```
**********************************************************
   18333 Tue Jul 29 20:47:46 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 Tue Jul 29 20:47:47 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 Tue Jul 29 20:47:47 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_
```

```
**********************************************************
    4210 Tue Jul 29 20:47:47 2014
new/usr/src/lib/libdisasm/Makefile.com
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 */
#

#
# 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

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

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

113 $(LINTLIB) := SRCS = $(COMDIR)/$(LINTSRC)

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

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

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

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

135 # For the x86 disassembler we have to include sources from usr/src/common
136 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))

138 CFLAGS_standalone = $(STAND_FLAGS_32)
139 CFLAGS_common =
140 CFLAGS += $(CFLAGS_$(CURTYPE)) $(CFLAGS_common)

142 CFLAGS64_standalone = $(STAND_FLAGS_64)
143 CFLAGS64 += $(CCVERBOSE) $(CFLAGS64_$(CURTYPE)) $(CFLAGS64_common)

145 DYNFLAGS +=     $(ZINTERPOSE)

147 .KEEP_STATE:
```

```
**********************************************************
    2492 Tue Jul 29 20:47:47 2014
new/usr/src/lib/libdisasm/Makefile.targ
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.
#
# ident "%Z%%M% %I%     %E% SMI"
#

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

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

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

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

```
**********************************************************
    1162 Tue Jul 29 20:47:47 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 */
```

```
**********************************************************
    6016 Tue Jul 29 20:47:48 2014
new/usr/src/lib/libdisasm/common/dis_i386.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 2012 Joshua M. Clulow <josh@sysmgr.org>
 */

