SpixTools User's Manual

V6.1 (beta 2.0.28)

Sun Microsystems, Inc.
Palo Alto, California 94303, U.S.A.

Copyright 2005 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303, U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, Sun Microelectronics, the Sun Logo, Solaris, and SunOS are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

U.S. Government approval required when exporting the product.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Govt is subject to restrictions of FAR 52.227-14(g) (2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015 (b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

NAME

sadd − add/subtract spixcounts files

SYNOPSIS

sadd −b bbout prog bbin [[±] bbin]...

DESCRIPTION

The **sadd** command adds or subtracts one or more **spixcounts**(5sh) format files, *bbin*, all corresponding to the program *prog* to produce a new **spixcounts**(5sh) format file *bbout*. Each *bbin* is added by default or if preceded by '+', but subtracted if preceded by '−'.

DIAGNOSTICS

Counter overflows or underflows cause **sadd** to terminate with a diagnostic (which includes the instruction address), but without writing *bbout*.

SEE ALSO

spixcounts(1sh), spixcounts(5sh).

BUGS

Negative instruction counts are not supported.

NAME

sdas − print disassembled code with spixcounts

SYNOPSIS

sdas [−**d**] [−**b** bbin [−**s**[**bi**]]] [±**t**A]... [±**t**[A],[B]]... [program]

DESCRIPTION

The **sdas** command prints a disassembled version of a program *program* (*a.out* by default) on standard output.

The −**d** option causes the (normally unprinted) data segment to be printed.

The −**b** option causes instruction execution counts (taken from the **spixcounts**(5sh) format file *bbin*) to be printed. For conditional branches, the number of times the branch was taken, followed by the number of times not taken, are shown.

The −**si** option causes instructions to be disassembled in order of decreasing execution count. Unexecuted instructions are not shown.

The −**sb** option causes **sdas** to disassemble a basic block at a time, sorting blocks by decreasing summed instruction execution count. Before each block **sdas** prints the instruction execution count sum, percent of total instructions executed, and the accumulated percentage for this and previously disassembled blocks. Unexecuted blocks are not shown.

The **+t**/−**t** options may be used to restrict disassembly to the indicated regions of text.

SEE ALSO

sprint(1sh), spixcounts(1sh), spixcounts(5sh).

NAME

spixjitcnt − postprocess spixcounts data from JIT compiler

SYNOPSIS

spixjitcnt −o exefile **−b** bbfile **−d** datafile_out datafile_in jit_sfile ...

DESCRIPTION

The **spixjitcnt** command is a postprocessor for the **spixcounts** Shade analyzer when run on a Java JIT compiler. JIT compilers generate code in their data space and then execute that code. The **spixcounts** analyzer reports such executed instructions in a special human-readable text file, which is not understood by the spix commands.

One deficiency of the special text file is that it does not correlate executed instructions with the Java class files or methods that they implement. However, the JIT compiler supports an option that causes it to write out a set of files with the disassembled instructions for each compiled Java class. The **spixjitcnt** tool combines the information from these disassemblies with the special text file written by the **spixcounts** analyzer. The result is a mock executable file and corresponding **spixcounts**(5sh) format file that describe the execution counts for the compiled Java classes. These files are suitable for input to the spix commands such as **spixstats, sdas,** etc.

The **spixjitcnt** command accepts several required switches. The **−o** switch specifies the name of the emitted mock executable file. This executable file contains the instructions and labels for all the compiled Java class methods. Note, this executable file cannot be run. It is only usable as input to the spix commands.

The **−b** switch specifies the name of the **spixcounts**(5sh) format file containing the execution counts for the instructions in the *exefile.* This file is suitable for input to the spix commands.

The **−d** switch specifies the name of an emitted human-readable text file. This file contains the instructions and execution counts for any instructions that do not belong to a compiled Java class method. The format of this file is identical to the *datafile_in* file created by the **spixcounts** analyzer. The *datafile_out* file may be used as input to a subsequent **spixjitcnt** command if desired.

After the switches, specify the names of the input files. The first input file must be the text file describing the execution counts of the instructions in an application's data space. Typically, this file is created from the **spixcounts** Shade analyzer, but it may also be the result of a prior **spixjitcnt** run (see the **−d** switch above). Following this file is a list of disassembly files created by the JIT compiler. These files contain the generated code and address locations for the compiled class files.

