```
**********************************************************
   79676 Wed Oct 14 16:30:24 2015
new/usr/src/lib/libdisasm/common/dis_s390x.c
patch fixups
**********************************************************
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
 */

#include <stdio.h>
#include <libdisasm.h>
#include <sys/sysmacros.h>
#include <sys/debug.h>
#endif /* ! codereview */
#include <sys/byteorder.h>

#include "libdisasm_impl.h"

#define ILC2LEN(ilc)    (2 * ((ilc) >= 2 ? (ilc) : (ilc) + 1))

/*
 * Throughout this file, the instruction format names based on:
 *   SA22-7832-09  z/Architecture Principles of Operation
 *
 * System/370, ESA/390, and earlier z/Architecture POP use slightly
 * different names for the formats (the variant names are numeric).  For the
 * sake of simplicity, we use the most detailed definitions - z/Architecture.
 *
 * For ESA/390 we map the formats:
 *   E   -> E
 *   I   -> I
 *   RR  -> RR
 *   RRE -> RRE
 *   RRF -> RRD & RRFa-e
 *   RX  -> RXa-b
 *   RXE -> RXE
 *   RXF -> RXF
 *   RS  -> RSa-b
 *   RSE -> RSYa-b
 *   RSL -> RSLa
 *   RSI -> RSI
 *   RI  -> RIa-c
 *   RIL -> RILa-c
 *   SI  -> SI
 *   S   -> S
 *   SS  -> SSa-b & SSd-e
 *   SSE -> SSE
 *
 * For System/370 we map the formats:
 *   RR -> RR
 *   RX -> RXa-b
 *   RS -> RSa-b
 *   SI -> SI
 *   S  -> S
 *   SS -> SSa-c
```

```
 *
 * Disassembly begins in tbl_xx.  The first byte of the instruction is used
 * as the index.  This yields either an instruction or a sub-table.
 *
 * If an instruction is encountered, its format field is used to format the
 * instruction.
 *
 * There are two types of sub-tables: extended opcode tables (indicated with
 * IF_TBL) or a multiple mnemonics tables (indicated with IF_MULTI).
 *
 * Extended opcode tables indicade which additional bits of the instruction
 * should be inspected.  These bits are used as an index into the sub table.
 *
 * Multiple mnemonic tables are used to print different mnemonics depending
 * Multiple mnemonic tables, are used to print different mnemonics depending
 * on the architecture.  Over the years, certain instructions got a new
 * preferred mnemonic.  For example, 0xa70 is test-under-mask-high (tmh) on
 * System/390.  On z/Architecture systems, the instruction behaves
 * identically (and the assembler hapilly accepts tmh), but the preferred
 * mnemonic is tmlh (test-under-mask-low-high) because z/Architecture
 * extended the general purpose registers from 32 bits to 64 bits.  The
 * current architecture flag (e.g., F_390) is used to index into the
 * sub-table.
 *
 * Regardless of which sub-table is encountered, the selected entry in the
 * sub-table is interpreted using the same rules as the contents of tbl_xx.
 *
 * Finally, we use the extended opcode sub-table mechanism to pretty print
 * the branching instructions.  All branches are conditional based on a
 * 4-bit mask indicating which value of the condition code will result in a
 * taken branch.  In order to produce a more human friendly output, we use
 * the 4-bit mask as an extended opcode to break up the branching
 * instruction into 16 different ones.  For example, instead of printing:
 *
 *    bc   7,0x123(%r1,%r2)
 *
 * we print:
 *
 *    bne  0x123(%r1,%r2)
 *
 * Note that we are using designated initializers via the INSTR/TABLE/MULTI
 * macros and therefore the below tables can be sparse.  We rely on unset
 * entries having zero format fields (aka. IF_INVAL) per C99.
