

```

*****
18988 Thu Jul 23 08:02:30 2015
new/usr/src/cmd/dis/dis_main.c
6066 dis: support for System/370, System/390, and z/Architecture ELF bins
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2007 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  *
26  * Copyright 2011 Jason King. All rights reserved.
27  * Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
28  * Copyright 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
29 #endif /* ! codereview */
30 */

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

40 #include <libdisasm.h>

42 #include "dis_target.h"
43 #include "dis_util.h"
44 #include "dis_list.h"

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

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

55 /*
56  * Section options for -d, -D, and -s
57  */
58 #define DIS_DATA_RELATIVE      1
59 #define DIS_DATA_ABSOLUTE     2
60 #define DIS_TEXT               3

```

```

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

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

76 /*
77  * Given a symbol+offset as returned by dis_tgt_lookup(), print an appropriately
78  * formatted symbol, based on the offset and current settings.
79  */
80 void
81 getsymname(uint64_t addr, const char *symbol, off_t offset, char *buf,
82            size_t buflen)
83 {
84     if (symbol == NULL || g_numeric) {
85         if (g_flags & DIS_OCTAL)
86             (void) snprintf(buf, buflen, "%011lo", addr);
87         else
88             (void) snprintf(buf, buflen, "0x%11lx", addr);
89     } else {
90         if (g_demangle)
91             symbol = dis_demangle(symbol);

93         if (offset == 0)
94             (void) snprintf(buf, buflen, "%s", symbol);
95         else if (g_flags & DIS_OCTAL)
96             (void) snprintf(buf, buflen, "%s+0%o", symbol, offset);
97         else
98             (void) snprintf(buf, buflen, "%s+0x%x", symbol, offset);
99     }
100 }

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

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

115     return (0);
116 }

118 /*
119  * The main disassembly routine. Given a fixed-sized buffer and starting
120  * address, disassemble the data using the supplied target and libdisasm handle.
121  */
122 void
123 dis_data(dis_tgt_t *tgt, dis_handle_t *dhp, uint64_t addr, void *data,
124         size_t datalen)
125 {
126     dis_buffer_t db = { 0 };
127     char buf[BUFSIZE];

```

```

128     char symbuf[BUFSIZE];
129     const char *symbol;
130     const char *last_symbol;
131     off_t symoffset;
132     int i;
133     int bytesperline;
134     size_t symsize;
135     int isfunc;
136     size_t symwidth = 0;
137     int ret;
138     int insz = insn_size(dhp);

140     db.db_tgt = tgt;
141     db.db_data = data;
142     db.db_addr = addr;
143     db.db_size = datalen;

145     dis_set_data(dhp, &db);

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

150     symbol = NULL;

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

154         ret = dis_disassemble(dhp, addr, buf, BUFSIZE);
155         if (ret != 0 && insz > 0) {
156             /*
157              * Since we know instructions are fixed size, we
158              * always know the address of the next instruction
159              */
160             (void) snprintf(buf, sizeof (buf),
161                 "**** invalid opcode ****");
162             db.db_nextaddr = addr + insz;

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

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

170             /*
171              * On architectures with variable sized instructions
172              * we have no way to figure out where the next
173              * instruction starts if we encounter an invalid
174              * instruction. Instead we print the rest of the
175              * instruction stream as hex until we reach the
176              * next valid symbol in the section.
177              */
178             if ((next = dis_tgt_next_symbol(tgt, addr)) == 0) {
179                 db.db_nextaddr = db.db_addr + db.db_size;
180             } else {
181                 if (next > db.db_size)
182                     db.db_nextaddr = db.db_addr +
183                         db.db_size;
184                 else
185                     db.db_nextaddr = addr + next;
186             }
187         }

189         /*
190          * Print out the line as:
191          *
192          *     address:      bytes  text
193          */

```

```

194         * If there are more than 6 bytes in any given instruction,
195         * spread the bytes across two lines. We try to get symbolic
196         * information for the address, but if that fails we print out
197         * the numeric address instead.
198         *
199         * We try to keep the address portion of the text aligned at
200         * MINSYMWIDTH characters. If we are disassembling a function
201         * with a long name, this can be annoying. So we pick a width
202         * based on the maximum width that the current symbol can be.
203         * This at least produces text aligned within each function.
204         */
205         last_symbol = symbol;
206         symbol = dis_tgt_lookup(tgt, addr, &symoffset, 1, &symsize,
207             &isfunc);
208         if (symbol == NULL) {
209             symbol = dis_find_section(tgt, addr, &symoffset);
210             symsize = symoffset;
211         }

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

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

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

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

230         /* print bytes */
231         for (i = 0; i < MIN(bytesperline, (db.db_nextaddr - addr));
232             i++) {
233             int byte = *((uchar_t *)data + (addr - db.db_addr) + i);
234             if (g_flags & DIS_OCTAL)
235                 (void) printf("%03o ", byte);
236             else
237                 (void) printf("%02x ", byte);
238         }

240         /* trailing spaces for missing bytes */
241         for (; i < bytesperline; i++) {
242             if (g_flags & DIS_OCTAL)
243                 (void) printf("    ");
244             else
245                 (void) printf(" ");
246         }

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

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

```

```

260     }
262     (void) printf("\n");
264     addr = db.db_nextaddr;
265 }
266 }

268 /*
269 * libdisasm wrapper around symbol lookup. Invoke the target-specific lookup
270 * function, and convert the result using getsymname().
271 */
272 int
273 do_lookup(void *data, uint64_t addr, char *buf, size_t buflen, uint64_t *start,
274           size_t *symlen)
275 {
276     dis_buffer_t *db = data;
277     const char *symbol;
278     off_t offset;
279     size_t size;

281     /*
282      * If NULL symbol is returned, getsymname takes care of
283      * printing appropriate address in buf instead of symbol.
284      */
285     symbol = dis_tgt_lookup(db->db_tgt, addr, &offset, 0, &size, NULL);

287     if (buf != NULL)
288         getsymname(addr, symbol, offset, buf, buflen);

290     if (start != NULL)
291         *start = addr - offset;
292     if (symlen != NULL)
293         *symlen = size;

295     if (symbol == NULL)
296         return (-1);

298     return (0);
299 }

301 /*
302 * libdisasm wrapper around target reading. libdisasm will always read data
303 * in order, so update our current offset within the buffer appropriately.
304 * We only support reading from within the current object; libdisasm should
305 * never ask us to do otherwise.
306 */
307 int
308 do_read(void *data, uint64_t addr, void *buf, size_t len)
309 {
310     dis_buffer_t *db = data;
311     size_t offset;

313     if (addr < db->db_addr || addr >= db->db_addr + db->db_size)
314         return (-1);

316     offset = addr - db->db_addr;
317     len = MIN(len, db->db_size - offset);

319     (void) memcpy(buf, (char *)db->db_data + offset, len);

321     db->db_nextaddr = addr + len;

323     return (len);
324 }

```

```

326 /*
327 * Routine to dump raw data in a human-readable format. Used by the -d and -D
328 * options. We model our output after the xxd(1) program, which gives nicely
329 * formatted output, along with an ASCII translation of the result.
330 */
331 void
332 dump_data(uint64_t addr, void *data, size_t datalen)
333 {
334     uintptr_t curaddr = addr & (~0xf);
335     uint8_t *bytes = data;
336     int i;
337     int width;

339     /*
340      * Determine if the address given to us fits in 32-bit range, in which
341      * case use a 4-byte width.
342      */
343     if (((addr + datalen) & 0xffffffff00000000ULL) == 0ULL)
344         width = 8;
345     else
346         width = 16;

348     while (curaddr < addr + datalen) {
349         /*
350          * Display leading address
351          */
352         (void) printf("%0*x: ", width, curaddr);

354         /*
355          * Print out data in two-byte chunks. If the current address
356          * is before the starting address or after the end of the
357          * section, print spaces.
358          */
359         for (i = 0; i < 16; i++) {
360             if (curaddr + i < addr || curaddr + i >= addr + datalen)
361                 (void) printf(" ");
362             else
363                 (void) printf("%02x",
364                               bytes[curaddr + i - addr]);

366             if (i & 1)
367                 (void) printf(" ");
368         }

370         (void) printf(" ");

372         /*
373          * Print out the ASCII representation
374          */
375         for (i = 0; i < 16; i++) {
376             if (curaddr + i < addr ||
377                 curaddr + i >= addr + datalen) {
378                 (void) printf(" ");
379             } else {
380                 uint8_t byte = bytes[curaddr + i - addr];
381                 if (isprint(byte))
382                     (void) printf("%c", byte);
383                 else
384                     (void) printf(".");
385             }
386         }

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

390         curaddr += 16;
391     }

```

```

392 }
393
394 /*
395  * Disassemble a section implicitly specified as part of a file. This function
396  * is called for all sections when no other flags are specified. We ignore any
397  * data sections, and print out only those sections containing text.
398  */
399 void
400 dis_text_section(dis_tgt_t *tgt, dis_scn_t *scn, void *data)
401 {
402     dis_handle_t *dhp = data;
403
404     /* ignore data sections */
405     if (!dis_section_istext(scn))
406         return;
407
408     if (!g_quiet)
409         (void) printf("\nsection %s\n", dis_section_name(scn));
410
411     dis_data(tgt, dhp, dis_section_addr(scn), dis_section_data(scn),
412             dis_section_size(scn));
413 }
414
415 /*
416  * Structure passed to dis_named_{section,function} which keeps track of both
417  * the target and the libdisasm handle.
418  */
419 typedef struct callback_arg {
420     dis_tgt_t      *ca_tgt;
421     dis_handle_t   *ca_handle;
422 } callback_arg_t;
423
424 /*
425  * Disassemble a section explicitly named with -s, -d, or -D. The 'type'
426  * argument contains the type of argument given. Pass the data onto the
427  * appropriate helper routine.
428  */
429 void
430 dis_named_section(dis_scn_t *scn, int type, void *data)
431 {
432     callback_arg_t *ca = data;
433
434     if (!g_quiet)
435         (void) printf("\nsection %s\n", dis_section_name(scn));
436
437     switch (type) {
438     case DIS_DATA_RELATIVE:
439         dump_data(0, dis_section_data(scn), dis_section_size(scn));
440         break;
441     case DIS_DATA_ABSOLUTE:
442         dump_data(dis_section_addr(scn), dis_section_data(scn),
443                 dis_section_size(scn));
444         break;
445     case DIS_TEXT:
446         dis_data(ca->ca_tgt, ca->ca_handle, dis_section_addr(scn),
447                 dis_section_data(scn), dis_section_size(scn));
448         break;
449     }
450 }
451
452 /*
453  * Disassemble a function explicitly specified with '-F'. The 'type' argument
454  * is unused.
455  */
456 /* ARGSUSED */
457 void

```

```

458 dis_named_function(dis_func_t *func, int type, void *data)
459 {
460     callback_arg_t *ca = data;
461
462     dis_data(ca->ca_tgt, ca->ca_handle, dis_function_addr(func),
463             dis_function_data(func), dis_function_size(func));
464 }
465
466 /*
467  * Disassemble a complete file. First, we determine the type of the file based
468  * on the ELF machine type, and instantiate a version of the disassembler
469  * appropriate for the file. We then resolve any named sections or functions
470  * against the file, and iterate over the results (or all sections if no flags
471  * were specified).
472  */
473 void
474 dis_file(const char *filename)
475 {
476     dis_tgt_t *tgt, *current;
477     dis_scnlist_t *sections;
478     dis_funclist_t *functions;
479     dis_handle_t *dhp;
480     GElf_Ehdr ehdr;
481
482     /*
483      * First, initialize the target
484      */
485     if ((tgt = dis_tgt_create(filename)) == NULL)
486         return;
487
488     if (!g_quiet)
489         (void) printf("disassembly for %s\n\n", filename);
490
491     /*
492      * A given file may contain multiple targets (if it is an archive, for
493      * example). We iterate over all possible targets if this is the case.
494      */
495     for (current = tgt; current != NULL; current = dis_tgt_next(current)) {
496         dis_tgt_ehdr(current, &ehdr);
497
498         /*
499          * Eventually, this should probably live within libdisasm, and
500          * we should be able to disassemble targets from different
501          * architectures. For now, we only support objects as the
502          * native machine type.
503          */
504         switch (ehdr.e_machine) {
505         case EM_SPARC:
506             if (ehdr.e_ident[EI_CLASS] != ELFCLASS32 ||
507                 ehdr.e_ident[EI_DATA] != ELFDATA2MSB) {
508                 warn("invalid E_IDENT field for SPARC object");
509                 return;
510             }
511             g_flags |= DIS_SPARC_V8;
512             break;
513
514         case EM_SPARC32PLUS:
515             {
516                 uint64_t flags = ehdr.e_flags & EF_SPARC32PLUS_MASK;
517
518                 if (ehdr.e_ident[EI_CLASS] != ELFCLASS32 ||
519                     ehdr.e_ident[EI_DATA] != ELFDATA2MSB) {
520                     warn("invalid E_IDENT field for SPARC object");
521                     return;
522                 }
523             }
524         }
525     }

```

```

524         if (flags != 0 &&
525             (flags & (EF_SPARC_32PLUS | EF_SPARC_SUN_US1 |
526                 EF_SPARC_SUN_US3)) != EF_SPARC_32PLUS)
527             g_flags |= DIS_SPARC_V9 | DIS_SPARC_V9_SGI;
528         else
529             g_flags |= DIS_SPARC_V9;
530         break;
531     }
532
533     case EM_SPARCV9:
534         if (ehdr.e_ident[EI_CLASS] != ELFCLASS64 ||
535             ehdr.e_ident[EI_DATA] != ELFDATA2MSB) {
536             warn("invalid E_IDENT field for SPARC object");
537             return;
538         }
539
540         g_flags |= DIS_SPARC_V9 | DIS_SPARC_V9_SGI;
541         break;
542
543     case EM_386:
544         g_flags |= DIS_X86_SIZE32;
545         break;
546
547     case EM_AMD64:
548         g_flags |= DIS_X86_SIZE64;
549         break;
550
551     case EM_S370:
552         g_flags |= DIS_S370;
553
554         if (ehdr.e_ident[EI_CLASS] != ELFCLASS32 ||
555             ehdr.e_ident[EI_DATA] != ELFDATA2MSB) {
556             warn("invalid E_IDENT field for S370 object");
557             return;
558         }
559         break;
560
561     case EM_S390:
562         if (ehdr.e_ident[EI_CLASS] == ELFCLASS32) {
563             g_flags |= DIS_S390_31;
564         } else if (ehdr.e_ident[EI_CLASS] == ELFCLASS64) {
565             g_flags |= DIS_S390_64;
566         } else {
567             warn("invalid E_IDENT field for S390 object");
568             return;
569         }
570
571         if (ehdr.e_ident[EI_DATA] != ELFDATA2MSB) {
572             warn("invalid E_IDENT field for S390 object");
573             return;
574         }
575 #endif /* ! codereview */
576         break;
577
578     default:
579         die("%s: unsupported ELF machine 0x%x", filename,
580             ehdr.e_machine);
581 }
582
583 /*
584 * If ET_REL (.o), printing immediate symbols is likely to
585 * result in garbage, as symbol lookups on unrelocated
586 * immediates find false and useless matches.
587 */
588
589 if (ehdr.e_type == ET_REL)

```

```

590         g_flags |= DIS_NOIMMSYM;
591
592         if (!g_quiet && dis_tgt_member(current) != NULL)
593             (void) printf("\narchive member %s\n",
594                 dis_tgt_member(current));
595
596         /*
597          * Instantiate a libdisasm handle based on the file type.
598          */
599         if ((dhp = dis_handle_create(g_flags, current, do_lookup,
600             do_read)) == NULL)
601             die("%s: failed to initialize disassembler: %s",
602                 filename, dis_strerror(dis_errno()));
603
604         if (g_doall) {
605             /*
606              * With no arguments, iterate over all sections and
607              * disassemble only those that contain text.
608              */
609             dis_tgt_section_iter(current, dis_text_section, dhp);
610         } else {
611             callback_arg_t ca;
612
613             ca.ca_tgt = current;
614             ca.ca_handle = dhp;
615
616             /*
617              * If sections or functions were explicitly specified,
618              * resolve those names against the object, and iterate
619              * over just the resulting data.
620              */
621             sections = dis_namelist_resolve_sections(g_seclist,
622                 current);
623             functions = dis_namelist_resolve_functions(g_funclist,
624                 current);
625
626             dis_scnlist_iter(sections, dis_named_section, &ca);
627             dis_funclist_iter(functions, dis_named_function, &ca);
628
629             dis_scnlist_destroy(sections);
630             dis_funclist_destroy(functions);
631         }
632
633         dis_handle_destroy(dhp);
634     }
635
636     dis_tgt_destroy(tgt);
637 }
638
639 void
640 usage(void)
641 {
642     (void) fprintf(stderr, "usage: dis [-CVOqn] [-d sec] \n");
643     (void) fprintf(stderr, "\t[-D sec] [-F function] [-t sec] file ..\n");
644     exit(2);
645 }
646
647 typedef struct lib_node {
648     char *path;
649     struct lib_node *next;
650 } lib_node_t;
651
652 int
653 main(int argc, char **argv)
654 {
655     int optchar;

```

```

656     int i;
657     lib_node_t *libs = NULL;

659     g_funclist = dis_namelist_create();
660     g_seclist = dis_namelist_create();

662     while ((optchar = getopt(argc, argv, "Cd:D:F:l:Lot:Vqn")) != -1) {
663         switch (optchar) {
664             case 'C':
665                 g_demangle = 1;
666                 break;
667             case 'd':
668                 dis_namelist_add(g_seclist, optarg, DIS_DATA_RELATIVE);
669                 break;
670             case 'D':
671                 dis_namelist_add(g_seclist, optarg, DIS_DATA_ABSOLUTE);
672                 break;
673             case 'F':
674                 dis_namelist_add(g_funclist, optarg, 0);
675                 break;
676             case 'l': {
677                 /*
678                  * The '-l foo' option historically would attempt to
679                  * disassemble '$LIBDIR/libfoo.a'. The $LIBDIR
680                  * environment variable has never been supported or
681                  * documented for our linker. However, until this
682                  * option is formally EOLed, we have to support it.
683                  */
684                 char *dir;
685                 lib_node_t *node;
686                 size_t len;

688                 if ((dir = getenv("LIBDIR")) == NULL ||
689                     dir[0] == '\0')
690                     dir = "/usr/lib";
691                 node = safe_malloc(sizeof (lib_node_t));
692                 len = strlen(optarg) + strlen(dir) + sizeof ("/lib.a");
693                 node->path = safe_malloc(len);

695                 (void) snprintf(node->path, len, "%s/lib%s.a", dir,
696                                 optarg);
697                 node->next = libs;
698                 libs = node;
699                 break;
700             }
701             case 'L':
702                 /*
703                  * The '-L' option historically would attempt to read
704                  * the .debug section of the target to determine source
705                  * line information in order to annotate the output.
706                  * No compiler has emitted these sections in many years,
707                  * and the option has never done what it purported to
708                  * do. We silently consume the option for
709                  * compatibility.
710                  */
711                 break;
712             case 'n':
713                 g_numeric = 1;
714                 break;
715             case 'o':
716                 g_flags |= DIS_OCTAL;
717                 break;
718             case 'q':
719                 g_quiet = 1;
720                 break;
721             case 't':

```

```

722                 dis_namelist_add(g_seclist, optarg, DIS_TEXT);
723                 break;
724             case 'V':
725                 (void) printf("Solaris disassembler version 1.0\n");
726                 return (0);
727             default:
728                 usage();
729                 break;
730         }
731     }

733     argc -= optind;
734     argv += optind;

736     if (argc == 0 && libs == NULL) {
737         warn("no objects specified");
738         usage();
739     }

741     if (dis_namelist_empty(g_funclist) && dis_namelist_empty(g_seclist))
742         g_doall = 1;

744     /*
745      * See comment for 'l' option, above.
746      */
747     while (libs != NULL) {
748         lib_node_t *node = libs->next;

750         dis_file(libs->path);
751         free(libs->path);
752         free(libs);
753         libs = node;
754     }

756     for (i = 0; i < argc; i++)
757         dis_file(argv[i]);

759     dis_namelist_destroy(g_funclist);
760     dis_namelist_destroy(g_seclist);

762     return (g_error);
763 }

