file name: Makefile

```
  1  ######################################################################
  2  ## SimCell: Cell/B.E. Processor Simulator        Arch Lab. TOKYO TECH ##
  3  ######################################################################
  4  CC      = g++
  5  OFLAG   = -Wall -O3
  6  LFLAG   = -static
  7  DEBUG   = -DDEBUG -g
  8
  9  TARGET  = SimCell
 10  HEADER  = define.h
 11  SOURCE  = main.cc memory.cc loaderbe.cc spe.cc spuinst.cc eib.cc etc.cc
 12  OBJECT  = $(SOURCE:.cc=.o)
 13  ######################################################################
 14  all:
 15   $(MAKE) $(TARGET)
 16  ######################################################################
 17  $(TARGET): $(OBJECT) $(HEADER) Makefile
 18   $(CC) $(LFLAG) $(OFLAG) -o $@ $(OBJECT)
 19  ######################################################################
 20  # $(TARGET): $(SOURCE) $(HEADER) Makefile
 21  #         $(CC) $(LFLAG) $(OFLAG) -o $@ $(SOURCE)
 22
 23  ######################################################################
 24  debug:
 25   $(CC) $(DEBUG) $(SOURCE) -o $(TARGET)
 26  ######################################################################
 27  .SUFFIXES :
 28  .SUFFIXES : .o .cc
 29
 30  .cc.o:
 31   $(CC) $(OFLAG) -c $<
 32
 33  $(OBJECT) : $(HEADER) Makefile
 34  ######################################################################
 35  wc:
 36   wc -l $(HEADER) $(SOURCE)
 37
 38  indent:
 39   indent -kr -ts4 -nut main.cc
 40
 41  text:
 42   cats Makefile > code.txt
 43   echo -e "\nFile Organization by [wc *.h *.cc]" >> code.txt
 44   echo -e "  lines words bytes" >> code.txt
 45   wc $(HEADER) $(SOURCE) >> code.txt
 46   echo -e "\f" >> code.txt
 47   cats -f $(HEADER) $(SOURCE) >> code.txt
 48   a2ps --medium=a4 -f 6.5 code.txt -o  code.ps
 49   ps2pdf13 -sPAPERSIZE=a4 code.ps
 50   rm -f code.txt code.ps
 51
 52  cflow:
 53   cflow *.cc
 54  clean:
 55   rm -f *.o *.*~ *.exe $(TARGET) code.cc code.ps
 56  ######################################################################
 57  run:
 58   ./$(TARGET) -n1 tst/hello
 59  run1:
 60   ./$(TARGET) -n6 -K300 -M30 -m tst/mem_catchball.txt tst/catchball
 61  ######################################################################

File Organization by [wc *.h *.cc]
  lines words bytes
   811   1871  19038 define.h
   298    821   8590 main.cc
   171    585   5277 memory.cc
   216    671   6167 loaderbe.cc
  1944   6310  60170 spe.cc
  1437   5752  61137 spuinst.cc
   722   1767  20987 eib.cc
   305    964   9200 etc.cc
  5904  18741 190566 total
```


file name: define.h

```
  1  /*************************************************************************/
  2  /* SimCell: Cell/B.E. Processor Simulator        Arch Lab. TOKYO TECH */
  3  /*************************************************************************/
  4  #include <stdio.h>
  5  #include <stdlib.h>
  6  #include <string.h>
  7  #include <math.h>
  8  #include <unistd.h>
  9  #include <fcntl.h>
 10  #include <elf.h>
 11  #include <sys/stat.h>
 12  #include <sys/time.h>
 13
 14  /** Type Definition ******************************************************/
 15  /*************************************************************************/
 16  typedef unsigned int       uint;
 17  typedef signed long long   llint;
 18  typedef unsigned long long ullint;
 19
 20  typedef float              flot_032t;
 21  typedef double             flot_064t;
 22  typedef signed char        sint_008t;
 23  typedef signed short       sint_016t;
 24  typedef signed int         sint_032t;
 25  typedef signed long long   sint_064t;
 26  typedef unsigned char      uint_008t;
 27  typedef unsigned short     uint_016t;
 28  typedef unsigned int       uint_032t;
 29  typedef unsigned long long uint_064t;
 30  typedef struct {
 31      uint_064t high;
 32      uint_064t low;
 33  } uint_128t;
 34
 35  /** Constant Definition **************************************************/
 36  /*************************************************************************/
 37  #define L_NAME "SimCell: Cell/B.E. Simulator"
 38  #define L_VER  "Version 0.8.0 2008-03-31"
 39
 40  #define PROCESSOR_FREQ 3200.0f      // processor frequency in MHz
 41  #define AVERAGE_IPC 1.0f            // processor average IPC
 42
 43  #define MAX_SPES 512
 44  #define MAX_CYCLE_DEF 1000000000    // limit of simulation cycle
 45  #define VERBOSE_INT_DEF  1000000    // verbose interval cycle
 46  #define LOGMODE_INT_DEF  1000000    // log-mode interval cycle
 47  #define LOGMODE_FILE "log.dat"      // memory access log written to
 48
 49  // EIB network latency
 50  #define MEM_RDDELAY_DEF 500
 51  #define MEM_WRDELAY_DEF 300
 52  #define MEM_WIDTH_DEF 16
 53  #define C2C_DELAY_DEF 300
 54  #define C2C_WIDTH_DEF 8
 55
 56  /*************************************************************************/
 57  enum {
 58      NREG = 128,                     // Number of Resisters
 59      REG_LENGTH = 16,                // Length of a Register (Byte)
 60      LS_SIZE = (256 * 1024),         // Local Store Size (Byte)
 61      CH_NUM = 31,                    // Number of Channels
 62      MFC_MAX_ENTRY = 16,
 63      EIB_MAX_ENTRY = 4,
 64      EIB_WAIT_ENTRY = MAX_SPES,
 65      MAX_DEBUG_MODE = 3,
 66      SPE_RUN = 0,
 67      SPE_STOP = 1,
 68      SPE_STALL = 2,
 69      SPE_HALT = 3,
 70      MEM_READ = 0,
 71      MEM_WRITE = 1,
 72      MEM_DMA = 2,
 73      MEM_INST_READ = 3,
 74      MEM_ACCESS_KIND = 4,
 75  };
```

```
 76
 77  /** Class and Function Definition ***********************************/
 78  /*******************************************************************/
 79  class Spe;
 80  class MainMemory;
 81  class Mmu;
 82  class Eib;
 83  class SpuInstinfo;
 84
 85  /*******************************************************************/
 86  class Chip {
 87   private:
 88      bool checkendian();
 89
 90   public:
 91      int spe_num;
 92      ullint cycle;
 93      ullint max_cycle;
 94      ullint verb_cycle;
 95      ullint log_cycle;
 96      int debug_mode;
 97      bool log_mode;
 98      bool force_mode;
 99      bool big_endian;
100      int mem_rddelay;
101      int mem_wrdelay;
102      int mem_width;
103      int c2c_delay;
104      int c2c_width;
105      int running_core;
106      int ***logdata;
107      int logpage;
108      FILE *logfile;
109      char *mem_imagefile;
110
111      Spe *spe[MAX_SPES];
112      MainMemory *mem;
113      Mmu *mmu;
114      Eib *eib;
115      SpuInstinfo *iinfo;
116
117      Chip();
118      void setlogmode();
119      void initlogmode();
120      void setlogdata(int, int, int);
121      void writelogdata();
122      void finilogmode();
123  };
124
125  /** SPU Global Perpose Register *************************************/
126  #define NSLOT(data_type) (int)(REG_LENGTH / sizeof(data_type))
127
128  typedef union {
129      uint_008t i008t[NSLOT(uint_008t)];
130      uint_016t i016t[NSLOT(uint_016t)];
131      uint_032t i032t[NSLOT(uint_032t)];
132      uint_064t i064t[NSLOT(uint_064t)];
133      flot_032t f032t[NSLOT(uint_032t)];
134      flot_064t f064t[NSLOT(uint_064t)];
135  } regdata_t;
136
137  /*******************************************************************/
138  class Register {
139    private:
140      Chip *chip;
141      regdata_t reg;
142
143    public:
144      Register(Chip *);
145
146      void print();
147
148      uint_008t get008t(int);
149      uint_016t get016t(int);
150      uint_032t get032t(int);
151      uint_064t get064t(int);
152      uint_128t get128t();
```



```
153      flot_032t getf32t(int);
154      flot_064t getf64t(int);
155
156      void put008t(int, uint_008t);
157      void put016t(int, uint_016t);
158      void put032t(int, uint_032t);
159      void put064t(int, uint_064t);
160      void put128t(uint_128t);
161      void putf32t(int, flot_032t);
162      void putf64t(int, flot_064t);
163  };
164
165  /** FP Status and Control Register **********************************/
166  class Fpscr {
167   private:
168      Chip *chip;
169      regdata_t reg;
170      regdata_t enable;
171
172   public:
173      Fpscr(Chip *);
174
175      uint_128t read();
176      void write(uint_128t);
177      void bitset(int);
178      void twobitwrite(int, int);
179  };
180
181  /** Channel and Channel Interface ***********************************/
182  class Channel {
183  private:
184      uint_032t *message, ch_count;
185      int head, tail;
186      const uint_032t size;
187
188  public:
189      const int blocking_mode, channel_type;
190
191      Channel(int, int, int);
192      ~Channel();
193
194      bool empty();
195      bool full();
196      uint_032t capacity();
197      uint_032t occupancy();
198      void inccnt();
199      void deccnt();
200      uint_032t read();
201      void write(uint_032t);
202  };
203
204  /*******************************************************************/
205  class ChannelInterface {
206   public:
207      Channel *ch[CH_NUM];
208
209      ChannelInterface();
210      ~ChannelInterface();
211
212      bool spureadable(int);
213      bool spuwritable(int);
214      int spureadch(int, uint_032t*);
215      int spuwritech(int, uint_032t);
216      uint_032t spureadchcnt(int);
217      void mfcreadch(int, uint_032t*);
218      void mfcwritech(int, uint_032t);
219  };
220
221  /** Local Store *****************************************************/
222  class LocalStore {
223   private:
224      uint_032t *mem;
225
226   public:
227      LocalStore(ullint);
228      ~LocalStore();
229
```

```
230      void read1b(uint_032t, uint_008t*);
231      void read4b(uint_032t, uint_032t*);
232      void readnb(uint_032t, int, uint_032t[]);
233      void write1b(uint_032t, uint_008t);
234      void write4b(uint_032t, uint_032t);
235      void writenb(uint_032t, int, uint_032t[]);
236      void print();
237  };
238
239  /** Architecture State ************************************************/
240  class SpeArchstate {
241   private:
242      Chip *chip;
243   public:
244      uint_032t pc;
245      uint lslr;
246      int state;
247      Register **reg;
248      Fpscr *fpscr;
249      int stop_reason;
250
251      SpeArchstate(Chip*);
252      ~SpeArchstate();
253  };
254
255  /** Simulator State ***************************************************/
256  class SpeSimstate {
257   public:
258      ullint instcnt;
259      int stall_reason;
260
261      SpeSimstate();
262  };
263
264  /** SPU Instructions **************************************************/
265  enum {
266      IT_NONE = 0x0,
267      IT_W_RT = 0x1,
268      IT_R_RA = 0x2,
269      IT_R_RB = 0x4,
270      IT_R_RC = 0x8,
271      IT_R_RT = 0x10,
272      IT_R_I18 = 0x20,
273      IT_R_I16 = 0x40,
274      IT_R_I10 = 0x80,
275      IT_R_I8 = 0x100,
276      IT_R_I7 = 0x200,
277      IT_R_CA = 0x400,
278      IT_R_RO1 = 0x800,
279      IT_R_RO2 = 0x1000,
280      NOADDR = 0xffffffff,
281      IINFO_OP_NUM = 0x800,
282      IINFO_ID_NUM = 0x1000,
283      PIPE_EVEN = 0,
284      PIPE_ODD = 1,
285  };
286
287  /*********************************************************************/
288  typedef struct {
289      uint type;
290      uint pipe;
291      uint latency;
292      uint stall;
293      char *name;
294  } spuinstattr_t;
295
296  /*********************************************************************/
297  class SpuInstinfo {
298   private:
299      int op_to_id[IINFO_OP_NUM];
300      spuinstattr_t id_to_attr[IINFO_ID_NUM];
301
302      int predecode(int);
303      void setattr();
304   public:
305      SpuInstinfo(Chip *);
306      int getinstid(int);
```



```
307      spuinstattr_t getinstattr(int);
308      char *getopname(int);
309  };
310
311  /*********************************************************************/
312  class SpuInst {
313   private:
314      Chip *chip;
315      Spe *spe;
316      SpeArchstate *as;
317      SpuInstinfo *iinfo;
318
319   public:
320      uint_032t ir, pc, nextpc, targ;
321      uint op, imm, ca, ro;
322      uint type, pipe, latency, stall;
323      uint depend;
324      uint ra, rb, rc, rt;
325      Register *rra, *rrb, *rrc, *rrt;
326      int nextstate;
327
328      SpuInst(Chip*, Spe*);
329      ~SpuInst();
330      char *getopname();
331      void exprinthead();
332      void exprinttail();
333      int isload();
334      int isstore();
335      int isbranch();
336      int ishintbranch();
337      void decode();
338      void clear();
339  };
340
341  /** Synergistic Processor Unit ****************************************/
342  class Spu {
343    private:
344      Chip *chip;
345      Spe *spe;
346      ChannelInterface *chi;
347      SpeArchstate *as;
348      SpeSimstate *ss;