#endif /* ! codereview */
 */

/* BEGIN CSTYLED */
enum ifmt {
        /* invalid */
        IF_INVAL = 0,

        /* indirection */
        IF_TBL,
        IF_MULTI,

        /* 2-byte */
        IF_ZERO,                /* 370, 390, z */
        IF_E,                   /*      390, z */
        IF_I,                   /*      390, z */
        IF_RR,                  /* 370, 390, z */

        /* 4-byte */
        IF_DIAG,                /* 370, 390, z */
        IF_IE,                  /*           z */
        IF_RIa,                 /*      390, z */
        IF_RIb,                 /*      390, z */
        IF_RIc,                 /*      390, z */
        IF_RRD,                 /*      390, z */ /* on 390 these are RRF */
        IF_RRE,                 /*      390, z */
        IF_RRFa,                /*      390, z */
        IF_RRFb,                /*      390, z */
        IF_RRFc,                /*      390, z */
        IF_RRFd,                /*      390, z */
        IF_RRFe,                /*      390, z */
        IF_RSa,                 /* 370, 390, z */
        IF_RSb,                 /* 370, 390, z */
        IF_RSI,                 /*      390, z */
        IF_RXa,                 /* 370, 390, z */
        IF_RXb,                 /* 370, 390, z */
        IF_S,                   /* 370, 390, z */
        IF_SI,                  /* 370, 390, z */

        /* 6-byte */
        IF_MII,                 /*           z */
        IF_RIEa,                /*           z */
        IF_RIEb,                /*           z */
        IF_RIEc,                /*           z */
        IF_RIEd,                /*           z */
        IF_RIEe,                /*           z */
        IF_RIEf,                /*           z */
        IF_RILa,                /*      390, z */
        IF_RILb,                /*      390, z */
        IF_RILc,                /*      390, z */
        IF_RIS,                 /*           z */
        IF_RRS,                 /*           z */
        IF_RSLa,                /*      390, z */
        IF_RSLb,                /*           z */
        IF_RSYa,                /*           z */
        IF_RSYb,                /*           z */
        IF_RXE,                 /*      390, z */
        IF_RXF,                 /*      390, z */
        IF_RXYa,                /*           z */
        IF_RXYb,                /*           z */
        IF_SIL,                 /*           z */
        IF_SIY,                 /*           z */
        IF_SMI,                 /*           z */
        IF_SSa,                 /* 370, 390, z */
        IF_SSb,                 /* 370, 390, z */
        IF_SSc,                 /* 370, 390, z */
        IF_SSd,                 /*      390, z */
        IF_SSe,                 /*      390, z */
        IF_SSf,                 /*      390, z */
        IF_SSE,                 /*      390, z */
        IF_SSF,                 /*           z */
};

#define IF_NFMTS                (IF_SSF + 1)

#define F_370                   0x0001                  /* 370          */
#define F_390                   0x0002                  /*      390     */
#define F_Z                     0x0004                  /*            z */
#define F_SIGNED_IMM            0x0010                  /* 370, 390, z */
#define F_CTL_REG               0x0020                  /* 370, 390, z */
#define F_HIDE_MASK             0x0040                  /* 370, 390, z */
#define F_R1_IS_MASK            0x0080                  /* 370, 390, z */
/* END CSTYLED */

struct inst_table {
        union {
                struct {
                        const char *it_name;
                        unsigned it_flags;
```

```
                } it_inst;
                struct {
                        const struct inst_table *it_ptr;
