```
**********************************************************
   18988 Wed Oct 14 16:45:11 2015
new/usr/src/cmd/dis/dis_main.c
6066 dis: support for System/370, System/390, and z/Architecture ELF bins
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

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

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

#include <libdisasm.h>

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

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

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

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



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

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

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

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

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

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

        return (0);
}

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

```
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 }
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;
404         /* ignore data sections */
405         if (!dis_section_istext(scn))
406                 return;
408         if (!g_quiet)
409                 (void) printf("\nsection %s\n", dis_section_name(scn));
411         dis_data(tgt, dhp, dis_section_addr(scn), dis_section_data(scn),
412             dis_section_size(scn));
413 }
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;
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;
434         if (!g_quiet)
435                 (void) printf("\nsection %s\n", dis_section_name(scn));
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 }
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;
462         dis_data(ca->ca_tgt, ca->ca_handle, dis_function_addr(func),
463             dis_function_data(func), dis_function_size(func));
464 }
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;
482         /*
483          * First, initialize the target
484          */
485         if ((tgt = dis_tgt_create(filename)) == NULL)
486                 return;
488         if (!g_quiet)
489                 (void) printf("disassembly for %s\n\n",  filename);
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);
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;
514                 case EM_SPARC32PLUS:
515                 {
516                         uint64_t flags = ehdr.e_flags & EF_SPARC_32PLUS_MASK;
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                         }
```

```
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                 }

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                         }

540                         g_flags |= DIS_SPARC_V9 | DIS_SPARC_V9_SGI;
541                         break;

543                 case EM_386:
544                         g_flags |= DIS_X86_SIZE32;
545                         break;

547                 case EM_AMD64:
548                         g_flags |= DIS_X86_SIZE64;
549                         break;

551                 case EM_S370:
552                         g_flags |= DIS_S370;

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;

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                         }

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;

578                 default:
579                         die("%s: unsupported ELF machine 0x%x", filename,
580                             ehdr.e_machine);
581                 }

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                  */

589                 if (ehdr.e_type == ET_REL)
```



```
590                         g_flags |= DIS_NOIMMSYM;

592                 if (!g_quiet && dis_tgt_member(current) != NULL)
593                         (void) printf("\narchive member %s\n",
594                             dis_tgt_member(current));

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()));

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;

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

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);

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

629                         dis_scnlist_destroy(sections);
630                         dis_funclist_destroy(functions);
631                 }

633                 dis_handle_destroy(dhp);
634         }

636         dis_tgt_destroy(tgt);
637 }

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 }

647 typedef struct lib_node {
648         char *path;
649         struct lib_node *next;
650 } lib_node_t;

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 Wed Oct 14 16:45:11 2015
new/usr/src/lib/libdisasm/Makefile.com
6066 dis: support for System/370, System/390, and z/Architecture ELF bins
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
# Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
# Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
#

#
# The build process for libdisasm is sightly different from that used by other
# libraries, because libdisasm must be built in two flavors - as a standalone
# for use by kmdb and as a normal library.  We use $(CURTYPE) to indicate the
# current flavor being built.
#

LIBRARY=        libdisasm.a
STANDLIBRARY=   libstanddisasm.so
VERS=           .1

# By default, we build the shared library.  Construction of the standalone
# is specifically requested by architecture-specific Makefiles.
TYPES=          library
CURTYPE=        library

COMDIR=         $(SRC)/lib/libdisasm/common

#
# Architecture-independent files
#
SRCS_common=            $(COMDIR)/libdisasm.c
OBJECTS_common=         libdisasm.o

#
# Architecture-dependent disassembly files
#
SRCS_i386=              $(COMDIR)/dis_i386.c \
                        $(SRC)/common/dis/i386/dis_tables.c
SRCS_sparc=             $(COMDIR)/dis_sparc.c \
                        $(COMDIR)/dis_sparc_fmt.c \
                        $(COMDIR)/dis_sparc_instr.c
SRCS_s390x=             $(COMDIR)/dis_s390x.c
#endif /* ! codereview */
```



```
OBJECTS_i386=           dis_i386.o \
                        dis_tables.o
OBJECTS_sparc=          dis_sparc.o \
                        dis_sparc_fmt.o \
                        dis_sparc_instr.o
OBJECTS_s390x=          dis_s390x.o
#endif /* ! codereview */

#
# We build the regular shared library with support for all architectures.
# The standalone version should only contain code for the native
# architecture to reduce the memory footprint of kmdb.
#
OBJECTS_library=        $(OBJECTS_common) \
                        $(OBJECTS_i386) \
                        $(OBJECTS_sparc) \
                        $(OBJECTS_s390x)
                        $(OBJECTS_sparc)
OBJECTS_standalone=     $(OBJECTS_common) \
                        $(OBJECTS_$(MACH))
OBJECTS=                $(OBJECTS_$(CURTYPE))

include $(SRC)/lib/Makefile.lib

SRCS_library=           $(SRCS_common) \
                        $(SRCS_i386) \
                        $(SRCS_sparc) \
                        $(SRCS_s390x)
                        $(SRCS_sparc)
SRCS_standalone=        $(SRCS_common) \
                        $(SRCS_$(MACH))
SRCS=                   $(SRCS_$(CURTYPE))

#
# Used to verify that the standalone doesn't have any unexpected external
# dependencies.
#
LINKTEST_OBJ = objs/linktest_stand.o

CLOBBERFILES_standalone = $(LINKTEST_OBJ)
CLOBBERFILES += $(CLOBBERFILES_$(CURTYPE))

LIBS_standalone = $(STANDLIBRARY)
LIBS_library = $(DYNLIB) $(LINTLIB)
LIBS = $(LIBS_$(CURTYPE))

MAPFILES =      $(COMDIR)/mapfile-vers

LDLIBS +=       -lc

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

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

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

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

# We want the thread-specific errno in the library, but we don't want it in
# the standalone.  $(DTS_ERRNO) is designed to add -D_TS_ERRNO to $(CPPFLAGS),
# 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:
```

```
**********************************************************
   79676 Wed Oct 14 16:45:11 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/debug.h>
  20 #include <sys/byteorder.h>

  22 #include "libdisasm_impl.h"

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

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



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

 105 /* BEGIN CSTYLED */
 106 enum ifmt {
 107         /* invalid */
 108         IF_INVAL = 0,

 110         /* indirection */
 111         IF_TBL,
 112         IF_MULTI,

 114         /* 2-byte */
 115         IF_ZERO,                /* 370, 390, z */
 116         IF_E,                   /*      390, z */
 117         IF_I,                   /*      390, z */
 118         IF_RR,                  /* 370, 390, z */

 120         /* 4-byte */
 121         IF_DIAG,                /* 370, 390, z */
 122         IF_IE,                  /*           z */
 123         IF_RIa,                 /*      390, z */
 124         IF_RIb,                 /*      390, z */
 125         IF_RIc,                 /*      390, z */
 126         IF_RRD,                 /*      390, z */ /* on 390 these are RRF */
 127         IF_RRE,                 /*      390, z */
```

```
128         IF_RRFa,                /*      390, z */
129         IF_RRFb,                /*      390, z */
130         IF_RRFc,                /*      390, z */
131         IF_RRFd,                /*      390, z */
132         IF_RRFe,                /*      390, z */
133         IF_RSa,                 /* 370, 390, z */
134         IF_RSb,                 /* 370, 390, z */
135         IF_RSI,                 /*      390, z */
136         IF_RXa,                 /* 370, 390, z */
137         IF_RXb,                 /* 370, 390, z */
138         IF_S,                   /* 370, 390, z */
139         IF_SI,                  /* 370, 390, z */

141         /* 6-byte */
142         IF_MII,                 /*           z */
143         IF_RIEa,                /*           z */
144         IF_RIEb,                /*           z */
145         IF_RIEc,                /*           z */
146         IF_RIEd,                /*           z */
147         IF_RIEe,                /*           z */
148         IF_RIEf,                /*           z */
149         IF_RILa,                /*      390, z */
150         IF_RILb,                /*      390, z */
151         IF_RILc,                /*      390, z */
152         IF_RIS,                 /*           z */
153         IF_RRS,                 /*           z */
154         IF_RSLa,                /*      390, z */
155         IF_RSLb,                /*           z */
156         IF_RSYa,                /*           z */
157         IF_RSYb,                /*           z */
158         IF_RXE,                 /*      390, z */
159         IF_RXF,                 /*      390, z */
160         IF_RXYa,                /*           z */
161         IF_RXYb,                /*           z */
162         IF_SIL,                 /*           z */
163         IF_SIY,                 /*           z */
164         IF_SMI,                 /*           z */
165         IF_SSa,                 /* 370, 390, z */
166         IF_SSb,                 /* 370, 390, z */
167         IF_SSc,                 /* 370, 390, z */
168         IF_SSd,                 /*      390, z */
169         IF_SSe,                 /*      390, z */
170         IF_SSf,                 /*      390, z */
171         IF_SSE,                 /*      390, z */
172         IF_SSF,                 /*           z */
173 };

175 #define IF_NFMTS                (IF_SSF + 1)

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

186 struct inst_table {
187         union {
188                 struct {
189                         const char *it_name;
190                         unsigned it_flags;
191                 } it_inst;
192                 struct {
193                         const struct inst_table *it_ptr;
```



```
194                         uint8_t it_off:4;
195                         uint8_t it_shift:4;
196                         uint8_t it_mask;
197                 } it_table;
198                 struct {
199                         const struct inst_table *it_ptr;
200                 } it_multi;
201         } it_u;
202         enum ifmt it_fmt;
203 };

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

207 union inst {
208         uint8_t raw[6];
209         struct {
210                 uint8_t op;
211                 uint8_t par1;
212                 uint16_t par2;
213         } diag;
214         struct {
215                 uint8_t op;
216                 uint8_t i;
217         } i;
218         struct {
219                 uint16_t op;
220                 uint8_t pad;
221                 BITFLD(i1, i2);
222         } ie;
223         struct {
224                 uint8_t op;
225                 BITFLD(m1, ri2h);
226                 uint8_t ri2l;
227                 uint8_t ri3h;
228                 uint16_t ri3l;
229         } mii;
230         struct {
231                 uint8_t op;
232                 BITFLD(r1, r2);
233         } rr;
234         struct {
235                 uint16_t op;
236                 BITFLD(r1, pad);
237                 BITFLD(r3, r2);
238         } rrd;
239         struct {
240                 uint16_t op;
241                 uint8_t pad;
242                 BITFLD(r1, r2);
243         } rre;
244         struct {
245                 uint16_t op;
246                 BITFLD(r1, m4);
247                 BITFLD(r3, r2);
248         } rrf_ab;
249         struct {
250                 uint16_t op;
251                 BITFLD(m3, m4);
252                 BITFLD(r1, r2);
253         } rrf_cde;
254         struct {
255                 uint8_t op1;
256                 BITFLD(r1, r2);
257                 BITFLD(b4, d4h);
258                 uint8_t d4l;
259                 BITFLD(m3, pad);
```

```
260                 uint8_t op2;
261         } rrs;
262         struct {
263                 uint8_t op;
264                 BITFLD(r1, x2);
265                 BITFLD(b2, d2h);
266                 uint8_t d2l;
267         } rx_a;
268         struct {
269                 uint8_t op;
270                 BITFLD(m1, x2);
271                 BITFLD(b2, d2h);
272                 uint8_t d2l;
273         } rx_b;
274         struct {
275                 uint8_t op1;
276                 BITFLD(r1, x2);
277                 BITFLD(b2, d2h);
278                 uint8_t d2l;
279                 uint8_t pad;
280                 uint8_t op2;
281         } rxe;
282         struct {
283                 uint8_t op1;
284                 BITFLD(r3, x2);
285                 BITFLD(b2, d2h);
286                 uint8_t d2l;
287                 BITFLD(r1, pad);
288                 uint8_t op2;
289         } rxf;
290         struct {
291                 uint8_t op1;
292                 BITFLD(r1, x2);
293                 BITFLD(b2, dl2h);
294                 uint8_t dl2l;
295                 uint8_t dh2;
296                 uint8_t op2;
297         } rxy_a;
298         struct {
299                 uint8_t op1;
300                 BITFLD(m1, x2);
301                 BITFLD(b2, dl2h);
302                 uint8_t dl2l;
303                 uint8_t dh2;
304                 uint8_t op2;
305         } rxy_b;
306         struct {
307                 uint8_t op;
308                 BITFLD(r1, r3);
309                 BITFLD(b2, d2h);
310                 uint8_t d2l;
311         } rs_a;
312         struct {
313                 uint8_t op;
314                 BITFLD(r1, m3);
315                 BITFLD(b2, d2h);
316                 uint8_t d2l;
317         } rs_b;
318         struct {
319                 uint8_t op1;
320                 BITFLD(l1, pad1);
321                 BITFLD(b1, d1h);
322                 uint8_t d1l;
323                 uint8_t pad2;
324                 uint8_t op2;
325         } rsl_a;
```



```
326         struct {
327                 uint8_t op1;
328                 uint8_t l2;
329                 BITFLD(b2, d2h);
330                 uint8_t d2l;
331                 BITFLD(r1, m3);
332                 uint8_t op2;
333         } rsl_b;
334         struct {
335                 uint8_t op;
336                 BITFLD(r1, r3);
337                 uint16_t ri2;
338         } rsi;
339         struct {
340                 uint8_t op1;
341                 BITFLD(r1, r3);
342                 BITFLD(b2, dl2h);
343                 uint8_t dl2l;
344                 uint8_t dh2;
345                 uint8_t op2;
346         } rsy_a;
347         struct {
348                 uint8_t op1;
349                 BITFLD(r1, m3);
350                 BITFLD(b2, dl2h);
351                 uint8_t dl2l;
352                 uint8_t dh2;
353                 uint8_t op2;
354         } rsy_b;
355         struct {
356                 uint8_t op1;
357                 BITFLD(r1, op2);
358                 uint16_t i2;
359         } ri_a;
360         struct {
361                 uint8_t op1;
362                 BITFLD(r1, op2);
363                 uint16_t ri2;
364         } ri_b;
365         struct {
366                 uint8_t op1;
367                 BITFLD(m1, op2);
368                 uint16_t ri2;
369         } ri_c;
370         struct {
371                 uint8_t op1;
372                 BITFLD(r1, _pad0);
373                 uint16_t i2;
374                 BITFLD(m3, _pad1);
375                 uint8_t op2;
376         } rie_a;
377         struct {
378                 uint8_t op1;
379                 BITFLD(r1, r2);
380                 uint16_t ri4;
381                 BITFLD(m3, _pad);
382                 uint8_t op2;
383         } rie_b;
384         struct {
385                 uint8_t op1;
386                 BITFLD(r1, m3);
387                 uint16_t ri4;
388                 uint8_t i2;
389                 uint8_t op2;
390         } rie_c;
391         struct {
```

```
392                 uint8_t op1;
393                 BITFLD(r1, r3);
394                 uint16_t i2;
395                 uint8_t _pad;
396                 uint8_t op2;
397         } rie_d;
398         struct {
399                 uint8_t op1;
400                 BITFLD(r1, r3);
401                 uint16_t ri2;
402                 uint8_t _pad;
403                 uint8_t op2;
404         } rie_e;
405         struct {
406                 uint8_t op1;
407                 BITFLD(r1, r2);
408                 uint8_t i3;
409                 uint8_t i4;
410                 uint8_t i5;
411                 uint8_t op2;
412         } rie_f;
413         struct {
414                 uint8_t op1;
415                 BITFLD(r1, op2);
416                 uint16_t i2h;
417                 uint16_t i2l;
418         } ril_a;
419         struct {
420                 uint8_t op1;
421                 BITFLD(r1, op2);
422                 uint16_t ri2h;
423                 uint16_t ri2l;
424         } ril_b;
425         struct {
426                 uint8_t op1;
427                 BITFLD(m1, op2);
428                 uint16_t ri2h;
429                 uint16_t ri2l;
430         } ril_c;
431         struct {
432                 uint8_t op1;
433                 BITFLD(r1, m3);
434                 BITFLD(b4, d4h);
435                 uint8_t d4l;
436                 uint8_t i2;
437                 uint8_t op2;
438         } ris;
439         struct {
440                 uint8_t op;
441                 uint8_t i2;
442                 BITFLD(b1, d1h);
443                 uint8_t d1l;
444         } si;
445         struct {
446                 uint16_t op;
447                 BITFLD(b1, d1h);
448                 uint8_t d1l;
449                 uint16_t i2;
450         } sil;
451         struct {
452                 uint8_t op1;
453                 uint8_t i2;
454                 BITFLD(b1, dl1h);
455                 uint8_t dl1l;
456                 uint8_t dh1;
457                 uint8_t op2;
```



