

```

*****
79676 Wed Oct 14 16:30:24 2015
new/usr/src/lib/libdisasm/common/dis_s390x.c
patch fixups
*****
1 /*
2  * This file and its contents are supplied under the terms of the
3  * Common Development and Distribution License ("CDDL"), version 1.0.
4  * You may only use this file in accordance with the terms of version
5  * 1.0 of the CDDL.
6  *
7  * A full copy of the text of the CDDL should have accompanied this
8  * source. A copy of the CDDL is also available via the Internet at
9  * http://www.illumos.org/license/CDDL.
10 */

12 /*
13  * Copyright 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
14  */

16 #include <stdio.h>
17 #include <libdisasm.h>
18 #include <sys/sysmacros.h>
19 #include <sys/debug.h>
20 #endif /* ! codereview */
21 #include <sys/byteorder.h>

23 #include "libdisasm_impl.h"

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

27 /*
28  * Throughout this file, the instruction format names based on:
29  * SA22-7832-09 z/Architecture Principles of Operation
30  *
31  * System/370, ESA/390, and earlier z/Architecture POP use slightly
32  * different names for the formats (the variant names are numeric). For the
33  * sake of simplicity, we use the most detailed definitions - z/Architecture.
34  *
35  * For ESA/390 we map the formats:
36  * E -> E
37  * I -> I
38  * RR -> RR
39  * RRE -> RRE
40  * RRF -> RRD & RRFa-e
41  * RX -> RXa-b
42  * RXE -> RXE
43  * RXF -> RXF
44  * RS -> RSa-b
45  * RSE -> RSYa-b
46  * RSL -> RSLa
47  * RSI -> RSI
48  * RI -> RIa-c
49  * RIL -> RILa-c
50  * SI -> SI
51  * S -> S
52  * SS -> SSa-b & SSd-e
53  * SSE -> SSE
54  *
55  * For System/370 we map the formats:
56  * RR -> RR
57  * RX -> RXa-b
58  * RS -> RSa-b
59  * SI -> SI
60  * S -> S
61  * SS -> SSa-c

```

```

62 *
63 * Disassembly begins in tbl_xx. The first byte of the instruction is used
64 * as the index. This yields either an instruction or a sub-table.
65 *
66 * If an instruction is encountered, its format field is used to format the
67 * instruction.
68 *
69 * There are two types of sub-tables: extended opcode tables (indicated with
70 * IF_TBL) or a multiple mnemonics tables (indicated with IF_MULTII).
71 *
72 * Extended opcode tables indicate which additional bits of the instruction
73 * should be inspected. These bits are used as an index into the sub table.
74 *
75 * Multiple mnemonic tables are used to print different mnemonics depending
19 * Multiple mnemonic tables, are used to print different mnemonics depending
76 * on the architecture. Over the years, certain instructions got a new
77 * preferred mnemonic. For example, 0xa70 is test-under-mask-high (tmh) on
78 * System/390. On z/Architecture systems, the instruction behaves
79 * identically (and the assembler happily accepts tmh), but the preferred
80 * mnemonic is tmlh (test-under-mask-low-high) because z/Architecture
81 * extended the general purpose registers from 32 bits to 64 bits. The
82 * current architecture flag (e.g., F_390) is used to index into the
83 * sub-table.
84 *
85 * Regardless of which sub-table is encountered, the selected entry in the
86 * sub-table is interpreted using the same rules as the contents of tbl_xx.
87 *
88 * Finally, we use the extended opcode sub-table mechanism to pretty print
89 * the branching instructions. All branches are conditional based on a
90 * 4-bit mask indicating which value of the condition code will result in a
91 * taken branch. In order to produce a more human friendly output, we use
92 * the 4-bit mask as an extended opcode to break up the branching
93 * instruction into 16 different ones. For example, instead of printing:
94 *
95 *     bc 7,0x123(%r1,%r2)
96 *
97 * we print:
98 *
99 *     bne 0x123(%r1,%r2)
100 *
101 * Note that we are using designated initializers via the INSTR/TABLE/MULTI
102 * macros and therefore the below tables can be sparse. We rely on unset
103 * entries having zero format fields (aka. IF_INVAL) per C99.
104 #endif /* ! codereview */
105 */

107 /* BEGIN CSTYLED */
108 enum ifmt {
109     /* invalid */
110     IF_INVAL = 0,

112     /* indirection */
113     IF_TBL,
114     IF_MULTII,

116     /* 2-byte */
117     IF_ZERO, /* 370, 390, z */
118     IF_E, /* 390, z */
119     IF_I, /* 390, z */
120     IF_RR, /* 370, 390, z */

122     /* 4-byte */
123     IF_DIAG, /* 370, 390, z */
124     IF_IE, /* z */
125     IF_RIa, /* 390, z */
126     IF_RIb, /* 390, z */