```

```

*****
4205 Thu Jul 23 08:02:30 2015
new/usr/src/lib/libdisasm/Makefile.com
6066 dis: support for System/370, System/390, and z/Architecture ELF bins
*****
1 #
2 # CDDL HEADER START
3 #
4 # The contents of this file are subject to the terms of the
5 # Common Development and Distribution License (the "License").
6 # You may not use this file except in compliance with the License.
7 #
8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9 # or http://www.opensolaris.org/os/licensing.
10 # See the License for the specific language governing permissions
11 # and limitations under the License.
12 #
13 # When distributing Covered Code, include this CDDL HEADER in each
14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 # If applicable, add the following below this CDDL HEADER, with the
16 # fields enclosed by brackets "[]" replaced with your own identifying
17 # information: Portions Copyright [yyyy] [name of copyright owner]
18 #
19 # CDDL HEADER END
20 #
21 #
22 # Copyright 2007 Sun Microsystems, Inc. All rights reserved.
23 # Use is subject to license terms.
24 # Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
25 # Copyright 2015 Nexenta Systems, Inc. All rights reserved.
26 #
27 #
28 #
29 # The build process for libdisasm is slightly different from that used by other
30 # libraries, because libdisasm must be built in two flavors - as a standalone
31 # for use by kmdb and as a normal library. We use $(CURTYPE) to indicate the
32 # current flavor being built.
33 #
34 #
35 LIBRARY=      libdisasm.a
36 STANDLIBRARY= libstanddisasm.so
37 VERS=        .1
38 #
39 # By default, we build the shared library. Construction of the standalone
40 # is specifically requested by architecture-specific Makefiles.
41 TYPES=       library
42 CURTYPE=     library
43 #
44 COMDIR=      $(SRC)/lib/libdisasm/common
45 #
46 #
47 # Architecture-independent files
48 #
49 SRCS_common= $(COMDIR)/libdisasm.c
50 OBJECTS_common= libdisasm.o
51 #
52 #
53 # Architecture-dependent disassembly files
54 #
55 SRCS_i386=   $(COMDIR)/dis_i386.c \
56             $(SRC)/common/dis/i386/dis_tables.c
57 SRCS_sparc=  $(COMDIR)/dis_sparc.c \
58             $(COMDIR)/dis_sparc_fmt.c \
59             $(COMDIR)/dis_sparc_instr.c
60 SRCS_s390x=  $(COMDIR)/dis_s390x.c
61 #endif /* ! codereview */

```

```

63 OBJECTS_i386= dis_i386.o \
64             dis_tables.o
65 OBJECTS_sparc= dis_sparc.o \
66             dis_sparc_fmt.o \
67             dis_sparc_instr.o
68 OBJECTS_s390x= dis_s390x.o
69 #endif /* ! codereview */
70 #
71 #
72 # We build the regular shared library with support for all architectures.
73 # The standalone version should only contain code for the native
74 # architecture to reduce the memory footprint of kmdb.
75 #
76 OBJECTS_library= $(OBJECTS_common) \
77                 $(OBJECTS_i386) \
78                 $(OBJECTS_sparc) \
79                 $(OBJECTS_s390x)
80 OBJECTS_standalone= $(OBJECTS_common) \
81                    $(OBJECTS_$(MACH))
82 OBJECTS=             $(OBJECTS_$(CURTYPE))
83 #
84 include $(SRC)/lib/Makefile.lib
85 #
86 SRCS_library=       $(SRCS_common) \
87                     $(SRCS_i386) \
88                     $(SRCS_sparc) \
89                     $(SRCS_s390x)
90 SRCS_standalone=    $(SRCS_common) \
91                     $(SRCS_$(MACH))
92 SRCS=               $(SRCS_$(CURTYPE))
93 #
94 #
95 # Used to verify that the standalone doesn't have any unexpected external
96 # dependencies.
97 #
98 LINKTEST_OBJ = objs/linktest_stand.o
99 #
100 CLOBBERFILES_standalone = $(LINKTEST_OBJ)
101 CLOBBERFILES += $(CLOBBERFILES_$(CURTYPE))
102 #
103 LIBS_standalone = $(STANDLIBRARY)
104 LIBS_library = $(DYNLIB) $(LINTLIB)
105 LIBS = $(LIBS_$(CURTYPE))
106 #
107 MAPFILES =          $(COMDIR)/mapfile-vers
108 #
109 LDLIBS +=           -lc
110 #
111 LDFLAGS_standalone = $(ZNOVERSION) $(BREDUCE) -dy -r
112 LDFLAGS = $(LDFLAGS_$(CURTYPE))
113 #
114 ASFLAGS_standalone = -DDIS_STANDALONE
115 ASFLAGS_library =
116 ASFLAGS += -P $(ASFLAGS_$(CURTYPE)) -D_ASM
117 #
118 $(LINTLIB) := SRCS = $(COMDIR)/$(LINTSRC)
119 #
120 CERRWARN +=        -gcc=-Wno-parentheses
121 CERRWARN +=        -gcc=-Wno-uninitialized
122 #
123 # We want the thread-specific errno in the library, but we don't want it in
124 # the standalone. $(DTS_ERRNO) is designed to add -D_TS_ERRNO to $(CPPFLAGS),
125 # in order to enable this feature. Conveniently, -D_REENTRANT does the same

```

```
126 # thing. As such, we null out $(DTS_ERRNO) to ensure that the standalone
127 # doesn't get it.
128 DTS_ERRNO=

130 CPPFLAGS_standalone = -DDIS_STANDALONE -I$(SRC)/cmd/mdb/common
131 CPPFLAGS_library = -D_REENTRANT
132 CPPFLAGS += -I$(COMDIR) $(CPPFLAGS_$(CURTYPE))

134 # For the x86 disassembler we have to include sources from usr/src/common
135 CPPFLAGS += -I$(SRC)/common/dis/i386 -DDIS_TEXT

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

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

144 C99MODE = $(C99_ENABLE)

146 DYNFLAGS += $(ZINTERPOSE)

148 .KEEP_STATE:
```



```

*****
79024 Thu Jul 23 08:02:30 2015
new/usr/src/lib/libdisasm/common/dis_s390x.c
6066 dis: support for System/370, System/390, and z/Architecture ELF bins
*****
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/byteorder.h>

21 #include "libdisasm_impl.h"

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

25 /*
26  * Throughout this file, the instruction format names based on:
27  * SA22-7832-09 z/Architecture Principles of Operation
28  *
29  * System/370, ESA/390, and earlier z/Architecture POP use slightly
30  * different names for the formats (the variant names are numeric). For the
31  * sake of simplicity, we use the most detailed definitions - z/Architecture.
32  *
33  * For ESA/390 we map the formats:
34  * E -> E
35  * I -> I
36  * RR -> RR
37  * RRE -> RRE
38  * RRF -> RRD & RRFa-e
39  * RX -> RXa-b
40  * RXE -> RXE
41  * RXF -> RXF
42  * RS -> RSa-b
43  * RSE -> RSYa-b
44  * RSL -> RSLa
45  * RSI -> RSI
46  * RI -> RIa-c
47  * RIL -> RILa-c
48  * SI -> SI
49  * S -> S
50  * SS -> SSa-b & SSd-e
51  * SSE -> SSE
52  *
53  * For System/370 we map the formats:
54  * RR -> RR
55  * RX -> RXa-b
56  * RS -> RSa-b
57  * SI -> SI
58  * S -> S
59  * SS -> SSa-c
60  *
61  * Disassembly begins in tbl_xx. The first byte of the instruction is used

```

```

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

100 /* BEGIN CSTYLED */
101 enum ifmt {
102     /* invalid */
103     IF_INVALID = 0,

105     /* indirection */
106     IF_TBL,
107     IF_MULTI,

109     /* 2-byte */
110     IF_ZERO, /* 370, 390, z */
111     IF_E, /* 390, z */
112     IF_I, /* 390, z */
113     IF_RR, /* 370, 390, z */

115     /* 4-byte */
116     IF_DIAG, /* 370, 390, z */
117     IF_IE, /* z */
118     IF_RIa, /* 390, z */
119     IF_RIb, /* 390, z */
120     IF_RIc, /* 390, z */
121     IF_RRD, /* 390, z */ /* on 390 these are RRF */
122     IF_RRE, /* 390, z */
123     IF_RRFa, /* 390, z */
124     IF_RRFb, /* 390, z */
125     IF_RRFc, /* 390, z */
126     IF_RRFd, /* 390, z */
127     IF_RRFe, /* 390, z */

```

```

128     IF_RSa,          /* 370, 390, z */
129     IF_Rsb,          /* 370, 390, z */
130     IF_RSI,          /*      390, z */
131     IF_RXa,          /* 370, 390, z */
132     IF_RXb,          /* 370, 390, z */
133     IF_S,            /* 370, 390, z */
134     IF_SI,           /* 370, 390, z */

136     /* 6-byte */
137     IF_MIL,          /*      z */
138     IF_RIEa,         /*      z */
139     IF_RIEb,         /*      z */
140     IF_RIEc,         /*      z */
141     IF_RIEd,         /*      z */
142     IF_RIEe,         /*      z */
143     IF_RIEf,         /*      z */
144     IF_RILa,         /*      390, z */
145     IF_RILb,         /*      390, z */
146     IF_RILc,         /*      390, z */
147     IF_RIS,          /*      z */
148     IF_RRS,          /*      z */
149     IF_RSLa,         /*      390, z */
150     IF_RSLb,         /*      z */
151     IF_RSYa,         /*      z */
152     IF_RSYb,         /*      z */
153     IF_RXE,          /*      390, z */
154     IF_RXF,          /*      390, z */
155     IF_RXYa,         /*      z */
156     IF_RXYb,         /*      z */
157     IF_SIL,          /*      z */
158     IF_SIY,          /*      z */
159     IF_SMI,          /*      z */
160     IF_SSa,          /* 370, 390, z */
161     IF_Ssb,          /* 370, 390, z */
162     IF_Ssc,          /* 370, 390, z */
163     IF_Ssd,          /*      390, z */
164     IF_Sse,          /*      390, z */
165     IF_Ssf,          /*      390, z */
166     IF_SSE,          /*      390, z */
167     IF_SSF,          /*      z */
168 };

170 #define IF_NFMTS      (IF_SSF + 1)

172 #define F_370          0x0001      /* 370 */
173 #define F_390          0x0002      /* 390 */
174 #define F_Z            0x0004      /*      z */
175 #define F_SIGNED_IMM  0x0010      /* 370, 390, z */
176 #define F_CTL_REG     0x0020      /* 370, 390, z */
177 #define F_HIDE_MASK   0x0040      /* 370, 390, z */
178 #define F_R1_IS_MASK  0x0080      /* 370, 390, z */
179 /* END CSTYLED */

181 struct inst_table {
182     union {
183         struct {
184             const char *name;
185             unsigned flags;
186         } inst;
187         struct {
188             const struct inst_table *ptr;
189             uint8_t off:4;
190             uint8_t shift:4;
191             uint8_t mask;
192         } table;
193     } struct {

```

```

194         const struct inst_table *ptr;
195     } multi;
196     } u;
197     enum ifmt fmt;
198 };

200 #define BITFLD(a, b)      DECL_BITFIELD2(b:4, a:4)

202 union inst {
203     uint8_t raw[6];
204     struct {
205         uint8_t op;
206         uint8_t par1;
207         uint16_t par2;
208     } diag;
209     struct {
210         uint8_t op;
211         uint8_t i;
212     } i;
213     struct {
214         uint16_t op;
215         uint8_t pad;
216         BITFLD(i1, i2);
217     } ie;
218     struct {
219         uint8_t op;
220         BITFLD(m1, ri2h);
221         uint8_t ri2l;
222         uint8_t ri3h;
223         uint16_t ri3l;
224     } mii;
225     struct {
226         uint8_t op;
227         BITFLD(r1, r2);
228     } rr;
229     struct {
230         uint16_t op;
231         BITFLD(r1, pad);
232         BITFLD(r3, r2);
233     } rrd;
234     struct {
235         uint16_t op;
236         uint8_t pad;
237         BITFLD(r1, r2);
238     } rre;
239     struct {
240         uint16_t op;
241         BITFLD(r1, m4);
242         BITFLD(r3, r2);
243     } rrf_ab;
244     struct {
245         uint16_t op;
246         BITFLD(m3, m4);
247         BITFLD(r1, r2);
248     } rrf_cde;
249     struct {
250         uint8_t opl;
251         BITFLD(r1, r2);
252         BITFLD(b4, d4h);
253         uint8_t d4l;
254         BITFLD(m3, pad);
255         uint8_t op2;
256     } rrs;
257     struct {
258         uint8_t op;
259         BITFLD(r1, x2);

```

```

260         BITFLD(b2, d2h);
261         uint8_t d2l;
262     } rx_a;
263     struct {
264         uint8_t op;
265         BITFLD(m1, x2);
266         BITFLD(b2, d2h);
267         uint8_t d2l;
268     } rx_b;
269     struct {
270         uint8_t op1;
271         BITFLD(r1, x2);
272         BITFLD(b2, d2h);
273         uint8_t d2l;
274         uint8_t pad;
275         uint8_t op2;
276     } rxe;
277     struct {
278         uint8_t op1;
279         BITFLD(r3, x2);
280         BITFLD(b2, d2h);
281         uint8_t d2l;
282         BITFLD(r1, pad);
283         uint8_t op2;
284     } rxf;
285     struct {
286         uint8_t op1;
287         BITFLD(r1, x2);
288         BITFLD(b2, dl2h);
289         uint8_t dl2l;
290         uint8_t dh2;
291         uint8_t op2;
292     } rxy_a;
293     struct {
294         uint8_t op1;
295         BITFLD(m1, x2);
296         BITFLD(b2, dl2h);
297         uint8_t dl2l;
298         uint8_t dh2;
299         uint8_t op2;
300     } rxy_b;
301     struct {
302         uint8_t op;
303         BITFLD(r1, r3);
304         BITFLD(b2, d2h);
305         uint8_t d2l;
306     } rs_a;
307     struct {
308         uint8_t op;
309         BITFLD(r1, m3);
310         BITFLD(b2, d2h);
311         uint8_t d2l;
312     } rs_b;
313     struct {
314         uint8_t op1;
315         BITFLD(l1, pad1);
316         BITFLD(b1, dlh);
317         uint8_t dl1;
318         uint8_t pad2;
319         uint8_t op2;
320     } rsl_a;
321     struct {
322         uint8_t op1;
323         uint8_t l2;
324         BITFLD(b2, d2h);
325         uint8_t d2l;

```

```

326         BITFLD(r1, m3);
327         uint8_t op2;
328     } rsl_b;
329     struct {
330         uint8_t op;
331         BITFLD(r1, r3);
332         uint16_t ri2;
333     } rsi;
334     struct {
335         uint8_t op1;
336         BITFLD(r1, r3);
337         BITFLD(b2, dl2h);
338         uint8_t dl2l;
339         uint8_t dh2;
340         uint8_t op2;
341     } rsy_a;
342     struct {
343         uint8_t op1;
344         BITFLD(r1, m3);
345         BITFLD(b2, dl2h);
346         uint8_t dl2l;
347         uint8_t dh2;
348         uint8_t op2;
349     } rsy_b;
350     struct {
351         uint8_t op1;
352         BITFLD(r1, op2);
353         uint16_t i2;
354     } ri_a;
355     struct {
356         uint8_t op1;
357         BITFLD(r1, op2);
358         uint16_t ri2;
359     } ri_b;
360     struct {
361         uint8_t op1;
362         BITFLD(m1, op2);
363         uint16_t ri2;
364     } ri_c;
365     struct {
366         uint8_t op1;
367         BITFLD(r1, _pad0);
368         uint16_t i2;
369         BITFLD(m3, _pad1);
370         uint8_t op2;
371     } rie_a;
372     struct {
373         uint8_t op1;
374         BITFLD(r1, r2);
375         uint16_t ri4;
376         BITFLD(m3, _pad);
377         uint8_t op2;
378     } rie_b;
379     struct {
380         uint8_t op1;
381         BITFLD(r1, m3);
382         uint16_t ri4;
383         uint8_t i2;
384         uint8_t op2;
385     } rie_c;
386     struct {
387         uint8_t op1;
388         BITFLD(r1, r3);
389         uint16_t i2;
390         uint8_t _pad;
391         uint8_t op2;

```

```

392     } rie_d;
393     struct {
394         uint8_t op1;
395         BITFLD(r1, r3);
396         uint16_t ri2;
397         uint8_t _pad;
398         uint8_t op2;
399     } rie_e;
400     struct {
401         uint8_t op1;
402         BITFLD(r1, r2);
403         uint8_t i3;
404         uint8_t i4;
405         uint8_t i5;
406         uint8_t op2;
407     } rie_f;
408     struct {
409         uint8_t op1;
410         BITFLD(r1, op2);
411         uint16_t i2h;
412         uint16_t i2l;
413     } ril_a;
414     struct {
415         uint8_t op1;
416         BITFLD(r1, op2);
417         uint16_t ri2h;
418         uint16_t ri2l;
419     } ril_b;
420     struct {
421         uint8_t op1;
422         BITFLD(m1, op2);
423         uint16_t ri2h;
424         uint16_t ri2l;
425     } ril_c;
426     struct {
427         uint8_t op1;
428         BITFLD(r1, m3);
429         BITFLD(b4, d4h);
430         uint8_t d4l;
431         uint8_t i2;
432         uint8_t op2;
433     } ris;
434     struct {
435         uint8_t op;
436         uint8_t i2;
437         BITFLD(b1, dlh);
438         uint8_t dll;
439     } si;
440     struct {
441         uint16_t op;
442         BITFLD(b1, dlh);
443         uint8_t dll;
444         uint16_t i2;
445     } sil;
446     struct {
447         uint8_t op1;
448         uint8_t i2;
449         BITFLD(b1, dllh);
450         uint8_t dll1;
451         uint8_t dhl;
452         uint8_t op2;
453     } siy;
454     struct {
455         uint8_t op;
456         BITFLD(m1, pad);
457         BITFLD(b3, d3h);

```

```

458         uint8_t d3l;
459         uint16_t ri2;
460     } smi;
461     struct {
462         uint8_t op1;
463         uint8_t op2;
464         BITFLD(b2, d2h);
465         uint8_t d2l;
466     } s;
467     struct {
468         uint8_t op;
469         uint8_t l;
470         BITFLD(b1, dlh);
471         uint8_t dll;
472         BITFLD(b2, d2h);
473         uint8_t d2l;
474     } ss_a;
475     struct {
476         uint8_t op;
477         BITFLD(l1, l2);
478         BITFLD(b1, dlh);
479         uint8_t dll;
480         BITFLD(b2, d2h);
481         uint8_t d2l;
482     } ss_b;
483     struct {
484         uint8_t op;
485         BITFLD(l1, i3);
486         BITFLD(b1, dlh);
487         uint8_t dll;
488         BITFLD(b2, d2h);
489         uint8_t d2l;
490     } ss_c;
491     struct {
492         uint8_t op;
493         BITFLD(r1, r3);
494         BITFLD(b1, dlh);
495         uint8_t dll;
496         BITFLD(b2, d2h);
497         uint8_t d2l;
498     } ss_d;
499     struct {
500         uint8_t op;
501         BITFLD(r1, r3);
502         BITFLD(b2, d2h);
503         uint8_t d2l;
504         BITFLD(b4, d4h);
505         uint8_t d4l;
506     } ss_e;
507     struct {
508         uint8_t op;
509         uint8_t l2;
510         BITFLD(b1, dlh);
511         uint8_t dll;
512         BITFLD(b2, d2h);
513         uint8_t d2l;
514     } ss_f;
515     struct {
516         uint16_t op;
517         BITFLD(b1, dlh);
518         uint8_t dll;
519         BITFLD(b2, d2h);
520         uint8_t d2l;
521     } sse;
522     struct {
523         uint8_t op1;

```

```

524         BITFLD(r3, op2);
525         BITFLD(b1, dlh);
526         uint8_t dll;
527         BITFLD(b2, d2h);
528         uint8_t d2l;
529     } ssf;
530 };

532 #define INSTR(op, m, fm, fl)    [op] = { \
533     .u.inst = { \
534         .name = (m), \
535         .flags = (fl), \
536     }, \
537     .fmt = (fm), \
538 }
539 #define TABLE(op, tbl, o, s, m) [op] = { \
540     .u.table = { \
541         .ptr = (tbl), \
542         .off = (o), \
543         .shift = (s), \
544         .mask = (m), \
545     }, \
546     .fmt = IF_TBL, \
547 }
548 #define MULTI(op, tbl)        [op] = { \
549     .u.multi.ptr = (tbl), \
550     .fmt = IF_MULTI, \
551 }

553 /*
554  * Instruction tables based on:
555  * GA22-7000-4   System/370 Principles of Operation
556  * SA22-7201-08 ESA/390 Principles of Operation
557  * SA22-7832-09 z/Architecture Principles of Operation
558  */

560 /* BEGIN CSTYLED */
561 static const struct inst_table tbl_01xx[256] = {
562     INSTR(0x01, "pr", IF_E, F_390 | F_Z),
563     INSTR(0x02, "upt", IF_E, F_390 | F_Z),
564     INSTR(0x04, "ptff", IF_E, F_Z),
565     INSTR(0x07, "sckpf", IF_E, F_390 | F_Z),
566     INSTR(0x0a, "pfpo", IF_E, F_Z),
567     INSTR(0x0b, "tam", IF_E, F_390 | F_Z),
568     INSTR(0x0c, "sam24", IF_E, F_390 | F_Z),
569     INSTR(0x0d, "sam31", IF_E, F_390 | F_Z),
570     INSTR(0x0e, "sam64", IF_E, F_Z),
571     INSTR(0xff, "trap2", IF_E, F_390 | F_Z),
572 };

574 static const struct inst_table tbl_07[] = {
575     INSTR(0x0, "nopr", IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
576     INSTR(0x1, "bor", IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
577     INSTR(0x2, "bhr", IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
578     INSTR(0x3, "bcr", IF_RR, F_370 | F_390 | F_Z | F_R1_IS_MASK),
579     INSTR(0x4, "blr", IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
580     INSTR(0x5, "bcr", IF_RR, F_370 | F_390 | F_Z | F_R1_IS_MASK),
581     INSTR(0x6, "bcr", IF_RR, F_370 | F_390 | F_Z | F_R1_IS_MASK),
582     INSTR(0x7, "bnzr", IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
583     INSTR(0x8, "ber", IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
584     INSTR(0x9, "bcr", IF_RR, F_370 | F_390 | F_Z | F_R1_IS_MASK),
585     INSTR(0xa, "bcr", IF_RR, F_370 | F_390 | F_Z | F_R1_IS_MASK),
586     INSTR(0xb, "bner", IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
587     INSTR(0xc, "bcr", IF_RR, F_370 | F_390 | F_Z | F_R1_IS_MASK),
588     INSTR(0xd, "bnhr", IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
589     INSTR(0xe, "bnor", IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),