```
458         } siy;
459         struct {
460                 uint8_t op;
461                 BITFLD(m1, pad);
462                 BITFLD(b3, d3h);
463                 uint8_t d3l;
464                 uint16_t ri2;
465         } smi;
466         struct {
467                 uint8_t op1;
468                 uint8_t op2;
469                 BITFLD(b2, d2h);
470                 uint8_t d2l;
471         } s;
472         struct {
473                 uint8_t op;
474                 uint8_t l;
475                 BITFLD(b1, d1h);
476                 uint8_t d1l;
477                 BITFLD(b2, d2h);
478                 uint8_t d2l;
479         } ss_a;
480         struct {
481                 uint8_t op;
482                 BITFLD(l1, l2);
483                 BITFLD(b1, d1h);
484                 uint8_t d1l;
485                 BITFLD(b2, d2h);
486                 uint8_t d2l;
487         } ss_b;
488         struct {
489                 uint8_t op;
490                 BITFLD(l1, i3);
491                 BITFLD(b1, d1h);
492                 uint8_t d1l;
493                 BITFLD(b2, d2h);
494                 uint8_t d2l;
495         } ss_c;
496         struct {
497                 uint8_t op;
498                 BITFLD(r1, r3);
499                 BITFLD(b1, d1h);
500                 uint8_t d1l;
501                 BITFLD(b2, d2h);
502                 uint8_t d2l;
503         } ss_d;
504         struct {
505                 uint8_t op;
506                 BITFLD(r1, r3);
507                 BITFLD(b2, d2h);
508                 uint8_t d2l;
509                 BITFLD(b4, d4h);
510                 uint8_t d4l;
511         } ss_e;
512         struct {
513                 uint8_t op;
514                 uint8_t l2;
515                 BITFLD(b1, d1h);
516                 uint8_t d1l;
517                 BITFLD(b2, d2h);
518                 uint8_t d2l;
519         } ss_f;
520         struct {
521                 uint16_t op;
522                 BITFLD(b1, d1h);
523                 uint8_t d1l;
```

```
524                 BITFLD(b2, d2h);
525                 uint8_t d2l;
526         } sse;
527         struct {
528                 uint8_t op1;
529                 BITFLD(r3, op2);
530                 BITFLD(b1, d1h);
531                 uint8_t d1l;
532                 BITFLD(b2, d2h);
533                 uint8_t d2l;
534         } ssf;
535 };

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

558 /*
559  * Instruction tables based on:
560  *   GA22-7000-4   System/370 Principles of Operation
561  *   SA22-7201-08  ESA/390 Principles of Operation
562  *   SA22-7832-09  z/Architecture Principles of Operation
563  */

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

579 static const struct inst_table tbl_07[] = {
580         INSTR(0x0, "nopr",  IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
581         INSTR(0x1, "bor",   IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
582         INSTR(0x2, "bhr",   IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
583         INSTR(0x3, "bcr",   IF_RR, F_370 | F_390 | F_Z | F_R1_IS_MASK),
584         INSTR(0x4, "blr",   IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
585         INSTR(0x5, "bcr",   IF_RR, F_370 | F_390 | F_Z | F_R1_IS_MASK),
586         INSTR(0x6, "bcr",   IF_RR, F_370 | F_390 | F_Z | F_R1_IS_MASK),
587         INSTR(0x7, "bnzr",  IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
588         INSTR(0x8, "ber",   IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
589         INSTR(0x9, "bcr",   IF_RR, F_370 | F_390 | F_Z | F_R1_IS_MASK),
```



```
590         INSTR(0xa, "bcr",   IF_RR, F_370 | F_390 | F_Z | F_R1_IS_MASK),
591         INSTR(0xb, "bner",  IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
592         INSTR(0xc, "bcr",   IF_RR, F_370 | F_390 | F_Z | F_R1_IS_MASK),
593         INSTR(0xd, "bnhr",  IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
594         INSTR(0xe, "bnor",  IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
595         INSTR(0xf, "br",    IF_RR, F_370 | F_390 | F_Z | F_HIDE_MASK),
596 };

598 static const struct inst_table tbl_47[] = {
599         INSTR(0x0, "nop",   IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
600         INSTR(0x1, "bo",    IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
601         INSTR(0x2, "bh",    IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
602         INSTR(0x3, "bc",    IF_RXb, F_370 | F_390 | F_Z),
603         INSTR(0x4, "bl",    IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
604         INSTR(0x5, "bc",    IF_RXb, F_370 | F_390 | F_Z),
605         INSTR(0x6, "bc",    IF_RXb, F_370 | F_390 | F_Z),
606         INSTR(0x7, "bne",   IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
607         INSTR(0x8, "be",    IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
608         INSTR(0x9, "bc",    IF_RXb, F_370 | F_390 | F_Z),
609         INSTR(0xa, "bc",    IF_RXb, F_370 | F_390 | F_Z),
610         INSTR(0xb, "bnl",   IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
611         INSTR(0xc, "bc",    IF_RXb, F_370 | F_390 | F_Z),
612         INSTR(0xd, "bnh",   IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
613         INSTR(0xe, "bno",   IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
614         INSTR(0xf, "b",     IF_RXb, F_370 | F_390 | F_Z | F_HIDE_MASK),
615 };

617 /* the preferred mnemonic changed over time */
618 static const struct inst_table tbl_25[] = {
619         INSTR(F_370, "lrdr",  IF_RR,   F_370),
620         INSTR(F_390, "ldxr",  IF_RR,   F_390),
621         INSTR(F_Z,   "ldxr",  IF_RR,   F_Z),
622 };

624 /* the preferred mnemonic changed over time */
625 static const struct inst_table tbl_35[] = {
626         INSTR(F_370, "lrer",  IF_RR,   F_370),
627         INSTR(F_390, "ledr",  IF_RR,   F_390),
628         INSTR(F_Z,   "ledr",  IF_RR,   F_Z),
629 };

631 /* the preferred mnemonic changed over time */
632 static const struct inst_table tbl_3c[] = {
633         INSTR(F_370, "mer",   IF_RR,   F_370),
634         INSTR(F_390, "mder",  IF_RR,   F_390),
635         INSTR(F_Z,   "mder",  IF_RR,   F_Z),
636 };

638 /* the preferred mnemonic changed over time */
639 static const struct inst_table tbl_7c[] = {
640         INSTR(F_370, "me",    IF_RXa,  F_370),
641         INSTR(F_390, "mde",   IF_RXa,  F_390),
642         INSTR(F_Z,   "mde",   IF_RXa,  F_Z),
643 };

645 /* the meaning of this instruction changed over time */
646 static const struct inst_table tbl_84[] = {
647         INSTR(F_370, "wrd",   IF_SI,   F_370),
648         INSTR(F_390, "brxh",  IF_RSI,  F_390),
649         INSTR(F_Z,   "brxh",  IF_RSI,  F_Z),
650 };

652 /* the meaning of this instruction changed over time */
653 static const struct inst_table tbl_85[] = {
654         INSTR(F_370, "rdd",   IF_SI,   F_370),
655         INSTR(F_390, "brxle", IF_RSI,  F_390),
```

```
656         INSTR(F_Z,   "brxle", IF_RSI,  F_Z),
657 };

659 static const struct inst_table tbl_a5x[16] = {
660         INSTR(0x0,  "iihh",  IF_RIa, F_Z),
661         INSTR(0x1,  "iihl",  IF_RIa, F_Z),
662         INSTR(0x2,  "iilh",  IF_RIa, F_Z),
663         INSTR(0x3,  "iill",  IF_RIa, F_Z),
664         INSTR(0x4,  "nihh",  IF_RIa, F_Z),
665         INSTR(0x5,  "nihl",  IF_RIa, F_Z),
666         INSTR(0x6,  "nilh",  IF_RIa, F_Z),
667         INSTR(0x7,  "nill",  IF_RIa, F_Z),
668         INSTR(0x8,  "oihh",  IF_RIa, F_Z),
669         INSTR(0x9,  "oihl",  IF_RIa, F_Z),
670         INSTR(0xa,  "oilh",  IF_RIa, F_Z),
671         INSTR(0xb,  "oill",  IF_RIa, F_Z),
672         INSTR(0xc,  "llihh", IF_RIa, F_Z),
673         INSTR(0xd,  "llihl", IF_RIa, F_Z),
674         INSTR(0xe,  "llilh", IF_RIa, F_Z),
675         INSTR(0xf,  "llill", IF_RIa, F_Z),
676 };

678 /* the preferred mnemonic changed over time */
679 static const struct inst_table tbl_a70[] = {
680         INSTR(F_390, "tmh",    IF_RIa, F_390),
681         INSTR(F_Z,   "tmlh",   IF_RIa, F_Z),
682 };

684 /* the preferred mnemonic changed over time */
685 static const struct inst_table tbl_a71[] = {
686         INSTR(F_390, "tml",    IF_RIa, F_390),
687         INSTR(F_Z,   "tmll",   IF_RIa, F_Z),
688 };

690 static const struct inst_table tbl_a74[16] = {
691         INSTR(0x0, "jnop", IF_RIc, F_390 | F_Z | F_HIDE_MASK),
692         INSTR(0x1, "jo",   IF_RIc, F_390 | F_Z | F_HIDE_MASK),
693         INSTR(0x2, "jh",   IF_RIc, F_390 | F_Z | F_HIDE_MASK),
694         INSTR(0x3, "brc",  IF_RIc, F_390 | F_Z),
695         INSTR(0x4, "jl",   IF_RIc, F_390 | F_Z | F_HIDE_MASK),
696         INSTR(0x5, "brc",  IF_RIc, F_390 | F_Z),
697         INSTR(0x6, "brc",  IF_RIc, F_390 | F_Z),
698         INSTR(0x7, "jne",  IF_RIc, F_390 | F_Z | F_HIDE_MASK),
699         INSTR(0x8, "je",   IF_RIc, F_390 | F_Z | F_HIDE_MASK),
700         INSTR(0x9, "brc",  IF_RIc, F_390 | F_Z),
701         INSTR(0xa, "brc",  IF_RIc, F_390 | F_Z),
702         INSTR(0xb, "jnl",  IF_RIc, F_390 | F_Z | F_HIDE_MASK),
703         INSTR(0xc, "brc",  IF_RIc, F_390 | F_Z),
704         INSTR(0xd, "jnh",  IF_RIc, F_390 | F_Z | F_HIDE_MASK),
705         INSTR(0xe, "jno",  IF_RIc, F_390 | F_Z | F_HIDE_MASK),
706         INSTR(0xf, "j",    IF_RIc, F_390 | F_Z | F_HIDE_MASK),
707 };

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



```
722         INSTR(0xc, "mhi",    IF_RIa, F_390 | F_Z),
723         INSTR(0xd, "mghi",   IF_RIa, F_Z),
724         INSTR(0xe, "chi",    IF_RIa, F_390 | F_Z | F_SIGNED_IMM),
725         INSTR(0xf, "cghi",   IF_RIa, F_Z | F_SIGNED_IMM),
726 };

728 static const struct inst_table tbl_b2a6[] = {
729         INSTR(F_390, "cuutf", IF_RRFc, F_390),
730         INSTR(F_Z,   "c21",   IF_RRFc, F_Z),
731 };

733 static const struct inst_table tbl_b2a7[] = {
734         INSTR(F_390, "cutfu",  IF_RRFc, F_390),
735         INSTR(F_Z,   "cu12",   IF_RRFc, F_Z),
736 };

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

```
788         INSTR(0x45, "sqer",   IF_RRE,  F_390 | F_Z),
789         INSTR(0x46, "stura",  IF_RRE,  F_390 | F_Z),
790         INSTR(0x47, "msta",   IF_RRE,  F_390 | F_Z),
791         INSTR(0x48, "palb",   IF_RRE,  F_390 | F_Z),
792         INSTR(0x49, "ereg",   IF_RRE,  F_390 | F_Z),
793         INSTR(0x4a, "esta",   IF_RRE,  F_390 | F_Z),
794         INSTR(0x4b, "lura",   IF_RRE,  F_390 | F_Z),
795         INSTR(0x4c, "tar",    IF_RRE,  F_390 | F_Z),
796         INSTR(0x4d, "cpya",   IF_RRE,  F_390 | F_Z),
797         INSTR(0x4e, "sar",    IF_RRE,  F_390 | F_Z),
798         INSTR(0x4f, "ear",    IF_RRE,  F_390 | F_Z),
799         INSTR(0x50, "csp",    IF_RRE,  F_390 | F_Z),
800         INSTR(0x52, "msr",    IF_RRE,  F_390 | F_Z),
801         INSTR(0x54, "mvpg",   IF_RRE,  F_390 | F_Z),
802         INSTR(0x55, "mvst",   IF_RRE,  F_390 | F_Z),
803         INSTR(0x57, "cuse",   IF_RRE,  F_390 | F_Z),
804         INSTR(0x58, "bsg",    IF_RRE,  F_390 | F_Z),
805         INSTR(0x5a, "bsa",    IF_RRE,  F_390 | F_Z),
806         INSTR(0x5d, "clst",   IF_RRE,  F_390 | F_Z),
807         INSTR(0x5e, "srst",   IF_RRE,  F_390 | F_Z),
808         INSTR(0x63, "cmpsc",  IF_RRE,  F_Z),
809         INSTR(0x76, "xsch",   IF_S,    F_Z),
810         INSTR(0x77, "rp",     IF_S,    F_390 | F_Z),
811         INSTR(0x78, "stcke",  IF_S,    F_390 | F_Z),
812         INSTR(0x79, "sacf",   IF_S,    F_390 | F_Z),
813         INSTR(0x7c, "stckf",  IF_S,    F_Z),
814         INSTR(0x7d, "stsi",   IF_S,    F_390 | F_Z),
815         INSTR(0x99, "srnm",   IF_S,    F_390 | F_Z),
816         INSTR(0x9c, "stfpc",  IF_S,    F_390 | F_Z),
817         INSTR(0x9d, "lfpc",   IF_S,    F_390 | F_Z),
818         INSTR(0xa5, "tre",    IF_RRE,  F_390 | F_Z),
819         MULTI(0xa6, tbl_b2a6),
820         MULTI(0xa7, tbl_b2a7),
821         INSTR(0xb0, "stfle",  IF_S,    F_Z),
822         INSTR(0xb1, "stfl",   IF_S,    F_390 | F_Z),
823         INSTR(0xb2, "lpswe",  IF_S,    F_Z),
824         INSTR(0xb8, "srnmb",  IF_S,    F_Z),
825         INSTR(0xb9, "srnmt",  IF_S,    F_Z),
826         INSTR(0xbd, "lfas",   IF_S,    F_Z),
827         INSTR(0xe8, "ppa",    IF_RRFc, F_Z),
828         INSTR(0xec, "etnd",   IF_RRE,  F_Z),
829         INSTR(0xf8, "tend",   IF_S,    F_Z),
830         INSTR(0xfa, "niai",   IF_IE,   F_Z),
831         INSTR(0xfc, "tabort", IF_S,    F_Z),
832         INSTR(0xff, "trap4",  IF_S,    F_390 | F_Z),
833 };

835 static const struct inst_table tbl_b344[] = {
836         INSTR(F_390, "ledbr",  IF_RRE,  F_390),
837         INSTR(F_Z,   "ledbra", IF_RRFe, F_Z),
838 };

840 static const struct inst_table tbl_b345[] = {
841         INSTR(F_390, "ldxbr",  IF_RRE,  F_390),
842         INSTR(F_Z,   "ldxbra", IF_RRFe, F_Z),
843 };