```

```

127     IF_RIc,          /*      390, z */
128     IF_RRD,          /*      390, z */ /* on 390 these are RRF */
129     IF_RRE,          /*      390, z */
130     IF_RRFa,         /*      390, z */
131     IF_RRFb,         /*      390, z */
132     IF_RRFc,         /*      390, z */
133     IF_RRFd,         /*      390, z */
134     IF_RRFe,         /*      390, z */
135     IF_RSa,          /* 370, 390, z */
136     IF_RSb,          /* 370, 390, z */
137     IF_RSI,          /*      390, z */
138     IF_RXa,          /* 370, 390, z */
139     IF_RXb,          /* 370, 390, z */
140     IF_S,            /* 370, 390, z */
141     IF_SI,           /* 370, 390, z */

143     /* 6-byte */
144     IF_MIL,          /*      z */
145     IF_RIEa,         /*      z */
146     IF_RIEb,         /*      z */
147     IF_RIEc,         /*      z */
148     IF_RIEd,         /*      z */
149     IF_RIEe,         /*      z */
150     IF_RIEf,         /*      z */
151     IF_RILa,         /*      390, z */
152     IF_RILb,         /*      390, z */
153     IF_RILc,         /*      390, z */
154     IF_RIS,          /*      z */
155     IF_RRS,          /*      z */
156     IF_RSLa,         /*      390, z */
157     IF_RSLb,         /*      z */
158     IF_RSYa,         /*      z */
159     IF_RSYb,         /*      z */
160     IF_RXE,          /*      390, z */
161     IF_RXF,          /*      390, z */
162     IF_RXYa,         /*      z */
163     IF_RXYb,         /*      z */
164     IF_SIL,          /*      z */
165     IF_SIY,          /*      z */
166     IF_SMI,          /*      z */
167     IF_SSa,          /* 370, 390, z */
168     IF_S Sb,         /* 370, 390, z */
169     IF_SSc,          /* 370, 390, z */
170     IF_S Sd,         /*      390, z */
171     IF_S Se,         /*      390, z */
172     IF_S Sf,         /*      390, z */
173     IF_S Se,         /*      390, z */
174     IF_S Sf,         /*      z */
175 };

177 #define IF_NFMTS      (IF_S Sf + 1)

179 #define F_370          0x0001      /* 370 */
180 #define F_390          0x0002      /*      390 */
181 #define F_Z            0x0004      /*      z */
182 #define F_SIGNED_IMM  0x0010      /* 370, 390, z */
183 #define F_CTL_REG     0x0020      /* 370, 390, z */
184 #define F_HIDE_MASK   0x0040      /* 370, 390, z */
185 #define F_R1_IS_MASK  0x0080      /* 370, 390, z */
186 /* END CSTYLED */

188 struct inst_table {
189     union {
190         struct {
191             const char *it_name;
192             unsigned it_flags;

```

```

193     } it_inst;
194     struct {
195         const struct inst_table *it_ptr;
196         uint8_t it_off:4;
197         uint8_t it_shift:4;
198         uint8_t it_mask;
199     } it_table;
200     struct {
201         const struct inst_table *it_ptr;
202     } it_multi;
203 } it_u;
204 enum ifmt it_fmt;
44     const char *name;
45     unsigned flags;
46 } inst;
47 struct {
48     const struct inst_table *ptr;
49     uint8_t off:4;
50     uint8_t shift:4;
51     uint8_t mask;
52 } table;
53 struct {
54     const struct inst_table *ptr;
55 } multi;
56 } u;
57 enum ifmt fmt;
205 };

unchanged_portion_omitted

539 #define INSTR(op, m, fm, fl) [op] = { \
540     .it_u.it_inst = { \
541         .it_name = (m), \
542         .it_flags = (fl), \
543     }, \
544     .it_fmt = (fm), \
545     .fmt = (fm), \
546 #define TABLE(op, tbl, o, s, m) [op] = { \
547     .it_u.it_table = { \
548         .it_ptr = (tbl), \
549         .it_off = (o), \
550         .it_shift = (s), \
551         .it_mask = (m), \
552     }, \
553     .it_fmt = IF_TBL, \
554     .fmt = IF_TBL, \
555 #define MULTI(op, tbl) [op] = { \
556     .it_u.it_multi.it_ptr = (tbl), \
557     .it_fmt = IF_MULTI, \
558     .u.multi.ptr = (tbl), \
559     .fmt = IF_MULTI, \
560 }

560 /*
561 * Instruction tables based on:
562 * GA22-7000-4 System/370 Principles of Operation
563 * SA22-7201-08 ESA/390 Principles of Operation