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

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

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

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

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

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

        return (1);
}
```



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

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

        dhp->dh_addr++;

        return ((int)byte);
}

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

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

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

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

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

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

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

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

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

134         dhx->dhx_dis.d86_data = dhp;

136         return (0);
137 }

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

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

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

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

159         return (0);
160 }

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

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

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

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

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



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

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

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

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

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

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

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

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

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

238         return (0);
239 }

241 static int
242 dis_i386_instrlen(dis_handle_t *dhp, uint64_t pc)
243 {
244         if (dis_disassemble(dhp, pc, NULL, 0) != 0)
245                 return (-1);

247         return (dhp->dh_addr - pc);
248 }

250 dis_arch_t dis_arch_i386 = {
251         dis_i386_supports_flags,
252         dis_i386_handle_attach,
253         dis_i386_handle_detach,
254         dis_i386_disassemble,
255         dis_i386_previnstr,
256         dis_i386_min_instrlen,
257         dis_i386_max_instrlen,
258         dis_i386_instrlen,
259 };
```

```
260 #endif /* ! codereview */
```

```
**********************************************************
    8991 Tue Jul 29 20:47:48 2014
new/usr/src/lib/libdisasm/common/dis_sparc.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 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
        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) 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 Tue Jul 29 20:47:48 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_
```

```
**********************************************************
   60194 Tue Jul 29 20:47:48 2014
new/usr/src/lib/libdisasm/common/dis_sparc_fmt.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 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;
#elif defined(_BIT_FIELDS_LTOH)
typedef struct format1 {
        uint32_t disp30:30;
        uint32_t op:2;
} format1_t;
#else
#error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
#endif

#if defined(_BIT_FIELDS_HTOL)
typedef struct format2 {
        uint32_t op:2;
        uint32_t rd:5;
        uint32_t op2:3;
        uint32_t imm22:22;
} format2_t;
#elif defined(_BIT_FIELDS_LTOH)
typedef struct format2 {
        uint32_t imm22:22;
        uint32_t op2:3;
        uint32_t rd:5;
        uint32_t op:2;
} format2_t;
#else
#error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
#endif

#if defined(_BIT_FIELDS_HTOL)
typedef struct format2a {
        uint32_t op:2;
        uint32_t a:1;
        uint32_t cond:4;
        uint32_t op2:3;
        uint32_t disp22:22;
} format2a_t;
#elif defined(_BIT_FIELDS_LTOH)
typedef struct format2a {
        uint32_t disp22:22;
```

```
128         uint32_t op2:3;
129         uint32_t cond:4;
130         uint32_t a:1;
131         uint32_t op:2;
132 } format2a_t;
133 #else
134 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
135 #endif

137 #if defined(_BIT_FIELDS_HTOL)
138 typedef struct format2b {
139         uint32_t op:2;
140         uint32_t a:1;
141         uint32_t cond:4;
142         uint32_t op2:3;
143         uint32_t cc:2;
144         uint32_t p:1;
145         uint32_t disp19:19;
146 } format2b_t;
147 #elif defined(_BIT_FIELDS_LTOH)
148 typedef struct format2b {
149         uint32_t disp19:19;
150         uint32_t p:1;
151         uint32_t cc:2;
152         uint32_t op2:3;
153         uint32_t cond:4;
154         uint32_t a:1;
155         uint32_t op:2;
156 } format2b_t;
157 #else
158 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
159 #endif

161 #if defined(_BIT_FIELDS_HTOL)
162 typedef struct format2c {
163         uint32_t op:2;
164         uint32_t a:1;
165         uint32_t cond:4;
166         uint32_t op2:3;
167         uint32_t d16hi:2;
168         uint32_t p:1;
169         uint32_t rs1:5;
170         uint32_t d16lo:14;
171 } format2c_t;
172 #elif defined(_BIT_FIELDS_LTOH)
173 typedef struct format2c {
174         uint32_t d16lo:14;
175         uint32_t rs1:5;
176         uint32_t p:1;
177         uint32_t d16hi:2;
178         uint32_t op2:3;
179         uint32_t cond:4;
180         uint32_t a:1;
181         uint32_t op:2;
182 } format2c_t;
183 #else
184 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
185 #endif

187 #if defined(_BIT_FIELDS_HTOL)
188 typedef struct format3 {
189         uint32_t op:2;
190         uint32_t rd:5;
191         uint32_t op3:6;
192         uint32_t rs1:5;
193         uint32_t i:1;
```



```
194         uint32_t asi:8;
195         uint32_t rs2:5;
196 } format3_t;
197 #elif defined(_BIT_FIELDS_LTOH)
198 typedef struct format3 {
199         uint32_t rs2:5;
200         uint32_t asi:8;
201         uint32_t i:1;
202         uint32_t rs1:5;
203         uint32_t op3:6;
204         uint32_t rd:5;
205         uint32_t op:2;
206 } format3_t;
207 #else
208 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
209 #endif

211 #if defined(_BIT_FIELDS_HTOL)
212 typedef struct format3a {
213         uint32_t op:2;
214         uint32_t rd:5;
215         uint32_t op3:6;
216         uint32_t rs1:5;
217         uint32_t i:1;
218         uint32_t simm13:13;
219 } format3a_t;
220 #elif defined(_BIT_FIELDS_LTOH)
221 typedef struct format3a {
222         uint32_t simm13:13;
223         uint32_t i:1;
224         uint32_t rs1:5;
225         uint32_t op3:6;
226         uint32_t rd:5;
227         uint32_t op:2;
228 } format3a_t;
229 #else
230 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
231 #endif

233 #if defined(_BIT_FIELDS_HTOL)
234 typedef struct format3b {
235         uint32_t op:2;
236         uint32_t rd:5;
237         uint32_t op3:6;
238         uint32_t rs1:5;
239         uint32_t i:1;
240         uint32_t x:1;
241         uint32_t undef:6;
242         uint32_t shcnt:6;
243 } format3b_t;
244 #elif defined(_BIT_FIELDS_LTOH)
245 typedef struct format3b {
246         uint32_t shcnt:6;
247         uint32_t undef:6;
248         uint32_t x:1;
249         uint32_t i:1;
250         uint32_t rs1:5;
251         uint32_t op3:6;
252         uint32_t rd:5;
253         uint32_t op:2;
254 } format3b_t;
255 #else
256 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
257 #endif

259 #if defined(_BIT_FIELDS_HTOL)
```

```
260 typedef struct format3c {
261         uint32_t op:2;
262         uint32_t rd:5;
263         uint32_t op3:6;
264         uint32_t cc2:1;
265         uint32_t cond:4;
266         uint32_t i:1;
267         uint32_t cc:2;
268         uint32_t simm11:11;
269 } format3c_t;
270 #elif defined(_BIT_FIELDS_LTOH)
271 typedef struct format3c {
272         uint32_t simm11:11;
273         uint32_t cc:2;
274         uint32_t i:1;
275         uint32_t cond:4;
276         uint32_t cc2:1;
277         uint32_t op3:6;
278         uint32_t rd:5;
279         uint32_t op:2;
280 } format3c_t;
281 #else
282 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
283 #endif

285 #if defined(_BIT_FIELDS_HTOL)
286 typedef struct format3d {
287         uint32_t op:2;
288         uint32_t rd:5;
289         uint32_t op3:6;
290         uint32_t rs1:5;
291         uint32_t i:1;
292         uint32_t rcond:3;
293         uint32_t simm10:10;
294 } format3d_t;
295 #elif defined(_BIT_FIELDS_LTOH)
296 typedef struct format3d {
297         uint32_t simm10:10;
298         uint32_t rcond:3;
299         uint32_t i:1;
300         uint32_t rs1:5;
301         uint32_t op3:6;
302         uint32_t rd:5;
303         uint32_t op:2;
304 } format3d_t;
305 #else
306 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
307 #endif

309 #if defined(_BIT_FIELDS_HTOL)
310 typedef struct formatcp {
311         uint32_t op:2;
312         uint32_t rd:5;
313         uint32_t op3:6;
314         uint32_t rs1:5;
315         uint32_t opc:9;
316         uint32_t rs2:5;
317 } formatcp_t;
318 #elif defined(_BIT_FIELDS_LTOH)
319 typedef struct formatcp {
320         uint32_t rs2:5;
321         uint32_t opc:9;
322         uint32_t rs1:5;
323         uint32_t op3:6;
324         uint32_t rd:5;
325         uint32_t op:2;
```



```
326 } formatcp_t;
327 #else
328 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
329 #endif

331 #if defined(_BIT_FIELDS_HTOL)
332 typedef struct formattcc {
333         uint32_t op:2;
334         uint32_t undef:1;
335         uint32_t cond:4;
336         uint32_t op3:6;
337         uint32_t rs1:5;
338         uint32_t i:1;
339         uint32_t cc:2;
340         uint32_t undef2:3;
341         uint32_t immtrap:8;
342 } formattcc_t;
343 #elif defined(_BIT_FIELDS_LTOH)
344 typedef struct formattcc {
345         uint32_t immtrap:8;
346         uint32_t undef2:3;
347         uint32_t cc:2;
348         uint32_t i:1;
349         uint32_t rs1:5;
350         uint32_t op3:6;
351         uint32_t cond:4;
352         uint32_t undef:1;
353         uint32_t op:2;
354 } formattcc_t;
355 #else
356 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
357 #endif

359 #if defined(_BIT_FIELDS_HTOL)
360 typedef struct formattcc2 {
361         uint32_t op:2;
362         uint32_t undef:1;
363         uint32_t cond:4;
364         uint32_t op3:6;
365         uint32_t rs1:5;
366         uint32_t i:1;
367         uint32_t cc:2;
368         uint32_t undef2:6;
369         uint32_t rs2:5;
370 } formattcc2_t;
371 #elif defined(_BIT_FIELDS_LTOH)
372 typedef struct formattcc2 {
373         uint32_t rs2:5;
374         uint32_t undef2:6;
375         uint32_t cc:2;
376         uint32_t i:1;
377         uint32_t rs1:5;
378         uint32_t op3:6;
379         uint32_t cond:4;
380         uint32_t undef:1;
381         uint32_t op:2;
382 } formattcc2_t;
383 #else
384 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
385 #endif

387 #if defined(_BIT_FIELDS_HTOL)
388 typedef struct formatmbr {
389         uint32_t op:2;
390         uint32_t rd:5;
391         uint32_t op3:6;
```

```
392         uint32_t rs1:5;
393         uint32_t i:1;
394         uint32_t undef:6;
395         uint32_t cmask:3;
396         uint32_t mmask:4;
397 } formatmbr_t;
398 #elif defined(_BIT_FIELDS_LTOH)
399 typedef struct formatmbr {
400         uint32_t mmask:4;
401         uint32_t cmask:3;
402         uint32_t undef:6;
403         uint32_t i:1;
404         uint32_t rs1:5;
405         uint32_t op3:6;
406         uint32_t rd:5;
407         uint32_t op:2;
408 } formatmbr_t;
409 #else
410 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
411 #endif

413 #if defined(_BIT_FIELDS_HTOL)
414 typedef struct formatfcmp {
415         uint32_t op:2;
416         uint32_t undef:3;
417         uint32_t cc:2;
418         uint32_t op3:6;
419         uint32_t rs1:5;
420         uint32_t opf:9;
421         uint32_t rs2:5;
422 } formatfcmp_t;
423 #elif defined(_BIT_FIELDS_LTOH)
424 typedef struct formatfcmp {
425         uint32_t rs2:5;
426         uint32_t opf:9;
427         uint32_t rs1:5;
428         uint32_t op3:6;
429         uint32_t cc:2;
430         uint32_t undef:3;
431         uint32_t op:2;
432 } formatfcmp_t;
433 #else
434 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
435 #endif

437 #if defined(_BIT_FIELDS_HTOL)
438 typedef struct formatfmov {
439         uint32_t op:2;
440         uint32_t rd:5;
441         uint32_t op3:6;
442         uint32_t undef:1;
443         uint32_t cond:4;
444         uint32_t cc:3;
445         uint32_t opf:6;
446         uint32_t rs2:5;
447 } formatfmov_t;
448 #elif defined(_BIT_FIELDS_LTOH)
449 typedef struct formatfmov {
450         uint32_t rs2:5;
451         uint32_t opf:6;
452         uint32_t cc:3;
453         uint32_t cond:4;
454         uint32_t undef:1;
455         uint32_t op3:6;
456         uint32_t rd:5;
457         uint32_t op:2;
```



```
458 } formatfmov_t;
459 #else
460 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
461 #endif

463 #if defined(_BIT_FIELDS_HTOL)
464 typedef struct formatfused {
465         uint32_t op:2;
466         uint32_t rd:5;
467         uint32_t op3:6;
468         uint32_t rs1:5;
469         uint32_t rs3:5;
470         uint32_t op5:4;
471         uint32_t rs2:5;
472 } formatfused_t;
473 #elif defined(_BIT_FIELDS_LTOH)
474 typedef struct formatfused {
475         uint32_t rs2:5;
476         uint32_t op5:4;
477         uint32_t rs3:5;
478         uint32_t rs1:5;
479         uint32_t op3:6;
480         uint32_t rd:5;
481         uint32_t op:2;
482 } formatfused_t;
483 #else
484 #error One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
485 #endif

487 typedef union ifmt {
488         uint32_t        i;
489         format1_t       f1;
490         format2_t       f2;
491         format2a_t      f2a;
492         format2b_t      f2b;
493         format2c_t      f2c;
494         format3_t       f3;
495         format3a_t      f3a;
496         format3b_t      f3b;
497         format3c_t      f3c;
498         format3d_t      f3d;
499         formatcp_t      fcp;
500         formattcc_t     ftcc;
501         formattcc2_t    ftcc2;
502         formatfcmp_t    fcmp;
503         formatmbr_t     fmb;
504         formatfmov_t    fmv;
505         formatfused_t   fused;
506 } ifmt_t;

508 /* integer register names */
509 static const char *reg_names[32] = {
510         "%g0", "%g1", "%g2", "%g3", "%g4", "%g5", "%g6", "%g7",
511         "%o0", "%o1", "%o2", "%o3", "%o4", "%o5", "%sp", "%o7",
512         "%l0", "%l1", "%l2", "%l3", "%l4", "%l5", "%l6", "%l7",
513         "%i0", "%i1", "%i2", "%i3", "%i4", "%i5", "%fp", "%i7"
514 };

516 /* floating point register names */
517 static const char *freg_names[32] = {
518         "%f0",  "%f1",  "%f2",  "%f3",  "%f4",  "%f5",  "%f6",  "%f7",
519         "%f8",  "%f9",  "%f10", "%f11", "%f12", "%f13", "%f14", "%f15",
520         "%f16", "%f17", "%f18", "%f19", "%f20", "%f21", "%f22", "%f23",
521         "%f24", "%f25", "%f26", "%f27", "%f28", "%f29", "%f30", "%f31"
522 };
```

```
524 /* double precision register names */
525 static const char *fdreg_names[32] = {
526         "%d0",  "%d32", "%d2",  "%d34", "%d4",  "%d36", "%d6",  "%d38",
527         "%d8",  "%d40", "%d10", "%d42", "%d12", "%d44", "%d14", "%d46",
528         "%d16", "%d48", "%d18", "%d50", "%d20", "%d52", "%d22", "%d54",
529         "%d24", "%d56", "%d26", "%d58", "%d28", "%d60", "%d30", "%d62"
530 };

532 static const char *compat_fdreg_names[32] = {
533         "%f0",  "%f32", "%f2",  "%f34", "%f4",  "%f36", "%f6",  "%f38",
534         "%f8",  "%f40", "%f10", "%f42", "%f12", "%f44", "%f14", "%f46",
535         "%f16", "%f48", "%f18", "%f50", "%f20", "%f52", "%f22", "%f54",
536         "%f24", "%f56", "%f26", "%f58", "%f28", "%f60", "%f30", "%f62"
537 };


540 static const char *fqreg_names[32] = {
541         "%q0",  "%q32", "%f2",  "%f3",  "%f4",  "%q4",  "%q36", "%f6",
542         "%f7",  "%q8",  "%q40", "%f10", "%f11", "%q12", "%q44", "%f14",
543         "%f15", "%q16", "%q48", "%f18", "%f19", "%q20", "%q52", "%f22",
544         "%f23", "%q24", "%q56", "%f26", "%f27", "%q28", "%q60", "%f30",
545 };


548 /* coprocessor register names -- sparcv8 only */
549 static const char *cpreg_names[32] = {
550         "%c0",  "%c1",  "%c2",  "%c3",  "%c4",  "%c5",  "%c6",  "%c7",
551         "%c8",  "%c9",  "%c10", "%c11", "%c12", "%c13", "%c14", "%c15",
552         "%c16", "%c17", "%c18", "%c19", "%c20", "%c21", "%c22", "%c23",
553         "%c24", "%c25", "%c26", "%c27", "%c28", "%c29", "%c30", "%c31",
554 };

556 /* floating point condition code names */
557 static const char *fcc_names[4] = {
558         "%fcc0", "%fcc1", "%fcc2", "%fcc3"
559 };

561 /* condition code names */
562 static const char *icc_names[4] = {
563         "%icc", NULL, "%xcc", NULL
564 };

566 /* bitmask values for membar */
567 static const char *membar_mmask[4] = {
568         "#LoadLoad", "#StoreLoad", "#LoadStore", "#StoreStore"
569 };

571 static const char *membar_cmask[3] = {
572         "#Lookaside", "#MemIssue", "#Sync"
573 };

575 /* v8 ancillary state register names */
576 static const char *asr_names[32] = {
577         "%y",    "%asr1",  "%asr2",  "%asr3",
578         "%asr4",  "%asr5",  "%asr6",  "%asr7",
579         "%asr8",  "%asr9",  "%asr10", "%asr11",
580         "%asr12", "%asr13", "%asr14", "%asr15",
581         NULL,   NULL,   NULL,   NULL,
582         NULL,   NULL,   NULL,   NULL,
583         NULL,   NULL,   NULL,   NULL,
584         NULL,   NULL,   NULL,   NULL
585 };
586 static const uint32_t asr_rdmask = 0x0000ffffL;
587 static const uint32_t asr_wrmask = 0x0000ffffL;

589 static const char *v9_asr_names[32] = {
```



```
590         "%y",           NULL,           "%ccr", "%asi",
591         "%tick",        "%pc",          "%fprs",        NULL,
592         NULL,           NULL,           NULL,   NULL,
593         NULL,           NULL,           NULL,   NULL,
594         "%pcr",         "%pic",         "%dcr", "%gsr",
595         "%softint_set", "%softint_clr", "%softint",     "%tick_cmpr",
596         "%stick",       "%stick_cmpr",  NULL,   NULL,
597         NULL,           NULL,           NULL,   NULL
598 };
599 /*
600  * on v9, only certain registers are valid for read or writing
601  * these are bitmasks corresponding to which registers are valid in which
602  * case. Any access to %dcr is illegal.
603  */
604 static const uint32_t v9_asr_rdmask = 0x03cb007d;
605 static const uint32_t v9_asr_wrmask = 0x03fb004d;

607 /* privledged register names on v9 */
608 /* TODO: compat - NULL to %priv_nn */
609 static const char *v9_privreg_names[32] = {
610         "%tpc",  "%tnpc",        "%tstate",  "%tt",
611         "%tick",         "%tba",  "%pstate",  "%tl",
612         "%pil",  "%cwp",          "%cansave", "%canrestore",
613         "%cleanwin", "%otherwin", "%wstate",  "%fq",
614         "%gl",  NULL,   NULL,   NULL,
615         NULL,   NULL,   NULL,   NULL,
616         NULL,   NULL,   NULL,   NULL,
617         NULL,   NULL,   NULL,   "%ver"
618 };

620 /* hyper privileged register names on v9 */
621 static const char *v9_hprivreg_names[32] = {
622         "%hpstate",      "%htstate",     NULL,  "%hintp",
623         NULL,   "%htba",         "%hver",  NULL,
624         NULL,   NULL,   NULL,   NULL,
625         NULL,   NULL,   NULL,   NULL,
626         NULL,   NULL,   NULL,   NULL,
627         NULL,   NULL,   NULL,   NULL,
628         NULL,   NULL,   NULL,   NULL,
629         NULL,   NULL,   NULL,   "%hstick_cmpr"
630 };

632 static const uint32_t v9_pr_rdmask = 0x80017fff;
633 static const uint32_t v9_pr_wrmask = 0x00017fff;
634 static const uint32_t v9_hpr_rdmask = 0x8000006b;
635 static const uint32_t v9_hpr_wrmask = 0x8000006b;

637 static const char *prefetch_str[32] = {
638         "#n_reads", "#one_read",
639         "#n_writes", "#one_write",
640         "#page",    NULL, NULL, NULL,
641         NULL, NULL, NULL, NULL,
642         NULL, NULL, NULL, NULL,
643         NULL, "#unified", NULL, NULL,
644         "#n_reads_strong", "#one_read_strong",
645         "#n_writes_strong", "#one_write_strong",
646         NULL, NULL, NULL, NULL,
647         NULL, NULL, NULL, NULL
648 };

650 static void prt_field(const char *, uint32_t, int);

652 static const char *get_regname(dis_handle_t *, int, uint32_t);
653 static int32_t sign_extend(int32_t, int32_t);

655 static void prt_name(dis_handle_t *, const char *, int);
```

```
#define IMM_SIGNED 0x01  /* Is immediate value signed           */
#define IMM_ADDR   0x02  /* Is immediate value part of an address */
static void prt_imm(dis_handle_t *, uint32_t, int);

static void prt_asi(dis_handle_t *, uint32_t);
static const char *get_asi_name(uint8_t);
static void prt_address(dis_handle_t *, uint32_t, int);
static void prt_aluargs(dis_handle_t *, uint32_t, uint32_t);
static void bprintf(dis_handle_t *, const char *, ...);

/*
 * print out val (which is 'bitlen' bits long) in binary
 */
#if defined(DIS_STANDALONE)
/* ARGSUSED */
void
prt_binary(uint32_t val, int bitlen)
{

}

#else

void
prt_binary(uint32_t val, int bitlen)
{
        int i;

        for (i = bitlen - 1; i >= 0; --i) {
                (void) fprintf(stderr, ((val & (1L << i)) != 0) ? "1" : "0");

                if (i % 4 == 0 && i != 0)
                        (void) fprintf(stderr, " ");
        }
}
#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);

        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,
            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);
}

int
fmt_sethi(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;

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

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

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

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

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

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

        return (0);
}

/* ARGSUSED3 */
int
fmt_branch(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
{
        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
#endif /* ! codereview */
```

```
782         const char *name = inp->in_data.in_def.in_name;
783         const char *r = NULL;
784         const char *annul = "";
785         const char *pred  = "";

787         char buf[15];

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

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

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

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

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

815                 case DISP16:
816                         prt_field("bit 28", ((instr & (1L << 28)) >> 28), 1);
817                         prt_field("rcond", f->f2c.cond, 3);
818                         prt_field("p", f->f2c.p, 1);
819                         prt_field("rs1", f->f2c.rs1, 5);
820                         prt_field("d16hi", f->f2c.d16hi, 2);
821                         prt_field("d16lo", f->f2c.d16lo, 14);
822                         break;
823                 }
824         }

826         if (f->f2b.op2 == 0x01 && idx == 0x00 && f->f2b.p == 1 &&
827             f->f2b.cc == 0x02 && ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) != 0)) {
125             f->f2b.cc == 0x02 && ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) != 0)) {
828                 name = "iprefetch";
829                 flags = FLG_RS1(REG_NONE)|FLG_DISP(DISP19);
830         }


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

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

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



