```
**********************************************************
   18333 Sun Dec 16 13:00:30 2012
new/usr/src/cmd/dis/dis_main.c
style fixes
comments; lint
unbreak dis
move fixed-size determination mostly into libdisasm
take to dis and libdisasm with an axe; does not yet compile
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  *
  26  * Copyright 2011 Jason King.  All rights reserved.
  27  * Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
  28  */

  30 #include <ctype.h>
  31 #include <getopt.h>
  32 #include <stdio.h>
  33 #include <stdlib.h>
  34 #include <string.h>
  35 #include <sys/sysmacros.h>
  36 #include <sys/elf_SPARC.h>

  38 #include <libdisasm.h>

  40 #include "dis_target.h"
  41 #include "dis_util.h"
  42 #include "dis_list.h"

  44 int g_demangle;         /* Demangle C++ names */
  45 int g_quiet;            /* Quiet mode */
  46 int g_numeric;          /* Numeric mode */
  47 int g_flags;            /* libdisasm language flags */
  48 int g_doall;            /* true if no functions or sections were given */

  50 dis_namelist_t *g_funclist;     /* list of functions to disassemble, if any */
  51 dis_namelist_t *g_seclist;      /* list of sections to disassemble, if any */

  53 /*
  54  * Section options for -d, -D, and -s
  55  */
  56 #define DIS_DATA_RELATIVE       1
  57 #define DIS_DATA_ABSOLUTE       2
```



```
  58 #define DIS_TEXT                3

  60 /*
  61  * libdisasm callback data.  Keeps track of current data (function or section)
  62  * and offset within that data.
  63  */
  64 typedef struct dis_buffer {
  65         dis_tgt_t       *db_tgt;        /* current dis target */
  66         void            *db_data;       /* function or section data */
  67         uint64_t        db_addr;        /* address of function start */
  68         size_t          db_size;        /* size of data */
  69         uint64_t        db_nextaddr;    /* next address to be read */
  70 } dis_buffer_t;
_____unchanged_portion_omitted_

 100 /*
 101  * Determine if we are on an architecture with fixed-size instructions,
 102  * and if so, what size they are.
 103  */
 104 static int
 105 insn_size(dis_handle_t *dhp)
 106 {
 107         int min = dis_min_instrlen(dhp);
 108         int max = dis_max_instrlen(dhp);

 110         if (min == max)
 111                 return (min);

 113         return (0);
 114 }

 116 /*
 117  * The main disassembly routine.  Given a fixed-sized buffer and starting
 118  * address, disassemble the data using the supplied target and libdisasm handle.
 119  */
 120 void
 121 dis_data(dis_tgt_t *tgt, dis_handle_t *dhp, uint64_t addr, void *data,
 122     size_t datalen)
 123 {
 124         dis_buffer_t db = { 0 };
 125         char buf[BUFSIZE];
 126         char symbuf[BUFSIZE];
 127         const char *symbol;
 128         const char *last_symbol;
 129         off_t symoffset;
 130         int i;
 131         int bytesperline;
 132         size_t symsize;
 133         int isfunc;
 134         size_t symwidth = 0;
 135         int ret;
 136         int insz = insn_size(dhp);

 138         db.db_tgt = tgt;
 139         db.db_data = data;
 140         db.db_addr = addr;
 141         db.db_size = datalen;

 143         dis_set_data(dhp, &db);

 145         if ((bytesperline = dis_max_instrlen(dhp)) > 6)
 146                 bytesperline = 6;

 148         symbol = NULL;

 150         while (addr < db.db_addr + db.db_size) {
```

```
152                 ret = dis_disassemble(dhp, addr, buf, BUFSIZE);
153                 if (ret != 0 && insz > 0) {
133                 if (dis_disassemble(dhp, addr, buf, BUFSIZE) != 0) {
134 #if defined(__sparc)
154                         /*
155                          * Since we know instructions are fixed size, we
136                          * Since sparc instructions are fixed size, we
156                          * always know the address of the next instruction
157                          */
158                         (void) snprintf(buf, sizeof (buf),
159                             "*** invalid opcode ***");
160                         db.db_nextaddr = addr + insz;
141                         db.db_nextaddr = addr + 4;

162                 } else if (ret != 0) {
143 #else
163                         off_t next;

165                         (void) snprintf(buf, sizeof (buf),
166                             "*** invalid opcode ***");

168                         /*
169                          * On architectures with variable sized instructions
170                          * we have no way to figure out where the next
171                          * instruction starts if we encounter an invalid
172                          * instruction.  Instead we print the rest of the
173                          * instruction stream as hex until we reach the
174                          * next valid symbol in the section.
175                          */
176                         if ((next = dis_tgt_next_symbol(tgt, addr)) == 0) {
177                                 db.db_nextaddr = db.db_addr + db.db_size;
178                         } else {
179                                 if (next > db.db_size)
180                                         db.db_nextaddr = db.db_addr +
181                                             db.db_size;
182                                 else
183                                         db.db_nextaddr = addr + next;
184                         }
166 #endif
185                 }

187                 /*
188                  * Print out the line as:
189                  *
190                  *      address:        bytes   text
191                  *
192                  * If there are more than 6 bytes in any given instruction,
193                  * spread the bytes across two lines.  We try to get symbolic
194                  * information for the address, but if that fails we print out
195                  * the numeric address instead.
196                  *
197                  * We try to keep the address portion of the text aligned at
198                  * MINSYMWIDTH characters.  If we are disassembling a function
199                  * with a long name, this can be annoying.  So we pick a width
200                  * based on the maximum width that the current symbol can be.
201                  * This at least produces text aligned within each function.
202                  */
203                 last_symbol = symbol;
204                 symbol = dis_tgt_lookup(tgt, addr, &symoffset, 1, &symsize,
205                     &isfunc);
206                 if (symbol == NULL) {
207                         symbol = dis_find_section(tgt, addr, &symoffset);
208                         symsize = symoffset;
209                 }
```



```
211                 if (symbol != last_symbol)
212                         getsymname(addr, symbol, symsize, symbuf,
213                             sizeof (symbuf));

215                 symwidth = MAX(symwidth, strlen(symbuf));
216                 getsymname(addr, symbol, symoffset, symbuf, sizeof (symbuf));

218                 /*
219                  * If we've crossed a new function boundary, print out the
220                  * function name on a blank line.
221                  */
222                 if (!g_quiet && symoffset == 0 && symbol != NULL && isfunc)
223                         (void) printf("%s()\n", symbol);

225                 (void) printf("    %s:%*s ", symbuf,
226                     symwidth - strlen(symbuf), "");

228                 /* print bytes */
229                 for (i = 0; i < MIN(bytesperline, (db.db_nextaddr - addr));
230                     i++) {
231                         int byte = *((uchar_t *)data + (addr - db.db_addr) + i);
232                         if (g_flags & DIS_OCTAL)
233                                 (void) printf("%03o ", byte);
234                         else
235                                 (void) printf("%02x ", byte);
236                 }

238                 /* trailing spaces for missing bytes */
239                 for (; i < bytesperline; i++) {
240                         if (g_flags & DIS_OCTAL)
241                                 (void) printf("    ");
242                         else
243                                 (void) printf("   ");
244                 }

246                 /* contents of disassembly */
247                 (void) printf(" %s", buf);

249                 /* excess bytes that spill over onto subsequent lines */
250                 for (; i < db.db_nextaddr - addr; i++) {
251                         int byte = *((uchar_t *)data + (addr - db.db_addr) + i);
252                         if (i % bytesperline == 0)
253                                 (void) printf("\n    %*s  ", symwidth, "");
254                         if (g_flags & DIS_OCTAL)
255                                 (void) printf("%03o ", byte);
256                         else
257                                 (void) printf("%02x ", byte);
258                 }

260                 (void) printf("\n");

262                 addr = db.db_nextaddr;
263         }
264 }
```

*unchanged_portion_omitted_*

```
464 /*
465  * Disassemble a complete file.  First, we determine the type of the file based
466  * on the ELF machine type, and instantiate a version of the disassembler
467  * appropriate for the file.  We then resolve any named sections or functions
468  * against the file, and iterate over the results (or all sections if no flags
469  * were specified).
470  */
471 void
472 dis_file(const char *filename)
473 {
```

```
474         dis_tgt_t *tgt, *current;
475         dis_scnlist_t *sections;
476         dis_funclist_t *functions;
477         dis_handle_t *dhp;
478         GElf_Ehdr ehdr;

480         /*
481          * First, initialize the target
482          */
483         if ((tgt = dis_tgt_create(filename)) == NULL)
484                 return;

486         if (!g_quiet)
487                 (void) printf("disassembly for %s\n\n",  filename);

489         /*
490          * A given file may contain multiple targets (if it is an archive, for
491          * example).  We iterate over all possible targets if this is the case.
492          */
493         for (current = tgt; current != NULL; current = dis_tgt_next(current)) {
494                 dis_tgt_ehdr(current, &ehdr);

496                 /*
497                  * Eventually, this should probably live within libdisasm, and
498                  * we should be able to disassemble targets from different
499                  * architectures.  For now, we only support objects as the
500                  * native machine type.
501                  */
502                 switch (ehdr.e_machine) {
485 #ifdef __sparc
503                 case EM_SPARC:
504                         if (ehdr.e_ident[EI_CLASS] != ELFCLASS32 ||
505                             ehdr.e_ident[EI_DATA] != ELFDATA2MSB) {
506                                 warn("invalid E_IDENT field for SPARC object");
507                                 return;
508                         }
509                         g_flags |= DIS_SPARC_V8;
510                         break;

512                 case EM_SPARC32PLUS:
513                 {
514                         uint64_t flags = ehdr.e_flags & EF_SPARC_32PLUS_MASK;

516                         if (ehdr.e_ident[EI_CLASS] != ELFCLASS32 ||
517                             ehdr.e_ident[EI_DATA] != ELFDATA2MSB) {
518                                 warn("invalid E_IDENT field for SPARC object");
519                                 return;
520                         }

522                         if (flags != 0 &&
523                             (flags & (EF_SPARC_32PLUS | EF_SPARC_SUN_US1 |
524                             EF_SPARC_SUN_US3)) != EF_SPARC_32PLUS)
525                                 g_flags |= DIS_SPARC_V9 | DIS_SPARC_V9_SGI;
526                         else
527                                 g_flags |= DIS_SPARC_V9;
528                         break;
529                 }

531                 case EM_SPARCV9:
532                         if (ehdr.e_ident[EI_CLASS] != ELFCLASS64 ||
533                             ehdr.e_ident[EI_DATA] != ELFDATA2MSB) {
534                                 warn("invalid E_IDENT field for SPARC object");
535                                 return;
536                         }

538                         g_flags |= DIS_SPARC_V9 | DIS_SPARC_V9_SGI;
```



```
539                         break;
523 #endif /* __sparc */

525 #if defined(__i386) || defined(__amd64)
541                 case EM_386:
542                         g_flags |= DIS_X86_SIZE32;
543                         break;

545                 case EM_AMD64:
546                         g_flags |= DIS_X86_SIZE64;
547                         break;
533 #endif /* __i386 || __amd64 */

549                 default:
550                         die("%s: unsupported ELF machine 0x%x", filename,
551                             ehdr.e_machine);
552                 }

554                 /*
555                  * If ET_REL (.o), printing immediate symbols is likely to
556                  * result in garbage, as symbol lookups on unrelocated
557                  * immediates find false and useless matches.
558                  */

560                 if (ehdr.e_type == ET_REL)
561                         g_flags |= DIS_NOIMMSYM;

563                 if (!g_quiet && dis_tgt_member(current) != NULL)
564                         (void) printf("\narchive member %s\n",
565                             dis_tgt_member(current));

567                 /*
568                  * Instantiate a libdisasm handle based on the file type.
569                  */
570                 if ((dhp = dis_handle_create(g_flags, current, do_lookup,
571                     do_read)) == NULL)
572                         die("%s: failed to initialize disassembler: %s",
573                             filename, dis_strerror(dis_errno()));

575                 if (g_doall) {
576                         /*
577                          * With no arguments, iterate over all sections and
578                          * disassemble only those that contain text.
579                          */
580                         dis_tgt_section_iter(current, dis_text_section, dhp);
581                 } else {
582                         callback_arg_t ca;

584                         ca.ca_tgt = current;
585                         ca.ca_handle = dhp;

587                         /*
588                          * If sections or functions were explicitly specified,
589                          * resolve those names against the object, and iterate
590                          * over just the resulting data.
591                          */
592                         sections = dis_namelist_resolve_sections(g_seclist,
593                             current);
594                         functions = dis_namelist_resolve_functions(g_funclist,
595                             current);

597                         dis_scnlist_iter(sections, dis_named_section, &ca);
598                         dis_funclist_iter(functions, dis_named_function, &ca);

600                         dis_scnlist_destroy(sections);
601                         dis_funclist_destroy(functions);
```

```
602                 }

604                 dis_handle_destroy(dhp);
605         }

607         dis_tgt_destroy(tgt);
608 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   22984 Sun Dec 16 13:00:30 2012
new/usr/src/cmd/dis/dis_target.c
remove crap I didnt end up using
take to dis and libdisasm with an axe; does not yet compile
**********************************************************
```

_____unchanged_portion_omitted_

```
712 #if !defined(__sparc)
712 /*
713  * Given an address, return the starting offset of the next symbol in the file.
714  * Only needed on variable length instruction architectures.
715  */
716 off_t
717 dis_tgt_next_symbol(dis_tgt_t *tgt, uint64_t addr)
718 {
719         sym_entry_t *sym;

721         for (sym = tgt->dt_symcache;
722             sym != tgt->dt_symtab + tgt->dt_symcount;
723             sym++) {
724                 if (sym->se_sym.st_value >= addr)
725                         return (sym->se_sym.st_value - addr);
726         }

728         return (0);
729 }
731 #endif

731 /*
732  * Iterate over all sections in the target, executing the given callback for
733  * each.
734  */
735 void
736 dis_tgt_section_iter(dis_tgt_t *tgt, section_iter_f func, void *data)
737 {
738         dis_scn_t sdata;
739         Elf_Scn *scn;
740         int idx;

742         for (scn = elf_nextscn(tgt->dt_elf, NULL), idx = 1; scn != NULL;
743             scn = elf_nextscn(tgt->dt_elf, scn), idx++) {

745                 if (gelf_getshdr(scn, &sdata.ds_shdr) == NULL) {
746                         warn("%s: failed to get section %d header",
747                             tgt->dt_filename, idx);
748                         continue;
749                 }

751                 if ((sdata.ds_name = elf_strptr(tgt->dt_elf, tgt->dt_shstrndx,
752                     sdata.ds_shdr.sh_name)) == NULL) {
753                         warn("%s: failed to get section %d name",
754                             tgt->dt_filename, idx);
755                         continue;
756                 }

758                 if ((sdata.ds_data = elf_getdata(scn, NULL)) == NULL) {
759                         warn("%s: failed to get data for section '%s'",
760                             tgt->dt_filename, sdata.ds_name);
761                         continue;
762                 }

764                 /*
765                  * dis_tgt_section_iter is also used before the section map
766                  * is initialized, so only check when we need to.  If the
767                  * section map is uninitialized, it will return 0 and have
```



```
768                  * no net effect.