349
350      void load(uint_032t, Register *);
351      void store(uint_032t, Register *);
352
353      int checkstate();
354      void fetch(SpuInst*);
355      void decode(SpuInst*);
356      void executeodd(SpuInst*);
357      void executeeven(SpuInst*);
358      void execute(SpuInst*);
359      void lsaccess(SpuInst*);
360      void writeback(SpuInst*);
361      void setnextpc(SpuInst*);
362
363    public:
364      SpuInst *inst;
365
366      Spu(Chip*, Spe*);
367      ~Spu();
368      void printregall();
369      void message();
370      void start();
371      void step();
372
373  };
374
375  /** Memory Flow Controller ********************************************/
376  typedef struct{
377      uint_032t logical;
378      uint_032t physical;
379      int unitid;
380  } address_t;
381
382  /*********************************************************************/
383  typedef struct {
```

```
384      address_t lsa;
385      address_t ea;
386      uint_032t size;
387      uint_032t tagid;
388      uint_032t classid;
389      uint_032t cmd;
390  } dmacmd_t;
391
392  /*****************************************************************/
393  typedef struct {
394      int delay;
395      int state;
396  } dmacmdtag_t;
397
398  /*****************************************************************/
399  class Mfc {
400   private:
401      Chip *chip;
402      Mmu *mmu;
403      Spe *spe;
404      ChannelInterface *chi;
405
406      dmacmd_t mfcspucmdq[MFC_MAX_ENTRY];
407      int head, tail, entry;
408      dmacmdtag_t cmdtag[MFC_MAX_ENTRY];
409
410      uint_032t tagmask;
411      uint_032t tagupdate;
412      uint_032t tagstat;
413      uint_032t decrementer;
414
415      dmacmd_t getcmdfromchi();
416      int enq_spucmd(dmacmd_t, dmacmdtag_t);
417      int deq_spucmd();
418      bool ismemread(dmacmd_t);
419      bool ismemwrite(dmacmd_t);
420      int getdelay(dmacmd_t);
421      void makedmacmd();
422      void checkchannel();
423      void checkdma();
424      uint_032t maketagstat();
425      void updatetagstate();
426      void decrementerstep();
427
428  public:
429      Mfc(Chip*, Spe*);
430
431      void step();
432  };
433
434  /** Synergistic Processor Element ************************************/
435  class Spe {
436   private:
437      Chip *chip;
438
439   public:
440      Spu *spu;
441      Mfc *mfc;
442      LocalStore *ls;
443      ChannelInterface *chi;
444      SpeArchstate *as;
445      SpeSimstate *ss;
446
447      int unitid;
448
449      Spe(Chip *, int);
450      ~Spe();
451
452      void message(ullint);
453  };
454
455  /** Main Memory ******************************************************/
456  enum {
457      MAIN_MEM_SIZE = 0x100000000ull, // Main Memory size
458      BLOCK_SIZE = (4 * 1024),        // Page size
459      BLOCK_TABLE_SIZE = (MAIN_MEM_SIZE / BLOCK_SIZE),
460  };
```



```
461
462  /*****************************************************************/
463  class MainMemory {
464   private:
465      uint_032t *block_table[BLOCK_TABLE_SIZE];
466
467      uint_032t *allocblock(uint_032t);
468
469   public:
470      MainMemory(Chip *);
471      ~MainMemory();
472
473      void read1b(uint_032t, uint_008t*);
474      void read2b(uint_032t, uint_016t*);
475      void read4b(uint_032t, uint_032t*);
476      void read8b(uint_032t, uint_064t*);
477      void readnb(uint_032t, int, uint_032t[]);
478      void write1b(uint_032t, uint_008t);
479      void write2b(uint_032t, uint_016t);
480      void write4b(uint_032t, uint_032t);
481      void write8b(uint_032t, uint_064t);
482      void writenb(uint_032t, int, uint_032t[]);
483      void print();
484  };
485
486  /** Memory Management Unit *******************************************/
487  typedef struct{
488      uint_032t addr;
489      uint_032t id;
490  }mapdata_t;
491
492  /*****************************************************************/
493  class Mmu {
494   private:
495      mapdata_t mmutable[BLOCK_TABLE_SIZE];
496
497   public:
498      Mmu(Chip *);
499
500      void transaddr(address_t*);
501      int mapping(uint_032t, int, uint_032t);
502      void mapls(uint_032t, int);
503  };
504
505  /** Element Interconnect Bus *****************************************/
506  typedef struct{
507      dmacmd_t *cmd;
508      dmacmdtag_t *tag;
509  } eibcmd_t;
510
511  /*****************************************************************/
512  class Eib {
513  private:
514      Chip *chip;
515      Spe *spe[MAX_SPES];
516      MainMemory *mem;
517      uint_032t *transdata_ptr;
518
519      eibcmd_t activecmd[EIB_MAX_ENTRY];
520      int valid[EIB_MAX_ENTRY];
521      int entry_count;
522      eibcmd_t waitcmdq[EIB_WAIT_ENTRY];
523      int head, tail;
524
525      int deq_waitcmd(eibcmd_t*);
526      bool emptyactivecmd();
527      bool fullactivecmd();
528      void activatecmd();
529      void checkcmd(dmacmd_t*);
530      void transdata(dmacmd_t*);
531      void comstep();
532
533   public:
534      Eib(Chip*);
535      ~Eib();
536
537      int enq_waitcmd(dmacmd_t*, dmacmdtag_t*);
```

```
538      void step();
539  };
540
541  /** ELF Loader ************************************************************/
542  #define EM_SPU 23
543
544  /*************************************************************************/
545  typedef struct {
546      uint id;
547      uint addr;
548      uint data;
549  } memdata;
550
551  /*************************************************************************/
552  typedef struct {
553      uint addr;
554      uint size;
555      char *name;
556  } functable;
557
558  /*************************************************************************/
559  class Loaderbe {
560   private:
561      uint getword(uint);
562      uint gethword(uint);
563      int checkendian();
564      uint big_endian;
565   public:
566      memdata *md;
567      int archtype;
568      int filetype;
569      int needintp;
570      uint entry;
571
572      functable *ft;
573
574      Loaderbe();
575      ~Loaderbe();
576      int loadelf(char *);
577  };
578
579  /** functions *************************************************************/
580  // in spe.cc
581  uint_032t exts32(uint_032t, int);
582  void spefunccall(Spe *);
583  // in etc.cc
584  ullint gettime();
585  void readmemimage(Chip *, uint_032t *);
586  void cyclelog(Chip *);
587  ullint atoi_postfix(const char *);
588
589  /** SPU instruction ID Definition *****************************************/
590  /*************************************************************************/
591  // Memory - Load/Store Instructions
592  #define LQD_______ 0x001
593  #define LQX_______ 0x002
594  #define LQA_______ 0x003
595  #define LQR_______ 0x004
596  #define STQD______ 0x005
597  #define STQX______ 0x006
598  #define STQA______ 0x007
599  #define STQR______ 0x008
600  #define CBD_______ 0x009
601  #define CBX_______ 0x00a
602  #define CHD_______ 0x00b
603  #define CHX_______ 0x00c
604  #define CWD_______ 0x00d
605  #define CWX_______ 0x00e
606  #define CDD_______ 0x00f
607  #define CDX_______ 0x010
608
609  // Constant-Formation Instructions
610  #define ILH_______ 0x101
611  #define ILHU______ 0x102
612  #define IL________ 0x103
613  #define ILA_______ 0x104
614  #define IOHL______ 0x105
```



```
615  #define FSMBI_____ 0x106
616
617  // Integer and Logical Instructions
618  #define AH________ 0x201
619  #define AHI_______ 0x202
620  #define A_________ 0x203
621  #define AI________ 0x204
622  #define SFH_______ 0x205
623  #define SFHI______ 0x206
624  #define SF________ 0x207
625  #define SFI_______ 0x208
626  #define ADDX______ 0x209
627  #define CG________ 0x20a
628  #define CGX_______ 0x20b
629  #define SFX_______ 0x20c
630  #define BG________ 0x20d
631  #define BGX_______ 0x20e
632  #define MPY_______ 0x20f
633  #define MPYU______ 0x210
634  #define MPYI______ 0x211
635  #define MPYUI_____ 0x212
636  #define MPYA______ 0x213
637  #define MPYH______ 0x214
638  #define MPYS______ 0x215
639  #define MPYHH_____ 0x216
640  #define MPYHHA____ 0x217
641  #define MPYHHU____ 0x218
642  #define MPYHHAU___ 0x219
643  #define CLZ_______ 0x21a
644  #define CNTB______ 0x21b
645  #define FSMB______ 0x21c
646  #define FSMH______ 0x21d
647  #define FSM_______ 0x21e
648  #define GBB_______ 0x21f
649  #define GBH_______ 0x220
650  #define GB________ 0x221
651  #define AVGB______ 0x222
652  #define ABSDB_____ 0x223
653  #define SUMB______ 0x224
654  #define XSBH______ 0x225
655  #define XSHW______ 0x226
656  #define XSWD______ 0x227
657  #define AND_______ 0x228
658  #define ANDC______ 0x229
659  #define ANDBI_____ 0x22a
660  #define ANDHI_____ 0x22b
661  #define ANDI______ 0x22c
662  #define OR________ 0x22d
663  #define ORC_______ 0x22e
664  #define ORBI______ 0x22f
665  #define ORHI______ 0x230
666  #define ORI_______ 0x231
667  #define ORX_______ 0x232
668  #define XOR_______ 0x233
669  #define XORBI_____ 0x234
670  #define XORHI_____ 0x235
671  #define XORI______ 0x236
672  #define NAND______ 0x237
673  #define NOR_______ 0x238
674  #define EQV_______ 0x239
675  #define SELB______ 0x23a
676  #define SHUFB_____ 0x23b
677
678  // Shift and Rotate Instructions
679  #define SHLH______ 0x301
680  #define SHLHI_____ 0x302
681  #define SHL_______ 0x303
682  #define SHLI______ 0x304
683  #define SHLQBI____ 0x305
684  #define SHLQBII___ 0x306
685  #define SHLQBY____ 0x307
686  #define SHLQBYI___ 0x308
687  #define SHLQBYBI_ 0x309
688  #define ROTH______ 0x30a
689  #define ROTHI_____ 0x30b
690  #define ROT_______ 0x30c
691  #define ROTI______ 0x30d
```

```
692  #define ROTQBY___ 0x30e
693  #define ROTQBYI__ 0x30f
694  #define ROTQBYBI_ 0x310
695  #define ROTQBI___ 0x311
696  #define ROTQBII__ 0x312
697  #define ROTHM____ 0x313
698  #define ROTHMI___ 0x314
699  #define ROTM_____ 0x315
700  #define ROTMI____ 0x316
701  #define ROTQMBY__ 0x317
702  #define ROTQMBYI_ 0x318
703  #define ROTQMBYBI 0x319
704  #define ROTQMBI__ 0x31a
705  #define ROTQMBII_ 0x31b
706  #define ROTMAH___ 0x31c
707  #define ROTMAHI__ 0x31d
708  #define ROTMA____ 0x31e
709  #define ROTMAI___ 0x31f
710
711  // Compare, Branch, and Halt Instructions
712  #define HEQ______ 0x401
713  #define HEQI_____ 0x402
714  #define HGT______ 0x403
715  #define HGTI_____ 0x404
716  #define HLGT_____ 0x405
717  #define HLGTI____ 0x406
718  #define CEQB_____ 0x407
719  #define CEQBI____ 0x408
720  #define CEQH_____ 0x409
721  #define CEQHI____ 0x40a
722  #define CEQ______ 0x40b
723  #define CEQI_____ 0x40c
724  #define CGTB_____ 0x40d
725  #define CGTBI____ 0x40e
726  #define CGTH_____ 0x40f
727  #define CGTHI____ 0x410
728  #define CGT______ 0x411
729  #define CGTI_____ 0x412
730  #define CLGTB____ 0x413
731  #define CLGTBI___ 0x414
732  #define CLGTH____ 0x415
733  #define CLGTHI___ 0x416
734  #define CLGT_____ 0x417
735  #define CLGTI____ 0x418
736  #define BR_______ 0x419
737  #define BRA______ 0x41a
738  #define BRSL_____ 0x41b
739  #define BRASL____ 0x41c
740  #define BI_______ 0x41d
741  #define IRET_____ 0x41e
742  #define BISLED___ 0x41f
743  #define BISL_____ 0x420
744  #define BRNZ_____ 0x421
745  #define BRZ______ 0x422
746  #define BRHNZ____ 0x423
747  #define BRHZ_____ 0x424
748  #define BIZ______ 0x425
749  #define BINZ_____ 0x426
750  #define BIHZ_____ 0x427
751  #define BIHNZ____ 0x428
752
753  // Hint-for-Branch Instructions
754  #define HBR______ 0x501
755  #define HBRA_____ 0x502
756  #define HBRR_____ 0x503
757
758  // Floating-Point Instructions
759  #define FA_______ 0x601
760  #define DFA______ 0x602
761  #define FS_______ 0x603
762  #define DFS______ 0x604
763  #define FM_______ 0x605
764  #define DFM______ 0x606
765  #define FMA______ 0x607
766  #define DFMA_____ 0x608
767  #define FNMS_____ 0x609
768  #define DFNMS____ 0x60a
```



```
769  #define FMS______ 0x60b
770  #define DFMS_____ 0x60c
771  #define DFNMA____ 0x60d
772  #define FREST____ 0x60e
773  #define FRSQEST__ 0x60f
774  #define FI_______ 0x610
775  #define CSFLT____ 0x611
776  #define CFLTS____ 0x612
777  #define CUFLT____ 0x613
778  #define CFLTU____ 0x614
779  #define FRDS_____ 0x615
780  #define FESD_____ 0x616
781  #define DFCEQ____ 0x617
782  #define DFCMEQ___ 0x618
783  #define DFCGT____ 0x619
784  #define DFCMGT___ 0x61a
785  #define DFTSV____ 0x61b
786  #define FCEQ_____ 0x61c
787  #define FCMEQ____ 0x61d
788  #define FCGT_____ 0x61e
789  #define FCMGT____ 0x61f
790  #define FSCRWR___ 0x620
791  #define FSCRRD___ 0x621
792
793  // Control Instructions
794  #define STOP_____ 0x701
795  #define STOPD____ 0x702
796  #define LNOP_____ 0x703
797  #define NOP______ 0x704
798  #define SYNC_____ 0x705
799  #define DSYNC____ 0x706
800  #define MFSPR____ 0x707
801  #define MTSPR____ 0x708
802