```
846         }

848         disp *= 4;

850         if ((FLG_RS1_VAL(flags) == REG_ICC) || (FLG_RS1_VAL(flags) == REG_FCC))
851                 r = get_regname(dhp, FLG_RS1_VAL(flags), f->f2b.cc);
852         else
853                 r = get_regname(dhp, FLG_RS1_VAL(flags), f->f2c.rs1);

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

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

861         if ((flags & FLG_PRED) != 0) {
862                 if (f->f2b.p == 0) {
863                         pred = ",pn";
864                 } else {
865                         if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0)
163                         if ((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0)
866                                 pred = ",pt";
867                 }
868         }

870         (void) snprintf(buf, sizeof (buf), "%s%s%s", name, annul, pred);
871         prt_name(dhp, buf, 1);


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

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

890         case DISP16:
891                 bprintf(dhp,
892                     (octal != 0) ? "%s, %s0%-6lo <" : "%s, %s0x%-5lx <",
893                     r,
894                     (disp < 0) ? "-" : "+",
895                     (disp < 0) ? (-disp) : disp);
896                 break;
897         }

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

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

905         return (0);
906 }
```

```
 910 /*
 911  * print out the compare and swap instructions (casa/casxa)
 912  * format: casa/casxa [%rs1] imm_asi, %rs2, %rd
 913  *          casa/casxa [%rs1] %asi, %rs2, %rd
 914  *
 915  * If DIS_DEBUG_SYN_ALL is set, synthetic instructions are emitted
 916  * when an immediate ASI value is given as follows:
 917  *
 918  * casa  [%rs1]#ASI_P, %rs2, %rd    -> cas   [%rs1], %rs2, %rd
 919  * casa  [%rs1]#ASI_P_L, %rs2, %rd  -> casl  [%rs1], %rs2, %rd
 920  * casxa [%rs1]#ASI_P, %rs2, %rd    -> casx  [%rs1], %rs2, %rd
 921  * casxa [%rs1]#ASI_P_L, %rs2, %rd  -> casxl [%rs1], %rs2, %rd
 922  */
 923 static int
 924 fmt_cas(dis_handle_t *dhp, uint32_t instr, const char *name)
 925 {
 926         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
 927 #endif /* ! codereview */
 928         ifmt_t *f = (ifmt_t *)&instr;
 929         const char *asistr = NULL;
 930         int noasi = 0;

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

 934         if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT)) != 0) {
 224         if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT)) != 0) {
 935                 if (f->f3.op3 == 0x3c && f->f3.i == 0) {
 936                         if (f->f3.asi == 0x80) {
 937                                 noasi = 1;
 938                                 name = "cas";
 939                         }

 941                         if (f->f3.asi == 0x88) {
 942                                 noasi = 1;
 943                                 name = "casl";
 944                         }
 945                 }

 947                 if (f->f3.op3 == 0x3e && f->f3.i == 0) {
 948                         if (f->f3.asi == 0x80) {
 949                                 noasi = 1;
 950                                 name = "casx";
 951                         }

 953                         if (f->f3.asi == 0x88) {
 954                                 noasi = 1;
 955                                 name = "casxl";
 956                         }
 957                 }
 958         }

 960         prt_name(dhp, name, 1);

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

 964         if (noasi == 0) {
 965                 (void) strlcat(dhx->dhx_buf, " ", dhx->dhx_buflen);
 255                 (void) strlcat(dhp->dh_buf, " ", dhp->dh_buflen);
 966                 prt_asi(dhp, instr);
 967         }

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

 971         if (noasi == 0 && asistr != NULL)
```



```
 972                 bprintf(dhp, "\t<%s>", asistr);