769                  */
770                 if (sdata.ds_shdr.sh_addr == 0)
771                         sdata.ds_shdr.sh_addr = tgt->dt_shnmap[idx].dm_start;

773                 func(tgt, &sdata, data);
774         }
775 }
```

_____unchanged_portion_omitted_

```
**********************************************************
    2672 Sun Dec 16 13:00:30 2012
new/usr/src/cmd/dis/dis_target.h
remove more unused crap
take to dis and libdisasm with an axe; does not yet compile
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  *
  26  * Copyright 2011 Jason King.  All rights reserved.
  27  */

  29 #ifndef _DIS_TARGET_H
  30 #define _DIS_TARGET_H

  32 #ifdef __cplusplus
  33 extern "C" {
  34 #endif

  36 #include <gelf.h>
  37 #include <sys/types.h>

  39 /*
  40  * Basic types
  41  */
  42 typedef struct dis_tgt dis_tgt_t;
  43 typedef struct dis_func dis_func_t;
  44 typedef struct dis_scn dis_scn_t;

  46 /*
  47  * Target management
  48  */
  49 dis_tgt_t *dis_tgt_create(const char *);
  50 void dis_tgt_destroy(dis_tgt_t *);
  51 const char *dis_tgt_lookup(dis_tgt_t *, uint64_t, off_t *, int, size_t *,
  52     int *);
  53 const char *dis_find_section(dis_tgt_t *, uint64_t, off_t *);
  54 const char *dis_tgt_name(dis_tgt_t *);
  55 const char *dis_tgt_member(dis_tgt_t *);
  56 void dis_tgt_ehdr(dis_tgt_t *, GElf_Ehdr *);
  57 #if !defined(__sparc)
  57 off_t dis_tgt_next_symbol(dis_tgt_t *, uint64_t);
  59 #endif
  58 dis_tgt_t *dis_tgt_next(dis_tgt_t *);
```



```
  60 /*
  61  * Section management
  62  */
  63 typedef void (*section_iter_f)(dis_tgt_t *, dis_scn_t *, void *);
  64 void dis_tgt_section_iter(dis_tgt_t *, section_iter_f, void *);

  66 int dis_section_istext(dis_scn_t *);
  67 void *dis_section_data(dis_scn_t *);
  68 size_t dis_section_size(dis_scn_t *);
  69 uint64_t dis_section_addr(dis_scn_t *);
  70 const char *dis_section_name(dis_scn_t *);
  71 dis_scn_t *dis_section_copy(dis_scn_t *);
  72 void dis_section_free(dis_scn_t *);

  74 /*
  75  * Function management
  76  */
  77 typedef void (*function_iter_f)(dis_tgt_t *, dis_func_t *, void *);
  78 void dis_tgt_function_iter(dis_tgt_t *, function_iter_f, void *);
  79 dis_func_t *dis_tgt_function_lookup(dis_tgt_t *, const char *);

  81 void *dis_function_data(dis_func_t *);
  82 size_t dis_function_size(dis_func_t *);
  83 uint64_t dis_function_addr(dis_func_t *);
  84 const char *dis_function_name(dis_func_t *);
  85 dis_func_t *dis_function_copy(dis_func_t *);
  86 void dis_function_free(dis_func_t *);

  88 #ifdef __cplusplus
  89 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    4210 Sun Dec 16 13:00:31 2012
new/usr/src/lib/libdisasm/Makefile.com
need includes from common/dis/i386 on sparc too
reorder some things in Makefile.com
style fixes
only include native support in standalone library
rename instr.c
take to dis and libdisasm with an axe; does not yet compile
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
# Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
#

#
# The build process for libdisasm is sightly different from that used by other
# libraries, because libdisasm must be built in two flavors - as a standalone
# for use by kmdb and as a normal library.  We use $(CURTYPE) to indicate the
# current flavor being built.
#
# The SPARC library is built from the closed gate.  This Makefile is shared
# between both environments, so all paths must be absolute.
#

LIBRARY=        libdisasm.a
STANDLIBRARY=   libstanddisasm.so
VERS=           .1

# By default, we build the shared library.  Construction of the standalone
# is specifically requested by architecture-specific Makefiles.
TYPES=          library
CURTYPE=        library

COMDIR=         $(SRC)/lib/libdisasm/common

#
# Architecture-independent files
# Architecture-dependent files common to both versions of libdisasm
#
SRCS_common=            $(COMDIR)/libdisasm.c
OBJECTS_common=         libdisasm.o
OBJECTS_common_i386 = dis_i386.o dis_tables.o
OBJECTS_common_sparc = dis_sparc.o instr.o dis_sparc_fmt.o
```



```
SRCS_common_i386 = $(ISASRCDIR)/dis_i386.c $(SRC)/common/dis/i386/dis_tables.c
SRCS_common_sparc = $(ISASRCDIR)/dis_sparc.c $(ISASRCDIR)/instr.c \
        $(ISASRCDIR)/dis_sparc_fmt.c

#
# Architecture-dependent disassembly files
# Architecture-independent files common to both version of libdisasm
#
SRCS_i386=              $(COMDIR)/dis_i386.c \
                        $(SRC)/common/dis/i386/dis_tables.c
SRCS_sparc=             $(COMDIR)/dis_sparc.c \
                        $(COMDIR)/dis_sparc_fmt.c \
                        $(COMDIR)/dis_sparc_instr.c
OBJECTS_common_common = libdisasm.o
SRC_common_common = $(OBJECTS_common_common:%.o=$(COMDIR)/%.c)

OBJECTS_i386=           dis_i386.o \
                        dis_tables.o
OBJECTS_sparc=          dis_sparc.o \
                        dis_sparc_fmt.o \
                        dis_sparc_instr.o

#
# We build the regular shared library with support for all architectures.
# The standalone version should only contain code for the native
# architecture to reduce the memory footprint of kmdb.
#
OBJECTS_library=        $(OBJECTS_common) \
                        $(OBJECTS_i386) \
                        $(OBJECTS_sparc)
OBJECTS_standalone=     $(OBJECTS_common) \
                        $(OBJECTS_$(MACH))
OBJECTS=                $(OBJECTS_$(CURTYPE))
OBJECTS=                                \
        $(OBJECTS_common_$(MACH))       \
        $(OBJECTS_common_common)

include $(SRC)/lib/Makefile.lib

SRCS_library=           $(SRCS_common) \
                        $(SRCS_i386) \
                        $(SRCS_sparc)
SRCS_standalone=        $(SRCS_common) \
                        $(SRCS_$(MACH))
SRCS=                   $(SRCS_$(CURTYPE))
SRCS=                                   \
        $(SRCS_$(CURTYPE))              \
        $(SRCS_common_$(MACH))          \
        $(SRCS_common_common)

#
# Used to verify that the standalone doesn’t have any unexpected external
# dependencies.
#
LINKTEST_OBJ = objs/linktest_stand.o

CLOBBERFILES_standalone = $(LINKTEST_OBJ)
CLOBBERFILES += $(CLOBBERFILES_$(CURTYPE))

LIBS_standalone = $(STANDLIBRARY)
LIBS_library = $(DYNLIB) $(LINTLIB)
LIBS = $(LIBS_$(CURTYPE))

MAPFILES =      $(COMDIR)/mapfile-vers

LDLIBS +=       -lc
```

```
104 LDFLAGS_standalone = $(ZNOVERSION) $(BREDUCE) -dy -r
105 LDFLAGS = $(LDFLAGS_$(CURTYPE))

107 ASFLAGS_standalone = -DDIS_STANDALONE
108 ASFLAGS_library =
109 ASFLAGS += -P $(ASFLAGS_$(CURTYPE)) -D_ASM

111 $(LINTLIB) := SRCS = $(COMDIR)/$(LINTSRC)

113 CERRWARN +=     -_gcc=-Wno-parentheses
114 CERRWARN +=     -_gcc=-Wno-uninitialized

116 # We want the thread-specific errno in the library, but we don't want it in
117 # the standalone.  $(DTS_ERRNO) is designed to add -D_TS_ERRNO to $(CPPFLAGS),
118 # in order to enable this feature.  Conveniently, -D_REENTRANT does the same
119 # thing.  As such, we null out $(DTS_ERRNO) to ensure that the standalone
120 # doesn't get it.
121 DTS_ERRNO=

123 # We need to rename some standard functions so we can easily implement them
124 # in consumers.
125 STAND_RENAMED_FUNCS= \
126         snprintf

128 CPPFLAGS_standalone = -DDIS_STANDALONE $(STAND_RENAMED_FUNCS:%=-D%=mdb_%) \
129         -Dvsnprintf=mdb_iob_vsnprintf -I$(SRC)/cmd/mdb/common
130 CPPFLAGS_library = -D_REENTRANT
131 CPPFLAGS +=     -I$(COMDIR) $(CPPFLAGS_$(CURTYPE))

133 # For the x86 disassembler we have to include sources from usr/src/common
134 CPPFLAGS += -I$(SRC)/common/dis/i386 -DDIS_TEXT
121 #
122 # For x86, we have to link to sources in usr/src/common
123 #
124 CPPFLAGS_dis_i386 = -I$(SRC)/common/dis/i386 -DDIS_TEXT
125 CPPFLAGS_dis_sparc =
126 CPPFLAGS +=     $(CPPFLAGS_dis_$(MACH))

136 CFLAGS_standalone = $(STAND_FLAGS_32)
137 CFLAGS_common =
138 CFLAGS += $(CFLAGS_$(CURTYPE)) $(CFLAGS_common)

140 CFLAGS64_standalone = $(STAND_FLAGS_64)
141 CFLAGS64 += $(CCVERBOSE) $(CFLAGS64_$(CURTYPE)) $(CFLAGS64_common)

143 DYNFLAGS +=     $(ZINTERPOSE)

145 .KEEP_STATE:
```

```
**********************************************************
    2492 Sun Dec 16 13:00:31 2012
new/usr/src/lib/libdisasm/Makefile.targ
only include native support in standalone library
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# ident "%Z%%M% %I%     %E% SMI"
#

#
# We build each flavor in a separate make invocation to improve clarity(!) in
# Makefile.com.  The subordinate makes have $(CURTYPE) set to indicate the
# flavor they're supposed to build.  This causes the correct set of source
# files and compiler and linker flags to be selected.
#
# The SPARC library is built from the closed gate.  This Makefile is shared
# between both environments, so all paths must be absolute.
#

install: $(TYPES:%=install.%)

all: $(TYPES:%=all.%)

$(TYPES:%=all.%):
        @CURTYPE=$(@:all.%=%) $(MAKE) $@.targ

$(TYPES:%=install.%):
        @CURTYPE=$(@:install.%=%) $(MAKE) $@.targ

install.library.targ: all.library $(INSTALL_DEPS_library)
install.standalone.targ: all.standalone $(INSTALL_DEPS_standalone)

all.library.targ: $(LIBS)
all.standalone.targ: $(STANDLIBRARY)

lint: $(TYPES:%=lint.%)

$(TYPES:%=lint.%):
        @CURTYPE=$(@:lint.%=%) $(MAKE) lintcheck

$(STANDLIBRARY): $(OBJS) $(LINKTEST_OBJ)
        $(LD) $(BREDUCE) $(ZDEFS) $(LDFLAGS) -o $@.linktest $(OBJS) $(LINKTEST_O
        rm $@.linktest
```



```
        $(LD) $(LDFLAGS) -o $@ $(OBJS)

clobber: $(TYPES:%=clobber.%)

$(TYPES:%=clobber.%):
        @CURTYPE=$(@:clobber.%=%) $(MAKE) clobber.targ

clobber.targ: clean
        -$(RM) $(CLOBBERTARGFILES)

# include library targets
include $(SRC)/lib/Makefile.targ

$(PICS): pics
$(OBJS): objs

objs/%.o pics/%.o: $(ISASRCDIR)/%.c
        $(COMPILE.c) -o $@ $<
        $(POST_PROCESS_O)

objs/%.o pics/%.o: $(ISASRCDIR)/%.s
        $(COMPILE.s) -o $@ $<
        $(POST_PROCESS_O)

objs/%.o pics/%.o: $(COMDIR)/%.c
        $(COMPILE.c) -o $@ $<
        $(POST_PROCESS_O)

# install rule for lint library target
$(ROOTLINTDIR)/%: $(COMDIR)/%
        $(INS.file)

# install rule for x86 common source
objs/%.o pics/%.o: $(SRC)/common/dis/i386/%.c
        $(COMPILE.c) -o $@ $<
        $(POST_PROCESS_O)
```

```
**********************************************************
    1162 Sun Dec 16 13:00:31 2012
new/usr/src/lib/libdisasm/amd64/Makefile
style fixes
sparc instr.c is C99
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #
  21 #
  22 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  23 # Use is subject to license terms.
  24 #
  25 # ident "%Z%%M% %I%     %E% SMI"

  26 ISASRCDIR=../$(MACH)/

  28 include ../Makefile.com
  29 include ../../Makefile.lib.64

  31 TYPES=library standalone

  33 INSTALL_DEPS_library = $(ROOTLINKS64) $(ROOTLINT64) $(ROOTLIBS64)
  34 INSTALL_DEPS_standalone = $(ROOTLIBS64)

  36 include ../Makefile.targ

  38 C99MODE = $(C99_ENABLE)
```

```
**********************************************************
    5839 Sun Dec 16 13:00:31 2012
new/usr/src/lib/libdisasm/common/dis_i386.c
take to dis and libdisasm with an axe; does not yet compile
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
 */

#include <libdisasm.h>
#include <stdlib.h>
#include <stdio.h>

#include "dis_tables.h"
#include "libdisasm_impl.h"

typedef struct dis_handle_i386 {
        int             dhx_mode;
        dis86_t         dhx_dis;
        uint64_t        dhx_end;
} dis_handle_i386_t;

/*
 * Returns true if we are near the end of a function.  This is a cheap hack at
 * detecting NULL padding between functions.  If we're within a few bytes of the
 * next function, or past the start, then return true.