```

```

590     INSTR(0xf, "br", IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
591 };

593 static const struct inst_table tbl_47[] = {
594     INSTR(0x0, "nop", IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
595     INSTR(0x1, "bo", IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
596     INSTR(0x2, "bh", IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
597     INSTR(0x3, "bc", IF_RXb, F_370 | F_390 | F_Z),
598     INSTR(0x4, "bl", IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
599     INSTR(0x5, "bc", IF_RXb, F_370 | F_390 | F_Z),
600     INSTR(0x6, "bc", IF_RXb, F_370 | F_390 | F_Z),
601     INSTR(0x7, "bne", IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
602     INSTR(0x8, "be", IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
603     INSTR(0x9, "bc", IF_RXb, F_370 | F_390 | F_Z),
604     INSTR(0xa, "bc", IF_RXb, F_370 | F_390 | F_Z),
605     INSTR(0xb, "bnl", IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
606     INSTR(0xc, "bc", IF_RXb, F_370 | F_390 | F_Z),
607     INSTR(0xd, "bnh", IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
608     INSTR(0xe, "bno", IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
609     INSTR(0xf, "b", IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
610 };

612 /* the preferred mnemonic changed over time */
613 static const struct inst_table tbl_25[] = {
614     INSTR(F_370, "lrd", IF_RR, F_370),
615     INSTR(F_390, "ldxr", IF_RR, F_390),
616     INSTR(F_Z, "ldxr", IF_RR, F_Z),
617 };

619 /* the preferred mnemonic changed over time */
620 static const struct inst_table tbl_35[] = {
621     INSTR(F_370, "lrr", IF_RR, F_370),
622     INSTR(F_390, "ledr", IF_RR, F_390),
623     INSTR(F_Z, "ledr", IF_RR, F_Z),
624 };

626 /* the preferred mnemonic changed over time */
627 static const struct inst_table tbl_3c[] = {
628     INSTR(F_370, "mer", IF_RR, F_370),
629     INSTR(F_390, "mder", IF_RR, F_390),
630     INSTR(F_Z, "mder", IF_RR, F_Z),
631 };

633 /* the preferred mnemonic changed over time */
634 static const struct inst_table tbl_7c[] = {
635     INSTR(F_370, "me", IF_RXa, F_370),
636     INSTR(F_390, "mde", IF_RXa, F_390),
637     INSTR(F_Z, "mde", IF_RXa, F_Z),
638 };

640 /* the meaning of this instruction changed over time */
641 static const struct inst_table tbl_84[] = {
642     INSTR(F_370, "wr", IF_SI, F_370),
643     INSTR(F_390, "brxh", IF_RSI, F_390),
644     INSTR(F_Z, "brxh", IF_RSI, F_Z),
645 };

647 /* the meaning of this instruction changed over time */
648 static const struct inst_table tbl_85[] = {
649     INSTR(F_370, "rdd", IF_SI, F_370),
650     INSTR(F_390, "brxle", IF_RSI, F_390),
651     INSTR(F_Z, "brxle", IF_RSI, F_Z),
652 };

654 static const struct inst_table tbl_a5x[16] = {
655     INSTR(0x0, "iihh", IF_RIa, F_Z),

```

```

656 INSTR(0x1, "iihl", IF_RIa, F_Z),
657 INSTR(0x2, "iih", IF_RIa, F_Z),
658 INSTR(0x3, "iill", IF_RIa, F_Z),
659 INSTR(0x4, "nihh", IF_RIa, F_Z),
660 INSTR(0x5, "nihl", IF_RIa, F_Z),
661 INSTR(0x6, "nilh", IF_RIa, F_Z),
662 INSTR(0x7, "niil", IF_RIa, F_Z),
663 INSTR(0x8, "oihh", IF_RIa, F_Z),
664 INSTR(0x9, "oihl", IF_RIa, F_Z),
665 INSTR(0xa, "oilh", IF_RIa, F_Z),
666 INSTR(0xb, "oill", IF_RIa, F_Z),
667 INSTR(0xc, "llihh", IF_RIa, F_Z),
668 INSTR(0xd, "llihl", IF_RIa, F_Z),
669 INSTR(0xe, "llilh", IF_RIa, F_Z),
670 INSTR(0xf, "llill", IF_RIa, F_Z),
671 };

673 /* the preferred mnemonic changed over time */
674 static const struct inst_table tbl_a70[] = {
675 INSTR(F_390, "tmh", IF_RIa, F_390),
676 INSTR(F_Z, "tmlh", IF_RIa, F_Z),
677 };

679 /* the preferred mnemonic changed over time */
680 static const struct inst_table tbl_a71[] = {
681 INSTR(F_390, "tml", IF_RIa, F_390),
682 INSTR(F_Z, "tmll", IF_RIa, F_Z),
683 };

685 static const struct inst_table tbl_a74[16] = {
686 INSTR(0x0, "jno", IF_RIc, F_390 | F_HIDE_MASK),
687 INSTR(0x1, "jo", IF_RIc, F_390 | F_HIDE_MASK),
688 INSTR(0x2, "jh", IF_RIc, F_390 | F_HIDE_MASK),
689 INSTR(0x3, "brc", IF_RIc, F_390 | F_Z),
690 INSTR(0x4, "jl", IF_RIc, F_390 | F_HIDE_MASK),
691 INSTR(0x5, "brc", IF_RIc, F_390 | F_Z),
692 INSTR(0x6, "brc", IF_RIc, F_390 | F_Z),
693 INSTR(0x7, "jne", IF_RIc, F_390 | F_HIDE_MASK),
694 INSTR(0x8, "je", IF_RIc, F_390 | F_HIDE_MASK),
695 INSTR(0x9, "brc", IF_RIc, F_390 | F_Z),
696 INSTR(0xa, "brc", IF_RIc, F_390 | F_Z),
697 INSTR(0xb, "jnl", IF_RIc, F_390 | F_HIDE_MASK),
698 INSTR(0xc, "brc", IF_RIc, F_390 | F_Z),
699 INSTR(0xd, "jnh", IF_RIc, F_390 | F_HIDE_MASK),
700 INSTR(0xe, "jno", IF_RIc, F_390 | F_Z),
701 INSTR(0xf, "j", IF_RIc, F_390 | F_HIDE_MASK),
702 };

704 static const struct inst_table tbl_a7x[16] = {
705 MULTI(0x0, tbl_a70),
706 MULTI(0x1, tbl_a71),
707 INSTR(0x2, "tmhh", IF_RIa, F_Z),
708 INSTR(0x3, "tmhl", IF_RIa, F_Z),
709 TABLE(0x4, tbl_a74, 1, 4, 0x0f),
710 INSTR(0x5, "bras", IF_RIb, F_390 | F_Z),
711 INSTR(0x6, "brct", IF_RIb, F_390 | F_Z),
712 INSTR(0x7, "brctg", IF_RIb, F_Z),
713 INSTR(0x8, "lhi", IF_RIa, F_390 | F_Z),
714 INSTR(0x9, "lghi", IF_RIa, F_Z),
715 INSTR(0xa, "ahi", IF_RIa, F_390 | F_Z | F_SIGNED_IMM),
716 INSTR(0xb, "aghi", IF_RIa, F_Z | F_SIGNED_IMM),
717 INSTR(0xc, "mhi", IF_RIa, F_390 | F_Z),
718 INSTR(0xd, "mghi", IF_RIa, F_Z),
719 INSTR(0xe, "chi", IF_RIa, F_390 | F_Z | F_SIGNED_IMM),
720 INSTR(0xf, "cghi", IF_RIa, F_Z | F_SIGNED_IMM),
721 };

```

```

723 static const struct inst_table tbl_b2a6[] = {
724 INSTR(F_390, "cuutf", IF_RRFc, F_390),
725 INSTR(F_Z, "c2l", IF_RRFc, F_Z),
726 };

728 static const struct inst_table tbl_b2a7[] = {
729 INSTR(F_390, "cutfu", IF_RRFc, F_390),
730 INSTR(F_Z, "cul2", IF_RRFc, F_Z),
731 };

733 static const struct inst_table tbl_b2xx[256] = {
734 INSTR(0x02, "stidp", IF_S, F_370 | F_390 | F_Z),
735 INSTR(0x04, "sck", IF_S, F_370 | F_390 | F_Z),
736 INSTR(0x05, "stck", IF_S, F_370 | F_390 | F_Z),
737 INSTR(0x06, "sckc", IF_S, F_370 | F_390 | F_Z),
738 INSTR(0x07, "stckc", IF_S, F_370 | F_390 | F_Z),
739 INSTR(0x08, "spt", IF_S, F_370 | F_390 | F_Z),
740 INSTR(0x09, "stpt", IF_S, F_370 | F_390 | F_Z),
741 INSTR(0x0a, "spka", IF_S, F_370 | F_390 | F_Z),
742 INSTR(0x0b, "ipk", IF_S, F_370 | F_390 | F_Z),
743 INSTR(0x0d, "ptlb", IF_S, F_370 | F_390 | F_Z),
744 INSTR(0x10, "spx", IF_S, F_370 | F_390 | F_Z),
745 INSTR(0x11, "stpx", IF_S, F_370 | F_390 | F_Z),
746 INSTR(0x12, "stap", IF_S, F_370 | F_390 | F_Z),
747 INSTR(0x13, "rrb", IF_S, F_370 | F_390 | F_Z),
748 INSTR(0x14, "sie", IF_S, F_390 | F_Z),
749 INSTR(0x18, "pc", IF_S, F_390 | F_Z),
750 INSTR(0x19, "sac", IF_S, F_390 | F_Z),
751 INSTR(0x1a, "cfc", IF_S, F_390 | F_Z),
752 INSTR(0x21, "ipte", IF_RRE, F_390 | F_Z),
753 INSTR(0x22, "ipm", IF_RRE, F_390 | F_Z),
754 INSTR(0x23, "ivsk", IF_RRE, F_390 | F_Z),
755 INSTR(0x24, "iac", IF_RRE, F_390 | F_Z),
756 INSTR(0x25, "ssar", IF_RRE, F_390 | F_Z),
757 INSTR(0x26, "epar", IF_RRE, F_390 | F_Z),
758 INSTR(0x27, "esar", IF_RRE, F_390 | F_Z),
759 INSTR(0x28, "pt", IF_RRE, F_390 | F_Z),
760 INSTR(0x29, "iske", IF_RRE, F_390 | F_Z),
761 INSTR(0x2a, "rrbe", IF_RRE, F_390 | F_Z),
762 INSTR(0x2b, "sske", IF_RRFc, F_390 | F_Z),
763 INSTR(0x2c, "tb", IF_RRE, F_390 | F_Z),
764 INSTR(0x2d, "dxr", IF_RRE, F_390 | F_Z),
765 INSTR(0x2e, "pgin", IF_RRE, F_390 | F_Z),
766 INSTR(0x2f, "pgout", IF_RRE, F_390 | F_Z),
767 INSTR(0x30, "csch", IF_S, F_Z),
768 INSTR(0x31, "hsch", IF_S, F_Z),
769 INSTR(0x32, "msch", IF_S, F_Z),
770 INSTR(0x33, "ssch", IF_S, F_Z),
771 INSTR(0x34, "stsch", IF_S, F_Z),
772 INSTR(0x35, "tsch", IF_S, F_Z),
773 INSTR(0x36, "tpi", IF_S, F_Z),
774 INSTR(0x37, "sal", IF_S, F_Z),
775 INSTR(0x38, "rsch", IF_S, F_Z),
776 INSTR(0x39, "stcrw", IF_S, F_Z),
777 INSTR(0x3a, "stcps", IF_S, F_Z),
778 INSTR(0x3b, "rchp", IF_S, F_Z),
779 INSTR(0x3d, "schm", IF_S, F_Z),
780 INSTR(0x40, "bakr", IF_RRE, F_390 | F_Z),
781 INSTR(0x41, "cksm", IF_RRE, F_390 | F_Z),
782 INSTR(0x44, "sqdr", IF_RRE, F_390 | F_Z),
783 INSTR(0x45, "sqer", IF_RRE, F_390 | F_Z),
784 INSTR(0x46, "stura", IF_RRE, F_390 | F_Z),
785 INSTR(0x47, "msta", IF_RRE, F_390 | F_Z),
786 INSTR(0x48, "palb", IF_RRE, F_390 | F_Z),
787 INSTR(0x49, "ereg", IF_RRE, F_390 | F_Z),

```

```

788 INSTR(0x4a, "esta", IF_RRE, F_390 | F_Z),
789 INSTR(0x4b, "lura", IF_RRE, F_390 | F_Z),
790 INSTR(0x4c, "tar", IF_RRE, F_390 | F_Z),
791 INSTR(0x4d, "cpya", IF_RRE, F_390 | F_Z),
792 INSTR(0x4e, "sar", IF_RRE, F_390 | F_Z),
793 INSTR(0x4f, "ear", IF_RRE, F_390 | F_Z),
794 INSTR(0x50, "csp", IF_RRE, F_390 | F_Z),
795 INSTR(0x52, "msr", IF_RRE, F_390 | F_Z),
796 INSTR(0x54, "mvpg", IF_RRE, F_390 | F_Z),
797 INSTR(0x55, "mvst", IF_RRE, F_390 | F_Z),
798 INSTR(0x57, "cuse", IF_RRE, F_390 | F_Z),
799 INSTR(0x58, "bsg", IF_RRE, F_390 | F_Z),
800 INSTR(0x5a, "bsa", IF_RRE, F_390 | F_Z),
801 INSTR(0x5d, "clst", IF_RRE, F_390 | F_Z),
802 INSTR(0x5e, "srst", IF_RRE, F_390 | F_Z),
803 INSTR(0x63, "cmpsc", IF_RRE, F_Z),
804 INSTR(0x76, "xsch", IF_S, F_Z),
805 INSTR(0x77, "rp", IF_S, F_390 | F_Z),
806 INSTR(0x78, "stcke", IF_S, F_390 | F_Z),
807 INSTR(0x79, "sacf", IF_S, F_390 | F_Z),
808 INSTR(0x7c, "stckf", IF_S, F_Z),
809 INSTR(0x7d, "stsi", IF_S, F_390 | F_Z),
810 INSTR(0x99, "srnm", IF_S, F_390 | F_Z),
811 INSTR(0x9c, "stfpc", IF_S, F_390 | F_Z),
812 INSTR(0x9d, "lfpc", IF_S, F_390 | F_Z),
813 INSTR(0xa5, "tre", IF_RRE, F_390 | F_Z),
814 MULTI(0xa6, tbl_b2a6),
815 MULTI(0xa7, tbl_b2a7),
816 INSTR(0xb0, "stfle", IF_S, F_Z),
817 INSTR(0xb1, "stfl", IF_S, F_390 | F_Z),
818 INSTR(0xb2, "lpswe", IF_S, F_Z),
819 INSTR(0xb8, "srnmb", IF_S, F_Z),
820 INSTR(0xb9, "srnmt", IF_S, F_Z),
821 INSTR(0xbd, "lfas", IF_S, F_Z),
822 INSTR(0xe8, "ppa", IF_RRFc, F_Z),
823 INSTR(0xec, "etnd", IF_RRE, F_Z),
824 INSTR(0xf8, "tend", IF_S, F_Z),
825 INSTR(0xfa, "niai", IF_IE, F_Z),
826 INSTR(0xfc, "tabort", IF_S, F_Z),
827 INSTR(0xff, "trap4", IF_S, F_390 | F_Z),
828 };

830 static const struct inst_table tbl_b344[] = {
831 INSTR(F_390, "ledbr", IF_RRE, F_390),
832 INSTR(F_Z, "ledbra", IF_RRFc, F_Z),
833 };

835 static const struct inst_table tbl_b345[] = {
836 INSTR(F_390, "ldxbr", IF_RRE, F_390),
837 INSTR(F_Z, "ldxbra", IF_RRFc, F_Z),
838 };

840 static const struct inst_table tbl_b346[] = {
841 INSTR(F_390, "lexbr", IF_RRE, F_390),
842 INSTR(F_Z, "lexbra", IF_RRFc, F_Z),
843 };

845 static const struct inst_table tbl_b347[] = {
846 INSTR(F_390, "fixbr", IF_RRFc, F_390),
847 INSTR(F_Z, "fixbra", IF_RRFc, F_Z),
848 };

850 static const struct inst_table tbl_b357[] = {
851 INSTR(F_390, "fiebr", IF_RRFc, F_390),
852 INSTR(F_Z, "fiebra", IF_RRFc, F_Z),
853 };

```

```

855 static const struct inst_table tbl_b35f[] = {
856 INSTR(F_390, "fidbr", IF_RRFc, F_390),
857 INSTR(F_Z, "fidbra", IF_RRFc, F_Z),
858 };

860 static const struct inst_table tbl_b394[] = {
861 INSTR(F_390, "cefbr", IF_RRE, F_390),
862 INSTR(F_Z, "cefbra", IF_RRFc, F_Z),
863 };

865 static const struct inst_table tbl_b395[] = {
866 INSTR(F_390, "cdfbr", IF_RRE, F_390),
867 INSTR(F_Z, "cdfbra", IF_RRFc, F_Z),
868 };

870 static const struct inst_table tbl_b396[] = {
871 INSTR(F_390, "cxfbr", IF_RRE, F_390),
872 INSTR(F_Z, "cxfbra", IF_RRFc, F_Z),
873 };

875 static const struct inst_table tbl_b398[] = {
876 INSTR(F_390, "cfibr", IF_RRFc, F_390),
877 INSTR(F_Z, "cfibra", IF_RRFc, F_Z),
878 };

880 static const struct inst_table tbl_b399[] = {
881 INSTR(F_390, "cfdbr", IF_RRFc, F_390),
882 INSTR(F_Z, "cfdbra", IF_RRFc, F_Z),
883 };

885 static const struct inst_table tbl_b39a[] = {
886 INSTR(F_390, "cfxbr", IF_RRFc, F_390),
887 INSTR(F_Z, "cfxbra", IF_RRFc, F_Z),
888 };

890 static const struct inst_table tbl_b3xx[256] = {
891 INSTR(0x00, "lpebr", IF_RRE, F_390 | F_Z),
892 INSTR(0x01, "lnebr", IF_RRE, F_390 | F_Z),
893 INSTR(0x02, "ltebr", IF_RRE, F_390 | F_Z),
894 INSTR(0x03, "lcebr", IF_RRE, F_390 | F_Z),
895 INSTR(0x04, "ldebr", IF_RRE, F_390 | F_Z),
896 INSTR(0x05, "lxdbr", IF_RRE, F_390 | F_Z),
897 INSTR(0x06, "lxebr", IF_RRE, F_390 | F_Z),
898 INSTR(0x07, "mxdbr", IF_RRE, F_390 | F_Z),
899 INSTR(0x08, "kebr", IF_RRE, F_390 | F_Z),
900 INSTR(0x09, "cebr", IF_RRE, F_390 | F_Z),
901 INSTR(0x0a, "aibr", IF_RRE, F_390 | F_Z),
902 INSTR(0x0b, "sebr", IF_RRE, F_390 | F_Z),
903 INSTR(0x0c, "mdebr", IF_RRE, F_390 | F_Z),
904 INSTR(0x0d, "debr", IF_RRE, F_390 | F_Z),
905 INSTR(0x0e, "maibr", IF_RRD, F_390 | F_Z),
906 INSTR(0x0f, "msebr", IF_RRD, F_390 | F_Z),
907 INSTR(0x10, "lpdbr", IF_RRE, F_390 | F_Z),
908 INSTR(0x11, "lndbr", IF_RRE, F_390 | F_Z),
909 INSTR(0x12, "ltdbr", IF_RRE, F_390 | F_Z),
910 INSTR(0x13, "lcdbr", IF_RRE, F_390 | F_Z),
911 INSTR(0x14, "sqebr", IF_RRE, F_390 | F_Z),
912 INSTR(0x15, "sqdbr", IF_RRE, F_390 | F_Z),
913 INSTR(0x16, "sqcbr", IF_RRE, F_390 | F_Z),
914 INSTR(0x17, "meebr", IF_RRE, F_390 | F_Z),
915 INSTR(0x18, "kdbr", IF_RRE, F_390 | F_Z),
916 INSTR(0x19, "cdbr", IF_RRE, F_390 | F_Z),
917 INSTR(0x1a, "adbr", IF_RRE, F_390 | F_Z),
918 INSTR(0x1b, "sdbr", IF_RRE, F_390 | F_Z),
919 INSTR(0x1c, "mdbr", IF_RRE, F_390 | F_Z),