                        uint8_t it_off:4;
                        uint8_t it_shift:4;
                        uint8_t it_mask;
                } it_table;
                struct {
                        const struct inst_table *it_ptr;
                } it_multi;
        } it_u;
        enum ifmt it_fmt;
                        const char *name;
                        unsigned flags;
                } inst;
                struct {
                        const struct inst_table *ptr;
                        uint8_t off:4;
                        uint8_t shift:4;
                        uint8_t mask;
                } table;
                struct {
                        const struct inst_table *ptr;
                } multi;
        } u;
        enum ifmt fmt;
};
______unchanged_portion_omitted_

#define INSTR(op, m, fm, fl)    [op] = { \
                                        .it_u.it_inst = { \
                                                .it_name = (m), \
                                                .it_flags = (fl), \
                                        .u.inst = { \
                                                .name = (m), \
                                                .flags = (fl), \
                                        }, \
                                        .it_fmt = (fm), \
                                        .fmt = (fm), \
                                }
#define TABLE(op, tbl, o, s, m) [op] = { \
                                        .it_u.it_table = { \
                                                .it_ptr = (tbl), \
                                                .it_off = (o), \
                                                .it_shift = (s), \
                                                .it_mask = (m), \
                                        .u.table = { \
                                                .ptr = (tbl), \
                                                .off = (o), \
                                                .shift = (s), \
                                                .mask = (m), \
                                        }, \
                                        .it_fmt = IF_TBL, \
                                        .fmt = IF_TBL, \
                                }
#define MULTI(op, tbl)          [op] = { \
                                        .it_u.it_multi.it_ptr = (tbl), \
                                        .it_fmt = IF_MULTI, \
                                        .u.multi.ptr = (tbl), \
                                        .fmt = IF_MULTI, \
                                }

/*
 * Instruction tables based on:
 *   GA22-7000-4   System/370 Principles of Operation
 *   SA22-7201-08  ESA/390 Principles of Operation
 *   SA22-7832-09  z/Architecture Principles of Operation
 */

/* BEGIN CSTYLED */
static const struct inst_table tbl_01xx[256] = {
        INSTR(0x01, "pr",    IF_E, F_390 | F_Z),
        INSTR(0x02, "upt",   IF_E, F_390 | F_Z),
        INSTR(0x04, "ptff",  IF_E, F_Z),
        INSTR(0x07, "sckpf", IF_E, F_390 | F_Z),
        INSTR(0x0a, "pfpo",  IF_E, F_Z),
        INSTR(0x0b, "tam",   IF_E, F_390 | F_Z),
        INSTR(0x0c, "sam24", IF_E, F_390 | F_Z),
        INSTR(0x0d, "sam31", IF_E, F_390 | F_Z),
        INSTR(0x0e, "sam64", IF_E, F_Z),
        INSTR(0xff, "trap2", IF_E, F_390 | F_Z),
};
```
______***unchanged_portion_omitted_***

```
/* B and X registers are still registers - print them the same way */
#define B       R
#define X       R

static inline uint32_t
val_8_4_8(uint32_t hi, uint32_t mid, uint32_t lo)
{
        ASSERT0(hi & ~0xff);
        ASSERT0(mid & ~0xf);
        ASSERT0(lo & ~0xff);
#endif /* ! codereview */
        return ((hi << 12) | (mid << 8) | lo);
}

static inline uint32_t
val_16_16(uint32_t hi, uint32_t lo)