 974         return (0);
 975 }

 977 /*
 978  * format a load/store instruction
 979  * format: ldXX [%rs1 + %rs2], %rd        load, i==0
 980  *          ldXX [%rs1 +/- nn], %rd       load, i==1
 981  *          ldXX [%rs1 + %rs2] #XX, %rd   load w/ imm_asi, i==0
 982  *          ldXX [%rs1 +/- nn] %asi, %rd  load from asi[%asi], i==1
 983  *
 984  *          stXX %rd, [%rs1 + %rs2]       store, i==0
 985  *          stXX %rd, [%rs1 +/- nn]       store, i==1
 986  *          stXX %rd, [%rs1 + %rs1] #XX   store to imm_asi, i==0
 987  *          stXX %rd, [%rs1 +/-nn] %asi   store to asi[%asi], i==1
 988  *
 989  * The register sets used for %rd are set in the instructions flags field
 990  * The asi variants are used if FLG_ASI is set in the instructions flags field
 991  *
 992  * If DIS_DEBUG_SYNTH_ALL or DIS_DEBUG_COMPAT are set,
 993  * When %rs1, %rs2 or nn are 0, they are not printed, i.e.
 994  * [ %rs1 + 0x0 ], %rd -> [%rs1], %rd for example
 995  *
 996  * The following synthetic instructions are also implemented:
 997  *
 998  * stb %g0, [addr] -> clrb [addr]    DIS_DEBUG_SYNTH_ALL
 999  * sth %g0, [addr] -> crlh [addr]    DIS_DEBUG_SYNTH_ALL
1000  * stw %g0, [addr] -> clr  [addr]    DIS_DEBUG_SYNTH_ALL|DIS_DEBUG_COMPAT
1001  * stx %g0, [addr] -> clrx [addr]    DIS_DEBUG_SYNTH_ALL
1002  *
1003  * If DIS_DEBUG_COMPAT is set, the following substitutions also take place
1004  *      lduw -> ld
1005  *      ldtw -> ld
1006  *      stuw -> st
1007  *      sttw -> st
1008  */
1009 int
1010 fmt_ls(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1011 {
1012         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1013 #endif /* ! codereview */
1014         ifmt_t *f = (ifmt_t *)&instr;
1015         const char *regstr = NULL;
1016         const char *asistr = NULL;

1018         const char *iname = inp->in_data.in_def.in_name;
1019         uint32_t flags = inp->in_data.in_def.in_flags;

1021         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
 302         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
1022                 prt_field("op", f->f3.op, 2);
1023                 prt_field("op3", f->f3.op3, 6);
1024                 prt_field("rs1", f->f3.rs1, 5);
1025                 prt_field("i", f->f3.i, 1);
1026                 if (f->f3.i != 0) {
1027                         prt_field("simm13", f->f3a.simm13, 13);
1028                 } else {
1029                         if ((flags & FLG_ASI) != 0)
1030                                 prt_field("imm_asi", f->f3.asi, 8);
1031                         prt_field("rs2", f->f3.rs2, 5);
1032                 }
1033                 prt_field("rd", f->f3.rd, 5);
1034         }

1036         if (idx == 0x2d || idx == 0x3d) {
```

```
1037                /* prefetch / prefetcha */
1039                prt_name(dhp, iname, 1);
1041                prt_address(dhp, instr, 0);
1043                if (idx == 0x3d) {
1044                        (void) strlcat(dhx->dhx_buf, " ", dhx->dhx_buflen);
 325                        (void) strlcat(dhp->dh_buf, " ", dhp->dh_buflen);
1045                        prt_asi(dhp, instr);
1046                }
1048                (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
 329                (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
1050                /* fcn field is the same as rd */
1051                if (prefetch_str[f->f3.rd] != NULL)
1052                        (void) strlcat(dhx->dhx_buf, prefetch_str[f->f3.rd],
1053                            dhx->dhx_buflen);
 333                        (void) strlcat(dhp->dh_buf, prefetch_str[f->f3.rd],
 334                            dhp->dh_buflen);
1054                else
1055                        prt_imm(dhp, f->f3.rd, 0);
1057                if (idx == 0x3d && f->f3.i == 0) {
1058                        asistr = get_asi_name(f->f3.asi);
1059                        if (asistr != NULL)
1060                                bprintf(dhp, "\t<%s>", asistr);
1061                }
1063                return (0);
1064        }
1066        /* casa / casxa */
1067        if (idx == 0x3c || idx == 0x3e)
1068                return (fmt_cas(dhp, instr, iname));
1070        /* synthetic instructions & special cases */
1071        switch (idx) {
1072        case 0x00:
1073                /* ld */
1074                if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
 355                if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
1075                        iname = "lduw";
1076                break;
1078        case 0x03:
1079                if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
 360                if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
1080                        iname = "ldtw";
1081                break;
1083        case 0x04:
1084                /* stw */
1085                if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
 366                if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
1086                        iname = "stuw";
1088                if ((dhp->dh_flags & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1089                    == 0)
1090                        break;
1092                if (f->f3.rd == 0) {
1093                        iname = "clr";
1094                        flags = FLG_RD(REG_NONE);
1095                }
```



```
1096                break;
1098        case 0x05:
1099                /* stb */
1100                if ((dhp->dh_flags & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1101                    == 0)
1102                        break;
1104                if (f->f3.rd == 0) {
1105                        iname = "clrb";
1106                        flags = FLG_RD(REG_NONE);
1107                }
1108                break;
1110        case 0x06:
1111                /* sth */
1112                if ((dhp->dh_flags & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1113                    == 0)
1114                        break;
1116                if (f->f3.rd == 0) {
1117                        iname = "clrh";
1118                        flags = FLG_RD(REG_NONE);
1119                }
1120                break;
1122        case 0x07:
1123                if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
 404                if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
1124                        iname = "sttw";
1125                break;
1127        case 0x0e:
1128                /* stx */
1130                if ((dhp->dh_flags & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1131                    == 0)
1132                        break;
1134                if (f->f3.rd == 0) {
1135                        iname = "clrx";
1136                        flags = FLG_RD(REG_NONE);
1137                }
1138                break;
1140        case 0x13:
1141                /* ldtwa */
1142                if (((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0) &&
 423                if (((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0) &&
1143                    ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0))
1144                        iname = "ldtwa";
1145                break;
1147        case 0x17:
1148                /* sttwa */
1149                if (((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0) &&
 430                if (((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0) &&
1150                    ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0))
1151                        iname = "sttwa";
1152                break;
1154        case 0x21:
1155        case 0x25:
1156                /*
1157                 * on sparcv8 it merely says that rd != 1 should generate an
1158                 * exception, on v9, it is illegal
```

```
1159                 */
1160                if ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) == 0)
1161                        break;

1163                iname = (idx == 0x21) ? "ldx" : "stx";

1165                if (f->f3.rd > 1)
1166                        return (-1);

1168                break;

1170        case 0x31:
1171                /* stda */
1172                switch (f->f3.asi) {
1173                        case 0xc0:
1174                        case 0xc1:
1175                        case 0xc8:
1176                        case 0xc9:
1177                        case 0xc2:
1178                        case 0xc3:
1179                        case 0xca:
1180                        case 0xcb:
1181                        case 0xc4:
1182                        case 0xc5:
1183                        case 0xcc:
1184                        case 0xcd:
1185                                /*
1186                                 * store partial floating point, only valid w/
1187                                 * vis
1188                                 *
1189                                 * Somewhat confusingly, it uses the same op
1190                                 * code as 'stda' -- store double to alternate
1191                                 * space.  It is distinguised by specific
1192                                 * imm_asi values (as seen above), and
1193                                 * has a slightly different output syntax
1194                                 */

1196                                if ((dhp->dh_flags & DIS_SPARC_V9_SGI) == 0)
1197                                        break;
1198                                if (f->f3.i != 0)
1199                                        break;
1200                                prt_name(dhp, iname, 1);
1201                                bprintf(dhp, "%s, %s, [%s] ",
1202                                    get_regname(dhp, REG_FPD, f->f3.rd),
1203                                    get_regname(dhp, REG_FPD, f->f3.rs2),
1204                                    get_regname(dhp, REG_FPD, f->f3.rs1));
1205                                prt_asi(dhp, instr);
1206                                asistr = get_asi_name(f->f3.asi);
1207                                if (asistr != NULL)
1208                                        bprintf(dhp, "\t<%s>", asistr);

1210                                return (0);

1212                        default:
1213                                break;
1214                }

1216        }

1218        regstr = get_regname(dhp, FLG_RD_VAL(flags), f->f3.rd);

1220        if (f->f3.i == 0)
1221                asistr = get_asi_name(f->f3.asi);

1223        prt_name(dhp, iname, 1);
```



```
1225        if ((flags & FLG_STORE) != 0) {
1226                if (regstr[0] != '\0') {
1227                        (void) strlcat(dhx->dhx_buf, regstr, dhx->dhx_buflen);
1228                        (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
 508                    (void) strlcat(dhp->dh_buf, regstr, dhp->dh_buflen);
 509                    (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
1229                }

1231                prt_address(dhp, instr, 0);
1232                if ((flags & FLG_ASI) != 0) {
1233                        (void) strlcat(dhx->dhx_buf, " ", dhx->dhx_buflen);
 514                    (void) strlcat(dhp->dh_buf, " ", dhp->dh_buflen);
1234                        prt_asi(dhp, instr);
1235                }
1236        } else {
1237                prt_address(dhp, instr, 0);
1238                if ((flags & FLG_ASI) != 0) {
1239                        (void) strlcat(dhx->dhx_buf, " ", dhx->dhx_buflen);
 520                    (void) strlcat(dhp->dh_buf, " ", dhp->dh_buflen);
1240                        prt_asi(dhp, instr);
1241                }

1243                if (regstr[0] != '\0') {
1244                        (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
1245                        (void) strlcat(dhx->dhx_buf, regstr, dhx->dhx_buflen);
 525                    (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
 526                    (void) strlcat(dhp->dh_buf, regstr, dhp->dh_buflen);
1246                }
1247        }