 */
static int
check_func(void *data)
{
        dis_handle_t *dhp = data;
        uint64_t start;
        size_t len;

        if (dhp->dh_lookup(dhp->dh_data, dhp->dh_addr, NULL, 0, &start, &len)
            != 0)
                return (0);

        if (start < dhp->dh_addr)
                return (dhp->dh_addr > start + len - 0x10);

        return (1);
}
```



```
static int
get_byte(void *data)
{
        uchar_t byte;
        dis_handle_t *dhp = data;

        if (dhp->dh_read(dhp->dh_data, dhp->dh_addr, &byte, sizeof (byte)) !=
            sizeof (byte))
                return (-1);

        dhp->dh_addr++;

        return ((int)byte);
}

static int
do_lookup(void *data, uint64_t addr, char *buf, size_t buflen)
{
        dis_handle_t *dhp = data;

        return (dhp->dh_lookup(dhp->dh_data, addr, buf, buflen, NULL, NULL));
}

static void
dis_i386_handle_detach(dis_handle_t *dhp)
{
        dis_free(dhp->dh_arch_private, sizeof (dis_handle_i386_t));
        dhp->dh_arch_private = NULL;
}

static int
dis_i386_handle_attach(dis_handle_t *dhp)
{
        dis_handle_i386_t *dhx;

        /*
         * Validate architecture flags
         */
        if (dhp->dh_flags & ~(DIS_X86_SIZE16 | DIS_X86_SIZE32 | DIS_X86_SIZE64 |
            DIS_OCTAL | DIS_NOIMMSYM)) {
                (void) dis_seterrno(E_DIS_INVALFLAG);
                return (-1);
        }

        /*
         * Create and initialize the internal structure
         */
        if ((dhx = dis_zalloc(sizeof (dis_handle_i386_t))) == NULL) {
                (void) dis_seterrno(E_DIS_NOMEM);
                return (-1);
        }
        dhp->dh_arch_private = dhx;

        /*
         * Initialize x86-specific architecture structure
         */
        if (dhp->dh_flags & DIS_X86_SIZE16)
                dhx->dhx_mode = SIZE16;
        else if (dhp->dh_flags & DIS_X86_SIZE64)
                dhx->dhx_mode = SIZE64;
        else
                dhx->dhx_mode = SIZE32;

        if (dhp->dh_flags & DIS_OCTAL)
                dhx->dhx_dis.d86_flags = DIS_F_OCTAL;
```

```
129         dhx->dhx_dis.d86_sprintf_func = snprintf;
130         dhx->dhx_dis.d86_get_byte = get_byte;
131         dhx->dhx_dis.d86_sym_lookup = do_lookup;
132         dhx->dhx_dis.d86_check_func = check_func;

134         dhx->dhx_dis.d86_data = dhp;

136         return (0);
137 }

139 static int
140 dis_i386_disassemble(dis_handle_t *dhp, uint64_t addr, char *buf,
141     size_t buflen)
142 {
143         dis_handle_i386_t *dhx = dhp->dh_arch_private;
144         dhp->dh_addr = addr;

146         /* DIS_NOIMMSYM might not be set until now, so update */
147         if (dhp->dh_flags & DIS_NOIMMSYM)
148                 dhx->dhx_dis.d86_flags |= DIS_F_NOIMMSYM;
149         else
150                 dhx->dhx_dis.d86_flags &= ~DIS_F_NOIMMSYM;

152         if (dtrace_disx86(&dhx->dhx_dis, dhx->dhx_mode) != 0)
153                 return (-1);

155         if (buf != NULL)
156                 dtrace_disx86_str(&dhx->dhx_dis, dhx->dhx_mode, addr, buf,
157                     buflen);

159         return (0);
160 }

162 /* ARGSUSED */
163 static int
164 dis_i386_max_instrlen(dis_handle_t *dhp)
165 {
166         return (15);
167 }

169 /* ARGSUSED */
170 static int
171 dis_i386_min_instrlen(dis_handle_t *dhp)
172 {
173         return (1);
174 }

176 #define MIN(a, b)       ((a) < (b) ? (a) : (b))

178 /*
179  * Return the previous instruction.  On x86, we have no choice except to
180  * disassemble everything from the start of the symbol, and stop when we have
181  * reached our instruction address.  If we're not in the middle of a known
182  * symbol, then we return the same address to indicate failure.
183  */
184 static uint64_t
185 dis_i386_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
186 {
187         uint64_t *hist, addr, start;
188         int cur, nseen;
189         uint64_t res = pc;

191         if (n <= 0)
192                 return (pc);
```



```
194         if (dhp->dh_lookup(dhp->dh_data, pc, NULL, 0, &start, NULL) != 0 ||
195             start == pc)
196                 return (res);

198         hist = dis_zalloc(sizeof (uint64_t) * n);

200         for (cur = 0, nseen = 0, addr = start; addr < pc; addr = dhp->dh_addr) {
201                 hist[cur] = addr;
202                 cur = (cur + 1) % n;
203                 nseen++;

205                 /* if we cannot make forward progress, give up */
206                 if (dis_disassemble(dhp, addr, NULL, 0) != 0)
207                         goto done;
208         }

210         if (addr != pc) {
211                 /*
212                  * We scanned past %pc, but didn't find an instruction that
213                  * started at %pc.  This means that either the caller specified
214                  * an invalid address, or we ran into something other than code
215                  * during our scan.  Virtually any combination of bytes can be
216                  * construed as a valid Intel instruction, so any non-code bytes
217                  * we encounter will have thrown off the scan.
218                  */
219                 goto done;
220         }

222         res = hist[(cur + n - MIN(n, nseen)) % n];

224 done:
225         dis_free(hist, sizeof (uint64_t) * n);
226         return (res);
227 }

229 static int
230 dis_i386_supports_flags(int flags)
231 {
232         int archflags = flags & DIS_ARCH_MASK;

234         if (archflags == DIS_X86_SIZE16 || archflags == DIS_X86_SIZE32 ||
235             archflags == DIS_X86_SIZE64)
236                 return (1);

238         return (0);
239 }

241 dis_arch_t dis_arch_i386 = {
242         dis_i386_supports_flags,
243         dis_i386_handle_attach,
244         dis_i386_handle_detach,
245         dis_i386_disassemble,
246         dis_i386_previnstr,
247         dis_i386_min_instrlen,
248         dis_i386_max_instrlen
249 };
```

```
**********************************************************
    8876 Sun Dec 16 13:00:31 2012
new/usr/src/lib/libdisasm/common/dis_sparc.c
BE_32 wont work in libstand mode
take to dis and libdisasm with an axe; does not yet compile
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright 2007 Jason King.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
 */


#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * The sparc disassembler is mostly straightforward, each instruction is
 * represented by an inst_t structure.  The inst_t definitions are organized
 * into tables.  The tables are correspond to the opcode maps documented in the
 * various sparc architecture manuals.  Each table defines the bit range of the
 * instruction whose value act as an index into the array of instructions.  A
 * table can also refer to another table if needed.  Each table also contains
 * a function pointer of type format_fcn that knows how to output the
 * instructions in the table, as well as handle any synthetic instructions
 *
 * Unfortunately, the changes from sparcv8 -> sparcv9 not only include new
 * instructions, they sometimes renamed or just reused the same instruction to
 * do different operations (i.e. the sparcv8 coprocessor instructions).  To
 * accommodate this, each table can define an overlay table.  The overlay table
 * is a list of (table index, architecture, new instruction definition) values.
 *
 *
 * Traversal starts with the first table,
 *   get index value from the instruction
 *   if an relevant overlay entry exists for this index,
 *        grab the overlay definition
 *   else
 *        grab the definition from the array (corresponding to the index value)
 *
 * If the entry is an instruction,
```



```
 *     call print function of instruction.
 * If the entry is a pointer to another table
 *     traverse the table
 * If not valid,
 *     return an error
 *
 *
 * To keep dis happy, for sparc, instead of actually returning an error, if
 * the instruction cannot be disassembled, we instead merely place the value
 * of the instruction into the output buffer.
 *
 * Adding new instructions:
 *
 * With the above information, it hopefully makes it clear how to add support
 * for decoding new instructions.  Presumably, with new instructions will come
 * a new dissassembly mode (I.e. DIS_SPARC_V8, DIS_SPARC_V9, etc.).
 *
 * If the dissassembled format does not correspond to one of the existing
 * formats, a new formatter will have to be written.  The 'flags' value of
 * inst_t is intended to instruct the corresponding formatter about how to
 * output the instruction.
 *
 * If the corresponding entry in the correct table is currently unoccupied,
 * simply replace the INVALID entry with the correct definition.  The INST and
 * TABLE macros are suggested to be used for this.  If there is already an
 * instruction defined, then the entry must be placed in an overlay table.  If
 * no overlay table exists for the instruction table, one will need to be
 * created.
 */

#include <libdisasm.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/byteorder.h>
#include <string.h>

#include "libdisasm_impl.h"
#include "dis_sparc.h"

static const inst_t *dis_get_overlay(dis_handle_t *, const table_t *,
    uint32_t);
static uint32_t dis_get_bits(uint32_t, int, int);

#if !defined(DIS_STANDALONE)
static void do_binary(uint32_t);
#endif /* DIS_STANDALONE */

static void
dis_sparc_handle_detach(dis_handle_t *dhp)
dis_handle_t *
dis_handle_create(int flags, void *data, dis_lookup_f lookup_func,
    dis_read_f read_func)
{
        dis_free(dhp->dh_arch_private, sizeof (dis_handle_sparc_t));
        dhp->dh_arch_private = NULL;
}

static int
dis_sparc_handle_attach(dis_handle_t *dhp)
{
        dis_handle_sparc_t *dhx;

#if !defined(DIS_STANDALONE)
        char *opt = NULL;
        char *opt2, *save, *end;
```

```
#endif
        dis_handle_t *dhp;

        /* Validate architecture flags */
        if ((dhp->dh_flags & (DIS_SPARC_V8|DIS_SPARC_V9|DIS_SPARC_V9_SGI))
            == 0) {
        if ((flags & (DIS_SPARC_V8|DIS_SPARC_V9|DIS_SPARC_V9_SGI)) == 0) {
                (void) dis_seterrno(E_DIS_INVALFLAG);
                return (-1);
                return (NULL);
        }

        if ((dhx = dis_zalloc(sizeof (dis_handle_sparc_t))) == NULL) {
        if ((dhp = dis_zalloc(sizeof (struct dis_handle))) == NULL) {
                (void) dis_seterrno(E_DIS_NOMEM);
                return (NULL);
        }
        dhx->dhx_debug = DIS_DEBUG_COMPAT;
        dhp->dh_arch_private = dhx;

        dhp->dh_lookup = lookup_func;
        dhp->dh_read = read_func;
        dhp->dh_flags = flags;
        dhp->dh_data = data;
        dhp->dh_debug = DIS_DEBUG_COMPAT;

#if !defined(DIS_STANDALONE)

        opt = getenv("_LIBDISASM_DEBUG");
        if (opt == NULL)
                return (0);
                return (dhp);

        opt2 = strdup(opt);
        if (opt2 == NULL) {
                dis_handle_destroy(dhp);
                dis_free(dhx, sizeof (dis_handle_sparc_t));
                (void) dis_seterrno(E_DIS_NOMEM);
                return (-1);
                return (NULL);
        }
        save = opt2;

        while (opt2 != NULL) {
                end = strchr(opt2, ',');

                if (end != 0)
                        *end++ = '\0';

                if (strcasecmp("synth-all", opt2) == 0)
                        dhx->dhx_debug |= DIS_DEBUG_SYN_ALL;
                        dhp->dh_debug |= DIS_DEBUG_SYN_ALL;

                if (strcasecmp("compat", opt2) == 0)
                        dhx->dhx_debug |= DIS_DEBUG_COMPAT;
                        dhp->dh_debug |= DIS_DEBUG_COMPAT;

                if (strcasecmp("synth-none", opt2) == 0)
                        dhx->dhx_debug &= ~(DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT);
                        dhp->dh_debug &= ~(DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT);

                if (strcasecmp("binary", opt2) == 0)
                        dhx->dhx_debug |= DIS_DEBUG_PRTBIN;
                        dhp->dh_debug |= DIS_DEBUG_PRTBIN;

                if (strcasecmp("format", opt2) == 0)
```



```
                        dhx->dhx_debug |= DIS_DEBUG_PRTFMT;
                        dhp->dh_debug |= DIS_DEBUG_PRTFMT;

                if (strcasecmp("all", opt2) == 0)
                        dhx->dhx_debug = DIS_DEBUG_ALL;
                        dhp->dh_debug = DIS_DEBUG_ALL;

                if (strcasecmp("none", opt2) == 0)
                        dhx->dhx_debug = DIS_DEBUG_NONE;
                        dhp->dh_debug = DIS_DEBUG_NONE;

                opt2 = end;
        }
        free(save);
#endif /* DIS_STANDALONE */
        return (0);
        return (dhp);
}

/* ARGSUSED */
static int
dis_sparc_max_instrlen(dis_handle_t *dhp)
void
dis_handle_destroy(dis_handle_t *dhp)
{
        return (4);
        dis_free(dhp, sizeof (dis_handle_t));
}

void
dis_set_data(dis_handle_t *dhp, void *data)
{
        dhp->dh_data = data;
}

void
dis_flags_set(dis_handle_t *dhp, int f)
{
        dhp->dh_flags |= f;
}

void
dis_flags_clear(dis_handle_t *dhp, int f)
{
        dhp->dh_flags &= ~f;
}

/* ARGSUSED */
static int
dis_sparc_min_instrlen(dis_handle_t *dhp)
int
dis_max_instrlen(dis_handle_t *dhp)
{
        return (4);
}

/*
 * The dis_i386.c comment for this says it returns the previous instruction,
 * however, I'm fairly sure it's actually returning the _address_ of the
 * nth previous instruction.
 */
/* ARGSUSED */
static uint64_t
dis_sparc_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
uint64_t
dis_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
```

```
209 {
210         if (n <= 0)
211                 return (pc);

213         if (pc < n)
214                 return (pc);

216         return (pc - n*4);
217 }

219 static int
220 dis_sparc_disassemble(dis_handle_t *dhp, uint64_t addr, char *buf,
221     size_t buflen)
232 int
233 dis_disassemble(dis_handle_t *dhp, uint64_t addr, char *buf, size_t buflen)
222 {
223         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
224         const table_t *tp = &initial_table;
225         const inst_t *inp = NULL;

227         uint32_t instr;
228         uint32_t idx = 0;

230         if (dhp->dh_read(dhp->dh_data, addr, &instr, sizeof (instr)) !=
231             sizeof (instr))
232                 return (-1);

234         dhx->dhx_buf    = buf;
235         dhx->dhx_buflen = buflen;
245         dhp->dh_buf    = buf;
246         dhp->dh_buflen = buflen;
236         dhp->dh_addr    = addr;

238         buf[0] = '\0';

240         /* this allows sparc code to be tested on x86 */
241 #if !defined(DIS_STANDALONE)
242         instr = BE_32(instr);
243 #endif /* DIS_STANDALONE */

245 #if !defined(DIS_STANDALONE)
246         if ((dhx->dhx_debug & DIS_DEBUG_PRTBIN) != 0)
255         if ((dhp->dh_debug & DIS_DEBUG_PRTBIN) != 0)
247                 do_binary(instr);
248 #endif /* DIS_STANDALONE */

250         /* CONSTCOND */
251         while (1) {
252                 idx = dis_get_bits(instr, tp->tbl_field, tp->tbl_len);
253                 inp = &tp->tbl_inp[idx];

255                 inp = dis_get_overlay(dhp, tp, idx);

257                 if ((inp->in_type == INST_NONE) ||
258                     ((inp->in_arch & dhp->dh_flags) == 0))
259                         goto error;

261                 if (inp->in_type == INST_TBL) {
262                         tp = inp->in_data.in_tbl;
263                         continue;
264                 }

266                 break;
267         }

269         if (tp->tbl_fmt(dhp, instr, inp, idx) == 0)
```



```
270                 return (0);

272 error:

274         (void) snprintf(buf, buflen,
275             ((dhp->dh_flags & DIS_OCTAL) != 0) ? "0%011lo" : "0x%08lx",
276             instr);

278         return (0);
279 }
```

unchanged_portion_omitted

```
330 #endif /* DIS_STANDALONE */

332 static int
333 dis_sparc_supports_flags(int flags)
334 {
335         int archflags = flags & DIS_ARCH_MASK;

337         if (archflags == DIS_SPARC_V8 ||
338             (archflags & (DIS_SPARC_V9 | DIS_SPARC_V8)) == DIS_SPARC_V9)
339                 return (1);

341         return (0);
342 }

344 const dis_arch_t dis_arch_sparc = {
345         dis_sparc_supports_flags,
346         dis_sparc_handle_attach,
347         dis_sparc_handle_detach,
348         dis_sparc_disassemble,
349         dis_sparc_previnstr,
350         dis_sparc_min_instrlen,
351         dis_sparc_max_instrlen
352 };
```

```
**********************************************************
    2261 Sun Dec 16 13:00:31 2012
new/usr/src/lib/libdisasm/common/dis_sparc.h
style fixes
take to dis and libdisasm with an axe; does not yet compile
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  */

  27 /*
  28  * Copyright 2007 Jason King.  All rights reserved.