803  // Channel Instructions
804  #define RDCH_____ 0x801
805  #define WRCH_____ 0x802
806  #define RCHCNT___ 0x803
807
808  // Undefined (Illegal Instruction)
809  #define UNDEF____ 0xfff
810
811  /*************************************************************************/
```

file name: main.cc

```
  1  /*********************************************************************/
  2  /* SimCell: Cell/B.E. Processor Simulator        Arch Lab. TOKYO TECH */
  3  /*********************************************************************/
  4  #include "define.h"
  5
  6  enum {
  7      LOOP_RUN,
  8      LOOP_EXIT,
  9  };
 10
 11  /*********************************************************************/
 12  void usage(void)
 13  {
 14      char usagemessage[] = "\
 15    -f: ignore errors due to DMA address violation\n\
 16    -m [filename]: specify the memory image file\n\
 17    -e[num][kmg]: stop simulation after num cycles executed\n\
 18    -l[num][kmg]: log mode, writes memory access log to log.dat\n\
 19    -v[num][kmg]: verbose mode, prints '.' by num\n\
 20    -n[num]: specify the number of SPEs\n\
 21    -K[num]: specify the memory read latency\n\
 22    -L[num]: specify the memory write latency\n\
 23    -M[num]: specify the memory bandwidth\n\
 24    -N[num]: specify the core to core latency\n\
 25    -O[num]: specify the core to core bandwidth\n\
 26    -d[num]: specify the mode level of debug mode\n\
 27    e.g. SimCell -n1 -K300 -M30 -m mem006.txt spe006\n";
 28
 29      printf("Usage: SimCell [-option] object_file_name \n");
 30      printf("%s", usagemessage);
 31  }
 32
 33  /*********************************************************************/
 34  char *checkarg(Chip *chip, int argc, char **argv)
 35  {
 36      printf("%s %s\n", L_NAME, L_VER);
 37      uint num;
 38      int bin_index = -1;
 39
 40      for (int i = 1; argv[i] != NULL; i++) {
 41          char *opt = argv[i];
 42          if (opt[0] != '-') {
 43              if (bin_index != -1) {
 44                  fprintf(stderr, "## can't handle multiple binary files\n");
 45                  exit(1);
 46              }
 47              bin_index = i;
 48              continue;
 49          }
 50          switch (opt[1]) {
 51          case 'n':
 52              num = atoi(&opt[2]);
 53              if (num > MAX_SPES) {
 54                  fprintf(stderr, "## too many SPEs\n");
 55                  exit(1);
 56              }
 57              chip->spe_num = num;
 58              break;
 59          case 'd':
 60              num = atoi(&opt[2]);
 61              if (num > MAX_DEBUG_MODE) {
 62                  fprintf(stderr, "## debug mode %d not available\n", num);
 63                  exit(1);
 64              }
 65              chip->debug_mode = num;
 66              break;
 67          case 'e':
 68              chip->max_cycle = atoi_postfix(&opt[2]);
 69              if (!chip->max_cycle)
 70                  chip->max_cycle = MAX_CYCLE_DEF;
 71              break;
 72          case 'l':
 73              chip->log_cycle = atoi_postfix(&opt[2]);
 74              if (!chip->log_cycle)
 75                  chip->log_cycle = LOGMODE_INT_DEF;
 76              chip->setlogmode();
```



```
 77              break;
 78          case 'v':
 79              chip->verb_cycle = atoi_postfix(&opt[2]);
 80              if (!chip->verb_cycle)
 81                  chip->verb_cycle = VERBOSE_INT_DEF;
 82              break;
 83          case 'f':
 84              chip->force_mode = 1;
 85              break;
 86          case 'K':
 87              chip->mem_rddelay = atoi(&opt[2]);
 88              break;
 89          case 'L':
 90              chip->mem_wrdelay = atoi(&opt[2]);
 91              break;
 92          case 'M':
 93              chip->mem_width = atoi(&opt[2]);
 94              break;
 95          case 'N':
 96              chip->c2c_delay = atoi(&opt[2]);
 97              break;
 98          case 'O':
 99              chip->c2c_width = atoi(&opt[2]);
100              break;
101          case 'm':
102              opt = argv[++i];
103              if (opt == NULL) {
104                  fprintf(stderr, "## image file is not specified\n");
105                  exit(1);
106              }
107              chip->mem_imagefile = opt;
108              break;
109          default:
110              fprintf(stderr, "## -%c: wrong option!\n", opt[1]);
111              usage();
112              exit(1);
113          }
114      }
115      if (bin_index == -1) {
116          usage();
117          exit(1);
118      }
119
120      printf("## SPEs: %d\n", chip->spe_num);
121      printf("## Mem Latency R/W, BW:%4d/%4d,%4d | ",
122             chip->mem_rddelay, chip->mem_wrdelay, chip->mem_width);
123      printf("C2C Latency, BW:%4d,%4d\n",
124             chip->c2c_delay, chip->c2c_width);
125      if (chip->debug_mode)
126          printf("## debug mode %d.\n", chip->debug_mode);
127
128      return argv[bin_index];
129  }
130
131  /*********************************************************************/
132  int speinit(Chip *chip, char *filename, Spe *spe)
133  {
134      Loaderbe *ld = new Loaderbe();
135
136      /*** load and check file ***/
137      if (ld->loadelf(filename)) {
138          delete ld;
139          return 1;
140      }
141
142      if (ld->archtype != EM_SPU) {
143          fprintf(stderr, "## incorrect architecture : %d\n",
144                  ld->archtype);
145          delete ld;
146          return 1;
147      }
148      if (ld->filetype != ET_EXEC) {
149          fprintf(stderr, "## this file isn't executable : %s\n",
150                  filename);
151          delete ld;
152          return 1;
153      }
```

```
154
155     /*** write program to memory ***/
156     spe->as->pc = ld->entry;
157
158     for (int i = 0; ld->md[i].id != 0; i++) {
159         if (ld->md[i].id == 4)
160             spe->ls->write4b((uint_032t) ld->md[i].addr,
161                              (uint_032t) ld->md[i].data);
162         else
163             spe->ls->write4b((uint_032t) ld->md[i].addr,
164                              (uint_032t) ld->md[i].data &
165                              (0xffffffff << (32 - ld->md[i].id * 8)));
166     }
167
168     delete ld;
169     ld = NULL;
170     return 0;
171 }
172
173 /*********************************************************************/
174 void initialize(Chip *chip, char *filename)
175 {
176     Mmu *mmu = chip->mmu;
177     Spe **spe = chip->spe;
178
179     for (int i = 0; i < chip->spe_num; i++) {
180         if (speinit(chip, filename, spe[i])) {
181             fprintf(stderr, "Program load Error\n");
182             exit(1);
183         }
184     }
185
186     if (chip->log_mode)
187         chip->initlogmode();
188
189     uint_032t map_addr[chip->spe_num];
190     for (int i = 0; i < chip->spe_num; i++)
191         map_addr[i] = 0x80000000 + LS_SIZE * i;
192     readmemimage(chip, map_addr);
193
194     for (int i = 0; i < chip->spe_num; i++)
195         mmu->mapls(map_addr[i], i + 1);
196
197     for (int i = 0; i < chip->spe_num; i++)
198         spe[i]->spu->start();
199
200     chip->running_core = chip->spe_num;
201
202     gettime();                  // reset elapsed time
203 }
204
205 /*********************************************************************/
206 void finalize(Chip *chip)
207 {
208     ullint sim_time = gettime();
209     MainMemory *mem = chip->mem;
210     Spe **spe = chip->spe;
211
212     /*** print result ***/
213     if (chip->debug_mode == 0) {
214         if (chip->cycle < chip->max_cycle)
215             printf("\n## All SPEs are stopped.\n");
216         else
217             printf("\n## Execution Cycles reached the limit.\n");
218     }
219
220     ullint inst_count = 0;
221     for (int i = 0; i < chip->spe_num; i++)
222         inst_count += spe[i]->ss->instcnt;
223
224     if (chip->debug_mode == 1) {
225         for (int i = 0; i < chip->spe_num; i++) {
226             printf("\n[[SPE %d]]\n", i);
227             printf("execute instruction : %10llu\n",
228                    spe[i]->ss->instcnt);
229             spe[i]->spu->printregall();
230             spe[i]->ls->print();
```



```
231         }
232         mem->print();
233     }
234
235     if (chip->debug_mode == 0) {
236         printf("## SPEs:%3d | Cycle:%11llu | Inst:%11llu | \n",
237                chip->spe_num, chip->cycle, inst_count);
238         printf("## Execution  Time:%10.4f | Freq:%7.3f\n",
239                (double)chip->cycle /
240                (double)(PROCESSOR_FREQ * AVERAGE_IPC * 1000000),
241                (double) PROCESSOR_FREQ / 1000.0);
242         printf("## Simulation Time:%10.4f | MIPS:%7.3f\n",
243                (double) sim_time / 1000000,
244                (double) inst_count / sim_time);
245     }
246
247     /*** free when log-mode ***/
248     if (chip->log_mode == 1) {
249         chip->writelogdata();
250         chip->finilogmode();
251     }
252 }
253
254 /*********************************************************************/
255 int loopcond(Chip * chip)
256 {
257     if (chip->cycle >= chip->max_cycle)
258         return LOOP_EXIT;
259
260     if (chip->running_core == 0)
261         return LOOP_EXIT;
262
263     return LOOP_RUN;
264 }
265
266 /*********************************************************************/
267 int main(int argc, char **argv)
268 {
269     Chip *chip = new Chip();
270     Spe *spe[MAX_SPES];
271     new MainMemory(chip);
272     new Mmu(chip);
273     new SpuInstinfo(chip);
274
275     char *bin_name = checkarg(chip, argc, argv);
276
277     for (int i = 0; i < chip->spe_num; i++)
278         spe[i] = new Spe(chip, i + 1);
279     Eib *eib = new Eib(chip);
280
281     initialize(chip, bin_name);
282
283     while (loopcond(chip) == LOOP_RUN) {
284         chip->cycle++;
285         for (int i = 0; i < chip->spe_num; i++) {
286             spe[i]->spu->step();
287             spe[i]->mfc->step();
288
289         }
290         eib->step();
291         cyclelog(chip);
292     }
293
294     finalize(chip);
295     return 0;
296 }
297
298 /*********************************************************************/
```

file name: memory.cc

```
  1  /*****************************************************************/
  2  /* SimCell: Cell/B.E. Processor Simulator        Arch Lab. TOKYO TECH */
  3  /*****************************************************************/
  4  #include "define.h"
  5
  6  /*****************************************************************/
  7  MainMemory::MainMemory(Chip * chip)
  8  {
  9      chip->mem = this;
 10      for (int i = 0; i < BLOCK_TABLE_SIZE; i++)
 11          block_table[i] = NULL;
 12  }
 13
 14  /*****************************************************************/
 15  MainMemory::~MainMemory()
 16  {
 17      for (int i = 0; i < BLOCK_TABLE_SIZE; i++) {
 18          delete[] block_table[i];
 19          block_table[i] = NULL;
 20      }
 21  }
 22
 23  /*****************************************************************/
 24  uint_032t *MainMemory::allocblock(uint_032t addr)
 25  {
 26      uint_032t *ret = new uint_032t[BLOCK_SIZE / sizeof(uint_032t)];
 27      for (int i = 0; i < (int)(BLOCK_SIZE / sizeof(uint_032t)); i++)
 28          ret[i] = 0;
 29      block_table[addr / BLOCK_SIZE] = ret;
 30      return ret;
 31  }
 32
 33  /*****************************************************************/
 34  void MainMemory::read4b(uint_032t addr, uint_032t * data)
 35  {
 36      uint_032t *block = block_table[addr / BLOCK_SIZE];
 37      block = (block != NULL) ? block : allocblock(addr);
 38      *data = block[(addr % BLOCK_SIZE) / sizeof(uint_032t)];
 39  }
 40
 41  /*****************************************************************/
 42  void MainMemory::read1b(uint_032t addr, uint_008t * data)
 43  {
 44      uint_032t temp;
 45      int offset = 24 - (addr & 0x3) * 8;
 46      read4b(addr, &temp);
 47      *data = (temp >> offset) & 0xff;
 48  }
 49
 50  /*****************************************************************/
 51  void MainMemory::read2b(uint_032t addr, uint_016t * data)
 52  {
 53      uint_032t temp;
 54      int offset = 16 - (addr & 0x2) * 8;
 55      read4b(addr, &temp);
 56      *data = (temp >> offset) & 0xffff;
 57  }
 58
 59  /*****************************************************************/
 60  void MainMemory::read8b(uint_032t addr, uint_064t * data)
 61  {
 62      addr = addr & ~0x7;
 63      uint_032t temp_h, temp_l;
 64      read4b(addr, &temp_h);
 65      read4b(addr + 4, &temp_l);
 66      *data = ((uint_064t) temp_h << 32) || temp_l;
 67  }
 68
 69  /*****************************************************************/