1249        if ((flags & FLG_ASI) != 0 && asistr != NULL)
1250                bprintf(dhp, "\t<%s>", asistr);

1252        return (0);
1253 }

1255 static int
1256 fmt_cpop(dis_handle_t *dhp, uint32_t instr, const inst_t *inp)
1257 {
1258        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1259 #endif /* ! codereview */
1260        ifmt_t *f = (ifmt_t *)&instr;
1261        int flags = FLG_P1(REG_CP)|FLG_P2(REG_CP)|FLG_NOIMM|FLG_P3(REG_CP);

1263        if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
 539        if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
1264                prt_field("op", f->fcp.op, 2);
1265                prt_field("op3", f->fcp.op3, 6);
1266                prt_field("opc", f->fcp.opc, 9);
1267                prt_field("rs1", f->fcp.rs1, 5);
1268                prt_field("rs2", f->fcp.rs2, 5);
1269                prt_field("rd", f->fcp.rd, 5);
1270        }

1272        prt_name(dhp, inp->in_data.in_def.in_name, 1);
1273        prt_imm(dhp, f->fcp.opc, 0);

1275        (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
 551        (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
1276        (void) prt_aluargs(dhp, instr, flags);

1278        return (0);
1279 }

1281 static int
1282 dis_fmt_rdwr(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
```

```
{
        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
#endif /* ! codereview */
        const char *psr_str = "%psr";
        const char *wim_str = "%wim";
        const char *tbr_str = "%tbr";

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

        ifmt_t *f = (ifmt_t *)&instr;

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

        int use_mask = 1;
        uint32_t mask;

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

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

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

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

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

                        first = 0;

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

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

                        return (0);
                }

                if (v9 != 0) {
```



```
                        regstr = v9_asr_names[ridx];
                        mask = v9_asr_rdmask;
                } else {
                        regstr = asr_names[ridx];
                        mask = asr_rdmask;
                }
                break;

        case 0x29:
                if (v9 != 0) {
                        regstr = v9_hprivreg_names[ridx];
                        mask = v9_hpr_rdmask;
                } else {
                        regstr = psr_str;
                        use_mask = 0;
                }
                break;

        case 0x2a:
                if (v9 != 0) {
                        regstr = v9_privreg_names[ridx];
                        mask = v9_pr_rdmask;
                } else {
                        regstr = wim_str;
                        use_mask = 0;
                }
                break;

        case 0x2b:
                if (v9 != 0) {
                        /* flushw */
                        prt_name(dhp, name, 0);
                        return (0);
                }

                regstr = tbr_str;
                use_mask = 0;
                break;

        case 0x30:
                if (v9 != 0) {
                        regstr = v9_asr_names[ridx];
                        mask = v9_asr_wrmask;
                } else {
                        regstr = asr_names[ridx];
                        mask = asr_wrmask;
                }

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

                /* synth: mov */
                if ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
                 560
                if ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
                    == 0)
                        break;
```

```
1415                if (v9 == 0) {
1416                        if (f->f3.rs1 == 0) {
1417                                name = "mov";
1418                                pr_rs1 = 0;
1419                        }

1421                        if ((f->f3.i == 0 && f->f3.rs2 == 0) ||
1422                            (f->f3.i == 1 && f->f3a.simm13 == 0)) {
1423                                name = "mov";
1424                                pr_rs2 = 0;
1425                        }
1426                }

1428                if (pr_rs1 == 0)
1429                        pr_rs2 = 1;

1431                break;

1433        case 0x31:
1434                /*
1435                 * NOTE: due to the presence of an overlay entry for another
1436                 * table, this case only happens when doing v8 instructions
1437                 * only
1438                 */
1439                regstr = psr_str;
1440                use_mask = 0;
1441                break;

1443        case 0x32:
1444                if (v9 != 0) {
1445                        regstr = v9_privreg_names[ridx];
1446                        mask = v9_pr_wrmask;
1447                } else {
1448                        regstr = wim_str;
1449                        use_mask = 0;
1450                }
1451                break;

1453        case 0x33:
1454                if (v9 != 0) {
1455                        regstr = v9_hprivreg_names[ridx];
1456                        mask = v9_hpr_wrmask;
1457                } else {
1458                        regstr = tbr_str;
1459                        use_mask = 0;
1460                }
1461                break;
1462        }

1464        if (regstr == NULL)
1465                return (-1);

1467        if (use_mask != 0 && ((1L << ridx) & mask) == 0)
1468                return (-1);

1470        prt_name(dhp, name, 1);

1472        if (rd != 0) {
1473                bprintf(dhp, "%s, %s", regstr, reg_names[f->f3.rd]);
1474        } else {
1475                if (pr_rs1 == 1)
1476                        bprintf(dhp, "%s, ", reg_names[f->f3.rs1]);

1478                if (pr_rs2 != 0) {
1479                        if (f->f3.i == 1)
```



```
1480                                prt_imm(dhp, sign_extend(f->f3a.simm13, 13),
1481                                    IMM_SIGNED);
1482                        else
1483                                (void) strlcat(dhx->dhx_buf,
1484                                    reg_names[f->f3.rs2], dhx->dhx_buflen);
1485                        (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
 632                                (void) strlcat(dhp->dh_buf,
 633                                    reg_names[f->f3.rs2], dhp->dh_buflen);
 634                        (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
1486                }

1488                (void) strlcat(dhx->dhx_buf, regstr, dhx->dhx_buflen);
 637                (void) strlcat(dhp->dh_buf, regstr, dhp->dh_buflen);
1489        }

1491        return (0);
1492 }

1494 /* ARGSUSED3 */
1495 int
1496 fmt_trap(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1497 {
1498        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1499 #endif /* ! codereview */
1500        ifmt_t *f = (ifmt_t *)&instr;

1502        int v9 = ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0);
1503        int p_rs1, p_t;

1505        if (f->ftcc.undef != 0)
1506                return (-1);

1508        if (icc_names[f->ftcc.cc] == NULL)
1509                return (-1);

1511        if (f->ftcc.i == 1 && f->ftcc.undef2 != 0)
1512                return (-1);

1514        if (f->ftcc2.i == 0 && f->ftcc2.undef2 != 0)
1515                return (-1);

1517        p_rs1 = ((f->ftcc.rs1 != 0) ||
1518            ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0));
 647            ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0));

1520        if (f->ftcc.i == 0) {
1521                p_t = (f->f3.rs2 != 0 || p_rs1 == 0);

1523                bprintf(dhp, "%-9s %s%s%s%s%s", inp->in_data.in_def.in_name,
1524                    (v9 != 0) ? icc_names[f->ftcc2.cc] : "",
1525                    (v9 != 0) ? ", " : "",
1526                    (p_rs1 != 0) ? reg_names[f->ftcc2.rs1] : "",
1527                    (p_rs1 != 0) ? " + " : "",
1528                    (p_t != 0) ? reg_names[f->f3.rs2] : "");
1529        } else {
1530                bprintf(dhp, "%-9s %s%s%s%s0x%x", inp->in_data.in_def.in_name,
1531                    (v9 != 0) ? icc_names[f->ftcc2.cc] : "",
1532                    (v9 != 0) ? ", " : "",
1533                    (p_rs1 != 0) ? reg_names[f->ftcc2.rs1] : "",
1534                    (p_rs1 != 0) ? " + " : "",
1535                    f->ftcc.immtrap);
1536        }
1537        return (0);
1538 }
```

_____***unchanged_portion_omitted_***

```
1572 /* ARGSUSED3 */
1573 static int
1574 prt_jmpl(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1575 {
1576         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1577 #endif /* ! codereview */
1578         const char *name = inp->in_data.in_def.in_name;
1579         ifmt_t *f = (ifmt_t *)&instr;

1581         if (f->f3.rd == 15 && ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0))
 705         if (f->f3.rd == 15 && ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0))
1582                 name = "call";

1584         if (f->f3.rd == 0) {
1585                 if (f->f3.i == 1 && f->f3a.simm13 == 8) {
1586                         if (f->f3.rs1 == 15) {
1587                                 prt_name(dhp, "retl", 0);
1588                                 return (0);
1589                         }

1591                         if (f->f3.rs1 == 31) {
1592                                 prt_name(dhp, "ret", 0);
1593                                 return (0);
1594                         }
1595                 }

1597                 name = "jmp";
1598         }

1600         prt_name(dhp, name, 1);
1601         prt_address(dhp, instr, 1);

1603         if (f->f3.rd == 0)
1604                 return (0);

1606         if (f->f3.rd == 15 && ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0))
 730         if (f->f3.rd == 15 && ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0))
1607                 return (0);

1609         bprintf(dhp, ", %s", reg_names[f->f3.rd]);

1611         return (0);
1612 }

1614 int
1615 fmt_alu(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1616 {
1617         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1618 #endif /* ! codereview */
1619         ifmt_t *f = (ifmt_t *)&instr;

1621         const char *name = inp->in_data.in_def.in_name;
1622         int flags = inp->in_data.in_def.in_flags;
1623         int arg = 0;

1625         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
 741         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
1626                 prt_field("op", f->f3.op, 2);
1627                 prt_field("op3", f->f3.op3, 6);
1628                 prt_field("rs1", f->f3.rs1, 5);

1630                 switch (idx) {
1631                         /* TODO: more formats */

1633                 default:
1634                         if (f->f3.i == 0)
```



```
1635                                 prt_field("rs2", f->f3.rs2, 5);
1636                         else
1637                                 prt_field("simm13", f->f3a.simm13, 13);

1639                         prt_field("rd", f->f3.rd, 5);
1640                 }

1642         }

1644         switch (idx) {
1645         case 0x00:
1646                 /* add */

1648                 if ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) == 0)
 764                 if ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) == 0)
1649                         break;

1651                 if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1652                     f->f3a.simm13 == 1) {
1653                         name = "inc";
1654                         flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1655                         break;
1656                 }