845 static const struct inst_table tbl_b346[] = {
846         INSTR(F_390, "lexbr",  IF_RRE,  F_390),
847         INSTR(F_Z,   "lexbra", IF_RRFe, F_Z),
848 };

850 static const struct inst_table tbl_b347[] = {
851         INSTR(F_390, "fixbr",  IF_RRFe, F_390),
852         INSTR(F_Z,   "fixbra", IF_RRFe, F_Z),
853 };
```



```
855 static const struct inst_table tbl_b357[] = {
856         INSTR(F_390, "fiebr",  IF_RRFe, F_390),
857         INSTR(F_Z,   "fiebre", IF_RRFe, F_Z),
858 };

860 static const struct inst_table tbl_b35f[] = {
861         INSTR(F_390, "fidbr",  IF_RRFe, F_390),
862         INSTR(F_Z,   "fidbra", IF_RRFe, F_Z),
863 };

865 static const struct inst_table tbl_b394[] = {
866         INSTR(F_390, "cefbr",  IF_RRE,  F_390),
867         INSTR(F_Z,   "cefbra", IF_RRFe, F_Z),
868 };

870 static const struct inst_table tbl_b395[] = {
871         INSTR(F_390, "cdfbr",  IF_RRE,  F_390),
872         INSTR(F_Z,   "cdfbra", IF_RRFe, F_Z),
873 };

875 static const struct inst_table tbl_b396[] = {
876         INSTR(F_390, "cxfbr",  IF_RRE,  F_390),
877         INSTR(F_Z,   "cxfbra", IF_RRFe, F_Z),
878 };

880 static const struct inst_table tbl_b398[] = {
881         INSTR(F_390, "cfebr",  IF_RRFe, F_390),
882         INSTR(F_Z,   "cfebra", IF_RRFe, F_Z),
883 };

885 static const struct inst_table tbl_b399[] = {
886         INSTR(F_390, "cfdbr",  IF_RRFe, F_390),
887         INSTR(F_Z,   "cfdbra", IF_RRFe, F_Z),
888 };

890 static const struct inst_table tbl_b39a[] = {
891         INSTR(F_390, "cfxbr",  IF_RRFe, F_390),
892         INSTR(F_Z,   "cfxbra", IF_RRFe, F_Z),
893 };

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

```
920         INSTR(0x18, "kdbr",   IF_RRE,  F_390 | F_Z),
921         INSTR(0x19, "cdbr",   IF_RRE,  F_390 | F_Z),
922         INSTR(0x1a, "adbr",   IF_RRE,  F_390 | F_Z),
923         INSTR(0x1b, "sdbr",   IF_RRE,  F_390 | F_Z),
924         INSTR(0x1c, "mdbr",   IF_RRE,  F_390 | F_Z),
925         INSTR(0x1d, "ddbr",   IF_RRE,  F_390 | F_Z),
926         INSTR(0x1e, "madbr",  IF_RRD,  F_390 | F_Z),
927         INSTR(0x1f, "msdbr",  IF_RRD,  F_390 | F_Z),
928         INSTR(0x24, "lder",   IF_RRE,  F_390 | F_Z),
929         INSTR(0x25, "lxdr",   IF_RRE,  F_390 | F_Z),
930         INSTR(0x26, "lxer",   IF_RRE,  F_390 | F_Z),
931         INSTR(0x2e, "maer",   IF_RRD,  F_390 | F_Z),
932         INSTR(0x2f, "mser",   IF_RRD,  F_390 | F_Z),
933         INSTR(0x36, "sqxr",   IF_RRE,  F_390 | F_Z),
934         INSTR(0x37, "meer",   IF_RRE,  F_390 | F_Z),
935         INSTR(0x38, "maylr",  IF_RRD,  F_Z),
936         INSTR(0x39, "mylr",   IF_RRD,  F_Z),
937         INSTR(0x3a, "mayr",   IF_RRD,  F_Z),
938         INSTR(0x3b, "myr",    IF_RRD,  F_Z),
939         INSTR(0x3c, "mayhr",  IF_RRD,  F_Z),
940         INSTR(0x3d, "myhr",   IF_RRD,  F_Z),
941         INSTR(0x3e, "madr",   IF_RRD,  F_390 | F_Z),
942         INSTR(0x3f, "msdr",   IF_RRD,  F_390 | F_Z),
943         INSTR(0x40, "lpxbr",  IF_RRE,  F_390 | F_Z),
944         INSTR(0x41, "lnxbr",  IF_RRE,  F_390 | F_Z),
945         INSTR(0x42, "ltxbr",  IF_RRE,  F_390 | F_Z),
946         INSTR(0x43, "lcxbr",  IF_RRE,  F_390 | F_Z),
947         MULTI(0x44, tbl_b344),
948         MULTI(0x45, tbl_b345),
949         MULTI(0x46, tbl_b346),
950         MULTI(0x47, tbl_b347),
951         INSTR(0x48, "kxbr",   IF_RRE,  F_390 | F_Z),
952         INSTR(0x49, "cxbr",   IF_RRE,  F_390 | F_Z),
953         INSTR(0x4a, "axbr",   IF_RRE,  F_390 | F_Z),
954         INSTR(0x4b, "sxbr",   IF_RRE,  F_390 | F_Z),
955         INSTR(0x4c, "mxbr",   IF_RRE,  F_390 | F_Z),
956         INSTR(0x4d, "dxbr",   IF_RRE,  F_390 | F_Z),
957         INSTR(0x50, "tbedr",  IF_RRFe, F_390 | F_Z),
958         INSTR(0x51, "tbdr",   IF_RRFe, F_390 | F_Z),
959         INSTR(0x53, "diebr",  IF_RRFb, F_390 | F_Z),
960         MULTI(0x57, tbl_b357),
961         INSTR(0x58, "thder",  IF_RRE,  F_390 | F_Z),
962         INSTR(0x59, "thdr",   IF_RRE,  F_390 | F_Z),
963         INSTR(0x5b, "didbr",  IF_RRFe, F_390 | F_Z),
964         MULTI(0x5f, tbl_b35f),
965         INSTR(0x60, "lpxr",   IF_RRE,  F_390 | F_Z),
966         INSTR(0x61, "lnxr",   IF_RRE,  F_390 | F_Z),
967         INSTR(0x62, "ltxr",   IF_RRE,  F_390 | F_Z),
968         INSTR(0x63, "lcxr",   IF_RRE,  F_390 | F_Z),
969         INSTR(0x65, "lxr",    IF_RRE,  F_390 | F_Z),
970         INSTR(0x66, "lexr",   IF_RRE,  F_390 | F_Z),
971         INSTR(0x67, "fixr",   IF_RRE,  F_390 | F_Z),
972         INSTR(0x69, "cxr",    IF_RRE,  F_390 | F_Z),
973         INSTR(0x70, "lpdfr",  IF_RRE,  F_Z),
974         INSTR(0x71, "lndfr",  IF_RRE,  F_Z),
975         INSTR(0x72, "cpsdr",  IF_RRFe, F_Z),
976         INSTR(0x73, "lcdfr",  IF_RRE,  F_Z),
977         INSTR(0x74, "lzer",   IF_RRE,  F_390 | F_Z),
978         INSTR(0x75, "lzdr",   IF_RRE,  F_390 | F_Z),
979         INSTR(0x76, "lzxr",   IF_RRE,  F_390 | F_Z),
980         INSTR(0x77, "fier",   IF_RRE,  F_390 | F_Z),
981         INSTR(0x7f, "fidr",   IF_RRE,  F_390 | F_Z),
982         INSTR(0x84, "sfpc",   IF_RRE,  F_390 | F_Z),
983         INSTR(0x85, "sfasr",  IF_RRE,  F_Z),
984         INSTR(0x8c, "efpc",   IF_RRE,  F_390 | F_Z),
985         INSTR(0x90, "celfbr", IF_RRFe, F_Z),
```



```
986         INSTR(0x91, "cdlfbr", IF_RRFe, F_Z),
987         INSTR(0x92, "cxlfbr", IF_RRFe, F_Z),
988         MULTI(0x94, tbl_b394),
989         MULTI(0x95, tbl_b395),
990         MULTI(0x96, tbl_b396),
991         MULTI(0x98, tbl_b398),
992         MULTI(0x99, tbl_b399),
993         MULTI(0x9a, tbl_b39a),
994         INSTR(0x9c, "clfebr", IF_RRFe, F_Z),
995         INSTR(0x9d, "clfdbr", IF_RRFe, F_Z),
996         INSTR(0x9e, "clfxbr", IF_RRFe, F_Z),
997         INSTR(0xa0, "celgbr", IF_RRFe, F_Z),
998         INSTR(0xa1, "cdlgbr", IF_RRFe, F_Z),
999         INSTR(0xa2, "cxlgbr", IF_RRFe, F_Z),
1000        INSTR(0xa4, "cegbra", IF_RRFe, F_Z),
1001        INSTR(0xa5, "cdgbra", IF_RRFe, F_Z),
1002        INSTR(0xa6, "cxgbra", IF_RRFe, F_Z),
1003        INSTR(0xa8, "cgebra", IF_RRFe, F_Z),
1004        INSTR(0xa9, "cgdbra", IF_RRFe, F_Z),
1005        INSTR(0xaa, "cgxbra", IF_RRFe, F_Z),
1006        INSTR(0xac, "clgebr", IF_RRFe, F_Z),
1007        INSTR(0xad, "clgdbr", IF_RRFe, F_Z),
1008        INSTR(0xae, "clgxbr", IF_RRFe, F_Z),
1009        INSTR(0xb4, "cefr",   IF_RRE,  F_390 | F_Z),
1010        INSTR(0xb5, "cdfr",   IF_RRE,  F_390 | F_Z),
1011        INSTR(0xb6, "cxfr",   IF_RRE,  F_390 | F_Z),
1012        INSTR(0xb8, "cfer",   IF_RRFe, F_390 | F_Z),
1013        INSTR(0xb9, "cfdr",   IF_RRFe, F_390 | F_Z),
1014        INSTR(0xba, "cfxr",   IF_RRFe, F_390 | F_Z),
1015        INSTR(0xc1, "ldgr",   IF_RRE,  F_Z),
1016        INSTR(0xc4, "cegr",   IF_RRE,  F_Z),
1017        INSTR(0xc5, "cdgr",   IF_RRE,  F_Z),
1018        INSTR(0xc6, "cxgr",   IF_RRE,  F_Z),
1019        INSTR(0xc8, "cger",   IF_RRFe, F_Z),
1020        INSTR(0xc9, "cgdr",   IF_RRFe, F_Z),
1021        INSTR(0xca, "cgxr",   IF_RRFe, F_Z),
1022        INSTR(0xcd, "lgdr",   IF_RRE,  F_Z),
1023        INSTR(0xd0, "mdtra",  IF_RRFa, F_Z),
1024        INSTR(0xd1, "ddtra",  IF_RRFa, F_Z),
1025        INSTR(0xd2, "adtra",  IF_RRFa, F_Z),
1026        INSTR(0xd3, "sdtra",  IF_RRFa, F_Z),
1027        INSTR(0xd4, "ldetr",  IF_RRFd, F_Z),
1028        INSTR(0xd5, "ledtr",  IF_RRFe, F_Z),
1029        INSTR(0xd6, "ltdtr",  IF_RRE,  F_Z),
1030        INSTR(0xd7, "fidtr",  IF_RRFe, F_Z),
1031        INSTR(0xd8, "mxtra",  IF_RRFa, F_Z),
1032        INSTR(0xd9, "dxtra",  IF_RRFa, F_Z),
1033        INSTR(0xda, "axtra",  IF_RRFa, F_Z),
1034        INSTR(0xdb, "sxtra",  IF_RRFa, F_Z),
1035        INSTR(0xdc, "lxdtr",  IF_RRFd, F_Z),
1036        INSTR(0xdd, "ldxtr",  IF_RRFe, F_Z),
1037        INSTR(0xde, "ltxtr",  IF_RRE,  F_Z),
1038        INSTR(0xdf, "fixtr",  IF_RRFe, F_Z),
1039        INSTR(0xe0, "kdtr",   IF_RRE,  F_Z),
1040        INSTR(0xe1, "cgdtra", IF_RRFe, F_Z),
1041        INSTR(0xe2, "cudtr",  IF_RRE,  F_Z),
1042        INSTR(0xe3, "csdtr",  IF_RRFd, F_Z),
1043        INSTR(0xe4, "cdtr",   IF_RRE,  F_Z),
1044        INSTR(0xe5, "eedtr",  IF_RRE,  F_Z),
1045        INSTR(0xe7, "esdtr",  IF_RRE,  F_Z),
1046        INSTR(0xe8, "kxtr",   IF_RRE,  F_Z),
1047        INSTR(0xe9, "cgxtra", IF_RRFe, F_Z),
1048        INSTR(0xea, "cuxtr",  IF_RRE,  F_Z),
1049        INSTR(0xeb, "csxtr",  IF_RRFd, F_Z),
1050        INSTR(0xec, "cxtr",   IF_RRE,  F_Z),
1051        INSTR(0xed, "eextr",  IF_RRE,  F_Z),
```

```
1052         INSTR(0xef, "esxtr",  IF_RRE,  F_Z),
1053         INSTR(0xf1, "cdgtra", IF_RRE,  F_Z),
1054         INSTR(0xf2, "cdutr",  IF_RRE,  F_Z),
1055         INSTR(0xf3, "cdstr",  IF_RRE,  F_Z),
1056         INSTR(0xf4, "cedtr",  IF_RRE,  F_Z),
1057         INSTR(0xf5, "qadtr",  IF_RRFb, F_Z),
1058         INSTR(0xf6, "iedtr",  IF_RRFb, F_Z),
1059         INSTR(0xf7, "rrdtr",  IF_RRFb, F_Z),
1060         INSTR(0xf9, "cxgtra", IF_RRE,  F_Z),
1061         INSTR(0xfa, "cxutr",  IF_RRE,  F_Z),
1062         INSTR(0xfb, "cxstr",  IF_RRE,  F_Z),
1063         INSTR(0xfc, "cextr",  IF_RRE,  F_Z),
1064         INSTR(0xfd, "qaxtr",  IF_RRFb, F_Z),
1065         INSTR(0xfe, "iextr",  IF_RRFb, F_Z),
1066         INSTR(0xff, "rrxtr",  IF_RRFb, F_Z),
1067 };

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