 70  void MainMemory::readnb(uint_032t addr, int size, uint_032t data[])
 71  {
 72      int i, j, index;
 73      uint_032t temp = 0;
 74      uint_008t c;
 75
 76      index = 0;
```



```
 77      for (i = 0; i < size; i += sizeof(uint_032t)) {
 78          read4b(addr + i, &data[index]);
 79          index++;
 80      }
 81      for (j = 0; i + j < size; j++) {
 82          read1b(addr + i + j, &c);
 83          temp = (temp << 8) || c;
 84      }
 85      if (j != 0)
 86          data[index] = temp << (32 - j * 8);
 87  }
 88
 89  /*****************************************************************/
 90  void MainMemory::write4b(uint_032t addr, uint_032t data)
 91  {
 92      uint_032t *block = block_table[addr / BLOCK_SIZE];
 93      block = (block != NULL) ? block : allocblock(addr);
 94      block[(addr % BLOCK_SIZE) / sizeof(uint_032t)] = data;
 95  }
 96
 97  /*****************************************************************/
 98  void MainMemory::write1b(uint_032t addr, uint_008t data)
 99  {
100      int offset = 24 - (addr & 0x3) * 8;
101      uint_032t ins = (uint_032t) data << offset;
102      uint_032t mask = 0xff << offset;
103      uint_032t temp;
104      read4b(addr, &temp);
105      temp = ins | (temp & ~mask);
106      write4b(addr, temp);
107  }
108
109  /*****************************************************************/
110  void MainMemory::write2b(uint_032t addr, uint_016t data)
111  {
112      int offset = 16 - (addr & 0x2) * 8;
113      uint_032t ins = (uint_032t) data << offset;
114      uint_032t mask = 0xffff << offset;
115      uint_032t temp;
116      read4b(addr, &temp);
117      temp = ins | (temp & ~mask);
118      write4b(addr, temp);
119  }
120
121  /*****************************************************************/
122  void MainMemory::write8b(uint_032t addr, uint_064t data)
123  {
124      addr = addr & ~0x7;
125      write4b(addr, (uint_032t) (data >> 32));
126      write4b(addr + 4, (uint_032t) data);
127  }
128
129  /*****************************************************************/
130  void MainMemory::writenb(uint_032t addr, int size, uint_032t data[])
131  {
132      int i, j, index;
133
134      index = 0;
135      for (i = 0; i < size; i += sizeof(uint_032t)) {
136          write4b(addr + i, data[index]);
137          index++;
138      }
139      for (j = 0; i + j < size; j++)
140          write1b(addr + i + j, (data[index] >> (24 - j * 8)) & 0xff);
141  }
142
143  /*****************************************************************/
144  void MainMemory::print()
145  {
146      int flag = 0;
147      uint_032t *block;
148      uint_032t addr;
149      printf("[[MAIN MEMORY]]\n");
150      for (uint i = 0; i < BLOCK_TABLE_SIZE; i++) {
151          block = block_table[i];
152          if (block != NULL) {
153              printf("[MEMORY BLOCK: 0x%08x]", i);
```

```
154             flag = 0;
155             for (uint j = 0; j < BLOCK_SIZE / sizeof(uint_032t); j++) {
156                 addr = i * BLOCK_SIZE + j * sizeof(uint_032t);
157                 if ((j % 16) == 0)
158                     printf("\n%08x: ", (uint) addr);
159                 if (block[j] == 0)
160                     printf("-------- ");
161                 else
162                     printf("%08x ", (uint) block[j]);
163             }
164             printf("\n\n");
165         } else if (flag == 0) {
166             flag = 1;
167         }
168     }
169 }
170
171 /*****************************************************************/
```


file name: loaderbe.cc

```
  1 /*****************************************************************/
  2 /*  SimCell: Simulator of Cell/B.E.              Arch Lab. TOKYO TECH */
  3 /*****************************************************************/
  4 #include "define.h"
  5
  6 /*****************************************************************/
  7 Loaderbe::Loaderbe()
  8 {
  9     md = NULL;
 10     ft = NULL;
 11     big_endian = checkendian();
 12 }
 13
 14 /*****************************************************************/
 15 Loaderbe::~Loaderbe()
 16 {
 17     if (ft != NULL) {
 18         functable *temp;
 19         for (temp = ft; temp->name != NULL; temp++) {
 20             delete[]temp->name;
 21             temp->name = NULL;
 22         }
 23         delete[]ft;
 24         ft = NULL;
 25     }
 26     if (md != NULL) {
 27         delete[]md;
 28         md = NULL;
 29     }
 30 }
 31
 32 /*****************************************************************/
 33 uint Loaderbe::gethword(uint data)
 34 {
 35     if (big_endian) {
 36         return data;
 37     } else {
 38         return ((data & 0xff) << 8) | ((data & 0xff00) >> 8);
 39     }
 40 }
 41
 42 /*****************************************************************/
 43 uint Loaderbe::getword(uint data)
 44 {
 45     if (big_endian) {
 46         return data;
 47     } else {
 48         return ((data & 0xff) << 24) | ((data & 0xff00) << 8) |
 49             ((data & 0xff0000) >> 8) | ((data & 0xff000000) >> 24);
 50     }
 51 }
 52
 53 /*****************************************************************/
 54 int Loaderbe::checkendian()
 55 {
 56     union {
 57         unsigned short sh;
 58         unsigned char ch[2];
 59     } test;
 60     test.sh = 0xfeff;
 61     return (test.ch[0] == 0xfe);
 62 }
 63
 64 /*****************************************************************/
 65 int Loaderbe::loadelf(char *filename)
 66 {
 67     int fd = 0;
 68     struct stat sb;
 69     char elf_magic[6] = { ELFMAG0, ELFMAG1, ELFMAG2, ELFMAG3,
 70         ELFCLASS32, ELFDATA2MSB
 71     };
 72     char *file;
 73     uint *temp;
 74     uint fsize;
 75     uint offset;
 76     int mdcount = 0;
```

```
 77      uint *mdoff;
 78      uint i, j, k, imax;
 79      Elf32_Ehdr *ehdr;
 80      Elf32_Phdr **phdr;
 81      Elf32_Shdr *shdr;
 82      Elf32_Sym *symtab;
 83      uint symtablen;
 84      char *shstrtab, *strtab, *funcname;
 85
 86      // open file
 87      fd = open(filename, O_RDONLY);
 88      if (fd == -1) {
 89          fprintf(stderr, "## Can't find file : %s\n", filename);
 90          close(fd);
 91          return 1;
 92      }
 93
 94      fstat(fd, &sb);
 95      fsize = sb.st_size / sizeof(char);
 96      file = new char[(fsize + 3) & 0xfffffffc];
 97
 98      // read file
 99      i = read(fd, file, fsize);
100      close(fd);
101      if (i != fsize) {
102          fprintf(stderr, "## Can't read file : %s\n", filename);
103          delete[]file;
104          file = NULL;
105          return 1;
106      }
107      // ELF header
108      ehdr = (Elf32_Ehdr *) file;
109      for (i = 0; i < sizeof(elf_magic); i++) {
110          if (ehdr->e_ident[i] != elf_magic[i]) {
111              fprintf(stderr, "## not a big-endian ELF32 : %s\n",
112                      filename);
113              delete[]file;
114              file = NULL;
115              return 1;
116          }
117      }
118      filetype = gethword(ehdr->e_type);
119      archtype = gethword(ehdr->e_machine);
120      entry = getword(ehdr->e_entry);
121      needintp = 0;
122
123      // program header
124      imax = gethword(ehdr->e_phnum);
125      phdr = new Elf32_Phdr *[imax];
126      mdoff = new uint[imax];
127
128      offset = getword(ehdr->e_phoff);
129      mdcount = 0;
130      for (i = 0; i < imax; i++) {
131          phdr[i] = (Elf32_Phdr *) (file + offset);
132          mdoff[i] = mdcount;
133          if (getword(phdr[i]->p_type) == PT_INTERP)
134              needintp = 1;
135          else if (getword(phdr[i]->p_type) == PT_LOAD)
136              mdcount += (getword(phdr[i]->p_memsz) + 3) >> 2;
137          offset += gethword(ehdr->e_phentsize);
138      }
139
140      // write struct memdata
141      md = new memdata[mdcount + 1];
142      md[mdcount].id = 0;
143      mdcount = 0;
144
145      for (i = 0; i < imax; i++) {
146          if (getword(phdr[i]->p_type) == PT_LOAD) {
147              k = mdoff[i];
148              for (j = 0; j < getword(phdr[i]->p_memsz); j += 4) {
149                  md[k].id = 4;
150                  md[k].addr = getword(phdr[i]->p_vaddr) + j;
151                  md[k].data = 0;
152                  k++;
153              }
```



```
154              if (getword(phdr[i]->p_memsz) % 4)
155                  md[k - 1].id = (getword(phdr[i]->p_memsz) % 4);
156              temp = (uint *) (file + getword(phdr[i]->p_offset));
157              k = mdoff[i];
158              for (j = 0; j < getword(phdr[i]->p_filesz); j += 4) {
159                  md[k].data = getword(*temp);
160                  k++;
161                  temp++;
162              }
163          }
164      }
165
166      // section header and symbol table
167      imax = gethword(ehdr->e_shnum);
168      offset = getword(ehdr->e_shoff);
169      symtab = NULL;
170      strtab = NULL;
171      symtablen = 0;
172      shdr = (Elf32_Shdr *) (file + offset + gethword(ehdr->e_shentsize)
173                             * gethword(ehdr->e_shstrndx));
174      shstrtab = file + getword(shdr->sh_offset);
175
176      for (i = 0; i < imax; i++) {
177          shdr = (Elf32_Shdr *) (file + offset);
178          if (getword(shdr->sh_type) == SHT_SYMTAB) {
179              symtab = (Elf32_Sym *) (file + getword(shdr->sh_offset));
180              symtablen = getword(shdr->sh_size) / sizeof(Elf32_Sym);
181          }
182          if (strcmp(shstrtab + getword(shdr->sh_name), ".strtab") == 0)
183              strtab = file + getword(shdr->sh_offset);
184          offset += gethword(ehdr->e_shentsize);
185      }
186      if (symtab != NULL) {
187          imax = 0;
188          for (i = 0; i < symtablen; i++) {
189              if (ELF32_ST_TYPE(symtab[i].st_info) == STT_FUNC)
190                  imax++;
191          }
192          ft = new functable[imax + 1];
193          j = 0;
194          for (i = 0; i < symtablen; i++) {
195              if (ELF32_ST_TYPE(symtab[i].st_info) == STT_FUNC) {
196                  ft[j].addr = getword(symtab[i].st_value);
197                  ft[j].size = getword(symtab[i].st_size);
198                  funcname = strtab + getword(symtab[i].st_name);
199                  k = strlen(funcname);
200                  ft[j].name = new char[k + 1];
201                  strncpy(ft[j].name, funcname, k + 1);
202                  j++;
203              }
204          }
205          ft[imax].name = NULL;
206      }
207      delete[]mdoff;
208      mdoff = NULL;
209      delete[]phdr;
210      phdr = NULL;
211      delete[]file;
212      file = NULL;
213      return 0;
214  }
215
216  /*****************************************************************************/
```

file name: spe.cc

```
  1  /*****************************************************************************/
  2  /*  SimCell: Simulator of Cell/B.E.              Arch Lab. TOKYO TECH */
  3  /*****************************************************************************/
  4  #include <stdio.h>
  5  #include "define.h"
  6
  7  /*****************************************************************************/
  8  enum {
  9      PSLOT = 0,
 10
 11      // reason for SPE stop
 12      SR_NORMALEXIT = 0x2000,
 13      SR_FUNCCALL   = 0x2100,
 14      SR_SIGNALMASK = 0xffff,
 15      SR_STOP       = 0x10000,
 16      SR_HALT       = 0x20000,
 17      SR_ERROR      = 0x40000,
 18
 19      // reason for SPE stall
 20      ST_NOTRDCH = 0x100,
 21      ST_NOTWRCH = 0x200,
 22      ST_CHMASK = 0x1f,
 23  };
 24
 25  /*****************************************************************************/
 26  typedef union {
 27      uint_064t lli;
 28      double dbl;
 29  } united_064t;
 30
 31  /*****************************************************************************/
 32  #define ulow16(x) ((x) & 0xffff)
 33  #define low16(x) exts32(ulow16(x), 16)
 34  #define uhigh16(x) ((x) >> 16)
 35  #define high16(x) exts32(uhigh16(x), 16)
 36  #define slot(x, y) ((chip->big_endian) ? (x) : (y) - 1 - (x))
 37
 38  /*****************************************************************************/
 39  uint_032t exts32(uint_032t x, int y)
 40  {
 41      if (y == 32)
 42          return x;
 43      uint_032t temp = 0xffffffff << y;
 44      if (x & (1 << (y - 1)))
 45          return (temp | (x & ~temp));
 46
 47      else
 48          return (x & ~temp);
 49  }
 50
 51  /*****************************************************************************/
 52  Spu::Spu(Chip *chip, Spe *spe)
 53  {
 54      this->chip = chip;
 55      this->spe = spe;
 56      this->as = spe->as;
 57      this->ss = spe->ss;
 58      this->chi = spe->chi;
 59      inst = new SpuInst(chip, spe);