EXAMPLE

The following example demonstrates how the **spixcounts, spixjitcnt,** and **spixstats** commands can be used to analyze the code generated for the Java clock applet. This assumes that java has been configured to run the JIT compiler and to dump disassembly files for any compiled class files.

```
% spixcounts -b "%p.bb" -shlibs -data "clock.data" -- \
   java sun.applet.AppletViewer clock.html

% spixjitcnt -o clock.exe -b clock.bb -d clock_out.data \
   clock.data *.s

% spixstats -b clock.bb clock.exe
```

SEE ALSO

spixcounts(1sh), spixstats(1sh).

NAME

spixstats − summarize spixcounts

SYNOPSIS

spixstats −b bbin [**−g** gfile] [±**t**A]... [±**t**[A],[B]]... [**−v**] [program]

DESCRIPTION

The **spixstats** command prints various statistics about the run-time behavior of the program *program* (*a.out* by default) using the **spixcounts**(5sh) format file *bbin*.

The **−g** option allows the user to specify an alternate grouping of opcodes for various tables. *gfile* should contain lines of the form:

    group-name opcode-names

An opcode may appear in at most one group. Unnamed opcodes are collected into an ‘‘other’’ group.

The **+t**/**−t** options may be used to restrict statistics to the indicated regions of text.

The **−v** option causes **spixstats** to produce expanded versions of some tables.

FILES

$SHADE/lib/spixstats.groups	default opcode group file

SEE ALSO

spixcounts(1sh), spixcounts(5sh).

NAME

sprint − print source code with spixcounts

SYNOPSIS

sprint **−b** bbin [**−d**] [**−i**] [**−l**] program source...

DESCRIPTION

The **sprint** command prints one or more source files, *source*, corresponding to the program *program* with executable lines preceded by the number of times they were executed. The execution counts are derived from the **spixcounts**(5sh) file *bbin*. The *program* must have been compiled with the **−g** option.

The **−d** option causes **sprint** to disassemble the instructions and print them along with the corresponding source line.

With the **−i** option, the number of instructions executed for each executable source line is printed instead of the number of times the line was executed.

The **−l** option causes **sprint** to line number the output source code.

SEE ALSO

cc(1), tcov(1).
sdas(1sh), spixcounts(1sh), spixcounts(5sh).

BUGS

Counts are occasionally off by one, or off by one line.

Compiling with **−g** precludes optimization.

NAME

spix_sparc_dis, spix_sparc_dis32 − Disassemble a SPARC instruction

SYNOPSIS

#include <spix_sparc.h>

size_t spix_sparc_dis(char ∗*buf***, size_t** *szbuf***, spix_sparc_iop_t** *iop***, const void** ∗*pinst***,**
spix_addr64_t *addr***);**

size_t spix_sparc_dis32(char ∗*buf***, size_t** *szbuf***, spix_sparc_iop_t** *iop***, const void** ∗*pinst***,**
spix_addr32_t *addr***);**

DESCRIPTION

These functions disassemble a single SPARC instruction into the given buffer. The buffer *buf* must have size *szbuf*. The instruction must be represented by the opcode value *iop*, which is the value returned by a previous call to **spix_sparc_iop(3sh)**. The pointer *pinst* must point to the beginning of the instruction, and *addr* must be the instruction's virtual address.

Both functions write the disassembly to *buf*. If there is not enough room in *buf*, only the first *szbuf* characters of the disassembly are written and the string is not NULL terminated. Otherwise, the string is terminated with a NULL character.

The **spix_sparc_dis32( )** function is provided for the benefit of compilers that do not support a 64-bit integral type. Aside from limiting the address to 32 bits, it is identical to **spix_sparc_dis( )**.

RETURN VALUES

Both functions return the number of bytes written to the buffer, not including any terminating NULL character. If an error is detected, both functions return zero. Note, when the returned value is *szbuf*, the disassembly is truncated because the buffer is too small.

SEE ALSO

spix_sparc_iop(3sh), spix_sparc_iop_name(3sh), spix_sparc_ireg_name(3sh), spix_sparc_asreg_name(3sh).