```

```

920 INSTR(0x1d, "ddbr", IF_RRE, F_390) F_Z),
921 INSTR(0x1e, "madbr", IF_RRD, F_390) F_Z),
922 INSTR(0x1f, "msdbr", IF_RRD, F_390) F_Z),
923 INSTR(0x24, "lder", IF_RRE, F_390) F_Z),
924 INSTR(0x25, "lxd", IF_RRE, F_390) F_Z),
925 INSTR(0x26, "lxer", IF_RRE, F_390) F_Z),
926 INSTR(0x2e, "maer", IF_RRD, F_390) F_Z),
927 INSTR(0x2f, "mser", IF_RRD, F_390) F_Z),
928 INSTR(0x36, "sqxr", IF_RRE, F_390) F_Z),
929 INSTR(0x37, "meer", IF_RRE, F_390) F_Z),
930 INSTR(0x38, "maylr", IF_RRD, F_Z),
931 INSTR(0x39, "mylr", IF_RRD, F_Z),
932 INSTR(0x3a, "mayr", IF_RRD, F_Z),
933 INSTR(0x3b, "myr", IF_RRD, F_Z),
934 INSTR(0x3c, "mayhr", IF_RRD, F_Z),
935 INSTR(0x3d, "myhr", IF_RRD, F_Z),
936 INSTR(0x3e, "madr", IF_RRD, F_390) F_Z),
937 INSTR(0x3f, "msdr", IF_RRD, F_390) F_Z),
938 INSTR(0x40, "lpxbr", IF_RRE, F_390) F_Z),
939 INSTR(0x41, "lnxbr", IF_RRE, F_390) F_Z),
940 INSTR(0x42, "ltxbr", IF_RRE, F_390) F_Z),
941 INSTR(0x43, "lctxbr", IF_RRE, F_390) F_Z),
942 MULTI(0x44, tbl_b344),
943 MULTI(0x45, tbl_b345),
944 MULTI(0x46, tbl_b346),
945 MULTI(0x47, tbl_b347),
946 INSTR(0x48, "kxbr", IF_RRE, F_390) F_Z),
947 INSTR(0x49, "cxbr", IF_RRE, F_390) F_Z),
948 INSTR(0x4a, "axbr", IF_RRE, F_390) F_Z),
949 INSTR(0x4b, "sxbr", IF_RRE, F_390) F_Z),
950 INSTR(0x4c, "mxbr", IF_RRE, F_390) F_Z),
951 INSTR(0x4d, "dxbr", IF_RRE, F_390) F_Z),
952 INSTR(0x50, "tbedr", IF_RRFe, F_390) F_Z),
953 INSTR(0x51, "tbd", IF_RRFe, F_390) F_Z),
954 INSTR(0x53, "diebr", IF_RRFb, F_390) F_Z),
955 MULTI(0x57, tbl_b357),
956 INSTR(0x58, "thder", IF_RRE, F_390) F_Z),
957 INSTR(0x59, "thdr", IF_RRE, F_390) F_Z),
958 INSTR(0x5b, "didbr", IF_RRFe, F_390) F_Z),
959 MULTI(0x5f, tbl_b35f),
960 INSTR(0x60, "lpxr", IF_RRE, F_390) F_Z),
961 INSTR(0x61, "lnxr", IF_RRE, F_390) F_Z),
962 INSTR(0x62, "ltxr", IF_RRE, F_390) F_Z),
963 INSTR(0x63, "lctx", IF_RRE, F_390) F_Z),
964 INSTR(0x65, "lxx", IF_RRE, F_390) F_Z),
965 INSTR(0x66, "lexr", IF_RRE, F_390) F_Z),
966 INSTR(0x67, "fixr", IF_RRE, F_390) F_Z),
967 INSTR(0x69, "cxr", IF_RRE, F_390) F_Z),
968 INSTR(0x70, "lpdfr", IF_RRE, F_Z),
969 INSTR(0x71, "lndfr", IF_RRE, F_Z),
970 INSTR(0x72, "cpsdr", IF_RRFe, F_Z),
971 INSTR(0x73, "lcdfr", IF_RRE, F_Z),
972 INSTR(0x74, "lzer", IF_RRE, F_390) F_Z),
973 INSTR(0x75, "lzdr", IF_RRE, F_390) F_Z),
974 INSTR(0x76, "lzxr", IF_RRE, F_390) F_Z),
975 INSTR(0x77, "fier", IF_RRE, F_390) F_Z),
976 INSTR(0x7f, "fidr", IF_RRE, F_390) F_Z),
977 INSTR(0x84, "sfpc", IF_RRE, F_390) F_Z),
978 INSTR(0x85, "sfasr", IF_RRE, F_Z),
979 INSTR(0x8c, "efpc", IF_RRE, F_390) F_Z),
980 INSTR(0x90, "celfbr", IF_RRFe, F_Z),
981 INSTR(0x91, "cdlfbr", IF_RRFe, F_Z),
982 INSTR(0x92, "cxlfr", IF_RRFe, F_Z),
983 MULTI(0x94, tbl_b394),
984 MULTI(0x95, tbl_b395),
985 MULTI(0x96, tbl_b396),

```

```

986 MULTI(0x98, tbl_b398),
987 MULTI(0x99, tbl_b399),
988 MULTI(0x9a, tbl_b39a),
989 INSTR(0x9c, "clfebr", IF_RRFe, F_Z),
990 INSTR(0x9d, "clfdbr", IF_RRFe, F_Z),
991 INSTR(0x9e, "clfxbr", IF_RRFe, F_Z),
992 INSTR(0xa0, "celgbr", IF_RRFe, F_Z),
993 INSTR(0xa1, "cdlgr", IF_RRFe, F_Z),
994 INSTR(0xa2, "cxlgr", IF_RRFe, F_Z),
995 INSTR(0xa4, "cegbra", IF_RRFe, F_Z),
996 INSTR(0xa5, "cdgbra", IF_RRFe, F_Z),
997 INSTR(0xa6, "cxgbra", IF_RRFe, F_Z),
998 INSTR(0xa8, "cgebra", IF_RRFe, F_Z),
999 INSTR(0xa9, "cggbra", IF_RRFe, F_Z),
1000 INSTR(0xaa, "cggbra", IF_RRFe, F_Z),
1001 INSTR(0xac, "clgebr", IF_RRFe, F_Z),
1002 INSTR(0xad, "clgdr", IF_RRFe, F_Z),
1003 INSTR(0xae, "clgxr", IF_RRFe, F_Z),
1004 INSTR(0xb4, "cefr", IF_RRE, F_390) F_Z),
1005 INSTR(0xb5, "cdf", IF_RRE, F_390) F_Z),
1006 INSTR(0xb6, "cxfr", IF_RRE, F_390) F_Z),
1007 INSTR(0xb8, "cfer", IF_RRFe, F_390) F_Z),
1008 INSTR(0xb9, "cdf", IF_RRFe, F_390) F_Z),
1009 INSTR(0xba, "cfxr", IF_RRFe, F_390) F_Z),
1010 INSTR(0xc1, "ldgr", IF_RRE, F_Z),
1011 INSTR(0xc4, "cegr", IF_RRE, F_Z),
1012 INSTR(0xc5, "cdgr", IF_RRE, F_Z),
1013 INSTR(0xc6, "cxgr", IF_RRE, F_Z),
1014 INSTR(0xc8, "cger", IF_RRFe, F_Z),
1015 INSTR(0xc9, "cgdr", IF_RRFe, F_Z),
1016 INSTR(0xca, "cgxr", IF_RRFe, F_Z),
1017 INSTR(0xcd, "lgdr", IF_RRE, F_Z),
1018 INSTR(0xd0, "mdtra", IF_RRfa, F_Z),
1019 INSTR(0xd1, "ddtra", IF_RRfa, F_Z),
1020 INSTR(0xd2, "adtra", IF_RRfa, F_Z),
1021 INSTR(0xd3, "sdtra", IF_RRfa, F_Z),
1022 INSTR(0xd4, "ldetr", IF_RRFd, F_Z),
1023 INSTR(0xd5, "ledtr", IF_RRFe, F_Z),
1024 INSTR(0xd6, "ltdtr", IF_RRE, F_Z),
1025 INSTR(0xd7, "fidtr", IF_RRFe, F_Z),
1026 INSTR(0xd8, "mxtra", IF_RRfa, F_Z),
1027 INSTR(0xd9, "dxtra", IF_RRfa, F_Z),
1028 INSTR(0xda, "axtra", IF_RRfa, F_Z),
1029 INSTR(0xdb, "sxtra", IF_RRfa, F_Z),
1030 INSTR(0xdc, "lxdtr", IF_RRFd, F_Z),
1031 INSTR(0xdd, "ldxtr", IF_RRFe, F_Z),
1032 INSTR(0xde, "ltxtr", IF_RRE, F_Z),
1033 INSTR(0xdf, "fixtr", IF_RRFe, F_Z),
1034 INSTR(0xe0, "kdtr", IF_RRE, F_Z),
1035 INSTR(0xe1, "cgdtra", IF_RRFe, F_Z),
1036 INSTR(0xe2, "cudtr", IF_RRE, F_Z),
1037 INSTR(0xe3, "csdtr", IF_RRFd, F_Z),
1038 INSTR(0xe4, "cdtr", IF_RRE, F_Z),
1039 INSTR(0xe5, "eedtr", IF_RRE, F_Z),
1040 INSTR(0xe7, "esdtr", IF_RRE, F_Z),
1041 INSTR(0xe8, "kxtr", IF_RRE, F_Z),
1042 INSTR(0xe9, "cgxtra", IF_RRFe, F_Z),
1043 INSTR(0xea, "cuxtr", IF_RRE, F_Z),
1044 INSTR(0xeb, "csxtr", IF_RRFd, F_Z),
1045 INSTR(0xec, "cxtr", IF_RRE, F_Z),
1046 INSTR(0xed, "eextr", IF_RRE, F_Z),
1047 INSTR(0xef, "esxtr", IF_RRE, F_Z),
1048 INSTR(0xf1, "cdgtra", IF_RRE, F_Z),
1049 INSTR(0xf2, "cdutr", IF_RRE, F_Z),
1050 INSTR(0xf3, "cdstr", IF_RRE, F_Z),
1051 INSTR(0xf4, "cedtr", IF_RRE, F_Z),

```



```

1052 INSTR(0xf5, "qadtr", IF_RRFb, F_Z),
1053 INSTR(0xf6, "iedtr", IF_RRFb, F_Z),
1054 INSTR(0xf7, "rrdtr", IF_RRFb, F_Z),
1055 INSTR(0xf9, "cxgtra", IF_RRE, F_Z),
1056 INSTR(0xfa, "cxutr", IF_RRE, F_Z),
1057 INSTR(0xfb, "cxstr", IF_RRE, F_Z),
1058 INSTR(0xfc, "cextr", IF_RRE, F_Z),
1059 INSTR(0xfd, "qaxtr", IF_RRFb, F_Z),
1060 INSTR(0xfe, "iextr", IF_RRFb, F_Z),
1061 INSTR(0xff, "rrxtr", IF_RRFb, F_Z),
1062 };

1064 static const struct inst_table tbl_b9xx[256] = {
1065 INSTR(0x00, "lpgr", IF_RRE, F_Z),
1066 INSTR(0x01, "lngr", IF_RRE, F_Z),
1067 INSTR(0x02, "ltgr", IF_RRE, F_Z),
1068 INSTR(0x03, "lcgr", IF_RRE, F_Z),
1069 INSTR(0x04, "lgr", IF_RRE, F_Z),
1070 INSTR(0x05, "lurag", IF_RRE, F_Z),
1071 INSTR(0x06, "lgr", IF_RRE, F_Z),
1072 INSTR(0x07, "lghr", IF_RRE, F_Z),
1073 INSTR(0x08, "agr", IF_RRE, F_Z),
1074 INSTR(0x09, "sgr", IF_RRE, F_Z),
1075 INSTR(0x0a, "algr", IF_RRE, F_Z),
1076 INSTR(0x0b, "slgr", IF_RRE, F_Z),
1077 INSTR(0x0c, "msgr", IF_RRE, F_Z),
1078 INSTR(0x0d, "dsgr", IF_RRE, F_Z),
1079 INSTR(0x0e, "eregg", IF_RRE, F_Z),
1080 INSTR(0x0f, "lrvgr", IF_RRE, F_Z),
1081 INSTR(0x10, "lpgfr", IF_RRE, F_Z),
1082 INSTR(0x11, "lngfr", IF_RRE, F_Z),
1083 INSTR(0x12, "ltgfr", IF_RRE, F_Z),
1084 INSTR(0x13, "lcfgr", IF_RRE, F_Z),
1085 INSTR(0x14, "lgfr", IF_RRE, F_Z),
1086 INSTR(0x16, "llgfr", IF_RRE, F_Z),
1087 INSTR(0x17, "llgtr", IF_RRE, F_Z),
1088 INSTR(0x18, "agfr", IF_RRE, F_Z),
1089 INSTR(0x19, "sgfr", IF_RRE, F_Z),
1090 INSTR(0x1a, "algfr", IF_RRE, F_Z),
1091 INSTR(0x1b, "slgfr", IF_RRE, F_Z),
1092 INSTR(0x1c, "msgfr", IF_RRE, F_Z),
1093 INSTR(0x1d, "dsgr", IF_RRE, F_Z),
1094 INSTR(0x1e, "kmac", IF_RRE, F_390 | F_Z),
1095 INSTR(0x1f, "lrvr", IF_RRE, F_390 | F_Z),
1096 INSTR(0x20, "cgr", IF_RRE, F_Z),
1097 INSTR(0x21, "clgr", IF_RRE, F_Z),
1098 INSTR(0x25, "sturg", IF_RRE, F_Z),
1099 INSTR(0x26, "lbr", IF_RRE, F_Z),
1100 INSTR(0x27, "lhr", IF_RRE, F_Z),
1101 INSTR(0x28, "pckmo", IF_RRE, F_Z),
1102 INSTR(0x2a, "kmf", IF_RRE, F_Z),
1103 INSTR(0x2b, "kmo", IF_RRE, F_Z),
1104 INSTR(0x2c, "pcc", IF_RRE, F_Z),
1105 INSTR(0x2d, "kmctr", IF_RRFd, F_Z),
1106 INSTR(0x2e, "km", IF_RRE, F_390 | F_Z),
1107 INSTR(0x2f, "kmc", IF_RRE, F_390 | F_Z),
1108 INSTR(0x30, "cgfr", IF_RRE, F_Z),
1109 INSTR(0x31, "clgfr", IF_RRE, F_Z),
1110 INSTR(0x3e, "kimd", IF_RRE, F_390 | F_Z),
1111 INSTR(0x3f, "klmd", IF_RRE, F_390 | F_Z),
1112 INSTR(0x41, "cfdtr", IF_RRFb, F_Z),
1113 INSTR(0x42, "clgdtr", IF_RRFb, F_Z),
1114 INSTR(0x43, "clfdtr", IF_RRFb, F_Z),
1115 INSTR(0x46, "bctgr", IF_RRE, F_Z),
1116 INSTR(0x49, "cfxtr", IF_RRFb, F_Z),
1117 INSTR(0x4a, "clgxtr", IF_RRFb, F_Z),

```

```

1118 INSTR(0x4b, "clfxtr", IF_RRFb, F_Z),
1119 INSTR(0x51, "cdftr", IF_RRE, F_Z),
1120 INSTR(0x52, "cdlgr", IF_RRFb, F_Z),
1121 INSTR(0x53, "cdftr", IF_RRFb, F_Z),
1122 INSTR(0x59, "cxftr", IF_RRE, F_Z),
1123 INSTR(0x5a, "cxlgr", IF_RRFb, F_Z),
1124 INSTR(0x5b, "cxlftr", IF_RRFb, F_Z),
1125 INSTR(0x60, "cgtr", IF_RRFb, F_Z),
1126 INSTR(0x61, "clgtr", IF_RRFb, F_Z),
1127 INSTR(0x72, "crt", IF_RRFb, F_Z),
1128 INSTR(0x73, "clrt", IF_RRFb, F_Z),
1129 INSTR(0x80, "ngr", IF_RRE, F_Z),
1130 INSTR(0x81, "ogr", IF_RRE, F_Z),
1131 INSTR(0x82, "xgr", IF_RRE, F_Z),
1132 INSTR(0x83, "flogr", IF_RRE, F_Z),
1133 INSTR(0x84, "llgcr", IF_RRE, F_Z),
1134 INSTR(0x85, "llghr", IF_RRE, F_Z),
1135 INSTR(0x86, "mlgr", IF_RRE, F_Z),
1136 INSTR(0x87, "dlgr", IF_RRE, F_Z),
1137 INSTR(0x88, "alcgr", IF_RRE, F_Z),
1138 INSTR(0x89, "slbgr", IF_RRE, F_Z),
1139 INSTR(0x8a, "cspg", IF_RRE, F_Z),
1140 INSTR(0x8d, "epsw", IF_RRE, F_390 | F_Z),
1141 INSTR(0x8e, "idte", IF_RRFb, F_Z),
1142 INSTR(0x8f, "crdte", IF_RRFb, F_Z),
1143 INSTR(0x90, "trtt", IF_RRFb, F_390 | F_Z),
1144 INSTR(0x91, "trto", IF_RRFb, F_390 | F_Z),
1145 INSTR(0x92, "trot", IF_RRFb, F_390 | F_Z),
1146 INSTR(0x93, "troo", IF_RRFb, F_390 | F_Z),
1147 INSTR(0x94, "llcr", IF_RRE, F_Z),
1148 INSTR(0x95, "llhr", IF_RRE, F_Z),
1149 INSTR(0x96, "mlr", IF_RRE, F_390 | F_Z),
1150 INSTR(0x97, "dlr", IF_RRE, F_390 | F_Z),
1151 INSTR(0x98, "alcr", IF_RRE, F_390 | F_Z),
1152 INSTR(0x99, "slbr", IF_RRE, F_390 | F_Z),
1153 INSTR(0x9a, "epair", IF_RRE, F_Z),
1154 INSTR(0x9b, "esair", IF_RRE, F_Z),
1155 INSTR(0x9d, "esea", IF_RRE, F_Z),
1156 INSTR(0x9e, "pti", IF_RRE, F_Z),
1157 INSTR(0x9f, "ssair", IF_RRE, F_Z),
1158 INSTR(0xa2, "ptf", IF_RRE, F_Z),
1159 INSTR(0xaa, "lptea", IF_RRFb, F_Z),
1160 INSTR(0xae, "rrbm", IF_RRE, F_Z),
1161 INSTR(0xaf, "pfmf", IF_RRE, F_Z),
1162 INSTR(0xb0, "cu14", IF_RRFb, F_Z),
1163 INSTR(0xb1, "cu24", IF_RRFb, F_Z),
1164 INSTR(0xb2, "cu41", IF_RRE, F_Z),
1165 INSTR(0xb3, "cu42", IF_RRE, F_Z),
1166 INSTR(0xbd, "trtre", IF_RRFb, F_Z),
1167 INSTR(0xbe, "srstu", IF_RRE, F_Z),
1168 INSTR(0xbf, "trte", IF_RRFb, F_Z),
1169 INSTR(0xc8, "ahhhr", IF_RRFa, F_Z),
1170 INSTR(0xc9, "shhhr", IF_RRFa, F_Z),
1171 INSTR(0xca, "alhhhr", IF_RRFa, F_Z),
1172 INSTR(0xcb, "slhhhr", IF_RRFa, F_Z),
1173 INSTR(0xcd, "chhr", IF_RRE, F_Z),
1174 INSTR(0xcf, "clhhr", IF_RRE, F_Z),
1175 INSTR(0xd8, "ahhhr", IF_RRFa, F_Z),
1176 INSTR(0xd9, "shhhr", IF_RRFa, F_Z),
1177 INSTR(0xda, "alhhhr", IF_RRFa, F_Z),
1178 INSTR(0xdb, "slhhhr", IF_RRFa, F_Z),
1179 INSTR(0xdd, "chlr", IF_RRE, F_Z),
1180 INSTR(0xdf, "clhhr", IF_RRE, F_Z),
1181 INSTR(0xe1, "popcnt", IF_RRE, F_Z),
1182 INSTR(0xe2, "locgr", IF_RRFb, F_Z),
1183 INSTR(0xe4, "ngrk", IF_RRFa, F_Z),