 60  }
 61
 62  /*****************************************************************************/
 63  Spu::~Spu()
 64  {
 65      delete inst;
 66      inst = NULL;
 67  }
 68
 69  /*****************************************************************************/
 70  void Spu::printregall()
 71  {
 72      int flag = 0;
 73      if (chip->debug_mode != 3)
 74          printf("[SPU Register]\n");
 75
 76      for (int i = 0; i < NREG; i++) {
```



```
 77          if ((as->reg[i]->get064t(0) | as->reg[i]->get064t(1)) == 0) {
 78              flag++;
 79              if (flag == 2)
 80                  printf("...\n");
 81          } else if (flag) {
 82              flag = 0;
 83          }
 84          if (flag < 2) {
 85              printf("r%3d            uint128 = ", i);
 86              as->reg[i]->print();
 87          }
 88      }
 89  }
 90
 91  /*****************************************************************************/
 92  void Spu::start()
 93  {
 94      as->state = SPE_RUN;
 95  }
 96
 97  /*****************************************************************************/
 98  void Spu::step()
 99  {
100      if (checkstate() != SPE_RUN)
101          return;
102      inst->clear();
103      fetch(inst);
104      decode(inst);
105      execute(inst);
106      if (inst->nextstate != SPE_STALL) {
107          lsaccess(inst);
108          writeback(inst);
109          ss->instcnt++;
110      }
111      setnextpc(inst);
112      as->state = inst->nextstate;
113  }
114
115  /*****************************************************************************/
116  inline int Spu::checkstate()
117  {
118      if (as->state == SPE_STALL) {
119          int ca = ss->stall_reason & ST_CHMASK;
120          if (ss->stall_reason & ST_NOTRDCH) {
121              if (spe->chi->spureadable(ca))
122                  start();
123          }
124          else if (ss->stall_reason & ST_NOTWRCH) {
125              if (spe->chi->spuwritable(ca))
126                  start();
127          }
128      }
129      if (as->state == SPE_STOP) {
130          if (as->stop_reason == (SR_STOP | SR_FUNCCALL)) {
131              spefunccall(spe);
132              start();
133          }
134          else {
135              chip->running_core--;
136              as->state = SPE_HALT;
137          }
138      }
139      return as->state;
140  }
141
142  /*****************************************************************************/
143  inline void Spu::fetch(SpuInst *inst)
144  {
145      spe->ls->read4b(as->pc, &inst->ir);
146      inst->pc = as->pc;
147      if (chip->log_mode)
148          chip->setlogdata(spe->unitid, inst->pc, MEM_INST_READ);
149  }
150
151  /*****************************************************************************/
152  inline void Spu::decode(SpuInst *inst)
153  {
```

```
154     inst->decode();
155 }
156
157 /*****************************************************************/
158 inline void Spu::execute(SpuInst *inst)
159 {
160     if (inst->pipe == PIPE_EVEN)
161         executeeven(inst);
162     else
163         executeodd(inst);
164 }
165
166 /*****************************************************************/
167 inline void Spu::load(uint_032t addr, Register * rt)
168 {
169     uint_032t data;
170     addr = addr & as->lslr & 0xfffffff0;
171     for (uint i = 0; i < NSLOT(uint_032t); i++) {
172         spe->ls->read4b(addr + (i * sizeof(uint_032t)), &data);
173         rt->put032t(i, data);
174     }
175     if (chip->log_mode)
176         chip->setlogdata(spe->unitid, addr, MEM_READ);
177 }
178
179 /*****************************************************************/
180 inline void Spu::store(uint_032t addr, Register * rt)
181 {
182     addr = addr & as->lslr & 0xfffffff0;
183     for (uint i = 0; i < NSLOT(uint_032t); i++)
184         spe->ls->write4b(addr + (i * sizeof(uint_032t)), rt->get032t(i));
185     if (chip->log_mode)
186         chip->setlogdata(spe->unitid, addr, MEM_WRITE);
187 }
188
189 /*****************************************************************/
190 inline void Spu::lsaccess(SpuInst *inst)
191 {
192     if (inst->isload())
193         load(inst->targ, inst->rrt);
194     else if (inst->isstore())
195         store(inst->targ, inst->rrt);
196 }
197
198 /*****************************************************************/
199 inline void Spu::writeback(SpuInst *inst)
200 {
201     if (inst->type & IT_W_RT)
202         as->reg[inst->rt]->put128t(inst->rrt->get128t());
203 }
204
205 /*****************************************************************/
206 inline void Spu::setnextpc(SpuInst *inst)
207 {
208     if(inst->nextstate != SPE_STALL)
209         as->pc = inst->nextpc;
210 }
211
212 /*****************************************************************/
213 inline void Spu::executeodd(SpuInst *inst)
214 {
215     Register *rra = inst->rra;
216     Register *rrb = inst->rrb;
217     Register *rrc = inst->rrc;
218     Register *rrt = inst->rrt;
219     uint imm = inst->imm;
220     uint pc = inst->pc;
221     uint_032t targ = inst->targ;
222     uint_032t nextpc = inst->nextpc;
223
224     if (chip->debug_mode == 2)
225         inst->exprinthead();
226     switch (inst->op) {
227     case LQD______:
228     case STQD_____:
229
230         // determine load/store target address
```



```
231         targ = (exts32(imm, 10) << 4) + rra->get032t(PSLOT);
232         break;
233     case LQX______:
234     case STQX_____:
235         targ = rra->get032t(PSLOT) + rrb->get032t(PSLOT);
236         break;
237     case LQA______:
238     case STQA_____:
239         targ = exts32(imm, 16) << 2;
240         break;
241     case LQR______:
242     case STQR_____:
243         targ = (exts32(imm, 16) << 2) + pc;
244         break;
245     case CBD______:
246     {
247         int t = (rra->get032t(PSLOT) + exts32(imm, 7)) & 0xf;
248         rrt->put064t(0, 0x1011121314151617ull);
249         rrt->put064t(1, 0x18191a1b1c1d1e1full);
250         rrt->put008t(t, 0x03);
251         break;
252     }
253     case CBX______:
254     {
255         int t = (rra->get032t(PSLOT) + rrb->get032t(PSLOT)) & 0xf;
256         rrt->put064t(0, 0x1011121314151617ull);
257         rrt->put064t(1, 0x18191a1b1c1d1e1full);
258         rrt->put008t(t, 0x03);
259         break;
260     }
261     case CHD______:
262     {
263         int t = (rra->get032t(PSLOT) + exts32(imm, 7)) & 0xe;
264         rrt->put064t(0, 0x1011121314151617ull);
265         rrt->put064t(1, 0x18191a1b1c1d1e1full);
266         rrt->put016t(t >> 1, 0x0203);
267         break;
268     }
269     case CHX______:
270     {
271         int t = (rra->get032t(PSLOT) + rrb->get032t(PSLOT)) & 0xe;
272         rrt->put064t(0, 0x1011121314151617ull);
273         rrt->put064t(1, 0x18191a1b1c1d1e1full);
274         rrt->put016t(t >> 1, 0x0203);
275         break;
276     }
277     case CWD______:
278     {
279         int t = (rra->get032t(PSLOT) + exts32(imm, 7)) & 0xc;
280         rrt->put064t(0, 0x1011121314151617ull);
281         rrt->put064t(1, 0x18191a1b1c1d1e1full);
282         rrt->put032t(t >> 2, 0x00010203);
283         break;
284     }
285     case CWX______:
286     {
287         int t = (rra->get032t(PSLOT) + rrb->get032t(PSLOT)) & 0xc;
288         rrt->put064t(0, 0x1011121314151617ull);
289         rrt->put064t(1, 0x18191a1b1c1d1e1full);
290         rrt->put032t(t >> 2, 0x00010203);
291         break;
292     }
293     case CDD______:
294     {
295         int t = (rra->get032t(PSLOT) + exts32(imm, 7)) & 0x8;
296         rrt->put064t(0, 0x1011121314151617ull);
297         rrt->put064t(1, 0x18191a1b1c1d1e1full);
298         rrt->put064t(t >> 3, 0x0001020304050607ull);
299         break;
300     }
301     case CDX______:
302     {
303         int t = (rra->get032t(PSLOT) + rrb->get032t(PSLOT)) & 0x8;
304         rrt->put064t(0, 0x1011121314151617ull);
305         rrt->put064t(1, 0x18191a1b1c1d1e1full);
306         rrt->put064t(t >> 3, 0x0001020304050607ull);
307         break;
```

```
308     }
309
310     /********/
311     case FSMBI____:
312     {
313         uint_032t s = imm;
314         for (int i = 0; i < NSLOT(uint_008t); i++)
315             rrt->put008t(i, (s & (0x8000 >> i)) ? 0xff : 0x00);
316         break;
317     }
318
319     /********/
320     case FSMB_____:
321     {
322         uint_032t s = rra->get032t(PSLOT);
323         for (int i = 0; i < NSLOT(uint_008t); i++)
324             rrt->put008t(i, (s & (0x8000 >> i)) ? 0xff : 0x00);
325         break;
326     }
327     case FSMH_____:
328     {
329         uint_032t s = rra->get032t(PSLOT);
330         for (int i = 0; i < NSLOT(uint_016t); i++)
331             rrt->put016t(i, (s & (0x80 >> i)) ? 0xffff : 0x0000);
332         break;
333     }
334     case FSM______:
335     {
336         uint_032t s = rra->get032t(PSLOT);
337         for (int i = 0; i < NSLOT(uint_032t); i++) {
338             rrt->put032t(i,
339                          (s & (0x8 >> i)) ? 0xffffffff : 0x00000000);
340         } break;
341     }
342     case GBB______:
343     {
344         uint_032t s = 0;
345         for (int i = 0; i < NSLOT(uint_008t); i++)
346             s = (s << 1) | (rra->get008t(i) & 0x1);
347         rrt->put032t(0, s);
348         rrt->put032t(1, 0);
349         rrt->put032t(2, 0);
350         rrt->put032t(3, 0);
351         break;
352     }
353     case GBH______:
354     {
355         uint_032t s = 0;
356         for (int i = 0; i < NSLOT(uint_016t); i++)
357             s = (s << 1) | (rra->get016t(i) & 0x1);
358         rrt->put032t(0, s);
359         rrt->put032t(1, 0);
360         rrt->put032t(2, 0);
361         rrt->put032t(3, 0);
362         break;
363     }
364     case GB_______:
365     {
366         uint_032t s = 0;
367         for (int i = 0; i < NSLOT(uint_032t); i++)
368             s = (s << 1) | (rra->get032t(i) & 0x1);
369         rrt->put032t(0, s);
370         rrt->put032t(1, 0);
371         rrt->put032t(2, 0);
372         rrt->put032t(3, 0);
373         break;
374     }
375     case ORX______:
376         rrt->put032t(0,
377                      rra->get032t(0) | rra->get032t(1) | rra->
378                      get032t(2) | rra->get032t(3));
379         rrt->put032t(1, 0);
380         rrt->put032t(2, 0);
381         rrt->put032t(3, 0);
382         break;
383     case SHUFB____:
384     {
```



```
385         uint_008t c;
386         for (int i = 0; i < NSLOT(uint_008t); i++) {
387             uint_008t b = rrc->get008t(i);
388             if ((b & 0xc0) == 0x80)    // 10xxxxxx
389                 c = 0x00;
390
391             else if ((b & 0xe0) == 0xc0)    // 110xxxxx
392                 c = 0xff;
393
394             else if ((b & 0xe0) == 0xe0)    // 111xxxxx
395                 c = 0x80;
396
397             else if (b & 0x10)  // 0xx1xxxx
398                 c = rrb->get008t(b & 0xf);
399
400             else            // 0xx0xxxx
401                 c = rra->get008t(b & 0xf);
402             rrt->put008t(i, c);
403         }
404         break;
405     }
406
407     /********/
408     case SHLQBI___:
409     {
410         int s = rrb->get032t(PSLOT) & 0x7;
411         if (s != 0) {
412             rrt->put064t(0, (rra->get064t(0) << s) |
413                          (rra->get064t(1) >> (64 - s)));
414             rrt->put064t(1, (rra->get064t(1) << s));
415         } else {
416             rrt->put128t(rra->get128t());
417         }
418         break;
419     }
420     case SHLQBII__:
421     {
422         int s = imm & 0x7;
423         if (s != 0) {
424             rrt->put064t(0, (rra->get064t(0) << s) |
425                          (rra->get064t(1) >> (64 - s)));
426             rrt->put064t(1, (rra->get064t(1) << s));
427         } else {
428             rrt->put128t(rra->get128t());
429         }
430         break;
431     }
432     case SHLQBY___:
433     {
434         int s = rrb->get032t(PSLOT) & 0x1f;
435         for (int b = 0; b < NSLOT(uint_008t); b++)
436             rrt->put008t(b, (b + s < 16) ? rra->get008t(b + s) : 0);
437         break;
438     }
439     case SHLQBYI__:
440     {
441         int s = imm & 0x1f;
442         for (int b = 0; b < NSLOT(uint_008t); b++)
443             rrt->put008t(b, (b + s < 16) ? rra->get008t(b + s) : 0);
444         break;
445     }
446     case SHLQBYBI_:
447     {
448         int s = (rrb->get032t(PSLOT) >> 3) & 0x1f;
449         for (int b = 0; b < NSLOT(uint_008t); b++)
450             rrt->put008t(b, (b + s < 16) ? rra->get008t(b + s) : 0);
451         break;
452     }
453     case ROTQBY___:
454     {
455         int s = rrb->get032t(PSLOT) & 0xf;
456         for (int b = 0; b < NSLOT(uint_008t); b++)
457             rrt->put008t(b, rra->get008t((b + s) & 0xf));