```

```

564 * SA22-7832-09 z/Architecture Principles of Operation
565 */

567 /* BEGIN CSTYLED */
568 static const struct inst_table tbl_01xx[256] = {
569     INSTR(0x01, "pr", IF_E, F_390 | F_Z),
570     INSTR(0x02, "upt", IF_E, F_390 | F_Z),
571     INSTR(0x04, "ptff", IF_E, F_Z),
572     INSTR(0x07, "sckpf", IF_E, F_390 | F_Z),
573     INSTR(0x0a, "pfpo", IF_E, F_Z),
574     INSTR(0x0b, "tam", IF_E, F_390 | F_Z),
575     INSTR(0x0c, "sam24", IF_E, F_390 | F_Z),
576     INSTR(0x0d, "sam31", IF_E, F_390 | F_Z),
577     INSTR(0x0e, "sam64", IF_E, F_Z),
578     INSTR(0xff, "trap2", IF_E, F_390 | F_Z),
579 };
    unchanged_portion_omitted

1793 /* B and X registers are still registers - print them the same way */
1794 #define B R
1795 #define X R

1797 static inline uint32_t
1798 val_8_4_8(uint32_t hi, uint32_t mid, uint32_t lo)
1799 {
1800     ASSERT0(hi & ~0xff);
1801     ASSERT0(mid & ~0xf);
1802     ASSERT0(lo & ~0xff);
1803 #endif /* ! codereview */
1804     return ((hi << 12) | (mid << 8) | lo);
1805 }

1807 static inline uint32_t
1808 val_16_16(uint32_t hi, uint32_t lo)
1809 {
1810     ASSERT0(hi & ~0xffff);
1811     ASSERT0(lo & ~0xffff);
1812 #endif /* ! codereview */
1813     return ((BE_16(hi) << 16) | BE_16(lo));
1814 }

1816 static inline int32_t
1817 sval_16_16(uint32_t hi, uint32_t lo)
1818 {
1819     return (val_16_16(hi, lo));
1820 }

1822 static inline uint32_t
1823 val_8_16(uint32_t hi, uint32_t lo)
1824 {
1825     ASSERT0(hi & ~0xff);
1826     ASSERT0(lo & ~0xffff);
1827 #endif /* ! codereview */
1828     return ((hi << 16) | BE_16(lo));
1829 }

1831 static inline int32_t
1832 sval_8_16(uint32_t hi, uint32_t lo)
1833 {
1834     int32_t tmp = val_8_16(hi, lo);

1836     /* sign extend */
1837 #endif /* ! codereview */
1838     if (tmp & 0x00800000)
1839         return (0xff000000 | tmp);
1840     return (tmp);

```

```

1841 }

1843 static inline uint32_t
1844 val_4_8(uint32_t hi, uint32_t lo)
1845 {
1846     ASSERT0(hi & ~0xf);
1847     ASSERT0(lo & ~0xff);
1848 #endif /* ! codereview */
1849     return ((hi << 8) | lo);
1850 }

1852 static inline int32_t
1853 sval_4_8(uint32_t hi, uint32_t lo)
1854 {
1855     uint32_t tmp = val_4_8(hi, lo);

1857     /* sign extend */
1858 #endif /* ! codereview */
1859     if (tmp & 0x800)
1860         return (0xffff000 | tmp);
1861     return (tmp);
1862 }

1864 /* ARGSUSED */
1865 static void
1866 fmt_zero(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1867 {
1868     (void) snprintf(buf, buflen, "0x00, 0x00");
1869 }

1871 /* ARGSUSED */
1872 static void
1873 fmt_diag(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1874 {
1875     (void) snprintf(buf, buflen, "%#x",
1876         val_8_16(inst->diag.par1, inst->diag.par2));
1877 }

1879 /* ARGSUSED */
1880 static void
1881 fmt_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1882 {
1883     /* nothing to do */
1884 }

1886 /* ARGSUSED */
1887 static void
1888 fmt_i(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1889 {
1890     (void) snprintf(buf, buflen, "%#x", inst->i.i);
1891 }

1893 /* ARGSUSED */
1894 static void
1895 fmt_ie(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1896 {
1897     (void) snprintf(buf, buflen, "%u,%u", inst->ie.i1, inst->ie.i2);
1898 }

1900 /* ARGSUSED */
1901 static void
1902 fmt_mii(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1903 {
1904     uint64_t ri2 = addr + 2 * sval_4_8(inst->mii.ri2h, inst->mii.ri2l);
1905     uint64_t ri3 = addr + 2 * sval_8_16(inst->mii.ri3h, inst->mii.ri3l);