NAME

spix_sparc_iop − Return a SPARC instruction’s opcode value

SYNOPSIS

#include <spix_sparc.h>

spix_sparc_iop_t spix_sparc_iop(spix_sparc_ver_t *ver***, const void** **pinst***);**

DESCRIPTION

The **spix_sparc_iop( )** function calculates the an "opcode" value for a SPARC instruction. This opcode value may be used with many of the functions defined in the Spix or Shade libraries, however, the opcode value may not match any of the opcodes listed in the SPARC Architecture Manual. See the <**spix_sparc_iop.h**> header for a list of possible opcode values.

The *ver* parameter specifies the SPARC architecture version to use when decoding the instruction. Possible values are:

SPIX_SPARC_V8
Identifies the SPARC V8 architecture.

SPIX_SPARC_V9
Identifies the SPARC V9 architecture.

The *pinst* parameter points to the start of the instruction to decode.

RETURN VALUES

The **spix_sparc_iop( )** function returns the instruction’s opcode value if the instruction is defined for the specified SPARC architecture version. It returns (spix_sparc_iop_t)-1 if it is not defined.

SEE ALSO

spix_sparc_dis(3sh), spix_sparc_iop_istype(3sh), spix_sparc_iop_name(3sh).

NAME

spix_sparc_iop_istype − Classify SPARC instruction by type

SYNOPSIS

#include <spix_sparc.h>

spix_bool_t spix_sparc_iop_istype(spix_sparc_iop_t *iop***, spix_sparc_itype_t** *itype***);**

DESCRIPTION

The **spix_sparc_iop_istype( )** function returns TRUE if the given instruction opcode is a member of the given instruction type. The opcode *iop* must be the value returned by a previous call to **spix_sparc_iop(3sh)**. The type *itype* may be any one of the following values. Note the first few correspond to the architecture independent instruction classifications in Shade.

SPIX_SPARC_ITYPE_FP
This type corresponds to the Shade **SHADE_ICLASS_FP** class. It selects all floating point instructions. This includes all FPOP1 and FPOP2 instructions, all loads or stores to floating point registers (including the %fsr register), the FBfcc and FBPfcc instructions, the MOVcc instructions that use the %fcc conditions, and all UltraSPARC extended "vis" instructions (including those that do not use floating point registers).

SPIX_SPARC_ITYPE_LOAD
This type corresponds to the Shade **SHADE_ICLASS_LOAD** class. It selects all instructions that load a value from memory. This includes all integer load instructions, all FP load instructions, and all atomic load/store instructions.

SPIX_SPARC_ITYPE_USTORE
This type corresponds to the Shade **SHADE_ICLASS_USTORE** class. It selects all instructions that unconditionally store a value to memory. This includes all integer store instructions, all FP store instructions, and the LDSTUB, LDSTUBA, SWAP, and SWAPA atomic load/store instructions.

SPIX_SPARC_ITYPE_CSTORE
This type corresponds to the Shade **SHADE_ICLASS_CSTORE** class. It selects all instructions that conditionally store a value to memory. This includes the CASA and CASXA instructions.

SPIX_SPARC_ITYPE_BRANCH
This type corresponds to the Shade **SHADE_ICLASS_BRANCH** class. It selects all conditional branch instructions, whether on the "always" condition, the "never" condition, or any other condition. Note, this type is the union of the **SPIX_SPARC_ITYPE_UBRANCH** and **SPIX_SPARC_ITYPE_CBRANCH** types below.

SPIX_SPARC_ITYPE_UBRANCH
This type corresponds to the Shade **SHADE_ICLASS_UBRANCH** class. It selects all unconditional branch instructions. This includes only conditional branch instructions on the "always" condition. Note, it does not include CALL, JMPL, or RETURN instructions.

SPIX_SPARC_ITYPE_CBRANCH
This type corresponds to the Shade **SHADE_ICLASS_CBRANCH** class. It selects all conditional branch instructions. This includes conditional branch instructions on any condition other than "always".

SPIX_SPARC_ITYPE_TRAP
This type corresponds to the Shade **SHADE_ICLASS_TRAP** class. It selects all TCC instructions.

SPIX_SPARC_ITYPE_V8
This type selects all instructions that are defined for the SPARC V8 architecture.

SPIX_SPARC_ITYPE_V9

This type selects all instructions that are defined for the SPARC V9 architecture. (It does not include any of the UltraSPARC extended instructions.)

SPIX_SPARC_ITYPE_VIS
This type selects all the UltraSPARC extended "vis" instructions.