458         break;
459     }
460     case ROTQBYI__:
461     {
```

```
462         int s = imm & 0xf;
463         for (int b = 0; b < NSLOT(uint_008t); b++)
464             rrt->put008t(b, rra->get008t((b + s) & 0xf));
465         break;
466     }
467     case ROTQBYBI_:
468     {
469         int s = (rrb->get032t(PSLOT) >> 3) & 0xf;
470         for (int b = 0; b < NSLOT(uint_008t); b++)
471             rrt->put008t(b, rra->get008t((b + s) & 0xf));
472         break;
473     }
474     case ROTQBI___:
475     {
476         int s = rrb->get032t(PSLOT) & 0x7;
477         if (s != 0) {
478             rrt->put064t(0, (rra->get064t(0) << s) |
479                             (rra->get064t(1) >> (64 - s)));
480             rrt->put064t(1, (rra->get064t(1) << s) |
481                             (rra->get064t(0) >> (64 - s)));
482         } else {
483             rrt->put128t(rra->get128t());
484         }
485         break;
486     }
487     case ROTQBII__:
488     {
489         int s = imm & 0x7;
490         if (s != 0) {
491             rrt->put064t(0, (rra->get064t(0) << s) |
492                             (rra->get064t(1) >> (64 - s)));
493             rrt->put064t(1, (rra->get064t(1) << s) |
494                             (rra->get064t(0) >> (64 - s)));
495         } else {
496             rrt->put128t(rra->get128t());
497         }
498         break;
499     }
500     case ROTQMBY__:
501     {
502         int s = (0 - rrb->get032t(PSLOT)) & 0x1f;
503         for (int b = 0; b < NSLOT(uint_008t); b++)
504             rrt->put008t(b, (b >= s) ? rra->get008t(b - s) : 0x00);
505         break;
506     }
507     case ROTQMBYI_:
508     {
509         int s = (0 - imm) & 0x1f;
510         for (int b = 0; b < NSLOT(uint_008t); b++)
511             rrt->put008t(b, (b >= s) ? rra->get008t(b - s) : 0x00);
512         break;
513     }
514     case ROTQMBYBI:
515     {
516         int s = (0 - (rrb->get032t(PSLOT) >> 3)) & 0x1f;
517         for (int b = 0; b < NSLOT(uint_008t); b++)
518             rrt->put008t(b, (b >= s) ? rra->get008t(b - s) : 0x00);
519         break;
520     }
521     case ROTQMBI__:
522     {
523         int s = (0 - rrb->get032t(PSLOT)) & 0x7;
524         if (s != 0) {
525             rrt->put064t(0, (rra->get064t(0) >> s));
526             rrt->put064t(1, (rra->get064t(1) >> s) |
527                             (rra->get064t(0) << (64 - s)));
528         } else {
529             rrt->put128t(rra->get128t());
530         }
531         break;
532     }
533     case ROTQMBII_:
534     {
535         int s = (0 - imm) & 0x7;
536         if (s != 0) {
537             rrt->put064t(0, (rra->get064t(0) >> s));
538             rrt->put064t(1, (rra->get064t(1) >> s) |
```



```
539                             (rra->get064t(0) << (64 - s)));
540         } else {
541             rrt->put128t(rra->get128t());
542         }
543         break;
544     }
545
546     /********/
547     case BR_______:
548
549         // determine branch target address
550         nextpc = (pc + exts32(imm << 2, 18)) & as->lslr;
551         break;
552     case BRA______:
553         nextpc = (exts32(imm << 2, 18)) & as->lslr;
554         break;
555     case BRSL_____:
556         rrt->put032t(PSLOT, (pc + 4) & as->lslr);
557         rrt->put032t(1, 0);
558         rrt->put032t(2, 0);
559         rrt->put032t(3, 0);
560         nextpc = (pc + exts32(imm << 2, 18)) & as->lslr;
561         break;
562     case BRASL____:
563         rrt->put032t(PSLOT, (pc + 4) & as->lslr);
564         rrt->put032t(1, 0);
565         rrt->put032t(2, 0);
566         rrt->put032t(3, 0);
567         nextpc = (exts32(imm << 2, 18)) & as->lslr;
568         break;
569     case BI_______:
570         nextpc = rra->get032t(PSLOT) & as->lslr & 0xfffffffc;
571
572         // enable/disable interrupts by E/D function bits //
573         break;
574     case BISL_____:
575         rrt->put032t(PSLOT, (pc + 4) & as->lslr);
576         rrt->put032t(1, 0);
577         rrt->put032t(2, 0);
578         rrt->put032t(3, 0);
579         nextpc = rra->get032t(PSLOT) & as->lslr & 0xfffffffc;
580
581         // enable/disable interrupts by E/D function bits //
582         break;
583     case BRNZ_____:
584         if (rrt->get032t(PSLOT) != 0)
585             nextpc = (pc + exts32(imm << 2, 18)) & as->lslr;
586         break;
587     case BRZ______:
588         if (rrt->get032t(PSLOT) == 0)
589             nextpc = (pc + exts32(imm << 2, 18)) & as->lslr;
590         break;
591     case BRHNZ____:
592         if ((rrt->get032t(PSLOT) & 0x0000ffff) != 0)
593             nextpc = (pc + exts32(imm << 2, 18)) & as->lslr;
594         break;
595     case BRHZ_____:
596         if ((rrt->get032t(PSLOT) & 0x0000ffff) == 0)
597             nextpc = (pc + exts32(imm << 2, 18)) & as->lslr;
598         break;
599     case BIZ______:
600         if (rrt->get032t(PSLOT) == 0) {
601             nextpc = rra->get032t(PSLOT) & as->lslr & 0xfffffffc;
602
603             // enable/disable interrupts by E/D function bits //
604         }
605         break;
606     case BINZ_____:
607         if (rrt->get032t(PSLOT) != 0) {
608             nextpc = rra->get032t(PSLOT) & as->lslr & 0xfffffffc;
609
610             // enable/disable interrupts by E/D function bits //
611         }
612         break;
613     case BIHZ_____:
614         if ((rrt->get032t(PSLOT) & 0x0000ffff) == 0) {
615             nextpc = rra->get032t(PSLOT) & as->lslr & 0xfffffffc;
```

```
616
617             // enable/disable interrupts by E/D function bits //
618         }
619         break;
620     case BIHNZ____:
621         if ((rrt->get032t(PSLOT) & 0x0000ffff) != 0) {
622             nextpc = rra->get032t(PSLOT) & as->lslr & 0xfffffffc;
623
624             // enable/disable interrupts by E/D function bits //
625         }
626         break;
627
628         /********/
629     case HBR______:
630
631         // determine branch target address
632         // hinted branch address has been computed and written in 'ro'
633         targ = rra->get032t(PSLOT) & as->lslr & 0xfffffffc;
634         break;
635     case HBRA_____:
636         targ = exts32(imm << 2, 18) & as->lslr;
637         break;
638     case HBRR_____:
639         targ = (exts32(imm << 2, 18) + pc) & as->lslr;
640         break;
641
642         /********/
643     case FREST____:
644         for (int i = 0; i < NSLOT(uint_032t); i++)
645             rrt->putf32t(i, 1 / rra->getf32t(i));
646         break;
647     case FRSQEST__:
648         for (int i = 0; i < NSLOT(uint_032t); i++)
649             rrt->putf32t(i, 1 / sqrt(fabs(rra->getf32t(i))));
650         break;
651
652         /********/
653     case STOP_____:
654         inst->nextstate = SPE_STOP;
655         as->stop_reason = SR_STOP | (inst->ir & SR_SIGNALMASK);
656         break;
657     case STOPD____:
658         inst->nextstate = SPE_STOP;
659         as->stop_reason = SR_STOP | (inst->ir & SR_SIGNALMASK);
660         break;
661     case LNOP_____:
662     case SYNC_____:
663     case DSYNC____:
664         break;
665     case MFSPR____:
666         rrt->put064t(0, 0);
667         rrt->put064t(1, 0);
668         break;
669     case MTSPR____:
670         break;
671
672         /********/
673     case RDCH_____:
674     {
675         uint_032t data;
676         if (spe->chi->spureadch(inst->ca, &data)) {
677             inst->nextstate = SPE_STALL;
678             ss->stall_reason = ST_NOTRDCH + inst->ca;
679             break;
680         }
681         rrt->put032t(PSLOT, data);
682         rrt->put032t(1, 0);
683         rrt->put032t(2, 0);
684         rrt->put032t(3, 0);
685         break;
686     }
687     case WRCH_____:
688         if (spe->chi->spuwritech(inst->ca, rrt->get032t(PSLOT))) {
689             inst->nextstate = SPE_STALL;
690             ss->stall_reason = ST_NOTWRCH + inst->ca;
691         }
692         break;
```



```
693     case RCHCNT___:
694         rrt->put032t(PSLOT, spe->chi->spureadchcnt(inst->ca));
695         rrt->put032t(1, 0);
696         rrt->put032t(2, 0);
697         rrt->put032t(3, 0);
698         break;
699     default:
700         printf("!! %5x:", (uint) pc);
701         for (int i = 0; i < 4; i++)
702             printf(" %02x", (uint) ((inst->ir >> (24 - i * 8)) & 0xff));
703         printf("\n%s is not supported yet. Stop.\n", inst->getopname());
704         inst->nextstate = SPE_STOP;
705         as->stop_reason = SR_ERROR;
706         break;
707     }
708     if (chip->debug_mode == 2)
709         inst->exprinttail();
710     inst->targ = targ;
711     inst->nextpc = nextpc;
712 }
713
714 /*************************************************************************/
715 inline void Spu::executeeven(SpuInst *inst)
716 {
717     Register *rra = inst->rra;
718     Register *rrb = inst->rrb;
719     Register *rrc = inst->rrc;
720     Register *rrt = inst->rrt;
721     uint imm = inst->imm;
722
723     if (chip->debug_mode == 2)
724         inst->exprinthead();
725     switch (inst->op) {
726     case ILH______:
727         for (int i = 0; i < NSLOT(uint_016t); i++)
728             rrt->put016t(i, imm);
729         break;
730     case ILHU_____:
731         for (int i = 0; i < NSLOT(uint_032t); i++)
732             rrt->put032t(i, imm << 16);
733         break;
734     case IL_______:
735         for (int i = 0; i < NSLOT(uint_032t); i++)
736             rrt->put032t(i, exts32(imm, 16));
737         break;
738     case ILA______:
739         for (int i = 0; i < NSLOT(uint_032t); i++)
740             rrt->put032t(i, imm);
741         break;
742     case IOHL_____:
743         for (int i = 0; i < NSLOT(uint_032t); i++)
744             rrt->put032t(i, rrt->get032t(i) | imm);
745         break;
746
747         /********/
748     case AH_______:
749         for (int i = 0; i < NSLOT(uint_016t); i++)
750             rrt->put016t(i, rra->get016t(i) + rrb->get016t(i));
751         break;
752     case AHI______:
753         for (int i = 0; i < NSLOT(uint_016t); i++)
754             rrt->put016t(i, rra->get016t(i) + exts32(imm, 10));
755         break;
756     case A________:
757         for (int i = 0; i < NSLOT(uint_032t); i++)
758             rrt->put032t(i, rra->get032t(i) + rrb->get032t(i));
759         break;
760     case AI_______:
761         for (int i = 0; i < NSLOT(uint_032t); i++)
762             rrt->put032t(i, rra->get032t(i) + exts32(imm, 10));
763         break;
764     case SFH______:
765         for (int i = 0; i < NSLOT(uint_016t); i++)
766             rrt->put016t(i, rrb->get016t(i) - rra->get016t(i));
767         break;
768     case SFHI_____:
769         for (int i = 0; i < NSLOT(uint_016t); i++)
```

```
770             rrt->put016t(i, exts32(imm, 10) - rra->get016t(i));
771         break;
772     case SF_______:
773         for (int i = 0; i < NSLOT(uint_032t); i++)
774             rrt->put032t(i, rrb->get032t(i) - rra->get032t(i));
775         break;
776     case SFI______:
777         for (int i = 0; i < NSLOT(uint_032t); i++)
778             rrt->put032t(i, exts32(imm, 10) - rra->get032t(i));
779         break;
780     case ADDX_____:
781         for (int i = 0; i < NSLOT(uint_032t); i++)
782             rrt->put032t(i,
783                          rra->get032t(i) + rrb->get032t(i) +
784                          (rrt->get032t(i) & 0x1));
785         break;
786     case CG_______:
787     {
788         for (int i = 0; i < NSLOT(uint_032t); i++) {
789             uint_032t data = rra->get032t(i) + rrb->get032t(i);
790             data = (data < rra->get032t(i));
791             rrt->put032t(i, data);
792         } break;
793     }
794     case CGX______:
795     {
796         for (int i = 0; i < NSLOT(uint_032t); i++) {
797             uint_032t data = rra->get032t(i) + rrb->get032t(i);
798             data = ((data < rra->get032t(i)) ||
799                     ((data == 0xffffffff)
800                      && (rrt->get032t(i) & 0x1)));
801             rrt->put032t(i, data);
802         } break;
803     }
804     case SFX______:
805     {
806         for (int i = 0; i < NSLOT(uint_032t); i++) {
807             uint_032t data = rrb->get032t(i) - rra->get032t(i);
808             data += (rrt->get032t(i) & 0x1) - 1;
809             rrt->put032t(i, data);
810         } break;
811     }
812     case BG_______:
813     {
814         for (int i = 0; i < NSLOT(uint_032t); i++) {
815             uint_032t data = (rrb->get032t(i) >= rra->get032t(i));
816             rrt->put032t(i, data);
817         } break;
818     }
819     case BGX______:
820     {
821         for (int i = 0; i < NSLOT(uint_032t); i++) {
822             uint_032t data = (rrt->get032t(i) & 0x1) ?