```
1118         INSTR(0x42, "clgdtr", IF_RRFe, F_Z),
1119         INSTR(0x43, "clfdtr", IF_RRFe, F_Z),
1120         INSTR(0x46, "bctgr",  IF_RRE,  F_Z),
1121         INSTR(0x49, "cfxtr",  IF_RRFe, F_Z),
1122         INSTR(0x4a, "clgxtr", IF_RRFe, F_Z),
1123         INSTR(0x4b, "clfxtr", IF_RRFe, F_Z),
1124         INSTR(0x51, "cdftr",  IF_RRE,  F_Z),
1125         INSTR(0x52, "cdlgtr", IF_RRFe, F_Z),
1126         INSTR(0x53, "cdlftr", IF_RRFe, F_Z),
1127         INSTR(0x59, "cxftr",  IF_RRE,  F_Z),
1128         INSTR(0x5a, "cxlgtr", IF_RRFe, F_Z),
1129         INSTR(0x5b, "cxlftr", IF_RRFe, F_Z),
1130         INSTR(0x60, "cgrt",   IF_RRFc, F_Z),
1131         INSTR(0x61, "clgrt",  IF_RRFc, F_Z),
1132         INSTR(0x72, "crt",    IF_RRFc, F_Z),
1133         INSTR(0x73, "clrt",   IF_RRFc, F_Z),
1134         INSTR(0x80, "ngr",    IF_RRE,  F_Z),
1135         INSTR(0x81, "ogr",    IF_RRE,  F_Z),
1136         INSTR(0x82, "xgr",    IF_RRE,  F_Z),
1137         INSTR(0x83, "flogr",  IF_RRE,  F_Z),
1138         INSTR(0x84, "llgcr",  IF_RRE,  F_Z),
1139         INSTR(0x85, "llghr",  IF_RRE,  F_Z),
1140         INSTR(0x86, "mlgr",   IF_RRE,  F_Z),
1141         INSTR(0x87, "dlgr",   IF_RRE,  F_Z),
1142         INSTR(0x88, "alcgr",  IF_RRE,  F_Z),
1143         INSTR(0x89, "slbgr",  IF_RRE,  F_Z),
1144         INSTR(0x8a, "cspg",   IF_RRE,  F_Z),
1145         INSTR(0x8d, "epsw",   IF_RRE,  F_390 | F_Z),
1146         INSTR(0x8e, "idte",   IF_RRFb, F_Z),
1147         INSTR(0x8f, "crdte",  IF_RRFb, F_Z),
1148         INSTR(0x90, "trtt",   IF_RRFc, F_390 | F_Z),
1149         INSTR(0x91, "trto",   IF_RRFc, F_390 | F_Z),
1150         INSTR(0x92, "trot",   IF_RRFc, F_390 | F_Z),
1151         INSTR(0x93, "troo",   IF_RRFc, F_390 | F_Z),
1152         INSTR(0x94, "llcr",   IF_RRE,  F_Z),
1153         INSTR(0x95, "llhr",   IF_RRE,  F_Z),
1154         INSTR(0x96, "mlr",    IF_RRE,  F_390 | F_Z),
1155         INSTR(0x97, "dlr",    IF_RRE,  F_390 | F_Z),
1156         INSTR(0x98, "alcr",   IF_RRE,  F_390 | F_Z),
1157         INSTR(0x99, "slbr",   IF_RRE,  F_390 | F_Z),
1158         INSTR(0x9a, "epair",  IF_RRE,  F_Z),
1159         INSTR(0x9b, "esair",  IF_RRE,  F_Z),
1160         INSTR(0x9d, "esea",   IF_RRE,  F_Z),
1161         INSTR(0x9e, "pti",    IF_RRE,  F_Z),
1162         INSTR(0x9f, "ssair",  IF_RRE,  F_Z),
1163         INSTR(0xa2, "ptf",    IF_RRE,  F_Z),
1164         INSTR(0xaa, "lptea",  IF_RRFb, F_Z),
1165         INSTR(0xae, "rrbm",   IF_RRE,  F_Z),
1166         INSTR(0xaf, "pfmf",   IF_RRE,  F_Z),
1167         INSTR(0xb0, "cu14",   IF_RRFc, F_Z),
1168         INSTR(0xb1, "cu24",   IF_RRFc, F_Z),
1169         INSTR(0xb2, "cu41",   IF_RRE,  F_Z),
1170         INSTR(0xb3, "cu42",   IF_RRE,  F_Z),
1171         INSTR(0xbd, "trtre",  IF_RRFc, F_Z),
1172         INSTR(0xbe, "srstu",  IF_RRE,  F_Z),
1173         INSTR(0xbf, "trte",   IF_RRFc, F_Z),
1174         INSTR(0xc8, "ahhhr",  IF_RRFa, F_Z),
1175         INSTR(0xc9, "shhhr",  IF_RRFa, F_Z),
1176         INSTR(0xca, "alhhhr", IF_RRFa, F_Z),
1177         INSTR(0xcb, "slhhhr", IF_RRFa, F_Z),
1178         INSTR(0xcd, "chhr",   IF_RRE,  F_Z),
1179         INSTR(0xcf, "clhhr",  IF_RRE,  F_Z),
1180         INSTR(0xd8, "ahhlr",  IF_RRFa, F_Z),
1181         INSTR(0xd9, "shhlr",  IF_RRFa, F_Z),
1182         INSTR(0xda, "alhhlr", IF_RRFa, F_Z),
1183         INSTR(0xdb, "slhhlr", IF_RRFa, F_Z),
```

```
1184         INSTR(0xdd, "chlr",   IF_RRE,  F_Z),
1185         INSTR(0xdf, "clhlr",  IF_RRE,  F_Z),
1186         INSTR(0xe1, "popcnt", IF_RRE,  F_Z),
1187         INSTR(0xe2, "locgr",  IF_RRFc, F_Z),
1188         INSTR(0xe4, "ngrk",   IF_RRFa, F_Z),
1189         INSTR(0xe6, "ogrk",   IF_RRFa, F_Z),
1190         INSTR(0xe7, "xgrk",   IF_RRFa, F_Z),
1191         INSTR(0xe8, "agrk",   IF_RRFa, F_Z),
1192         INSTR(0xe9, "sgrk",   IF_RRFa, F_Z),
1193         INSTR(0xea, "algrk",  IF_RRFa, F_Z),
1194         INSTR(0xeb, "slgrk",  IF_RRFa, F_Z),
1195         INSTR(0xf2, "locgr",  IF_RRFc, F_Z),
1196         INSTR(0xf4, "nrk",    IF_RRFa, F_Z),
1197         INSTR(0xf6, "ork",    IF_RRFa, F_Z),
1198         INSTR(0xf7, "xrk",    IF_RRFa, F_Z),
1199         INSTR(0xf8, "ark",    IF_RRFa, F_Z),
1200         INSTR(0xf9, "srk",    IF_RRFa, F_Z),
1201         INSTR(0xfa, "alrk",   IF_RRFa, F_Z),
1202         INSTR(0xfb, "slrk",   IF_RRFa, F_Z),
1203 };

1205 static const struct inst_table tbl_c0x[16] = {
1206         INSTR(0x0, "larl",   IF_RILb, F_390 | F_Z),
1207         INSTR(0x1, "lgfi",   IF_RILa, F_Z),
1208         INSTR(0x4, "brcl",   IF_RILc, F_390 | F_Z),
1209         INSTR(0x5, "brasl",  IF_RILb, F_390 | F_Z),
1210         INSTR(0x6, "xihf",   IF_RILa, F_Z),
1211         INSTR(0x7, "xilf",   IF_RILa, F_Z),
1212         INSTR(0x8, "iihf",   IF_RILa, F_Z),
1213         INSTR(0x9, "iilf",   IF_RILa, F_Z),
1214         INSTR(0xa, "nihf",   IF_RILa, F_Z),
1215         INSTR(0xb, "nilf",   IF_RILa, F_Z),
1216         INSTR(0xc, "oihf",   IF_RILa, F_Z),
1217         INSTR(0xd, "oilf",   IF_RILa, F_Z),
1218         INSTR(0xe, "llihf",  IF_RILa, F_Z),
1219         INSTR(0xf, "llilf",  IF_RILa, F_Z),
1220 };

1222 static const struct inst_table tbl_c2x[16] = {
1223         INSTR(0x0, "msgfi",  IF_RILa, F_Z),
1224         INSTR(0x1, "msfi",   IF_RILa, F_Z),
1225         INSTR(0x4, "slgfi",  IF_RILa, F_Z),
1226         INSTR(0x5, "slfi",   IF_RILa, F_Z),
1227         INSTR(0x8, "agfi",   IF_RILa, F_Z),
1228         INSTR(0x9, "afi",    IF_RILa, F_Z),
1229         INSTR(0xa, "algfi",  IF_RILa, F_Z),
1230         INSTR(0xb, "alfi",   IF_RILa, F_Z),
1231         INSTR(0xc, "cgfi",   IF_RILa, F_Z),
1232         INSTR(0xd, "cfi",    IF_RILa, F_Z),
1233         INSTR(0xe, "clgfi",  IF_RILa, F_Z),
1234         INSTR(0xf, "clfi",   IF_RILa, F_Z),
1235 };

1237 static const struct inst_table tbl_c4x[16] = {
1238         INSTR(0x2, "llhrl",  IF_RILb, F_Z),
1239         INSTR(0x4, "lghrl",  IF_RILb, F_Z),
1240         INSTR(0x5, "lhrl",   IF_RILb, F_Z),
1241         INSTR(0x6, "llghrl", IF_RILb, F_Z),
1242         INSTR(0x7, "sthrl",  IF_RILb, F_Z),
1243         INSTR(0x8, "lgrl",   IF_RILb, F_Z),
1244         INSTR(0xb, "stgrl",  IF_RILb, F_Z),
1245         INSTR(0xc, "lgfrl",  IF_RILb, F_Z),
1246         INSTR(0xd, "lrl",    IF_RILb, F_Z),
1247         INSTR(0xe, "llgfrl", IF_RILb, F_Z),
1248         INSTR(0xf, "strl",   IF_RILb, F_Z),
1249 };
```



```
1251 static const struct inst_table tbl_c6x[16] = {
1252         INSTR(0x0, "exrl",   IF_RILb, F_Z),
1253         INSTR(0x2, "pfdrl",  IF_RILc, F_Z),
1254         INSTR(0x4, "cghrl",  IF_RILb, F_Z),
1255         INSTR(0x5, "chrl",   IF_RILb, F_Z),
1256         INSTR(0x6, "clghrl", IF_RILb, F_Z),
1257         INSTR(0x7, "clhrl",  IF_RILb, F_Z),
1258         INSTR(0x8, "cgrl",   IF_RILb, F_Z),
1259         INSTR(0xa, "clgrl",  IF_RILb, F_Z),
1260         INSTR(0xc, "cgfrl",  IF_RILb, F_Z),
1261         INSTR(0xd, "crl",    IF_RILb, F_Z),
1262         INSTR(0xe, "clgfrl", IF_RILb, F_Z),
1263         INSTR(0xf, "clrl",   IF_RILb, F_Z),
1264 };

1266 static const struct inst_table tbl_c8x[16] = {
1267         INSTR(0x0, "mvcos",  IF_SSF, F_Z),
1268         INSTR(0x1, "ectg",   IF_SSF, F_Z),
1269         INSTR(0x2, "csst",   IF_SSF, F_Z),
1270         INSTR(0x4, "lpd",    IF_SSF, F_Z),
1271         INSTR(0x5, "lpdg",   IF_SSF, F_Z),
1272 };

1274 static const struct inst_table tbl_ccx[16] = {
1275         INSTR(0x6, "brcth",  IF_RILb, F_Z),
1276         INSTR(0x8, "aih",    IF_RILa, F_Z),
1277         INSTR(0xa, "alsih",  IF_RILa, F_Z),
1278         INSTR(0xb, "alsihn", IF_RILa, F_Z),
1279         INSTR(0xd, "cih",    IF_RILa, F_Z),
1280         INSTR(0xf, "clih",   IF_RILa, F_Z),
1281 };

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

```
1316         INSTR(0x2f, "strvg",  IF_RXYa, F_Z),
1317         INSTR(0x30, "cgf",    IF_RXYa, F_Z),
1318         INSTR(0x31, "clgf",   IF_RXYa, F_Z),
1319         INSTR(0x32, "ltgf",   IF_RXYa, F_Z),
1320         INSTR(0x34, "cgh",    IF_RXYa, F_Z),
1321         INSTR(0x36, "pfd",    IF_RXYb, F_Z),
1322         INSTR(0x3e, "strv",   IF_RXYa, F_390 | F_Z),
1323         INSTR(0x3f, "strvh",  IF_RXYa, F_390 | F_Z),
1324         INSTR(0x46, "bctg",   IF_RXYa, F_Z),
1325         INSTR(0x50, "sty",    IF_RXYa, F_Z),
1326         INSTR(0x51, "msy",    IF_RXYa, F_Z),
1327         INSTR(0x54, "ny",     IF_RXYa, F_Z),
1328         INSTR(0x55, "cly",    IF_RXYa, F_Z),
1329         INSTR(0x56, "oy",     IF_RXYa, F_Z),
1330         INSTR(0x57, "xy",     IF_RXYa, F_Z),
1331         INSTR(0x58, "ly",     IF_RXYa, F_Z),
1332         INSTR(0x59, "cy",     IF_RXYa, F_Z),
1333         INSTR(0x5a, "ay",     IF_RXYa, F_Z),
1334         INSTR(0x5b, "sy",     IF_RXYa, F_Z),
1335         INSTR(0x5c, "mfy",    IF_RXYa, F_Z),
1336         INSTR(0x5e, "aly",    IF_RXYa, F_Z),
1337         INSTR(0x5f, "sly",    IF_RXYa, F_Z),
1338         INSTR(0x70, "sthy",   IF_RXYa, F_Z),
1339         INSTR(0x71, "lay",    IF_RXYa, F_Z),
1340         INSTR(0x72, "stcy",   IF_RXYa, F_Z),
1341         INSTR(0x73, "icy",    IF_RXYa, F_Z),
1342         INSTR(0x75, "laey",   IF_RXYa, F_Z),
1343         INSTR(0x76, "lb",     IF_RXYa, F_Z),
1344         INSTR(0x77, "lgb",    IF_RXYa, F_Z),
1345         INSTR(0x78, "lhy",    IF_RXYa, F_Z),
1346         INSTR(0x79, "chy",    IF_RXYa, F_Z),
1347         INSTR(0x7a, "ahy",    IF_RXYa, F_Z),
1348         INSTR(0x7b, "shy",    IF_RXYa, F_Z),
1349         INSTR(0x7c, "mhy",    IF_RXYa, F_Z),
1350         INSTR(0x80, "ng",     IF_RXYa, F_Z),
1351         INSTR(0x81, "og",     IF_RXYa, F_Z),
1352         INSTR(0x82, "xg",     IF_RXYa, F_Z),
1353         INSTR(0x85, "lgat",   IF_RXYa, F_Z),
1354         INSTR(0x86, "mlg",    IF_RXYa, F_Z),
1355         INSTR(0x87, "dlg",    IF_RXYa, F_Z),
1356         INSTR(0x88, "alcg",   IF_RXYa, F_Z),
1357         INSTR(0x89, "slbg",   IF_RXYa, F_Z),
1358         INSTR(0x8e, "stpq",   IF_RXYa, F_Z),
1359         INSTR(0x8f, "lpq",    IF_RXYa, F_Z),
1360         INSTR(0x90, "llgc",   IF_RXYa, F_Z),
1361         INSTR(0x91, "llgh",   IF_RXYa, F_Z),
1362         INSTR(0x94, "llc",    IF_RXYa, F_Z),
1363         INSTR(0x95, "llh",    IF_RXYa, F_Z),
1364         INSTR(0x96, "ml",     IF_RXYa, F_390 | F_Z),
1365         INSTR(0x97, "dl",     IF_RXYa, F_390 | F_Z),
1366         INSTR(0x98, "alc",    IF_RXYa, F_390 | F_Z),
1367         INSTR(0x99, "slb",    IF_RXYa, F_390 | F_Z),
1368         INSTR(0x9c, "llgtat", IF_RXYa, F_Z),
1369         INSTR(0x9d, "llgfat", IF_RXYa, F_Z),
1370         INSTR(0x9f, "lat",    IF_RXYa, F_Z),
1371         INSTR(0xc0, "lbh",    IF_RXYa, F_Z),
1372         INSTR(0xc2, "llch",   IF_RXYa, F_Z),
1373         INSTR(0xc3, "stch",   IF_RXYa, F_Z),
1374         INSTR(0xc4, "lhh",    IF_RXYa, F_Z),
1375         INSTR(0xc6, "llhh",   IF_RXYa, F_Z),
1376         INSTR(0xc7, "sthh",   IF_RXYa, F_Z),
1377         INSTR(0xc8, "lfhat",  IF_RXYa, F_Z),
1378         INSTR(0xca, "lfh",    IF_RXYa, F_Z),
1379         INSTR(0xcb, "stfh",   IF_RXYa, F_Z),
1380         INSTR(0xcd, "chf",    IF_RXYa, F_Z),
1381         INSTR(0xcf, "clhf",   IF_RXYa, F_Z),
```



```
1382 };

1384 static const struct inst_table tbl_e5xx[256] = {
1385         INSTR(0x00, "lasp",    IF_SSE, F_390 | F_Z),
1386         INSTR(0x01, "tprot",   IF_SSE, F_390 | F_Z),
1387         INSTR(0x02, "strag",   IF_SSE, F_Z),
1388         INSTR(0x0e, "mvcsk",   IF_SSE, F_390 | F_Z),
1389         INSTR(0x0f, "mvcdk",   IF_SSE, F_390 | F_Z),
1390         INSTR(0x44, "mvhhi",   IF_SIL, F_Z),
1391         INSTR(0x48, "mvghi",   IF_SIL, F_Z),
1392         INSTR(0x4c, "mvhi",    IF_SIL, F_Z),
1393         INSTR(0x54, "chhsi",   IF_SIL, F_Z),
1394         INSTR(0x55, "clhhsi",  IF_SIL, F_Z),
1395         INSTR(0x58, "cghsi",   IF_SIL, F_Z),
1396         INSTR(0x59, "clghsi",  IF_SIL, F_Z),
1397         INSTR(0x5c, "chsi",    IF_SIL, F_Z),
1398         INSTR(0x5d, "clfhsi",  IF_SIL, F_Z),
1399         INSTR(0x60, "tbegin",  IF_SIL, F_Z),
1400         INSTR(0x61, "tbeginc", IF_SIL, F_Z),
1401 };

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