  29  * Use is subject to license terms.
  30  */


  33 #ifndef _DIS_SPARC_H
  34 #define _DIS_SPARC_H

  36 #pragma ident   "%Z%%M% %I%     %E% SMI"

  36 #ifdef  __cplusplus
  37 extern "C" {
  38 #endif

  40 #include <sys/types.h>

  42 #define DIS_DEBUG_NONE          0x00L
  43 #define DIS_DEBUG_COMPAT        0x01L
  44 #define DIS_DEBUG_SYN_ALL       0x02L
  45 #define DIS_DEBUG_PRTBIN        0x04L
  46 #define DIS_DEBUG_PRTFMT        0x08L

  48 #define DIS_DEBUG_ALL DIS_DEBUG_SYN_ALL|DIS_DEBUG_PRTBIN|DIS_DEBUG_PRTFMT

  50 typedef struct dis_handle_sparc {
  51         char            *dhx_buf;
  52         size_t          dhx_buflen;
  53         int             dhx_debug;
  54 } dis_handle_sparc_t;
  52 struct dis_handle {
  53         void            *dh_data;
  54         dis_lookup_f    dh_lookup;
  55         dis_read_f      dh_read;
```



```
  56         int             dh_flags;

  58         char            *dh_buf;
  59         size_t          dh_buflen;
  60         uint64_t        dh_addr;
  61         int             dh_debug;
  62 };

  56 /* different types of things we can have in inst_t */
  57 #define INST_NONE       0x00
  58 #define INST_DEF        0x01
  59 #define INST_TBL        0x02

  61 struct inst;
  62 struct overlay;

  64 typedef struct inst inst_t;
  65 typedef struct overlay overlay_t;

  67 typedef int (*format_fcn)(dis_handle_t *, uint32_t, const inst_t *, int);

  69 typedef struct table {
  70         const struct inst       *tbl_inp;
  71         const struct overlay    *tbl_ovp;
  72         format_fcn              tbl_fmt;
  73         uint32_t                tbl_field;
  74         uint32_t                tbl_len;
  75 } table_t;
______unchanged_portion_omitted_
```

```
**********************************************************
   60194 Sun Dec 16 13:00:31 2012
new/usr/src/lib/libdisasm/common/dis_sparc_fmt.c
style fixes
comments; lint
take to dis and libdisasm with an axe; does not yet compile
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright 2009 Jason King.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
 */


#include <sys/byteorder.h>
#include <stdarg.h>

#if !defined(DIS_STANDALONE)
#include <stdio.h>
#endif /* DIS_STANDALONE */

#include "libdisasm.h"
#include "libdisasm_impl.h"
#include "dis_sparc.h"
#include "dis_sparc_fmt.h"

extern char *strncpy(char *, const char *, size_t);
extern size_t strlen(const char *);
extern int strcmp(const char *, const char *);
extern int strncmp(const char *, const char *, size_t);
extern size_t strlcat(char *, const char *, size_t);
extern size_t strlcpy(char *, const char *, size_t);
extern int snprintf(char *, size_t, const char *, ...);
extern int vsnprintf(char *, size_t, const char *, va_list);

/*
 * This file has the functions that do all the dirty work of outputting the
 * disassembled instruction
 *
 * All the non-static functions follow the format_fcn (in dis_sparc.h):
```



```
 * Input:
 *      disassembler handle/context
 *      instruction to disassemble
 *      instruction definition pointer (inst_t *)
 *      index in the table of the instruction
 * Return:
 *      0 Success
 *    !0 Invalid instruction
 *
 * Generally, instructions found in the same table use the same output format
 * or have a few minor differences (which are described in the 'flags' field
 * of the instruction definition. In some cases, certain instructions differ
 * radically enough from those in the same table, that their own format
 * function is used.
 *
 * Typically each table has a unique format function defined in this file.  In
 * some cases (such as branches) a common one for all the tables is used.
 *
 * When adding support for new instructions, it is largely a judgement call
 * as to when a new format function is defined.
 */

/* The various instruction formats of a sparc instruction */

#if defined(_BIT_FIELDS_HTOL)
typedef struct format1 {
        uint32_t op:2;
        uint32_t disp30:30;
} format1_t;
_____unchanged_portion_omitted_
#endif /* DIS_STANDALONE */


/*
 * print out a call instruction
 * format: call address  <name>
 */
/* ARGSUSED1 */
int
fmt_call(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
{
        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
        ifmt_t *f = (ifmt_t *)&instr;

        int32_t disp;
        size_t curlen;

        int octal = ((dhp->dh_flags & DIS_OCTAL) != 0);

        if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
        if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
                prt_field("op", f->f1.op, 2);
                prt_field("disp30", f->f1.disp30, 30);
        }

        disp = sign_extend(f->f1.disp30, 30) * 4;

        prt_name(dhp, inp->in_data.in_def.in_name, 1);

        bprintf(dhp, (octal != 0) ? "%s0%-11lo" : "%s0x%-10lx",
            (disp < 0) ? "-" : "+",
            (disp < 0) ? (-disp) : disp);

        (void) strlcat(dhx->dhx_buf, " <", dhx->dhx_buflen);
        (void) strlcat(dhp->dh_buf, " <", dhp->dh_buflen);
```

```
725         curlen = strlen(dhx->dhx_buf);
723         curlen = strlen(dhp->dh_buf);
726         dhp->dh_lookup(dhp->dh_data, dhp->dh_addr + (int64_t)disp,
727             dhx->dhx_buf + curlen, dhx->dhx_buflen - curlen - 1, NULL,
725             dhp->dh_buf + curlen, dhp->dh_buflen - curlen - 1, NULL,
728             NULL);
729         (void) strlcat(dhx->dhx_buf, ">", dhx->dhx_buflen);
727         (void) strlcat(dhp->dh_buf, ">", dhp->dh_buflen);


732         return (0);
733 }

735 int
736 fmt_sethi(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
737 {
738         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
739         ifmt_t *f = (ifmt_t *)&instr;

741         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
738         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
742                 prt_field("op", f->f2.op, 2);
743                 prt_field("op2", f->f2.op2, 3);
744                 prt_field("rd", f->f2.rd, 5);
745                 prt_field("imm22", f->f2.imm22, 22);
746         }

748         if (idx == 0) {
749                 /* unimp / illtrap */
750                 prt_name(dhp, inp->in_data.in_def.in_name, 1);
751                 prt_imm(dhp, f->f2.imm22, 0);
752                 return (0);
753         }

755         if (f->f2.imm22 == 0 && f->f2.rd == 0) {
756                 prt_name(dhp, "nop", 0);
757                 return (0);
758         }

760         /* ?? Should we return -1 if rd == 0 && disp != 0 */

762         prt_name(dhp, inp->in_data.in_def.in_name, 1);

764         bprintf(dhp,
765             ((dhp->dh_flags & DIS_OCTAL) != 0) ?
766             "%%hi(0%lo), %s" : "%%hi(0x%lx), %s",
767             f->f2.imm22 << 10,
768             reg_names[f->f2.rd]);

770         return (0);
771 }

773 /* ARGSUSED3 */
774 int
775 fmt_branch(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
776 {
777         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
778         const char *name = inp->in_data.in_def.in_name;
779         const char *r = NULL;
780         const char *annul = "";
781         const char *pred  = "";

783         char buf[15];

785         ifmt_t *f = (ifmt_t *)&instr;
```



```
787         size_t curlen;
788         int32_t disp;
789         uint32_t flags = inp->in_data.in_def.in_flags;
790         int octal = ((dhp->dh_flags & DIS_OCTAL) != 0);

792         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
788         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
793                 prt_field("op", f->f2.op, 2);
794                 prt_field("op2", f->f2.op2, 3);

796                 switch (FLG_DISP_VAL(flags)) {
797                 case DISP22:
798                         prt_field("cond", f->f2a.cond, 4);
799                         prt_field("a", f->f2a.a, 1);
800                         prt_field("disp22", f->f2a.disp22, 22);
801                         break;

803                 case DISP19:
804                         prt_field("cond", f->f2a.cond, 4);
805                         prt_field("a", f->f2a.a, 1);
806                         prt_field("p", f->f2b.p, 1);
807                         prt_field("cc", f->f2b.cc, 2);
808                         prt_field("disp19", f->f2b.disp19, 19);
809                         break;

811                 case DISP16:
812                         prt_field("bit 28", ((instr & (1L << 28)) >> 28), 1);
813                         prt_field("rcond", f->f2c.cond, 3);
814                         prt_field("p", f->f2c.p, 1);
815                         prt_field("rs1", f->f2c.rs1, 5);
816                         prt_field("d16hi", f->f2c.d16hi, 2);
817                         prt_field("d16lo", f->f2c.d16lo, 14);
818                         break;
819                 }
820         }

822         if (f->f2b.op2 == 0x01 && idx == 0x00 && f->f2b.p == 1 &&
823             f->f2b.cc == 0x02 && ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) != 0)) {
819             f->f2b.cc == 0x02 && ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) != 0)) {
824                 name = "iprefetch";
825                 flags = FLG_RS1(REG_NONE)|FLG_DISP(DISP19);
826         }


829         switch (FLG_DISP_VAL(flags)) {
830         case DISP22:
831                 disp = sign_extend(f->f2a.disp22, 22);
832                 break;

834         case DISP19:
835                 disp = sign_extend(f->f2b.disp19, 19);
836                 break;

838         case DISP16:
839                 disp = sign_extend((f->f2c.d16hi << 14)|f->f2c.d16lo, 16);
840                 break;

842         }

844         disp *= 4;

846         if ((FLG_RS1_VAL(flags) == REG_ICC) || (FLG_RS1_VAL(flags) == REG_FCC))
847                 r = get_regname(dhp, FLG_RS1_VAL(flags), f->f2b.cc);
848         else
849                 r = get_regname(dhp, FLG_RS1_VAL(flags), f->f2c.rs1);
```

```
851         if (r == NULL)
852                 return (-1);
853
854         if (f->f2a.a == 1)
855                 annul = ",a";
856
857         if ((flags & FLG_PRED) != 0) {
858                 if (f->f2b.p == 0) {
859                         pred = ",pn";
860                 } else {
861                         if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0)
857                         if ((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0)
862                                 pred = ",pt";
863                 }
864         }
865
866         (void) snprintf(buf, sizeof (buf), "%s%s%s", name, annul, pred);
867         prt_name(dhp, buf, 1);
868
869
870         switch (FLG_DISP_VAL(flags)) {
871         case DISP22:
872                 bprintf(dhp,
873                     (octal != 0) ? "%s0%-11lo <" : "%s0x%-10lx <",
874                     (disp < 0) ? "-" : "+",
875                     (disp < 0) ? (-disp) : disp);
876                 break;
877
878         case DISP19:
879                 bprintf(dhp,
880                     (octal != 0) ? "%s, %s0%-5lo <" :
881                     "%s, %s0x%-04lx <", r,
882                     (disp < 0) ? "-" : "+",
883                     (disp < 0) ? (-disp) : disp);
884                 break;
885
886         case DISP16:
887                 bprintf(dhp,
888                     (octal != 0) ? "%s, %s0%-6lo <" : "%s, %s0x%-5lx <",
889                     r,
890                     (disp < 0) ? "-" : "+",
891                     (disp < 0) ? (-disp) : disp);
892                 break;
893         }
894
895         curlen = strlen(dhx->dhx_buf);
891         curlen = strlen(dhp->dh_buf);
896         dhp->dh_lookup(dhp->dh_data, dhp->dh_addr + (int64_t)disp,
897             dhx->dhx_buf + curlen, dhx->dhx_buflen - curlen - 1, NULL, NULL);
893             dhp->dh_buf + curlen, dhp->dh_buflen - curlen - 1, NULL, NULL);
894
899         (void) strlcat(dhx->dhx_buf, ">", dhx->dhx_buflen);
895         (void) strlcat(dhp->dh_buf, ">", dhp->dh_buflen);
896
901         return (0);
902 }
903
904
905
906 /*
907  * print out the compare and swap instructions (casa/casxa)
908  * format: casa/casxa [%rs1] imm_asi, %rs2, %rd
909  *          casa/casxa [%rs1] %asi, %rs2, %rd
910  *
911  * If DIS_DEBUG_SYN_ALL is set, synthetic instructions are emitted
912  * when an immediate ASI value is given as follows:
```



```
913  *
914  * casa  [%rs1]#ASI_P, %rs2, %rd    -> cas   [%rs1], %rs2, %rd
915  * casa  [%rs1]#ASI_P_L, %rs2, %rd  -> casl  [%rs1], %rs2, %rd
916  * casxa [%rs1]#ASI_P, %rs2, %rd    -> casx  [%rs1], %rs2, %rd
917  * casxa [%rs1]#ASI_P_L, %rs2, %rd  -> casxl [%rs1], %rs2, %rd
918  */
919 static int
920 fmt_cas(dis_handle_t *dhp, uint32_t instr, const char *name)
921 {
922         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
923         ifmt_t *f = (ifmt_t *)&instr;
924         const char *asistr = NULL;
925         int noasi = 0;
926
927         asistr = get_asi_name(f->f3.asi);
928
929         if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT)) != 0) {
924         if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT)) != 0) {
930                 if (f->f3.op3 == 0x3c && f->f3.i == 0) {
931                         if (f->f3.asi == 0x80) {
932                                 noasi = 1;
933                                 name = "cas";
934                         }
935
936                         if (f->f3.asi == 0x88) {
937                                 noasi = 1;
938                                 name = "casl";
939                         }
940                 }
941
942                 if (f->f3.op3 == 0x3e && f->f3.i == 0) {
943                         if (f->f3.asi == 0x80) {
944                                 noasi = 1;
945                                 name = "casx";
946                         }
947
948                         if (f->f3.asi == 0x88) {
949                                 noasi = 1;
950                                 name = "casxl";
951                         }
952                 }
953         }
954
955         prt_name(dhp, name, 1);
956
957         bprintf(dhp, "[%s]", reg_names[f->f3.rs1]);
958
959         if (noasi == 0) {
960                 (void) strlcat(dhx->dhx_buf, " ", dhx->dhx_buflen);
955                 (void) strlcat(dhp->dh_buf, " ", dhp->dh_buflen);
961                 prt_asi(dhp, instr);
962         }
963
964         bprintf(dhp, ", %s, %s", reg_names[f->f3.rs2], reg_names[f->f3.rd]);
965
966         if (noasi == 0 && asistr != NULL)
967                 bprintf(dhp, "\t<%s>", asistr);
968
969         return (0);
970 }
971
972 /*
973  * format a load/store instruction
974  * format: ldXX [%rs1 + %rs2], %rd        load, i==0
975  *          ldXX [%rs1 +/- nn], %rd       load, i==1
976  *          ldXX [%rs1 + %rs2] #XX, %rd   load w/ imm_asi, i==0
```

```
 977  *          ldXX [%rs1 +/- nn] %asi, %rd  load from asi[%asi], i==1
 978  *
 979  *          stXX %rd, [%rs1 + %rs2]       store, i==0
 980  *          stXX %rd, [%rs1 +/- nn]       store, i==1
 981  *          stXX %rd, [%rs1 + %rs1] #XX   store to imm_asi, i==0
 982  *          stXX %rd, [%rs1 +/-nn] %asi   store to asi[%asi], i==1
 983  *
 984  * The register sets used for %rd are set in the instructions flags field
 985  * The asi variants are used if FLG_ASI is set in the instructions flags field
 986  *
 987  * If DIS_DEBUG_SYNTH_ALL or DIS_DEBUG_COMPAT are set,
 988  * When %rs1, %rs2 or nn are 0, they are not printed, i.e.