1658                 if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1659                     f->f3a.simm13 != 1) {
1660                         name = "inc";
1661                         flags = FLG_P1(REG_NONE);
1662                         break;
1663                 }
1664                 break;

1666         case 0x02:
1667                 /* or */

1669                 if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
 785                 if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1670                     == 0)
1671                         break;

1673                 if ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) != 0) {
 789                 if ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) != 0) {
1674                         if (f->f3.rs1 == f->f3.rd) {
1675                                 name = "bset";
1676                                 flags = FLG_P1(REG_NONE);
1677                                 break;
1678                         }
1679                 }

1681                 if (((f->f3.i == 0 && f->f3.rs2 == 0) ||
1682                     (f->f3.i == 1 && f->f3a.simm13 == 0)) &&
1683                     (f->f3.rs1 == 0)) {
1684                         name = "clr";
1685                         flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1686                         break;
1687                 }

1689                 if (f->f3.rs1 == 0) {
1690                         name = "mov";
1691                         flags = FLG_P1(REG_NONE);
1692                         break;
1693                 }
1694                 break;

1696         case 0x04:
1697                 /* sub */
```

```
1699                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
 815                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1700                    == 0)
1701                        break;

1703                if (f->f3.rs1 == 0 && f->f3.i == 0 && f->f3.rs2 == f->f3.rd) {
1704                        name = "neg";
1705                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE);
1706                        break;
1707                }

1709                if (f->f3.rs1 == 0 && f->f3.i == 0 && f->f3.rs2 != f->f3.rd) {
1710                        name = "neg";
1711                        flags = FLG_P1(REG_NONE);
1712                        break;
1713                }

1715                if ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) == 0)
 831                if ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) == 0)
1716                        break;

1718                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1719                    f->f3a.simm13 == 1) {
1720                        name = "dec";
1721                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1722                        break;
1723                }

1725                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1726                    f->f3a.simm13 != 1) {
1727                        name = "dec";
1728                        flags = FLG_P1(REG_NONE);
1729                        break;
1730                }
1731                break;

1733        case 0x07:
1734                /* xnor */

1736                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
 852                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1737                    == 0)
1738                        break;

1740                /*
1741                 * xnor -> not when you have:
1742                 *       xnor %rs1, 0x0 or %g0, %rd
1743                 */
1744                if ((f->f3.i == 0 && f->f3.rs2 != 0) ||
1745                    (f->f3.i == 1 && f->f3a.simm13 != 0))
1746                        break;

1748                name = "not";

1750                if (f->f3.rs1 == f->f3.rd)
1751                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM|
1752                            FLG_P3(REG_INT);
1753                else
1754                        flags = FLG_P1(REG_INT)|FLG_P2(REG_NONE)|FLG_NOIMM|
1755                            FLG_P3(REG_INT);

1757                break;

1759        case 0x10:
1760                /* addcc */
```



```
1762                if ((dhx->dhx_debug & DIS_DEBUG_SYN_ALL) == 0)
 878                if ((dhp->dh_debug & DIS_DEBUG_SYN_ALL) == 0)
1763                        break;

1765                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1766                    f->f3a.simm13 == 1) {
1767                        name = "inccc";
1768                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1769                        break;
1770                }

1772                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1773                    f->f3a.simm13 != 1) {
1774                        name = "inccc";
1775                        flags = FLG_P1(REG_NONE);
1776                        break;
1777                }
1778                break;

1780        case 0x11:
1781                /* andcc */

1783                if (f->f3.rd != 0)
1784                        break;

1786                if ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
 902                if ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL))
1787                    == 0)
1788                        break;

1790                if (((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0) &&
 906                if (((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0) &&
1791                    ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) == 0))
1792                        break;

1794                name = "btst";
1795                flags = FLG_P1(REG_NONE);
1796                f->f3.rd = f->f3.rs1;
1797                break;

1799        case 0x12:
1800                /* orcc */

1802                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
 918                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1803                    == 0)
1804                        break;

1806                if (f->f3.rs1 == 0 && f->f3.rd == 0 && f->f3.i == 0) {
1807                        name = "tst";
1808                        flags = FLG_P1(REG_NONE)|FLG_P3(REG_NONE);
1809                        break;
1810                }

1812                if (f->f3.rs2 == 0 && f->f3.rd == 0 && f->f3.i == 0) {
1813                        name = "tst";
1814                        flags = FLG_P2(REG_NONE)|FLG_P3(REG_NONE);
1815                        break;
1816                }

1818                break;

1820        case 0x14:
1821                /* subcc */
```

```
1823                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
 939                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1824                    == 0)
1825                        break;

1827                if (f->f3.rd == 0) {
1828                        name = "cmp";
1829                        flags = FLG_P3(REG_NONE);
1830                        break;
1831                }

1833                if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0)
 949                if ((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0)
1834                        break;

1836                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1837                    f->f3a.simm13 == 1) {
1838                        name = "deccc";
1839                        flags = FLG_P1(REG_NONE)|FLG_P2(REG_NONE)|FLG_NOIMM;
1840                        break;
1841                }

1843                if (f->f3.rs1 == f->f3.rd && f->f3.i == 1 &&
1844                    f->f3a.simm13 != 1) {
1845                        name = "deccc";
1846                        flags = FLG_P1(REG_NONE);
1847                        break;
1848                }

1850                break;

1852        case 0x25:
1853        case 0x26:
1854        case 0x27:
1855                return (prt_shift(dhp, instr, inp));

1857        case 0x28:
1858        case 0x29:
1859        case 0x2a:
1860        case 0x2b:
1861        case 0x30:
1862        case 0x31:
1863        case 0x32:
1864        case 0x33:
1865                return (dis_fmt_rdwr(dhp, instr, inp, idx));

1867        case 0x36:
1868        case 0x37:
1869                /* NOTE: overlayed on v9 */
1870                if ((dhp->dh_flags & DIS_SPARC_V8) != 0)
1871                        return (fmt_cpop(dhp, instr, inp));
1872                break;

1874        case 0x38:
1875                /* jmpl */
1876                return (prt_jmpl(dhp, instr, inp, idx));

1878        case 0x39:
1879                /* rett / return */
1880                prt_name(dhp, name, 1);
1881                prt_address(dhp, instr, 1);
1882                return (0);

1884        case 0x3b:
1885                /* flush */
1886                prt_name(dhp, name, 1);
```



```
1887                prt_address(dhp, instr, 0);
1888                return (0);

1890        case 0x3c:
1891        case 0x3d:
1892                /* save / restore */
1893                if ((dhx->dhx_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1009                if ((dhp->dh_debug & (DIS_DEBUG_SYN_ALL|DIS_DEBUG_COMPAT))
1894                    == 0)
1895                        break;

1897                if (f->f3.rs1 != 0 || f->f3.rs2 != 0 || f->f3.rd != 0)
1898                        break;

1900                if (f->f3.i != 0 && ((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0))
1016                if (f->f3.i != 0 && ((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0))
1901                        break;

1903                prt_name(dhp, name, 0);
1904                return (0);
1905        }

1907        if (FLG_P1_VAL(flags) != REG_NONE || FLG_P2_VAL(flags) != REG_NONE ||
1908            FLG_P3_VAL(flags) != REG_NONE)
1909                arg = 1;

1911        prt_name(dhp, name, (arg != 0));
1912        prt_aluargs(dhp, instr, flags);

1914        return (0);
1915 }
______unchanged_portion_omitted_

1940 /* ARGSUSED3 */
1941 int
1942 fmt_movcc(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1943 {
1944        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1945 #endif /* ! codereview */
1946        ifmt_t *f = (ifmt_t *)&instr;
1947        const char **regs = NULL;

1949        if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1060        if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
1950                prt_field("op", f->f3c.op, 2);
1951                prt_field("op3", f->f3c.op3, 6);
1952                prt_field("cond", f->f3c.cond, 4);
1953                prt_field("cc2", f->f3c.cc2, 1);
1954                prt_field("cc", f->f3c.cc, 2);
1955                prt_field("i", f->f3c.i, 1);

1957                if (f->f3c.i == 0)
1958                        prt_field("rs2", f->f3.rs2, 5);
1959                else
1960                        prt_field("simm11", f->f3c.simm11, 11);

1962                prt_field("rd", f->f3.rd, 5);
1963        }

1965        if (f->f3c.cc2 == 0) {
1966                regs = fcc_names;
1967        } else {
1968                regs = icc_names;
1969                if (regs[f->f3c.cc] == NULL)
1970                        return (-1);
1971        }
```

```
1973         prt_name(dhp, inp->in_data.in_def.in_name, 1);

1975         bprintf(dhp, "%s, ", regs[f->f3c.cc]);

1977         if (f->f3c.i == 1)
1978                 prt_imm(dhp, sign_extend(f->f3c.simm11, 11), IMM_SIGNED);
1979         else
1980                 (void) strlcat(dhx->dhx_buf, reg_names[f->f3.rs2],
1981                     dhx->dhx_buflen);
1091                 (void) strlcat(dhp->dh_buf, reg_names[f->f3.rs2],
1092                     dhp->dh_buflen);

1983         bprintf(dhp, ", %s", reg_names[f->f3.rd]);

1985         return (0);
1986 }

1988 /* ARGSUSED3 */
1989 int
1990 fmt_movr(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
1991 {
1992         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
1993 #endif /* ! codereview */
1994         ifmt_t *f = (ifmt_t *)&instr;

1996         prt_name(dhp, inp->in_data.in_def.in_name, 1);

1998         bprintf(dhp, "%s, ", reg_names[f->f3d.rs1]);