{
        ASSERT0(hi & ~0xffff);
        ASSERT0(lo & ~0xffff);
#endif /* ! codereview */
        return ((BE_16(hi) << 16) | BE_16(lo));
}

static inline int32_t
sval_16_16(uint32_t hi, uint32_t lo)
{
        return (val_16_16(hi, lo));
}

static inline uint32_t
val_8_16(uint32_t hi, uint32_t lo)
{
        ASSERT0(hi & ~0xff);
        ASSERT0(lo & ~0xffff);
#endif /* ! codereview */
        return ((hi << 16) | BE_16(lo));
}

static inline int32_t
sval_8_16(uint32_t hi, uint32_t lo)
{
        int32_t tmp = val_8_16(hi, lo);

        /* sign extend */
#endif /* ! codereview */
        if (tmp & 0x00800000)
                return (0xff000000 | tmp);
        return (tmp);
```

```
}

static inline uint32_t
val_4_8(uint32_t hi, uint32_t lo)
{
        ASSERT0(hi & ~0xf);
        ASSERT0(lo & ~0xff);
#endif /* ! codereview */
        return ((hi << 8) | lo);
}

static inline int32_t
sval_4_8(uint32_t hi, uint32_t lo)
{
        uint32_t tmp = val_4_8(hi, lo);

        /* sign extend */
#endif /* ! codereview */
        if (tmp & 0x800)
                return (0xfffff000 | tmp);
        return (tmp);
}

/* ARGSUSED */
static void
fmt_zero(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        (void) snprintf(buf, buflen, "0x00, 0x00");
}

/* ARGSUSED */
static void
fmt_diag(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        (void) snprintf(buf, buflen, "%#x",
            val_8_16(inst->diag.par1, inst->diag.par2));
}

/* ARGSUSED */
static void
fmt_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        /* nothing to do */
}

/* ARGSUSED */
static void
fmt_i(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        (void) snprintf(buf, buflen, "%#x", inst->i.i);
}

/* ARGSUSED */
static void
fmt_ie(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        (void) snprintf(buf, buflen, "%u,%u", inst->ie.i1, inst->ie.i2);
}

/* ARGSUSED */
static void
fmt_mii(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        uint64_t ri2 = addr + 2 * sval_4_8(inst->mii.ri2h, inst->mii.ri2l);
        uint64_t ri3 = addr + 2 * sval_8_16(inst->mii.ri3h, inst->mii.ri3l);
        (void) snprintf(buf, buflen, "%s,%#x,%#x", M[inst->mii.m1], ri2, ri3);
}

/* ARGSUSED */
static void
fmt_ril_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        (void) snprintf(buf, buflen, "%s,%u", R[inst->ril_a.r1],
            val_16_16(inst->ril_a.i2h, inst->ril_a.i2l));
}

/* ARGSUSED */
static void
fmt_ril_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        uint64_t ri2 = addr + 2 *
            sval_16_16(inst->ril_b.ri2h, inst->ril_b.ri2l);

        (void) snprintf(buf, buflen, "%s,%#x", R[inst->ril_b.r1], ri2);
}

/* ARGSUSED */
static void
fmt_ril_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        uint64_t ri2 = addr + 2 *
            sval_16_16(inst->ril_c.ri2h, inst->ril_c.ri2l);

        (void) snprintf(buf, buflen, "%s,%#x", M[inst->ril_c.m1], ri2);
}

/* ARGSUSED */
static void
fmt_ris(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        uint32_t d4 = val_4_8(inst->ris.d4h, inst->ris.d4l);

        (void) snprintf(buf, buflen, "%s,%u,%s,%u(%s)",
            R[inst->ris.r1], inst->ris.i2, M[inst->ris.m3], d4,
            B[inst->ris.b4]);
}

/* ARGSUSED */
static void
fmt_ri_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        uint16_t i2 = BE_16(inst->ri_a.i2);

        if (flags & F_SIGNED_IMM)
                (void) snprintf(buf, buflen, "%s,%d", R[inst->ri_a.r1],
                    (int16_t)i2);
        else
                (void) snprintf(buf, buflen, "%s,%u", R[inst->ri_a.r1],
                    i2);
}

/* ARGSUSED */
static void
fmt_ri_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->ri_b.ri2);

        (void) snprintf(buf, buflen, "%s,%#x", R[inst->ri_b.r1], ri2);
}

static void