```

```

1184 INSTR(0xe6, "ogrkr", IF_RRFa, F_Z),
1185 INSTR(0xe7, "xgrkr", IF_RRFa, F_Z),
1186 INSTR(0xe8, "agrkr", IF_RRFa, F_Z),
1187 INSTR(0xe9, "sgrkr", IF_RRFa, F_Z),
1188 INSTR(0xea, "algrkr", IF_RRFa, F_Z),
1189 INSTR(0xeb, "slgrkr", IF_RRFa, F_Z),
1190 INSTR(0xf2, "locgr", IF_RRFc, F_Z),
1191 INSTR(0xf4, "nrkr", IF_RRFa, F_Z),
1192 INSTR(0xf6, "ork", IF_RRFa, F_Z),
1193 INSTR(0xf7, "xrkr", IF_RRFa, F_Z),
1194 INSTR(0xf8, "ark", IF_RRFa, F_Z),
1195 INSTR(0xf9, "srkr", IF_RRFa, F_Z),
1196 INSTR(0xfa, "alrkr", IF_RRFa, F_Z),
1197 INSTR(0xfb, "slrkr", IF_RRFa, F_Z),
1198 };

1200 static const struct inst_table tbl_c0x[16] = {
1201 INSTR(0x0, "larl", IF_RILb, F_390 | F_Z),
1202 INSTR(0x1, "lgfi", IF_RILa, F_Z),
1203 INSTR(0x4, "brcl", IF_RILc, F_390 | F_Z),
1204 INSTR(0x5, "brasl", IF_RILb, F_390 | F_Z),
1205 INSTR(0x6, "xihf", IF_RILa, F_Z),
1206 INSTR(0x7, "xilf", IF_RILa, F_Z),
1207 INSTR(0x8, "iihf", IF_RILa, F_Z),
1208 INSTR(0x9, "iilf", IF_RILa, F_Z),
1209 INSTR(0xa, "nihf", IF_RILa, F_Z),
1210 INSTR(0xb, "nilf", IF_RILa, F_Z),
1211 INSTR(0xc, "oihf", IF_RILa, F_Z),
1212 INSTR(0xd, "oilf", IF_RILa, F_Z),
1213 INSTR(0xe, "llihf", IF_RILa, F_Z),
1214 INSTR(0xf, "llilf", IF_RILa, F_Z),
1215 };

1217 static const struct inst_table tbl_c2x[16] = {
1218 INSTR(0x0, "msgfi", IF_RILa, F_Z),
1219 INSTR(0x1, "msfi", IF_RILa, F_Z),
1220 INSTR(0x4, "slgfi", IF_RILa, F_Z),
1221 INSTR(0x5, "slfi", IF_RILa, F_Z),
1222 INSTR(0x8, "agfi", IF_RILa, F_Z),
1223 INSTR(0x9, "afi", IF_RILa, F_Z),
1224 INSTR(0xa, "algfi", IF_RILa, F_Z),
1225 INSTR(0xb, "alfi", IF_RILa, F_Z),
1226 INSTR(0xc, "cgfi", IF_RILa, F_Z),
1227 INSTR(0xd, "cfi", IF_RILa, F_Z),
1228 INSTR(0xe, "clgfi", IF_RILa, F_Z),
1229 INSTR(0xf, "clfi", IF_RILa, F_Z),
1230 };

1232 static const struct inst_table tbl_c4x[16] = {
1233 INSTR(0x2, "llhrl", IF_RILb, F_Z),
1234 INSTR(0x4, "lghrl", IF_RILb, F_Z),
1235 INSTR(0x5, "lhrl", IF_RILb, F_Z),
1236 INSTR(0x6, "llghrl", IF_RILb, F_Z),
1237 INSTR(0x7, "sthrl", IF_RILb, F_Z),
1238 INSTR(0x8, "lgrl", IF_RILb, F_Z),
1239 INSTR(0xb, "stgrl", IF_RILb, F_Z),
1240 INSTR(0xc, "lgfrl", IF_RILb, F_Z),
1241 INSTR(0xd, "lrl", IF_RILb, F_Z),
1242 INSTR(0xe, "llgfrl", IF_RILb, F_Z),
1243 INSTR(0xf, "strl", IF_RILb, F_Z),
1244 };

1246 static const struct inst_table tbl_c6x[16] = {
1247 INSTR(0x0, "exrl", IF_RILb, F_Z),
1248 INSTR(0x2, "pfdrl", IF_RILc, F_Z),
1249 INSTR(0x4, "cghrl", IF_RILb, F_Z),

```

```

1250 INSTR(0x5, "chr1", IF_RILb, F_Z),
1251 INSTR(0x6, "clghrl", IF_RILb, F_Z),
1252 INSTR(0x7, "clhrl", IF_RILb, F_Z),
1253 INSTR(0x8, "cgrl", IF_RILb, F_Z),
1254 INSTR(0xa, "clgrl", IF_RILb, F_Z),
1255 INSTR(0xc, "cgfrl", IF_RILb, F_Z),
1256 INSTR(0xd, "crl", IF_RILb, F_Z),
1257 INSTR(0xe, "clgfrl", IF_RILb, F_Z),
1258 INSTR(0xf, "clrl", IF_RILb, F_Z),
1259 };

1261 static const struct inst_table tbl_c8x[16] = {
1262 INSTR(0x0, "mvcos", IF_SSF, F_Z),
1263 INSTR(0x1, "ectg", IF_SSF, F_Z),
1264 INSTR(0x2, "csst", IF_SSF, F_Z),
1265 INSTR(0x4, "lpd", IF_SSF, F_Z),
1266 INSTR(0x5, "lpdg", IF_SSF, F_Z),
1267 };

1269 static const struct inst_table tbl_ccx[16] = {
1270 INSTR(0x6, "brcth", IF_RILb, F_Z),
1271 INSTR(0x8, "aih", IF_RILa, F_Z),
1272 INSTR(0xa, "alsih", IF_RILa, F_Z),
1273 INSTR(0xb, "alsihn", IF_RILa, F_Z),
1274 INSTR(0xd, "cih", IF_RILa, F_Z),
1275 INSTR(0xf, "clih", IF_RILa, F_Z),
1276 };

1278 static const struct inst_table tbl_e3xx[256] = {
1279 INSTR(0x02, "ltg", IF_RXYa, F_Z),
1280 INSTR(0x03, "lrag", IF_RXYa, F_Z),
1281 INSTR(0x04, "lg", IF_RXYa, F_Z),
1282 INSTR(0x06, "cvby", IF_RXYa, F_Z),
1283 INSTR(0x08, "ag", IF_RXYa, F_Z),
1284 INSTR(0x09, "sg", IF_RXYa, F_Z),
1285 INSTR(0x0a, "alg", IF_RXYa, F_Z),
1286 INSTR(0x0b, "slg", IF_RXYa, F_Z),
1287 INSTR(0x0c, "msg", IF_RXYa, F_Z),
1288 INSTR(0x0d, "dsg", IF_RXYa, F_Z),
1289 INSTR(0x0e, "cvbg", IF_RXYa, F_Z),
1290 INSTR(0x0f, "lrvg", IF_RXYa, F_Z),
1291 INSTR(0x12, "lt", IF_RXYa, F_Z),
1292 INSTR(0x13, "lray", IF_RXYa, F_Z),
1293 INSTR(0x14, "lgf", IF_RXYa, F_Z),
1294 INSTR(0x15, "lgh", IF_RXYa, F_Z),
1295 INSTR(0x16, "llgf", IF_RXYa, F_Z),
1296 INSTR(0x17, "llgt", IF_RXYa, F_Z),
1297 INSTR(0x18, "agf", IF_RXYa, F_Z),
1298 INSTR(0x19, "sgf", IF_RXYa, F_Z),
1299 INSTR(0x1a, "algf", IF_RXYa, F_Z),
1300 INSTR(0x1b, "slgf", IF_RXYa, F_Z),
1301 INSTR(0x1c, "msgf", IF_RXYa, F_Z),
1302 INSTR(0x1d, "dsgf", IF_RXYa, F_Z),
1303 INSTR(0x1e, "lrv", IF_RXYa, F_390 | F_Z),
1304 INSTR(0x1f, "lrvh", IF_RXYa, F_390 | F_Z),
1305 INSTR(0x20, "cg", IF_RXYa, F_Z),
1306 INSTR(0x21, "clg", IF_RXYa, F_Z),
1307 INSTR(0x24, "stg", IF_RXYa, F_Z),
1308 INSTR(0x25, "ntstg", IF_RXYa, F_Z),
1309 INSTR(0x26, "cvdy", IF_RXYa, F_Z),
1310 INSTR(0x2e, "cvdg", IF_RXYa, F_Z),
1311 INSTR(0x2f, "strvg", IF_RXYa, F_Z),
1312 INSTR(0x30, "cgf", IF_RXYa, F_Z),
1313 INSTR(0x31, "clgf", IF_RXYa, F_Z),
1314 INSTR(0x32, "ltgf", IF_RXYa, F_Z),
1315 INSTR(0x34, "cgh", IF_RXYa, F_Z),

```

```

1316 INSTR(0x36, "pfd", IF_RXYb, F_Z),
1317 INSTR(0x3e, "strv", IF_RXYa, F_390 | F_Z),
1318 INSTR(0x3f, "strvh", IF_RXYa, F_390 | F_Z),
1319 INSTR(0x46, "bctg", IF_RXYa, F_Z),
1320 INSTR(0x50, "sty", IF_RXYa, F_Z),
1321 INSTR(0x51, "msy", IF_RXYa, F_Z),
1322 INSTR(0x54, "ny", IF_RXYa, F_Z),
1323 INSTR(0x55, "cly", IF_RXYa, F_Z),
1324 INSTR(0x56, "oy", IF_RXYa, F_Z),
1325 INSTR(0x57, "xy", IF_RXYa, F_Z),
1326 INSTR(0x58, "ly", IF_RXYa, F_Z),
1327 INSTR(0x59, "cy", IF_RXYa, F_Z),
1328 INSTR(0x5a, "ay", IF_RXYa, F_Z),
1329 INSTR(0x5b, "sy", IF_RXYa, F_Z),
1330 INSTR(0x5c, "mfy", IF_RXYa, F_Z),
1331 INSTR(0x5e, "aly", IF_RXYa, F_Z),
1332 INSTR(0x5f, "sly", IF_RXYa, F_Z),
1333 INSTR(0x70, "sthy", IF_RXYa, F_Z),
1334 INSTR(0x71, "lay", IF_RXYa, F_Z),
1335 INSTR(0x72, "stcy", IF_RXYa, F_Z),
1336 INSTR(0x73, "icy", IF_RXYa, F_Z),
1337 INSTR(0x75, "laey", IF_RXYa, F_Z),
1338 INSTR(0x76, "lb", IF_RXYa, F_Z),
1339 INSTR(0x77, "lgb", IF_RXYa, F_Z),
1340 INSTR(0x78, "lhy", IF_RXYa, F_Z),
1341 INSTR(0x79, "chy", IF_RXYa, F_Z),
1342 INSTR(0x7a, "ahy", IF_RXYa, F_Z),
1343 INSTR(0x7b, "shy", IF_RXYa, F_Z),
1344 INSTR(0x7c, "mhy", IF_RXYa, F_Z),
1345 INSTR(0x80, "ng", IF_RXYa, F_Z),
1346 INSTR(0x81, "og", IF_RXYa, F_Z),
1347 INSTR(0x82, "xg", IF_RXYa, F_Z),
1348 INSTR(0x85, "lgat", IF_RXYa, F_Z),
1349 INSTR(0x86, "mlg", IF_RXYa, F_Z),
1350 INSTR(0x87, "dlg", IF_RXYa, F_Z),
1351 INSTR(0x88, "alcg", IF_RXYa, F_Z),
1352 INSTR(0x89, "slbg", IF_RXYa, F_Z),
1353 INSTR(0x8e, "stpq", IF_RXYa, F_Z),
1354 INSTR(0x8f, "lpq", IF_RXYa, F_Z),
1355 INSTR(0x90, "llgc", IF_RXYa, F_Z),
1356 INSTR(0x91, "llgh", IF_RXYa, F_Z),
1357 INSTR(0x94, "llc", IF_RXYa, F_Z),
1358 INSTR(0x95, "llh", IF_RXYa, F_Z),
1359 INSTR(0x96, "ml", IF_RXYa, F_390 | F_Z),
1360 INSTR(0x97, "dl", IF_RXYa, F_390 | F_Z),
1361 INSTR(0x98, "alc", IF_RXYa, F_390 | F_Z),
1362 INSTR(0x99, "slb", IF_RXYa, F_390 | F_Z),
1363 INSTR(0x9c, "llgtat", IF_RXYa, F_Z),
1364 INSTR(0x9d, "llgfat", IF_RXYa, F_Z),
1365 INSTR(0x9f, "lat", IF_RXYa, F_Z),
1366 INSTR(0xc0, "lbh", IF_RXYa, F_Z),
1367 INSTR(0xc2, "llch", IF_RXYa, F_Z),
1368 INSTR(0xc3, "stch", IF_RXYa, F_Z),
1369 INSTR(0xc4, "lhh", IF_RXYa, F_Z),
1370 INSTR(0xc6, "llhh", IF_RXYa, F_Z),
1371 INSTR(0xc7, "sthh", IF_RXYa, F_Z),
1372 INSTR(0xc8, "lfhat", IF_RXYa, F_Z),
1373 INSTR(0xca, "lfh", IF_RXYa, F_Z),
1374 INSTR(0xcb, "stfh", IF_RXYa, F_Z),
1375 INSTR(0xcd, "chf", IF_RXYa, F_Z),
1376 INSTR(0xcf, "clhf", IF_RXYa, F_Z),
1377 };

1379 static const struct inst_table tbl_e5xx[256] = {
1380 INSTR(0x00, "lasp", IF_SSE, F_390 | F_Z),
1381 INSTR(0x01, "tprot", IF_SSE, F_390 | F_Z),

```

```

1382 INSTR(0x02, "strag", IF_SSE, F_Z),
1383 INSTR(0x0e, "mvcsk", IF_SSE, F_390 | F_Z),
1384 INSTR(0x0f, "mvdck", IF_SSE, F_390 | F_Z),
1385 INSTR(0x44, "mvhhi", IF_SIL, F_Z),
1386 INSTR(0x48, "mvghi", IF_SIL, F_Z),
1387 INSTR(0x4c, "mvhi", IF_SIL, F_Z),
1388 INSTR(0x54, "chhsi", IF_SIL, F_Z),
1389 INSTR(0x55, "clhhsi", IF_SIL, F_Z),
1390 INSTR(0x58, "cghsi", IF_SIL, F_Z),
1391 INSTR(0x59, "clghsi", IF_SIL, F_Z),
1392 INSTR(0x5c, "chsi", IF_SIL, F_Z),
1393 INSTR(0x5d, "clfhsi", IF_SIL, F_Z),
1394 INSTR(0x60, "tbegin", IF_SIL, F_Z),
1395 INSTR(0x61, "tbeginc", IF_SIL, F_Z),
1396 };

1398 static const struct inst_table tbl_ebxx[256] = {
1399 INSTR(0x04, "lmg", IF_RSYa, F_Z),
1400 INSTR(0x0a, "srag", IF_RSYa, F_Z),
1401 INSTR(0x0b, "slag", IF_RSYa, F_Z),
1402 INSTR(0x0c, "srlg", IF_RSYa, F_Z),
1403 INSTR(0x0d, "sllg", IF_RSYa, F_Z),
1404 INSTR(0x0f, "tracg", IF_RSYa, F_Z),
1405 INSTR(0x14, "csy", IF_RSYa, F_Z),
1406 INSTR(0x1c, "rllg", IF_RSYa, F_Z),
1407 INSTR(0x1d, "rll", IF_RSYa, F_390 | F_Z),
1408 INSTR(0x20, "clmh", IF_RSYb, F_Z),
1409 INSTR(0x21, "clmy", IF_RSYb, F_Z),
1410 INSTR(0x23, "clt", IF_RSYb, F_Z),
1411 INSTR(0x24, "stmg", IF_RSYa, F_Z),
1412 INSTR(0x25, "stctg", IF_RSYa, F_Z | F_CTL_REG),
1413 INSTR(0x26, "stmh", IF_RSYa, F_Z),
1414 INSTR(0x2b, "clgt", IF_RSYb, F_Z),
1415 INSTR(0x2c, "stcmh", IF_RSYb, F_Z),
1416 INSTR(0x2d, "stcmy", IF_RSYb, F_Z),
1417 INSTR(0x2f, "lctlg", IF_RSYa, F_Z | F_CTL_REG),
1418 INSTR(0x30, "csg", IF_RSYa, F_Z),
1419 INSTR(0x31, "cdsy", IF_RSYa, F_Z),
1420 INSTR(0x3e, "cdsg", IF_RSYa, F_Z),
1421 INSTR(0x44, "bxhg", IF_RSYa, F_Z),
1422 INSTR(0x45, "bxleg", IF_RSYa, F_Z),
1423 INSTR(0x4c, "ecag", IF_RSYa, F_Z),
1424 INSTR(0x51, "tmy", IF_Siy, F_Z),
1425 INSTR(0x52, "mviy", IF_Siy, F_Z),
1426 INSTR(0x54, "niy", IF_Siy, F_Z),
1427 INSTR(0x55, "cliy", IF_Siy, F_Z),
1428 INSTR(0x56, "oiy", IF_Siy, F_Z),
1429 INSTR(0x57, "xiy", IF_Siy, F_Z),
1430 INSTR(0x6a, "asi", IF_Siy, F_Z),
1431 INSTR(0x6e, "alsi", IF_Siy, F_Z),
1432 INSTR(0x80, "icmh", IF_RSYb, F_Z),
1433 INSTR(0x81, "icmy", IF_RSYb, F_Z),
1434 INSTR(0x8e, "mvclu", IF_RSYa, F_390 | F_Z),
1435 INSTR(0x8f, "clclu", IF_RSYa, F_390 | F_Z),
1436 INSTR(0x90, "stmy", IF_RSYa, F_Z),
1437 INSTR(0x96, "lmh", IF_RSYa, F_Z),
1438 INSTR(0x98, "lmy", IF_RSYa, F_Z),
1439 INSTR(0x9a, "lamy", IF_RSYa, F_Z),
1440 INSTR(0x9b, "stamy", IF_RSYa, F_Z),
1441 INSTR(0xc0, "tp", IF_RSLa, F_390 | F_Z),
1442 INSTR(0xdc, "srak", IF_RSYa, F_Z),
1443 INSTR(0xdd, "slak", IF_RSYa, F_Z),
1444 INSTR(0xde, "srllk", IF_RSYa, F_Z),
1445 INSTR(0xdf, "sllk", IF_RSYa, F_Z),
1446 INSTR(0xe2, "locg", IF_RSYb, F_Z),
1447 INSTR(0xe3, "stocg", IF_RSYb, F_Z),

```

```

1448 INSTR(0xe4, "lang", IF_RSYa, F_Z),
1449 INSTR(0xe6, "laog", IF_RSYa, F_Z),
1450 INSTR(0xe7, "laxg", IF_RSYa, F_Z),
1451 INSTR(0xe8, "laag", IF_RSYa, F_Z),
1452 INSTR(0xea, "laalg", IF_RSYa, F_Z),
1453 INSTR(0xf2, "loc", IF_RSYb, F_Z),
1454 INSTR(0xf3, "stoc", IF_RSYb, F_Z),
1455 INSTR(0xf4, "lan", IF_RSYa, F_Z),
1456 INSTR(0xf6, "lao", IF_RSYa, F_Z),
1457 INSTR(0xf7, "lax", IF_RSYa, F_Z),
1458 INSTR(0xf8, "laa", IF_RSYa, F_Z),
1459 INSTR(0xfa, "laal", IF_RSYa, F_Z),
1460 };

```

```

1462 static const struct inst_table tbl_ecxx[256] = {
1463 INSTR(0x44, "brxhg", IF_RIEe, F_Z),
1464 INSTR(0x45, "brxlg", IF_RIEe, F_Z),
1465 INSTR(0x51, "risblg", IF_RIEf, F_Z),
1466 INSTR(0x54, "rnsbg", IF_RIEf, F_Z),
1467 INSTR(0x55, "risbg", IF_RIEf, F_Z),
1468 INSTR(0x56, "rosbg", IF_RIEf, F_Z),
1469 INSTR(0x57, "rxsbg", IF_RIEf, F_Z),
1470 INSTR(0x59, "risbgn", IF_RIEf, F_Z),
1471 INSTR(0x5d, "risbhg", IF_RIEf, F_Z),
1472 INSTR(0x64, "cgrj", IF_RIEb, F_Z),
1473 INSTR(0x65, "clgrj", IF_RIEb, F_Z),
1474 INSTR(0x70, "cgit", IF_RIEa, F_Z),
1475 INSTR(0x71, "clgit", IF_RIEa, F_Z),
1476 INSTR(0x72, "cit", IF_RIEa, F_Z),
1477 INSTR(0x73, "clfit", IF_RIEa, F_Z),
1478 INSTR(0x76, "crj", IF_RIEb, F_Z),
1479 INSTR(0x77, "clrj", IF_RIEb, F_Z),
1480 INSTR(0x7c, "cgij", IF_RIEc, F_Z),
1481 INSTR(0x7d, "clgij", IF_RIEc, F_Z),
1482 INSTR(0x7e, "cij", IF_RIEc, F_Z),
1483 INSTR(0x7f, "clij", IF_RIEc, F_Z),
1484 INSTR(0xd8, "ahik", IF_RIEd, F_Z),
1485 INSTR(0xd9, "aghik", IF_RIEd, F_Z),
1486 INSTR(0xda, "alhsik", IF_RIEd, F_Z),
1487 INSTR(0xdb, "alghsik", IF_RIEd, F_Z),
1488 INSTR(0xe4, "cgrb", IF_RRS, F_Z),
1489 INSTR(0xe5, "clgrb", IF_RRS, F_Z),
1490 INSTR(0xf6, "crb", IF_RRS, F_Z),
1491 INSTR(0xf7, "clrb", IF_RRS, F_Z),
1492 INSTR(0xfc, "cgib", IF_RIS, F_Z),
1493 INSTR(0xfd, "clgib", IF_RIS, F_Z),
1494 INSTR(0xfe, "cib", IF_RIS, F_Z),
1495 INSTR(0xff, "clib", IF_RIS, F_Z),
1496 };