SPIX_SPARC_ITYPE_PRIV
This type selects all of the privileged instructions. It does not select instructions that are privileged only for certain operands (such as RDASR) or that are only privileged depending on the processor's configuration (such as RDTICK).

SPIX_SPARC_ITYPE_BAA
This type selects all conditional branches on the "always" condition that also have the "annul" attribute set.

SPIX_SPARC_ITYPE_DCTI
This type selects all delayed control transfer instructions.

SPIX_SPARC_ITYPE_USECCR
This type selects all instructions that use either the %xcc or %icc condition codes. This includes the RDCCR instruction.

SPIX_SPARC_ITYPE_SETCCR
This type selects all instructions that set either the %xcc or %icc condition codes. This includes the WRCCR instruction.

SPIX_SPARC_ITYPE_SETFCC
This type selects all instructions that set any of the floating point condition codes. This includes the LDFSR and LDXFSR instructions.

SPIX_SPARC_ITYPE_MOVCC
This type selects all conditional integer register move instructions that are contingent on either the integer or floating point condition codes.

SPIX_SPARC_ITYPE_FMOVCC
This type selects all conditional floating point register move instructions that are contingent on either the integer or floating point condition codes.

SPIX_SPARC_ITYPE_MOVR
This type selects all conditional integer register move instructions that are contingent on the value of a register.

SPIX_SPARC_ITYPE_FMOVR
This type selects all conditional floating point register move instructions that are contingent on the value of a register.

SPIX_SPARC_ITYPE_BICC
This type selects all non-predicted branch instructions that are contingent on the integer condition codes.

SPIX_SPARC_ITYPE_FBFCC
This type selects all non-predicted branch instructions that are contingent on the floating point condition codes.

SPIX_SPARC_ITYPE_BPCC
This type selects all predicted branch instructions that are contingent on the integer condition codes.

SPIX_SPARC_ITYPE_FBPFCC
This type selects all predicted branch instructions that are contingent on the floating point condition codes.

SPIX_SPARC_ITYPE_BPR

This type selects all branch instructions that are contingent on the value of a register.

SPIX_SPARC_ITYPE_FPOP1
This type selects all FPOP1 instructions as defined by the SPARC Architecture Manual.

SPIX_SPARC_ITYPE_ALU
This type selects all instructions that perform an integer arithmetic or logical operation. This does not include load, store, branch, or floating point instructions.

SPIX_SPARC_ITYPE_ILOAD
This type selects all instructions that load a value from memory into an integer register. This includes all atomic load/store instructions.

SPIX_SPARC_ITYPE_ISTORE
This type selects all instructions that conditionally or unconditionally store a value from an integer register into memory. This includes all atomic load/store instructions.

Note, the Shade instruction class **SHADE_ICLASS_IWSTART** does not have a corresponding instruction type above. Since SPARC is not a VLIW architecture, the **SHADE_ICLASS_IWSTART** class selects all instructions.

The <**spix_sparc.h**> header also defines the following macros that provide a convenient way to test if an instruction opcode belongs to any of the types listed above.

```
spix_bool_t spix_sparc_iop_isfp(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isload(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isustore(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_iscstore(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isbranch(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isubranch(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_iscbranch(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_istrap(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isv8(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isv9(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isvis(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_ispriv(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isbaa(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isdcti(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isuseccr(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_issetccr(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_issetfcc(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_ismovcc(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isfmovcc(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_ismovr(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isfmovr(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isbicc(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isfbfcc(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isbpcc(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isfbpfcc(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isbpr(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isfpop1(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isalu(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isiload(spix_sparc_iop_t)
spix_bool_t spix_sparc_iop_isistore(spix_sparc_iop_t)
```

RETURN VALUES

The **spix_sparc_iop_istype( )** function and all the macros above return TRUE if the instruction is a member of the requested type. They return FALSE if the instruction is not a member.

SEE ALSO

spix_sparc_iop(3sh), shade_iset(3sh).

NAME

spix_sparc_iop_memsize − Return size of memory referenced by SPARC instruction

SYNOPSIS

#include <spix_sparc.h>

size_t spix_sparc_iop_memsize(spix_sparc_iop_t *iop* **);**

DESCRIPTION

The **spix_sparc_iop_memsize( )** instruction returns the number of bytes of memory referenced by a load or store instruction with the given opcode value. If the opcode *iop* does not correspond to a load or store instruction, **spix_sparc_iop_memsize( )** returns zero.