823                 (rrb->get032t(i) >= rra->get032t(i)) :
824                 (rrb->get032t(i) > rra->get032t(i));
825             rrt->put032t(i, data);
826         } break;
827     }
828     case MPY______:
829         for (int i = 0; i < NSLOT(uint_032t); i++)
830             rrt->put032t(i,
831                          low16(rra->get032t(i)) * low16(rrb->get032t(i)));
832         break;
833     case MPYU_____:
834         for (int i = 0; i < NSLOT(uint_032t); i++)
835             rrt->put032t(i,
836                          ulow16(rra->get032t(i)) *
837                          ulow16(rrb->get032t(i)));
838         break;
839     case MPYI_____:
840         for (int i = 0; i < NSLOT(uint_032t); i++)
841             rrt->put032t(i, low16(rra->get032t(i)) * exts32(imm, 10));
842         break;
843     case MPYUI____:
844         for (int i = 0; i < NSLOT(uint_032t); i++)
845             rrt->put032t(i,
846                          ulow16(rra->get032t(i)) *
```



```
847                          ulow16(exts32(imm, 10)));
848         break;
849     case MPYA_____:
850         for (int i = 0; i < NSLOT(uint_032t); i++)
851             rrt->put032t(i,
852                          (low16(rra->get032t(i)) *
853                           low16(rrb->get032t(i)) + rrc->get032t(i)));
854         break;
855     case MPYH_____:
856         for (int i = 0; i < NSLOT(uint_032t); i++)
857             rrt->put032t(i,
858                          (high16(rra->get032t(i)) *
859                           ulow16(rrb->get032t(i))) << 16);
860         break;
861     case MPYS_____:
862     {
863         for (int i = 0; i < NSLOT(uint_032t); i++) {
864             uint_032t data =
865                 low16(rra->get032t(i)) * low16(rrb->get032t(i));
866             rrt->put032t(i, exts32(data >> 16, 16));
867         } break;
868     }
869     case MPYHH____:
870         for (int i = 0; i < NSLOT(uint_032t); i++)
871             rrt->put032t(i,
872                          high16(rra->get032t(i)) *
873                          high16(rrb->get032t(i)));
874         break;
875     case MPYHHA___:
876         for (int i = 0; i < NSLOT(uint_032t); i++)
877             rrt->put032t(i,
878                          high16(rra->get032t(i)) *
879                          high16(rrb->get032t(i)) + rrt->get032t(i));
880         break;
881     case MPYHHU___:
882         for (int i = 0; i < NSLOT(uint_032t); i++)
883             rrt->put032t(i,
884                          (uhigh16(rra->get032t(i)) *
885                           uhigh16(rrb->get032t(i))));
886         break;
887     case MPYHHAU__:
888         for (int i = 0; i < NSLOT(uint_032t); i++)
889             rrt->put032t(i,
890                          (uhigh16(rra->get032t(i)) *
891                           uhigh16(rrb->get032t(i)) + rrt->get032t(i)));
892         break;
893     case CLZ______:
894     {
895         uint_032t t;
896         for (int i = 0; i < NSLOT(uint_032t); i++) {
897             uint_032t u = rra->get032t(i);
898             for (t = 0; t < 32; t++)
899                 if (u & (0x80000000 >> t))
900                     break;
901             rrt->put032t(i, t);
902         }
903         break;
904     }
905     case CNTB_____:
906     {
907         for (int i = 0; i < NSLOT(uint_008t); i++) {
908             uint_008t b = rra->get008t(i);
909             uint_008t c = 0;
910             for (uint m = 0; m < 8; m++) {
911                 c += ((b & (0x80 >> m)) != 0);
912             }
913             rrt->put008t(i, c);
914         }
915         break;
916     }
917     case AVGB_____:
918     {
919         for (int i = 0; i < NSLOT(uint_008t); i++) {
920             uint_016t data =
921                 (rra->get008t(i) + rrb->get008t(i) + 1) >> 1;
922             rrt->put008t(i, (uint_008t) data);
923         } break;
```

```
924         }
925         case ABSDB____:
926         {
927             for (int i = 0; i < NSLOT(uint_008t); i++)
928                 rrt->put008t(i,
929                              ((rrb->get008t(i) >
930                                rra->get008t(i)) ? rrb->get008t(i) -
931                               rra->get008t(i) : rra->get008t(i) -
932                               rrb->get008t(i)));
933             break;
934         }
935         case SUMB_____:
936         {
937             for (int i = 0; i < NSLOT(uint_032t); i++) {
938                 uint_016t ra_sum = 0, rb_sum = 0;
939                 for (uint j = 0; j < 4; j++) {
940                     ra_sum += rrb->get008t(i * 4 + j);
941                     rb_sum += rra->get008t(i * 4 + j);
942                 }
943                 rrt->put016t(i * 2, ra_sum);
944                 rrt->put016t(i * 2 + 1, rb_sum);
945             }
946             break;
947         }
948         case XSBH_____:
949         {
950             for (int i = 0; i < NSLOT(uint_016t); i++) {
951                 uint_016t data =
952                     ulow16(exts32(rra->get008t(i * 2 + 1), 8));
953                 rrt->put016t(i, data);
954             } break;
955         }
956         case XSHW_____:
957             for (int i = 0; i < NSLOT(uint_032t); i++)
958                 rrt->put032t(i, exts32(rra->get032t(i), 16));
959             break;
960         case XSWD_____:
961         {
962             for (int i = 0; i < NSLOT(uint_064t); i++) {
963                 uint_064t data = rra->get032t(i * 2 + 1);
964                 if (data & 0x80000000)
965                     data = data | 0xffffffff00000000ull;
966                 rrt->put064t(i, data);
967             }
968             break;
969         }
970         case AND______:
971             for (int i = 0; i < NSLOT(uint_032t); i++)
972                 rrt->put032t(i, rra->get032t(i) & rrb->get032t(i));
973             break;
974         case ANDC_____:
975             for (int i = 0; i < NSLOT(uint_032t); i++)
976                 rrt->put032t(i, rra->get032t(i) & ~rrb->get032t(i));
977             break;
978         case ANDBI____:
979             for (int i = 0; i < NSLOT(uint_008t); i++)
980                 rrt->put008t(i, rra->get008t(i) & imm);
981             break;
982         case ANDHI____:
983             for (int i = 0; i < NSLOT(uint_016t); i++)
984                 rrt->put016t(i, rra->get016t(i) & exts32(imm, 10));
985             break;
986         case ANDI_____:
987             for (int i = 0; i < NSLOT(uint_032t); i++)
988                 rrt->put032t(i, rra->get032t(i) & exts32(imm, 10));
989             break;
990         case OR_______:
991             for (int i = 0; i < NSLOT(uint_032t); i++)
992                 rrt->put032t(i, rra->get032t(i) | rrb->get032t(i));
993             break;
994         case ORC______:
995             for (int i = 0; i < NSLOT(uint_032t); i++)
996                 rrt->put032t(i, rra->get032t(i) | ~rrb->get032t(i));
997             break;
998         case ORBI_____:
999             for (int i = 0; i < NSLOT(uint_008t); i++)
1000                rrt->put008t(i, rra->get008t(i) | imm);
```



```
1001            break;
1002        case ORHI_____:
1003            for (int i = 0; i < NSLOT(uint_016t); i++)
1004                rrt->put016t(i, rra->get016t(i) | exts32(imm, 10));
1005            break;
1006        case ORI______:
1007            for (int i = 0; i < NSLOT(uint_032t); i++)
1008                rrt->put032t(i, rra->get032t(i) | exts32(imm, 10));
1009            break;
1010        case XOR______:
1011            for (int i = 0; i < NSLOT(uint_032t); i++)
1012                rrt->put032t(i, rra->get032t(i) ^ rrb->get032t(i));
1013            break;
1014        case XORBI____:
1015            for (int i = 0; i < NSLOT(uint_008t); i++)
1016                rrt->put008t(i, rra->get008t(i) ^ imm);
1017            break;
1018        case XORHI____:
1019            for (int i = 0; i < NSLOT(uint_016t); i++)
1020                rrt->put016t(i, rra->get016t(i) ^ exts32(imm, 10));
1021            break;
1022        case XORI_____:
1023            for (int i = 0; i < NSLOT(uint_032t); i++)
1024                rrt->put032t(i, rra->get032t(i) ^ exts32(imm, 10));
1025            break;
1026        case NAND_____:
1027            for (int i = 0; i < NSLOT(uint_032t); i++)
1028                rrt->put032t(i, ~(rra->get032t(i) & rrb->get032t(i)));
1029            break;
1030        case NOR______:
1031            for (int i = 0; i < NSLOT(uint_032t); i++)
1032                rrt->put032t(i, ~(rra->get032t(i) | rrb->get032t(i)));
1033            break;
1034        case EQV______:
1035            for (int i = 0; i < NSLOT(uint_032t); i++)
1036                rrt->put032t(i, rra->get032t(i) ^ ~rrb->get032t(i));
1037            break;
1038        case SELB_____:
1039            for (int i = 0; i < NSLOT(uint_032t); i++)
1040                rrt->put032t(i, ((rrc->get032t(i) & rrb->get032t(i)) |
1041                                 (~rrc->get032t(i) & rra->get032t(i))));
1042            break;
1043
1044            /********/
1045        case SHLH_____:
1046        {
1047            for (int i = 0; i < NSLOT(uint_016t); i++) {
1048                int s = rrb->get016t(i) & 0x1f;
1049                rrt->put016t(i, (s < 16) ? rra->get016t(i) << s : 0);
1050            } break;
1051        }
1052        case SHLHI____:
1053        {
1054            int s = imm & 0x1f;
1055            for (int i = 0; i < NSLOT(uint_016t); i++)
1056                rrt->put016t(i, (s < 16) ? rra->get016t(i) << s : 0);
1057            break;
1058        }
1059        case SHL______:
1060        {
1061            for (int i = 0; i < NSLOT(uint_032t); i++) {
1062                int s = rrb->get032t(i) & 0x3f;
1063                rrt->put032t(i, (s < 32) ? rra->get032t(i) << s : 0);
1064            } break;
1065        }
1066        case SHLI_____:
1067        {
1068            int s = imm & 0x3f;
1069            for (int i = 0; i < NSLOT(uint_032t); i++)
1070                rrt->put032t(i, (s < 32) ? rra->get032t(i) << s : 0);
1071            break;
1072        }
1073        case ROTH_____:
1074        {
1075            for (int i = 0; i < NSLOT(uint_016t); i++) {
1076                int s = rrb->get016t(i) & 0xf;
1077                rrt->put016t(i, ((rra->get016t(i) << s) |
```

```
1078                            (rra->get016t(i) >> (16 - s))));
1079          } break;
1080      }
1081      case ROTHI____:
1082      {
1083          int s = imm & 0xf;
1084          for (int i = 0; i < NSLOT(uint_016t); i++) {
1085              rrt->put016t(i, ((rra->get016t(i) << s) |
1086                              (rra->get016t(i) >> (16 - s))));
1087          } break;
1088      }
1089      case ROT______:
1090      {
1091          for (int i = 0; i < NSLOT(uint_032t); i++) {
1092              int s = rrb->get032t(i) & 0x1f;
1093              rrt->put032t(i, ((rra->get032t(i) << s) |
1094                              (rra->get032t(i) >> (32 - s))));
1095          } break;
1096      }
1097      case ROTI_____:
1098      {
1099          int s = imm & 0x1f;
1100          for (int i = 0; i < NSLOT(uint_032t); i++) {
1101              rrt->put032t(i, ((rra->get032t(i) << s) |
1102                              (rra->get032t(i) >> (32 - s))));
1103          } break;
1104      }
1105      case ROTHM____:
1106      {
1107          for (int i = 0; i < NSLOT(uint_016t); i++) {
