```
**********************************************************
   18333 Fri Aug  1 16:03:18 2014
new/usr/src/cmd/dis/dis_main.c
3317 dis(1) should support cross-target disassembly
**********************************************************
```

```
/*
 * 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 2011 Jason King.  All rights reserved.
 * Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
#endif /* ! codereview */
 */

#include <ctype.h>
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/sysmacros.h>
#include <sys/elf_SPARC.h>

#include <libdisasm.h>

#include "dis_target.h"
#include "dis_util.h"
#include "dis_list.h"

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

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

/*
 * Section options for -d, -D, and -s
 */
#define DIS_DATA_RELATIVE       1
#define DIS_DATA_ABSOLUTE       2
#define DIS_TEXT                3

/*
```



```
 * libdisasm callback data.  Keeps track of current data (function or section)
 * and offset within that data.
 */
typedef struct dis_buffer {
        dis_tgt_t       *db_tgt;        /* current dis target */
        void            *db_data;       /* function or section data */
        uint64_t        db_addr;        /* address of function start */
        size_t          db_size;        /* size of data */
        uint64_t        db_nextaddr;    /* next address to be read */
} dis_buffer_t;

#define MINSYMWIDTH     22      /* Minimum width of symbol portion of line */

/*
 * Given a symbol+offset as returned by dis_tgt_lookup(), print an appropriately
 * formatted symbol, based on the offset and current setttings.
 */
void
getsymname(uint64_t addr, const char *symbol, off_t offset, char *buf,
    size_t buflen)
{
        if (symbol == NULL || g_numeric) {
                if (g_flags & DIS_OCTAL)
                        (void) snprintf(buf, buflen, "0%llo", addr);
                else
                        (void) snprintf(buf, buflen, "0x%llx", addr);
        } else {
                if (g_demangle)
                        symbol = dis_demangle(symbol);

                if (offset == 0)
                        (void) snprintf(buf, buflen, "%s", symbol);
                else if (g_flags & DIS_OCTAL)
                        (void) snprintf(buf, buflen, "%s+0%o", symbol, offset);
                else
                        (void) snprintf(buf, buflen, "%s+0x%x", symbol, offset);
        }
}

/*
 * Determine if we are on an architecture with fixed-size instructions,
 * and if so, what size they are.
 */
static int
insn_size(dis_handle_t *dhp)
{
        int min = dis_min_instrlen(dhp);
        int max = dis_max_instrlen(dhp);

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

        return (0);
}

/*
#endif /* ! codereview */
 * The main disassembly routine.  Given a fixed-sized buffer and starting
 * address, disassemble the data using the supplied target and libdisasm handle.
 */
void
dis_data(dis_tgt_t *tgt, dis_handle_t *dhp, uint64_t addr, void *data,
    size_t datalen)
{
        dis_buffer_t db = { 0 };
        char buf[BUFSIZE];
```

```
        char symbuf[BUFSIZE];
        const char *symbol;
        const char *last_symbol;
        off_t symoffset;
        int i;
        int bytesperline;
        size_t symsize;
        int isfunc;
        size_t symwidth = 0;
        int ret;
        int insz = insn_size(dhp);
#endif /* ! codereview */

        db.db_tgt = tgt;
        db.db_data = data;
        db.db_addr = addr;
        db.db_size = datalen;

        dis_set_data(dhp, &db);

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

        symbol = NULL;

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

                ret = dis_disassemble(dhp, addr, buf, BUFSIZE);
                if (ret != 0 && insz > 0) {
                if (dis_disassemble(dhp, addr, buf, BUFSIZE) != 0) {
#if defined(__sparc)
                        /*
                         * Since we know instructions are fixed size, we
                         * Since sparc instructions are fixed size, we
                         * always know the address of the next instruction
                         */
                        (void) snprintf(buf, sizeof (buf),
                            "*** invalid opcode ***");
                        db.db_nextaddr = addr + insz;
                        db.db_nextaddr = addr + 4;

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

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

                        /*
                         * On architectures with variable sized instructions
                         * we have no way to figure out where the next
                         * instruction starts if we encounter an invalid
                         * instruction.  Instead we print the rest of the
                         * instruction stream as hex until we reach the
                         * next valid symbol in the section.
                         */
                        if ((next = dis_tgt_next_symbol(tgt, addr)) == 0) {
                                db.db_nextaddr = db.db_addr + db.db_size;
                        } else {
                                if (next > db.db_size)
                                        db.db_nextaddr = db.db_addr +
                                            db.db_size;
                                else
                                        db.db_nextaddr = addr + next;
                        }
#endif
```



```
                }

                /*
                 * Print out the line as:
                 *
                 *      address:        bytes   text
                 *
                 * If there are more than 6 bytes in any given instruction,
                 * spread the bytes across two lines.  We try to get symbolic
                 * information for the address, but if that fails we print out
                 * the numeric address instead.
                 *
                 * We try to keep the address portion of the text aligned at
                 * MINSYMWIDTH characters.  If we are disassembling a function
                 * with a long name, this can be annoying.  So we pick a width
                 * based on the maximum width that the current symbol can be.
                 * This at least produces text aligned within each function.
                 */
                last_symbol = symbol;
                symbol = dis_tgt_lookup(tgt, addr, &symoffset, 1, &symsize,
                    &isfunc);
                if (symbol == NULL) {
                        symbol = dis_find_section(tgt, addr, &symoffset);
                        symsize = symoffset;
                }

                if (symbol != last_symbol)
                        getsymname(addr, symbol, symsize, symbuf,
                            sizeof (symbuf));

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

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

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

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

                /* trailing spaces for missing bytes */
                for (; i < bytesperline; i++) {
                        if (g_flags & DIS_OCTAL)
                                (void) printf("    ");
                        else
                                (void) printf("   ");
                }

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

                /* excess bytes that spill over onto subsequent lines */
                for (; i < db.db_nextaddr - addr; i++) {
```

```
254                         int byte = *((uchar_t *)data + (addr - db.db_addr) + i);
255                         if (i % bytesperline == 0)
256                                 (void) printf("\n    %*s  ", symwidth, "");
257                         if (g_flags & DIS_OCTAL)
258                                 (void) printf("%03o ", byte);
259                         else
260                                 (void) printf("%02x ", byte);
261                 }

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

265                 addr = db.db_nextaddr;
266         }
267 }
______unchanged_portion_omitted_

467 /*
468  * Disassemble a complete file.  First, we determine the type of the file based
469  * on the ELF machine type, and instantiate a version of the disassembler
470  * appropriate for the file.  We then resolve any named sections or functions
471  * against the file, and iterate over the results (or all sections if no flags
472  * were specified).
473  */
474 void
475 dis_file(const char *filename)
476 {
477         dis_tgt_t *tgt, *current;
478         dis_scnlist_t *sections;
479         dis_funclist_t *functions;
480         dis_handle_t *dhp;
481         GElf_Ehdr ehdr;

483         /*
484          * First, initialize the target
485          */
486         if ((tgt = dis_tgt_create(filename)) == NULL)
487                 return;

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

492         /*
493          * A given file may contain multiple targets (if it is an archive, for
494          * example).  We iterate over all possible targets if this is the case.
495          */
496         for (current = tgt; current != NULL; current = dis_tgt_next(current)) {
497                 dis_tgt_ehdr(current, &ehdr);

499                 /*
500                  * Eventually, this should probably live within libdisasm, and
501                  * we should be able to disassemble targets from different
502                  * architectures.  For now, we only support objects as the
503                  * native machine type.
504                  */
505                 switch (ehdr.e_machine) {
379 #ifdef __sparc
506                 case EM_SPARC:
507                         if (ehdr.e_ident[EI_CLASS] != ELFCLASS32 ||
508                             ehdr.e_ident[EI_DATA] != ELFDATA2MSB) {
509                                 warn("invalid E_IDENT field for SPARC object");
510                                 return;
511                         }
512                         g_flags |= DIS_SPARC_V8;
513                         break;

515                 case EM_SPARC32PLUS:
```



```
516                 {
517                         uint64_t flags = ehdr.e_flags & EF_SPARC_32PLUS_MASK;

519                         if (ehdr.e_ident[EI_CLASS] != ELFCLASS32 ||
520                             ehdr.e_ident[EI_DATA] != ELFDATA2MSB) {
521                                 warn("invalid E_IDENT field for SPARC object");
522                                 return;
523                         }

525                         if (flags != 0 &&
526                             (flags & (EF_SPARC_32PLUS | EF_SPARC_SUN_US1 |
527                             EF_SPARC_SUN_US3)) != EF_SPARC_32PLUS)
528                                 g_flags |= DIS_SPARC_V9 | DIS_SPARC_V9_SGI;
529                         else
530                                 g_flags |= DIS_SPARC_V9;
531                         break;
532                 }

534                 case EM_SPARCV9:
535                         if (ehdr.e_ident[EI_CLASS] != ELFCLASS64 ||
536                             ehdr.e_ident[EI_DATA] != ELFDATA2MSB) {
537                                 warn("invalid E_IDENT field for SPARC object");
538                                 return;
539                         }

541                         g_flags |= DIS_SPARC_V9 | DIS_SPARC_V9_SGI;
542                         break;
417 #endif /* __sparc */

419 #if defined(__i386) || defined(__amd64)
544                 case EM_386:
545                         g_flags |= DIS_X86_SIZE32;
546                         break;

548                 case EM_AMD64:
549                         g_flags |= DIS_X86_SIZE64;
550                         break;
427 #endif /* __i386 || __amd64 */

552                 default:
553                         die("%s: unsupported ELF machine 0x%x", filename,
554                             ehdr.e_machine);
555                 }

557                 /*
558                  * If ET_REL (.o), printing immediate symbols is likely to
559                  * result in garbage, as symbol lookups on unrelocated
560                  * immediates find false and useless matches.
561                  */

563                 if (ehdr.e_type == ET_REL)
564                         g_flags |= DIS_NOIMMSYM;

566                 if (!g_quiet && dis_tgt_member(current) != NULL)
567                         (void) printf("\narchive member %s\n",
568                             dis_tgt_member(current));

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

578                 if (g_doall) {
```

```
                        /*
                         * With no arguments, iterate over all sections and
                         * disassemble only those that contain text.
                         */
                        dis_tgt_section_iter(current, dis_text_section, dhp);
                } else {
                        callback_arg_t ca;

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

                        /*
                         * If sections or functions were explicitly specified,
                         * resolve those names against the object, and iterate
                         * over just the resulting data.
                         */
                        sections = dis_namelist_resolve_sections(g_seclist,
                            current);
                        functions = dis_namelist_resolve_functions(g_funclist,
                            current);

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

                        dis_scnlist_destroy(sections);
                        dis_funclist_destroy(functions);
                }

                dis_handle_destroy(dhp);
        }

        dis_tgt_destroy(tgt);
}
```