```

```

1907     (void) snprintf(buf, buflen, "%s,%#x,%#x", M[inst->mii.ml], ri2, ri3);
1908 }

1910 /* ARGSUSED */
1911 static void
1912 fmt_ril_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1913 {
1914     (void) snprintf(buf, buflen, "%s,%u", R[inst->ril_a.r1],
1915                   val_16_16(inst->ril_a.i2h, inst->ril_a.i2l));
1916 }

1918 /* ARGSUSED */
1919 static void
1920 fmt_ril_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1921 {
1922     uint64_t ri2 = addr + 2 *
1923             sval_16_16(inst->ril_b.ri2h, inst->ril_b.ri2l);
1924
1925     (void) snprintf(buf, buflen, "%s,%#x", R[inst->ril_b.r1], ri2);
1926 }

1928 /* ARGSUSED */
1929 static void
1930 fmt_ril_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1931 {
1932     uint64_t ri2 = addr + 2 *
1933             sval_16_16(inst->ril_c.ri2h, inst->ril_c.ri2l);
1934
1935     (void) snprintf(buf, buflen, "%s,%#x", M[inst->ril_c.m1], ri2);
1936 }

1938 /* ARGSUSED */
1939 static void
1940 fmt_ris(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1941 {
1942     uint32_t d4 = val_4_8(inst->ris.d4h, inst->ris.d4l);
1943
1944     (void) snprintf(buf, buflen, "%s,%u,%s,%u(%s)",
1945                   R[inst->ris.r1], inst->ris.i2, M[inst->ris.m3], d4,
1946                   B[inst->ris.b4]);
1947 }

1949 /* ARGSUSED */
1950 static void
1951 fmt_ri_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1952 {
1953     uint16_t i2 = BE_16(inst->ri_a.i2);
1954
1955     if (flags & F_SIGNED_IMM)
1956         (void) snprintf(buf, buflen, "%s,%d", R[inst->ri_a.r1],
1957                       (int16_t)i2);
1958     else
1959         (void) snprintf(buf, buflen, "%s,%u", R[inst->ri_a.r1],
1960                       i2);
1961 }

1963 /* ARGSUSED */
1964 static void
1965 fmt_ri_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1966 {
1967     uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->ri_b.ri2);
1968
1969     (void) snprintf(buf, buflen, "%s,%#x", R[inst->ri_b.r1], ri2);
1970 }

1972 static void

```

```

1973 fmt_ri_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1974 {
1975     uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->ri_c.ri2);
1976
1977     if (flags & F_HIDE_MASK)
1978         (void) snprintf(buf, buflen, "%#x", ri2);
1979     else
1980         (void) snprintf(buf, buflen, "%s,%#x", M[inst->ri_c.m1], ri2);
1981 }

1983 /* ARGSUSED */
1984 static void
1985 fmt_rie_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1986 {
1987     (void) snprintf(buf, buflen, "%s,%u,%s", R[inst->rie_a.r1],
1988                   BE_16(inst->rie_a.i2), M[inst->rie_a.m3]);
1989 }

1991 /* ARGSUSED */
1992 static void
1993 fmt_rie_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1994 {
1995     uint64_t ri4 = addr + 2 * (int16_t)BE_16(inst->rie_b.ri4);
1996
1997     (void) snprintf(buf, buflen, "%s,%s,%s,%#x", R[inst->rie_b.r1],
1998                   R[inst->rie_b.r2], M[inst->rie_b.m3], ri4);
1999 }

2001 /* ARGSUSED */
2002 static void
2003 fmt_rie_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2004 {
2005     uint64_t ri4 = addr + 2 * (int16_t)BE_16(inst->rie_c.ri4);
2006
2007     (void) snprintf(buf, buflen, "%s,%u,%s,%#x", R[inst->rie_c.r1],
2008                   inst->rie_c.i2, M[inst->rie_c.m3], ri4);
2009 }

2011 /* ARGSUSED */
2012 static void
2013 fmt_rie_d(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2014 {
2015     (void) snprintf(buf, buflen, "%s,%s,%u", R[inst->rie_d.r1],
2016                   R[inst->rie_d.r3], BE_16(inst->rie_d.i2));
2017 }

2019 /* ARGSUSED */
2020 static void
2021 fmt_rie_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2022 {
2023     uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->rie_e.ri2);
2024
2025     (void) snprintf(buf, buflen, "%s,%s,%#x", R[inst->rie_e.r1],
2026                   R[inst->rie_e.r3], ri2);
2027 }

2029 /* ARGSUSED */
2030 static void
2031 fmt_rie_f(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2032 {
2033     (void) snprintf(buf, buflen, "%s,%s,%u,%u,%u", R[inst->rie_f.r1],
2034                   R[inst->rie_f.r2], inst->rie_f.i3, inst->rie_f.i4,
2035                   inst->rie_f.i5);
2036 }

2038 /* ARGSUSED */