```
1448         INSTR(0xdd, "slak",  IF_RSYa, F_Z),
1449         INSTR(0xde, "srlk",  IF_RSYa, F_Z),
1450         INSTR(0xdf, "sllk",  IF_RSYa, F_Z),
1451         INSTR(0xe2, "locg",  IF_RSYb, F_Z),
1452         INSTR(0xe3, "stocg", IF_RSYb, F_Z),
1453         INSTR(0xe4, "lang",  IF_RSYa, F_Z),
1454         INSTR(0xe6, "laog",  IF_RSYa, F_Z),
1455         INSTR(0xe7, "laxg",  IF_RSYa, F_Z),
1456         INSTR(0xe8, "laag",  IF_RSYa, F_Z),
1457         INSTR(0xea, "laalg", IF_RSYa, F_Z),
1458         INSTR(0xf2, "loc",   IF_RSYb, F_Z),
1459         INSTR(0xf3, "stoc",  IF_RSYb, F_Z),
1460         INSTR(0xf4, "lan",   IF_RSYa, F_Z),
1461         INSTR(0xf6, "lao",   IF_RSYa, F_Z),
1462         INSTR(0xf7, "lax",   IF_RSYa, F_Z),
1463         INSTR(0xf8, "laa",   IF_RSYa, F_Z),
1464         INSTR(0xfa, "laal",  IF_RSYa, F_Z),
1465 };

1467 static const struct inst_table tbl_ecxx[256] = {
1468         INSTR(0x44, "brxhg",   IF_RIEe, F_Z),
1469         INSTR(0x45, "brxlg",   IF_RIEe, F_Z),
1470         INSTR(0x51, "risblg",  IF_RIEf, F_Z),
1471         INSTR(0x54, "rnsbg",   IF_RIEf, F_Z),
1472         INSTR(0x55, "risbg",   IF_RIEf, F_Z),
1473         INSTR(0x56, "rosbg",   IF_RIEf, F_Z),
1474         INSTR(0x57, "rxsbg",   IF_RIEf, F_Z),
1475         INSTR(0x59, "risbgn",  IF_RIEf, F_Z),
1476         INSTR(0x5d, "risbhg",  IF_RIEf, F_Z),
1477         INSTR(0x64, "cgrj",    IF_RIEb, F_Z),
1478         INSTR(0x65, "clgrj",   IF_RIEb, F_Z),
1479         INSTR(0x70, "cgit",    IF_RIEa, F_Z),
1480         INSTR(0x71, "clgit",   IF_RIEa, F_Z),
1481         INSTR(0x72, "cit",     IF_RIEa, F_Z),
1482         INSTR(0x73, "clfit",   IF_RIEa, F_Z),
1483         INSTR(0x76, "crj",     IF_RIEb, F_Z),
1484         INSTR(0x77, "clrj",    IF_RIEb, F_Z),
1485         INSTR(0x7c, "cgij",    IF_RIEc, F_Z),
1486         INSTR(0x7d, "clgij",   IF_RIEc, F_Z),
1487         INSTR(0x7e, "cij",     IF_RIEc, F_Z),
1488         INSTR(0x7f, "clij",    IF_RIEc, F_Z),
1489         INSTR(0xd8, "ahik",    IF_RIEd, F_Z),
1490         INSTR(0xd9, "aghik",   IF_RIEd, F_Z),
1491         INSTR(0xda, "alhsik",  IF_RIEd, F_Z),
1492         INSTR(0xdb, "alghsik", IF_RIEd, F_Z),
1493         INSTR(0xe4, "cgrb",    IF_RRS,  F_Z),
1494         INSTR(0xe5, "clgrb",   IF_RRS,  F_Z),
1495         INSTR(0xf6, "crb",     IF_RRS,  F_Z),
1496         INSTR(0xf7, "clrb",    IF_RRS,  F_Z),
1497         INSTR(0xfc, "cgib",    IF_RIS,  F_Z),
1498         INSTR(0xfd, "clgib",   IF_RIS,  F_Z),
1499         INSTR(0xfe, "cib",     IF_RIS,  F_Z),
1500         INSTR(0xff, "clib",    IF_RIS,  F_Z),
1501 };

1503 static const struct inst_table tbl_edxx[256] = {
1504         INSTR(0x04, "ldeb",   IF_RXE,  F_390 | F_Z),
1505         INSTR(0x05, "lxdb",   IF_RXE,  F_390 | F_Z),
1506         INSTR(0x06, "lxeb",   IF_RXE,  F_390 | F_Z),
1507         INSTR(0x07, "mxdb",   IF_RXE,  F_390 | F_Z),
1508         INSTR(0x08, "keb",    IF_RXE,  F_390 | F_Z),
1509         INSTR(0x09, "ceb",    IF_RXE,  F_390 | F_Z),
1510         INSTR(0x0a, "aeb",    IF_RXE,  F_390 | F_Z),
1511         INSTR(0x0b, "seb",    IF_RXE,  F_390 | F_Z),
1512         INSTR(0x0c, "mdeb",   IF_RXE,  F_390 | F_Z),
1513         INSTR(0x0d, "deb",    IF_RXE,  F_390 | F_Z),
```



```
1514         INSTR(0x0e, "maeb",   IF_RXF,  F_390 | F_Z),
1515         INSTR(0x0f, "mseb",   IF_RXF,  F_390 | F_Z),
1516         INSTR(0x10, "tceb",   IF_RXE,  F_390 | F_Z),
1517         INSTR(0x11, "tcdb",   IF_RXE,  F_390 | F_Z),
1518         INSTR(0x12, "tcxb",   IF_RXE,  F_390 | F_Z),
1519         INSTR(0x14, "sqeb",   IF_RXE,  F_390 | F_Z),
1520         INSTR(0x15, "sqdb",   IF_RXE,  F_390 | F_Z),
1521         INSTR(0x17, "meeb",   IF_RXE,  F_390 | F_Z),
1522         INSTR(0x18, "kdb",    IF_RXE,  F_390 | F_Z),
1523         INSTR(0x19, "cdb",    IF_RXE,  F_390 | F_Z),
1524         INSTR(0x1a, "adb",    IF_RXE,  F_390 | F_Z),
1525         INSTR(0x1b, "sdb",    IF_RXE,  F_390 | F_Z),
1526         INSTR(0x1c, "mdb",    IF_RXE,  F_390 | F_Z),
1527         INSTR(0x1d, "ddb",    IF_RXE,  F_390 | F_Z),
1528         INSTR(0x1e, "madb",   IF_RXF,  F_390 | F_Z),
1529         INSTR(0x1f, "msdb",   IF_RXF,  F_390 | F_Z),
1530         INSTR(0x24, "lde",    IF_RXE,  F_390 | F_Z),
1531         INSTR(0x25, "lxd",    IF_RXE,  F_390 | F_Z),
1532         INSTR(0x26, "lxe",    IF_RXE,  F_390 | F_Z),
1533         INSTR(0x2e, "mae",    IF_RXF,  F_390 | F_Z),
1534         INSTR(0x2f, "mse",    IF_RXF,  F_390 | F_Z),
1535         INSTR(0x34, "sqe",    IF_RXE,  F_390 | F_Z),
1536         INSTR(0x35, "sqd",    IF_RXE,  F_390 | F_Z),
1537         INSTR(0x37, "mee",    IF_RXE,  F_390 | F_Z),
1538         INSTR(0x38, "mayl",   IF_RXF,  F_Z),
1539         INSTR(0x39, "myl",    IF_RXF,  F_Z),
1540         INSTR(0x3a, "may",    IF_RXF,  F_Z),
1541         INSTR(0x3b, "my",     IF_RXF,  F_Z),
1542         INSTR(0x3c, "mayh",   IF_RXF,  F_Z),
1543         INSTR(0x3d, "myh",    IF_RXF,  F_Z),
1544         INSTR(0x3e, "mad",    IF_RXF,  F_390 | F_Z),
1545         INSTR(0x3f, "msd",    IF_RXF,  F_390 | F_Z),
1546         INSTR(0x40, "sldt",   IF_RXF,  F_Z),
1547         INSTR(0x41, "srdt",   IF_RXF,  F_Z),
1548         INSTR(0x48, "slxt",   IF_RXF,  F_Z),
1549         INSTR(0x49, "srxt",   IF_RXF,  F_Z),
1550         INSTR(0x50, "tdcet",  IF_RXE,  F_Z),
1551         INSTR(0x51, "tdget",  IF_RXE,  F_Z),
1552         INSTR(0x54, "tdcdt",  IF_RXE,  F_Z),
1553         INSTR(0x55, "tdgdt",  IF_RXE,  F_Z),
1554         INSTR(0x58, "tdcxt",  IF_RXE,  F_Z),
1555         INSTR(0x59, "tdgxt",  IF_RXE,  F_Z),
1556         INSTR(0x64, "ley",    IF_RXYa, F_Z),
1557         INSTR(0x65, "ldy",    IF_RXYa, F_Z),
1558         INSTR(0x66, "stey",   IF_RXYa, F_Z),
1559         INSTR(0x67, "stdy",   IF_RXYa, F_Z),
1560         INSTR(0xa8, "czdt",   IF_RSLb, F_Z),
1561         INSTR(0xa9, "czxt",   IF_RSLb, F_Z),
1562         INSTR(0xaa, "cdzt",   IF_RSLb, F_Z),
1563         INSTR(0xab, "cxzt",   IF_RSLb, F_Z),
1564 };

1566 static const struct inst_table tbl_xx[256] = {
1567         INSTR(0x00, ".byte", IF_ZERO, F_370 | F_390 | F_Z),
1568         TABLE(0x01, tbl_01xx, 1, 0, 0xff),
1569         INSTR(0x04, "spm",    IF_RR,   F_370 | F_Z),
1570         INSTR(0x05, "balr",   IF_RR,   F_370 | F_Z),
1571         INSTR(0x06, "bctr",   IF_RR,   F_370 | F_Z),
1572         TABLE(0x07, tbl_07, 1, 4, 0x0f),
1573         INSTR(0x08, "ssk",    IF_RR,   F_370),
1574         INSTR(0x09, "isk",    IF_RR,   F_370),
1575         INSTR(0x0a, "svc",    IF_I,    F_370 | F_390 | F_Z),
1576         INSTR(0x0b, "bsm",    IF_RR,   F_390 | F_Z),
1577         INSTR(0x0c, "bassm",  IF_RR,   F_390 | F_Z),
1578         INSTR(0x0d, "basr",   IF_RR,   F_390 | F_Z),
1579         INSTR(0x0e, "mvcl",   IF_RR,   F_370 | F_390 | F_Z),
```

```
1580        INSTR(0x0f, "clcl",  IF_RR,   F_370 | F_390 | F_Z),
1581        INSTR(0x10, "lpr",   IF_RR,   F_370 | F_390 | F_Z),
1582        INSTR(0x11, "lnr",   IF_RR,   F_370 | F_390 | F_Z),
1583        INSTR(0x12, "ltr",   IF_RR,   F_370 | F_390 | F_Z),
1584        INSTR(0x13, "lcr",   IF_RR,   F_370 | F_390 | F_Z),
1585        INSTR(0x14, "nr",    IF_RR,   F_370 | F_390 | F_Z),
1586        INSTR(0x15, "clr",   IF_RR,   F_370 | F_390 | F_Z),
1587        INSTR(0x16, "or",    IF_RR,   F_370 | F_390 | F_Z),
1588        INSTR(0x17, "xr",    IF_RR,   F_370 | F_390 | F_Z),
1589        INSTR(0x18, "lr",    IF_RR,   F_370 | F_390 | F_Z),
1590        INSTR(0x19, "cr",    IF_RR,   F_370 | F_390 | F_Z),
1591        INSTR(0x1a, "ar",    IF_RR,   F_370 | F_390 | F_Z),
1592        INSTR(0x1b, "sr",    IF_RR,   F_370 | F_390 | F_Z),
1593        INSTR(0x1c, "mr",    IF_RR,   F_370 | F_390 | F_Z),
1594        INSTR(0x1d, "dr",    IF_RR,   F_370 | F_390 | F_Z),
1595        INSTR(0x1e, "alr",   IF_RR,   F_370 | F_390 | F_Z),
1596        INSTR(0x1f, "slr",   IF_RR,   F_370 | F_390 | F_Z),
1597        INSTR(0x20, "lpdr",  IF_RR,   F_370 | F_390 | F_Z),
1598        INSTR(0x21, "lndr",  IF_RR,   F_370 | F_390 | F_Z),
1599        INSTR(0x22, "ltdr",  IF_RR,   F_370 | F_390 | F_Z),
1600        INSTR(0x23, "lcdr",  IF_RR,   F_370 | F_390 | F_Z),
1601        INSTR(0x24, "hdr",   IF_RR,   F_370 | F_390 | F_Z),
1602        MULTI(0x25, tbl_25),
1603        INSTR(0x26, "mxr",   IF_RR,   F_370 | F_390 | F_Z),
1604        INSTR(0x27, "mxdr",  IF_RR,   F_370 | F_390 | F_Z),
1605        INSTR(0x28, "ldr",   IF_RR,   F_370 | F_390 | F_Z),
1606        INSTR(0x29, "cdr",   IF_RR,   F_370 | F_390 | F_Z),
1607        INSTR(0x2a, "adr",   IF_RR,   F_370 | F_390 | F_Z),
1608        INSTR(0x2b, "sdr",   IF_RR,   F_370 | F_390 | F_Z),
1609        INSTR(0x2c, "mdr",   IF_RR,   F_370 | F_390 | F_Z),
1610        INSTR(0x2d, "ddr",   IF_RR,   F_370 | F_390 | F_Z),
1611        INSTR(0x2e, "awr",   IF_RR,   F_370 | F_390 | F_Z),
1612        INSTR(0x2f, "swr",   IF_RR,   F_370 | F_390 | F_Z),
1613        INSTR(0x30, "lper",  IF_RR,   F_370 | F_390 | F_Z),
1614        INSTR(0x31, "lner",  IF_RR,   F_370 | F_390 | F_Z),
1615        INSTR(0x32, "lter",  IF_RR,   F_370 | F_390 | F_Z),
1616        INSTR(0x33, "lcer",  IF_RR,   F_370 | F_390 | F_Z),
1617        INSTR(0x34, "her",   IF_RR,   F_370 | F_390 | F_Z),
1618        MULTI(0x35, tbl_35),
1619        INSTR(0x36, "axr",   IF_RR,   F_370 | F_390 | F_Z),
1620        INSTR(0x37, "sxr",   IF_RR,   F_370 | F_390 | F_Z),
1621        INSTR(0x38, "ler",   IF_RR,   F_370 | F_390 | F_Z),
1622        INSTR(0x39, "cer",   IF_RR,   F_370 | F_390 | F_Z),
1623        INSTR(0x3a, "aer",   IF_RR,   F_370 | F_390 | F_Z),
1624        INSTR(0x3b, "ser",   IF_RR,   F_370 | F_390 | F_Z),
1625        MULTI(0x3c, tbl_3c),
1626        INSTR(0x3d, "der",   IF_RR,   F_370 | F_390 | F_Z),
1627        INSTR(0x3e, "aur",   IF_RR,   F_370 | F_390 | F_Z),
1628        INSTR(0x3f, "sur",   IF_RR,   F_370 | F_390 | F_Z),
1629        INSTR(0x40, "sth",   IF_RXa,  F_370 | F_390 | F_Z),
1630        INSTR(0x41, "la",    IF_RXa,  F_370 | F_390 | F_Z),
1631        INSTR(0x42, "stc",   IF_RXa,  F_370 | F_390 | F_Z),
1632        INSTR(0x43, "ic",    IF_RXa,  F_370 | F_390 | F_Z),
1633        INSTR(0x44, "ex",    IF_RXa,  F_370 | F_390 | F_Z),
1634        INSTR(0x45, "bal",   IF_RXa,  F_370 | F_390 | F_Z),
1635        INSTR(0x46, "bct",   IF_RXa,  F_370 | F_390 | F_Z),
1636        TABLE(0x47, tbl_47, 1, 4, 0x0f),
1637        INSTR(0x48, "lh",    IF_RXa,  F_370 | F_390 | F_Z),
1638        INSTR(0x49, "ch",    IF_RXa,  F_370 | F_390 | F_Z),
1639        INSTR(0x4a, "ah",    IF_RXa,  F_370 | F_390 | F_Z),
1640        INSTR(0x4b, "sh",    IF_RXa,  F_370 | F_390 | F_Z),
1641        INSTR(0x4c, "mh",    IF_RXa,  F_370 | F_390 | F_Z),
1642        INSTR(0x4d, "bas",   IF_RXa,  F_390 | F_Z),
1643        INSTR(0x4e, "cvd",   IF_RXa,  F_370 | F_390 | F_Z),
1644        INSTR(0x4f, "cvb",   IF_RXa,  F_370 | F_390 | F_Z),
1645        INSTR(0x50, "st",    IF_RXa,  F_370 | F_390 | F_Z),
```