*unchanged_portion_omitted_*

```
**********************************************************
   23336 Fri Aug  1 16:03:18 2014
new/usr/src/cmd/dis/dis_target.c
3317 dis(1) should support cross-target disassembly
**********************************************************
______unchanged_portion_omitted_

729 #if !defined(__sparc)
729 /*
730  * Given an address, return the starting offset of the next symbol in the file.
731  * Only needed on variable length instruction architectures.
732  */
733 off_t
734 dis_tgt_next_symbol(dis_tgt_t *tgt, uint64_t addr)
735 {
736         sym_entry_t *sym;

738         sym = (tgt->dt_symcache != NULL) ? tgt->dt_symcache : tgt->dt_symtab;

740         while (sym != (tgt->dt_symtab + tgt->dt_symcount)) {
741                 if (sym->se_sym.st_value >= addr)
742                         return (sym->se_sym.st_value - addr);
743                 sym++;
744         }

746         return (0);
747 }
749 #endif

749 /*
750  * Iterate over all sections in the target, executing the given callback for
751  * each.
752  */
753 void
754 dis_tgt_section_iter(dis_tgt_t *tgt, section_iter_f func, void *data)
755 {
756         dis_scn_t sdata;
757         Elf_Scn *scn;
758         int idx;

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

763                 if (gelf_getshdr(scn, &sdata.ds_shdr) == NULL) {
764                         warn("%s: failed to get section %d header",
765                             tgt->dt_filename, idx);
766                         continue;
767                 }

769                 if ((sdata.ds_name = elf_strptr(tgt->dt_elf, tgt->dt_shstrndx,
770                     sdata.ds_shdr.sh_name)) == NULL) {
771                         warn("%s: failed to get section %d name",
772                             tgt->dt_filename, idx);
773                         continue;
774                 }

776                 if ((sdata.ds_data = elf_getdata(scn, NULL)) == NULL) {
777                         warn("%s: failed to get data for section '%s'",
778                             tgt->dt_filename, sdata.ds_name);
779                         continue;
780                 }

782                 /*
783                  * dis_tgt_section_iter is also used before the section map
784                  * is initialized, so only check when we need to.  If the
785                  * section map is uninitialized, it will return 0 and have
```



```
786                  * no net effect.
787                  */
788                 if (sdata.ds_shdr.sh_addr == 0)
789                         sdata.ds_shdr.sh_addr = tgt->dt_shnmap[idx].dm_start;

791                 func(tgt, &sdata, data);
792         }
793 }
______unchanged_portion_omitted_
```

```
**********************************************************
    2672 Fri Aug  1 16:03:18 2014
new/usr/src/cmd/dis/dis_target.h
3317 dis(1) should support cross-target disassembly
**********************************************************
```

```
/*
 * 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 2011 Jason King.  All rights reserved.
 */

#ifndef _DIS_TARGET_H
#define _DIS_TARGET_H

#ifdef __cplusplus
extern "C" {
#endif

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

/*
 * Basic types
 */
typedef struct dis_tgt dis_tgt_t;
typedef struct dis_func dis_func_t;
typedef struct dis_scn dis_scn_t;

/*
 * Target management
 */
dis_tgt_t *dis_tgt_create(const char *);
void dis_tgt_destroy(dis_tgt_t *);
const char *dis_tgt_lookup(dis_tgt_t *, uint64_t, off_t *, int, size_t *,
    int *);
const char *dis_find_section(dis_tgt_t *, uint64_t, off_t *);
const char *dis_tgt_name(dis_tgt_t *);
const char *dis_tgt_member(dis_tgt_t *);
void dis_tgt_ehdr(dis_tgt_t *, GElf_Ehdr *);
#if !defined(__sparc)
off_t dis_tgt_next_symbol(dis_tgt_t *, uint64_t);
#endif
dis_tgt_t *dis_tgt_next(dis_tgt_t *);
```



```
/*
 * Section management
 */
typedef void (*section_iter_f)(dis_tgt_t *, dis_scn_t *, void *);
void dis_tgt_section_iter(dis_tgt_t *, section_iter_f, void *);

int dis_section_istext(dis_scn_t *);
void *dis_section_data(dis_scn_t *);
size_t dis_section_size(dis_scn_t *);
uint64_t dis_section_addr(dis_scn_t *);
const char *dis_section_name(dis_scn_t *);
dis_scn_t *dis_section_copy(dis_scn_t *);
void dis_section_free(dis_scn_t *);

/*
 * Function management
 */
typedef void (*function_iter_f)(dis_tgt_t *, dis_func_t *, void *);
void dis_tgt_function_iter(dis_tgt_t *, function_iter_f, void *);
dis_func_t *dis_tgt_function_lookup(dis_tgt_t *, const char *);

void *dis_function_data(dis_func_t *);
size_t dis_function_size(dis_func_t *);
uint64_t dis_function_addr(dis_func_t *);
const char *dis_function_name(dis_func_t *);
dis_func_t *dis_function_copy(dis_func_t *);
void dis_function_free(dis_func_t *);

#ifdef __cplusplus
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    4076 Fri Aug  1 16:03:18 2014
new/usr/src/lib/libdisasm/Makefile.com
patch fix-lint
3317 dis(1) should support cross-target disassembly
**********************************************************
```

```
#
# 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>
# Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
#endif /* ! codereview */
#

#
# 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
#endif /* ! codereview */

#
# 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
```

```
107 LDFLAGS_standalone = $(ZNOVERSION) $(BREDUCE) -dy -r
108 LDFLAGS = $(LDFLAGS_$(CURTYPE))

110 ASFLAGS_standalone = -DDIS_STANDALONE
111 ASFLAGS_library =
112 ASFLAGS += -P $(ASFLAGS_$(CURTYPE)) -D_ASM

114 $(LINTLIB) := SRCS = $(COMDIR)/$(LINTSRC)

116 CERRWARN +=     -_gcc=-Wno-parentheses
117 CERRWARN +=     -_gcc=-Wno-uninitialized

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

126 CPPFLAGS_standalone = -DDIS_STANDALONE -I$(SRC)/cmd/mdb/common
102 # We need to rename some standard functions so we can easily implement them
103 # in consumers.
104 STAND_RENAMED_FUNCS= \
105         snprintf

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

130 # For the x86 disassembler we have to include sources from usr/src/common
131 CPPFLAGS += -I$(SRC)/common/dis/i386 -DDIS_TEXT
112 #
113 # For x86, we have to link to sources in usr/src/common
114 #
115 CPPFLAGS_dis_i386 = -I$(SRC)/common/dis/i386 -DDIS_TEXT
116 CPPFLAGS_dis_sparc =
117 CPPFLAGS +=     $(CPPFLAGS_dis_$(MACH))

133 CFLAGS_standalone = $(STAND_FLAGS_32)
134 CFLAGS_common =
135 CFLAGS += $(CFLAGS_$(CURTYPE)) $(CFLAGS_common)

137 CFLAGS64_standalone = $(STAND_FLAGS_64)
138 CFLAGS64 += $(CCVERBOSE) $(CFLAGS64_$(CURTYPE)) $(CFLAGS64_common)

140 DYNFLAGS +=     $(ZINTERPOSE)

142 .KEEP_STATE:
```

```
**********************************************************
    2492 Fri Aug  1 16:03:19 2014
new/usr/src/lib/libdisasm/Makefile.targ
3317 dis(1) should support cross-target disassembly
**********************************************************
```

```
   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 2007 Sun Microsystems, Inc.  All rights reserved.
  23 # Use is subject to license terms.
  24 #
  25 # ident "%Z%%M% %I%     %E% SMI"
  26 #

  26 #
  27 # We build each flavor in a separate make invocation to improve clarity(!) in
  28 # Makefile.com.  The subordinate makes have $(CURTYPE) set to indicate the
  29 # flavor they're supposed to build.  This causes the correct set of source
  30 # files and compiler and linker flags to be selected.
  31 #
  34 # The SPARC library is built from the closed gate.  This Makefile is shared
  35 # between both environments, so all paths must be absolute.
  36 #

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

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

  37 $(TYPES:%=all.%):
  38         @CURTYPE=$(@:all.%=%) $(MAKE) $@.targ

  40 $(TYPES:%=install.%):
  41         @CURTYPE=$(@:install.%=%) $(MAKE) $@.targ

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

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

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

  51 $(TYPES:%=lint.%):
  52         @CURTYPE=$(@:lint.%=%) $(MAKE) lintcheck

  54 $(STANDLIBRARY): $(OBJS) $(LINKTEST_OBJ)
  55         $(LD) $(BREDUCE) $(ZDEFS) $(LDFLAGS) -o $@.linktest $(OBJS) $(LINKTEST_O
  56         rm $@.linktest
```



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

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

  61 $(TYPES:%=clobber.%):
  62         @CURTYPE=$(@:clobber.%=%) $(MAKE) clobber.targ

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

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

  70 $(PICS): pics
  71 $(OBJS): objs

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

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

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

  85 # install rule for lint library target
  86 $(ROOTLINTDIR)/%: $(COMDIR)/%
  87         $(INS.file)

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

```
**********************************************************
    1162 Fri Aug  1 16:03:19 2014
new/usr/src/lib/libdisasm/amd64/Makefile
3317 dis(1) should support cross-target disassembly
**********************************************************
```

```
   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)
  39 #endif /* ! codereview */
```

```
**********************************************************
    6000 Fri Aug  1 16:03:19 2014
new/usr/src/lib/libdisasm/common/dis_i386.c
patch fix-lint
3317 dis(1) should support cross-target disassembly
**********************************************************
```

```
/*
 * 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>
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
 */

#include <libdisasm.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;
```

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

133         dhx->dhx_dis.d86_data = dhp;

135         return (0);
136 }

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

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

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

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

158         return (0);
159 }

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

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

175 /*
176  * Return the previous instruction.  On x86, we have no choice except to
177  * disassemble everything from the start of the symbol, and stop when we have
178  * reached our instruction address.  If we're not in the middle of a known
179  * symbol, then we return the same address to indicate failure.
180  */
181 static uint64_t
182 dis_i386_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
183 {
184         uint64_t *hist, addr, start;
185         int cur, nseen;
186         uint64_t res = pc;

188         if (n <= 0)
189                 return (pc);

191         if (dhp->dh_lookup(dhp->dh_data, pc, NULL, 0, &start, NULL) != 0 ||
192             start == pc)
```



```
193                 return (res);

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

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

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

207         if (addr != pc) {
208                 /*
209                  * We scanned past %pc, but didn't find an instruction that
210                  * started at %pc.  This means that either the caller specified
211                  * an invalid address, or we ran into something other than code
212                  * during our scan.  Virtually any combination of bytes can be
213                  * construed as a valid Intel instruction, so any non-code bytes
214                  * we encounter will have thrown off the scan.
215                  */
216                 goto done;
217         }

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

221 done:
222         dis_free(hist, sizeof (uint64_t) * n);
223         return (res);
224 }

226 static int
227 dis_i386_supports_flags(int flags)
228 {
229         int archflags = flags & DIS_ARCH_MASK;

231         if (archflags == DIS_X86_SIZE16 || archflags == DIS_X86_SIZE32 ||
232             archflags == DIS_X86_SIZE64)
233                 return (1);

235         return (0);
236 }

238 static int
239 dis_i386_instrlen(dis_handle_t *dhp, uint64_t pc)
240 {
241         if (dis_disassemble(dhp, pc, NULL, 0) != 0)
242                 return (-1);

244         return (dhp->dh_addr - pc);
245 }

247 dis_arch_t dis_arch_i386 = {
248         dis_i386_supports_flags,
249         dis_i386_handle_attach,
250         dis_i386_handle_detach,
251         dis_i386_disassemble,
252         dis_i386_previnstr,
253         dis_i386_min_instrlen,
254         dis_i386_max_instrlen,
255         dis_i386_instrlen,
256 };
257 #endif /* ! codereview */
```

```
**********************************************************
    8995 Fri Aug  1 16:03:19 2014
new/usr/src/lib/libdisasm/common/dis_sparc.c
patch fix-lint
3317 dis(1) should support cross-target disassembly
**********************************************************
```

```
/*
 * 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>
#endif /* ! codereview */
 */

/*
 * 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_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_t *
dis_handle_create(int flags, void *data, dis_lookup_f lookup_func,
    dis_read_f read_func)
{
        dis_handle_sparc_t *dhx;
#endif /* ! codereview */