```

```
fmt_ri_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->ri_c.ri2);

        if (flags & F_HIDE_MASK)
                (void) snprintf(buf, buflen, "%#x", ri2);
        else
                (void) snprintf(buf, buflen, "%s,%#x", M[inst->ri_c.m1], ri2);
}

/* ARGSUSED */
static void
fmt_rie_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        (void) snprintf(buf, buflen, "%s,%u,%s", R[inst->rie_a.r1],
            BE_16(inst->rie_a.i2), M[inst->rie_a.m3]);
}

/* ARGSUSED */
static void
fmt_rie_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        uint64_t ri4 = addr + 2 * (int16_t)BE_16(inst->rie_b.ri4);

        (void) snprintf(buf, buflen, "%s,%s,%s,%#x", R[inst->rie_b.r1],
            R[inst->rie_b.r2], M[inst->rie_b.m3], ri4);
}

/* ARGSUSED */
static void
fmt_rie_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        uint64_t ri4 = addr + 2 * (int16_t)BE_16(inst->rie_c.ri4);

        (void) snprintf(buf, buflen, "%s,%u,%s,%#x", R[inst->rie_c.r1],
            inst->rie_c.i2, M[inst->rie_c.m3], ri4);
}

/* ARGSUSED */
static void
fmt_rie_d(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        (void) snprintf(buf, buflen, "%s,%s,%u", R[inst->rie_d.r1],
            R[inst->rie_d.r3], BE_16(inst->rie_d.i2));
}

/* ARGSUSED */
static void
fmt_rie_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->rie_e.ri2);

        (void) snprintf(buf, buflen, "%s,%s,%#x", R[inst->rie_e.r1],
            R[inst->rie_e.r3], ri2);
}

/* ARGSUSED */
static void
fmt_rie_f(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
        (void) snprintf(buf, buflen, "%s,%s,%u,%u,%u", R[inst->rie_f.r1],
            R[inst->rie_f.r2], inst->rie_f.i3, inst->rie_f.i4,
            inst->rie_f.i5);
}

/* ARGSUSED */
```

```c
static void
fmt_rre(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	(void) snprintf(buf, buflen, "%s,%s", R[inst->rre.r1], R[inst->rre.r2]);
}

/* ARGSUSED */
static void
fmt_rrf_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	(void) snprintf(buf, buflen, "%s,%s,%s",
	    R[inst->rrf_ab.r1], R[inst->rrf_ab.r2], R[inst->rrf_ab.r3]);
}

/* ARGSUSED */
static void
fmt_rrf_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	(void) snprintf(buf, buflen, "%s,%s,%s",
	    R[inst->rrf_ab.r1], R[inst->rrf_ab.r3], R[inst->rrf_ab.r2]);
}

/* ARGSUSED */
static void
fmt_rrf_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	(void) snprintf(buf, buflen, "%s,%s,%s",
	    R[inst->rrf_cde.r1], R[inst->rrf_cde.r2], M[inst->rrf_cde.m3]);
}

/* ARGSUSED */
static void
fmt_rrf_d(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	(void) snprintf(buf, buflen, "%s,%s,%s",
	    R[inst->rrf_cde.r1], R[inst->rrf_cde.r2], M[inst->rrf_cde.m4]);
}

/* ARGSUSED */
static void
fmt_rrf_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	(void) snprintf(buf, buflen, "%s,%s,%s,%s",
	    R[inst->rrf_cde.r1], M[inst->rrf_cde.m3],
	    R[inst->rrf_cde.r2], M[inst->rrf_cde.m4]);
}

/* ARGSUSED */
static void
fmt_rrs(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	(void) snprintf(buf, buflen, "%s,%s,%s,%u(%s)", R[inst->rrs.r1],
	    R[inst->rrs.r2], M[inst->rrs.m3],
	    val_4_8(inst->rrs.d4h, inst->rrs.d4l), B[inst->rrs.b4]);
}

/* ARGSUSED */
static void
fmt_rr(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	/* a branch uses r1 as a mask */
	if (flags & F_HIDE_MASK)
		(void) snprintf(buf, buflen, "%s", R[inst->rr.r2]);
	else if (flags & F_R1_IS_MASK)
		(void) snprintf(buf, buflen, "%s,%s", M[inst->rr.r1],
		    R[inst->rr.r2]);
```

```c
	else
		(void) snprintf(buf, buflen, "%s,%s", R[inst->rr.r1],
		    R[inst->rr.r2]);
}

/* ARGSUSED */
static void
fmt_rrd(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	(void) snprintf(buf, buflen, "%s,%s,%s", R[inst->rrd.r1],
	    R[inst->rrd.r3], R[inst->rrd.r2]);
}