```

```

2039 static void
2040 fmt_rre(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2041 {
2042     (void) snprintf(buf, buflen, "%s,%s", R[inst->rre.r1], R[inst->rre.r2]);
2043 }
2044
2045 /* ARGSUSED */
2046 static void
2047 fmt_rrf_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2048 {
2049     (void) snprintf(buf, buflen, "%s,%s,%s",
2050                    R[inst->rrf_ab.r1], R[inst->rrf_ab.r2], R[inst->rrf_ab.r3]);
2051 }
2052
2053 /* ARGSUSED */
2054 static void
2055 fmt_rrf_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2056 {
2057     (void) snprintf(buf, buflen, "%s,%s,%s",
2058                    R[inst->rrf_ab.r1], R[inst->rrf_ab.r3], R[inst->rrf_ab.r2]);
2059 }
2060
2061 /* ARGSUSED */
2062 static void
2063 fmt_rrf_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2064 {
2065     (void) snprintf(buf, buflen, "%s,%s,%s",
2066                    R[inst->rrf_cde.r1], R[inst->rrf_cde.r2], M[inst->rrf_cde.m3]);
2067 }
2068
2069 /* ARGSUSED */
2070 static void
2071 fmt_rrf_d(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2072 {
2073     (void) snprintf(buf, buflen, "%s,%s,%s",
2074                    R[inst->rrf_cde.r1], R[inst->rrf_cde.r2], M[inst->rrf_cde.m4]);
2075 }
2076
2077 /* ARGSUSED */
2078 static void
2079 fmt_rrf_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2080 {
2081     (void) snprintf(buf, buflen, "%s,%s,%s,%s",
2082                    R[inst->rrf_cde.r1], M[inst->rrf_cde.m3],
2083                    R[inst->rrf_cde.r2], M[inst->rrf_cde.m4]);
2084 }
2085
2086 /* ARGSUSED */
2087 static void
2088 fmt_rrs(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2089 {
2090     (void) snprintf(buf, buflen, "%s,%s,%s,%u(%s)", R[inst->rrs.r1],
2091                    R[inst->rrs.r2], M[inst->rrs.m3],
2092                    val_4_8(inst->rrs.d4h, inst->rrs.d4l), B[inst->rrs.b4]);
2093 }
2094
2095 /* ARGSUSED */
2096 static void
2097 fmt_rr(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2098 {
2099     /* a branch uses r1 as a mask */
2100     if (flags & F_HIDE_MASK)
2101         (void) snprintf(buf, buflen, "%s", R[inst->rr.r2]);
2102     else if (flags & F_R1_IS_MASK)
2103         (void) snprintf(buf, buflen, "%s,%s", M[inst->rr.r1],
2104                        R[inst->rr.r2]);

```

```

2105     else
2106         (void) snprintf(buf, buflen, "%s,%s", R[inst->rr.r1],
2107                        R[inst->rr.r2]);
2108 }
2109
2110 /* ARGSUSED */
2111 static void
2112 fmt_rrd(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2113 {
2114     (void) snprintf(buf, buflen, "%s,%s,%s", R[inst->rrd.r1],
2115                    R[inst->rrd.r3], R[inst->rrd.r2]);
2116 }
2117
2118 /* ARGSUSED */
2119 static void
2120 fmt_rx_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2121 {
2122     uint32_t d2 = val_4_8(inst->rx_a.d2h, inst->rx_b.d2l);
2123
2124     (void) snprintf(buf, buflen, "%s,%u(%s,%s)", R[inst->rx_a.r1],
2125                    d2, X[inst->rx_a.x2], B[inst->rx_a.b2]);
2126 }
2127
2128 /* ARGSUSED */
2129 static void
2130 fmt_rx_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2131 {
2132     uint32_t d2 = val_4_8(inst->rx_b.d2h, inst->rx_b.d2l);
2133
2134     if (flags & F_HIDE_MASK)
2135         (void) snprintf(buf, buflen, "%u(%s,%s)",
2136                        d2, X[inst->rx_b.x2], B[inst->rx_b.b2]);
2137     else
2138         (void) snprintf(buf, buflen, "%s,%u(%s,%s)", M[inst->rx_b.m1],
2139                        d2, X[inst->rx_b.x2], B[inst->rx_b.b2]);
2140 }
2141
2142 /* ARGSUSED */
2143 static void
2144 fmt_rxe(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2145 {
2146     uint32_t d2 = val_4_8(inst->rxe.d2h, inst->rxe.d2l);
2147
2148     (void) snprintf(buf, buflen, "%s,%u(%s,%s)",
2149                    R[inst->rxe.r1], d2, X[inst->rxe.x2], B[inst->rxe.b2]);
2150 }
2151
2152 /* ARGSUSED */
2153 static void
2154 fmt_rxf(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2155 {
2156     uint32_t d2 = val_4_8(inst->rxf.d2h, inst->rxf.d2l);
2157
2158     (void) snprintf(buf, buflen, "%s,%s,%u(%s,%s)",
2159                    R[inst->rxf.r1], R[inst->rxf.r3], d2, X[inst->rxf.x2],
2160                    B[inst->rxf.b2]);
2161 }
2162
2163 /* ARGSUSED */
2164 static void
2165 fmt_rxy_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2166 {
2167     uint32_t d2;
2168
2169     d2 = val_8_4_8(inst->rxy_a.dh2, inst->rxy_a.dl2h, inst->rxy_a.dl2l);