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

```
 34         dis_handle_t *dhp;

125         /* Validate architecture flags */
126         if ((dhp->dh_flags & (DIS_SPARC_V8|DIS_SPARC_V9|DIS_SPARC_V9_SGI))
127             == 0) {
 36         if ((flags & (DIS_SPARC_V8|DIS_SPARC_V9|DIS_SPARC_V9_SGI)) == 0) {
128                 (void) dis_seterrno(E_DIS_INVALFLAG);
129                 return (-1);
 38                 return (NULL);
130         }

132         if ((dhx = dis_zalloc(sizeof (dis_handle_sparc_t))) == NULL) {
 41         if ((dhp = dis_zalloc(sizeof (struct dis_handle))) == NULL) {
133                 (void) dis_seterrno(E_DIS_NOMEM);
134                 return (NULL);
135         }
136         dhx->dhx_debug = DIS_DEBUG_COMPAT;
137         dhp->dh_arch_private = dhx;

 46         dhp->dh_lookup = lookup_func;
 47         dhp->dh_read = read_func;
 48         dhp->dh_flags = flags;
 49         dhp->dh_data = data;
 50         dhp->dh_debug = DIS_DEBUG_COMPAT;

139 #if !defined(DIS_STANDALONE)

141         opt = getenv("_LIBDISASM_DEBUG");
142         if (opt == NULL)
143                 return (0);
 56                 return (dhp);

145         opt2 = strdup(opt);
146         if (opt2 == NULL) {
147                 dis_handle_destroy(dhp);
148                 dis_free(dhx, sizeof (dis_handle_sparc_t));
149 #endif /* ! codereview */
150                 (void) dis_seterrno(E_DIS_NOMEM);
151                 return (-1);
 61                 return (NULL);
152         }
153         save = opt2;

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

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

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

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

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

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

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



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

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

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

182                 opt2 = end;
183         }
184         free(save);
185 #endif /* DIS_STANDALONE */
186         return (0);
 96         return (dhp);
187 }

189 /* ARGSUSED */
190 static int
191 dis_sparc_max_instrlen(dis_handle_t *dhp)
 99 void
100 dis_handle_destroy(dis_handle_t *dhp)
192 {
193         return (4);
102         dis_free(dhp, sizeof (dis_handle_t));
103 }

105 void
106 dis_set_data(dis_handle_t *dhp, void *data)
107 {
108         dhp->dh_data = data;
109 }

111 void
112 dis_flags_set(dis_handle_t *dhp, int f)
113 {
114         dhp->dh_flags |= f;
115 }

117 void
118 dis_flags_clear(dis_handle_t *dhp, int f)
119 {
120         dhp->dh_flags &= ~f;
194 }

196 /* ARGSUSED */
197 static int
198 dis_sparc_min_instrlen(dis_handle_t *dhp)
124 int
125 dis_max_instrlen(dis_handle_t *dhp)
199 {
200         return (4);
201 }

203 /*
204  * The dis_i386.c comment for this says it returns the previous instruction,
205  * however, I'm fairly sure it's actually returning the _address_ of the
206  * nth previous instruction.
207  */
208 /* ARGSUSED */
209 static uint64_t
210 dis_sparc_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
136 uint64_t
137 dis_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
```

```
 211 {
 212         if (n <= 0)
 213                 return (pc);

 215         if (pc < n)
 216                 return (pc);

 218         return (pc - n*4);
 219 }

 221 /* ARGSUSED */
 222 static int
 223 dis_sparc_instrlen(dis_handle_t *dhp, uint64_t pc)
 149 int
 150 dis_instrlen(dis_handle_t *dhp, uint64_t pc)
 224 {
 225         return (4);
 226 }

 228 static int
 229 dis_sparc_disassemble(dis_handle_t *dhp, uint64_t addr, char *buf,
 230     size_t buflen)
 155 int
 156 dis_disassemble(dis_handle_t *dhp, uint64_t addr, char *buf, size_t buflen)
 231 {
 232         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
 233 #endif /* ! codereview */
 234         const table_t *tp = &initial_table;
 235         const inst_t *inp = NULL;

 237         uint32_t instr;
 238         uint32_t idx = 0;

 240         if (dhp->dh_read(dhp->dh_data, addr, &instr, sizeof (instr)) !=
 241             sizeof (instr))
 242                 return (-1);

 244         dhx->dhx_buf    = buf;
 245         dhx->dhx_buflen = buflen;
 158         dhp->dh_buf    = buf;
 159         dhp->dh_buflen = buflen;
 246         dhp->dh_addr    = addr;

 248         buf[0] = '\0';

 250         /* this allows sparc code to be tested on x86 */
 251 #if !defined(DIS_STANDALONE)
 252 #endif /* ! codereview */
 253         instr = BE_32(instr);
 254 #endif /* DIS_STANDALONE */
 255 #endif /* ! codereview */

 257 #if !defined(DIS_STANDALONE)
 258         if ((dhx->dhx_debug & DIS_DEBUG_PRTBIN) != 0)
 165         if ((dhp->dh_debug & DIS_DEBUG_PRTBIN) != 0)
 259                 do_binary(instr);
 260 #endif /* DIS_STANDALONE */

 262         /* CONSTCOND */
 263         while (1) {
 264                 idx = dis_get_bits(instr, tp->tbl_field, tp->tbl_len);
 265                 inp = &tp->tbl_inp[idx];

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

 269                 if ((inp->in_type == INST_NONE) ||
```



```
 270                     ((inp->in_arch & dhp->dh_flags) == 0))
 271                         goto error;

 273                 if (inp->in_type == INST_TBL) {
 274                         tp = inp->in_data.in_tbl;
 275                         continue;
 276                 }

 278                 break;
 279         }

 281         if (tp->tbl_fmt(dhp, instr, inp, idx) == 0)
 282                 return (0);

 284 error:

 286         (void) dis_snprintf(buf, buflen,
 193         (void) snprintf(buf, buflen,
 287             ((dhp->dh_flags & DIS_OCTAL) != 0) ? "0%011lo" : "0x%08lx",
 288             instr);

 290         return (0);
 291 }
_____unchanged_portion_omitted_
 342 #endif /* DIS_STANDALONE */

 344 static int
 345 dis_sparc_supports_flags(int flags)
 346 {
 347         int archflags = flags & DIS_ARCH_MASK;

 349         if (archflags == DIS_SPARC_V8 ||
 350             (archflags & (DIS_SPARC_V9 | DIS_SPARC_V8)) == DIS_SPARC_V9)
 351                 return (1);

 353         return (0);
 354 }

 356 const dis_arch_t dis_arch_sparc = {
 357         dis_sparc_supports_flags,
 358         dis_sparc_handle_attach,
 359         dis_sparc_handle_detach,
 360         dis_sparc_disassemble,
 361         dis_sparc_previnstr,
 362         dis_sparc_min_instrlen,
 363         dis_sparc_max_instrlen,
 364         dis_sparc_instrlen
 365 };
 366 #endif /* ! codereview */
```

```
**********************************************************
    2261 Fri Aug  1 16:03:19 2014
new/usr/src/lib/libdisasm/common/dis_sparc.h
3317 dis(1) should support cross-target disassembly
**********************************************************
```

```
/*
 * 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.
 */


#ifndef _DIS_SPARC_H
#define _DIS_SPARC_H

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

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/types.h>

#define DIS_DEBUG_NONE          0x00L
#define DIS_DEBUG_COMPAT        0x01L
#define DIS_DEBUG_SYN_ALL       0x02L
#define DIS_DEBUG_PRTBIN        0x04L
#define DIS_DEBUG_PRTFMT        0x08L

#define DIS_DEBUG_ALL DIS_DEBUG_SYN_ALL|DIS_DEBUG_PRTBIN|DIS_DEBUG_PRTFMT

typedef struct dis_handle_sparc {
        char            *dhx_buf;
        size_t          dhx_buflen;
        int             dhx_debug;
} dis_handle_sparc_t;
struct dis_handle {
        void            *dh_data;
        dis_lookup_f    dh_lookup;
        dis_read_f      dh_read;
        int             dh_flags;
```



```
        char            *dh_buf;
        size_t          dh_buflen;
        uint64_t        dh_addr;
        int             dh_debug;
};

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

struct inst;
struct overlay;

typedef struct inst inst_t;
typedef struct overlay overlay_t;

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

typedef struct table {
        const struct inst       *tbl_inp;
        const struct overlay    *tbl_ovp;
        format_fcn              tbl_fmt;
        uint32_t                tbl_field;
        uint32_t                tbl_len;
} table_t;
______unchanged_portion_omitted_
```

```
**********************************************************
   60085 Fri Aug  1 16:03:19 2014
new/usr/src/lib/libdisasm/common/dis_sparc_fmt.c
patch fix-lint
3317 dis(1) should support cross-target disassembly
**********************************************************
```

```
/*
 * 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>
#endif /* ! codereview */
 */


#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;
#endif /* ! codereview */
        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);
695         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,
697             dhp->dh_buf + curlen, dhp->dh_buflen - curlen - 1, NULL,
728             NULL);
729         (void) strlcat(dhx->dhx_buf, ">", dhx->dhx_buflen);
699         (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 #endif /* ! codereview */
740         ifmt_t *f = (ifmt_t *)&instr;

742         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
708         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
743                 prt_field("op", f->f2.op, 2);
744                 prt_field("op2", f->f2.op2, 3);
745                 prt_field("rd", f->f2.rd, 5);
746                 prt_field("imm22", f->f2.imm22, 22);
747         }

749         if (idx == 0) {
750                 /* unimp / illtrap */
751                 prt_name(dhp, inp->in_data.in_def.in_name, 1);
752                 prt_imm(dhp, f->f2.imm22, 0);
753                 return (0);
754         }

756         if (f->f2.imm22 == 0 && f->f2.rd == 0) {
757                 prt_name(dhp, "nop", 0);
758                 return (0);
759         }

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

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

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

771         return (0);
772 }

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

785         char buf[15];
```



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

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

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

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

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

813                 case DISP16:
814                         prt_field("bit 28", ((instr & (1L << 28)) >> 28), 1);
815                         prt_field("rcond", f->f2c.cond, 3);
816                         prt_field("p", f->f2c.p, 1);
817                         prt_field("rs1", f->f2c.rs1, 5);
818                         prt_field("d16hi", f->f2c.d16hi, 2);
819                         prt_field("d16lo", f->f2c.d16lo, 14);
820                         break;
821                 }
822         }

824         if (f->f2b.op2 == 0x01 && idx == 0x00 && f->f2b.p == 1 &&
825             f->f2b.cc == 0x02 && ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) != 0)) {
775             f->f2b.cc == 0x02 && ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) != 0)) {
826                 name = "iprefetch";
827                 flags = FLG_RS1(REG_NONE)|FLG_DISP(DISP19);
828         }


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

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

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

844         }

846         disp *= 4;