 989  * [ %rs1 + 0x0 ], %rd -> [%rs1], %rd for example
 990  *
 991  * The following synthetic instructions are also implemented:
 992  *
 993  * stb %g0, [addr] -> clrb [addr]    DIS_DEBUG_SYNTH_ALL
 994  * sth %g0, [addr] -> crlh [addr]    DIS_DEBUG_SYNTH_ALL
 995  * stw %g0, [addr] -> clr  [addr]    DIS_DEBUG_SYNTH_ALL|DIS_DEBUG_COMPAT
 996  * stx %g0, [addr] -> clrx [addr]    DIS_DEBUG_SYNTH_ALL
 997  *
 998  * If DIS_DEBUG_COMPAT is set, the following substitutions also take place
 999  *      lduw -> ld
1000  *      ldtw -> ld
1001  *      stuw -> st
1002  *      sttw -> st
1003  */
1004 int
1005 fmt_ls(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1006 {
1007         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1008         ifmt_t *f = (ifmt_t *)&instr;
1009         const char *regstr = NULL;
1010         const char *asistr = NULL;

1012         const char *iname = inp->in_data.in_def.in_name;
1013         uint32_t flags = inp->in_data.in_def.in_flags;

1015         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1009         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
1016                 prt_field("op", f->f3.op, 2);
1017                 prt_field("op3", f->f3.op3, 6);
1018                 prt_field("rs1", f->f3.rs1, 5);
1019                 prt_field("i", f->f3.i, 1);
1020                 if (f->f3.i != 0) {
1021                         prt_field("simm13", f->f3a.simm13, 13);
1022                 } else {
1023                         if ((flags & FLG_ASI) != 0)
1024                                 prt_field("imm_asi", f->f3.asi, 8);
1025                         prt_field("rs2", f->f3.rs2, 5);
1026                 }
1027                 prt_field("rd", f->f3.rd, 5);
1028         }

1030         if (idx == 0x2d || idx == 0x3d) {
1031                 /* prefetch / prefetcha */

1033                 prt_name(dhp, iname, 1);

1035                 prt_address(dhp, instr, 0);

1037                 if (idx == 0x3d) {
1038                         (void) strlcat(dhx->dhx_buf, " ", dhx->dhx_buflen);
1032                         (void) strlcat(dhp->dh_buf, " ", dhp->dh_buflen);
1039                         prt_asi(dhp, instr);
1040                 }
```



```
1042                 (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
1036                 (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);

1044                 /* fcn field is the same as rd */
1045                 if (prefetch_str[f->f3.rd] != NULL)
1046                         (void) strlcat(dhx->dhx_buf, prefetch_str[f->f3.rd],
1047                             dhx->dhx_buflen);
1040                         (void) strlcat(dhp->dh_buf, prefetch_str[f->f3.rd],
1041                             dhp->dh_buflen);
1048                 else
1049                         prt_imm(dhp, f->f3.rd, 0);

1051                 if (idx == 0x3d && f->f3.i == 0) {
1052                         asistr = get_asi_name(f->f3.asi);
1053                         if (asistr != NULL)
1054                                 bprintf(dhp, "\t<%s>", asistr);
1055                 }

1057                 return (0);
1058         }

1060         /* casa / casxa */
1061         if (idx == 0x3c || idx == 0x3e)
1062                 return (fmt_cas(dhp, instr, iname));

1064         /* synthetic instructions & special cases */
1065         switch (idx) {
1066         case 0x00:
1067                 /* ld */
1068                 if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
1062                 if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
1069                         iname = "lduw";
1070                 break;

1072         case 0x03:
1073                 if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
1067                 if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
1074                         iname = "ldtw";
1075                 break;

1077         case 0x04:
1078                 /* stw */
1079                 if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
1073                 if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
1080                         iname = "stuw";

1082                 if ((dhp->dh_flags & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1083                     == 0)
1084                         break;

1086                 if (f->f3.rd == 0) {
1087                         iname = "clr";
1088                         flags = FLG_RD(REG_NONE);
1089                 }
1090                 break;

1092         case 0x05:
1093                 /* stb */
1094                 if ((dhp->dh_flags & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1095                     == 0)
1096                         break;

1098                 if (f->f3.rd == 0) {
1099                         iname = "clrb";
1100                         flags = FLG_RD(REG_NONE);
```

```
1101                }
1102                break;

1104        case 0x06:
1105                /* sth */
1106                if ((dhp->dh_flags & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1107                    == 0)
1108                        break;

1110                if (f->f3.rd == 0) {
1111                        iname = "clrh";
1112                        flags = FLG_RD(REG_NONE);
1113                }
1114                break;

1116        case 0x07:
1117                if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
1111                if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
1118                        iname = "sttw";
1119                break;

1121        case 0x0e:
1122                /* stx */

1124                if ((dhp->dh_flags & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1125                    == 0)
1126                        break;

1128                if (f->f3.rd == 0) {
1129                        iname = "clrx";
1130                        flags = FLG_RD(REG_NONE);
1131                }
1132                break;

1134        case 0x13:
1135                /* ldtwa */
1136                if (((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0) &&
1130                if (((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0) &&
1137                    ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0))
1138                        iname = "ldtwa";
1139                break;

1141        case 0x17:
1142                /* sttwa */
1143                if (((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0) &&
1137                if (((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0) &&
1144                    ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0))
1145                        iname = "sttwa";
1146                break;

1148        case 0x21:
1149        case 0x25:
1150                /*
1151                 * on sparcv8 it merely says that rd != 1 should generate an
1152                 * exception, on v9, it is illegal
1153                 */
1154                if ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) == 0)
1155                        break;

1157                iname = (idx == 0x21) ? "ldx" : "stx";

1159                if (f->f3.rd > 1)
1160                        return (-1);

1162                break;
```



```
1164        case 0x31:
1165                /* stda */
1166                switch (f->f3.asi) {
1167                        case 0xc0:
1168                        case 0xc1:
1169                        case 0xc8:
1170                        case 0xc9:
1171                        case 0xc2:
1172                        case 0xc3:
1173                        case 0xca:
1174                        case 0xcb:
1175                        case 0xc4:
1176                        case 0xc5:
1177                        case 0xcc:
1178                        case 0xcd:
1179                                /*
1180                                 * store partial floating point, only valid w/
1181                                 * vis
1182                                 *
1183                                 * Somewhat confusingly, it uses the same op
1184                                 * code as 'stda' -- store double to alternate
1185                                 * space.  It is distinguised by specific
1186                                 * imm_asi values (as seen above), and
1187                                 * has a slightly different output syntax
1188                                 */

1190                                if ((dhp->dh_flags & DIS_SPARC_V9_SGI) == 0)
1191                                        break;
1192                                if (f->f3.i != 0)
1193                                        break;
1194                                prt_name(dhp, iname, 1);
1195                                bprintf(dhp, "%s, %s, [%s] ",
1196                                    get_regname(dhp, REG_FPD, f->f3.rd),
1197                                    get_regname(dhp, REG_FPD, f->f3.rs2),
1198                                    get_regname(dhp, REG_FPD, f->f3.rs1));
1199                                prt_asi(dhp, instr);
1200                                asistr = get_asi_name(f->f3.asi);
1201                                if (asistr != NULL)
1202                                        bprintf(dhp, "\t<%s>", asistr);

1204                                return (0);

1206                        default:
1207                                break;
1208                }

1210        }

1212        regstr = get_regname(dhp, FLG_RD_VAL(flags), f->f3.rd);

1214        if (f->f3.i == 0)
1215                asistr = get_asi_name(f->f3.asi);

1217        prt_name(dhp, iname, 1);

1219        if ((flags & FLG_STORE) != 0) {
1220                if (regstr[0] != '\0') {
1221                        (void) strlcat(dhx->dhx_buf, regstr, dhx->dhx_buflen);
1222                        (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
1215                        (void) strlcat(dhp->dh_buf, regstr, dhp->dh_buflen);
1216                        (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
1223                }

1225                prt_address(dhp, instr, 0);
1226                if ((flags & FLG_ASI) != 0) {
1227                        (void) strlcat(dhx->dhx_buf, " ", dhx->dhx_buflen);
```

```
1221                        (void) strlcat(dhp->dh_buf, " ", dhp->dh_buflen);
1228                        prt_asi(dhp, instr);
1229                }
1230        } else {
1231                prt_address(dhp, instr, 0);
1232                if ((flags & FLG_ASI) != 0) {
1233                        (void) strlcat(dhx->dhx_buf, " ", dhx->dhx_buflen);
1227                        (void) strlcat(dhp->dh_buf, " ", dhp->dh_buflen);
1234                        prt_asi(dhp, instr);
1235                }

1237                if (regstr[0] != '\0') {
1238                        (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
1239                        (void) strlcat(dhx->dhx_buf, regstr, dhx->dhx_buflen);
1232                        (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
1233                        (void) strlcat(dhp->dh_buf, regstr, dhp->dh_buflen);
1240                }
1241        }

1243        if ((flags & FLG_ASI) != 0 && asistr != NULL)
1244                bprintf(dhp, "\t<%s>", asistr);

1246        return (0);
1247 }

1249 static int
1250 fmt_cpop(dis_handle_t *dhp, uint32_t instr, const inst_t *inp)
1251 {
1252        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1253        ifmt_t *f = (ifmt_t *)&instr;
1254        int flags = FLG_P1(REG_CP)|FLG_P2(REG_CP)|FLG_NOIMM|FLG_P3(REG_CP);

1256        if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1249        if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
1257                prt_field("op", f->fcp.op, 2);
1258                prt_field("op3", f->fcp.op3, 6);
1259                prt_field("opc", f->fcp.opc, 9);
1260                prt_field("rs1", f->fcp.rs1, 5);
1261                prt_field("rs2", f->fcp.rs2, 5);
1262                prt_field("rd", f->fcp.rd, 5);
1263        }

1265        prt_name(dhp, inp->in_data.in_def.in_name, 1);
1266        prt_imm(dhp, f->fcp.opc, 0);

1268        (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
1261        (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
1269        (void) prt_aluargs(dhp, instr, flags);

1271        return (0);
1272 }

1274 static int
1275 dis_fmt_rdwr(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1276 {
1277        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1278        const char *psr_str = "%psr";
1279        const char *wim_str = "%wim";
1280        const char *tbr_str = "%tbr";

1282        const char *name = inp->in_data.in_def.in_name;
1283        const char *regstr = NULL;

1285        ifmt_t *f = (ifmt_t *)&instr;

1287        int rd = (idx < 0x30);
```



```
1288        int v9 = (dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI));
1289        int ridx = f->f3.rs1;
1290        int i, first;
1291        int pr_rs1 = 1;
1292        int pr_rs2 = 1;

1294        int use_mask = 1;
1295        uint32_t mask;

1297        if (rd == 0)
1298                ridx = f->f3.rd;

1300        switch (idx) {
1301        case 0x28:
1302                /* rd */

1304                /* stbar */
1305                if ((f->f3.rd == 0) && (f->f3.rs1 == 15) && (f->f3.i == 0)) {
1306                        prt_name(dhp, "stbar", 0);
1307                        return (0);
1308                }

1310                /* membar */
1311                if ((v9 != 0) && (f->f3.rd == 0) && (f->f3.rs1 == 15) &&
1312                    (f->f3.i == 1) && ((f->i & (1L << 12)) == 0)) {

1314                        prt_name(dhp, "membar",
1315                            ((f->fmb.cmask != 0) || (f->fmb.mmask != 0)));

1317                        first = 0;

1319                        for (i = 0; i < 4; ++i) {
1320                                if ((f->fmb.cmask & (1L << i)) != 0) {
1321                                        bprintf(dhp, "%s%s",
1322                                            (first != 0) ? "|" : "",
1323                                            membar_cmask[i]);
1324                                        first = 1;
1325                                }
1326                        }

1328                        for (i = 0; i < 5; ++i) {
1329                                if ((f->fmb.mmask & (1L << i)) != 0) {
1330                                        bprintf(dhp, "%s%s",
1331                                            (first != 0) ? "|" : "",
1332                                            membar_mmask[i]);
1333                                        first = 1;
1334                                }
1335                        }

1337                        return (0);
1338                }

1340                if (v9 != 0) {
1341                        regstr = v9_asr_names[ridx];
1342                        mask = v9_asr_rdmask;
1343                } else {
1344                        regstr = asr_names[ridx];
1345                        mask = asr_rdmask;
1346                }
1347                break;

1349        case 0x29:
1350                if (v9 != 0) {
1351                        regstr = v9_hprivreg_names[ridx];
1352                        mask = v9_hpr_rdmask;
1353                } else {
```

```
1354                        regstr = psr_str;
1355                        use_mask = 0;
1356                }
1357                break;

1359        case 0x2a:
1360                if (v9 != 0) {
1361                        regstr = v9_privreg_names[ridx];
1362                        mask = v9_pr_rdmask;
1363                } else {
1364                        regstr = wim_str;
1365                        use_mask = 0;
1366                }
1367                break;

1369        case 0x2b:
1370                if (v9 != 0) {
1371                        /* flushw */
1372                        prt_name(dhp, name, 0);
1373                        return (0);
1374                }

1376                regstr = tbr_str;
1377                use_mask = 0;
1378                break;

1380        case 0x30:
1381                if (v9 != 0) {
1382                        regstr = v9_asr_names[ridx];
1383                        mask = v9_asr_wrmask;
1384                } else {
1385                        regstr = asr_names[ridx];
1386                        mask = asr_wrmask;
1387                }

1389                /*
1390                 * sir is shoehorned in here, per Ultrasparc 2007
1391                 * hyperprivileged edition, section 7.88, all of
1392                 * these must be true to distinguish from WRasr
1393                 */
1394                if (v9 != 0 && f->f3.rd == 15 && f->f3.rs1 == 0 &&
1395                    f->f3.i == 1) {
1396                        prt_name(dhp, "sir", 1);
1397                        prt_imm(dhp, sign_extend(f->f3a.simm13, 13),
1398                            IMM_SIGNED);
1399                        return (0);
1400                }

1402                /* synth: mov */
1403                if ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1395                if ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1404                    == 0)
1405                        break;

1407                if (v9 == 0) {
1408                        if (f->f3.rs1 == 0) {
1409                                name = "mov";
1410                                pr_rs1 = 0;
1411                        }

1413                        if ((f->f3.i == 0 && f->f3.rs2 == 0) ||
1414                            (f->f3.i == 1 && f->f3a.simm13 == 0)) {