```

```

1498 static const struct inst_table tbl_edxx[256] = {
1499 INSTR(0x04, "ldeb", IF_RXE, F_390 | F_Z),
1500 INSTR(0x05, "lxdb", IF_RXE, F_390 | F_Z),
1501 INSTR(0x06, "lxeb", IF_RXE, F_390 | F_Z),
1502 INSTR(0x07, "mxdb", IF_RXE, F_390 | F_Z),
1503 INSTR(0x08, "keb", IF_RXE, F_390 | F_Z),
1504 INSTR(0x09, "ceb", IF_RXE, F_390 | F_Z),
1505 INSTR(0x0a, "aeb", IF_RXE, F_390 | F_Z),
1506 INSTR(0x0b, "seb", IF_RXE, F_390 | F_Z),
1507 INSTR(0x0c, "mdeb", IF_RXE, F_390 | F_Z),
1508 INSTR(0x0d, "deb", IF_RXE, F_390 | F_Z),
1509 INSTR(0x0e, "maeb", IF_RXF, F_390 | F_Z),
1510 INSTR(0x0f, "mseb", IF_RXF, F_390 | F_Z),
1511 INSTR(0x10, "tceb", IF_RXE, F_390 | F_Z),
1512 INSTR(0x11, "tcdb", IF_RXE, F_390 | F_Z),
1513 INSTR(0x12, "tcxb", IF_RXE, F_390 | F_Z),

```

```

1514 INSTR(0x14, "sqeb", IF_RXE, F_390 | F_Z),
1515 INSTR(0x15, "sqdb", IF_RXE, F_390 | F_Z),
1516 INSTR(0x17, "meeb", IF_RXE, F_390 | F_Z),
1517 INSTR(0x18, "kdb", IF_RXE, F_390 | F_Z),
1518 INSTR(0x19, "cdb", IF_RXE, F_390 | F_Z),
1519 INSTR(0x1a, "adb", IF_RXE, F_390 | F_Z),
1520 INSTR(0x1b, "sdb", IF_RXE, F_390 | F_Z),
1521 INSTR(0x1c, "mdb", IF_RXE, F_390 | F_Z),
1522 INSTR(0x1d, "ddb", IF_RXE, F_390 | F_Z),
1523 INSTR(0x1e, "madb", IF_RXF, F_390 | F_Z),
1524 INSTR(0x1f, "msdb", IF_RXF, F_390 | F_Z),
1525 INSTR(0x24, "lde", IF_RXE, F_390 | F_Z),
1526 INSTR(0x25, "lxd", IF_RXE, F_390 | F_Z),
1527 INSTR(0x26, "lxe", IF_RXE, F_390 | F_Z),
1528 INSTR(0x2e, "mae", IF_RXF, F_390 | F_Z),
1529 INSTR(0x2f, "mse", IF_RXF, F_390 | F_Z),
1530 INSTR(0x34, "sqe", IF_RXE, F_390 | F_Z),
1531 INSTR(0x35, "sqd", IF_RXE, F_390 | F_Z),
1532 INSTR(0x37, "mee", IF_RXE, F_390 | F_Z),
1533 INSTR(0x38, "mayl", IF_RXF, F_Z),
1534 INSTR(0x39, "myl", IF_RXF, F_Z),
1535 INSTR(0x3a, "may", IF_RXF, F_Z),
1536 INSTR(0x3b, "my", IF_RXF, F_Z),
1537 INSTR(0x3c, "mayh", IF_RXF, F_Z),
1538 INSTR(0x3d, "myh", IF_RXF, F_Z),
1539 INSTR(0x3e, "mad", IF_RXF, F_390 | F_Z),
1540 INSTR(0x3f, "msd", IF_RXF, F_390 | F_Z),
1541 INSTR(0x40, "sldt", IF_RXF, F_Z),
1542 INSTR(0x41, "srdt", IF_RXF, F_Z),
1543 INSTR(0x48, "slxt", IF_RXF, F_Z),
1544 INSTR(0x49, "srxt", IF_RXF, F_Z),
1545 INSTR(0x50, "tdcet", IF_RXE, F_Z),
1546 INSTR(0x51, "tdget", IF_RXE, F_Z),
1547 INSTR(0x54, "tdcddt", IF_RXE, F_Z),
1548 INSTR(0x55, "tdgdt", IF_RXE, F_Z),
1549 INSTR(0x58, "tdcxt", IF_RXE, F_Z),
1550 INSTR(0x59, "tdgxt", IF_RXE, F_Z),
1551 INSTR(0x64, "ley", IF_RXYa, F_Z),
1552 INSTR(0x65, "ldy", IF_RXYa, F_Z),
1553 INSTR(0x66, "stey", IF_RXYa, F_Z),
1554 INSTR(0x67, "stdy", IF_RXYa, F_Z),
1555 INSTR(0xa8, "czdt", IF_RSLb, F_Z),
1556 INSTR(0xa9, "czxt", IF_RSLb, F_Z),
1557 INSTR(0xaa, "cdzt", IF_RSLb, F_Z),
1558 INSTR(0xab, "cxzt", IF_RSLb, F_Z),
1559 };

```

```

1561 static const struct inst_table tbl_xx[256] = {
1562 INSTR(0x00, ".byte", IF_ZERO, F_370 | F_390 | F_Z),
1563 TABLE(0x01, tbl_0lxx, 1, 0, 0xff),
1564 INSTR(0x04, "spm", IF_RR, F_370 | F_Z),
1565 INSTR(0x05, "balr", IF_RR, F_370 | F_Z),
1566 INSTR(0x06, "bctr", IF_RR, F_370 | F_Z),
1567 TABLE(0x07, tbl_07, 1, 4, 0x0f),
1568 INSTR(0x08, "ssk", IF_RR, F_370),
1569 INSTR(0x09, "isk", IF_RR, F_370),
1570 INSTR(0x0a, "svc", IF_I, F_370 | F_390 | F_Z),
1571 INSTR(0x0b, "bsm", IF_RR, F_390 | F_Z),
1572 INSTR(0x0c, "bassm", IF_RR, F_390 | F_Z),
1573 INSTR(0x0d, "basr", IF_RR, F_390 | F_Z),
1574 INSTR(0x0e, "mvcl", IF_RR, F_370 | F_390 | F_Z),
1575 INSTR(0x0f, "clcl", IF_RR, F_370 | F_390 | F_Z),
1576 INSTR(0x10, "lpr", IF_RR, F_370 | F_390 | F_Z),
1577 INSTR(0x11, "lnr", IF_RR, F_370 | F_390 | F_Z),
1578 INSTR(0x12, "ltr", IF_RR, F_370 | F_390 | F_Z),
1579 INSTR(0x13, "lcr", IF_RR, F_370 | F_390 | F_Z),

```

1580	INSTR(0x14, "nr", IF_RR, F_370 F_390 F_Z),
1581	INSTR(0x15, "clr", IF_RR, F_370 F_390 F_Z),
1582	INSTR(0x16, "or", IF_RR, F_370 F_390 F_Z),
1583	INSTR(0x17, "xr", IF_RR, F_370 F_390 F_Z),
1584	INSTR(0x18, "lr", IF_RR, F_370 F_390 F_Z),
1585	INSTR(0x19, "cr", IF_RR, F_370 F_390 F_Z),
1586	INSTR(0x1a, "ar", IF_RR, F_370 F_390 F_Z),
1587	INSTR(0x1b, "sr", IF_RR, F_370 F_390 F_Z),
1588	INSTR(0x1c, "mr", IF_RR, F_370 F_390 F_Z),
1589	INSTR(0x1d, "dr", IF_RR, F_370 F_390 F_Z),
1590	INSTR(0x1e, "alr", IF_RR, F_370 F_390 F_Z),
1591	INSTR(0x1f, "slr", IF_RR, F_370 F_390 F_Z),
1592	INSTR(0x20, "lpdr", IF_RR, F_370 F_390 F_Z),
1593	INSTR(0x21, "lndr", IF_RR, F_370 F_390 F_Z),
1594	INSTR(0x22, "ltdr", IF_RR, F_370 F_390 F_Z),
1595	INSTR(0x23, "lcdr", IF_RR, F_370 F_390 F_Z),
1596	INSTR(0x24, "hdr", IF_RR, F_370 F_390 F_Z),
1597	MULTI(0x25, tbl_25),
1598	INSTR(0x26, "mxr", IF_RR, F_370 F_390 F_Z),
1599	INSTR(0x27, "mxd", IF_RR, F_370 F_390 F_Z),
1600	INSTR(0x28, "ldr", IF_RR, F_370 F_390 F_Z),
1601	INSTR(0x29, "cdr", IF_RR, F_370 F_390 F_Z),
1602	INSTR(0x2a, "adr", IF_RR, F_370 F_390 F_Z),
1603	INSTR(0x2b, "sdr", IF_RR, F_370 F_390 F_Z),
1604	INSTR(0x2c, "mdr", IF_RR, F_370 F_390 F_Z),
1605	INSTR(0x2d, "ddr", IF_RR, F_370 F_390 F_Z),
1606	INSTR(0x2e, "awr", IF_RR, F_370 F_390 F_Z),
1607	INSTR(0x2f, "swr", IF_RR, F_370 F_390 F_Z),
1608	INSTR(0x30, "lper", IF_RR, F_370 F_390 F_Z),
1609	INSTR(0x31, "lner", IF_RR, F_370 F_390 F_Z),
1610	INSTR(0x32, "lter", IF_RR, F_370 F_390 F_Z),
1611	INSTR(0x33, "lcer", IF_RR, F_370 F_390 F_Z),
1612	INSTR(0x34, "her", IF_RR, F_370 F_390 F_Z),
1613	MULTI(0x35, tbl_35),
1614	INSTR(0x36, "axr", IF_RR, F_370 F_390 F_Z),
1615	INSTR(0x37, "sxr", IF_RR, F_370 F_390 F_Z),
1616	INSTR(0x38, "ler", IF_RR, F_370 F_390 F_Z),
1617	INSTR(0x39, "cer", IF_RR, F_370 F_390 F_Z),
1618	INSTR(0x3a, "aer", IF_RR, F_370 F_390 F_Z),
1619	INSTR(0x3b, "ser", IF_RR, F_370 F_390 F_Z),
1620	MULTI(0x3c, tbl_3c),
1621	INSTR(0x3d, "der", IF_RR, F_370 F_390 F_Z),
1622	INSTR(0x3e, "aur", IF_RR, F_370 F_390 F_Z),
1623	INSTR(0x3f, "sur", IF_RR, F_370 F_390 F_Z),
1624	INSTR(0x40, "sth", IF_RXa, F_370 F_390 F_Z),
1625	INSTR(0x41, "la", IF_RXa, F_370 F_390 F_Z),
1626	INSTR(0x42, "stc", IF_RXa, F_370 F_390 F_Z),
1627	INSTR(0x43, "ic", IF_RXa, F_370 F_390 F_Z),
1628	INSTR(0x44, "ex", IF_RXa, F_370 F_390 F_Z),
1629	INSTR(0x45, "bal", IF_RXa, F_370 F_390 F_Z),
1630	INSTR(0x46, "bct", IF_RXa, F_370 F_390 F_Z),
1631	TABLE(0x47, tbl_47, 1, 4, 0x0f),
1632	INSTR(0x48, "lh", IF_RXa, F_370 F_390 F_Z),
1633	INSTR(0x49, "ch", IF_RXa, F_370 F_390 F_Z),
1634	INSTR(0x4a, "ah", IF_RXa, F_370 F_390 F_Z),
1635	INSTR(0x4b, "sh", IF_RXa, F_370 F_390 F_Z),
1636	INSTR(0x4c, "mh", IF_RXa, F_370 F_390 F_Z),
1637	INSTR(0x4d, "bas", IF_RXa, F_390 F_Z),
1638	INSTR(0x4e, "cvd", IF_RXa, F_370 F_390 F_Z),
1639	INSTR(0x4f, "cvb", IF_RXa, F_370 F_390 F_Z),
1640	INSTR(0x50, "st", IF_RXa, F_370 F_390 F_Z),
1641	INSTR(0x51, "lae", IF_RXa, F_390 F_Z),
1642	INSTR(0x54, "n", IF_RXa, F_370 F_390 F_Z),
1643	INSTR(0x55, "cl", IF_RXa, F_370 F_390 F_Z),
1644	INSTR(0x56, "o", IF_RXa, F_370 F_390 F_Z),
1645	INSTR(0x57, "x", IF_RXa, F_370 F_390 F_Z),

1646	INSTR(0x58, "l", IF_RXa, F_370 F_390 F_Z),
1647	INSTR(0x59, "c", IF_RXa, F_370 F_390 F_Z),
1648	INSTR(0x5a, "a", IF_RXa, F_370 F_390 F_Z),
1649	INSTR(0x5b, "s", IF_RXa, F_370 F_390 F_Z),
1650	INSTR(0x5c, "m", IF_RXa, F_370 F_390 F_Z),
1651	INSTR(0x5d, "d", IF_RXa, F_370 F_390 F_Z),
1652	INSTR(0x5e, "al", IF_RXa, F_370 F_390 F_Z),
1653	INSTR(0x5f, "sl", IF_RXa, F_370 F_390 F_Z),
1654	INSTR(0x60, "std", IF_RXa, F_370 F_390 F_Z),
1655	INSTR(0x67, "mxd", IF_RXa, F_370 F_390 F_Z),
1656	INSTR(0x68, "ld", IF_RXa, F_370 F_390 F_Z),
1657	INSTR(0x69, "cd", IF_RXa, F_370 F_390 F_Z),
1658	INSTR(0x6a, "ad", IF_RXa, F_370 F_390 F_Z),
1659	INSTR(0x6b, "sd", IF_RXa, F_370 F_390 F_Z),
1660	INSTR(0x6c, "md", IF_RXa, F_370 F_390 F_Z),
1661	INSTR(0x6d, "dd", IF_RXa, F_370 F_390 F_Z),
1662	INSTR(0x6e, "aw", IF_RXa, F_370 F_390 F_Z),
1663	INSTR(0x6f, "sw", IF_RXa, F_370 F_390 F_Z),
1664	INSTR(0x70, "ste", IF_RXa, F_370 F_390 F_Z),
1665	INSTR(0x71, "ms", IF_RXa, F_390 F_Z),
1666	INSTR(0x78, "le", IF_RXa, F_370 F_390 F_Z),
1667	INSTR(0x79, "ce", IF_RXa, F_370 F_390 F_Z),
1668	INSTR(0x7a, "ae", IF_RXa, F_370 F_390 F_Z),
1669	INSTR(0x7b, "se", IF_RXa, F_370 F_390 F_Z),
1670	MULTI(0x7c, tbl_7c),
1671	INSTR(0x7d, "de", IF_RXa, F_370 F_390 F_Z),
1672	INSTR(0x7e, "au", IF_RXa, F_370 F_390 F_Z),
1673	INSTR(0x7f, "su", IF_RXa, F_370 F_390 F_Z),
1674	INSTR(0x80, "ssm", IF_S, F_370 F_390 F_Z),
1675	INSTR(0x82, "lpw", IF_S, F_370 F_390 F_Z),
1676	INSTR(0x83, "diag", IF_DIAG, F_370 F_390 F_Z),
1677	MULTI(0x84, tbl_84),
1678	MULTI(0x85, tbl_85),
1679	INSTR(0x86, "bxh", IF_RSa, F_370 F_390 F_Z),
1680	INSTR(0x87, "bxle", IF_RSa, F_370 F_390 F_Z),
1681	INSTR(0x88, "srl", IF_RSa, F_370 F_390 F_Z),
1682	INSTR(0x89, "sll", IF_RSa, F_370 F_390 F_Z),
1683	INSTR(0x8a, "sra", IF_RSa, F_370 F_390 F_Z),
1684	INSTR(0x8b, "sla", IF_RSa, F_370 F_390 F_Z),
1685	INSTR(0x8c, "srdl", IF_RSa, F_370 F_390 F_Z),
1686	INSTR(0x8d, "sldl", IF_RSa, F_370 F_390 F_Z),
1687	INSTR(0x8e, "srda", IF_RSa, F_370 F_390 F_Z),
1688	INSTR(0x8f, "slda", IF_RSa, F_370 F_390 F_Z),
1689	INSTR(0x90, "stm", IF_RSa, F_370 F_390 F_Z),
1690	INSTR(0x91, "tm", IF_SI, F_370 F_390 F_Z),
1691	INSTR(0x92, "mvi", IF_SI, F_370 F_390 F_Z),
1692	INSTR(0x93, "ts", IF_S, F_370 F_390 F_Z),
1693	INSTR(0x94, "ni", IF_SI, F_370 F_390 F_Z),
1694	INSTR(0x95, "cli", IF_SI, F_370 F_390 F_Z),
1695	INSTR(0x96, "oi", IF_SI, F_370 F_390 F_Z),
1696	INSTR(0x97, "xi", IF_SI, F_370 F_390 F_Z),
1697	INSTR(0x98, "lm", IF_RSa, F_370 F_390 F_Z),
1698	INSTR(0x99, "trace", IF_RSa, F_390 F_Z),
1699	INSTR(0x9a, "lam", IF_RSa, F_390 F_Z),
1700	INSTR(0x9b, "stam", IF_RSa, F_390 F_Z),
1701	TABLE(0xa5, tbl_a5x, 1, 0, 0x0f),
1702	TABLE(0xa7, tbl_a7x, 1, 0, 0x0f),
1703	INSTR(0xa8, "mvcle", IF_RSa, F_390 F_Z),
1704	INSTR(0xa9, "clcle", IF_RSa, F_390 F_Z),
1705	INSTR(0xac, "stnsm", IF_SI, F_370 F_390 F_Z),
1706	INSTR(0xad, "stosm", IF_SI, F_370 F_390 F_Z),
1707	INSTR(0xae, "sigp", IF_RSa, F_370 F_390 F_Z),
1708	INSTR(0xaf, "mc", IF_SI, F_370 F_390 F_Z),
1709	INSTR(0xb1, "lra", IF_RXa, F_370 F_390 F_Z),
1710	TABLE(0xb2, tbl_b2xx, 1, 0, 0xff),
1711	TABLE(0xb3, tbl_b3xx, 1, 0, 0xff),

```

1712 INSTR(0xb6, "stctl", IF_RSa, F_370 | F_390 | F_Z | F_CTL_REG),
1713 INSTR(0xb7, "lctl", IF_RSa, F_370 | F_390 | F_Z | F_CTL_REG),
1714 TABLE(0xb9, tbl_b9xx, 1, 0, 0xff),
1715 INSTR(0xba, "cs", IF_RSa, F_370 | F_390 | F_Z),
1716 INSTR(0xbb, "cds", IF_RSa, F_370 | F_390 | F_Z),
1717 INSTR(0xbd, "clm", IF_Rsb, F_370 | F_390 | F_Z),
1718 INSTR(0xbe, "stcm", IF_Rsb, F_370 | F_390 | F_Z),
1719 INSTR(0xbf, "icm", IF_Rsb, F_370 | F_390 | F_Z),
1720 TABLE(0xc0, tbl_c0x, 1, 0, 0x0f),
1721 TABLE(0xc2, tbl_c2x, 1, 0, 0x0f),
1722 TABLE(0xc4, tbl_c4x, 1, 0, 0x0f),
1723 INSTR(0xc5, "bprp", IF_MII, F_Z),
1724 TABLE(0xc6, tbl_c6x, 1, 0, 0x0f),
1725 INSTR(0xc7, "bpp", IF_SMI, F_Z),
1726 TABLE(0xc8, tbl_c8x, 1, 0, 0x0f),
1727 TABLE(0xcc, tbl_ccx, 1, 0, 0x0f),
1728 INSTR(0xd0, "trtr", IF_Ssa, F_Z),
1729 INSTR(0xd1, "mvn", IF_Ssa, F_370 | F_390 | F_Z),
1730 INSTR(0xd2, "mvc", IF_Ssa, F_370 | F_390 | F_Z),
1731 INSTR(0xd3, "mvz", IF_Ssa, F_370 | F_390 | F_Z),
1732 INSTR(0xd4, "nc", IF_Ssa, F_370 | F_390 | F_Z),
1733 INSTR(0xd5, "clc", IF_Ssa, F_370 | F_390 | F_Z),
1734 INSTR(0xd6, "oc", IF_Ssa, F_370 | F_390 | F_Z),
1735 INSTR(0xd7, "xc", IF_Ssa, F_370 | F_390 | F_Z),
1736 INSTR(0xd9, "mvck", IF_Ssd, F_390 | F_Z),
1737 INSTR(0xda, "mvcp", IF_Ssd, F_390 | F_Z),
1738 INSTR(0xdb, "mvcs", IF_Ssd, F_390 | F_Z),
1739 INSTR(0xdc, "tr", IF_Ssa, F_370 | F_390 | F_Z),
1740 INSTR(0xdd, "trt", IF_Ssa, F_370 | F_390 | F_Z),
1741 INSTR(0xde, "ed", IF_Ssa, F_370 | F_390 | F_Z),
1742 INSTR(0xdf, "edmk", IF_Ssa, F_370 | F_390 | F_Z),
1743 INSTR(0xe1, "pku", IF_Ssf, F_390 | F_Z),
1744 INSTR(0xe2, "unpku", IF_Ssa, F_390 | F_Z),
1745 TABLE(0xe3, tbl_e3xx, 5, 0, 0xff),
1746 TABLE(0xe5, tbl_e5xx, 1, 0, 0xff),
1747 INSTR(0xe8, "mvcin", IF_Ssa, F_390 | F_Z),
1748 INSTR(0xe9, "pka", IF_Ssf, F_390 | F_Z),
1749 INSTR(0xea, "unpka", IF_Ssa, F_390 | F_Z),
1750 TABLE(0xeb, tbl_ebxx, 5, 0, 0xff),
1751 TABLE(0xec, tbl_ecxx, 5, 0, 0xff),
1752 TABLE(0xed, tbl_edxx, 5, 0, 0xff),
1753 INSTR(0xee, "plo", IF_Sse, F_390 | F_Z),
1754 INSTR(0xef, "lmd", IF_Sse, F_Z),
1755 INSTR(0xf0, "srp", IF_Ssc, F_370 | F_390 | F_Z),
1756 INSTR(0xf1, "mvo", IF_Ssb, F_370 | F_390 | F_Z),
1757 INSTR(0xf2, "pack", IF_Ssb, F_370 | F_390 | F_Z),
1758 INSTR(0xf3, "unpk", IF_Ssb, F_370 | F_390 | F_Z),
1759 INSTR(0xf8, "zap", IF_Ssb, F_370 | F_390 | F_Z),
1760 INSTR(0xf9, "cp", IF_Ssb, F_370 | F_390 | F_Z),
1761 INSTR(0xfa, "ap", IF_Ssb, F_370 | F_390 | F_Z),
1762 INSTR(0xfb, "sp", IF_Ssb, F_370 | F_390 | F_Z),
1763 INSTR(0xfc, "mp", IF_Ssb, F_370 | F_390 | F_Z),
1764 INSTR(0xfd, "dp", IF_Ssb, F_370 | F_390 | F_Z),
1765 };
1766 /* END CSTYLED */

1768 /* how masks are printed */
1769 static const char *M[16] = {
1770     "0", "1", "2", "3", "4", "5", "6", "7",
1771     "8", "9", "10", "11", "12", "13", "14", "15",
1772 };

1774 /* how general purpose regs are printed */
1775 static const char *R[16] = {
1776     "%r0", "%r1", "%r2", "%r3", "%r4", "%r5", "%r6", "%r7",
1777     "%r8", "%r9", "%r10", "%r11", "%r12", "%r13", "%r14", "%r15",