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

```
        else
                r = get_regname(dhp, FLG_RS1_VAL(flags), f->f2c.rs1);

        if (r == NULL)
                return (-1);

        if (f->f2a.a == 1)
                annul = ",a";

        if ((flags & FLG_PRED) != 0) {
                if (f->f2b.p == 0) {
                        pred = ",pn";
                } else {
                        if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0)
                        (void) dis_snprintf(buf, sizeof (buf), "%s%s%s", name, annul, pred);
                        if ((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0)
                        (void) snprintf(buf, sizeof (buf), "%s%s%s", name, annul, pred);
                                pred = ",pt";
                }
        }
        prt_name(dhp, buf, 1);


        switch (FLG_DISP_VAL(flags)) {
        case DISP22:
                bprintf(dhp,
                    (octal != 0) ? "%s0%-11lo <" : "%s0x%-10lx <",
                    (disp < 0) ? "-" : "+",
                    (disp < 0) ? (-disp) : disp);
                break;

        case DISP19:
                bprintf(dhp,
                    (octal != 0) ? "%s, %s0%-5lo <" :
                    "%s, %s0x%-04lx <", r,
                    (disp < 0) ? "-" : "+",
                    (disp < 0) ? (-disp) : disp);
                break;

        case DISP16:
                bprintf(dhp,
                    (octal != 0) ? "%s, %s0%-6lo <" : "%s, %s0x%-5lx <",
                    r,
                    (disp < 0) ? "-" : "+",
                    (disp < 0) ? (-disp) : disp);
                break;
        }

        curlen = strlen(dhx->dhx_buf);
        curlen = strlen(dhp->dh_buf);
        dhp->dh_lookup(dhp->dh_data, dhp->dh_addr + (int64_t)disp,
            dhx->dhx_buf + curlen, dhx->dhx_buflen - curlen - 1, NULL, NULL);
            dhp->dh_buf + curlen, dhp->dh_buflen - curlen - 1, NULL, NULL);

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

        return (0);
}



/*
 * print out the compare and swap instructions (casa/casxa)
 * format: casa/casxa [%rs1] imm_asi, %rs2, %rd
```



```
 *          casa/casxa [%rs1] %asi, %rs2, %rd
 *
 * If DIS_DEBUG_SYN_ALL is set, synthetic instructions are emitted
 * when an immediate ASI value is given as follows:
 *
 * casa  [%rs1]#ASI_P, %rs2, %rd    -> cas   [%rs1], %rs2, %rd
 * casa  [%rs1]#ASI_P_L, %rs2, %rd  -> casl  [%rs1], %rs2, %rd
 * casxa [%rs1]#ASI_P, %rs2, %rd    -> casx  [%rs1], %rs2, %rd
 * casxa [%rs1]#ASI_P_L, %rs2, %rd  -> casxl [%rs1], %rs2, %rd
 */
static int
fmt_cas(dis_handle_t *dhp, uint32_t instr, const char *name)
{
        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
#endif /* ! codereview */
        ifmt_t *f = (ifmt_t *)&instr;
        const char *asistr = NULL;
        int noasi = 0;

        asistr = get_asi_name(f->f3.asi);

        if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT)) != 0) {
        if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT)) != 0) {
                if (f->f3.op3 == 0x3c && f->f3.i == 0) {
                        if (f->f3.asi == 0x80) {
                                noasi = 1;
                                name = "cas";
                        }

                        if (f->f3.asi == 0x88) {
                                noasi = 1;
                                name = "casl";
                        }
                }

                if (f->f3.op3 == 0x3e && f->f3.i == 0) {
                        if (f->f3.asi == 0x80) {
                                noasi = 1;
                                name = "casx";
                        }

                        if (f->f3.asi == 0x88) {
                                noasi = 1;
                                name = "casxl";
                        }
                }
        }

        prt_name(dhp, name, 1);

        bprintf(dhp, "[%s]", reg_names[f->f3.rs1]);

        if (noasi == 0) {
                (void) strlcat(dhx->dhx_buf, " ", dhx->dhx_buflen);
                (void) strlcat(dhp->dh_buf, " ", dhp->dh_buflen);
                prt_asi(dhp, instr);
        }

        bprintf(dhp, ", %s, %s", reg_names[f->f3.rs2], reg_names[f->f3.rd]);

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

        return (0);
}
```

```
 975 /*
 976  * format a load/store instruction
 977  * format: ldXX [%rs1 + %rs2], %rd        load, i==0
 978  *          ldXX [%rs1 +/- nn], %rd       load, i==1
 979  *          ldXX [%rs1 + %rs2] #XX, %rd   load w/ imm_asi, i==0
 980  *          ldXX [%rs1 +/- nn] %asi, %rd  load from asi[%asi], i==1
 981  *
 982  *          stXX %rd, [%rs1 + %rs2]       store, i==0
 983  *          stXX %rd, [%rs1 +/- nn]       store, i==1
 984  *          stXX %rd, [%rs1 + %rs1] #XX   store to imm_asi, i==0
 985  *          stXX %rd, [%rs1 +/-nn] %asi   store to asi[%asi], i==1
 986  *
 987  * The register sets used for %rd are set in the instructions flags field
 988  * The asi variants are used if FLG_ASI is set in the instructions flags field
 989  *
 990  * If DIS_DEBUG_SYNTH_ALL or DIS_DEBUG_COMPAT are set,
 991  * When %rs1, %rs2 or nn are 0, they are not printed, i.e.
 992  * [ %rs1 + 0x0 ], %rd -> [%rs1], %rd for example
 993  *
 994  * The following synthetic instructions are also implemented:
 995  *
 996  * stb %g0, [addr] -> clrb [addr]    DIS_DEBUG_SYNTH_ALL
 997  * sth %g0, [addr] -> crlh [addr]    DIS_DEBUG_SYNTH_ALL
 998  * stw %g0, [addr] -> clr  [addr]    DIS_DEBUG_SYNTH_ALL|DIS_DEBUG_COMPAT
 999  * stx %g0, [addr] -> clrx [addr]    DIS_DEBUG_SYNTH_ALL
1000  *
1001  * If DIS_DEBUG_COMPAT is set, the following substitutions also take place
1002  *      lduw -> ld
1003  *      ldtw -> ld
1004  *      stuw -> st
1005  *      sttw -> st
1006  */
1007 int
1008 fmt_ls(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1009 {
1010         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1011 #endif /* ! codereview */
1012         ifmt_t *f = (ifmt_t *)&instr;
1013         const char *regstr = NULL;
1014         const char *asistr = NULL;

1016         const char *iname = inp->in_data.in_def.in_name;
1017         uint32_t flags = inp->in_data.in_def.in_flags;

1019         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
 952         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
1020                 prt_field("op", f->f3.op, 2);
1021                 prt_field("op3", f->f3.op3, 6);
1022                 prt_field("rs1", f->f3.rs1, 5);
1023                 prt_field("i", f->f3.i, 1);
1024                 if (f->f3.i != 0) {
1025                         prt_field("simm13", f->f3a.simm13, 13);
1026                 } else {
1027                         if ((flags & FLG_ASI) != 0)
1028                                 prt_field("imm_asi", f->f3.asi, 8);
1029                         prt_field("rs2", f->f3.rs2, 5);
1030                 }
1031                 prt_field("rd", f->f3.rd, 5);
1032         }

1034         if (idx == 0x2d || idx == 0x3d) {
1035                 /* prefetch / prefetcha */

1037                 prt_name(dhp, iname, 1);

1039                 prt_address(dhp, instr, 0);
```



```
1041                 if (idx == 0x3d) {
1042                         (void) strlcat(dhx->dhx_buf, " ", dhx->dhx_buflen);
 975                         (void) strlcat(dhp->dh_buf, " ", dhp->dh_buflen);
1043                         prt_asi(dhp, instr);
1044                 }

1046                 (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
 979                 (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);

1048                 /* fcn field is the same as rd */
1049                 if (prefetch_str[f->f3.rd] != NULL)
1050                         (void) strlcat(dhx->dhx_buf, prefetch_str[f->f3.rd],
1051                             dhx->dhx_buflen);
 983                         (void) strlcat(dhp->dh_buf, prefetch_str[f->f3.rd],
 984                             dhp->dh_buflen);
1052                 else
1053                         prt_imm(dhp, f->f3.rd, 0);

1055                 if (idx == 0x3d && f->f3.i == 0) {
1056                         asistr = get_asi_name(f->f3.asi);
1057                         if (asistr != NULL)
1058                                 bprintf(dhp, "\t<%s>", asistr);
1059                 }

1061                 return (0);
1062         }

1064         /* casa / casxa */
1065         if (idx == 0x3c || idx == 0x3e)
1066                 return (fmt_cas(dhp, instr, iname));

1068         /* synthetic instructions & special cases */
1069         switch (idx) {
1070         case 0x00:
1071                 /* ld */
1072                 if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
1005                 if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
1073                         iname = "lduw";
1074                 break;

1076         case 0x03:
1077                 if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
1010                 if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
1078                         iname = "ldtw";
1079                 break;

1081         case 0x04:
1082                 /* stw */
1083                 if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
1016                 if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
1084                         iname = "stuw";

1086                 if ((dhp->dh_flags & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1087                     == 0)
1088                         break;

1090                 if (f->f3.rd == 0) {
1091                         iname = "clr";
1092                         flags = FLG_RD(REG_NONE);
1093                 }
1094                 break;

1096         case 0x05:
1097                 /* stb */
1098                 if ((dhp->dh_flags & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
```

```
1099                    == 0)
1100                        break;

1102                if (f->f3.rd == 0) {
1103                        iname = "clrb";
1104                        flags = FLG_RD(REG_NONE);
1105                }
1106                break;

1108        case 0x06:
1109                /* sth */
1110                if ((dhp->dh_flags & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1111                    == 0)
1112                        break;

1114                if (f->f3.rd == 0) {
1115                        iname = "clrh";
1116                        flags = FLG_RD(REG_NONE);
1117                }
1118                break;

1120        case 0x07:
1121                if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
1054                if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
1122                        iname = "sttw";
1123                break;

1125        case 0x0e:
1126                /* stx */

1128                if ((dhp->dh_flags & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1129                    == 0)
1130                        break;

1132                if (f->f3.rd == 0) {
1133                        iname = "clrx";
1134                        flags = FLG_RD(REG_NONE);
1135                }
1136                break;

1138        case 0x13:
1139                /* ldtwa */
1140                if (((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0) &&
1073                if (((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0) &&
1141                    ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0))
1142                        iname = "ldtwa";
1143                break;

1145        case 0x17:
1146                /* sttwa */
1147                if (((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0) &&
1080                if (((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0) &&
1148                    ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0))
1149                        iname = "sttwa";
1150                break;

1152        case 0x21:
1153        case 0x25:
1154                /*
1155                 * on sparcv8 it merely says that rd != 1 should generate an
1156                 * exception, on v9, it is illegal
1157                 */
1158                if ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) == 0)
1159                        break;

1161                iname = (idx == 0x21) ? "ldx" : "stx";
```



```
1163                if (f->f3.rd > 1)
1164                        return (-1);

1166                break;

1168        case 0x31:
1169                /* stda */
1170                switch (f->f3.asi) {
1171                        case 0xc0:
1172                        case 0xc1:
1173                        case 0xc8:
1174                        case 0xc9:
1175                        case 0xc2:
1176                        case 0xc3:
1177                        case 0xca:
1178                        case 0xcb:
1179                        case 0xc4:
1180                        case 0xc5:
1181                        case 0xcc:
1182                        case 0xcd:
1183                                /*
1184                                 * store partial floating point, only valid w/
1185                                 * vis
1186                                 *
1187                                 * Somewhat confusingly, it uses the same op
1188                                 * code as 'stda' -- store double to alternate
1189                                 * space.  It is distinguised by specific
1190                                 * imm_asi values (as seen above), and
1191                                 * has a slightly different output syntax
1192                                 */

1194                                if ((dhp->dh_flags & DIS_SPARC_V9_SGI) == 0)
1195                                        break;
1196                                if (f->f3.i != 0)
1197                                        break;
1198                                prt_name(dhp, iname, 1);
1199                                bprintf(dhp, "%s, %s, [%s] ",
1200                                    get_regname(dhp, REG_FPD, f->f3.rd),
1201                                    get_regname(dhp, REG_FPD, f->f3.rs2),
1202                                    get_regname(dhp, REG_FPD, f->f3.rs1));
1203                                prt_asi(dhp, instr);
1204                                asistr = get_asi_name(f->f3.asi);
1205                                if (asistr != NULL)
1206                                        bprintf(dhp, "\t<%s>", asistr);

1208                                return (0);

1210                        default:
1211                                break;
1212                }

1214        }

1216        regstr = get_regname(dhp, FLG_RD_VAL(flags), f->f3.rd);

1218        if (f->f3.i == 0)
1219                asistr = get_asi_name(f->f3.asi);

1221        prt_name(dhp, iname, 1);

1223        if ((flags & FLG_STORE) != 0) {
1224                if (regstr[0] != '\0') {
1225                        (void) strlcat(dhx->dhx_buf, regstr, dhx->dhx_buflen);
1226                        (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
1158                        (void) strlcat(dhp->dh_buf, regstr, dhp->dh_buflen);
```

```
1159                        (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
1227                }

1229                prt_address(dhp, instr, 0);
1230                if ((flags & FLG_ASI) != 0) {
1231                        (void) strlcat(dhx->dhx_buf, " ", dhx->dhx_buflen);
1164                        (void) strlcat(dhp->dh_buf, " ", dhp->dh_buflen);
1232                        prt_asi(dhp, instr);
1233                }
1234        } else {
1235                prt_address(dhp, instr, 0);
1236                if ((flags & FLG_ASI) != 0) {
1237                        (void) strlcat(dhx->dhx_buf, " ", dhx->dhx_buflen);
1170                        (void) strlcat(dhp->dh_buf, " ", dhp->dh_buflen);
1238                        prt_asi(dhp, instr);
1239                }

1241                if (regstr[0] != '\0') {
1242                        (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
1243                        (void) strlcat(dhx->dhx_buf, regstr, dhx->dhx_buflen);
1175                        (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