/* ARGSUSED */
static void
fmt_rx_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d2 = val_4_8(inst->rx_a.d2h, inst->rx_b.d2l);

	(void) snprintf(buf, buflen, "%s,%u(%s,%s)", R[inst->rx_a.r1],
	    d2, X[inst->rx_a.x2], B[inst->rx_a.b2]);
}

/* ARGSUSED */
static void
fmt_rx_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d2 = val_4_8(inst->rx_b.d2h, inst->rx_b.d2l);

	if (flags & F_HIDE_MASK)
		(void) snprintf(buf, buflen, "%u(%s,%s)",
		    d2, X[inst->rx_b.x2], B[inst->rx_b.b2]);
	else
		(void) snprintf(buf, buflen, "%s,%u(%s,%s)", M[inst->rx_b.m1],
		    d2, X[inst->rx_b.x2], B[inst->rx_b.b2]);
}

/* ARGSUSED */
static void
fmt_rxe(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d2 = val_4_8(inst->rxe.d2h, inst->rxe.d2l);

	(void) snprintf(buf, buflen, "%s,%u(%s,%s)",
	    R[inst->rxe.r1], d2, X[inst->rxe.x2], B[inst->rxe.b2]);
}

/* ARGSUSED */
static void
fmt_rxf(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d2 = val_4_8(inst->rxf.d2h, inst->rxf.d2l);

	(void) snprintf(buf, buflen, "%s,%s,%u(%s,%s)",
	    R[inst->rxf.r1], R[inst->rxf.r3], d2, X[inst->rxf.x2],
	    B[inst->rxf.b2]);
}

/* ARGSUSED */
static void
fmt_rxy_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d2;

	d2 = val_8_4_8(inst->rxy_a.dh2, inst->rxy_a.dl2h, inst->rxy_a.dl2l);
	(void) snprintf(buf, buflen, "%s,%u(%s,%s)",
	    R[inst->rxy_a.r1], d2, X[inst->rxy_a.x2], B[inst->rxy_a.b2]);
}

/* ARGSUSED */
static void
fmt_rxy_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d2;

	d2 = val_8_4_8(inst->rxy_b.dh2, inst->rxy_b.dl2h, inst->rxy_b.dl2l);

	(void) snprintf(buf, buflen, "%s,%u(%s,%s)",
	    M[inst->rxy_b.m1], d2, X[inst->rxy_b.x2], B[inst->rxy_b.b2]);
}

/* ARGSUSED */
static void
fmt_rs_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	const char *r1, *r3;

	if (flags & F_CTL_REG) {
		r1 = C[inst->rs_a.r1];
		r3 = C[inst->rs_a.r3];
	} else {
		r1 = R[inst->rs_a.r1];
		r3 = R[inst->rs_a.r3];
	}

	(void) snprintf(buf, buflen, "%s,%s,%u(%s)", r1, r3,
	    val_4_8(inst->rs_a.d2h, inst->rs_a.d2l), B[inst->rs_a.b2]);
}

/* ARGSUSED */
static void
fmt_rs_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	(void) snprintf(buf, buflen, "%s,%s,%u(%s)", R[inst->rs_b.r1],
	    M[inst->rs_b.m3], val_4_8(inst->rs_b.d2h, inst->rs_b.d2l),
	    B[inst->rs_b.b2]);
}

/* ARGSUSED */
static void
fmt_rsl_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	(void) snprintf(buf, buflen, "%u(%u,%s)",
	    val_4_8(inst->rsl_a.d1h, inst->rsl_a.d1l), inst->rsl_a.l1,
	    B[inst->rsl_a.b1]);
}

/* ARGSUSED */
static void
fmt_rsl_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	(void) snprintf(buf, buflen, "%s,%u(%u,%s),%s",
	    R[inst->rsl_b.r1],
	    val_4_8(inst->rsl_b.d2h, inst->rsl_b.d2l), inst->rsl_b.l2,
	    B[inst->rsl_b.b2], M[inst->rsl_b.m3]);
}

/* ARGSUSED */
static void
fmt_rsy_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
```

```
	const char *r1, *r3;
	uint32_t d2;

	d2 = val_8_4_8(inst->rsy_a.dh2, inst->rsy_a.dl2h, inst->rsy_a.dl2l);

	if (flags & F_CTL_REG) {
		r1 = C[inst->rsy_a.r1];
		r3 = C[inst->rsy_a.r3];
	} else {
		r1 = R[inst->rsy_a.r1];
		r3 = R[inst->rsy_a.r3];
	}

	(void) snprintf(buf, buflen, "%s,%s,%u(%s)", r1, r3, d2,
	    B[inst->rsy_a.b2]);
}

/* ARGSUSED */
static void
fmt_rsy_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d2;

	d2 = val_8_4_8(inst->rsy_b.dh2, inst->rsy_b.dl2h, inst->rsy_b.dl2l);

	(void) snprintf(buf, buflen, "%s,%s,%u(%s)",
	    R[inst->rsy_b.r1], M[inst->rsy_b.m3],
	    d2, B[inst->rsy_b.b2]);
}

/* ARGSUSED */
static void
fmt_rsi(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->rsi.ri2);

	(void) snprintf(buf, buflen, "%s,%s,%#x", R[inst->rsi.r1],
	    R[inst->rsi.r3], ri2);
}

/* ARGSUSED */
static void
fmt_si(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d1 = val_4_8(inst->si.d1h, inst->si.d1l);