```

```

1778 };

1780 /* how control regs are printed */
1781 static const char *C[16] = {
1782     "%c0", "%c1", "%c2", "%c3", "%c4", "%c5", "%c6", "%c7",
1783     "%c8", "%c9", "%c10", "%c11", "%c12", "%c13", "%c14", "%c15",
1784 };

1786 /* B and X registers are still registers - print them the same way */
1787 #define B      R
1788 #define X      R

1790 static inline uint32_t
1791 val_8_4_8(uint32_t hi, uint32_t mid, uint32_t lo)
1792 {
1793     return ((hi << 12) | (mid << 8) | lo);
1794 }

1796 static inline uint32_t
1797 val_16_16(uint32_t hi, uint32_t lo)
1798 {
1799     return ((BE_16(hi) << 16) | BE_16(lo));
1800 }

1802 static inline int32_t
1803 sval_16_16(uint32_t hi, uint32_t lo)
1804 {
1805     return (val_16_16(hi, lo));
1806 }

1808 static inline uint32_t
1809 val_8_16(uint32_t hi, uint32_t lo)
1810 {
1811     return ((hi << 16) | BE_16(lo));
1812 }

1814 static inline int32_t
1815 sval_8_16(uint32_t hi, uint32_t lo)
1816 {
1817     int32_t tmp = val_8_16(hi, lo);
1818
1819     if (tmp & 0x00800000)
1820         return (0xffff0000 | tmp);
1821     return (tmp);
1822 }

1824 static inline uint32_t
1825 val_4_8(uint32_t hi, uint32_t lo)
1826 {
1827     return ((hi << 8) | lo);
1828 }

1830 static inline int32_t
1831 sval_4_8(uint32_t hi, uint32_t lo)
1832 {
1833     uint32_t tmp = val_4_8(hi, lo);
1834
1835     if (tmp & 0x800)
1836         return (0xfffff000 | tmp);
1837     return (tmp);
1838 }

1840 /* ARGSUSED */
1841 static void
1842 fmt_zero(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1843 {

```

```

1844     (void) snprintf(buf, buflen, "0x00, 0x00");
1845 }

1847 /* ARGSUSED */
1848 static void
1849 fmt_diag(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1850 {
1851     (void) snprintf(buf, buflen, "%#x",
1852                    val_8_16(inst->diag.par1, inst->diag.par2));
1853 }

1855 /* ARGSUSED */
1856 static void
1857 fmt_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1858 {
1859     /* nothing to do */
1860 }

1862 /* ARGSUSED */
1863 static void
1864 fmt_i(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1865 {
1866     (void) snprintf(buf, buflen, "%#x", inst->i.i);
1867 }

1869 /* ARGSUSED */
1870 static void
1871 fmt_ie(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1872 {
1873     (void) snprintf(buf, buflen, "%u,%u", inst->ie.i1, inst->ie.i2);
1874 }

1876 /* ARGSUSED */
1877 static void
1878 fmt_mii(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1879 {
1880     uint64_t ri2 = addr + 2 * sval_4_8(inst->mii.ri2h, inst->mii.ri2l);
1881     uint64_t ri3 = addr + 2 * sval_8_16(inst->mii.ri3h, inst->mii.ri3l);

1883     (void) snprintf(buf, buflen, "%s,%#x,%#x", M[inst->mii.m1], ri2, ri3);
1884 }

1886 /* ARGSUSED */
1887 static void
1888 fmt_ril_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1889 {
1890     (void) snprintf(buf, buflen, "%s,%u", R[inst->ril_a.r1],
1891                    val_16_16(inst->ril_a.i2h, inst->ril_a.i2l));
1892 }

1894 /* ARGSUSED */
1895 static void
1896 fmt_ril_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1897 {
1898     uint64_t ri2 = addr + 2 *
1899         sval_16_16(inst->ril_b.ri2h, inst->ril_b.ri2l);

1901     (void) snprintf(buf, buflen, "%s,%#x", R[inst->ril_b.r1], ri2);
1902 }

1904 /* ARGSUSED */
1905 static void
1906 fmt_ril_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1907 {
1908     uint64_t ri2 = addr + 2 *
1909         sval_16_16(inst->ril_c.ri2h, inst->ril_c.ri2l);

```

```

1911     (void) snprintf(buf, buflen, "%s,%#x", M[inst->ril_c.m1], ri2);
1912 }

1914 /* ARGSUSED */
1915 static void
1916 fmt_ris(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1917 {
1918     uint32_t d4 = val_4_8(inst->ris.d4h, inst->ris.d4l);

1920     (void) snprintf(buf, buflen, "%s,%u,%s,%u(%s)",
1921                    R[inst->ris.r1], inst->ris.i2, M[inst->ris.m3], d4,
1922                    B[inst->ris.b4]);
1923 }

1925 /* ARGSUSED */
1926 static void
1927 fmt_ri_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1928 {
1929     uint16_t i2 = BE_16(inst->ri_a.i2);

1931     if (flags & F_SIGNED_IMM)
1932         (void) snprintf(buf, buflen, "%s,%d", R[inst->ri_a.r1],
1933                        (int16_t)i2);
1934     else
1935         (void) snprintf(buf, buflen, "%s,%u", R[inst->ri_a.r1],
1936                        i2);
1937 }

1939 /* ARGSUSED */
1940 static void
1941 fmt_ri_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1942 {
1943     uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->ri_b.ri2);

1945     (void) snprintf(buf, buflen, "%s,%#x", R[inst->ri_b.r1], ri2);
1946 }

1948 static void
1949 fmt_ri_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1950 {
1951     uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->ri_c.ri2);

1953     if (flags & F_HIDE_MASK)
1954         (void) snprintf(buf, buflen, "%#x", ri2);
1955     else
1956         (void) snprintf(buf, buflen, "%s,%#x", M[inst->ri_c.m1], ri2);
1957 }

1959 /* ARGSUSED */
1960 static void
1961 fmt_rie_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1962 {
1963     (void) snprintf(buf, buflen, "%s,%u,%s", R[inst->rie_a.r1],
1964                    BE_16(inst->rie_a.i2), M[inst->rie_a.m3]);
1965 }

1967 /* ARGSUSED */
1968 static void
1969 fmt_rie_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1970 {
1971     uint64_t ri4 = addr + 2 * (int16_t)BE_16(inst->rie_b.ri4);

1973     (void) snprintf(buf, buflen, "%s,%s,%s,%#x", R[inst->rie_b.r1],
1974                    R[inst->rie_b.r2], M[inst->rie_b.m3], ri4);
1975 }

```

```

1977 /* ARGSUSED */
1978 static void
1979 fmt_rie_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1980 {
1981     uint64_t ri4 = addr + 2 * (int16_t)BE_16(inst->rie_c.ri4);
1982
1983     (void) snprintf(buf, buflen, "%s,%u,%s,%#x", R[inst->rie_c.r1],
1984                   inst->rie_c.i2, M[inst->rie_c.m3], ri4);
1985 }
1986
1987 /* ARGSUSED */
1988 static void
1989 fmt_rie_d(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1990 {
1991     (void) snprintf(buf, buflen, "%s,%s,%u", R[inst->rie_d.r1],
1992                   R[inst->rie_d.r3], BE_16(inst->rie_d.i2));
1993 }
1994
1995 /* ARGSUSED */
1996 static void
1997 fmt_rie_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1998 {
1999     uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->rie_e.ri2);
2000
2001     (void) snprintf(buf, buflen, "%s,%s,%#x", R[inst->rie_e.r1],
2002                   R[inst->rie_e.r3], ri2);
2003 }
2004
2005 /* ARGSUSED */
2006 static void
2007 fmt_rie_f(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2008 {
2009     (void) snprintf(buf, buflen, "%s,%s,%u,%u,%u", R[inst->rie_f.r1],
2010                   R[inst->rie_f.r2], inst->rie_f.i3, inst->rie_f.i4,
2011                   inst->rie_f.i5);
2012 }
2013
2014 /* ARGSUSED */
2015 static void
2016 fmt_rre(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2017 {
2018     (void) snprintf(buf, buflen, "%s,%s", R[inst->rre.r1], R[inst->rre.r2]);
2019 }
2020
2021 /* ARGSUSED */
2022 static void
2023 fmt_rrf_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2024 {
2025     (void) snprintf(buf, buflen, "%s,%s,%s",
2026                   R[inst->rrf_ab.r1], R[inst->rrf_ab.r2], R[inst->rrf_ab.r3]);
2027 }
2028
2029 /* ARGSUSED */
2030 static void
2031 fmt_rrf_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2032 {
2033     (void) snprintf(buf, buflen, "%s,%s,%s",
2034                   R[inst->rrf_ab.r1], R[inst->rrf_ab.r3], R[inst->rrf_ab.r2]);
2035 }
2036
2037 /* ARGSUSED */
2038 static void
2039 fmt_rrf_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2040 {
2041     (void) snprintf(buf, buflen, "%s,%s,%s",

```

```

2042     R[inst->rrf_cde.r1], R[inst->rrf_cde.r2], M[inst->rrf_cde.m3]);
2043 }
2044
2045 /* ARGSUSED */
2046 static void
2047 fmt_rrf_d(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_cde.r1], R[inst->rrf_cde.r2], M[inst->rrf_cde.m4]);
2051 }
2052
2053 /* ARGSUSED */
2054 static void
2055 fmt_rrf_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2056 {
2057     (void) snprintf(buf, buflen, "%s,%s,%s,%s",
2058                   R[inst->rrf_cde.r1], M[inst->rrf_cde.m3],
2059                   R[inst->rrf_cde.r2], M[inst->rrf_cde.m4]);
2060 }
2061
2062 /* ARGSUSED */
2063 static void
2064 fmt_rrs(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2065 {
2066     (void) snprintf(buf, buflen, "%s,%s,%s,%u(%s)", R[inst->rrs.r1],
2067                   R[inst->rrs.r2], M[inst->rrs.m3],
2068                   val_4_8(inst->rrs.d4h, inst->rrs.d4l), B[inst->rrs.b4]);
2069 }
2070
2071 /* ARGSUSED */
2072 static void
2073 fmt_rr(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2074 {
2075     /* a branch uses r1 as a mask */
2076     if (flags & F_HIDE_MASK)
2077         (void) snprintf(buf, buflen, "%s", R[inst->rr.r2]);
2078     else if (flags & F_R1_IS_MASK)
2079         (void) snprintf(buf, buflen, "%s,%s", M[inst->rr.r1],
2080                       R[inst->rr.r2]);
2081     else
2082         (void) snprintf(buf, buflen, "%s,%s", R[inst->rr.r1],
2083                       R[inst->rr.r2]);
2084 }
2085
2086 /* ARGSUSED */
2087 static void
2088 fmt_rrd(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2089 {
2090     (void) snprintf(buf, buflen, "%s,%s,%s", R[inst->rrd.r1],
2091                   R[inst->rrd.r3], R[inst->rrd.r2]);
2092 }
2093
2094 /* ARGSUSED */
2095 static void
2096 fmt_rx_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2097 {
2098     uint32_t d2 = val_4_8(inst->rx_a.d2h, inst->rx_b.d2l);
2099
2100     (void) snprintf(buf, buflen, "%s,%u(%s,%s)", R[inst->rx_a.r1],
2101                   d2, X[inst->rx_a.x2], B[inst->rx_a.b2]);
2102 }
2103
2104 /* ARGSUSED */
2105 static void
2106 fmt_rx_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2107 {

```



```

2108     uint32_t d2 = val_4_8(inst->rx_b.d2h, inst->rx_b.d2l);
2110     if (flags & F_HIDE_MASK)
2111         (void) snprintf(buf, buflen, "%u(%s,%s)",
2112             d2, X[inst->rx_b.x2], B[inst->rx_b.b2]);
2113     else
2114         (void) snprintf(buf, buflen, "%s,%u(%s,%s)", M[inst->rx_b.m1],
2115             d2, X[inst->rx_b.x2], B[inst->rx_b.b2]);
2116 }
2118 /* ARGSUSED */
2119 static void
2120 fmt_rxe(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2121 {
2122     uint32_t d2 = val_4_8(inst->rxe.d2h, inst->rxe.d2l);
2124     (void) snprintf(buf, buflen, "%s,%u(%s,%s)",
2125         R[inst->rxe.r1], d2, X[inst->rxe.x2], B[inst->rxe.b2]);
2126 }
2128 /* ARGSUSED */
2129 static void
2130 fmt_rxf(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2131 {
2132     uint32_t d2 = val_4_8(inst->rxf.d2h, inst->rxf.d2l);
2134     (void) snprintf(buf, buflen, "%s,%s,%u(%s,%s)",
2135         R[inst->rxf.r1], R[inst->rxf.r3], d2, X[inst->rxf.x2],
2136         B[inst->rxf.b2]);
2137 }
2139 /* ARGSUSED */
2140 static void
2141 fmt_rxy_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2142 {
2143     uint32_t d2;
2145     d2 = val_8_4_8(inst->rxy_a.dh2, inst->rxy_a.dl2h, inst->rxy_a.dl2l);
2147     (void) snprintf(buf, buflen, "%s,%u(%s,%s)",
2148         R[inst->rxy_a.r1], d2, X[inst->rxy_a.x2], B[inst->rxy_a.b2]);
2149 }
2151 /* ARGSUSED */
2152 static void
2153 fmt_rxy_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2154 {
2155     uint32_t d2;
2157     d2 = val_8_4_8(inst->rxy_b.dh2, inst->rxy_b.dl2h, inst->rxy_b.dl2l);
2159     (void) snprintf(buf, buflen, "%s,%u(%s,%s)",
2160         M[inst->rxy_b.m1], d2, X[inst->rxy_b.x2], B[inst->rxy_b.b2]);
2161 }
2163 /* ARGSUSED */
2164 static void
2165 fmt_rs_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2166 {
2167     const char *r1, *r3;
2169     if (flags & F_CTL_REG) {
2170         r1 = C[inst->rs_a.r1];
2171         r3 = C[inst->rs_a.r3];
2172     } else {
2173         r1 = R[inst->rs_a.r1];

```

```

2174         r3 = R[inst->rs_a.r3];
2175     }
2177     (void) snprintf(buf, buflen, "%s,%s,%u(%s)", r1, r3,
2178         val_4_8(inst->rs_a.d2h, inst->rs_a.d2l), B[inst->rs_a.b2]);
2179 }
2181 /* ARGSUSED */
2182 static void
2183 fmt_rs_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2184 {
2185     (void) snprintf(buf, buflen, "%s,%s,%u(%s)", R[inst->rs_b.r1],
2186         M[inst->rs_b.m3], val_4_8(inst->rs_b.d2h, inst->rs_b.d2l),
2187         B[inst->rs_b.b2]);
2188 }
2190 /* ARGSUSED */
2191 static void
2192 fmt_rsl_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2193 {
2194     (void) snprintf(buf, buflen, "%u(%u,%s)",
2195         val_4_8(inst->rsl_a.dlh, inst->rsl_a.dll), inst->rsl_a.l1,
2196         B[inst->rsl_a.bl]);
2197 }
2199 /* ARGSUSED */
2200 static void
2201 fmt_rsl_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2202 {
2203     (void) snprintf(buf, buflen, "%s,%u(%u,%s),%s",
2204         R[inst->rsl_b.r1],
2205         val_4_8(inst->rsl_b.d2h, inst->rsl_b.d2l), inst->rsl_b.l2,
2206         B[inst->rsl_b.b2], M[inst->rsl_b.m3]);
2207 }
2209 /* ARGSUSED */
2210 static void
2211 fmt_rsy_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2212 {
2213     const char *r1, *r3;
2214     uint32_t d2;
2216     d2 = val_8_4_8(inst->rsy_a.dh2, inst->rsy_a.dl2h, inst->rsy_a.dl2l);
2218     if (flags & F_CTL_REG) {
2219         r1 = C[inst->rsy_a.r1];
2220         r3 = C[inst->rsy_a.r3];
2221     } else {
2222         r1 = R[inst->rsy_a.r1];
2223         r3 = R[inst->rsy_a.r3];
2224     }
2226     (void) snprintf(buf, buflen, "%s,%s,%u(%s)", r1, r3, d2,
2227         B[inst->rsy_a.b2]);
2228 }
2230 /* ARGSUSED */
2231 static void
2232 fmt_rsy_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2233 {
2234     uint32_t d2;
2236     d2 = val_8_4_8(inst->rsy_b.dh2, inst->rsy_b.dl2h, inst->rsy_b.dl2l);
2238     (void) snprintf(buf, buflen, "%s,%s,%u(%s)",
2239         R[inst->rsy_b.r1], M[inst->rsy_b.m3],

```

```

2240         d2, B[inst->rsy_b.b2]);
2241 }

2243 /* ARGSUSED */
2244 static void
2245 fmt_rsi(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2246 {
2247     uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->rsi.ri2);

2249     (void) snprintf(buf, buflen, "%s,%s,%#x", R[inst->rsi.r1],
2250                    R[inst->rsi.r3], ri2);
2251 }

2253 /* ARGSUSED */
2254 static void
2255 fmt_si(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2256 {
2257     uint32_t d1 = val_4_8(inst->si.dlh, inst->si.d1l);

2259     (void) snprintf(buf, buflen, "%u(%s),%u", d1, B[inst->si.b1],
2260                    inst->si.i2);
2261 }

2263 /* ARGSUSED */
2264 static void
2265 fmt_sil(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2266 {
2267     (void) snprintf(buf, buflen, "%u(%s),%u",
2268                    val_4_8(inst->sil.dlh, inst->sil.d1l), B[inst->sil.b1],
2269                    BE_16(inst->sil.i2));
2270 }

2272 /* ARGSUSED */
2273 static void
2274 fmt_siy(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2275 {
2276     (void) snprintf(buf, buflen, "%u(%s),%u",
2277                    val_8_4_8(inst->siy.dh1, inst->siy.d1l1, inst->siy.d1l1h, inst->siy.d1l1l),
2278                    B[inst->siy.b1], inst->siy.i2);
2279 }

2281 /* ARGSUSED */
2282 static void
2283 fmt_smi(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2284 {
2285     uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->smi.ri2);

2287     (void) snprintf(buf, buflen, "%s,%#x,%u(%s)", M[inst->smi.m1], ri2,
2288                    val_4_8(inst->smi.d3h, inst->smi.d3l), B[inst->smi.b3]);
2289 }

2291 /* ARGSUSED */
2292 static void
2293 fmt_s(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2294 {
2295     uint32_t d = val_4_8(inst->s.d2h, inst->s.d2l);

2297     (void) snprintf(buf, buflen, "%u(%s)", d, B[inst->s.b2]);
2298 }

2300 /* ARGSUSED */
2301 static void
2302 fmt_ss_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2303 {
2304     uint32_t d1, d2;