```
1646        INSTR(0x51, "lae",   IF_RXa,  F_390 | F_Z),
1647        INSTR(0x54, "n",     IF_RXa,  F_370 | F_390 | F_Z),
1648        INSTR(0x55, "cl",    IF_RXa,  F_370 | F_390 | F_Z),
1649        INSTR(0x56, "o",     IF_RXa,  F_370 | F_390 | F_Z),
1650        INSTR(0x57, "x",     IF_RXa,  F_370 | F_390 | F_Z),
1651        INSTR(0x58, "l",     IF_RXa,  F_370 | F_390 | F_Z),
1652        INSTR(0x59, "c",     IF_RXa,  F_370 | F_390 | F_Z),
1653        INSTR(0x5a, "a",     IF_RXa,  F_370 | F_390 | F_Z),
1654        INSTR(0x5b, "s",     IF_RXa,  F_370 | F_390 | F_Z),
1655        INSTR(0x5c, "m",     IF_RXa,  F_370 | F_390 | F_Z),
1656        INSTR(0x5d, "d",     IF_RXa,  F_370 | F_390 | F_Z),
1657        INSTR(0x5e, "al",    IF_RXa,  F_370 | F_390 | F_Z),
1658        INSTR(0x5f, "sl",    IF_RXa,  F_370 | F_390 | F_Z),
1659        INSTR(0x60, "std",   IF_RXa,  F_370 | F_390 | F_Z),
1660        INSTR(0x67, "mxd",   IF_RXa,  F_370 | F_390 | F_Z),
1661        INSTR(0x68, "ld",    IF_RXa,  F_370 | F_390 | F_Z),
1662        INSTR(0x69, "cd",    IF_RXa,  F_370 | F_390 | F_Z),
1663        INSTR(0x6a, "ad",    IF_RXa,  F_370 | F_390 | F_Z),
1664        INSTR(0x6b, "sd",    IF_RXa,  F_370 | F_390 | F_Z),
1665        INSTR(0x6c, "md",    IF_RXa,  F_370 | F_390 | F_Z),
1666        INSTR(0x6d, "dd",    IF_RXa,  F_370 | F_390 | F_Z),
1667        INSTR(0x6e, "aw",    IF_RXa,  F_370 | F_390 | F_Z),
1668        INSTR(0x6f, "sw",    IF_RXa,  F_370 | F_390 | F_Z),
1669        INSTR(0x70, "ste",   IF_RXa,  F_370 | F_390 | F_Z),
1670        INSTR(0x71, "ms",    IF_RXa,  F_390 | F_Z),
1671        INSTR(0x78, "le",    IF_RXa,  F_370 | F_390 | F_Z),
1672        INSTR(0x79, "ce",    IF_RXa,  F_370 | F_390 | F_Z),
1673        INSTR(0x7a, "ae",    IF_RXa,  F_370 | F_390 | F_Z),
1674        INSTR(0x7b, "se",    IF_RXa,  F_370 | F_390 | F_Z),
1675        MULTI(0x7c, tbl_7c),
1676        INSTR(0x7d, "de",    IF_RXa,  F_370 | F_390 | F_Z),
1677        INSTR(0x7e, "au",    IF_RXa,  F_370 | F_390 | F_Z),
1678        INSTR(0x7f, "su",    IF_RXa,  F_370 | F_390 | F_Z),
1679        INSTR(0x80, "ssm",   IF_S,    F_370 | F_390 | F_Z),
1680        INSTR(0x82, "lpsw",  IF_S,    F_370 | F_390 | F_Z),
1681        INSTR(0x83, "diag",  IF_DIAG, F_370 | F_390 | F_Z),
1682        MULTI(0x84, tbl_84),
1683        MULTI(0x85, tbl_85),
1684        INSTR(0x86, "bxh",   IF_RSa,  F_370 | F_390 | F_Z),
1685        INSTR(0x87, "bxle",  IF_RSa,  F_370 | F_390 | F_Z),
1686        INSTR(0x88, "srl",   IF_RSa,  F_370 | F_390 | F_Z),
1687        INSTR(0x89, "sll",   IF_RSa,  F_370 | F_390 | F_Z),
1688        INSTR(0x8a, "sra",   IF_RSa,  F_370 | F_390 | F_Z),
1689        INSTR(0x8b, "sla",   IF_RSa,  F_370 | F_390 | F_Z),
1690        INSTR(0x8c, "srdl",  IF_RSa,  F_370 | F_390 | F_Z),
1691        INSTR(0x8d, "sldl",  IF_RSa,  F_370 | F_390 | F_Z),
1692        INSTR(0x8e, "srda",  IF_RSa,  F_370 | F_390 | F_Z),
1693        INSTR(0x8f, "slda",  IF_RSa,  F_370 | F_390 | F_Z),
1694        INSTR(0x90, "stm",   IF_RSa,  F_370 | F_390 | F_Z),
1695        INSTR(0x91, "tm",    IF_SI,   F_370 | F_390 | F_Z),
1696        INSTR(0x92, "mvi",   IF_SI,   F_370 | F_390 | F_Z),
1697        INSTR(0x93, "ts",    IF_S,    F_370 | F_390 | F_Z),
1698        INSTR(0x94, "ni",    IF_SI,   F_370 | F_390 | F_Z),
1699        INSTR(0x95, "cli",   IF_SI,   F_370 | F_390 | F_Z),
1700        INSTR(0x96, "oi",    IF_SI,   F_370 | F_390 | F_Z),
1701        INSTR(0x97, "xi",    IF_SI,   F_370 | F_390 | F_Z),
1702        INSTR(0x98, "lm",    IF_RSa,  F_370 | F_390 | F_Z),
1703        INSTR(0x99, "trace", IF_RSa,  F_390 | F_Z),
1704        INSTR(0x9a, "lam",   IF_RSa,  F_390 | F_Z),
1705        INSTR(0x9b, "stam",  IF_RSa,  F_390 | F_Z),
1706        TABLE(0xa5, tbl_a5x, 1, 0, 0x0f),
1707        TABLE(0xa7, tbl_a7x, 1, 0, 0x0f),
1708        INSTR(0xa8, "mvcle", IF_RSa,  F_390 | F_Z),
1709        INSTR(0xa9, "clcle", IF_RSa,  F_390 | F_Z),
1710        INSTR(0xac, "stnsm", IF_SI,   F_370 | F_390 | F_Z),
1711        INSTR(0xad, "stosm", IF_SI,   F_370 | F_390 | F_Z),
```

```
1712         INSTR(0xae, "sigp",  IF_RSa,  F_370 | F_390 | F_Z),
1713         INSTR(0xaf, "mc",    IF_SI,   F_370 | F_390 | F_Z),
1714         INSTR(0xb1, "lra",   IF_RXa,  F_370 | F_390 | F_Z),
1715         TABLE(0xb2, tbl_b2xx, 1, 0, 0xff),
1716         TABLE(0xb3, tbl_b3xx, 1, 0, 0xff),
1717         INSTR(0xb6, "stctl", IF_RSa,  F_370 | F_390 | F_Z | F_CTL_REG),
1718         INSTR(0xb7, "lctl",  IF_RSa,  F_370 | F_390 | F_Z | F_CTL_REG),
1719         TABLE(0xb9, tbl_b9xx, 1, 0, 0xff),
1720         INSTR(0xba, "cs",    IF_RSa,  F_370 | F_390 | F_Z),
1721         INSTR(0xbb, "cds",   IF_RSa,  F_370 | F_390 | F_Z),
1722         INSTR(0xbd, "clm",   IF_RSb,  F_370 | F_390 | F_Z),
1723         INSTR(0xbe, "stcm",  IF_RSb,  F_370 | F_390 | F_Z),
1724         INSTR(0xbf, "icm",   IF_RSb,  F_370 | F_390 | F_Z),
1725         TABLE(0xc0, tbl_c0x, 1, 0, 0x0f),
1726         TABLE(0xc2, tbl_c2x, 1, 0, 0x0f),
1727         TABLE(0xc4, tbl_c4x, 1, 0, 0x0f),
1728         INSTR(0xc5, "bprp",  IF_MII,  F_Z),
1729         TABLE(0xc6, tbl_c6x, 1, 0, 0x0f),
1730         INSTR(0xc7, "bpp",   IF_SMI,  F_Z),
1731         TABLE(0xc8, tbl_c8x, 1, 0, 0x0f),
1732         TABLE(0xcc, tbl_ccx, 1, 0, 0x0f),
1733         INSTR(0xd0, "trtr",  IF_SSa,  F_Z),
1734         INSTR(0xd1, "mvn",   IF_SSa,  F_370 | F_390 | F_Z),
1735         INSTR(0xd2, "mvc",   IF_SSa,  F_370 | F_390 | F_Z),
1736         INSTR(0xd3, "mvz",   IF_SSa,  F_370 | F_390 | F_Z),
1737         INSTR(0xd4, "nc",    IF_SSa,  F_370 | F_390 | F_Z),
1738         INSTR(0xd5, "clc",   IF_SSa,  F_370 | F_390 | F_Z),
1739         INSTR(0xd6, "oc",    IF_SSa,  F_370 | F_390 | F_Z),
1740         INSTR(0xd7, "xc",    IF_SSa,  F_370 | F_390 | F_Z),
1741         INSTR(0xd9, "mvck",  IF_SSd,  F_390 | F_Z),
1742         INSTR(0xda, "mvcp",  IF_SSd,  F_390 | F_Z),
1743         INSTR(0xdb, "mvcs",  IF_SSd,  F_390 | F_Z),
1744         INSTR(0xdc, "tr",    IF_SSa,  F_370 | F_390 | F_Z),
1745         INSTR(0xdd, "trt",   IF_SSa,  F_370 | F_390 | F_Z),
1746         INSTR(0xde, "ed",    IF_SSa,  F_370 | F_390 | F_Z),
1747         INSTR(0xdf, "edmk",  IF_SSa,  F_370 | F_390 | F_Z),
1748         INSTR(0xe1, "pku",   IF_SSf,  F_390 | F_Z),
1749         INSTR(0xe2, "unpku", IF_SSa,  F_390 | F_Z),
1750         TABLE(0xe3, tbl_e3xx, 5, 0, 0xff),
1751         TABLE(0xe5, tbl_e5xx, 1, 0, 0xff),
1752         INSTR(0xe8, "mvcin", IF_SSa,  F_390 | F_Z),
1753         INSTR(0xe9, "pka",   IF_SSf,  F_390 | F_Z),
1754         INSTR(0xea, "unpka", IF_SSa,  F_390 | F_Z),
1755         TABLE(0xeb, tbl_ebxx, 5, 0, 0xff),
1756         TABLE(0xec, tbl_ecxx, 5, 0, 0xff),
1757         TABLE(0xed, tbl_edxx, 5, 0, 0xff),
1758         INSTR(0xee, "plo",   IF_SSe,  F_390 | F_Z),
1759         INSTR(0xef, "lmd",   IF_SSe,  F_Z),
1760         INSTR(0xf0, "srp",   IF_SSc,  F_370 | F_390 | F_Z),
1761         INSTR(0xf1, "mvo",   IF_SSb,  F_370 | F_390 | F_Z),
1762         INSTR(0xf2, "pack",  IF_SSb,  F_370 | F_390 | F_Z),
1763         INSTR(0xf3, "unpk",  IF_SSb,  F_370 | F_390 | F_Z),
1764         INSTR(0xf8, "zap",   IF_SSb,  F_370 | F_390 | F_Z),
1765         INSTR(0xf9, "cp",    IF_SSb,  F_370 | F_390 | F_Z),
1766         INSTR(0xfa, "ap",    IF_SSb,  F_370 | F_390 | F_Z),
1767         INSTR(0xfb, "sp",    IF_SSb,  F_370 | F_390 | F_Z),
1768         INSTR(0xfc, "mp",    IF_SSb,  F_370 | F_390 | F_Z),
1769         INSTR(0xfd, "dp",    IF_SSb,  F_370 | F_390 | F_Z),
1770 };
1771 /* END CSTYLED */

1773 /* how masks are printed */
1774 static const char *M[16] = {
1775         "0",  "1",  "2",  "3",  "4",  "5",  "6",  "7",
1776         "8",  "9", "10", "11", "12", "13", "14", "15",
1777 };
```



```
1779 /* how general purpose regs are printed */
1780 static const char *R[16] = {
1781         "%r0",  "%r1",  "%r2",  "%r3",  "%r4",  "%r5",  "%r6",  "%r7",
1782         "%r8",  "%r9", "%r10", "%r11", "%r12", "%r13", "%r14", "%r15",
1783 };

1785 /* how control regs are printed */
1786 static const char *C[16] = {
1787         "%c0",  "%c1",  "%c2",  "%c3",  "%c4",  "%c5",  "%c6",  "%c7",
1788         "%c8",  "%c9", "%c10", "%c11", "%c12", "%c13", "%c14", "%c15",
1789 };

1791 /* B and X registers are still registers - print them the same way */
1792 #define B       R
1793 #define X       R

1795 static inline uint32_t
1796 val_8_4_8(uint32_t hi, uint32_t mid, uint32_t lo)
1797 {
1798         ASSERT0(hi & ~0xff);
1799         ASSERT0(mid & ~0xf);
1800         ASSERT0(lo & ~0xff);
1801         return ((hi << 12) | (mid << 8) | lo);
1802 }

1804 static inline uint32_t
1805 val_16_16(uint32_t hi, uint32_t lo)
1806 {
1807         ASSERT0(hi & ~0xffff);
1808         ASSERT0(lo & ~0xffff);
1809         return ((BE_16(hi) << 16) | BE_16(lo));
1810 }

1812 static inline int32_t
1813 sval_16_16(uint32_t hi, uint32_t lo)
1814 {
1815         return (val_16_16(hi, lo));
1816 }

1818 static inline uint32_t
1819 val_8_16(uint32_t hi, uint32_t lo)
1820 {
1821         ASSERT0(hi & ~0xff);
1822         ASSERT0(lo & ~0xffff);
1823         return ((hi << 16) | BE_16(lo));
1824 }

1826 static inline int32_t
1827 sval_8_16(uint32_t hi, uint32_t lo)
1828 {
1829         int32_t tmp = val_8_16(hi, lo);

1831         /* sign extend */
1832         if (tmp & 0x00800000)
1833                 return (0xff000000 | tmp);
1834         return (tmp);
1835 }

1837 static inline uint32_t
1838 val_4_8(uint32_t hi, uint32_t lo)
1839 {
1840         ASSERT0(hi & ~0xf);
1841         ASSERT0(lo & ~0xff);
1842         return ((hi << 8) | lo);
1843 }
```

```
1845 static inline int32_t
1846 sval_4_8(uint32_t hi, uint32_t lo)
1847 {
1848         uint32_t tmp = val_4_8(hi, lo);