	(void) snprintf(buf, buflen, "%u(%s),%u", d1, B[inst->si.b1],
	    inst->si.i2);
}

/* ARGSUSED */
static void
fmt_sil(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	(void) snprintf(buf, buflen, "%u(%s),%u",
	    val_4_8(inst->sil.d1h, inst->sil.d1l), B[inst->sil.b1],
	    BE_16(inst->sil.i2));
}

/* ARGSUSED */
static void
fmt_siy(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	(void) snprintf(buf, buflen, "%u(%s),%u",
	    val_8_4_8(inst->siy.dh1, inst->siy.dl1h, inst->siy.dl1l),
	    B[inst->siy.b1], inst->siy.i2);
```

```
}

/* ARGSUSED */
static void
fmt_smi(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->smi.ri2);

	(void) snprintf(buf, buflen, "%s,%#x,%u(%s)", M[inst->smi.m1], ri2,
	    val_4_8(inst->smi.d3h, inst->smi.d3l), B[inst->smi.b3]);
}

/* ARGSUSED */
static void
fmt_s(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d = val_4_8(inst->s.d2h, inst->s.d2l);

	(void) snprintf(buf, buflen, "%u(%s)", d, B[inst->s.b2]);
}

/* ARGSUSED */
static void
fmt_ss_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d1, d2;

	d1 = val_4_8(inst->ss_a.d1h, inst->ss_a.d1l);
	d2 = val_4_8(inst->ss_a.d2h, inst->ss_a.d2l);

	(void) snprintf(buf, buflen, "%u(%u,%s),%u(%s)",
	    d1, inst->ss_a.l + 1, B[inst->ss_a.b1],
	    d2, B[inst->ss_a.b2]);
}

/* ARGSUSED */
static void
fmt_ss_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d1, d2;

	d1 = val_4_8(inst->ss_b.d1h, inst->ss_b.d1l);
	d2 = val_4_8(inst->ss_b.d2h, inst->ss_b.d2l);

	(void) snprintf(buf, buflen, "%u(%u,%s),%u(%u,%s)",
	    d1, inst->ss_b.l1 + 1, B[inst->ss_b.b1],
	    d2, inst->ss_b.l2 + 1, B[inst->ss_b.b2]);
}

/* ARGSUSED */
static void
fmt_ss_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d1, d2;

	d1 = val_4_8(inst->ss_c.d1h, inst->ss_c.d1l);
	d2 = val_4_8(inst->ss_c.d2h, inst->ss_c.d2l);

	(void) snprintf(buf, buflen, "%u(%u,%s),%u(%s),%u",
	    d1, inst->ss_c.l1, B[inst->ss_c.b1],
	    d2, B[inst->ss_c.b2], inst->ss_c.i3);
}

/* ARGSUSED */
static void
fmt_ss_d(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
```

```
{
	uint32_t d1, d2;

	d1 = val_4_8(inst->ss_d.d1h, inst->ss_d.d1l);
	d2 = val_4_8(inst->ss_d.d2h, inst->ss_d.d2l);

	(void) snprintf(buf, buflen, "%u(%s,%s),%u(%s),%s",
	    d1, R[inst->ss_d.r1], B[inst->ss_d.b1],
	    d2, B[inst->ss_d.b2], R[inst->ss_d.r3]);
}

/* ARGSUSED */
static void
fmt_ss_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d2, d4;

	d2 = val_4_8(inst->ss_e.d2h, inst->ss_e.d2l);
	d4 = val_4_8(inst->ss_e.d4h, inst->ss_e.d4l);

	(void) snprintf(buf, buflen, "%s,%u(%s),%s,%u(%s)",
	    R[inst->ss_e.r1], d2, B[inst->ss_e.b2],
	    R[inst->ss_e.r3], d4, B[inst->ss_e.b4]);
}

/* ARGSUSED */
static void
fmt_ss_f(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d1, d2;

	d1 = val_4_8(inst->ss_f.d1h, inst->ss_f.d1l);
	d2 = val_4_8(inst->ss_f.d2h, inst->ss_f.d2l);

	(void) snprintf(buf, buflen, "%u(%s),%u(%u,%s)",
	    d1, B[inst->ss_f.b1], d2, inst->ss_f.l2,
	    B[inst->ss_f.b2]);
}

/* ARGSUSED */
static void
fmt_sse(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d1 = val_4_8(inst->sse.d1h, inst->sse.d1l);
	uint32_t d2 = val_4_8(inst->sse.d2h, inst->sse.d2l);

	(void) snprintf(buf, buflen, "%u(%s),%u(%s)",
	    d1, B[inst->sse.b1], d2, B[inst->sse.b2]);
}

/* ARGSUSED */
static void
fmt_ssf(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
{
	uint32_t d1 = val_4_8(inst->ssf.d1h, inst->ssf.d1l);
	uint32_t d2 = val_4_8(inst->ssf.d2h, inst->ssf.d2l);

	(void) snprintf(buf, buflen, "%u(%s),%u(%s),%s",
	    d1, B[inst->ssf.b1],
	    d2, B[inst->ssf.b2], R[inst->ssf.r3]);
}

static void (*fmt_fxns[IF_NFMTS])(uint64_t, union inst *, char *, size_t,
    int) = {
	[IF_ZERO]	= fmt_zero,
	[IF_DIAG]	= fmt_diag,
        [IF_E]          = fmt_e,
        [IF_I]          = fmt_i,