1176                        (void) strlcat(dhp->dh_buf, regstr, dhp->dh_buflen);
1244                }
1245        }

1247        if ((flags & FLG_ASI) != 0 && asistr != NULL)
1248                bprintf(dhp, "\t<%s>", asistr);

1250        return (0);
1251 }

1253 static int
1254 fmt_cpop(dis_handle_t *dhp, uint32_t instr, const inst_t *inp)
1255 {
1256        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1257 #endif /* ! codereview */
1258        ifmt_t *f = (ifmt_t *)&instr;
1259        int flags = FLG_P1(REG_CP)|FLG_P2(REG_CP)|FLG_NOIMM|FLG_P3(REG_CP);

1261        if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1189        if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
1262                prt_field("op", f->fcp.op, 2);
1263                prt_field("op3", f->fcp.op3, 6);
1264                prt_field("opc", f->fcp.opc, 9);
1265                prt_field("rs1", f->fcp.rs1, 5);
1266                prt_field("rs2", f->fcp.rs2, 5);
1267                prt_field("rd", f->fcp.rd, 5);
1268        }

1270        prt_name(dhp, inp->in_data.in_def.in_name, 1);
1271        prt_imm(dhp, f->fcp.opc, 0);

1273        (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
1201        (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
1274        (void) prt_aluargs(dhp, instr, flags);

1276        return (0);
1277 }

1279 static int
1280 dis_fmt_rdwr(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1281 {
1282        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1283 #endif /* ! codereview */
1284        const char *psr_str = "%psr";
1285        const char *wim_str = "%wim";
```



```
1286        const char *tbr_str = "%tbr";

1288        const char *name = inp->in_data.in_def.in_name;
1289        const char *regstr = NULL;

1291        ifmt_t *f = (ifmt_t *)&instr;

1293        int rd = (idx < 0x30);
1294        int v9 = (dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI));
1295        int ridx = f->f3.rs1;
1296        int i, first;
1297        int pr_rs1 = 1;
1298        int pr_rs2 = 1;

1300        int use_mask = 1;
1301        uint32_t mask;

1303        if (rd == 0)
1304                ridx = f->f3.rd;

1306        switch (idx) {
1307        case 0x28:
1308                /* rd */

1310                /* stbar */
1311                if ((f->f3.rd == 0) && (f->f3.rs1 == 15) && (f->f3.i == 0)) {
1312                        prt_name(dhp, "stbar", 0);
1313                        return (0);
1314                }

1316                /* membar */
1317                if ((v9 != 0) && (f->f3.rd == 0) && (f->f3.rs1 == 15) &&
1318                    (f->f3.i == 1) && ((f->i & (1L << 12)) == 0)) {

1320                        prt_name(dhp, "membar",
1321                            ((f->fmb.cmask != 0) || (f->fmb.mmask != 0)));

1323                        first = 0;

1325                        for (i = 0; i < 4; ++i) {
1326                                if ((f->fmb.cmask & (1L << i)) != 0) {
1327                                        bprintf(dhp, "%s%s",
1328                                            (first != 0) ? "|" : "",
1329                                            membar_cmask[i]);
1330                                        first = 1;
1331                                }
1332                        }

1334                        for (i = 0; i < 5; ++i) {
1335                                if ((f->fmb.mmask & (1L << i)) != 0) {
1336                                        bprintf(dhp, "%s%s",
1337                                            (first != 0) ? "|" : "",
1338                                            membar_mmask[i]);
1339                                        first = 1;
1340                                }
1341                        }

1343                        return (0);
1344                }

1346                if (v9 != 0) {
1347                        regstr = v9_asr_names[ridx];
1348                        mask = v9_asr_rdmask;
1349                } else {
1350                        regstr = asr_names[ridx];
1351                        mask = asr_rdmask;
```

```
1352                }
1353                break;

1355        case 0x29:
1356                if (v9 != 0) {
1357                        regstr = v9_hprivreg_names[ridx];
1358                        mask = v9_hpr_rdmask;
1359                } else {
1360                        regstr = psr_str;
1361                        use_mask = 0;
1362                }
1363                break;

1365        case 0x2a:
1366                if (v9 != 0) {
1367                        regstr = v9_privreg_names[ridx];
1368                        mask = v9_pr_rdmask;
1369                } else {
1370                        regstr = wim_str;
1371                        use_mask = 0;
1372                }
1373                break;

1375        case 0x2b:
1376                if (v9 != 0) {
1377                        /* flushw */
1378                        prt_name(dhp, name, 0);
1379                        return (0);
1380                }

1382                regstr = tbr_str;
1383                use_mask = 0;
1384                break;

1386        case 0x30:
1387                if (v9 != 0) {
1388                        regstr = v9_asr_names[ridx];
1389                        mask = v9_asr_wrmask;
1390                } else {
1391                        regstr = asr_names[ridx];
1392                        mask = asr_wrmask;
1393                }

1395                /*
1396                 * sir is shoehorned in here, per Ultrasparc 2007
1397                 * hyperprivileged edition, section 7.88, all of
1398                 * these must be true to distinguish from WRasr
1399                 */
1400                if (v9 != 0 && f->f3.rd == 15 && f->f3.rs1 == 0 &&
1401                    f->f3.i == 1) {
1402                        prt_name(dhp, "sir", 1);
1403                        prt_imm(dhp, sign_extend(f->f3a.simm13, 13),
1404                            IMM_SIGNED);
1405                        return (0);
1406                }

1408                /* synth: mov */
1409                if ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1210                *if ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))*
1410                    == 0)
1411                        break;

1413                if (v9 == 0) {
1414                        if (f->f3.rs1 == 0) {
1415                                name = "mov";
1416                                pr_rs1 = 0;
```



```
1417                        }

1419                        if ((f->f3.i == 0 && f->f3.rs2 == 0) ||
1420                            (f->f3.i == 1 && f->f3a.simm13 == 0)) {
1421                                name = "mov";
1422                                pr_rs2 = 0;
1423                        }
1424                }

1426                if (pr_rs1 == 0)
1427                        pr_rs2 = 1;

1429                break;

1431        case 0x31:
1432                /*
1433                 * NOTE: due to the presence of an overlay entry for another
1434                 * table, this case only happens when doing v8 instructions
1435                 * only
1436                 */
1437                regstr = psr_str;
1438                use_mask = 0;
1439                break;

1441        case 0x32:
1442                if (v9 != 0) {
1443                        regstr = v9_privreg_names[ridx];
1444                        mask = v9_pr_wrmask;
1445                } else {
1446                        regstr = wim_str;
1447                        use_mask = 0;
1448                }
1449                break;

1451        case 0x33:
1452                if (v9 != 0) {
1453                        regstr = v9_hprivreg_names[ridx];
1454                        mask = v9_hpr_wrmask;
1455                } else {
1456                        regstr = tbr_str;
1457                        use_mask = 0;
1458                }
1459                break;
1460        }

1462        if (regstr == NULL)
1463                return (-1);

1465        if (use_mask != 0 && ((1L << ridx) & mask) == 0)
1466                return (-1);

1468        prt_name(dhp, name, 1);

1470        if (rd != 0) {
1471                bprintf(dhp, "%s, %s", regstr, reg_names[f->f3.rd]);
1472        } else {
1473                if (pr_rs1 == 1)
1474                        bprintf(dhp, "%s, ", reg_names[f->f3.rs1]);

1476                if (pr_rs2 != 0) {
1477                        if (f->f3.i == 1)
1478                                prt_imm(dhp, sign_extend(f->f3a.simm13, 13),
1479                                    IMM_SIGNED);
1480                        else
1481                                (void) strlcat(dhx->dhx_buf,
1482                                    reg_names[f->f3.rs2], dhx->dhx_buflen);
```

```
1483                        (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
1282                                (void) strlcat(dhp->dh_buf,
1283                                    reg_names[f->f3.rs2], dhp->dh_buflen);
1284                        (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
1484                }

1486                (void) strlcat(dhx->dhx_buf, regstr, dhx->dhx_buflen);
1287                (void) strlcat(dhp->dh_buf, regstr, dhp->dh_buflen);
1487        }

1489        return (0);
1490 }

1492 /* ARGSUSED3 */
1493 int
1494 fmt_trap(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1495 {
1496        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1497 #endif /* ! codereview */
1498        ifmt_t *f = (ifmt_t *)&instr;

1500        int v9 = ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0);
1501        int p_rs1, p_t;

1503        if (f->ftcc.undef != 0)
1504                return (-1);

1506        if (icc_names[f->ftcc.cc] == NULL)
1507                return (-1);

1509        if (f->ftcc.i == 1 && f->ftcc.undef2 != 0)
1510                return (-1);

1512        if (f->ftcc2.i == 0 && f->ftcc2.undef2 != 0)
1513                return (-1);

1515        p_rs1 = ((f->ftcc.rs1 != 0) ||
1516            ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0));
1297            ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0));

1518        if (f->ftcc.i == 0) {
1519                p_t = (f->f3.rs2 != 0 || p_rs1 == 0);

1521                bprintf(dhp, "%-9s %s%s%s%s%s", 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                    (p_t != 0) ? reg_names[f->f3.rs2] : "");
1527        } else {
1528                bprintf(dhp, "%-9s %s%s%s%s0x%x", inp->in_data.in_def.in_name,
1529                    (v9 != 0) ? icc_names[f->ftcc2.cc] : "",
1530                    (v9 != 0) ? ", " : "",
1531                    (p_rs1 != 0) ? reg_names[f->ftcc2.rs1] : "",
1532                    (p_rs1 != 0) ? " + " : "",
1533                    f->ftcc.immtrap);
1534        }
1535        return (0);
1536 }
______unchanged_portion_omitted_

1570 /* ARGSUSED3 */
1571 static int
1572 prt_jmpl(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1573 {
1574        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
```



```
1575 #endif /* ! codereview */
1576        const char *name = inp->in_data.in_def.in_name;
1577        ifmt_t *f = (ifmt_t *)&instr;

1579        if (f->f3.rd == 15 && ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0))
1355        if (f->f3.rd == 15 && ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0))
1580                name = "call";

1582        if (f->f3.rd == 0) {
1583                if (f->f3.i == 1 && f->f3a.simm13 == 8) {
1584                        if (f->f3.rs1 == 15) {
1585                                prt_name(dhp, "retl", 0);
1586                                return (0);
1587                        }

1589                        if (f->f3.rs1 == 31) {
1590                                prt_name(dhp, "ret", 0);
1591                                return (0);
1592                        }
1593                }

1595                name = "jmp";
1596        }

1598        prt_name(dhp, name, 1);
1599        prt_address(dhp, instr, 1);

1601        if (f->f3.rd == 0)
1602                return (0);

1604        if (f->f3.rd == 15 && ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0))
1380        if (f->f3.rd == 15 && ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0))
1605                return (0);

1607        bprintf(dhp, ", %s", reg_names[f->f3.rd]);

1609        return (0);
1610 }

1612 int
1613 fmt_alu(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1614 {
1615        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1616 #endif /* ! codereview */
1617        ifmt_t *f = (ifmt_t *)&instr;

1619        const char *name = inp->in_data.in_def.in_name;
1620        int flags = inp->in_data.in_def.in_flags;
1621        int arg = 0;

1623        if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1391        if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
1624                prt_field("op", f->f3.op, 2);
1625                prt_field("op3", f->f3.op3, 6);
1626                prt_field("rs1", f->f3.rs1, 5);

1628                switch (idx) {
1629                        /* TODO: more formats */

1631                default:
1632                        if (f->f3.i == 0)
1633                                prt_field("rs2", f->f3.rs2, 5);
1634                        else
1635                                prt_field("simm13", f->f3a.simm13, 13);

1637                        prt_field("rd", f->f3.rd, 5);
```

```
1638                }

1640        }

1642        switch (idx) {
1643        case 0x00:
1644                /* add */

1646                if ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) == 0)
1414                if ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) == 0)
1647                        break;

1649                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1650                    f->f3a.simm13 == 1) {
1651                        name = "inc";
1652                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1653                        break;
1654                }

1656                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1657                    f->f3a.simm13 != 1) {
1658                        name = "inc";
1659                        flags = FLG_P1(REG_NONE);
1660                        break;
1661                }
1662                break;

1664        case 0x02:
1665                /* or */

1667                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1435                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1668                    == 0)
1669                        break;

1671                if ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) != 0) {
1439                if ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) != 0) {
1672                        if (f->f3.rs1 == f->f3.rd) {
1673                                name = "bset";
1674                                flags = FLG_P1(REG_NONE);
1675                                break;
1676                        }
1677                }

1679                if (((f->f3.i == 0 && f->f3.rs2 == 0) ||
1680                    (f->f3.i == 1 && f->f3a.simm13 == 0)) &&
1681                    (f->f3.rs1 == 0)) {
1682                        name = "clr";
1683                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1684                        break;
1685                }

1687                if (f->f3.rs1 == 0) {
1688                        name = "mov";
1689                        flags = FLG_P1(REG_NONE);
1690                        break;
1691                }
1692                break;

1694        case 0x04:
1695                /* sub */

1697                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1465                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1698                    == 0)
1699                        break;
```



```
1701                if (f->f3.rs1 == 0 && f->f3.i == 0 && f->f3.rs2 == f->f3.rd) {
1702                        name = "neg";
1703                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE);
1704                        break;
1705                }

1707                if (f->f3.rs1 == 0 && f->f3.i == 0 && f->f3.rs2 != f->f3.rd) {
1708                        name = "neg";