1850         /* sign extend */
1851         if (tmp & 0x800)
1852                 return (0xfffff000 | tmp);
1853         return (tmp);
1854 }

1856 /* ARGSUSED */
1857 static void
1858 fmt_zero(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1859 {
1860         (void) snprintf(buf, buflen, "0x00, 0x00");
1861 }

1863 /* ARGSUSED */
1864 static void
1865 fmt_diag(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1866 {
1867         (void) snprintf(buf, buflen, "%#x",
1868             val_8_16(inst->diag.par1, inst->diag.par2));
1869 }

1871 /* ARGSUSED */
1872 static void
1873 fmt_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1874 {
1875         /* nothing to do */
1876 }

1878 /* ARGSUSED */
1879 static void
1880 fmt_i(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1881 {
1882         (void) snprintf(buf, buflen, "%#x", inst->i.i);
1883 }

1885 /* ARGSUSED */
1886 static void
1887 fmt_ie(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1888 {
1889         (void) snprintf(buf, buflen, "%u,%u", inst->ie.i1, inst->ie.i2);
1890 }

1892 /* ARGSUSED */
1893 static void
1894 fmt_mii(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1895 {
1896         uint64_t ri2 = addr + 2 * sval_4_8(inst->mii.ri2h, inst->mii.ri2l);
1897         uint64_t ri3 = addr + 2 * sval_8_16(inst->mii.ri3h, inst->mii.ri3l);

1899         (void) snprintf(buf, buflen, "%s,%#x,%#x", M[inst->mii.m1], ri2, ri3);
1900 }

1902 /* ARGSUSED */
1903 static void
1904 fmt_ril_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1905 {
1906         (void) snprintf(buf, buflen, "%s,%u", R[inst->ril_a.r1],
1907             val_16_16(inst->ril_a.i2h, inst->ril_a.i2l));
1908 }
```



```
1910 /* ARGSUSED */
1911 static void
1912 fmt_ril_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1913 {
1914         uint64_t ri2 = addr + 2 *
1915             sval_16_16(inst->ril_b.ri2h, inst->ril_b.ri2l);

1917         (void) snprintf(buf, buflen, "%s,%#x", R[inst->ril_b.r1], ri2);
1918 }

1920 /* ARGSUSED */
1921 static void
1922 fmt_ril_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1923 {
1924         uint64_t ri2 = addr + 2 *
1925             sval_16_16(inst->ril_c.ri2h, inst->ril_c.ri2l);

1927         (void) snprintf(buf, buflen, "%s,%#x", M[inst->ril_c.m1], ri2);
1928 }

1930 /* ARGSUSED */
1931 static void
1932 fmt_ris(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1933 {
1934         uint32_t d4 = val_4_8(inst->ris.d4h, inst->ris.d4l);

1936         (void) snprintf(buf, buflen, "%s,%u,%s,%u(%s)",
1937             R[inst->ris.r1], inst->ris.i2, M[inst->ris.m3], d4,
1938             B[inst->ris.b4]);
1939 }

1941 /* ARGSUSED */
1942 static void
1943 fmt_ri_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1944 {
1945         uint16_t i2 = BE_16(inst->ri_a.i2);

1947         if (flags & F_SIGNED_IMM)
1948                 (void) snprintf(buf, buflen, "%s,%d", R[inst->ri_a.r1],
1949                     (int16_t)i2);
1950         else
1951                 (void) snprintf(buf, buflen, "%s,%u", R[inst->ri_a.r1],
1952                     i2);
1953 }

1955 /* ARGSUSED */
1956 static void
1957 fmt_ri_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1958 {
1959         uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->ri_b.ri2);

1961         (void) snprintf(buf, buflen, "%s,%#x", R[inst->ri_b.r1], ri2);
1962 }

1964 static void
1965 fmt_ri_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1966 {
1967         uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->ri_c.ri2);

1969         if (flags & F_HIDE_MASK)
1970                 (void) snprintf(buf, buflen, "%#x", ri2);
1971         else
1972                 (void) snprintf(buf, buflen, "%s,%#x", M[inst->ri_c.m1], ri2);
1973 }

1975 /* ARGSUSED */
```

```
1976 static void
1977 fmt_rie_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1978 {
1979         (void) snprintf(buf, buflen, "%s,%u,%s", R[inst->rie_a.r1],
1980             BE_16(inst->rie_a.i2), M[inst->rie_a.m3]);
1981 }

1983 /* ARGSUSED */
1984 static void
1985 fmt_rie_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1986 {
1987         uint64_t ri4 = addr + 2 * (int16_t)BE_16(inst->rie_b.ri4);

1989         (void) snprintf(buf, buflen, "%s,%s,%s,%#x", R[inst->rie_b.r1],
1990             R[inst->rie_b.r2], M[inst->rie_b.m3], ri4);
1991 }

1993 /* ARGSUSED */
1994 static void
1995 fmt_rie_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
1996 {
1997         uint64_t ri4 = addr + 2 * (int16_t)BE_16(inst->rie_c.ri4);

1999         (void) snprintf(buf, buflen, "%s,%u,%s,%#x", R[inst->rie_c.r1],
2000             inst->rie_c.i2, M[inst->rie_c.m3], ri4);
2001 }

2003 /* ARGSUSED */
2004 static void
2005 fmt_rie_d(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2006 {
2007         (void) snprintf(buf, buflen, "%s,%s,%u", R[inst->rie_d.r1],
2008             R[inst->rie_d.r3], BE_16(inst->rie_d.i2));
2009 }

2011 /* ARGSUSED */
2012 static void
2013 fmt_rie_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2014 {
2015         uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->rie_e.ri2);

2017         (void) snprintf(buf, buflen, "%s,%s,%#x", R[inst->rie_e.r1],
2018             R[inst->rie_e.r3], ri2);
2019 }

2021 /* ARGSUSED */
2022 static void
2023 fmt_rie_f(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2024 {
2025         (void) snprintf(buf, buflen, "%s,%s,%u,%u,%u", R[inst->rie_f.r1],
2026             R[inst->rie_f.r2], inst->rie_f.i3, inst->rie_f.i4,
2027             inst->rie_f.i5);
2028 }

2030 /* ARGSUSED */
2031 static void
2032 fmt_rre(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2033 {
2034         (void) snprintf(buf, buflen, "%s,%s", R[inst->rre.r1], R[inst->rre.r2]);
2035 }

2037 /* ARGSUSED */
2038 static void
2039 fmt_rrf_a(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_ab.r1], R[inst->rrf_ab.r2], R[inst->rrf_ab.r3]);
2043 }

2045 /* ARGSUSED */
2046 static void
2047 fmt_rrf_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2048 {
2049         (void) snprintf(buf, buflen, "%s,%s,%s",
2050             R[inst->rrf_ab.r1], R[inst->rrf_ab.r3], R[inst->rrf_ab.r2]);
2051 }

2053 /* ARGSUSED */
2054 static void
2055 fmt_rrf_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2056 {
2057         (void) snprintf(buf, buflen, "%s,%s,%s",
2058             R[inst->rrf_cde.r1], R[inst->rrf_cde.r2], M[inst->rrf_cde.m3]);
2059 }

2061 /* ARGSUSED */
2062 static void
2063 fmt_rrf_d(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2064 {
2065         (void) snprintf(buf, buflen, "%s,%s,%s",
2066             R[inst->rrf_cde.r1], R[inst->rrf_cde.r2], M[inst->rrf_cde.m4]);
2067 }

2069 /* ARGSUSED */
2070 static void
2071 fmt_rrf_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2072 {
2073         (void) snprintf(buf, buflen, "%s,%s,%s,%s",
2074             R[inst->rrf_cde.r1], M[inst->rrf_cde.m3],
2075             R[inst->rrf_cde.r2], M[inst->rrf_cde.m4]);
2076 }

2078 /* ARGSUSED */
2079 static void
2080 fmt_rrs(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2081 {
2082         (void) snprintf(buf, buflen, "%s,%s,%s,%u(%s)", R[inst->rrs.r1],
2083             R[inst->rrs.r2], M[inst->rrs.m3],
2084             val_4_8(inst->rrs.d4h, inst->rrs.d4l), B[inst->rrs.b4]);
2085 }

2087 /* ARGSUSED */
2088 static void
2089 fmt_rr(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2090 {
2091         /* a branch uses r1 as a mask */
2092         if (flags & F_HIDE_MASK)
2093                 (void) snprintf(buf, buflen, "%s", R[inst->rr.r2]);
2094         else if (flags & F_R1_IS_MASK)
2095                 (void) snprintf(buf, buflen, "%s,%s", M[inst->rr.r1],
2096                     R[inst->rr.r2]);
2097         else
2098                 (void) snprintf(buf, buflen, "%s,%s", R[inst->rr.r1],
2099                     R[inst->rr.r2]);
2100 }

2102 /* ARGSUSED */
2103 static void
2104 fmt_rrd(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2105 {
2106         (void) snprintf(buf, buflen, "%s,%s,%s", R[inst->rrd.r1],
2107             R[inst->rrd.r3], R[inst->rrd.r2]);
```

```
2108 }
2110 /* ARGSUSED */
2111 static void
2112 fmt_rx_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2113 {
2114         uint32_t d2 = val_4_8(inst->rx_a.d2h, inst->rx_b.d2l);
2116         (void) snprintf(buf, buflen, "%s,%u(%s,%s)", R[inst->rx_a.r1],
2117             d2, X[inst->rx_a.x2], B[inst->rx_a.b2]);
2118 }
2120 /* ARGSUSED */
2121 static void
2122 fmt_rx_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2123 {
2124         uint32_t d2 = val_4_8(inst->rx_b.d2h, inst->rx_b.d2l);
2126         if (flags & F_HIDE_MASK)
2127                 (void) snprintf(buf, buflen, "%u(%s,%s)",
2128                     d2, X[inst->rx_b.x2], B[inst->rx_b.b2]);
2129         else
2130                 (void) snprintf(buf, buflen, "%s,%u(%s,%s)", M[inst->rx_b.m1],
2131                     d2, X[inst->rx_b.x2], B[inst->rx_b.b2]);
2132 }
2134 /* ARGSUSED */
2135 static void
2136 fmt_rxe(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2137 {
2138         uint32_t d2 = val_4_8(inst->rxe.d2h, inst->rxe.d2l);
2140         (void) snprintf(buf, buflen, "%s,%u(%s,%s)",
2141             R[inst->rxe.r1], d2, X[inst->rxe.x2], B[inst->rxe.b2]);
2142 }
2144 /* ARGSUSED */
2145 static void
2146 fmt_rxf(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2147 {
2148         uint32_t d2 = val_4_8(inst->rxf.d2h, inst->rxf.d2l);
2150         (void) snprintf(buf, buflen, "%s,%s,%u(%s,%s)",
2151             R[inst->rxf.r1], R[inst->rxf.r3], d2, X[inst->rxf.x2],
2152             B[inst->rxf.b2]);
2153 }
2155 /* ARGSUSED */
2156 static void
2157 fmt_rxy_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2158 {
2159         uint32_t d2;
2161         d2 = val_8_4_8(inst->rxy_a.dh2, inst->rxy_a.dl2h, inst->rxy_a.dl2l);
2163         (void) snprintf(buf, buflen, "%s,%u(%s,%s)",
2164             R[inst->rxy_a.r1], d2, X[inst->rxy_a.x2], B[inst->rxy_a.b2]);
2165 }
2167 /* ARGSUSED */
2168 static void
2169 fmt_rxy_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2170 {
2171         uint32_t d2;
2173         d2 = val_8_4_8(inst->rxy_b.dh2, inst->rxy_b.dl2h, inst->rxy_b.dl2l);
```



```
2175         (void) snprintf(buf, buflen, "%s,%u(%s,%s)",
2176             M[inst->rxy_b.m1], d2, X[inst->rxy_b.x2], B[inst->rxy_b.b2]);
2177 }
2179 /* ARGSUSED */
2180 static void
2181 fmt_rs_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2182 {
2183         const char *r1, *r3;
2185         if (flags & F_CTL_REG) {
2186                 r1 = C[inst->rs_a.r1];
2187                 r3 = C[inst->rs_a.r3];
2188         } else {
2189                 r1 = R[inst->rs_a.r1];
2190                 r3 = R[inst->rs_a.r3];
2191         }
2193         (void) snprintf(buf, buflen, "%s,%s,%u(%s)", r1, r3,
2194             val_4_8(inst->rs_a.d2h, inst->rs_a.d2l), B[inst->rs_a.b2]);
2195 }
2197 /* ARGSUSED */
2198 static void
2199 fmt_rs_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2200 {
2201         (void) snprintf(buf, buflen, "%s,%s,%u(%s)", R[inst->rs_b.r1],
2202             M[inst->rs_b.m3], val_4_8(inst->rs_b.d2h, inst->rs_b.d2l),
2203             B[inst->rs_b.b2]);
2204 }
2206 /* ARGSUSED */
2207 static void
2208 fmt_rsl_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2209 {
2210         (void) snprintf(buf, buflen, "%u(%u,%s)",
2211             val_4_8(inst->rsl_a.d1h, inst->rsl_a.d1l), inst->rsl_a.l1,
2212             B[inst->rsl_a.b1]);
2213 }
2215 /* ARGSUSED */
2216 static void
2217 fmt_rsl_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2218 {
2219         (void) snprintf(buf, buflen, "%s,%u(%u,%s),%s",
2220             R[inst->rsl_b.r1],
2221             val_4_8(inst->rsl_b.d2h, inst->rsl_b.d2l), inst->rsl_b.l2,
2222             B[inst->rsl_b.b2], M[inst->rsl_b.m3]);
2223 }
2225 /* ARGSUSED */
2226 static void
2227 fmt_rsy_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2228 {
2229         const char *r1, *r3;
2230         uint32_t d2;
2232         d2 = val_8_4_8(inst->rsy_a.dh2, inst->rsy_a.dl2h, inst->rsy_a.dl2l);
2234         if (flags & F_CTL_REG) {
2235                 r1 = C[inst->rsy_a.r1];
2236                 r3 = C[inst->rsy_a.r3];
2237         } else {
2238                 r1 = R[inst->rsy_a.r1];
2239                 r3 = R[inst->rsy_a.r3];
```

```
2240         }

2242         (void) snprintf(buf, buflen, "%s,%s,%u(%s)", r1, r3, d2,
2243             B[inst->rsy_a.b2]);
2244 }

2246 /* ARGSUSED */
2247 static void
2248 fmt_rsy_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2249 {
2250         uint32_t d2;

2252         d2 = val_8_4_8(inst->rsy_b.dh2, inst->rsy_b.dl2h, inst->rsy_b.dl2l);

2254         (void) snprintf(buf, buflen, "%s,%s,%u(%s)",
2255             R[inst->rsy_b.r1], M[inst->rsy_b.m3],
2256             d2, B[inst->rsy_b.b2]);
2257 }

2259 /* ARGSUSED */
2260 static void
2261 fmt_rsi(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2262 {
2263         uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->rsi.ri2);

2265         (void) snprintf(buf, buflen, "%s,%s,%#x", R[inst->rsi.r1],
2266             R[inst->rsi.r3], ri2);
2267 }

2269 /* ARGSUSED */
2270 static void
2271 fmt_si(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2272 {
2273         uint32_t d1 = val_4_8(inst->si.d1h, inst->si.d1l);

2275         (void) snprintf(buf, buflen, "%u(%s),%u", d1, B[inst->si.b1],
2276             inst->si.i2);
2277 }

2279 /* ARGSUSED */
2280 static void
2281 fmt_sil(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2282 {
2283         (void) snprintf(buf, buflen, "%u(%s),%u",
2284             val_4_8(inst->sil.d1h, inst->sil.d1l), B[inst->sil.b1],
2285             BE_16(inst->sil.i2));
2286 }

2288 /* ARGSUSED */
2289 static void
2290 fmt_siy(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2291 {
2292         (void) snprintf(buf, buflen, "%u(%s),%u",
2293             val_8_4_8(inst->siy.dh1, inst->siy.dl1h, inst->siy.dl1l),
2294             B[inst->siy.b1], inst->siy.i2);
2295 }