```

```

2171     (void) snprintf(buf, buflen, "%s,%u(%s,%s)",
2172                    R[inst->rxy_a.r1], d2, X[inst->rxy_a.x2], B[inst->rxy_a.b2]);
2173 }

2175 /* ARGSUSED */
2176 static void
2177 fmt_rxy_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2178 {
2179     uint32_t d2;

2181     d2 = val_8_4_8(inst->rxy_b.dh2, inst->rxy_b.dl2h, inst->rxy_b.dl2l);

2183     (void) snprintf(buf, buflen, "%s,%u(%s,%s)",
2184                    M[inst->rxy_b.m1], d2, X[inst->rxy_b.x2], B[inst->rxy_b.b2]);
2185 }

2187 /* ARGSUSED */
2188 static void
2189 fmt_rs_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2190 {
2191     const char *r1, *r3;

2193     if (flags & F_CTL_REG) {
2194         r1 = C[inst->rs_a.r1];
2195         r3 = C[inst->rs_a.r3];
2196     } else {
2197         r1 = R[inst->rs_a.r1];
2198         r3 = R[inst->rs_a.r3];
2199     }

2201     (void) snprintf(buf, buflen, "%s,%s,%u(%s)", r1, r3,
2202                    val_4_8(inst->rs_a.d2h, inst->rs_a.d2l), B[inst->rs_a.b2]);
2203 }

2205 /* ARGSUSED */
2206 static void
2207 fmt_rs_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2208 {
2209     (void) snprintf(buf, buflen, "%s,%s,%u(%s)", R[inst->rs_b.r1],
2210                    M[inst->rs_b.m3], val_4_8(inst->rs_b.d2h, inst->rs_b.d2l),
2211                    B[inst->rs_b.b2]);
2212 }

2214 /* ARGSUSED */
2215 static void
2216 fmt_rsl_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2217 {
2218     (void) snprintf(buf, buflen, "%u(%u,%s)",
2219                    val_4_8(inst->rsl_a.dlh, inst->rsl_a.dll), inst->rsl_a.l1,
2220                    B[inst->rsl_a.b1]);
2221 }

2223 /* ARGSUSED */
2224 static void
2225 fmt_rsl_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2226 {
2227     (void) snprintf(buf, buflen, "%s,%u(%u,%s),%s",
2228                    R[inst->rsl_b.r1],
2229                    val_4_8(inst->rsl_b.d2h, inst->rsl_b.d2l), inst->rsl_b.l2,
2230                    B[inst->rsl_b.b2], M[inst->rsl_b.m3]);
2231 }

2233 /* ARGSUSED */
2234 static void
2235 fmt_rsy_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2236 {

```

```

2237     const char *r1, *r3;
2238     uint32_t d2;

2240     d2 = val_8_4_8(inst->rsy_a.dh2, inst->rsy_a.dl2h, inst->rsy_a.dl2l);

2242     if (flags & F_CTL_REG) {
2243         r1 = C[inst->rsy_a.r1];
2244         r3 = C[inst->rsy_a.r3];
2245     } else {
2246         r1 = R[inst->rsy_a.r1];
2247         r3 = R[inst->rsy_a.r3];
2248     }

2250     (void) snprintf(buf, buflen, "%s,%s,%u(%s)", r1, r3, d2,
2251                    B[inst->rsy_a.b2]);
2252 }

2254 /* ARGSUSED */
2255 static void
2256 fmt_rsy_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2257 {
2258     uint32_t d2;

2260     d2 = val_8_4_8(inst->rsy_b.dh2, inst->rsy_b.dl2h, inst->rsy_b.dl2l);

2262     (void) snprintf(buf, buflen, "%s,%s,%u(%s)",
2263                    R[inst->rsy_b.r1], M[inst->rsy_b.m3],
2264                    d2, B[inst->rsy_b.b2]);
2265 }

2267 /* ARGSUSED */
2268 static void
2269 fmt_rsi(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2270 {
2271     uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->rsi.ri2);

2273     (void) snprintf(buf, buflen, "%s,%s,%#x", R[inst->rsi.r1],
2274                    R[inst->rsi.r3], ri2);
2275 }

2277 /* ARGSUSED */
2278 static void
2279 fmt_si(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2280 {
2281     uint32_t d1 = val_4_8(inst->si.dlh, inst->si.dll);

2283     (void) snprintf(buf, buflen, "%u(%s),%u", d1, B[inst->si.b1],
2284                    inst->si.i2);
2285 }

2287 /* ARGSUSED */
2288 static void
2289 fmt_sil(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2290 {
2291     (void) snprintf(buf, buflen, "%u(%s),%u",
2292                    val_4_8(inst->sil.dlh, inst->sil.dll), B[inst->sil.b1],
2293                    BE_16(inst->sil.i2));
2294 }

2296 /* ARGSUSED */
2297 static void
2298 fmt_siy(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2299 {
2300     (void) snprintf(buf, buflen, "%u(%s),%u",
2301                    val_8_4_8(inst->siy.dh1, inst->siy.dlh, inst->siy.dll1),
2302                    B[inst->siy.b1], inst->siy.i2);

