

```
Function only available when using sections
Group ‘%n’ is too large (%l bytes)
```

Group exceeds value in SIZE attribute

```
GROUP’s parameter must be a defined name
```

In GROUP ( ) function call

```
GROUPEND’s parameter must be a group name
```

Error in call to GROUPEND ( ) function

```
GROUPORG’s parameter must be a group
```

In call to GROUPORG ( ) function

```
GROUPSIZE’s parameter must be a group name
```

Error in call to GROUPSIZE ( ) function

```
IF does not have matching ENDIF/ENDC
Illegal addressing mode
```

Addressing mode not allowed for current op code

```
Illegal character ‘%c’ (%d) in input
```

Strange (e.g. control) character in input file

```
Illegal character ‘%c’ in opcode field
Illegal digit in suffixed binary number
```

In alternate number form 101b

```
Illegal digit in suffixed decimal number
```

In alternate number form 123d

```
Illegal digit in suffixed hexadecimal number
```

In alternate number form 1abh

```
Illegal group name
Illegal index value in SUBSTR
Illegal label
```

Label in left hand column starts with illegal character

```
Illegal name for macro parameter
```

In macro definition

```
Illegal name in command
```

Target name in ALIAS statement

```
Illegal name in locals list
```

In LOCAL statement

```
Illegal name in XDEF/XREF list
Illegal parameter number
```

Maximum of 32 parameters

```
Illegal section name
Illegal size specifier for absolute address
```

Can only use .w and .l on absolute addressing

```
Illegal start position/length in INCBIN
Illegal use of register equate
```

E.g. using a register equate in an expression

```
Illegal value (%l)
Illegal value (%l) for boundary in CNOP
Illegal value (%l) for offset in CNOP
Illegal value for base in INSTR
Illegal zero length short branch
```

68000 short branches must not be zero offset

```
Initialized data in BSS section
```

BSS sections must be uninitialized

```
Instruction moved to even address
```

Warning that a padding byte was inserted

```
Label ‘%n’ multiply defined
LOCAL can only be used inside a macro
```

LOCAL statement found outside macro

```
Local labels may not be strings
```

@x EQUS ... is illegal

```
Local symbols cannot be XDEF’d/XREF’d
MEXIT illegal outside of macros
Missing ‘ (‘ in function call
Missing ‘)’ after function parameter (s)
Missing ‘)’ after file name
```

In FILE attribute

```
Missing closing bracket in expression
Missing comma in list of case options
```

In =... case selector

```
Missing comma in XDEF/XREF list
MODULE has no corresponding MODEND
Module may not end until macro/loop expansion is complete
```

If a loop / macro call starts inside a module then there must not be a MODEND until the loop / macro call finishes

```
Module must end before end of macro/loop expansion - MODEND inserted
```

A module started inside a loop / macro call must end before the loop / macro call does

```
More than one label specified
```

Only one label per line (can occur when second label does not start in left column but ends in ':')

```
Move workspace command can only be used when downloading
```

In WORKSPACE statement

```
Names declared with local must not start with '%c'
```

In LOCAL statement

```
NARG can only be used inside a macro
```

Use of NARG outside macro

```
NARG's parameter must be a number or a macro parameter name
```

Illegal operand for NARG ( ) function

```
No closing quote on string
No corresponding IF
```

ENDIF/ELSE without IF

```
No corresponding DO
```

UNTIL without DO

```
No corresponding REPT
```

ENDR without REPT

```
No corresponding WHILE
```

ENDW without WHILE

```
No matching CASE statement for ENDCASE
```

ENDCASE without CASE

```
No source file specified
```

No source file on command line

```
Non-binary character following %
Non-hexadecimal character '%c' encountered
```

In HEX statement

```
Non-hexadecimal character starting number
```

Expecting 0-9 or A-F after $

```
Non-numeric value in DATA statement
OBJ cannot be specified when producing linkable output
```

OBJ attribute on group

```
Odd number of nibbles specified
```

In HEX statement

```
OFFSET's parameter must be a defined name
```

Error in OFFSET ( ) function call

```
Old version of %n cannot be purged
```

Only macros can be purged

```
One string equate can only be equated to another
```

Attempt to equate to expression, etc.

```
Only one of /p and /l may be specified
```

On Command line

```
Only one ORG may be specified before SECTION directive
Op-code not recognized
Operand value %d will be sign extended
```

Warning given in 68000 MOVEQ instructions when operand is in the range 128-255 which will be sign extended, giving -1 to -128

```
Option stack is empty
```

POPO without PUSHO

```
Options /l and /p not available when downloading to target
```

On Command line

```
ORG ? can only be used when downloading output
ORG address cannot be specified when producing linkable output
```

No ORG group attributes when producing linkable output

```
ORG cannot be used after SECTION directive
ORG cannot be used when producing linkable output
ORG must be specified before first section directive
```

When using sections only one ORG statement may appear before all section statements (other than as group attributes)

```
Out of memory, Assembler aborting
Out of stack space, possibly due to recursive equates
```