2297 /* ARGSUSED */
2298 static void
2299 fmt_smi(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2300 {
2301         uint64_t ri2 = addr + 2 * (int16_t)BE_16(inst->smi.ri2);

2303         (void) snprintf(buf, buflen, "%s,%#x,%u(%s)", M[inst->smi.m1], ri2,
2304             val_4_8(inst->smi.d3h, inst->smi.d3l), B[inst->smi.b3]);
2305 }
```



```
2307 /* ARGSUSED */
2308 static void
2309 fmt_s(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2310 {
2311         uint32_t d = val_4_8(inst->s.d2h, inst->s.d2l);

2313         (void) snprintf(buf, buflen, "%u(%s)", d, B[inst->s.b2]);
2314 }

2316 /* ARGSUSED */
2317 static void
2318 fmt_ss_a(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2319 {
2320         uint32_t d1, d2;

2322         d1 = val_4_8(inst->ss_a.d1h, inst->ss_a.d1l);
2323         d2 = val_4_8(inst->ss_a.d2h, inst->ss_a.d2l);

2325         (void) snprintf(buf, buflen, "%u(%u,%s),%u(%s)",
2326             d1, inst->ss_a.l + 1, B[inst->ss_a.b1],
2327             d2, B[inst->ss_a.b2]);
2328 }

2330 /* ARGSUSED */
2331 static void
2332 fmt_ss_b(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2333 {
2334         uint32_t d1, d2;

2336         d1 = val_4_8(inst->ss_b.d1h, inst->ss_b.d1l);
2337         d2 = val_4_8(inst->ss_b.d2h, inst->ss_b.d2l);

2339         (void) snprintf(buf, buflen, "%u(%u,%s),%u(%u,%s)",
2340             d1, inst->ss_b.l1 + 1, B[inst->ss_b.b1],
2341             d2, inst->ss_b.l2 + 1, B[inst->ss_b.b2]);
2342 }

2344 /* ARGSUSED */
2345 static void
2346 fmt_ss_c(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2347 {
2348         uint32_t d1, d2;

2350         d1 = val_4_8(inst->ss_c.d1h, inst->ss_c.d1l);
2351         d2 = val_4_8(inst->ss_c.d2h, inst->ss_c.d2l);

2353         (void) snprintf(buf, buflen, "%u(%u,%s),%u(%s),%u",
2354             d1, inst->ss_c.l1, B[inst->ss_c.b1],
2355             d2, B[inst->ss_c.b2], inst->ss_c.i3);
2356 }

2358 /* ARGSUSED */
2359 static void
2360 fmt_ss_d(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2361 {
2362         uint32_t d1, d2;

2364         d1 = val_4_8(inst->ss_d.d1h, inst->ss_d.d1l);
2365         d2 = val_4_8(inst->ss_d.d2h, inst->ss_d.d2l);

2367         (void) snprintf(buf, buflen, "%u(%s,%s),%u(%s),%s",
2368             d1, R[inst->ss_d.r1], B[inst->ss_d.b1],
2369             d2, B[inst->ss_d.b2], R[inst->ss_d.r3]);
2370 }
```

```
2372 /* ARGSUSED */
2373 static void
2374 fmt_ss_e(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2375 {
2376         uint32_t d2, d4;

2378         d2 = val_4_8(inst->ss_e.d2h, inst->ss_e.d2l);
2379         d4 = val_4_8(inst->ss_e.d4h, inst->ss_e.d4l);

2381         (void) snprintf(buf, buflen, "%s,%u(%s),%s,%u(%s)",
2382             R[inst->ss_e.r1], d2, B[inst->ss_e.b2],
2383             R[inst->ss_e.r3], d4, B[inst->ss_e.b4]);
2384 }

2386 /* ARGSUSED */
2387 static void
2388 fmt_ss_f(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2389 {
2390         uint32_t d1, d2;

2392         d1 = val_4_8(inst->ss_f.d1h, inst->ss_f.d1l);
2393         d2 = val_4_8(inst->ss_f.d2h, inst->ss_f.d2l);

2395         (void) snprintf(buf, buflen, "%u(%s),%u(%u,%s)",
2396             d1, B[inst->ss_f.b1], d2, inst->ss_f.l2,
2397             B[inst->ss_f.b2]);
2398 }

2400 /* ARGSUSED */
2401 static void
2402 fmt_sse(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2403 {
2404         uint32_t d1 = val_4_8(inst->sse.d1h, inst->sse.d1l);
2405         uint32_t d2 = val_4_8(inst->sse.d2h, inst->sse.d2l);

2407         (void) snprintf(buf, buflen, "%u(%s),%u(%s)",
2408             d1, B[inst->sse.b1], d2, B[inst->sse.b2]);
2409 }

2411 /* ARGSUSED */
2412 static void
2413 fmt_ssf(uint64_t addr, union inst *inst, char *buf, size_t buflen, int flags)
2414 {
2415         uint32_t d1 = val_4_8(inst->ssf.d1h, inst->ssf.d1l);
2416         uint32_t d2 = val_4_8(inst->ssf.d2h, inst->ssf.d2l);

2418         (void) snprintf(buf, buflen, "%u(%s),%u(%s),%s",
2419             d1, B[inst->ssf.b1],
2420             d2, B[inst->ssf.b2], R[inst->ssf.r3]);
2421 }

2423 static void (*fmt_fxns[IF_NFMTS])(uint64_t, union inst *, char *, size_t,
2424     int) = {
2425         [IF_ZERO]       = fmt_zero,
2426         [IF_DIAG]       = fmt_diag,
2427         [IF_E]          = fmt_e,
2428         [IF_I]          = fmt_i,
2429         [IF_IE]         = fmt_ie,
2430         [IF_MII]        = fmt_mii,
2431         [IF_RIa]        = fmt_ri_a,
2432         [IF_RIb]        = fmt_ri_b,
2433         [IF_RIc]        = fmt_ri_c,
2434         [IF_RIEa]       = fmt_rie_a,
2435         [IF_RIEb]       = fmt_rie_b,
2436         [IF_RIEc]       = fmt_rie_c,
2437         [IF_RIEd]       = fmt_rie_d,
```



```
2438         [IF_RIEe]       = fmt_rie_e,
2439         [IF_RIEf]       = fmt_rie_f,
2440         [IF_RILa]       = fmt_ril_a,
2441         [IF_RILb]       = fmt_ril_b,
2442         [IF_RILc]       = fmt_ril_c,
2443         [IF_RIS]        = fmt_ris,
2444         [IF_RR]         = fmt_rr,
2445         [IF_RRD]        = fmt_rrd,
2446         [IF_RRE]        = fmt_rre,
2447         [IF_RRFa]       = fmt_rrf_a,
2448         [IF_RRFb]       = fmt_rrf_b,
2449         [IF_RRFc]       = fmt_rrf_c,
2450         [IF_RRFd]       = fmt_rrf_d,
2451         [IF_RRFe]       = fmt_rrf_e,
2452         [IF_RRS]        = fmt_rrs,
2453         [IF_RSa]        = fmt_rs_a,
2454         [IF_RSb]        = fmt_rs_b,
2455         [IF_RSI]        = fmt_rsi,
2456         [IF_RSLa]       = fmt_rsl_a,
2457         [IF_RSLb]       = fmt_rsl_b,
2458         [IF_RSYa]       = fmt_rsy_a,
2459         [IF_RSYb]       = fmt_rsy_b,
2460         [IF_RXa]        = fmt_rx_a,
2461         [IF_RXb]        = fmt_rx_b,
2462         [IF_RXE]        = fmt_rxe,
2463         [IF_RXF]        = fmt_rxf,
2464         [IF_RXYa]       = fmt_rxy_a,
2465         [IF_RXYb]       = fmt_rxy_b,
2466         [IF_S]          = fmt_s,
2467         [IF_SI]         = fmt_si,
2468         [IF_SIL]        = fmt_sil,
2469         [IF_SIY]        = fmt_siy,
2470         [IF_SMI]        = fmt_smi,
2471         [IF_SSa]        = fmt_ss_a,
2472         [IF_SSb]        = fmt_ss_b,
2473         [IF_SSc]        = fmt_ss_c,
2474         [IF_SSd]        = fmt_ss_d,
2475         [IF_SSe]        = fmt_ss_e,
2476         [IF_SSf]        = fmt_ss_f,
2477         [IF_SSE]        = fmt_sse,
2478         [IF_SSF]        = fmt_ssf,
2479 };

2481 static int
2482 dis_s390(uint64_t addr, union inst *inst, char *buf, size_t buflen, int mach)
2483 {
2484         const struct inst_table *tbl = &tbl_xx[inst->raw[0]];
2485         int tmp;

2487         while (tbl->it_fmt == IF_TBL || tbl->it_fmt == IF_MULTI) {
2488                 if (tbl->it_fmt == IF_TBL) {
2489                         int idx;

2491                         idx   = inst->raw[tbl->it_u.it_table.it_off];
2492                         idx >>= tbl->it_u.it_table.it_shift;
2493                         idx  &= tbl->it_u.it_table.it_mask;

2495                         tbl = &tbl->it_u.it_table.it_ptr[idx];
2496                 } else if (tbl->it_fmt == IF_MULTI) {
2497                         tbl = &tbl->it_u.it_multi.it_ptr[mach];
2498                 }
2499         }

2501         if (tbl->it_fmt == IF_INVAL)
2502                 goto inval;
```

```
2504         if ((tbl->it_u.it_inst.it_flags & mach) == 0)
2505                 goto inval;

2507         tmp = snprintf(buf, buflen, "%-7s ", tbl->it_u.it_inst.it_name);

2509         fmt_fxns[tbl->it_fmt](addr, inst, buf + tmp, buflen - tmp,
2510             tbl->it_u.it_inst.it_flags);

2512         return (0);

2514 inval:
2515         (void) snprintf(buf, buflen, "??");

2517         /*
2518          * Even if we don't know how to disassemble the instruction, we know
2519          * how long it is, so we "succeed" even when we fail.
2520          */
2521         return (0);
2522 }

2524 static int
2525 dis_s390_supports_flags(int flags)
2526 {
2527         int archflags = flags & DIS_ARCH_MASK;

2529         if (archflags == DIS_S370 || archflags == DIS_S390_31 ||
2530             archflags == DIS_S390_64)
2531                 return (1);

2533         return (0);
2534 }

2536 static int
2537 dis_s390_disassemble(dis_handle_t *dhp, uint64_t addr, char *buf,
2538     size_t buflen)
2539 {
2540         union inst inst;
2541         int mach;
2542         int len;

2544         if (dhp->dh_read(dhp->dh_data, addr, &inst.raw[0], 2) != 2)
2545                 return (-1);

2547         len = ILC2LEN(inst.raw[0] >> 6) - 2;

2549         if (len > 0 &&
2550             dhp->dh_read(dhp->dh_data, addr + 2, &inst.raw[2], len) != len)
2551                         return (-1);

2553         switch (dhp->dh_flags & (DIS_S370 | DIS_S390_31 | DIS_S390_64)) {
2554                 case DIS_S370:
2555                         mach = F_370;
2556                         break;
2557                 case DIS_S390_31:
2558                         mach = F_390;
2559                         break;
2560                 case DIS_S390_64:
2561                         mach = F_Z;
2562                         break;
2563         }

2565         return (dis_s390(addr, &inst, buf, buflen, mach));
2566 }

2568 /* ARGSUSED */
2569 static int
```



```
2570 dis_s390_min_instrlen(dis_handle_t *dhp)
2571 {
2572         return (2);
2573 }

2575 /* ARGSUSED */
2576 static int
2577 dis_s390_max_instrlen(dis_handle_t *dhp)
2578 {
2579         return (6);
2580 }

2582 dis_arch_t dis_arch_s390 = {
2583         .da_supports_flags      = dis_s390_supports_flags,
2584         .da_disassemble         = dis_s390_disassemble,
2585         .da_min_instrlen        = dis_s390_min_instrlen,
2586         .da_max_instrlen        = dis_s390_max_instrlen,
2587 };
2588 #endif /* ! codereview */
```

```
**********************************************************
    7051 Wed Oct 14 16:45:11 2015
new/usr/src/lib/libdisasm/common/libdisasm.c
6066 dis: support for System/370, System/390, and z/Architecture ELF bins
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

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

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

#include "libdisasm_impl.h"

static int _dis_errno;

/*
 * If we're building the standalone library, then we only want to
 * include support for disassembly of the native architecture.
 * The regular shared library should include support for all
 * architectures.
 */
#if !defined(DIS_STANDALONE) || defined(__i386) || defined(__amd64)
extern dis_arch_t dis_arch_i386;
#endif
#if !defined(DIS_STANDALONE) || defined(__sparc)
extern dis_arch_t dis_arch_sparc;
#endif
#if !defined(DIS_STANDALONE) || defined(__s390) || defined(__s390x)
extern dis_arch_t dis_arch_s390;
#endif
#endif /* ! codereview */

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



```
        &dis_arch_i386,
#endif
#if !defined(DIS_STANDALONE) || defined(__sparc)
        &dis_arch_sparc,
#endif
#if !defined(DIS_STANDALONE) || defined(__s390) || defined(__s390x)
        &dis_arch_s390,
#endif /* ! codereview */
#endif
        NULL
};

/*
 * For the standalone library, we need to link against mdb's malloc/free.
 * Otherwise, use the standard malloc/free.
 */
#ifdef DIS_STANDALONE
void *
dis_zalloc(size_t bytes)
{
        return (mdb_zalloc(bytes, UM_SLEEP));
}

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

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

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

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

const char *
dis_strerror(int error)
{
        switch (error) {
        case E_DIS_NOMEM:
                return ("out of memory");
        case E_DIS_INVALFLAG:
                return ("invalid flags for this architecture");
        case E_DIS_UNSUPARCH:
                return ("unsupported machine architecture");
```

```
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));
270
271         return (dhp->dh_arch->da_previnstr(dhp, pc, n));
272 }
273
274 int
275 dis_min_instrlen(dis_handle_t *dhp)
276 {
277         return (dhp->dh_arch->da_min_instrlen(dhp));
278 }
279
280 int
281 dis_max_instrlen(dis_handle_t *dhp)
282 {
283         return (dhp->dh_arch->da_max_instrlen(dhp));
284 }
285
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);
291
292         return (dhp->dh_addr - pc);
293 }
294
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));
300
301         return (dhp->dh_arch->da_instrlen(dhp, pc));
302 }
303
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 }
314
315 int
316 dis_snprintf(char *restrict s, size_t n, const char *restrict format, ...)
317 {
318         va_list args;
319
320         va_start(args, format);
321         n = dis_vsnprintf(s, n, format, args);
322         va_end(args);
323
324         return (n);
325 }
```

```
**********************************************************
    2897 Wed Oct 14 16:45:11 2015
new/usr/src/lib/libdisasm/common/libdisasm.h
6066 dis: support for System/370, System/390, and z/Architecture ELF bins
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

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

#ifndef _LIBDISASM_H
#define _LIBDISASM_H

#include <sys/types.h>

#ifdef  __cplusplus
extern "C" {
#endif

typedef struct dis_handle dis_handle_t;

#define DIS_DEFAULT             0x0

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

/* x86 diassembler flags */
#define DIS_X86_SIZE16          0x100
#define DIS_X86_SIZE32          0x010
#define DIS_X86_SIZE64          0x020

/* s390 disassembler flags */
#define DIS_S370                0x200
#define DIS_S390_31             0x400
#define DIS_S390_64             0x800

#endif /* ! codereview */
/* generic disassembler flags */
#define DIS_OCTAL               0x040
```



```
#define DIS_NOIMMSYM            0x080

#define DIS_ARCH_MASK           (DIS_SPARC_V8 | \
                DIS_SPARC_V9 | DIS_SPARC_V9_SGI | DIS_SPARC_V9_OPL | \
                DIS_X86_SIZE16 | DIS_X86_SIZE32 | DIS_X86_SIZE64 | \
                DIS_S370 | DIS_S390_31 | DIS_S390_64)
                DIS_X86_SIZE16 | DIS_X86_SIZE32 | DIS_X86_SIZE64)

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

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

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

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

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

#ifdef  __cplusplus
}
```

*unchanged_portion_omitted_*