2000         if (f->f3d.i == 1)
2001                 prt_imm(dhp, sign_extend(f->f3d.simm10, 10), IMM_SIGNED);
2002         else
2003                 (void) strlcat(dhx->dhx_buf, reg_names[f->f3.rs2],
2004                     dhx->dhx_buflen);
1103                 (void) strlcat(dhp->dh_buf, reg_names[f->f3.rs2],
1104                     dhp->dh_buflen);

2006         bprintf(dhp, ", %s", reg_names[f->f3.rd]);

2008         return (0);
2009 }

2011 /* ARGSUSED3 */
2012 int
2013 fmt_fpop1(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
2014 {
2015         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2016 #endif /* ! codereview */
2017         ifmt_t *f = (ifmt_t *)&instr;
2018         int flags = inp->in_data.in_def.in_flags;

2020         flags |= FLG_NOIMM;

2022         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1115         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
2023                 prt_field("op", f->f3.op, 2);
2024                 prt_field("op3", f->f3.op3, 6);
2025                 prt_field("opf", f->fcmp.opf, 9);
2026                 prt_field("rs1", f->f3.rs1, 5);
2027                 prt_field("rs2", f->f3.rs2, 5);
2028                 prt_field("rd", f->f3.rd, 5);
2029         }

2031         prt_name(dhp, inp->in_data.in_def.in_name, 1);
2032         prt_aluargs(dhp, instr, flags);
```



```
2034         return (0);
2035 }

2037 int
2038 fmt_fpop2(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
2039 {
2040         static const char *condstr_icc[16] = {
2041                 "n", "e",  "le", "l",  "leu", "lu",  "neg", "vs",
2042                 "a", "nz", "g",  "ge", "gu",  "geu", "pos", "vc"
2043         };

2045         static const char *condstr_fcc[16] = {
2046                 "n", "nz", "lg", "ul", "l",   "ug", "g",   "u",
2047                 "a", "e",  "ue", "ge", "uge", "le", "ule", "o"
2048         };

2050         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2051 #endif /* ! codereview */
2052         ifmt_t *f = (ifmt_t *)&instr;
2053         const char *ccstr = "";
2054         char name[15];

2056         int flags = inp->in_data.in_def.in_flags;
2057         int is_cmp = (idx == 0x51 || idx == 0x52 || idx == 0x53 ||
2058             idx == 0x55 || idx == 0x56 || idx == 0x57);
2059         int is_fmov = (idx & 0x3f);
2060         int is_v9 = ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0);
2061         int is_compat = ((dhx->dhx_debug & DIS_DEBUG_COMPAT) != 0);
1143         int is_compat = ((dhp->dh_debug & DIS_DEBUG_COMPAT) != 0);

2063         int p_cc = 0;

2065         is_fmov = (is_fmov == 0x1 || is_fmov == 0x2 || is_fmov == 0x3);

2067         if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
1149         if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
2068                 prt_field("op", f->f3.op, 2);
2069                 prt_field("op3", f->f3.op3, 6);
2070                 prt_field("opf", f->fcmp.opf, 9);

2072                 switch (idx & 0x3f) {
2073                 case 0x51:
2074                 case 0x52:
2075                 case 0x53:
2076                 case 0x55:
2077                 case 0x56:
2078                 case 0x57:
2079                         prt_field("cc", f->fcmp.cc, 2);
2080                         prt_field("rs1", f->f3.rs1, 5);
2081                         prt_field("rs2", f->f3.rs2, 5);
2082                         break;

2084                 case 0x01:
2085                 case 0x02:
2086                 case 0x03:
2087                         prt_field("opf_low", f->fmv.opf, 6);
2088                         prt_field("cond", f->fmv.cond, 4);
2089                         prt_field("opf_cc", f->fmv.cc, 3);
2090                         prt_field("rs2", f->fmv.rs2, 5);
2091                         break;

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

```
2097                }
2098        }

2100        name[0] = '\0';
2101        (void) strlcat(name, inp->in_data.in_def.in_name, sizeof (name));

2103        if (is_fmov != 0) {
2104                (void) strlcat(name,
2105                    (f->fmv.cc < 4) ? condstr_fcc[f->fmv.cond]
2106                    : condstr_icc[f->fmv.cond],
2107                    sizeof (name));
2108        }

2110        prt_name(dhp, name, 1);

2112        if (is_cmp != 0)
2113                ccstr = fcc_names[f->fcmp.cc];

2115        if (is_fmov != 0)
2116                ccstr = (f->fmv.cc < 4) ? fcc_names[f->fmv.cc & 0x3]
2117                    : icc_names[f->fmv.cc & 0x3];

2119        if (ccstr == NULL)
2120                return (-1);

2122        p_cc = (is_compat == 0 || is_v9 != 0 ||
2123            (is_cmp != 0 && f->fcmp.cc != 0) ||
2124            (is_fmov != 0 && f->fmv.cc != 0));

2126        if (p_cc != 0)
2127                bprintf(dhp, "%s, ", ccstr);

2129        prt_aluargs(dhp, instr, flags);

2131        return (0);
2132 }

2134 int
2135 fmt_vis(dis_handle_t *dhp, uint32_t instr, const inst_t *inp, int idx)
2136 {
2137        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2138 #endif /* ! codereview */
2139        ifmt_t *f = (ifmt_t *)&instr;
2140        int flags = inp->in_data.in_def.in_flags;

2142        if ((dhx->dhx_debug & DIS_DEBUG_PRTFMT) != 0) {
2219        if ((dhp->dh_debug & DIS_DEBUG_PRTFMT) != 0) {
2143                prt_field("op", f->f3.op, 2);
2144                prt_field("op3", f->f3.op3, 6);
2145                prt_field("opf", f->fcmp.opf, 9);

2147                if (idx == 0x081) {
2148                        prt_field("mode", instr & 02L, 2);
2149                } else {
2150                        prt_field("rs1", f->f3.rs1, 5);
2151                        prt_field("rs2", f->f3.rs2, 5);
2152                        prt_field("rd", f->f3.rd, 5);
2153                }
2154        }

2156        prt_name(dhp, inp->in_data.in_def.in_name, 1);

2158        if (idx == 0x081) {
2159                /* siam */
2160                bprintf(dhp, "%d", instr & 0x7L);
2161                return (0);
```



```
2162        }

2164        prt_aluargs(dhp, instr, flags);

2166        return (0);
2167 }
______unchanged_portion_omitted_

2273 /*
2274  * return the symbolic name of a register
2275  * regset is one of the REG_* values indicating which type of register it is
2276  * such as integer, floating point, etc.
2277  * idx is the numeric value of the register
2278  *
2279  * If regset is REG_NONE, an empty, but non-NULL string is returned
2280  * NULL may be returned if the index indicates an invalid register value
2281  * such as with the %icc/%xcc sets
2282  */
2283 static const char *
2284 get_regname(dis_handle_t *dhp, int regset, uint32_t idx)
2285 {
2286        dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2287 #endif /* ! codereview */
2288        const char *regname = NULL;

2290        switch (regset) {
2291        case REG_INT:
2292                regname = reg_names[idx];
2293                break;

2295        case REG_FP:
2296                regname = freg_names[idx];
2297                break;

2299        case REG_FPD:
2300                if (((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0) ||
1363                if (((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0) ||
2301                    ((dhp->dh_flags & (DIS_SPARC_V9|DIS_SPARC_V9_SGI)) != 0))
2302                        regname = fdreg_names[idx];
2303                else
2304                        regname = compat_fdreg_names[idx];

2306                break;

2308        case REG_FPQ:
2309                if ((dhx->dhx_debug & DIS_DEBUG_COMPAT) == 0)
1372                if ((dhp->dh_debug & DIS_DEBUG_COMPAT) == 0)
2310                        regname = fqreg_names[idx];
2311                else
2312                        regname = freg_names[idx];

2314                break;

2316        case REG_CP:
2317                regname = cpreg_names[idx];
2318                break;

2320        case REG_ICC:
2321                regname = icc_names[idx];
2322                break;

2324        case REG_FCC:
2325                regname = fcc_names[idx];
2326                break;

2328        case REG_FSR:
```

```
2329                 regname = "%fsr";
2330                 break;

2332         case REG_CSR:
2333                 regname = "%csr";
2334                 break;

2336         case REG_CQ:
2337                 regname = "%cq";
2338                 break;

2340         case REG_NONE:
2341                 regname = "";
2342                 break;
2343         }

2345         return (regname);
2346 }
```