1709                        flags = FLG_P1(REG_NONE);
1710                        break;
1711                }

1713                if ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) == 0)
1481                if ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) == 0)
1714                        break;

1716                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1717                    f->f3a.simm13 == 1) {
1718                        name = "dec";
1719                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1720                        break;
1721                }

1723                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1724                    f->f3a.simm13 != 1) {
1725                        name = "dec";
1726                        flags = FLG_P1(REG_NONE);
1727                        break;
1728                }
1729                break;

1731        case 0x07:
1732                /* xnor */

1734                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1502                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1735                    == 0)
1736                        break;

1738                /*
1739                 * xnor -> not when you have:
1740                 *       xnor %rs1, 0x0 or %g0, %rd
1741                 */
1742                if ((f->f3.i == 0 && f->f3.rs2 != 0) ||
1743                    (f->f3.i == 1 && f->f3a.simm13 != 0))
1744                        break;

1746                name = "not";

1748                if (f->f3.rs1 == f->f3.rd)
1749                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM|
1750                            FLG_P3(REG_INT);
1751                else
1752                        flags = FLG_P1(REG_INT)|FLG_P2(REG_NONE)|FLG_NOIMM|
1753                            FLG_P3(REG_INT);

1755                break;

1757        case 0x10:
1758                /* addcc */

1760                if ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) == 0)
1528                if ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) == 0)
1761                        break;
```

```
1763                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1764                    f->f3a.simm13 == 1) {
1765                        name = "inccc";
1766                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1767                        break;
1768                }

1770                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1771                    f->f3a.simm13 != 1) {
1772                        name = "inccc";
1773                        flags = FLG_P1(REG_NONE);
1774                        break;
1775                }
1776                break;

1778        case 0x11:
1779                /* andcc */

1781                if (f->f3.rd != 0)
1782                        break;

1784                if ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1552                if ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1785                    == 0)
1786                        break;

1788                if (((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0) &&
1556                if (((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0) &&
1789                    ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) == 0))
1790                        break;

1792                name = "btst";
1793                flags = FLG_P1(REG_NONE);
1794                f->f3.rd = f->f3.rs1;
1795                break;

1797        case 0x12:
1798                /* orcc */

1800                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1568                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1801                    == 0)
1802                        break;

1804                if (f->f3.rs1 == 0 && f->f3.rd == 0 && f->f3.i == 0) {
1805                        name = "tst";
1806                        flags = FLG_P1(REG_NONE)|FLG_P3(REG_NONE);
1807                        break;
1808                }

1810                if (f->f3.rs2 == 0 && f->f3.rd == 0 && f->f3.i == 0) {
1811                        name = "tst";
1812                        flags = FLG_P2(REG_NONE)|FLG_P3(REG_NONE);
1813                        break;
1814                }

1816                break;

1818        case 0x14:
1819                /* subcc */

1821                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1589                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1822                    == 0)
1823                        break;
```



```
1825                if (f->f3.rd == 0) {
1826                        name = "cmp";
1827                        flags = FLG_P3(REG_NONE);
1828                        break;
1829                }

1831                if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0)
1599                if ((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0)
1832                        break;

1834                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1835                    f->f3a.simm13 == 1) {
1836                        name = "deccc";
1837                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1838                        break;
1839                }

1841                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1842                    f->f3a.simm13 != 1) {
1843                        name = "deccc";
1844                        flags = FLG_P1(REG_NONE);
1845                        break;
1846                }

1848                break;

1850        case 0x25:
1851        case 0x26:
1852        case 0x27:
1853                return (prt_shift(dhp, instr, inp));

1855        case 0x28:
1856        case 0x29:
1857        case 0x2a:
1858        case 0x2b:
1859        case 0x30:
1860        case 0x31:
1861        case 0x32:
1862        case 0x33:
1863                return (dis_fmt_rdwr(dhp, instr, inp, idx));

1865        case 0x36:
1866        case 0x37:
1867                /* NOTE: overlayed on v9 */
1868                if ((dhp->dh_flags & DIS_SPARC_V8) != 0)
1869                        return (fmt_cpop(dhp, instr, inp));
1870                break;

1872        case 0x38:
1873                /* jmpl */
1874                return (prt_jmpl(dhp, instr, inp, idx));

1876        case 0x39:
1877                /* rett / return */
1878                prt_name(dhp, name, 1);
1879                prt_address(dhp, instr, 1);
1880                return (0);

1882        case 0x3b:
1883                /* flush */
1884                prt_name(dhp, name, 1);
1885                prt_address(dhp, instr, 0);
1886                return (0);

1888        case 0x3c:
1889        case 0x3d:
```

```
1890                 /* save / restore */
1891                 if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1659                 if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1892                     == 0)
1893                         break;

1895                 if (f->f3.rs1 != 0 || f->f3.rs2 != 0 || f->f3.rd != 0)
1896                         break;

1898                 if (f->f3.i != 0 && ((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0))
1666                 if (f->f3.i != 0 && ((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0))
1899                         break;

1901                 prt_name(dhp, name, 0);
1902                 return (0);
1903         }

1905         if (FLG_P1_VAL(flags) != REG_NONE || FLG_P2_VAL(flags) != REG_NONE ||
1906             FLG_P3_VAL(flags) != REG_NONE)
1907                 arg = 1;

1909         prt_name(dhp, name, (arg != 0));
1910         prt_aluargs(dhp, instr, flags);

1912         return (0);
1913 }
______unchanged_portion_omitted_

1938 /* ARGSUSED3 */
1939 int
1940 fmt_movcc(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1941 {
1942         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1943 #endif /* ! codereview */
1944         ifmt_t *f = (ifmt_t *)&instr;
1945         const char **regs = NULL;

1947         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1710         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
1948                 prt_field("op", f->f3c.op, 2);
1949                 prt_field("op3", f->f3c.op3, 6);
1950                 prt_field("cond", f->f3c.cond, 4);
1951                 prt_field("cc2", f->f3c.cc2, 1);
1952                 prt_field("cc", f->f3c.cc, 2);
1953                 prt_field("i", f->f3c.i, 1);

1955                 if (f->f3c.i == 0)
1956                         prt_field("rs2", f->f3.rs2, 5);
1957                 else
1958                         prt_field("simm11", f->f3c.simm11, 11);

1960                 prt_field("rd", f->f3.rd, 5);
1961         }

1963         if (f->f3c.cc2 == 0) {
1964                 regs = fcc_names;
1965         } else {
1966                 regs = icc_names;
1967                 if (regs[f->f3c.cc] == NULL)
1968                         return (-1);
1969         }

1971         prt_name(dhp, inp->in_data.in_def.in_name, 1);

1973         bprintf(dhp, "%s, ", regs[f->f3c.cc]);
```



```
1975         if (f->f3c.i == 1)
1976                 prt_imm(dhp, sign_extend(f->f3c.simm11, 11), IMM_SIGNED);
1977         else
1978                 (void) strlcat(dhx->dhx_buf, reg_names[f->f3.rs2],
1979                     dhx->dhx_buflen);
1741                 (void) strlcat(dhp->dh_buf, reg_names[f->f3.rs2],
1742                     dhp->dh_buflen);

1981         bprintf(dhp, ", %s", reg_names[f->f3.rd]);

1983         return (0);
1984 }

1986 /* ARGSUSED3 */
1987 int
1988 fmt_movr(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1989 {
1990         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1991 #endif /* ! codereview */
1992         ifmt_t *f = (ifmt_t *)&instr;

1994         prt_name(dhp, inp->in_data.in_def.in_name, 1);

1996         bprintf(dhp, "%s, ", reg_names[f->f3d.rs1]);

1998         if (f->f3d.i == 1)
1999                 prt_imm(dhp, sign_extend(f->f3d.simm10, 10), IMM_SIGNED);
2000         else
2001                 (void) strlcat(dhx->dhx_buf, reg_names[f->f3.rs2],
2002                     dhx->dhx_buflen);
1753                 (void) strlcat(dhp->dh_buf, reg_names[f->f3.rs2],
1754                     dhp->dh_buflen);

2004         bprintf(dhp, ", %s", reg_names[f->f3.rd]);

2006         return (0);
2007 }

2009 /* ARGSUSED3 */
2010 int
2011 fmt_fpop1(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
2012 {
2013         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2014 #endif /* ! codereview */
2015         ifmt_t *f = (ifmt_t *)&instr;
2016         int flags = inp->in_data.in_def.in_flags;

2018         flags |= FLG_NOIMM;

2020         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1765         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
2021                 prt_field("op", f->f3.op, 2);
2022                 prt_field("op3", f->f3.op3, 6);
2023                 prt_field("opf", f->fcmp.opf, 9);
2024                 prt_field("rs1", f->f3.rs1, 5);
2025                 prt_field("rs2", f->f3.rs2, 5);
2026                 prt_field("rd", f->f3.rd, 5);
2027         }

2029         prt_name(dhp, inp->in_data.in_def.in_name, 1);
2030         prt_aluargs(dhp, instr, flags);

2032         return (0);
2033 }

2035 int
```

```
2036 fmt_fpop2(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
2037 {
2038         static const char *condstr_icc[16] = {
2039                 "n", "e",  "le", "l",  "leu", "lu",  "neg", "vs",
2040                 "a", "nz", "g",  "ge", "gu",  "geu", "pos", "vc"
2041         };

2043         static const char *condstr_fcc[16] = {
2044                 "n", "nz", "lg", "ul", "l",   "ug", "g",   "u",
2045                 "a", "e",  "ue", "ge", "uge", "le", "ule", "o"
2046         };

2048         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2049 #endif /* ! codereview */
2050         ifmt_t *f = (ifmt_t *)&instr;
2051         const char *ccstr = "";
2052         char name[15];

2054         int flags = inp->in_data.in_def.in_flags;
2055         int is_cmp = (idx == 0x51 || idx == 0x52 || idx == 0x53 ||
2056             idx == 0x55 || idx == 0x56 || idx == 0x57);
2057         int is_fmov = (idx & 0x3f);
2058         int is_v9 = ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0);
2059         int is_compat = ((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0);
1793         int is_compat = ((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0);

2061         int p_cc = 0;

2063         is_fmov = (is_fmov == 0x1 || is_fmov == 0x2 || is_fmov == 0x3);

2065         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1799         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
2066                 prt_field("op", f->f3.op, 2);
2067                 prt_field("op3", f->f3.op3, 6);
2068                 prt_field("opf", f->fcmp.opf, 9);

2070                 switch (idx & 0x3f) {
2071                 case 0x51:
2072                 case 0x52:
2073                 case 0x53:
2074                 case 0x55:
2075                 case 0x56:
2076                 case 0x57:
2077                         prt_field("cc", f->fcmp.cc, 2);
2078                         prt_field("rs1", f->f3.rs1, 5);
2079                         prt_field("rs2", f->f3.rs2, 5);
2080                         break;

2082                 case 0x01:
2083                 case 0x02:
2084                 case 0x03:
2085                         prt_field("opf_low", f->fmv.opf, 6);
2086                         prt_field("cond", f->fmv.cond, 4);
2087                         prt_field("opf_cc", f->fmv.cc, 3);
2088                         prt_field("rs2", f->fmv.rs2, 5);
2089                         break;

2091                 default:
2092                         prt_field("rs1", f->f3.rs1, 5);
2093                         prt_field("rs2", f->f3.rs2, 5);
2094                         prt_field("rd", f->f3.rd, 5);
2095                 }
2096         }

2098         name[0] = '\0';
2099         (void) strlcat(name, inp->in_data.in_def.in_name, sizeof (name));
```



```
2101         if (is_fmov != 0) {
2102                 (void) strlcat(name,
2103                     (f->fmv.cc < 4) ? condstr_fcc[f->fmv.cond]
2104                     : condstr_icc[f->fmv.cond],
2105                     sizeof (name));
2106         }

2108         prt_name(dhp, name, 1);

2110         if (is_cmp != 0)
2111                 ccstr = fcc_names[f->fcmp.cc];

2113         if (is_fmov != 0)
2114                 ccstr = (f->fmv.cc < 4) ? fcc_names[f->fmv.cc & 0x3]
2115                     : icc_names[f->fmv.cc & 0x3];

2117         if (ccstr == NULL)
2118                 return (-1);

2120         p_cc = (is_compat == 0 || is_v9 != 0 ||
2121             (is_cmp != 0 && f->fcmp.cc != 0) ||
2122             (is_fmov != 0 && f->fmv.cc != 0));

2124         if (p_cc != 0)
2125                 bprintf(dhp, "%s, ", ccstr);

2127         prt_aluargs(dhp, instr, flags);

2129         return (0);
2130 }

2132 int
2133 fmt_vis(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
2134 {
2135         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2136 #endif /* ! codereview */
2137         ifmt_t *f = (ifmt_t *)&instr;
2138         int flags = inp->in_data.in_def.in_flags;

2140         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1869         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
2141                 prt_field("op", f->f3.op, 2);
2142                 prt_field("op3", f->f3.op3, 6);
2143                 prt_field("opf", f->fcmp.opf, 9);

2145                 if (idx == 0x081) {
2146                         prt_field("mode", instr & 02L, 2);
2147                 } else {
2148                         prt_field("rs1", f->f3.rs1, 5);
2149                         prt_field("rs2", f->f3.rs2, 5);
2150                         prt_field("rd", f->f3.rd, 5);
2151                 }
2152         }

2154         prt_name(dhp, inp->in_data.in_def.in_name, 1);

2156         if (idx == 0x081) {
2157                 /* siam */
2158                 bprintf(dhp, "%d", instr & 0x7L);
2159                 return (0);
2160         }

2162         prt_aluargs(dhp, instr, flags);

2164         return (0);
```

```
2165 }
```