1415                                name = "mov";
1416                                pr_rs2 = 0;
1417                        }
1418                }
```



```
1420                if (pr_rs1 == 0)
1421                        pr_rs2 = 1;

1423                break;

1425        case 0x31:
1426                /*
1427                 * NOTE: due to the presence of an overlay entry for another
1428                 * table, this case only happens when doing v8 instructions
1429                 * only
1430                 */
1431                regstr = psr_str;
1432                use_mask = 0;
1433                break;

1435        case 0x32:
1436                if (v9 != 0) {
1437                        regstr = v9_privreg_names[ridx];
1438                        mask = v9_pr_wrmask;
1439                } else {
1440                        regstr = wim_str;
1441                        use_mask = 0;
1442                }
1443                break;

1445        case 0x33:
1446                if (v9 != 0) {
1447                        regstr = v9_hprivreg_names[ridx];
1448                        mask = v9_hpr_wrmask;
1449                } else {
1450                        regstr = tbr_str;
1451                        use_mask = 0;
1452                }
1453                break;
1454        }

1456        if (regstr == NULL)
1457                return (-1);

1459        if (use_mask != 0 && ((1L << ridx) & mask) == 0)
1460                return (-1);

1462        prt_name(dhp, name, 1);

1464        if (rd != 0) {
1465                bprintf(dhp, "%s, %s", regstr, reg_names[f->f3.rd]);
1466        } else {
1467                if (pr_rs1 == 1)
1468                        bprintf(dhp, "%s, ", reg_names[f->f3.rs1]);

1470                if (pr_rs2 != 0) {
1471                        if (f->f3.i == 1)
1472                                prt_imm(dhp, sign_extend(f->f3a.simm13, 13),
1473                                    IMM_SIGNED);
1474                        else
1475                                (void) strlcat(dhx->dhx_buf,
1476                                    reg_names[f->f3.rs2], dhx->dhx_buflen);
1477                        (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
1467                                (void) strlcat(dhp->dh_buf,
1468                                    reg_names[f->f3.rs2], dhp->dh_buflen);
1469                        (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
1478                }

1480                (void) strlcat(dhx->dhx_buf, regstr, dhx->dhx_buflen);
1472                (void) strlcat(dhp->dh_buf, regstr, dhp->dh_buflen);
```

```
1481         }

1483         return (0);
1484 }

1486 /* ARGSUSED3 */
1487 int
1488 fmt_trap(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1489 {
1490         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1491         ifmt_t *f = (ifmt_t *)&instr;

1493         int v9 = ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0);
1494         int p_rs1, p_t;

1496         if (f->ftcc.undef != 0)
1497                 return (-1);

1499         if (icc_names[f->ftcc.cc] == NULL)
1500                 return (-1);

1502         if (f->ftcc.i == 1 && f->ftcc.undef2 != 0)
1503                 return (-1);

1505         if (f->ftcc2.i == 0 && f->ftcc2.undef2 != 0)
1506                 return (-1);

1508         p_rs1 = ((f->ftcc.rs1 != 0) ||
1509             ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0));
1500             ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0));

1511         if (f->ftcc.i == 0) {
1512                 p_t = (f->f3.rs2 != 0 || p_rs1 == 0);

1514                 bprintf(dhp, "%-9s %s%s%s%s%s", inp->in_data.in_def.in_name,
1515                     (v9 != 0) ? icc_names[f->ftcc2.cc] : "",
1516                     (v9 != 0) ? ", " : "",
1517                     (p_rs1 != 0) ? reg_names[f->ftcc2.rs1] : "",
1518                     (p_rs1 != 0) ? " + " : "",
1519                     (p_t != 0) ? reg_names[f->f3.rs2] : "");
1520         } else {
1521                 bprintf(dhp, "%-9s %s%s%s%s0x%x", inp->in_data.in_def.in_name,
1522                     (v9 != 0) ? icc_names[f->ftcc2.cc] : "",
1523                     (v9 != 0) ? ", " : "",
1524                     (p_rs1 != 0) ? reg_names[f->ftcc2.rs1] : "",
1525                     (p_rs1 != 0) ? " + " : "",
1526                     f->ftcc.immtrap);
1527         }
1528         return (0);
1529 }
```

*unchanged_portion_omitted_*

```
1563 /* ARGSUSED3 */
1564 static int
1565 prt_jmpl(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1566 {
1567         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1568         const char *name = inp->in_data.in_def.in_name;
1569         ifmt_t *f = (ifmt_t *)&instr;

1571         if (f->f3.rd == 15 && ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0))
1561         if (f->f3.rd == 15 && ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0))
1572                 name = "call";

1574         if (f->f3.rd == 0) {
1575                 if (f->f3.i == 1 && f->f3a.simm13 == 8) {
```



```
1576                         if (f->f3.rs1 == 15) {
1577                                 prt_name(dhp, "retl", 0);
1578                                 return (0);
1579                         }

1581                         if (f->f3.rs1 == 31) {
1582                                 prt_name(dhp, "ret", 0);
1583                                 return (0);
1584                         }
1585                 }

1587                 name = "jmp";
1588         }

1590         prt_name(dhp, name, 1);
1591         prt_address(dhp, instr, 1);

1593         if (f->f3.rd == 0)
1594                 return (0);

1596         if (f->f3.rd == 15 && ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0))
1586         if (f->f3.rd == 15 && ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0))
1597                 return (0);

1599         bprintf(dhp, ", %s", reg_names[f->f3.rd]);

1601         return (0);
1602 }

1604 int
1605 fmt_alu(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1606 {
1607         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1608         ifmt_t *f = (ifmt_t *)&instr;

1610         const char *name = inp->in_data.in_def.in_name;
1611         int flags = inp->in_data.in_def.in_flags;
1612         int arg = 0;

1614         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1603         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
1615                 prt_field("op", f->f3.op, 2);
1616                 prt_field("op3", f->f3.op3, 6);
1617                 prt_field("rs1", f->f3.rs1, 5);

1619                 switch (idx) {
1620                         /* TODO: more formats */

1622                 default:
1623                         if (f->f3.i == 0)
1624                                 prt_field("rs2", f->f3.rs2, 5);
1625                         else
1626                                 prt_field("simm13", f->f3a.simm13, 13);

1628                         prt_field("rd", f->f3.rd, 5);
1629                 }

1631         }

1633         switch (idx) {
1634         case 0x00:
1635                 /* add */

1637                 if ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) == 0)
1626                 if ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) == 0)
1638                         break;
```

```
1640                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1641                    f->f3a.simm13 == 1) {
1642                        name = "inc";
1643                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1644                        break;
1645                }

1647                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1648                    f->f3a.simm13 != 1) {
1649                        name = "inc";
1650                        flags = FLG_P1(REG_NONE);
1651                        break;
1652                }
1653                break;

1655        case 0x02:
1656                /* or */

1658                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1647                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1659                    == 0)
1660                        break;

1662                if ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) != 0) {
1651                if ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) != 0) {
1663                        if (f->f3.rs1 == f->f3.rd) {
1664                                name = "bset";
1665                                flags = FLG_P1(REG_NONE);
1666                                break;
1667                        }
1668                }

1670                if (((f->f3.i == 0 && f->f3.rs2 == 0) ||
1671                    (f->f3.i == 1 && f->f3a.simm13 == 0)) &&
1672                    (f->f3.rs1 == 0)) {
1673                        name = "clr";
1674                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1675                        break;
1676                }

1678                if (f->f3.rs1 == 0) {
1679                        name = "mov";
1680                        flags = FLG_P1(REG_NONE);
1681                        break;
1682                }
1683                break;

1685        case 0x04:
1686                /* sub */

1688                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1677                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1689                    == 0)
1690                        break;

1692                if (f->f3.rs1 == 0 && f->f3.i == 0 && f->f3.rs2 == f->f3.rd) {
1693                        name = "neg";
1694                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE);
1695                        break;
1696                }

1698                if (f->f3.rs1 == 0 && f->f3.i == 0 && f->f3.rs2 != f->f3.rd) {
1699                        name = "neg";
1700                        flags = FLG_P1(REG_NONE);
1701                        break;
```



```
1702                }

1704                if ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) == 0)
1693                if ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) == 0)
1705                        break;

1707                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1708                    f->f3a.simm13 == 1) {
1709                        name = "dec";
1710                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1711                        break;
1712                }

1714                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1715                    f->f3a.simm13 != 1) {
1716                        name = "dec";
1717                        flags = FLG_P1(REG_NONE);
1718                        break;
1719                }
1720                break;

1722        case 0x07:
1723                /* xnor */

1725                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1714                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1726                    == 0)
1727                        break;

1729                /*
1730                 * xnor -> not when you have:
1731                 *       xnor %rs1, 0x0 or %g0, %rd
1732                 */
1733                if ((f->f3.i == 0 && f->f3.rs2 != 0) ||
1734                    (f->f3.i == 1 && f->f3a.simm13 != 0))
1735                        break;

1737                name = "not";

1739                if (f->f3.rs1 == f->f3.rd)
1740                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM|
1741                            FLG_P3(REG_INT);
1742                else
1743                        flags = FLG_P1(REG_INT)|FLG_P2(REG_NONE)|FLG_NOIMM|
1744                            FLG_P3(REG_INT);

1746                break;

1748        case 0x10:
1749                /* addcc */

1751                if ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) == 0)
1740                if ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) == 0)
1752                        break;

1754                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1755                    f->f3a.simm13 == 1) {
1756                        name = "inccc";
1757                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1758                        break;
1759                }

1761                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1762                    f->f3a.simm13 != 1) {
1763                        name = "inccc";
1764                        flags = FLG_P1(REG_NONE);
```

```
1765                        break;
1766                }
1767                break;

1769        case 0x11:
1770                /* andcc */

1772                if (f->f3.rd != 0)
1773                        break;

1775                if ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1764                if ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1776                    == 0)
1777                        break;

1779                if (((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0) &&
1768                if (((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0) &&
1780                    ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) == 0))
1781                        break;

1783                name = "btst";
1784                flags = FLG_P1(REG_NONE);
1785                f->f3.rd = f->f3.rs1;
1786                break;

1788        case 0x12:
1789                /* orcc */

1791                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1780                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1792                    == 0)
1793                        break;

1795                if (f->f3.rs1 == 0 && f->f3.rd == 0 && f->f3.i == 0) {
1796                        name = "tst";
1797                        flags = FLG_P1(REG_NONE)|FLG_P3(REG_NONE);
1798                        break;
1799                }

1801                if (f->f3.rs2 == 0 && f->f3.rd == 0 && f->f3.i == 0) {
1802                        name = "tst";
1803                        flags = FLG_P2(REG_NONE)|FLG_P3(REG_NONE);
1804                        break;
1805                }

1807                break;

1809        case 0x14:
1810                /* subcc */

1812                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1801                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1813                    == 0)
1814                        break;

1816                if (f->f3.rd == 0) {
1817                        name = "cmp";
1818                        flags = FLG_P3(REG_NONE);
1819                        break;
1820                }

1822                if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0)
1811                if ((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0)
1823                        break;

1825                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
```



```
1826                    f->f3a.simm13 == 1) {
1827                        name = "deccc";
1828                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1829                        break;
1830                }

1832                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1833                    f->f3a.simm13 != 1) {
1834                        name = "deccc";
1835                        flags = FLG_P1(REG_NONE);
1836                        break;
1837                }

1839                break;

1841        case 0x25:
1842        case 0x26:
1843        case 0x27:
1844                return (prt_shift(dhp, instr, inp));

1846        case 0x28:
1847        case 0x29:
1848        case 0x2a:
1849        case 0x2b:
1850        case 0x30:
1851        case 0x31:
1852        case 0x32:
1853        case 0x33:
1854                return (dis_fmt_rdwr(dhp, instr, inp, idx));

1856        case 0x36:
1857        case 0x37:
1858                /* NOTE: overlayed on v9 */
1859                if ((dhp->dh_flags & DIS_SPARC_V8) != 0)
1860                        return (fmt_cpop(dhp, instr, inp));
1861                break;

1863        case 0x38:
1864                /* jmpl */
1865                return (prt_jmpl(dhp, instr, inp, idx));

1867        case 0x39:
1868                /* rett / return */
1869                prt_name(dhp, name, 1);
1870                prt_address(dhp, instr, 1);
1871                return (0);

1873        case 0x3b:
1874                /* flush */
1875                prt_name(dhp, name, 1);
1876                prt_address(dhp, instr, 0);
1877                return (0);

1879        case 0x3c:
1880        case 0x3d:
1881                /* save / restore */
1882                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1871                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1883                    == 0)
1884                        break;

1886                if (f->f3.rs1 != 0 || f->f3.rs2 != 0 || f->f3.rd != 0)
1887                        break;

1889                if (f->f3.i != 0 && ((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0))
1878                if (f->f3.i != 0 && ((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0))
```

```
1890                        break;

1892                prt_name(dhp, name, 0);
1893                return (0);
1894        }

1896        if (FLG_P1_VAL(flags) != REG_NONE || FLG_P2_VAL(flags) != REG_NONE ||
1897            FLG_P3_VAL(flags) != REG_NONE)
1898                arg = 1;

1900        prt_name(dhp, name, (arg != 0));
1901        prt_aluargs(dhp, instr, flags);

1903        return (0);
1904 }
______unchanged_portion_omitted_

1929 /* ARGSUSED3 */
1930 int