```

```

2306     d1 = val_4_8(inst->ss_a.d1h, inst->ss_a.d1l);
2307     d2 = val_4_8(inst->ss_a.d2h, inst->ss_a.d2l);

2309     (void) snprintf(buf, buflen, "%u(%u,%s),%u(%s)",
2310                    d1, inst->ss_a.l + 1, B[inst->ss_a.b1],
2311                    d2, B[inst->ss_a.b2]);
2312 }

2314 /* ARGSUSED */
2315 static void
2316 fmt_ss_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2317 {
2318     uint32_t d1, d2;

2320     d1 = val_4_8(inst->ss_b.d1h, inst->ss_b.d1l);
2321     d2 = val_4_8(inst->ss_b.d2h, inst->ss_b.d2l);

2323     (void) snprintf(buf, buflen, "%u(%u,%s),%u(%u,%s)",
2324                    d1, inst->ss_b.l1 + 1, B[inst->ss_b.b1],
2325                    d2, inst->ss_b.l2 + 1, B[inst->ss_b.b2]);
2326 }

2328 /* ARGSUSED */
2329 static void
2330 fmt_ss_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2331 {
2332     uint32_t d1, d2;

2334     d1 = val_4_8(inst->ss_c.d1h, inst->ss_c.d1l);
2335     d2 = val_4_8(inst->ss_c.d2h, inst->ss_c.d2l);

2337     (void) snprintf(buf, buflen, "%u(%u,%s),%u(%s),%u",
2338                    d1, inst->ss_c.l1, B[inst->ss_c.b1],
2339                    d2, B[inst->ss_c.b2], inst->ss_c.i3);
2340 }

2342 /* ARGSUSED */
2343 static void
2344 fmt_ss_d(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2345 {
2346     uint32_t d1, d2;

2348     d1 = val_4_8(inst->ss_d.d1h, inst->ss_d.d1l);
2349     d2 = val_4_8(inst->ss_d.d2h, inst->ss_d.d2l);

2351     (void) snprintf(buf, buflen, "%u(%s,%s),%u(%s),%s",
2352                    d1, R[inst->ss_d.r1], B[inst->ss_d.b1],
2353                    d2, B[inst->ss_d.b2], R[inst->ss_d.r3]);
2354 }

2356 /* ARGSUSED */
2357 static void
2358 fmt_ss_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2359 {
2360     uint32_t d2, d4;

2362     d2 = val_4_8(inst->ss_e.d2h, inst->ss_e.d2l);
2363     d4 = val_4_8(inst->ss_e.d4h, inst->ss_e.d4l);

2365     (void) snprintf(buf, buflen, "%s,%u(%s),%s,%u(%s)",
2366                    R[inst->ss_e.r1], d2, B[inst->ss_e.b2],
2367                    R[inst->ss_e.r3], d4, B[inst->ss_e.b4]);
2368 }

2370 /* ARGSUSED */
2371 static void

```

```

2372 fmt_ss_f(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2373 {
2374     uint32_t d1, d2;

2376     d1 = val_4_8(inst->ss.f.d1h, inst->ss.f.d1l);
2377     d2 = val_4_8(inst->ss.f.d2h, inst->ss.f.d2l);

2379     (void) snprintf(buf, buflen, "%u(%s),%u(%u,%s)",
2380                    d1, B[inst->ss.f.b1], d2, inst->ss.f.l2,
2381                    B[inst->ss.f.b2]);
2382 }

2384 /* ARGSUSED */
2385 static void
2386 fmt_sse(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2387 {
2388     uint32_t d1 = val_4_8(inst->sse.d1h, inst->sse.d1l);
2389     uint32_t d2 = val_4_8(inst->sse.d2h, inst->sse.d2l);

2391     (void) snprintf(buf, buflen, "%u(%s),%u(%s)",
2392                    d1, B[inst->sse.b1], d2, B[inst->sse.b2]);
2393 }

2395 /* ARGSUSED */
2396 static void
2397 fmt_ssf(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2398 {
2399     uint32_t d1 = val_4_8(inst->ssf.d1h, inst->ssf.d1l);
2400     uint32_t d2 = val_4_8(inst->ssf.d2h, inst->ssf.d2l);

2402     (void) snprintf(buf, buflen, "%u(%s),%u(%s),%s",
2403                    d1, B[inst->ssf.b1],
2404                    d2, B[inst->ssf.b2], R[inst->ssf.r3]);
2405 }

2407 static void (*fmt_fxns[IF_NFMTS])(uint64_t, union inst *, char *, size_t,
2408 int) = {
2409     [IF_ZERO] = fmt_zero,
2410     [IF_DIAG] = fmt_diag,
2411     [IF_E] = fmt_e,
2412     [IF_I] = fmt_i,
2413     [IF_IE] = fmt_ie,
2414     [IF_MII] = fmt_mii,
2415     [IF_RIa] = fmt_ri_a,
2416     [IF_RIb] = fmt_ri_b,
2417     [IF_RIc] = fmt_ri_c,
2418     [IF_RIEa] = fmt_rie_a,
2419     [IF_RIEb] = fmt_rie_b,
2420     [IF_RIEc] = fmt_rie_c,
2421     [IF_RIED] = fmt_rie_d,
2422     [IF_RIEe] = fmt_rie_e,
2423     [IF_RIEf] = fmt_rie_f,
2424     [IF_RILa] = fmt_ril_a,
2425     [IF_RILb] = fmt_ril_b,
2426     [IF_RILc] = fmt_ril_c,
2427     [IF_RIS] = fmt_ris,
2428     [IF_RR] = fmt_rr,
2429     [IF_RRD] = fmt_rrd,
2430     [IF_RRE] = fmt_rre,
2431     [IF_RRFa] = fmt_rrf_a,
2432     [IF_RRFb] = fmt_rrf_b,
2433     [IF_RRFc] = fmt_rrf_c,
2434     [IF_RRFd] = fmt_rrf_d,
2435     [IF_RRFe] = fmt_rrf_e,
2436     [IF_RRS] = fmt_rrs,
2437     [IF_RSa] = fmt_rs_a,

```

```

2438     [IF_RSb] = fmt_rs_b,
2439     [IF_RSI] = fmt_rsi,
2440     [IF_RSLa] = fmt_rsl_a,
2441     [IF_RSLb] = fmt_rsl_b,
2442     [IF_RSYa] = fmt_rsy_a,
2443     [IF_RSYb] = fmt_rsy_b,
2444     [IF_RXa] = fmt_rx_a,
2445     [IF_RXb] = fmt_rx_b,
2446     [IF_RXE] = fmt_rxe,
2447     [IF_RXF] = fmt_rxf,
2448     [IF_RXYa] = fmt_rxy_a,
2449     [IF_RXYb] = fmt_rxy_b,
2450     [IF_S] = fmt_s,
2451     [IF_SI] = fmt_si,
2452     [IF_SIL] = fmt_sil,
2453     [IF_SIY] = fmt_siy,
2454     [IF_SMI] = fmt_smi,
2455     [IF_SSa] = fmt_ss_a,
2456     [IF_Ssb] = fmt_ss_b,
2457     [IF_SSc] = fmt_ss_c,
2458     [IF_Ssd] = fmt_ss_d,
2459     [IF_SSe] = fmt_ss_e,
2460     [IF_Ssf] = fmt_ss_f,
2461     [IF_SSE] = fmt_sse,
2462     [IF_SSF] = fmt_ssf,
2463 };

2465 static int
2466 dis_s390(uint64_t addr, union inst *inst, char *buf, size_t buflen, int mach)
2467 {
2468     const struct inst_table *tbl = &tbl_xx[inst->raw[0]];
2469     int tmp;

2471     while (tbl->fmt == IF_TBL || tbl->fmt == IF_MULTI) {
2472         if (tbl->fmt == IF_TBL) {
2473             int idx;

2475             idx = inst->raw[tbl->u.table.off];
2476             idx >>= tbl->u.table.shift;
2477             idx &= tbl->u.table.mask;

2479             tbl = &tbl->u.table.ptr[idx];
2480         } else if (tbl->fmt == IF_MULTI) {
2481             tbl = &tbl->u.multi.ptr[mach];
2482         }
2483     }

2485     if (tbl->fmt == IF_INVALID)
2486         goto inval;

2488     if ((tbl->u.inst.flags & mach) == 0)
2489         goto inval;

2491     tmp = snprintf(buf, buflen, "%-7s ", tbl->u.inst.name);

2493     fmt_fxns[tbl->fmt](addr, inst, buf + tmp, buflen - tmp,
2494                      tbl->u.inst.flags);

2496     return (0);

2498 inval:
2499     (void) snprintf(buf, buflen, "??");

2501     /*
2502     * Even if we don't know how to disassemble the instruction, we know
2503     * how long it is, so we "succeed" even when we fail.

```

```

2504     */
2505     return (0);
2506 }

2508 static int
2509 dis_s390_supports_flags(int flags)
2510 {
2511     int archflags = flags & DIS_ARCH_MASK;

2513     if (archflags == DIS_S370 || archflags == DIS_S390_31 ||
2514         archflags == DIS_S390_64)
2515         return (1);

2517     return (0);
2518 }

2520 static int
2521 dis_s390_disassemble(dis_handle_t *dhp, uint64_t addr, char *buf,
2522                     size_t buflen)
2523 {
2524     union inst inst;
2525     int mach;
2526     int len;

2528     if (dhp->dh_read(dhp->dh_data, addr, &inst.raw[0], 2) != 2)
2529         return (-1);

2531     len = ILC2LEN(inst.raw[0] >> 6) - 2;

2533     if (len > 0 &&
2534         dhp->dh_read(dhp->dh_data, addr + 2, &inst.raw[2], len) != len)
2535         return (-1);

2537     switch (dhp->dh_flags & (DIS_S370 | DIS_S390_31 | DIS_S390_64)) {
2538     case DIS_S370:
2539         mach = F_370;
2540         break;
2541     case DIS_S390_31:
2542         mach = F_390;
2543         break;
2544     case DIS_S390_64:
2545         mach = F_Z;
2546         break;
2547     }

2549     return (dis_s390(addr, &inst, buf, buflen, mach));
2550 }

2552 /* ARGSUSED */
2553 static int
2554 dis_s390_min_instrlen(dis_handle_t *dhp)
2555 {
2556     return (2);
2557 }

2559 /* ARGSUSED */
2560 static int
2561 dis_s390_max_instrlen(dis_handle_t *dhp)
2562 {
2563     return (6);
2564 }

2566 dis_arch_t dis_arch_s390 = {
2567     .da_supports_flags = dis_s390_supports_flags,
2568     .da_disassemble = dis_s390_disassemble,
2569     .da_min_instrlen = dis_s390_min_instrlen,

```

```

2570     .da_max_instrlen = dis_s390_max_instrlen,
2571 };
2572 #endif /* ! codereview */

```

```

*****
7051 Thu Jul 23 08:02:31 2015
new/usr/src/lib/libdisasm/common/libdisasm.c
6066 dis: support for System/370, System/390, and z/Architecture ELF bins
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  * Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
26  * Copyright 2015 Nexenta Systems, Inc. All rights reserved.
27  */

29 #include <libdisasm.h>
30 #include <stdlib.h>
31 #ifdef DIS_STANDALONE
32 #include <mdb/mdb_modapi.h>
33 #define _MDB
34 #include <mdb/mdb_io.h>
35 #else
36 #include <stdio.h>
37 #endif

39 #include "libdisasm_impl.h"

41 static int _dis_errno;

43 /*
44  * If we're building the standalone library, then we only want to
45  * include support for disassembly of the native architecture.
46  * The regular shared library should include support for all
47  * architectures.
48  */
49 #if !defined(DIS_STANDALONE) || defined(__i386) || defined(__amd64)
50 extern dis_arch_t dis_arch_i386;
51 #endif
52 #if !defined(DIS_STANDALONE) || defined(__sparc)
53 extern dis_arch_t dis_arch_sparc;
54 #endif
55 #if !defined(DIS_STANDALONE) || defined(__s390) || defined(__s390x)
56 extern dis_arch_t dis_arch_s390;
57 #endif
58 #endif /* ! codereview */

60 static dis_arch_t *dis_archs[] = {
61 #if !defined(DIS_STANDALONE) || defined(__i386) || defined(__amd64)

```

```

62     &dis_arch_i386,
63 #endif
64 #if !defined(DIS_STANDALONE) || defined(__sparc)
65     &dis_arch_sparc,
66 #endif
67 #if !defined(DIS_STANDALONE) || defined(__s390) || defined(__s390x)
68     &dis_arch_s390,
69 #endif /* ! codereview */
70 #endif
71     NULL
72 };

74 /*
75  * For the standalone library, we need to link against mdb's malloc/free.
76  * Otherwise, use the standard malloc/free.
77  */
78 #ifdef DIS_STANDALONE
79 void *
80 dis_zalloc(size_t bytes)
81 {
82     return (mdb_zalloc(bytes, UM_SLEEP));
83 }

85 void
86 dis_free(void *ptr, size_t bytes)
87 {
88     mdb_free(ptr, bytes);
89 }
90 #else
91 void *
92 dis_zalloc(size_t bytes)
93 {
94     return (calloc(1, bytes));
95 }

97 /*ARGSUSED*/
98 void
99 dis_free(void *ptr, size_t bytes)
100 {
101     free(ptr);
102 }
103 #endif

105 int
106 dis_seterrno(int error)
107 {
108     _dis_errno = error;
109     return (-1);
110 }

112 int
113 dis_errno(void)
114 {
115     return (_dis_errno);
116 }

118 const char *
119 dis_strerror(int error)
120 {
121     switch (error) {
122     case E_DIS_NOMEM:
123         return ("out of memory");
124     case E_DIS_INVALIDFLAG:
125         return ("invalid flags for this architecture");
126     case E_DIS_UNSUPARCH:
127         return ("unsupported machine architecture");

```

```

128     default:
129         return ("unknown error");
130     }
131 }

133 void
134 dis_set_data(dis_handle_t *dhp, void *data)
135 {
136     dhp->dh_data = data;
137 }

139 void
140 dis_flags_set(dis_handle_t *dhp, int f)
141 {
142     dhp->dh_flags |= f;
143 }

145 void
146 dis_flags_clear(dis_handle_t *dhp, int f)
147 {
148     dhp->dh_flags &= ~f;
149 }

151 void
152 dis_handle_destroy(dis_handle_t *dhp)
153 {
154     if (dhp->dh_arch->da_handle_detach != NULL)
155         dhp->dh_arch->da_handle_detach(dhp);

157     dis_free(dhp, sizeof (dis_handle_t));
158 }

160 dis_handle_t *
161 dis_handle_create(int flags, void *data, dis_lookup_f lookup_func,
162                 dis_read_f read_func)
163 {
164     dis_handle_t *dhp;
165     dis_arch_t *arch = NULL;
166     int i;

168     /* Select an architecture based on flags */
169     for (i = 0; dis_archs[i] != NULL; i++) {
170         if (dis_archs[i]->da_supports_flags(flags)) {
171             arch = dis_archs[i];
172             break;
173         }
174     }
175     if (arch == NULL) {
176         (void) dis_seterrno(E_DIS_UNSUPARCH);
177         return (NULL);
178     }

180     if ((dhp = dis_zalloc(sizeof (dis_handle_t))) == NULL) {
181         (void) dis_seterrno(E_DIS_NOMEM);
182         return (NULL);
183     }
184     dhp->dh_arch = arch;
185     dhp->dh_lookup = lookup_func;
186     dhp->dh_read = read_func;
187     dhp->dh_flags = flags;
188     dhp->dh_data = data;

190     /*
191      * Allow the architecture-specific code to allocate
192      * its private data.
193      */

```

```

194     if (arch->da_handle_attach != NULL &&
195         arch->da_handle_attach(dhp) != 0) {
196         dis_free(dhp, sizeof (dis_handle_t));
197         /* dis errno already set */
198         return (NULL);
199     }

201     return (dhp);
202 }

204 int
205 dis_disassemble(dis_handle_t *dhp, uint64_t addr, char *buf, size_t buflen)
206 {
207     return (dhp->dh_arch->da_disassemble(dhp, addr, buf, buflen));
208 }

210 /*
211  * On some instruction sets (e.g., x86), we have no choice except to
212  * disassemble everything from the start of the symbol, and stop when we
213  * have reached our instruction address.  If we're not in the middle of a
214  * known symbol, then we return the same address to indicate failure.
215  */
216 static uint64_t
217 dis_generic_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
218 {
219     uint64_t *hist, addr, start;
220     int cur, nseen;
221     uint64_t res = pc;

223     if (n <= 0)
224         return (pc);

226     if (dhp->dh_lookup(dhp->dh_data, pc, NULL, 0, &start, NULL) != 0 ||
227         start == pc)
228         return (res);

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

232     for (cur = 0, nseen = 0, addr = start; addr < pc; addr = dhp->dh_addr) {
233         hist[cur] = addr;
234         cur = (cur + 1) % n;
235         nseen++;

237         /* if we cannot make forward progress, give up */
238         if (dis_disassemble(dhp, addr, NULL, 0) != 0)
239             goto done;
240     }

242     if (addr != pc) {
243         /*
244          * We scanned past %pc, but didn't find an instruction that
245          * started at %pc.  This means that either the caller specified
246          * an invalid address, or we ran into something other than code
247          * during our scan.  Virtually any combination of bytes can be
248          * construed as a valid Intel instruction, so any non-code bytes
249          * we encounter will have thrown off the scan.
250          */
251         goto done;
252     }

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

256 done:
257     dis_free(hist, sizeof (uint64_t) * n);
258     return (res);
259 }

```

```
261 /*
262  * Return the nth previous instruction's address. Return the same address
263  * to indicate failure.
264  */
265 uint64_t
266 dis_previnstr(dis_handle_t *dhp, uint64_t pc, int n)
267 {
268     if (dhp->dh_arch->da_previnstr == NULL)
269         return (dis_generic_previnstr(dhp, pc, n));
271     return (dhp->dh_arch->da_previnstr(dhp, pc, n));
272 }
274 int
275 dis_min_instrlen(dis_handle_t *dhp)
276 {
277     return (dhp->dh_arch->da_min_instrlen(dhp));
278 }
280 int
281 dis_max_instrlen(dis_handle_t *dhp)
282 {
283     return (dhp->dh_arch->da_max_instrlen(dhp));
284 }
286 static int
287 dis_generic_instrlen(dis_handle_t *dhp, uint64_t pc)
288 {
289     if (dis_disassemble(dhp, pc, NULL, 0) != 0)
290         return (-1);
292     return (dhp->dh_addr - pc);
293 }
295 int
296 dis_instrlen(dis_handle_t *dhp, uint64_t pc)
297 {
298     if (dhp->dh_arch->da_instrlen == NULL)
299         return (dis_generic_instrlen(dhp, pc));
301     return (dhp->dh_arch->da_instrlen(dhp, pc));
302 }
304 int
305 dis_vsnprintf(char *restrict s, size_t n, const char *restrict format,
306              va_list args)
307 {
308     #ifdef DIS_STANDALONE
309         return (mdb_iob_vsnprintf(s, n, format, args));
310     #else
311         return (vsnprintf(s, n, format, args));
312     #endif
313 }
315 int
316 dis_snprintf(char *restrict s, size_t n, const char *restrict format, ...)
317 {
318     va_list args;
320     va_start(args, format);
321     n = dis_vsnprintf(s, n, format, args);
322     va_end(args);
324     return (n);
325 }
```

```

*****
2897 Thu Jul 23 08:02:31 2015
new/usr/src/lib/libdisasm/common/libdisasm.h
6066 dis: support for System/370, System/390, and z/Architecture ELF bins
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */

22 /*
23  * Copyright 2007 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25  * Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
26  * Copyright 2015 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
27 #endif /* ! codereview */
28 */

30 #ifndef _LIBDISASM_H
31 #define _LIBDISASM_H

33 #include <sys/types.h>

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

39 typedef struct dis_handle dis_handle_t;

41 #define DIS_DEFAULT          0x0

43 /* SPARC disassembler flags */
44 #define DIS_SPARC_V8        0x001
45 #define DIS_SPARC_V9        0x002
46 #define DIS_SPARC_V9_SGI    0x004
47 #define DIS_SPARC_V9_OPL    0x008

49 /* x86 disassembler flags */
50 #define DIS_X86_SIZE16      0x100
51 #define DIS_X86_SIZE32      0x010
52 #define DIS_X86_SIZE64      0x020

54 /* s390 disassembler flags */
55 #define DIS_S370            0x200
56 #define DIS_S390_31         0x400
57 #define DIS_S390_64         0x800

59 #endif /* ! codereview */
60 /* generic disassembler flags */
61 #define DIS_OCTAL           0x040

```

```

62 #define DIS_NOIMMSYM        0x080

64 #define DIS_ARCH_MASK      (DIS_SPARC_V8 | \
65                             DIS_SPARC_V9 | DIS_SPARC_V9_SGI | DIS_SPARC_V9_OPL | \
66                             DIS_X86_SIZE16 | DIS_X86_SIZE32 | DIS_X86_SIZE64 | \
67                             DIS_S370 | DIS_S390_31 | DIS_S390_64)
26                             DIS_X86_SIZE16 | DIS_X86_SIZE32 | DIS_X86_SIZE64)

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

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

76 extern int dis_disassemble(dis_handle_t *, uint64_t, char *, size_t);
77 extern uint64_t dis_previnstr(dis_handle_t *, uint64_t, int n);
78 extern void dis_set_data(dis_handle_t *, void *);
79 extern void dis_flags_set(dis_handle_t *, int f);
80 extern void dis_flags_clear(dis_handle_t *, int f);
81 extern int dis_max_instrlen(dis_handle_t *);
82 extern int dis_min_instrlen(dis_handle_t *);
83 extern int dis_instrlen(dis_handle_t *, uint64_t);

85 /* libdisasm errors */
86 #define E_DIS_NOMEM          1          /* Out of memory */
87 #define E_DIS_INVALIDFLAG    2          /* Invalid flag for this architecture */
88 #define E_DIS_UNSUPARCH      3          /* Unsupported architecture */

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

93 #ifdef __cplusplus
94 }

```

unchanged portion omitted