*unchanged_portion_omitted_*

```
2271 /*
2272  * return the symbolic name of a register
2273  * regset is one of the REG_* values indicating which type of register it is
2274  * such as integer, floating point, etc.
2275  * idx is the numeric value of the register
2276  *
2277  * If regset is REG_NONE, an empty, but non-NULL string is returned
2278  * NULL may be returned if the index indicates an invalid register value
2279  * such as with the %icc/%xcc sets
2280  */
2281 static const char *
2282 get_regname(dis_handle_t *dhp, int regset, uint32_t idx)
2283 {
2284         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2285 #endif /* ! codereview */
2286         const char *regname = NULL;

2288         switch (regset) {
2289         case REG_INT:
2290                 regname = reg_names[idx];
2291                 break;

2293         case REG_FP:
2294                 regname = freg_names[idx];
2295                 break;

2297         case REG_FPD:
2298                 if (((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0) ||
2013                 if (((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0) ||
2299                     ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0))
2300                         regname = fdreg_names[idx];
2301                 else
2302                         regname = compat_fdreg_names[idx];

2304                 break;

2306         case REG_FPQ:
2307                 if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
2022                 if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
2308                         regname = fqreg_names[idx];
2309                 else
2310                         regname = freg_names[idx];

2312                 break;

2314         case REG_CP:
2315                 regname = cpreg_names[idx];
2316                 break;

2318         case REG_ICC:
2319                 regname = icc_names[idx];
2320                 break;

2322         case REG_FCC:
2323                 regname = fcc_names[idx];
2324                 break;

2326         case REG_FSR:
2327                 regname = "%fsr";
2328                 break;

2330         case REG_CSR:
2331                 regname = "%csr";
```



```
2332                 break;

2334         case REG_CQ:
2335                 regname = "%cq";
2336                 break;

2338         case REG_NONE:
2339                 regname = "";
2340                 break;
2341         }

2343         return (regname);
2344 }
```