```

```

2303 }

2305 /* ARGSUSED */
2306 static void
2307 fmt_smi(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2308 {
2309     uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->smi.ri2);

2311     (void) snprintf(buf, buflen, "%s,%#x,%u(%s)", M[inst->smi.m1], ri2,
2312                  val_4_8(inst->smi.d3h, inst->smi.d3l), B[inst->smi.b3]);
2313 }

2315 /* ARGSUSED */
2316 static void
2317 fmt_s(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2318 {
2319     uint32_t d = val_4_8(inst->s.d2h, inst->s.d2l);

2321     (void) snprintf(buf, buflen, "%u(%s)", d, B[inst->s.b2]);
2322 }

2324 /* ARGSUSED */
2325 static void
2326 fmt_ss_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2327 {
2328     uint32_t d1, d2;

2330     d1 = val_4_8(inst->ss_a.dlh, inst->ss_a.d1l);
2331     d2 = val_4_8(inst->ss_a.d2h, inst->ss_a.d2l);

2333     (void) snprintf(buf, buflen, "%u(%u,%s),%u(%s)",
2334                  d1, inst->ss_a.l + 1, B[inst->ss_a.b1],
2335                  d2, B[inst->ss_a.b2]);
2336 }

2338 /* ARGSUSED */
2339 static void
2340 fmt_ss_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2341 {
2342     uint32_t d1, d2;

2344     d1 = val_4_8(inst->ss_b.dlh, inst->ss_b.d1l);
2345     d2 = val_4_8(inst->ss_b.d2h, inst->ss_b.d2l);

2347     (void) snprintf(buf, buflen, "%u(%u,%s),%u(%u,%s)",
2348                  d1, inst->ss_b.l1 + 1, B[inst->ss_b.b1],
2349                  d2, inst->ss_b.l2 + 1, B[inst->ss_b.b2]);
2350 }

2352 /* ARGSUSED */
2353 static void
2354 fmt_ss_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2355 {
2356     uint32_t d1, d2;

2358     d1 = val_4_8(inst->ss_c.dlh, inst->ss_c.d1l);
2359     d2 = val_4_8(inst->ss_c.d2h, inst->ss_c.d2l);

2361     (void) snprintf(buf, buflen, "%u(%u,%s),%u(%s),%u",
2362                  d1, inst->ss_c.l1, B[inst->ss_c.b1],
2363                  d2, B[inst->ss_c.b2], inst->ss_c.i3);
2364 }

2366 /* ARGSUSED */
2367 static void
2368 fmt_ss_d(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)

```

```

2369 {
2370     uint32_t d1, d2;

2372     d1 = val_4_8(inst->ss_d.dlh, inst->ss_d.d1l);
2373     d2 = val_4_8(inst->ss_d.d2h, inst->ss_d.d2l);

2375     (void) snprintf(buf, buflen, "%u(%s,%s),%u(%s),%s",
2376                  d1, R[inst->ss_d.r1], B[inst->ss_d.b1],
2377                  d2, B[inst->ss_d.b2], R[inst->ss_d.r3]);
2378 }

2380 /* ARGSUSED */
2381 static void
2382 fmt_ss_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2383 {
2384     uint32_t d2, d4;

2386     d2 = val_4_8(inst->ss_e.d2h, inst->ss_e.d2l);
2387     d4 = val_4_8(inst->ss_e.d4h, inst->ss_e.d4l);

2389     (void) snprintf(buf, buflen, "%s,%u(%s),%s,%u(%s)",
2390                  R[inst->ss_e.r1], d2, B[inst->ss_e.b2],
2391                  R[inst->ss_e.r3], d4, B[inst->ss_e.b4]);
2392 }

2394 /* ARGSUSED */
2395 static void
2396 fmt_ss_f(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2397 {
2398     uint32_t d1, d2;

2400     d1 = val_4_8(inst->ss_f.dlh, inst->ss_f.d1l);
2401     d2 = val_4_8(inst->ss_f.d2h, inst->ss_f.d2l);

2403     (void) snprintf(buf, buflen, "%u(%s),%u(%u,%s)",
2404                  d1, B[inst->ss_f.b1], d2, inst->ss_f.l2,
2405                  B[inst->ss_f.b2]);
2406 }

2408 /* ARGSUSED */
2409 static void
2410 fmt_sse(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2411 {
2412     uint32_t d1 = val_4_8(inst->sse.dlh, inst->sse.d1l);
2413     uint32_t d2 = val_4_8(inst->sse.d2h, inst->sse.d2l);

2415     (void) snprintf(buf, buflen, "%u(%s),%u(%s)",
2416                  d1, B[inst->sse.b1], d2, B[inst->sse.b2]);
2417 }

2419 /* ARGSUSED */
2420 static void
2421 fmt_ssf(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2422 {
2423     uint32_t d1 = val_4_8(inst->ssf.dlh, inst->ssf.d1l);
2424     uint32_t d2 = val_4_8(inst->ssf.d2h, inst->ssf.d2l);

2426     (void) snprintf(buf, buflen, "%u(%s),%u(%s),%s",
2427                  d1, B[inst->ssf.b1],
2428                  d2, B[inst->ssf.b2], R[inst->ssf.r3]);
2429 }

2431 static void (*fmt_fxns[IF_NFMTS])(uint64_t, union inst *, char *, size_t,
2432 int) = {
2433     [IF_ZERO] = fmt_zero,
2434     [IF_DIAG] = fmt_diag,