Assemblers stack is full, possible cause is recursive equates, e.g. x equ y+1 , y equ x*2

```
Overflow in DATA value
```

DATA value too big

`Overlay cannot be specified when producing linkable output`

No OVER group attributes when producing linkable output

`Overlay must specify a previously defined group name`

Error in OVER group attribute

`Parameter stack is empty`

POPP encountered but nothing to pop

`POPP must specify a string or undefined name`
`Possible infinite loop in string substitution`

E.g. reference to x where x is defined as x equs x+1

`Previous group was not OBJ'd`

OBJ ( ) attribute specified but previous group had no obj attribute to follow on from

`SDevTC needs DOS version 3.1 or later`
`Purge must specify a macro name`
`Radix must be in range 2 to 16`
`REF's parameter must be a name`
`Error in REF ( ) function reference`
`Register not recognized`

Expecting a register name but did not recognize

`Remainder by zero`

As for division by 0 but for % (remainder)

`Repeat count must not be negative`

REPT statement error

`Replicated text too big`

Text being replicated in a loop must be buffered in memory but this loop was too big to fit

`Resident SCSI drivers not present`

PSYBIOS does not appear to be loaded

`SCSI card not present - assembly aborted`
`SECT's parameter must be a defined name`

Error in SECT ( ) function call

`SECTEND's parameter must be a section name`

Error in call to SECTEND ( ) function

`Section stack is empty`

POPS without PUSHS

`Section was previously in a different group`

Section assigned to a different group on second invocation

`SECTSIZE's parameter must be a section name`

Error in call to SECTSIZE ( ) function

`Seek in output file failed`

DOS seek call returned error status

`Severity value must be in range 0 to 3`

In INFORM statement

`SHIFT can only be used inside a macro`

SHIFT statement outside macro

`Short macro calls in loops/macros must be defined before loop/macro`
`Short macros may not contain labels`
`Size cannot be specified when producing linkable output`

SIZE attribute on group

`Size specified in /b option must be in range 2 to 64`

On command line

`Square root of negative number`
`Statement must have a label`

No label on, for example, EQU op

`STRCMP requires constant strings as parameters`
`String ‘%n’ cannot be shifted`

String specified in SHIFT statement is not a multi-element string (i.e. {...} bracketed) and so cannot be shifted.

`STRLEN’s operand must be a quoted string`
`Symbol ‘%n’ cannot be XDEF’d/XREF’d`
`Symbol ‘%n’ is already XDEF’d/XREF’d`
`Symbol ‘%n’ not defined in this module`

Undefined name encountered

`Syntax error in expression`
`Timed out sending data to target`

Target did not respond

`Too many characters in character constant`

Character constants can be from 1 to 4 characters

`Too many different sections`

There is a maximum of 256 sections

`Too many file names specified`

On command line

`Too many INCLUDE files`

Limit of 512 INCLUDE files

`Too many INCLUDE paths specified`

Too many INCLUDE paths in /j options on command line

`Too many output files specified`

Maximum of 256 output files

`Too many parameters in macro call`

Maximum number of parameters (32) exceeded

`Too much temporary data`

Assembler limit of 16m bytes of temporary data reached

`TYPE's parameter must be a name`

Call of TYPE ( ) function

`Unable to open command file`

From Command line

`Undefined name in command`

Target name in ALIAS statement

`Unexpected case option outside CASE statement`

Found =... statement outside CASE/ENDCASE block

`Unexpected characters at end of Command line`
`Unexpected characters at end of line`

End of line expected but there were more characters encountered (other than comments)

`Unexpected end of line`

Line ended but more input was expected

`Unexpected end of line in macro parameter`
`Unexpected end of line in list parameter`

In { ... } list

`Unexpected MODEND encountered`

MODEND without preceding MODULE

`UNIT can only be specified once`

In UNIT statement

`UNIT cannot be used when producing linkable output`

In UNIT statement

`Unknown option`

In OPT statement

`Unknown option /%c`

Unknown option on Command line

`Unrecognized attribute in GROUP directive`
`Unrecognized optimization switch '%c'`

In OPT statement or Command line

`User pressed Break/Ctrl-C`

Assembly aborted by user

`XDEF'd symbol %n not defined`

Symbol was XDEF'd but never defined

`XDEF/XREF can only be used when producing linkable output`
`Zero length INCBIN`

Warning of zero length INCBIN statement

Psylink Error Messages

`%t %n redefined as section`

New definition of previously defined symbol

`%t '%n' redefined as group`

New definition of previously defined symbol

`%t '%n' redefined as XDEF symbol`

New definition of previously defined symbol

`Attempt to switch section to %t '%n'`

Non-section type symbol referenced in section switch

`Attempt to use %t '%n' as a section in expression`

Section type symbol required

`Branch (%l bytes) is out of range`

68000 branch instruction cannot reach target

`Branch to odd address`

Warning that 68000 branch instruction goes to an odd address

`Code in BSS section '%n'`

BSS type sections should not contain initialized data

`COFF file has incorrect format`

COFF format files are those produced by Sierra C cross compiler, etc.