1931 fmt_movcc(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1932 {
1933         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1934         ifmt_t *f = (ifmt_t *)&instr;
1935         const char **regs = NULL;

1937         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1925         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
1938                 prt_field("op", f->f3c.op, 2);
1939                 prt_field("op3", f->f3c.op3, 6);
1940                 prt_field("cond", f->f3c.cond, 4);
1941                 prt_field("cc2", f->f3c.cc2, 1);
1942                 prt_field("cc", f->f3c.cc, 2);
1943                 prt_field("i", f->f3c.i, 1);

1945                 if (f->f3c.i == 0)
1946                         prt_field("rs2", f->f3.rs2, 5);
1947                 else
1948                         prt_field("simm11", f->f3c.simm11, 11);

1950                 prt_field("rd", f->f3.rd, 5);
1951         }

1953         if (f->f3c.cc2 == 0) {
1954                 regs = fcc_names;
1955         } else {
1956                 regs = icc_names;
1957                 if (regs[f->f3c.cc] == NULL)
1958                         return (-1);
1959         }

1961         prt_name(dhp, inp->in_data.in_def.in_name, 1);

1963         bprintf(dhp, "%s, ", regs[f->f3c.cc]);

1965         if (f->f3c.i == 1)
1966                 prt_imm(dhp, sign_extend(f->f3c.simm11, 11), IMM_SIGNED);
1967         else
1968                 (void) strlcat(dhx->dhx_buf, reg_names[f->f3.rs2],
1969                     dhx->dhx_buflen);
1956                 (void) strlcat(dhp->dh_buf, reg_names[f->f3.rs2],
1957                     dhp->dh_buflen);

1971         bprintf(dhp, ", %s", reg_names[f->f3.rd]);

1973         return (0);
1974 }
```



```
1976 /* ARGSUSED3 */
1977 int
1978 fmt_movr(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1979 {
1980         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1981         ifmt_t *f = (ifmt_t *)&instr;

1983         prt_name(dhp, inp->in_data.in_def.in_name, 1);

1985         bprintf(dhp, "%s, ", reg_names[f->f3d.rs1]);

1987         if (f->f3d.i == 1)
1988                 prt_imm(dhp, sign_extend(f->f3d.simm10, 10), IMM_SIGNED);
1989         else
1990                 (void) strlcat(dhx->dhx_buf, reg_names[f->f3.rs2],
1991                     dhx->dhx_buflen);
1977                 (void) strlcat(dhp->dh_buf, reg_names[f->f3.rs2],
1978                     dhp->dh_buflen);

1993         bprintf(dhp, ", %s", reg_names[f->f3.rd]);

1995         return (0);
1996 }

1998 /* ARGSUSED3 */
1999 int
2000 fmt_fpop1(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
2001 {
2002         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2003         ifmt_t *f = (ifmt_t *)&instr;
2004         int flags = inp->in_data.in_def.in_flags;

2006         flags |= FLG_NOIMM;

2008         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1994         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
2009                 prt_field("op", f->f3.op, 2);
2010                 prt_field("op3", f->f3.op3, 6);
2011                 prt_field("opf", f->fcmp.opf, 9);
2012                 prt_field("rs1", f->f3.rs1, 5);
2013                 prt_field("rs2", f->f3.rs2, 5);
2014                 prt_field("rd", f->f3.rd, 5);
2015         }

2017         prt_name(dhp, inp->in_data.in_def.in_name, 1);
2018         prt_aluargs(dhp, instr, flags);

2020         return (0);
2021 }

2023 int
2024 fmt_fpop2(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
2025 {
2026         static const char *condstr_icc[16] = {
2027                 "n",  "e",   "le",  "l",   "leu", "lu",   "neg", "vs",
2028                 "a",  "nz",  "g",   "ge",  "gu",  "geu",  "pos", "vc"
2029         };

2031         static const char *condstr_fcc[16] = {
2032                 "n",  "nz",  "lg",  "ul",  "l",   "ug",  "g",   "u",
2033                 "a",  "e",   "ue",  "ge",  "uge", "le",  "ule", "o"
2034         };

2036         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2037         ifmt_t *f = (ifmt_t *)&instr;
```

```
2038         const char *ccstr = "";
2039         char name[15];

2041         int flags = inp->in_data.in_def.in_flags;
2042         int is_cmp = (idx == 0x51 || idx == 0x52 || idx == 0x53 ||
2043             idx == 0x55 || idx == 0x56 || idx == 0x57);
2044         int is_fmov = (idx & 0x3f);
2045         int is_v9 = ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0);
2046         int is_compat = ((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0);
2031         int is_compat = ((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0);

2048         int p_cc = 0;

2050         is_fmov = (is_fmov == 0x1 || is_fmov == 0x2 || is_fmov == 0x3);

2052         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
2037         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
2053                 prt_field("op", f->f3.op, 2);
2054                 prt_field("op3", f->f3.op3, 6);
2055                 prt_field("opf", f->fcmp.opf, 9);

2057                 switch (idx & 0x3f) {
2058                 case 0x51:
2059                 case 0x52:
2060                 case 0x53:
2061                 case 0x55:
2062                 case 0x56:
2063                 case 0x57:
2064                         prt_field("cc", f->fcmp.cc, 2);
2065                         prt_field("rs1", f->f3.rs1, 5);
2066                         prt_field("rs2", f->f3.rs2, 5);
2067                         break;

2069                 case 0x01:
2070                 case 0x02:
2071                 case 0x03:
2072                         prt_field("opf_low", f->fmv.opf, 6);
2073                         prt_field("cond", f->fmv.cond, 4);
2074                         prt_field("opf_cc", f->fmv.cc, 3);
2075                         prt_field("rs2", f->fmv.rs2, 5);
2076                         break;

2078                 default:
2079                         prt_field("rs1", f->f3.rs1, 5);
2080                         prt_field("rs2", f->f3.rs2, 5);
2081                         prt_field("rd", f->f3.rd, 5);
2082                 }
2083         }

2085         name[0] = '\0';
2086         (void) strlcat(name, inp->in_data.in_def.in_name, sizeof (name));

2088         if (is_fmov != 0) {
2089                 (void) strlcat(name,
2090                     (f->fmv.cc < 4) ? condstr_fcc[f->fmv.cond]
2091                     : condstr_icc[f->fmv.cond],
2092                     sizeof (name));
2093         }

2095         prt_name(dhp, name, 1);

2097         if (is_cmp != 0)
2098                 ccstr = fcc_names[f->fcmp.cc];

2100         if (is_fmov != 0)
2101                 ccstr = (f->fmv.cc < 4) ? fcc_names[f->fmv.cc & 0x3]
```



```
2102                     : icc_names[f->fmv.cc & 0x3];

2104         if (ccstr == NULL)
2105                 return (-1);

2107         p_cc = (is_compat == 0 || is_v9 != 0 ||
2108             (is_cmp != 0 && f->fcmp.cc != 0) ||
2109             (is_fmov != 0 && f->fmv.cc != 0));

2111         if (p_cc != 0)
2112                 bprintf(dhp, "%s, ", ccstr);

2114         prt_aluargs(dhp, instr, flags);

2116         return (0);
2117 }

2119 int
2120 fmt_vis(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
2121 {
2122         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2123         ifmt_t *f = (ifmt_t *)&instr;
2124         int flags = inp->in_data.in_def.in_flags;

2126         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
2110         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
2127                 prt_field("op", f->f3.op, 2);
2128                 prt_field("op3", f->f3.op3, 6);
2129                 prt_field("opf", f->fcmp.opf, 9);

2131                 if (idx == 0x081) {
2132                         prt_field("mode", instr & 02L, 2);
2133                 } else {
2134                         prt_field("rs1", f->f3.rs1, 5);
2135                         prt_field("rs2", f->f3.rs2, 5);
2136                         prt_field("rd", f->f3.rd, 5);
2137                 }
2138         }

2140         prt_name(dhp, inp->in_data.in_def.in_name, 1);

2142         if (idx == 0x081) {
2143                 /* siam */
2144                 bprintf(dhp, "%d", instr & 0x7L);
2145                 return (0);
2146         }

2148         prt_aluargs(dhp, instr, flags);

2150         return (0);
2151 }
```

*unchanged_portion_omitted_*

```
2257 /*
2258  * return the symbolic name of a register
2259  * regset is one of the REG_* values indicating which type of register it is
2260  * such as integer, floating point, etc.
2261  * idx is the numeric value of the register
2262  *
2263  * If regset is REG_NONE, an empty, but non-NULL string is returned
2264  * NULL may be returned if the index indicates an invalid register value
2265  * such as with the %icc/%xcc sets
2266  */
2267 static const char *
2268 get_regname(dis_handle_t *dhp, int regset, uint32_t idx)
2269 {
```

```
2270         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2271         const char *regname = NULL;

2273         switch (regset) {
2274         case REG_INT:
2275                 regname = reg_names[idx];
2276                 break;

2278         case REG_FP:
2279                 regname = freg_names[idx];
2280                 break;

2282         case REG_FPD:
2283                 if (((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0) ||
2266                 if (((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0) ||
2284                     ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0))
2285                         regname = fdreg_names[idx];
2286                 else
2287                         regname = compat_fdreg_names[idx];

2289                 break;

2291         case REG_FPQ:
2292                 if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
2275                 if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
2293                         regname = fqreg_names[idx];
2294                 else
2295                         regname = freg_names[idx];

2297                 break;

2299         case REG_CP:
2300                 regname = cpreg_names[idx];
2301                 break;

2303         case REG_ICC:
2304                 regname = icc_names[idx];
2305                 break;

2307         case REG_FCC:
2308                 regname = fcc_names[idx];
2309                 break;

2311         case REG_FSR:
2312                 regname = "%fsr";
2313                 break;

2315         case REG_CSR:
2316                 regname = "%csr";
2317                 break;

2319         case REG_CQ:
2320                 regname = "%cq";
2321                 break;

2323         case REG_NONE:
2324                 regname = "";
2325                 break;
2326         }

2328         return (regname);
2329 }
```

*unchanged_portion_omitted_*

```
2350 /*
2351  * put an address expression into the output buffer
```



```
2352  *
2353  * instr is the instruction to use
2354  * if nobrackets != 0, [] are not added around the instruction
2355  *
2356  * Currently this option is set when printing out the address portion
2357  * of a jmpl instruction, but otherwise 0 for load/stores
2358  *
2359  * If no debug flags are set, the full expression is output, even when
2360  * %g0 or 0x0 appears in the address
2361  *
2362  * If DIS_DEBUG_SYN_ALL or DIS_DEBUG_COMPAT are set, when %g0 or 0x0
2363  * appear in the address, they are not output.  If the wierd (and probably
2364  * shouldn't happen) address of [%g0 + %g0] or [%g0 + 0x0] is encountered,
2365  * [%g0] is output
2366  */
2367 static void
2368 prt_address(dis_handle_t *dhp, uint32_t instr, int nobrackets)
2369 {
2370         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2371         ifmt_t *f = (ifmt_t *)&instr;
2372         int32_t simm13;
2373         int octal = ((dhp->dh_flags & DIS_OCTAL) != 0);
2374         int p1 = ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0);
2375         int p2 = ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0);
2356         int p1 = ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0);
2357         int p2 = ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0);

2377         if (f->f3a.i == 0) {
2378                 p1 |= ((f->f3a.rs1 != 0) || f->f3.rs2 == 0);
2379                 p2 |= (f->f3.rs2 != 0);

2381                 bprintf(dhp, "%s%s%s%s%s",
2382                     (nobrackets == 0) ? "[" : "",
2383                     (p1 != 0) ? reg_names[f->f3a.rs1] : "",
2384                     (p1 != 0 && p2 != 0) ? " + " : "",
2385                     (p2 != 0) ? reg_names[f->f3.rs2] : "",
2386                     (nobrackets == 0) ? "]" : "");
2387         } else {
2388                 const char *sign;

2390                 simm13 = sign_extend(f->f3a.simm13, 13);
2391                 sign = (simm13 < 0) ? "-" : "+";

2393                 p1 |= (f->f3a.rs1 != 0);
2394                 p2 |= (p1 == 0 || simm13 != 0);

2396                 if (p1 == 0 && simm13 == 0)
2397                         p2 = 1;

2399                 if (p1 == 0 && simm13 >= 0)
2400                         sign = "";

2402                 if (p2 != 0)
2403                         bprintf(dhp,
2404                             (octal != 0) ? "%s%s%s%s%s0%lo%s" :
2405                             "%s%s%s%s%s0x%lx%s",
2406                             (nobrackets == 0) ? "[" : "",
2407                             (p1 != 0) ? reg_names[f->f3a.rs1] : "",
2408                             (p1 != 0) ? " " : "",
2409                             sign,
2410                             (p1 != 0) ? " " : "",
2411                             (simm13 < 0) ? -(simm13) : simm13,
2412                             (nobrackets == 0) ? "]" : "");
2413                 else
2414                         bprintf(dhp, "%s%s%s",
2415                             (nobrackets == 0) ? "[" : "",
```

```
2416                            reg_names[f->f3a.rs1],
2417                            (nobrackets == 0) ? "]" : "");
2418        }
2419 }

2421 /*
2422  * print out the arguments to an alu operation (add, sub, etc.)
2423  * conatined in 'instr'
2424  *
2425  * alu instructions have the following format:
2426  *       %rs1, %rs2, %rd    (i == 0)
2427  *       %rs1, 0xnnn, %rd   (i == 1)
2428  *         ^    ^       ^
2429  *         |    |       |
2430  *        p1    p2      p3
2431  *
2432  * flags indicates the register set to use for each position (p1, p2, p3)
2433  * as well as if immediate values (i == 1) are allowed
2434  *
2435  * if flags indicates a specific position has REG_NONE set as it's register
2436  * set, it is omitted from the output.  This is primarly used for certain
2437  * floating point operations
2438  */
2439 static void
2440 prt_aluargs(dis_handle_t *dhp, uint32_t instr, uint32_t flags)
2441 {
2442         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2443         ifmt_t *f = (ifmt_t *)&instr;
2444         const char *r1, *r2, *r3;
2445         int p1, p2, p3;
2446         unsigned int opf = 0;

2448         r1 = get_regname(dhp, FLG_P1_VAL(flags), f->f3.rs1);
2449         r2 = get_regname(dhp, FLG_P2_VAL(flags), f->f3.rs2);
2450         r3 = get_regname(dhp, FLG_P3_VAL(flags), f->f3.rd);

2452         p1 = (FLG_P1_VAL(flags) != REG_NONE);
2453         p2 = (((flags & FLG_NOIMM) == 0) || (FLG_P2_VAL(flags) != REG_NONE));
2454         p3 = (FLG_RD_VAL(flags) != REG_NONE);

2456         if (r1 == NULL || r1[0] == '\0')
2457                 p1 = 0;

2459         if (f->f3a.i == 0 && (r2 == NULL || r2[0] == '\0'))
2460                 p2 = 0;

2462         if (r3 == NULL || r3[0] == '\0')
2463                 p3 = 0;

2465         if ((f->fcmp.op == 2) && (f->fcmp.op3 == 0x36) && (f->fcmp.cc != 0))
2466                 opf = f->fcmp.opf;

2468         if ((opf == 0x151) || (opf == 0x152)) {
2469                 (void) strlcat(dhx->dhx_buf, r3, dhx->dhx_buflen);
2470                 (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
2450                 (void) strlcat(dhp->dh_buf, r3, dhp->dh_buflen);
2451                 (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
2471                 p3 = 0;
2472         }

2474         if (p1 != 0) {
2475                 (void) strlcat(dhx->dhx_buf, r1, dhx->dhx_buflen);
2456                 (void) strlcat(dhp->dh_buf, r1, dhp->dh_buflen);
2476                 if (p2 != 0 || p3 != 0)
2477                         (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
2458                         (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
```



```
2478         }

2480         if (p2 != 0) {
2481                 if (f->f3.i == 0 || ((flags & FLG_NOIMM) != 0))
2482                         (void) strlcat(dhx->dhx_buf, r2, dhx->dhx_buflen);
2463                         (void) strlcat(dhp->dh_buf, r2, dhp->dh_buflen);
2483                 else
2484                         prt_imm(dhp, sign_extend(f->f3a.simm13, 13),
2485                             IMM_SIGNED);

2487                 if (p3 != 0)
2488                         (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
2469                         (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
2489         }

2491         if (p3 != 0)
2492                 (void) strlcat(dhx->dhx_buf, r3, dhx->dhx_buflen);
2473                 (void) strlcat(dhp->dh_buf, r3, dhp->dh_buflen);
2493 }
______unchanged_portion_omitted_

2753 /*
2754  * just a handy function that takes care of managing the buffer length
2755  * w/ printf
2756  */

2758 /*
2759  * PRINTF LIKE 1
2760  */
2761 static void
2762 bprintf(dis_handle_t *dhp, const char *fmt, ...)