*unchanged_portion_omitted_*

```
2365 /*
2366  * put an address expression into the output buffer
2367  *
2368  * instr is the instruction to use
2369  * if nobrackets != 0, [] are not added around the instruction
2370  *
2371  * Currently this option is set when printing out the address portion
2372  * of a jmpl instruction, but otherwise 0 for load/stores
2373  *
2374  * If no debug flags are set, the full expression is output, even when
2375  * %g0 or 0x0 appears in the address
2376  *
2377  * If DIS_DEBUG_SYN_ALL or DIS_DEBUG_COMPAT are set, when %g0 or 0x0
2378  * appear in the address, they are not output.  If the wierd (and probably
2379  * shouldn't happen) address of [%g0 + %g0] or [%g0 + 0x0] is encountered,
2380  * [%g0] is output
2381  */
2382 static void
2383 prt_address(dis_handle_t *dhp, uint32_t instr, int nobrackets)
2384 {
2385         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2386 #endif /* ! codereview */
2387         ifmt_t *f = (ifmt_t *)&instr;
2388         int32_t simm13;
2389         int octal = ((dhp->dh_flags & DIS_OCTAL) != 0);
2390         int p1 = ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0);
2391         int p2 = ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0);
2100         int p1 = ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0);
2101         int p2 = ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0);

2393         if (f->f3a.i == 0) {
2394                 p1 |= ((f->f3a.rs1 != 0) || f->f3.rs2 == 0);
2395                 p2 |= (f->f3.rs2 != 0);

2397                 bprintf(dhp, "%s%s%s%s%s",
2398                     (nobrackets == 0) ? "[" : "",
2399                     (p1 != 0) ? reg_names[f->f3a.rs1] : "",
2400                     (p1 != 0 && p2 != 0) ? " + " : "",
2401                     (p2 != 0) ? reg_names[f->f3.rs2] : "",
2402                     (nobrackets == 0) ? "]" : "");
2403         } else {
2404                 const char *sign;

2406                 simm13 = sign_extend(f->f3a.simm13, 13);
2407                 sign = (simm13 < 0) ? "-" : "+";

2409                 p1 |= (f->f3a.rs1 != 0);
2410                 p2 |= (p1 == 0 || simm13 != 0);

2412                 if (p1 == 0 && simm13 == 0)
2413                         p2 = 1;
```

```
2415                 if (p1 == 0 && simm13 >= 0)
2416                         sign = "";

2418                 if (p2 != 0)
2419                         bprintf(dhp,
2420                             (octal != 0) ? "%s%s%s%s%s0%lo%s" :
2421                             "%s%s%s%s%s0x%lx%s",
2422                             (nobrackets == 0) ? "[" : "",
2423                             (p1 != 0) ? reg_names[f->f3a.rs1] : "",
2424                             (p1 != 0) ? " " : "",
2425                             sign,
2426                             (p1 != 0) ? " " : "",
2427                             (simm13 < 0) ? -(simm13) : simm13,
2428                             (nobrackets == 0) ? "]" : "");
2429                 else
2430                         bprintf(dhp, "%s%s%s",
2431                             (nobrackets == 0) ? "[" : "",
2432                             reg_names[f->f3a.rs1],
2433                             (nobrackets == 0) ? "]" : "");
2434         }
2435 }

2437 /*
2438  * print out the arguments to an alu operation (add, sub, etc.)
2439  * conatined in 'instr'
2440  *
2441  * alu instructions have the following format:
2442  *       %rs1, %rs2, %rd    (i == 0)
2443  *       %rs1, 0xnnn, %rd   (i == 1)
2444  *         ^    ^       ^
2445  *         |    |       |
2446  *        p1    p2      p3
2447  *
2448  * flags indicates the register set to use for each position (p1, p2, p3)
2449  * as well as if immediate values (i == 1) are allowed
2450  *
2451  * if flags indicates a specific position has REG_NONE set as it's register
2452  * set, it is omitted from the output.  This is primarly used for certain
2453  * floating point operations
2454  */
2455 static void
2456 prt_aluargs(dis_handle_t *dhp, uint32_t instr, uint32_t flags)
2457 {
2458         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2459 #endif /* ! codereview */
2460         ifmt_t *f = (ifmt_t *)&instr;
2461         const char *r1, *r2, *r3;
2462         int p1, p2, p3;
2463         unsigned int opf = 0;

2465         r1 = get_regname(dhp, FLG_P1_VAL(flags), f->f3.rs1);
2466         r2 = get_regname(dhp, FLG_P2_VAL(flags), f->f3.rs2);
2467         r3 = get_regname(dhp, FLG_P3_VAL(flags), f->f3.rd);

2469         p1 = (FLG_P1_VAL(flags) != REG_NONE);
2470         p2 = (((flags & FLG_NOIMM) == 0) || (FLG_P2_VAL(flags) != REG_NONE));
2471         p3 = (FLG_RD_VAL(flags) != REG_NONE);

2473         if (r1 == NULL || r1[0] == '\0')
2474                 p1 = 0;

2476         if (f->f3a.i == 0 && (r2 == NULL || r2[0] == '\0'))
2477                 p2 = 0;

2479         if (r3 == NULL || r3[0] == '\0')
```



```
2480                 p3 = 0;

2482         if ((f->fcmp.op == 2) && (f->fcmp.op3 == 0x36) && (f->fcmp.cc != 0))
2483                 opf = f->fcmp.opf;

2485         if ((opf == 0x151) || (opf == 0x152)) {
2486                 (void) strlcat(dhx->dhx_buf, r3, dhx->dhx_buflen);
2487                 (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
2168                 (void) strlcat(dhp->dh_buf, r3, dhp->dh_buflen);
2169                 (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
2488                 p3 = 0;
2489         }

2491         if (p1 != 0) {
2492                 (void) strlcat(dhx->dhx_buf, r1, dhx->dhx_buflen);
2174                 (void) strlcat(dhp->dh_buf, r1, dhp->dh_buflen);
2493                 if (p2 != 0 || p3 != 0)
2494                         (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
2176                         (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
2495         }

2497         if (p2 != 0) {
2498                 if (f->f3.i == 0 || ((flags & FLG_NOIMM) != 0))
2499                         (void) strlcat(dhx->dhx_buf, r2, dhx->dhx_buflen);
2181                         (void) strlcat(dhp->dh_buf, r2, dhp->dh_buflen);
2500                 else
2501                         prt_imm(dhp, sign_extend(f->f3a.simm13, 13),
2502                             IMM_SIGNED);

2504                 if (p3 != 0)
2505                         (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
2187                         (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
2506         }

2508         if (p3 != 0)
2509                 (void) strlcat(dhx->dhx_buf, r3, dhx->dhx_buflen);
2191                 (void) strlcat(dhp->dh_buf, r3, dhp->dh_buflen);
2510 }
______unchanged_portion_omitted_

2770 /*
2771  * just a handy function that takes care of managing the buffer length
2772  * w/ printf
2773  */

2775 /*
2776  * PRINTF LIKE 1
2777  */
2778 static void
2779 bprintf(dis_handle_t *dhp, const char *fmt, ...)
2780 {
2781         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2782 #endif /* ! codereview */
2783         size_t curlen;
2784         va_list ap;

2786         curlen = strlen(dhx->dhx_buf);
2463         curlen = strlen(dhp->dh_buf);

2788         va_start(ap, fmt);
2789         (void) dis_vsnprintf(dhx->dhx_buf + curlen, dhx->dhx_buflen -
2790             curlen, fmt, ap);
2466         (void) vsnprintf(dhp->dh_buf + curlen, dhp->dh_buflen - curlen, fmt,
2467             ap);
2791         va_end(ap);
2792 }
```

```
**********************************************************
    3871 Fri Aug  1 16:03:20 2014
new/usr/src/lib/libdisasm/common/dis_sparc_fmt.h
3317 dis(1) should support cross-target disassembly
**********************************************************
```

```
   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_
```

```
**********************************************************
    4934 Fri Aug  1 16:03:20 2014
new/usr/src/lib/libdisasm/common/libdisasm.c
patch fix-lint
3317 dis(1) should support cross-target disassembly
**********************************************************
```

```
/*
 * 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>
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
#endif /* ! codereview */
 */

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

#include <libdisasm.h>
#include <stdlib.h>
#ifdef DIS_STANDALONE
#include <mdb/mdb_modapi.h>
#define _MDB
#include <mdb/mdb_io.h>
#else
#include <stdio.h>
#endif /* ! codereview */
#endif

#include "libdisasm_impl.h"

#endif /* ! codereview */
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
};

/*
#endif /* ! codereview */
 * 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));
}

void
dis_free(void *ptr, size_t bytes)
{
        mdb_free(ptr, bytes);
}
#else
void *
dis_zalloc(size_t bytes)
{
        return (calloc(1, bytes));
}

/*ARGSUSED*/
void
dis_free(void *ptr, size_t bytes)
{
        free(ptr);
}
#endif

int
dis_seterrno(int error)
{
        _dis_errno = error;
        return (-1);
}

int
dis_errno(void)
{
        return (_dis_errno);
}

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");
#endif /* ! codereview */
```

```
125         default:
126                 return ("unknown error");
127         }
128 }

130 void
131 dis_set_data(dis_handle_t *dhp, void *data)
132 {
133         dhp->dh_data = data;
134 }

136 void
137 dis_flags_set(dis_handle_t *dhp, int f)
138 {
139         dhp->dh_flags |= f;
140 }

142 void
143 dis_flags_clear(dis_handle_t *dhp, int f)
144 {
145         dhp->dh_flags &= ~f;
146 }

148 void
149 dis_handle_destroy(dis_handle_t *dhp)
150 {
151         dhp->dh_arch->da_handle_detach(dhp);
152         dis_free(dhp, sizeof (dis_handle_t));
153 }

155 dis_handle_t *
156 dis_handle_create(int flags, void *data, dis_lookup_f lookup_func,
157     dis_read_f read_func)
158 {
159         dis_handle_t *dhp;
160         dis_arch_t *arch = NULL;
161         int i;

163         /* Select an architecture based on flags */
164         for (i = 0; dis_archs[i] != NULL; i++) {
165                 if (dis_archs[i]->da_supports_flags(flags)) {
166                         arch = dis_archs[i];
167                         break;
168                 }
169         }
170         if (arch == NULL) {
171                 (void) dis_seterrno(E_DIS_UNSUPARCH);
172                 return (NULL);
173         }

175         if ((dhp = dis_zalloc(sizeof (dis_handle_t))) == NULL) {
176                 (void) dis_seterrno(E_DIS_NOMEM);
177                 return (NULL);
178         }
179         dhp->dh_arch = arch;
180         dhp->dh_lookup = lookup_func;
181         dhp->dh_read = read_func;
182         dhp->dh_flags = flags;
183         dhp->dh_data = data;

185         /*
186          * Allow the architecture-specific code to allocate
187          * its private data.
188          */
189         if (arch->da_handle_attach(dhp) != 0) {
190                 dis_free(dhp, sizeof (dis_handle_t));
```



```
191                 /* dis errno already set */
192                 return (NULL);
193         }

195         return (dhp);
196 }

198 int
199 dis_disassemble(dis_handle_t *dhp, uint64_t addr, char *buf, size_t buflen)
200 {
201         return (dhp->dh_arch->da_disassemble(dhp, addr, buf, buflen));
202 }

204 uint64_t
205 dis_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
206 {
207         return (dhp->dh_arch->da_previnstr(dhp, pc, n));
208 }

210 int
211 dis_min_instrlen(dis_handle_t *dhp)
212 {
213         return (dhp->dh_arch->da_min_instrlen(dhp));
214 }

216 int
217 dis_max_instrlen(dis_handle_t *dhp)
218 {
219         return (dhp->dh_arch->da_max_instrlen(dhp));
220 }

222 int
223 dis_instrlen(dis_handle_t *dhp, uint64_t pc)
224 {
225         return (dhp->dh_arch->da_instrlen(dhp, pc));
226 }

228 int
229 dis_vsnprintf(char *restrict s, size_t n, const char *restrict format,
230     va_list args)
231 {
232 #ifdef DIS_STANDALONE
233         return (mdb_iob_vsnprintf(s, n, format, args));
234 #else
235         return (vsnprintf(s, n, format, args));
236 #endif
237 }

239 int
240 dis_snprintf(char *restrict s, size_t n, const char *restrict format, ...)
241 {
242         va_list args;

244         va_start(args, format);
245         n = dis_vsnprintf(s, n, format, args);
246         va_end(args);

248         return (n);
249 }
250 #endif /* ! codereview */
```

```
**********************************************************
    2684 Fri Aug  1 16:03:20 2014
new/usr/src/lib/libdisasm/common/libdisasm.h
3317 dis(1) should support cross-target disassembly
**********************************************************
```

```
/*
 * 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>
#endif /* ! codereview */
 */

#ifndef _LIBDISASM_H
#define _LIBDISASM_H

#include <sys/types.h>

#ifdef  __cplusplus
extern "C" {
#endif

typedef struct dis_handle dis_handle_t;

#define DIS_DEFAULT             0x0

/* SPARC disassembler flags */
#define DIS_SPARC_V8            0x001
#define DIS_SPARC_V9            0x002
#define DIS_SPARC_V9_SGI        0x004
#define DIS_SPARC_V9_OPL        0x008

/* x86 diassembler flags */
#define DIS_X86_SIZE16          0x100
#define DIS_X86_SIZE32          0x010
#define DIS_X86_SIZE64          0x020
#define DIS_SPARC_V8            0x01
#define DIS_SPARC_V9            0x02
#define DIS_SPARC_V9_SGI        0x04
#define DIS_SPARC_V9_OPL        0x08

/* x86 diassembler flags (mutually exclusive) */
#define DIS_X86_SIZE16          0x08
#define DIS_X86_SIZE32          0x10
#define DIS_X86_SIZE64          0x20
```



```
/* generic disassembler flags */
#define DIS_OCTAL               0x040
#define DIS_NOIMMSYM            0x080

#define DIS_ARCH_MASK           (DIS_SPARC_V8 | \
                DIS_SPARC_V9 | DIS_SPARC_V9_SGI | DIS_SPARC_V9_OPL | \
                DIS_X86_SIZE16 | DIS_X86_SIZE32 | DIS_X86_SIZE64)
#define DIS_OCTAL               0x40
#define DIS_NOIMMSYM            0x80

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

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

extern int dis_disassemble(dis_handle_t *, uint64_t, char *, size_t);
extern uint64_t dis_previnstr(dis_handle_t *, uint64_t, int n);
extern void dis_set_data(dis_handle_t *, void *);
extern void dis_flags_set(dis_handle_t *, int f);
extern void dis_flags_clear(dis_handle_t *, int f);
extern int dis_max_instrlen(dis_handle_t *);
extern int dis_min_instrlen(dis_handle_t *);
#endif /* ! codereview */
extern int dis_instrlen(dis_handle_t *, uint64_t);

/* libdisasm errors */
#define E_DIS_NOMEM             1       /* Out of memory */
#define E_DIS_INVALFLAG         2       /* Invalid flag for this architecture */
#define E_DIS_UNSUPARCH         3       /* Unsupported architecture */
#endif /* ! codereview */

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

#ifdef  __cplusplus
}
#endif

#endif  /* _LIBDISASM_H */
```

```
**********************************************************
    2087 Fri Aug  1 16:03:20 2014
new/usr/src/lib/libdisasm/common/libdisasm_impl.h
patch fix-lint
3317 dis(1) should support cross-target disassembly
**********************************************************
```

```
/*
 * 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>
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
#endif /* ! codereview */
 */

#ifndef _LIBDISASM_IMPL_H
#define _LIBDISASM_IMPL_H

#include <stdarg.h>
#include <sys/sysmacros.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 *);
        int (*da_instrlen)(dis_handle_t *, uint64_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;
```



```
};

#endif /* ! codereview */
extern int dis_seterrno(int);

extern void *dis_zalloc(size_t);
extern void dis_free(void *, size_t);
extern int dis_vsnprintf(char *restrict, size_t, const char *restrict, va_list);
extern int dis_snprintf(char *restrict, size_t, const char *restrict, ...);
#endif /* ! codereview */

#ifdef  __cplusplus
}
#endif

#endif  /* _LIBDISASM_IMPL_H */
```

```
**********************************************************
    1331 Fri Aug  1 16:03:20 2014
new/usr/src/lib/libdisasm/common/linktest_stand.c
patch fix-lint
**********************************************************
/*
 * 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.
 */

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

/*
 * This file is used to verify that the standalone's external dependencies
 * haven't changed in a way that'll break things that use it.
 */

void mdb_free(void) {}
void mdb_snprintf(void) {}
void mdb_iob_vsnprintf(void) {}
void snprintf(void) {}
void vsnprintf(void) {}
void mdb_zalloc(void) {}
void strcmp(void) {}
void strlen(void) {}
void strlcat(void) {}
void strncpy(void) {}
void strncmp(void) {}
void memcpy(void) {}
void _memcpy(void) {}
```

```
***********************************************************
    1519 Fri Aug  1 16:03:20 2014
new/usr/src/lib/libdisasm/common/mapfile-vers
3317 dis(1) should support cross-target disassembly
***********************************************************
```

```
#
# 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 (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
#

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

$mapfile_version 2

SYMBOL_VERSION SUNWprivate_1.1 {
    global:
        dis_disassemble;
        dis_errno;
        dis_handle_create;
        dis_handle_destroy;
        dis_instrlen;
        dis_max_instrlen;
        dis_min_instrlen;
#endif /* ! codereview */
        dis_previnstr;
        dis_set_data;
        dis_flags_set;
        dis_flags_clear;
        dis_strerror;
    local:
        *;
};
```

```
**********************************************************
    1115 Fri Aug  1 16:03:21 2014
new/usr/src/lib/libdisasm/i386/Makefile
3317 dis(1) should support cross-target disassembly
**********************************************************
   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)
  38 #endif /* ! codereview */
```