```

```

2435     [IF_E]      = fmt_e,
2436     [IF_I]      = fmt_i,
2437     [IF_IE]     = fmt_ie,
2438     [IF_MII]    = fmt_mii,
2439     [IF_RIa]    = fmt_ri_a,
2440     [IF_RIb]    = fmt_ri_b,
2441     [IF_RIc]    = fmt_ri_c,
2442     [IF_RIEa]   = fmt_rie_a,
2443     [IF_RIEb]   = fmt_rie_b,
2444     [IF_RIEc]   = fmt_rie_c,
2445     [IF_RIED]   = fmt_rie_d,
2446     [IF_RIEe]   = fmt_rie_e,
2447     [IF_RIEf]   = fmt_rie_f,
2448     [IF_RILa]   = fmt_ril_a,
2449     [IF_RILb]   = fmt_ril_b,
2450     [IF_RILc]   = fmt_ril_c,
2451     [IF_RIS]    = fmt_ris,
2452     [IF_RR]     = fmt_rr,
2453     [IF_RRD]    = fmt_rrd,
2454     [IF_RRE]    = fmt_rre,
2455     [IF_RRFa]   = fmt_rrf_a,
2456     [IF_RRFb]   = fmt_rrf_b,
2457     [IF_RRFc]   = fmt_rrf_c,
2458     [IF_RRFd]   = fmt_rrf_d,
2459     [IF_RRFe]   = fmt_rrf_e,
2460     [IF_RRS]    = fmt_rrs,
2461     [IF_RSa]    = fmt_rs_a,
2462     [IF_RSb]    = fmt_rs_b,
2463     [IF_RSI]    = fmt_rsi,
2464     [IF_RSLa]   = fmt_rsl_a,
2465     [IF_RSLb]   = fmt_rsl_b,
2466     [IF_RSYa]   = fmt_rsy_a,
2467     [IF_RSYb]   = fmt_rsy_b,
2468     [IF_RXa]    = fmt_rx_a,
2469     [IF_RXb]    = fmt_rx_b,
2470     [IF_RXE]    = fmt_rxe,
2471     [IF_RXF]    = fmt_rxf,
2472     [IF_RXYa]   = fmt_rxy_a,
2473     [IF_RXYb]   = fmt_rxy_b,
2474     [IF_S]      = fmt_s,
2475     [IF_SI]     = fmt_si,
2476     [IF_SIL]    = fmt_sil,
2477     [IF_SIY]    = fmt_siy,
2478     [IF_SMI]    = fmt_smi,
2479     [IF_SSa]    = fmt_ss_a,
2480     [IF_Ssb]    = fmt_ss_b,
2481     [IF_Ssc]    = fmt_ss_c,
2482     [IF_Ssd]    = fmt_ss_d,
2483     [IF_SSe]    = fmt_ss_e,
2484     [IF_Ssf]    = fmt_ss_f,
2485     [IF_SSE]    = fmt_sse,
2486     [IF_SSF]    = fmt_ssf,
2487 };

2489 static int
2490 dis_s390(uint64_t addr, union inst *inst, char *buf, size_t buflen, int mach)
2491 {
2492     const struct inst_table *tbl = &tbl_xx[inst->raw[0]];
2493     int tmp;

2495     while (tbl->it_fmt == IF_TBL || tbl->it_fmt == IF_MULTI) {
2496         if (tbl->it_fmt == IF_TBL) {
1653     while (tbl->fmt == IF_TBL || tbl->fmt == IF_MULTI) {
1654         if (tbl->fmt == IF_TBL) {
2497             int idx;

```

```

2499         idx = inst->raw[tbl->it_u.it_table.it_off];
2500         idx >>= tbl->it_u.it_table.it_shift;
2501         idx &= tbl->it_u.it_table.it_mask;

2503         tbl = &tbl->it_u.it_table.it_ptr[idx];
2504     } else if (tbl->it_fmt == IF_MULTI) {
2505         tbl = &tbl->it_u.it_multi.it_ptr[mach];
1657         idx = inst->raw[tbl->u.table.off];
1658         idx >>= tbl->u.table.shift;
1659         idx &= tbl->u.table.mask;

1661         tbl = &tbl->u.table.ptr[idx];
1662     } else if (tbl->fmt == IF_MULTI) {
1663         tbl = &tbl->u.multi.ptr[mach];
2506     }
2507 }

2509 if (tbl->it_fmt == IF_INVALID)
1667     if (tbl->fmt == IF_INVALID)
2510         goto inval;

2512 if ((tbl->it_u.it_inst.it_flags & mach) == 0)
1670     if ((tbl->u.inst.flags & mach) == 0)
2513         goto inval;

2515     tmp = snprintf(buf, buflen, "%-7s ", tbl->it_u.it_inst.it_name);
1673     tmp = snprintf(buf, buflen, "%-7s ", tbl->u.inst.name);

2517     fmt_fxns[tbl->it_fmt](addr, inst, buf + tmp, buflen - tmp,
2518         tbl->it_u.it_inst.it_flags);
1675     fmt_fxns[tbl->fmt](addr, inst, buf + tmp, buflen - tmp,
1676         tbl->u.inst.flags);

2520     return (0);

2522 inval:
2523     (void) snprintf(buf, buflen, "??");

2525     /*
2526     * Even if we don't know how to disassemble the instruction, we know
2527     * how long it is, so we "succeed" even when we fail.
2528     */
2529     return (0);
2530 }

```

\_\_\_\_\_unchanged\_portion\_omitted\_\_\_\_\_