SEE ALSO

spix_sparc_iop(3sh).

NAME

spix_sparc_iop_name, spix_sparc_iop_Lname − Return the name for a SPARC instruction opcode

SYNOPSIS

#include <spix_sparc.h>

const char *spix_sparc_iop_name(spix_sparc_iop_t *iop*);

extern const size_t spix_sparc_iop_Lname;

DESCRIPTION

The **spix_sparc_iop_name( )** function returns a NULL terminated string representation of the name for the given opcode value. The constant **spix_sparc_iop_Lname** is the length of the longest string returned by **spix_sparc_iop_name( )**.

SEE ALSO

spix_sparc_iop(3sh), spix_sparc_dis(3sh).

NAME

spix_sparc_iop_regpos − Return register usage at position in SPARC instruction

SYNOPSIS

#include <spix_sparc.h>

spix_sparc_ruact_t spix_sparc_iop_reguse(spix_sparc_iop_t *iop***, spix_sparc_rupos_t** *pos***);**

DESCRIPTION

The **spix_sparc_iop_reguse( )** function returns an indication of how instructions with a particular opcode use a given register field. The *iop* parameter specifies the opcode to query, and *pos* specifies the register position. The *pos* parameter may have one of the following values.

SPIX_SPARC_RUPOS_RS1
Query the RS1 field of the instruction.

SPIX_SPARC_RUPOS_RS2
Query the RS2 field of the instruction.

SPIX_SPARC_RUPOS_RD
Query the RD field of the instruction.

RETURN VALUES

The **spix_sparc_iop_reguse( )** function returns either **SPIX_SPARC_RUACT_NONE** to indicate that the instruction does not use the given register field or one of the other **spix_sparc_ruact_t** values as defined on **spix_sparc_reguse(3sh)**.

SEE ALSO

spix_sparc_iop(3sh), spix_sparc_reguse(3sh).

NAME

spix_sparc_ireg_name, spix_sparc_asreg_name − SPARC register names

SYNOPSIS

#include <spix_sparc.h>

const char *spix_sparc_ireg_name(unsigned *reg***);**

const char *spix_sparc_asreg_name(unsigned *reg***);**

DESCRIPTION

These functions return software names for SPARC registers. The **spix_sparc_ireg_name( )** function returns the software name for the integer register *reg* or NULL if *reg* is an invalid register number. The **spix_sparc_asreg_name( )** function returns the name of the given ancillary state register or NULL if *reg* is not a valid ancillary state register number.

SEE ALSO

spix_sparc_dis(3sh).

NAME

spix_sparc_reguse − Determine registers used by a SPARC instruction

SYNOPSIS

#include <spix_sparc.h>

void spix_sparc_reguse(spix_sparc_iop_t *iop***, const void ∗***pinst***,**
  void (∗*pfun***)(spix_sparc_ruact_t, spix_sparc_rupos_t, unsigned, void ∗),**
  void ∗*pdata***);**

DESCRIPTION

The **spix_sparc_reguse( )** function reports the registers that are used, either explicitly or implicitly, by the given instruction. The instruction's opcode value *iop* must be the valued returned from a previous call to **spix_sparc_iop(3sh)**. The pointer *pinst* must point to the text of the instruction.

The **spix_sparc_reguse( )** function calls the user-supplied *pfun* function once fore each register used by the instruction. Four parameters are passed to this function. The first is one of the following values, which indicates how the register is used.

SPIX_SPARC_RUACT_RI
SPIX_SPARC_RUACT_R2I
  Instruction reads an integer register or an integer register pair.

SPIX_SPARC_RUACT_WI
SPIX_SPARC_RUACT_W2I
  Instruction writes an integer register or an integer register pair.

SPIX_SPARC_RUACT_RWI
SPIX_SPARC_RUACT_RW2I
  Instruction both reads and writes an integer register or integer register pair.

SPIX_SPARC_RUACT_RF
SPIX_SPARC_RUACT_R2F
SPIX_SPARC_RUACT_R4F
  Instruction reads a single-precision, double precision, or quad precision floating point register.

SPIX_SPARC_RUACT_WF
SPIX_SPARC_RUACT_W2F
SPIX_SPARC_RUACT_W4F
  Instruction writes a single-precision, double precision, or quad precision floating point register.