*unchanged_portion_omitted_*

```
2367 /*
2368  * put an address expression into the output buffer
2369  *
2370  * instr is the instruction to use
2371  * if nobrackets != 0, [] are not added around the instruction
2372  *
2373  * Currently this option is set when printing out the address portion
2374  * of a jmpl instruction, but otherwise 0 for load/stores
2375  *
2376  * If no debug flags are set, the full expression is output, even when
2377  * %g0 or 0x0 appears in the address
2378  *
2379  * If DIS_DEBUG_SYN_ALL or DIS_DEBUG_COMPAT are set, when %g0 or 0x0
2380  * appear in the address, they are not output.  If the wierd (and probably
2381  * shouldn't happen) address of [%g0 + %g0] or [%g0 + 0x0] is encountered,
2382  * [%g0] is output
2383  */
2384 static void
2385 prt_address(dis_handle_t *dhp, uint32_t instr, int nobrackets)
2386 {
2387         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2388 #endif /* ! codereview */
2389         ifmt_t *f = (ifmt_t *)&instr;
2390         int32_t simm13;
2391         int octal = ((dhp->dh_flags & DIS_OCTAL) != 0);
2392         int p1 = ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0);
2393         int p2 = ((dhx->dhx_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0);
1450         int p1 = ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0);
1451         int p2 = ((dhp->dh_debug & (DIS_DEBUG_COMPAT|DIS_DEBUG_SYN_ALL)) == 0);

2395         if (f->f3a.i == 0) {
2396                 p1 |= ((f->f3a.rs1 != 0) || f->f3.rs2 == 0);
2397                 p2 |= (f->f3.rs2 != 0);

2399                 bprintf(dhp, "%s%s%s%s%s",
2400                     (nobrackets == 0) ? "[" : "",
2401                     (p1 != 0) ? reg_names[f->f3a.rs1] : "",
2402                     (p1 != 0 && p2 != 0) ? " + " : "",
2403                     (p2 != 0) ? reg_names[f->f3.rs2] : "",
2404                     (nobrackets == 0) ? "]" : "");
2405         } else {
2406                 const char *sign;

2408                 simm13 = sign_extend(f->f3a.simm13, 13);
2409                 sign = (simm13 < 0) ? "-" : "+";
```



```
2411                 p1 |= (f->f3a.rs1 != 0);
2412                 p2 |= (p1 == 0 || simm13 != 0);

2414                 if (p1 == 0 && simm13 == 0)
2415                         p2 = 1;

2417                 if (p1 == 0 && simm13 >= 0)
2418                         sign = "";

2420                 if (p2 != 0)
2421                         bprintf(dhp,
2422                             (octal != 0) ? "%s%s%s%s%s0%lo%s" :
2423                             "%s%s%s%s%s0x%lx%s",
2424                             (nobrackets == 0) ? "[" : "",
2425                             (p1 != 0) ? reg_names[f->f3a.rs1] : "",
2426                             (p1 != 0) ? " " : "",
2427                             sign,
2428                             (p1 != 0) ? " " : "",
2429                             (simm13 < 0) ? -(simm13) : simm13,
2430                             (nobrackets == 0) ? "]" : "");
2431                 else
2432                         bprintf(dhp, "%s%s%s",
2433                             (nobrackets == 0) ? "[" : "",
2434                             reg_names[f->f3a.rs1],
2435                             (nobrackets == 0) ? "]" : "");
2436         }
2437 }

2439 /*
2440  * print out the arguments to an alu operation (add, sub, etc.)
2441  * conatined in 'instr'
2442  *
2443  * alu instructions have the following format:
2444  *       %rs1, %rs2, %rd    (i == 0)
2445  *       %rs1, 0xnnn, %rd   (i == 1)
2446  *         ^    ^       ^
2447  *         |    |       |
2448  *        p1    p2      p3
2449  *
2450  * flags indicates the register set to use for each position (p1, p2, p3)
2451  * as well as if immediate values (i == 1) are allowed
2452  *
2453  * if flags indicates a specific position has REG_NONE set as it's register
2454  * set, it is omitted from the output.  This is primarly used for certain
2455  * floating point operations
2456  */
2457 static void
2458 prt_aluargs(dis_handle_t *dhp, uint32_t instr, uint32_t flags)
2459 {
2460         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2461 #endif /* ! codereview */
2462         ifmt_t *f = (ifmt_t *)&instr;
2463         const char *r1, *r2, *r3;
2464         int p1, p2, p3;
2465         unsigned int opf = 0;

2467         r1 = get_regname(dhp, FLG_P1_VAL(flags), f->f3.rs1);
2468         r2 = get_regname(dhp, FLG_P2_VAL(flags), f->f3.rs2);
2469         r3 = get_regname(dhp, FLG_P3_VAL(flags), f->f3.rd);

2471         p1 = (FLG_P1_VAL(flags) != REG_NONE);
2472         p2 = (((flags & FLG_NOIMM) == 0) || (FLG_P2_VAL(flags) != REG_NONE));
2473         p3 = (FLG_RD_VAL(flags) != REG_NONE);

2475         if (r1 == NULL || r1[0] == '\0')
2476                 p1 = 0;
```

```
2478         if (f->f3a.i == 0 && (r2 == NULL || r2[0] == '\0'))
2479                 p2 = 0;

2481         if (r3 == NULL || r3[0] == '\0')
2482                 p3 = 0;

2484         if ((f->fcmp.op == 2) && (f->fcmp.op3 == 0x36) && (f->fcmp.cc != 0))
2485                 opf = f->fcmp.opf;

2487         if ((opf == 0x151) || (opf == 0x152)) {
2488                 (void) strlcat(dhx->dhx_buf, r3, dhx->dhx_buflen);
2489                 (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
1518                 (void) strlcat(dhp->dh_buf, r3, dhp->dh_buflen);
1519                 (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
2490                 p3 = 0;
2491         }

2493         if (p1 != 0) {
2494                 (void) strlcat(dhx->dhx_buf, r1, dhx->dhx_buflen);
1524                 (void) strlcat(dhp->dh_buf, r1, dhp->dh_buflen);
2495                 if (p2 != 0 || p3 != 0)
2496                         (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
1526                         (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
2497         }

2499         if (p2 != 0) {
2500                 if (f->f3.i == 0 || ((flags & FLG_NOIMM) != 0))
2501                         (void) strlcat(dhx->dhx_buf, r2, dhx->dhx_buflen);
1531                         (void) strlcat(dhp->dh_buf, r2, dhp->dh_buflen);
2502                 else
2503                         prt_imm(dhp, sign_extend(f->f3a.simm13, 13),
2504                             IMM_SIGNED);

2506                 if (p3 != 0)
2507                         (void) strlcat(dhx->dhx_buf, ", ", dhx->dhx_buflen);
1537                         (void) strlcat(dhp->dh_buf, ", ", dhp->dh_buflen);
2508         }

2510         if (p3 != 0)
2511                 (void) strlcat(dhx->dhx_buf, r3, dhx->dhx_buflen);
1541                 (void) strlcat(dhp->dh_buf, r3, dhp->dh_buflen);
2512 }
______unchanged_portion_omitted_

2772 /*
2773  * just a handy function that takes care of managing the buffer length
2774  * w/ printf
2775  */

2777 /*
2778  * PRINTF LIKE 1
2779  */
2780 static void
2781 bprintf(dis_handle_t *dhp, const char *fmt, ...)
2782 {
2783         dis_handle_sparc_t *dhx = dhp->dh_arch_private;
2784 #endif /* ! codereview */
2785         size_t curlen;
2786         va_list ap;

2788         curlen = strlen(dhx->dhx_buf);
1813         curlen = strlen(dhp->dh_buf);

2790         va_start(ap, fmt);
2791         (void) vsnprintf(dhx->dhx_buf + curlen, dhx->dhx_buflen - curlen, fmt,
```



```
1816         (void) vsnprintf(dhp->dh_buf + curlen, dhp->dh_buflen - curlen, fmt,
2792             ap);
2793         va_end(ap);
2794 }
______unchanged_portion_omitted_
```

```
**********************************************************
    3871 Tue Jul 29 20:47:48 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_
```

```
**********************************************************
    4393 Tue Jul 29 20:47:49 2014
new/usr/src/lib/libdisasm/common/libdisasm.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 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
#endif /* ! codereview */
 */

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

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

#include "libdisasm_impl.h"

#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 */
        default:
                return ("unknown error");
        }
}

void
dis_set_data(dis_handle_t *dhp, void *data)
```

```
126 {
127         dhp->dh_data = data;
128 }

130 void
131 dis_flags_set(dis_handle_t *dhp, int f)
132 {
133         dhp->dh_flags |= f;
134 }

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

142 void
143 dis_handle_destroy(dis_handle_t *dhp)
144 {
145         dhp->dh_arch->da_handle_detach(dhp);
146         dis_free(dhp, sizeof (dis_handle_t));
147 }

149 dis_handle_t *
150 dis_handle_create(int flags, void *data, dis_lookup_f lookup_func,
151     dis_read_f read_func)
152 {
153         dis_handle_t *dhp;
154         dis_arch_t *arch = NULL;
155         int i;

157         /* Select an architecture based on flags */
158         for (i = 0; dis_archs[i] != NULL; i++) {
159                 if (dis_archs[i]->da_supports_flags(flags)) {
160                         arch = dis_archs[i];
161                         break;
162                 }
163         }
164         if (arch == NULL) {
165                 (void) dis_seterrno(E_DIS_UNSUPARCH);
166                 return (NULL);
167         }

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

179         /*
180          * Allow the architecture-specific code to allocate
181          * its private data.
182          */
183         if (arch->da_handle_attach(dhp) != 0) {
184                 dis_free(dhp, sizeof (dis_handle_t));
185                 /* dis errno already set */
186                 return (NULL);
187         }

189         return (dhp);
190 }
```



```
192 int
193 dis_disassemble(dis_handle_t *dhp, uint64_t addr, char *buf, size_t buflen)
194 {
195         return (dhp->dh_arch->da_disassemble(dhp, addr, buf, buflen));
196 }

198 uint64_t
199 dis_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
200 {
201         return (dhp->dh_arch->da_previnstr(dhp, pc, n));
202 }

204 int
205 dis_min_instrlen(dis_handle_t *dhp)
206 {
207         return (dhp->dh_arch->da_min_instrlen(dhp));
208 }

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

216 int
217 dis_instrlen(dis_handle_t *dhp, uint64_t pc)
218 {
219         return (dhp->dh_arch->da_instrlen(dhp, pc));
220 }
221 #endif /* ! codereview */
```

```
**********************************************************
    2684 Tue Jul 29 20:47:49 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 */
```

```
**********************************************************
    1820 Tue Jul 29 20:47:49 2014
new/usr/src/lib/libdisasm/common/libdisasm_impl.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 2012 Joshua M. Clulow <josh@sysmgr.org>
#endif /* ! codereview */
 */

#ifndef _LIBDISASM_IMPL_H
#define _LIBDISASM_IMPL_H

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

#ifdef  __cplusplus
extern "C" {
#endif

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

#ifdef  __cplusplus
}
#endif

#endif  /* _LIBDISASM_IMPL_H */
```

```
**********************************************************
    1519 Tue Jul 29 20:47:49 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 Tue Jul 29 20:47:49 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 */
```