`Different processor type specified`

Object code is for different processor type than target or attempt was made to link code for different processor types

`Division by zero`
`Error closing file`

DOS close file call returned error status

`Error in /e option`

On Command line

`Error in /o option`

On Command line

`Error in /x option`

On Command line

`Error in command file`
`Error in Linker options`

On Command line

`Error in REGS expression`
`Error reading file %f`

DOS read file call returned error status

`Error writing object file`

DOS write file call returned error status - probably disk full

`Errors during pass 1 - pass 2 aborted`

Pass 2 does not take place if there were errors on Pass 1

`Expecting a decimal or hex number`

`/o` option on Command line

`File %f is in out-of-date format`

File should be rebuilt be reassembling

`File %f is not a valid library file`
`File %f is not in PsyLink file format`
`Group '%n' is too large (%l bytes)`

Group is larger than its size attribute allows

`Group '%n' specified with different attributes`

Different definitions of a group specify different attributes

`Illegal XREF reference to %t '%n'`

Object file defines xref to symbol which cannot be xref'd, e.g. a Section name

`Illegal zero length short branch`

68000 short branches must not have offset of 0

`Multiple run addresses specified`

More than one run address specified

`No source files specified`

No source file on Command line

`Object file made with out-of-date assembler`

File should be rebuilt before reassembling

`Only built in groups can be used when making relocatable output`

When `/r` command line option is used, only the built in groups can be used, i.e. no new group's may be defined

`Operand value %d will be sign extended`

Warning for 68000 MOVEQ instruction that an operand, in range 128 to 255, will be sign extended and end up in range -1 to -128

`Option /p not available when downloading to target`
`Options /p and /r cannot be used together`

On Command line

`ORG ? can only be used when downloading output`
`Out of memory, Linker aborting`

`Previous group was not OBJ'd`

Cannot specify OBJ () attribute if previous group did not have obj attribute

`Reference to %t '%n' in expression`

Use of, e.g. a section name in an expression

`Reference to undefined symbol %h`

There is an internal error in the object file

`Relocatable output cannot be ORG'd`
`Remainder by zero`

`Run-time patch to odd address`

Warning that a run-time longword patch to an odd address will occur which may cause some Amiga systems to crash

`SCSI card not present - linking aborted`

Could not find SCSI card

`SCSI drivers not loaded`

PSYBIOS does not appear to be present

`Section '%n' must be in one of groups code, data or BSS`

When producing Amiga format code

`Section '%n' placed in non-group symbol %h`

There is an internal error in the object file

`Section '%n' placed in non-group symbol '%n'`

An attempt was made to place a section in a non-group type symbol

`Section '%n' placed in two different groups`

Section is placed in different groups

`Section '%n' placed in unknown group symbol %h`

There is an internal error in the object file

`Section ‘%n’ must be in one of groups text, data or BSS`

When producing ST format code

`Specified patch cannot be represented in target’s relocation format`

When producing relocatable code, certain run time relocations are allowed, depending on the target output file format. This error occurs when the type of patch required cannot be represented in the output file format, e.g. patching a byte in the ST file format which allows only longwords to be patched.

`Symbol ‘%n’ multiply defined`

New definition of previously defined symbol

`Symbol ‘%n’ not defined`

Undefined symbol

`Symbol ‘%n’ placed in non-section symbol %h`

There is an internal error in the object file

`Symbol ‘%n’ placed in unknown section symbol %h`

There is an internal error in the object file

`Symbol in COFF format file has unrecognized class`

COFF format files are those produced by Sierra C cross compiler, etc.

`Timed out sending data to target`

Target not responding or offline

`Too many file names specified`

Too many parameters on command line

`Too many modules to link`

Maximum of 256 modules may be linked

`Too many symbols in COFF format file`

COFF format files are those produced by Sierra C cross compiler, etc.

`Unable to open output file`

Could not open specified output file

`Undefined symbol in COFF file patch record`

COFF format files are those produced by Sierra C cross compiler, etc.

`Unit number must be in range 0-127`
`Unknown option /%c`

On Command line

`Unknown processor type ‘%s’`

Could not recognize target processor type

`Unrecognized relocatable output format`

`/r` option on command line

`User pressed Break/Ctrl-C`

Linking aborted by user

`Value (%l) out of range in instruction patch`

Value to be patched in is out of range

`WORKSPACE command can only be used when downloading output`

Psylib Error Messages

`Cannot add module : it already exists`

Module may only appear in a library once

`Could not create object file`

Error creating object file when extracting

`Could not create temporary file`

Error creating temporary file

`Could not open/create`

DOS error opening file

`Error reading library file`

DOS error reading file

`Error writing library file`

DOS error writing file, probably disk full

`Incorrect format in object file`

Error in object file format - rebuild it

`No files matching`

No object files matching the specifications were found

```
No library file specified
No object files specified
No option specified
```

An action option must be specified on the command line

`Unknown option /`

On Command line, option not recognized

Index