2763 {
2764         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2765         size_t curlen;
2766         va_list ap;

2768         curlen = strlen(dhx->dhx_buf);
2748         curlen = strlen(dhp->dh_buf);

2770         va_start(ap, fmt);
2771         (void) vsnprintf(dhx->dhx_buf + curlen, dhx->dhx_buflen - curlen, fmt,
2751         (void) vsnprintf(dhp->dh_buf + curlen, dhp->dh_buflen - curlen, fmt,
2772             ap);
2773         va_end(ap);
2774 }
______unchanged_portion_omitted_
```

```
**********************************************************
    3871 Sun Dec 16 13:00:32 2012
new/usr/src/lib/libdisasm/common/dis_sparc_fmt.h
style fixes
take to dis and libdisasm with an axe; does not yet compile
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  */

  27 /*
  28  * Copyright 2007 Jason King.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma ident   "%Z%%M% %I%     %E% SMI"

  32 #ifndef _DIS_SPARC_FMT_H
  33 #define _DIS_SPARC_FMT_H

  35 #ifdef __cplusplus
  36 extern "C" {
  37 #endif

  39 #include <sys/types.h>
  40 #include "libdisasm.h"
  41 #include "dis_sparc.h"

  43 /* which set of registers are used with an instruction */
  44 #define REG_INT         0x00   /* regular integer registers */
  45 #define REG_FP          0x01   /* single-precision fp registers */
  46 #define REG_FPD         0x02   /* double-precision fp registers */
  47 #define REG_FPQ         0x03   /* quad-precision fp registers */
  48 #define REG_CP          0x04   /* coprocessor registers (v8) */
  49 #define REG_ICC         0x05   /* %icc / % xcc */
  50 #define REG_FCC         0x06   /* %fccn */
  51 #define REG_FSR         0x07   /* %fsr */
  52 #define REG_CSR         0x08   /* %csr */
  53 #define REG_CQ          0x09   /* %cq */
  54 #define REG_NONE        0x0a   /* no registers */

  56 /* the size fo the displacement for branches */
  57 #define DISP22  0x00
  58 #define DISP19  0x01
```



```
  59 #define DISP16  0x02
  60 #define CONST22 0x03

  62 /* get/set the register set name for the rd field of an instruction */
  63 #define FLG_RD(x)       (x)
  64 #define FLG_RD_VAL(x)   (x & 0xfL)

  66 #define FLG_STORE       (0x1L << 24) /* the instruction is not a load */
  67 #define FLG_ASI         (0x2L << 24) /* the load/store includes an asi value */


  70 /* flags for ALU instructions */

  72 /* set/get register set name for 1st argument position */
  73 #define FLG_P1(x)       (x << 8)
  74 #define FLG_P1_VAL(x)   ((x >> 8) & 0xfL)

  76 /* get/set reg set for 2nd argument position */
  77 #define FLG_P2(x)       (x << 4)
  78 #define FLG_P2_VAL(x)   ((x >> 4) & 0xfL)

  80 /* get/set for 3rd argument position */
  81 #define FLG_P3(x)       (x)
  82 #define FLG_P3_VAL(x)   (x & 0xfL)

  84 /* set if the arguments do not contain immediate values */
  85 #define FLG_NOIMM       (0x01L << 24)



  89 /* flags for branch instructions */

  91 /* has branch prediction */
  92 #define FLG_PRED        (0x01L << 24)

  94 /* get/set condition code register set -- usually REG_NONE */
  95 #define FLG_RS1(x)      (x)
  96 #define FLG_RS1_VAL(x)  (x & 0xfL)

  98 /* get/set displacement size */
  99 #define FLG_DISP(x)     (x << 4L)
 100 #define FLG_DISP_VAL(x) ((x >> 4L) & 0x0fL)


 103 int fmt_call(dis_handle_t *, uint32_t, const inst_t *, int);
 104 int fmt_ls(dis_handle_t *, uint32_t, const inst_t *, int);
 105 int fmt_alu(dis_handle_t *, uint32_t, const inst_t *, int);
 106 int fmt_branch(dis_handle_t *, uint32_t, const inst_t *, int);
 107 int fmt_sethi(dis_handle_t *, uint32_t, const inst_t *, int);
 108 int fmt_fpop1(dis_handle_t *, uint32_t, const inst_t *, int);
 109 int fmt_fpop2(dis_handle_t *, uint32_t, const inst_t *, int);
 110 int fmt_vis(dis_handle_t *, uint32_t, const inst_t *, int);
 111 int fmt_trap(dis_handle_t *, uint32_t, const inst_t *, int);
 112 int fmt_regwin(dis_handle_t *, uint32_t, const inst_t *, int);
 113 int fmt_trap_ret(dis_handle_t *, uint32_t, const inst_t *, int);
 114 int fmt_movcc(dis_handle_t *, uint32_t, const inst_t *, int);
 115 int fmt_movr(dis_handle_t *, uint32_t, const inst_t *, int);
 116 int fmt_fused(dis_handle_t *, uint32_t, const inst_t *, int);

 118 #ifdef __cplusplus
 119 }
______unchanged_portion_omitted_
```

```
**********************************************************
    4293 Sun Dec 16 13:00:32 2012
new/usr/src/lib/libdisasm/common/libdisasm.c
style fixes
only include native support in standalone library
take to dis and libdisasm with an axe; does not yet compile
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

#include <libdisasm.h>
#include <stdlib.h>
#ifdef DIS_STANDALONE
#include <mdb/mdb_modapi.h>
#endif

#include "libdisasm_impl.h"

static int _dis_errno;

/*
 * If we're building the standalone library, then we only want to
 * include support for disassembly of the native architecture.
 * The regular shared library should include support for all
 * architectures.
 */
#if !defined(DIS_STANDALONE) || defined(__i386) || defined(__amd64)
extern dis_arch_t dis_arch_i386;
#endif
#if !defined(DIS_STANDALONE) || defined(__sparc)
extern dis_arch_t dis_arch_sparc;
#endif

static dis_arch_t *dis_archs[] = {
#if !defined(DIS_STANDALONE) || defined(__i386) || defined(__amd64)
        &dis_arch_i386,
#endif
#if !defined(DIS_STANDALONE) || defined(__sparc)
        &dis_arch_sparc,
#endif
```



```
        NULL
};

/*
 * For the standalone library, we need to link against mdb's malloc/free.
 * Otherwise, use the standard malloc/free.
 */
#ifdef DIS_STANDALONE
void *
dis_zalloc(size_t bytes)
{
        return (mdb_zalloc(bytes, UM_SLEEP));
}
_____unchanged_portion_omitted_

const char *
dis_strerror(int error)
{
        switch (error) {
        case E_DIS_NOMEM:
                return ("out of memory");
        case E_DIS_INVALFLAG:
                return ("invalid flags for this architecture");
        case E_DIS_UNSUPARCH:
                return ("unsupported machine architecture");
        default:
                return ("unknown error");
        }
}

void
dis_set_data(dis_handle_t *dhp, void *data)
{
        dhp->dh_data = data;
}

void
dis_flags_set(dis_handle_t *dhp, int f)
{
        dhp->dh_flags |= f;
}

void
dis_flags_clear(dis_handle_t *dhp, int f)
{
        dhp->dh_flags &= ~f;
}

void
dis_handle_destroy(dis_handle_t *dhp)
{
        dhp->dh_arch->da_handle_detach(dhp);
        dis_free(dhp, sizeof (dis_handle_t));
}

dis_handle_t *
dis_handle_create(int flags, void *data, dis_lookup_f lookup_func,
    dis_read_f read_func)
{
        dis_handle_t *dhp;
        dis_arch_t *arch = NULL;
        int i;

        /* Select an architecture based on flags */
        for (i = 0; dis_archs[i] != NULL; i++) {
                if (dis_archs[i]->da_supports_flags(flags)) {
```

```
156                         arch = dis_archs[i];
157                         break;
158                 }
159         }
160         if (arch == NULL) {
161                 (void) dis_seterrno(E_DIS_UNSUPARCH);
162                 return (NULL);
163         }

165         if ((dhp = dis_zalloc(sizeof (dis_handle_t))) == NULL) {
166                 (void) dis_seterrno(E_DIS_NOMEM);
167                 return (NULL);
168         }
169         dhp->dh_arch = arch;
170         dhp->dh_lookup = lookup_func;
171         dhp->dh_read = read_func;
172         dhp->dh_flags = flags;
173         dhp->dh_data = data;

175         /*
176          * Allow the architecture-specific code to allocate
177          * its private data.
178          */
179         if (arch->da_handle_attach(dhp) != 0) {
180                 dis_free(dhp, sizeof (dis_handle_t));
181                 /* dis errno already set */
182                 return (NULL);
183         }

185         return (dhp);
186 }

188 int
189 dis_disassemble(dis_handle_t *dhp, uint64_t addr, char *buf, size_t buflen)
190 {
191         return (dhp->dh_arch->da_disassemble(dhp, addr, buf, buflen));
192 }

194 uint64_t
195 dis_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
196 {
197         return (dhp->dh_arch->da_previnstr(dhp, pc, n));
198 }

200 int
201 dis_min_instrlen(dis_handle_t *dhp)
202 {
203         return (dhp->dh_arch->da_min_instrlen(dhp));
204 }

206 int
207 dis_max_instrlen(dis_handle_t *dhp)
208 {
209         return (dhp->dh_arch->da_max_instrlen(dhp));
210 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    2633 Sun Dec 16 13:00:32 2012
new/usr/src/lib/libdisasm/common/libdisasm.h
fixup dis_min_instrlen
take to dis and libdisasm with an axe; does not yet compile
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  * Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
  26  */

  28 #ifndef _LIBDISASM_H
  29 #define _LIBDISASM_H

  30 #pragma ident   "%Z%%M% %I%     %E% SMI"

  31 #include <sys/types.h>

  33 #ifdef  __cplusplus
  34 extern "C" {
  35 #endif

  37 typedef struct dis_handle dis_handle_t;

  39 #define DIS_DEFAULT             0x0

  41 /* SPARC disassembler flags */
  42 #define DIS_SPARC_V8            0x001
  43 #define DIS_SPARC_V9            0x002
  44 #define DIS_SPARC_V9_SGI        0x004
  45 #define DIS_SPARC_V9_OPL        0x008
  43 #define DIS_SPARC_V8            0x01
  44 #define DIS_SPARC_V9            0x02
  45 #define DIS_SPARC_V9_SGI        0x04
  46 #define DIS_SPARC_V9_OPL        0x08

  47 /* x86 diassembler flags */
  48 #define DIS_X86_SIZE16          0x100
  49 #define DIS_X86_SIZE32          0x010
  50 #define DIS_X86_SIZE64          0x020
  48 /* x86 diassembler flags (mutually exclusive) */
  49 #define DIS_X86_SIZE16          0x08
  50 #define DIS_X86_SIZE32          0x10
  51 #define DIS_X86_SIZE64          0x20
```



```
  52 /* generic disassembler flags */
  53 #define DIS_OCTAL               0x040
  54 #define DIS_NOIMMSYM            0x080
  54 #define DIS_OCTAL               0x40
  55 #define DIS_NOIMMSYM            0x80

  56 #define DIS_ARCH_MASK           (DIS_SPARC_V8 | \
  57                 DIS_SPARC_V9 | DIS_SPARC_V9_SGI | DIS_SPARC_V9_OPL | \
  58                 DIS_X86_SIZE16 | DIS_X86_SIZE32 | DIS_X86_SIZE64)

  60 typedef int (*dis_lookup_f)(void *, uint64_t, char *, size_t, uint64_t *,
  61     size_t *);
  62 typedef int (*dis_read_f)(void *, uint64_t, void *, size_t);

  64 extern dis_handle_t *dis_handle_create(int, void *, dis_lookup_f, dis_read_f);
  65 extern void dis_handle_destroy(dis_handle_t *);

  67 extern int dis_disassemble(dis_handle_t *, uint64_t, char *, size_t);
  68 extern uint64_t dis_previnstr(dis_handle_t *, uint64_t, int n);
  69 extern void dis_set_data(dis_handle_t *, void *);
  70 extern void dis_flags_set(dis_handle_t *, int f);
  71 extern void dis_flags_clear(dis_handle_t *, int f);
  72 extern int dis_max_instrlen(dis_handle_t *);
  73 extern int dis_min_instrlen(dis_handle_t *);

  75 /* libdisasm errors */
  76 #define E_DIS_NOMEM             1       /* Out of memory */
  77 #define E_DIS_INVALFLAG         2       /* Invalid flag for this architecture */
  78 #define E_DIS_UNSUPARCH         3       /* Unsupported architecture */

  80 extern int dis_errno(void);
  81 extern const char *dis_strerror(int);

  83 #ifdef  __cplusplus
  84 }
______unchanged_portion_omitted_
```

```
**********************************************************
    1773 Sun Dec 16 13:00:32 2012
new/usr/src/lib/libdisasm/common/libdisasm_impl.h
fixup dis_min_instrlen
take to dis and libdisasm with an axe; does not yet compile
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
 */

#ifndef _LIBDISASM_IMPL_H
#define _LIBDISASM_IMPL_H

#pragma ident   "%Z%%M% %I%     %E% SMI"

#ifdef  __cplusplus
extern "C" {
#endif

typedef struct dis_arch {
        int (*da_supports_flags)(int);
        int (*da_handle_attach)(dis_handle_t *);
        void (*da_handle_detach)(dis_handle_t *);
        int (*da_disassemble)(dis_handle_t *, uint64_t, char *, size_t);
        uint64_t (*da_previnstr)(dis_handle_t *, uint64_t, int n);
        int (*da_min_instrlen)(dis_handle_t *);
        int (*da_max_instrlen)(dis_handle_t *);
} dis_arch_t;

struct dis_handle {
        void            *dh_data;
        int             dh_flags;
        dis_lookup_f    dh_lookup;
        dis_read_f      dh_read;
        uint64_t        dh_addr;

        dis_arch_t      *dh_arch;
        void            *dh_arch_private;
};

extern int dis_seterrno(int);

extern void *dis_zalloc(size_t);
```



```
extern void dis_free(void *, size_t);

#ifdef  __cplusplus
}
#endif
```

*unchanged_portion_omitted_*

****
1504 Sun Dec 16 13:00:32 2012
new/usr/src/lib/libdisasm/common/mapfile-vers
move fixed-size determination mostly into libdisasm
****

```
  1 #
  2 # CDDL HEADER START
  3 #
  4 # The contents of this file are subject to the terms of the
  5 # Common Development and Distribution License (the "License").
  6 # You may not use this file except in compliance with the License.
  7 #
  8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  9 # or http://www.opensolaris.org/os/licensing.
 10 # See the License for the specific language governing permissions
 11 # and limitations under the License.
 12 #
 13 # When distributing Covered Code, include this CDDL HEADER in each
 14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 15 # If applicable, add the following below this CDDL HEADER, with the
 16 # fields enclosed by brackets "[]" replaced with your own identifying
 17 # information: Portions Copyright [yyyy] [name of copyright owner]
 18 #
 19 # CDDL HEADER END
 20 #
 21 #
 22 # Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
 23 #

 25 #
 26 # MAPFILE HEADER START
 27 #
 28 # WARNING:  STOP NOW.  DO NOT MODIFY THIS FILE.
 29 # Object versioning must comply with the rules detailed in
 30 #
 31 #       usr/src/lib/README.mapfiles
 32 #
 33 # You should not be making modifications here until you've read the most current
 34 # copy of that file. If you need help, contact a gatekeeper for guidance.
 35 #
 36 # MAPFILE HEADER END
 37 #

 39 $mapfile_version 2

 41 SYMBOL_VERSION SUNWprivate_1.1 {
 42     global:
 43         dis_disassemble;
 44         dis_errno;
 45         dis_handle_create;
 46         dis_handle_destroy;
 47         dis_max_instrlen;
 48         dis_min_instrlen;
 49         dis_previnstr;
 50         dis_set_data;
 51         dis_flags_set;
 52         dis_flags_clear;
 53         dis_strerror;
 54     local:
 55         *;
 56 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    1115 Sun Dec 16 13:00:33 2012
new/usr/src/lib/libdisasm/i386/Makefile
style fixes
sparc instr.c is C99
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #
  21 #
  22 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  23 # Use is subject to license terms.
  24 #
  25 # ident "%Z%%M% %I%     %E% SMI"

  26 ISASRCDIR=.

  28 include ../Makefile.com

  30 TYPES=library standalone

  32 INSTALL_DEPS_library =          $(ROOTLINKS) $(ROOTLINT) $(ROOTLIBS)
  33 INSTALL_DEPS_standalone =       $(ROOTLIBS)

  35 include ../Makefile.targ

  37 C99MODE = $(C99_ENABLE)
```