        [IF_IE]         = fmt_ie,
        [IF_MII]        = fmt_mii,
        [IF_RIa]        = fmt_ri_a,
        [IF_RIb]        = fmt_ri_b,
        [IF_RIc]        = fmt_ri_c,
        [IF_RIEa]       = fmt_rie_a,
        [IF_RIEb]       = fmt_rie_b,
        [IF_RIEc]       = fmt_rie_c,
        [IF_RIEd]       = fmt_rie_d,
        [IF_RIEe]       = fmt_rie_e,
        [IF_RIEf]       = fmt_rie_f,
        [IF_RILa]       = fmt_ril_a,
        [IF_RILb]       = fmt_ril_b,
        [IF_RILc]       = fmt_ril_c,
        [IF_RIS]        = fmt_ris,
        [IF_RR]         = fmt_rr,
        [IF_RRD]        = fmt_rrd,
        [IF_RRE]        = fmt_rre,
        [IF_RRFa]       = fmt_rrf_a,
        [IF_RRFb]       = fmt_rrf_b,
        [IF_RRFc]       = fmt_rrf_c,
        [IF_RRFd]       = fmt_rrf_d,
        [IF_RRFe]       = fmt_rrf_e,
        [IF_RRS]        = fmt_rrs,
        [IF_RSa]        = fmt_rs_a,
        [IF_RSb]        = fmt_rs_b,
        [IF_RSI]        = fmt_rsi,
        [IF_RSLa]       = fmt_rsl_a,
        [IF_RSLb]       = fmt_rsl_b,
        [IF_RSYa]       = fmt_rsy_a,
        [IF_RSYb]       = fmt_rsy_b,
        [IF_RXa]        = fmt_rx_a,
        [IF_RXb]        = fmt_rx_b,
        [IF_RXE]        = fmt_rxe,
        [IF_RXF]        = fmt_rxf,
        [IF_RXYa]       = fmt_rxy_a,
        [IF_RXYb]       = fmt_rxy_b,
        [IF_S]          = fmt_s,
        [IF_SI]         = fmt_si,
        [IF_SIL]        = fmt_sil,
        [IF_SIY]        = fmt_siy,
        [IF_SMI]        = fmt_smi,
        [IF_SSa]        = fmt_ss_a,
        [IF_SSb]        = fmt_ss_b,
        [IF_SSc]        = fmt_ss_c,
        [IF_SSd]        = fmt_ss_d,
        [IF_SSe]        = fmt_ss_e,
        [IF_SSf]        = fmt_ss_f,
        [IF_SSE]        = fmt_sse,
        [IF_SSF]        = fmt_ssf,
};

static int
dis_s390(uint64_t addr, union inst *inst, char *buf, size_t buflen, int mach)
{
        const struct inst_table *tbl = &tbl_xx[inst->raw[0]];
        int tmp;

        while (tbl->it_fmt == IF_TBL || tbl->it_fmt == IF_MULTI) {
                if (tbl->it_fmt == IF_TBL) {
        while (tbl->fmt == IF_TBL || tbl->fmt == IF_MULTI) {
                if (tbl->fmt == IF_TBL) {
                        int idx;
```

```
                        idx   = inst->raw[tbl->it_u.it_table.it_off];
                        idx >>= tbl->it_u.it_table.it_shift;
                        idx  &= tbl->it_u.it_table.it_mask;

                        tbl = &tbl->it_u.it_table.it_ptr[idx];
                } else if (tbl->it_fmt == IF_MULTI) {
                        tbl = &tbl->it_u.it_multi.it_ptr[mach];
                        idx   = inst->raw[tbl->u.table.off];
                        idx >>= tbl->u.table.shift;
                        idx  &= tbl->u.table.mask;

                        tbl = &tbl->u.table.ptr[idx];
                } else if (tbl->fmt == IF_MULTI) {
                        tbl = &tbl->u.multi.ptr[mach];
                }
        }

        if (tbl->it_fmt == IF_INVAL)
        if (tbl->fmt == IF_INVAL)
                goto inval;

        if ((tbl->it_u.it_inst.it_flags & mach) == 0)
        if ((tbl->u.inst.flags & mach) == 0)
                goto inval;

        tmp = snprintf(buf, buflen, "%-7s ", tbl->it_u.it_inst.it_name);
        tmp = snprintf(buf, buflen, "%-7s ", tbl->u.inst.name);

        fmt_fxns[tbl->it_fmt](addr, inst, buf + tmp, buflen - tmp,
            tbl->it_u.it_inst.it_flags);
        fmt_fxns[tbl->fmt](addr, inst, buf + tmp, buflen - tmp,
            tbl->u.inst.flags);

        return (0);

inval:
        (void) snprintf(buf, buflen, "??");

        /*
         * Even if we don't know how to disassemble the instruction, we know
         * how long it is, so we "succeed" even when we fail.
         */
        return (0);
}
```

______***unchanged_portion_omitted_***