SPIX_SPARC_RUACT_RWF
SPIX_SPARC_RUACT_RW2F
SPIX_SPARC_RUACT_RW4F
  Instruction both reads and writes a single-precision, double precision, or quad precision floating point register.

SPIX_SPARC_RUACT_RS
SPIX_SPARC_RUACT_WS
SPIX_SPARC_RUACT_RWS
  Instruction reads, writes, or both reads and writes a special register.

The second parameter to the *pfun* function is one of the following values, which indicates the register's position in the instruction.

SPIX_SPARC_RUPOS_RS1
  Register is in the RS1 position.

SPIX_SPARC_RUPOS_RS2
  Register is in the RS2 position.

SPIX_SPARC_RUPOS_RD
Register is in the RD position.

SPIX_SPARC_RUPOS_IMP
The instruction does not explicitly reference the register, but its use is implied.

The third parameter to the *pfun* function is the register number. The function must interpret this register as either an integer, floating point, or special register number according to the value of the first parameter. If this is special register, the third parameter has one of the following values.

SPIX_SPARC_SREG_Y
The %y register.

SPIX_SPARC_SREG_ASI
The %asi register.

SPIX_SPARC_SREG_CCR
The %ccr register.

SPIX_SPARC_SREG_FPRS
The %fprs register.

SPIX_SPARC_SREG_FCC
One of the condition code fields of the %fsr register.

SPIX_SPARC_SREG_RM
The rounding mode field of the %fsr.

SPIX_SPARC_SREG_FSR
Another other field of the %fsr.

SPIX_SPARC_SREG_TICK
The %tick register.

SPIX_SPARC_SREG_GSR
The UltraSPARC extended %gsr register.

The final parameter to the *pfun* function is the *pdata* value passed to **spix_sparc_reguse( )**.

SEE ALSO

spix_sparc_iop(3sh), spix_sparc_iop_regpos(3sh).

NAME

spixcounts − spixcounts file format

SYNOPSIS

#include <spixcounts.h>

DESCRIPTION

The **spixcounts** file format is generated by the **spixcounts(1sh)** Shade analyzer, and is consumed by several of the SpixTools. This file format contains three parts: a header, basic block lengths, and basic block execution counts.

The header has the following format:

```
typedef struct {
        unsigned  sc_magic;
        unsigned  sc_tsize;
        unsigned  sc_tsum;
        unsigned  sc_nbb;
} spix_schdr_t;
```

The *sc_magic* field helps distinguish this type of file from other types, and this version of spixcounts files from other versions. It currently has the value:

```
#define SPIX_SC_MAGIC 0x62620011
```

The *sc_tsize* field and the *sc_tsim* field contain the size (in bytes) and a checksum of the original program's text segment.

The *sc_nbb* field contains the number of basic blocks in the original program.

After the header comes an array of bytes representing the lengths of each basic block. Each byte contains the number of instructions (4 byte words) in its basic block. Blocks longer than 255 instructions must be split, so their lengths can be represented in a byte. This section of the spixcounts file is padded on the end to a word boundary.

Finally comes for each basic block a pair of (4 byte) counters. For basic blocks containing a branch, the first counter contains the number of times the branch was taken, and the second contains the number of times the branch wasn't taken. For other executable blocks, the first counter contains the number of times the block was executed. For non-executable blocks (e.g. data) both counters will have the value:

```
#define SPIX_SC_BADCNT 0xffffffff
```

DIAGNOSTICS

Commands which read spixcounts files may produce the following diagnostics:

spixcounts-file: bad magic

The magic number (*sc_magic*) field in the file *spixcounts-file* was incorrect (not **SPIX_SC_MAGIC**). The file is not (the expected version of) a spixcounts file.

program and *spixcounts-file* don't jibe

This indicates a mismatch in text size (*sc_size*) or checksum (*sc_tsum*) between the named program and spixcounts file. The spixcounts file must correspond to a different program.

spixcounts-file: unexpected EOF

The file *spixcounts-file* is incomplete. The instrumented program may have terminated before writing the spixcounts, or an output error (e.g. insufficient disk space) may have precluded writing the spixcounts.

SEE ALSO

spixcounts(1sh), sadd(1sh), sdas(1sh), spixstats(1sh), sprint(1sh).