1108              int s = (0 - rrb->get016t(i)) & 0x1f;
1109              rrt->put016t(i, (s < 16) ? rra->get016t(i) >> s : 0);
1110          } break;
1111      }
1112      case ROTHMI___:
1113      {
1114          int s = (0 - exts32(imm, 7)) & 0x1f;
1115          for (int i = 0; i < NSLOT(uint_016t); i++)
1116              rrt->put016t(i, (s < 16) ? rra->get016t(i) >> s : 0);
1117          break;
1118      }
1119      case ROTM_____:
1120      {
1121          for (int i = 0; i < NSLOT(uint_032t); i++) {
1122              int s = (0 - rrb->get032t(i)) & 0x3f;
1123              rrt->put032t(i, (s < 32) ? rra->get032t(i) >> s : 0);
1124          } break;
1125      }
1126      case ROTMI____:
1127      {
1128          int s = (0 - exts32(imm, 7)) & 0x3f;
1129          for (int i = 0; i < NSLOT(uint_032t); i++)
1130              rrt->put032t(i, (s < 32) ? rra->get032t(i) >> s : 0);
1131          break;
1132      }
1133      case ROTMAH___:
1134      {
1135          for (int i = 0; i < NSLOT(uint_016t); i++) {
1136              int s = (0 - rrb->get016t(i)) & 0x1f;
1137              s = (s < 16) ? s : 15;
1138              rrt->put016t(i, (uint_016t) exts32(rra->get016t(i) >> s,
1139                                                 (16 - s)));
1140          } break;
1141      }
1142      case ROTMAHI__:
1143      {
1144          int s = (0 - exts32(imm, 7)) & 0x1f;
1145          s = (s < 16) ? s : 15;
1146          for (int i = 0; i < NSLOT(uint_016t); i++)
1147              rrt->put016t(i, (uint_016t) exts32(rra->get016t(i) >> s,
1148                                                 (16 - s)));
1149          break;
1150      }
1151      case ROTMA____:
1152      {
1153          for (int i = 0; i < NSLOT(uint_032t); i++) {
1154              int s = (0 - rrb->get032t(i)) & 0x3f;
```



```
1155              s = (s < 32) ? s : 31;
1156              rrt->put032t(i, exts32(rra->get032t(i) >> s, (32 - s)));
1157          } break;
1158      }
1159      case ROTMAI___:
1160      {
1161          int s = (0 - exts32(imm, 7)) & 0x3f;
1162          s = (s < 32) ? s : 31;
1163          for (int i = 0; i < NSLOT(uint_032t); i++)
1164              rrt->put032t(i, exts32(rra->get032t(i) >> s, (32 - s)));
1165          break;
1166      }
1167
1168      /********/
1169      case HEQ______:
1170          if (rra->get032t(PSLOT) == rrb->get032t(PSLOT)) {
1171              inst->nextstate = SPE_STOP;
1172              as->stop_reason = SR_HALT;
1173          }
1174          break;
1175      case HEQI_____:
1176          if (rra->get032t(PSLOT) == exts32(imm, 10)) {
1177              inst->nextstate = SPE_STOP;
1178              as->stop_reason = SR_HALT;
1179          }
1180          break;
1181      case HGT______:
1182          if ((int) rra->get032t(PSLOT) > (int) rrb->get032t(PSLOT)) {
1183              inst->nextstate = SPE_STOP;
1184              as->stop_reason = SR_HALT;
1185          }
1186          break;
1187      case HGTI_____:
1188          if ((int) rra->get032t(PSLOT) > (int) exts32(imm, 10)) {
1189              inst->nextstate = SPE_STOP;
1190              as->stop_reason = SR_HALT;
1191          }
1192          break;
1193      case HLGT_____:
1194          if (rra->get032t(PSLOT) > rrb->get032t(PSLOT)) {
1195              inst->nextstate = SPE_STOP;
1196              as->stop_reason = SR_HALT;
1197          }
1198          break;
1199      case HLGTI____:
1200          if (rra->get032t(PSLOT) > exts32(imm, 10)) {
1201              inst->nextstate = SPE_STOP;
1202              as->stop_reason = SR_HALT;
1203          }
1204          break;
1205      case CEQB_____:
1206          for (int i = 0; i < NSLOT(uint_008t); i++)
1207              rrt->put008t(i, (rra->get008t(i) == rrb->get008t(i))
1208                              ? 0xff : 0x00);
1209          break;
1210      case CEQBI____:
1211          for (int i = 0; i < NSLOT(uint_008t); i++)
1212              rrt->put008t(i, (rra->get008t(i) == (imm & 0xff))
1213                              ? 0xff : 0x00);
1214          break;
1215      case CEQH_____:
1216          for (int i = 0; i < NSLOT(uint_016t); i++)
1217              rrt->put016t(i, (rra->get016t(i) == rrb->get016t(i))
1218                              ? 0xffff : 0x0000);
1219          break;
1220      case CEQHI____:
1221          for (int i = 0; i < NSLOT(uint_016t); i++)
1222              rrt->put016t(i, (rra->get016t(i) == ulow16(exts32(imm, 10)))
1223                              ? 0xffff : 0x0000);
1224          break;
1225      case CEQ______:
1226          for (int i = 0; i < NSLOT(uint_032t); i++)
1227              rrt->put032t(i, (rra->get032t(i) == rrb->get032t(i))
1228                              ? 0xffffffff : 0x00000000);
1229          break;
1230      case CEQI_____:
1231          for (int i = 0; i < NSLOT(uint_032t); i++)
```

```
1232              rrt->put032t(i, (rra->get032t(i) == exts32(imm, 10))
1233                           ? 0xffffffff : 0x00000000);
1234          break;
1235      case CGTB_____:
1236          for (int i = 0; i < NSLOT(uint_008t); i++)
1237              rrt->put008t(i, ((sint_008t) rra->get008t(i) >
1238                               (sint_008t) rrb->get008t(i))
1239                           ? 0xff : 0x00);
1240          break;
1241      case CGTBI____:
1242          for (int i = 0; i < NSLOT(uint_008t); i++)
1243              rrt->put008t(i, ((sint_008t) rra->get008t(i) >
1244                               (sint_008t) (imm & 0xff))  ? 0xff : 0x00);
1245          break;
1246      case CGTH_____:
1247          for (int i = 0; i < NSLOT(uint_016t); i++)
1248              rrt->put016t(i, ((sint_016t) rra->get016t(i) >
1249                               (sint_016t) rrb->get016t(i))
1250                           ? 0xffff : 0x0000);
1251          break;
1252      case CGTHI____:
1253          for (int i = 0; i < NSLOT(uint_016t); i++)
1254              rrt->put016t(i, ((sint_016t) rra->get016t(i) >
1255                               (sint_016t) exts32(imm, 10))
1256                           ? 0xffff : 0x0000);
1257          break;
1258      case CGT______:
1259          for (int i = 0; i < NSLOT(uint_032t); i++)
1260              rrt->put032t(i, ((sint_032t) rra->get032t(i) >
1261                               (sint_032t) rrb->get032t(i))
1262                           ? 0xffffffff : 0x00000000);
1263          break;
1264      case CGTI_____:
1265          for (int i = 0; i < NSLOT(uint_032t); i++)
1266              rrt->put032t(i, ((sint_032t) rra->get032t(i) >
1267                               (sint_032t) exts32(imm, 10))
1268                           ? 0xffffffff : 0x00000000);
1269          break;
1270      case CLGTB____:
1271          for (int i = 0; i < NSLOT(uint_008t); i++)
1272              rrt->put008t(i, (rra->get008t(i) > rrb->get008t(i))
1273                           ? 0xff : 0x00);
1274          break;
1275      case CLGTBI___:
1276          for (int i = 0; i < NSLOT(uint_008t); i++)
1277              rrt->put008t(i, (rra->get008t(i) > (imm & 0xff))
1278                           ? 0xff : 0x00);
1279          break;
1280      case CLGTH____:
1281          for (int i = 0; i < NSLOT(uint_016t); i++)
1282              rrt->put016t(i, (rra->get016t(i) > rrb->get016t(i))
1283                           ? 0xffff : 0x0000);
1284          break;
1285      case CLGTHI___:
1286          for (int i = 0; i < NSLOT(uint_016t); i++)
1287              rrt->put016t(i, (rra->get016t(i) > ulow16(exts32(imm, 10)))
1288                           ? 0xffff : 0x0000);
1289          break;
1290      case CLGT_____:
1291          for (int i = 0; i < NSLOT(uint_032t); i++)
1292              rrt->put032t(i, (rra->get032t(i) > rrb->get032t(i))
1293                           ? 0xffffffff : 0x00000000);
1294          break;
1295      case CLGTI____:
1296          for (int i = 0; i < NSLOT(uint_032t); i++)
1297              rrt->put032t(i, (rra->get032t(i) > exts32(imm, 10))
1298                           ? 0xffffffff : 0x00000000);
1299          break;
1300
1301          /********/
1302      case FA_______:
1303          for (int i = 0; i < NSLOT(uint_032t); i++)
1304              rrt->putf32t(i, rra->getf32t(i) + rrb->getf32t(i));
1305          break;
1306      case DFA______:
1307          for (int i = 0; i < NSLOT(uint_064t); i++)
1308              rrt->putf64t(i, rra->getf64t(i) + rrb->getf64t(i));
```



```
1309          break;
1310      case FS_______:
1311          for (int i = 0; i < NSLOT(uint_032t); i++)
1312              rrt->putf32t(i, rra->getf32t(i) - rrb->getf32t(i));
1313          break;
1314      case DFS______:
1315          for (int i = 0; i < NSLOT(uint_064t); i++)
1316              rrt->putf64t(i, rra->getf64t(i) - rrb->getf64t(i));
1317          break;
1318      case FM_______:
1319          for (int i = 0; i < NSLOT(uint_032t); i++)
1320              rrt->putf32t(i, rra->getf32t(i) * rrb->getf32t(i));
1321          break;
1322      case DFM______:
1323          for (int i = 0; i < NSLOT(uint_064t); i++)
1324              rrt->putf64t(i, rra->getf64t(i) * rrb->getf64t(i));
1325          break;
1326      case FMA______:
1327          for (int i = 0; i < NSLOT(uint_032t); i++)
1328              rrt->putf32t(i,
1329                           (rra->getf32t(i) * rrb->getf32t(i) +
1330                            rrc->getf32t(i)));
1331          break;
1332      case DFMA_____:
1333          for (int i = 0; i < NSLOT(uint_064t); i++)
1334              rrt->putf64t(i,
1335                           (rra->getf64t(i) * rrb->getf64t(i) +
1336                            rrt->getf64t(i)));
1337          break;
1338      case FNMS_____:
1339          for (int i = 0; i < NSLOT(uint_032t); i++)
1340              rrt->putf32t(i,
1341                           (rrc->getf32t(i) -
1342                            rra->getf32t(i) * rrb->getf32t(i)));
1343          break;
1344      case DFNMS____:
1345          for (int i = 0; i < NSLOT(uint_064t); i++)
1346              rrt->putf64t(i,
1347                           (rrt->getf64t(i) -
1348                            rra->getf64t(i) * rrb->getf64t(i)));
1349          break;
1350      case FMS______:
1351          for (int i = 0; i < NSLOT(uint_032t); i++)
1352              rrt->putf32t(i,
1353                           (rra->getf32t(i) * rrb->getf32t(i) -
1354                            rrc->getf32t(i)));
1355          break;
1356      case DFMS_____:
1357          for (int i = 0; i < NSLOT(uint_064t); i++)
1358              rrt->putf64t(i,
1359                           (rra->getf64t(i) * rrb->getf64t(i) -
1360                            rrt->getf64t(i)));
1361          break;
1362      case DFNMA____:
1363          for (int i = 0; i < NSLOT(uint_064t); i++)
1364              rrt->putf64t(i,
1365                           (-rra->getf64t(i) * rrb->getf64t(i) -
1366                            rrt->getf64t(i)));
1367          break;
1368      case FI_______:
1369          for (int i = 0; i < NSLOT(uint_032t); i++)
1370              rrt->putf32t(i, rrb->getf32t(i));
1371          break;
1372      case CSFLT____:
1373      {
1374          for (int i = 0; i < NSLOT(uint_032t); i++) {
1375              float fdata = (float) ((sint_032t) rra->get032t(i));
1376              for (int j = 0; j < 155 - (sint_032t) imm; j++)
1377                  fdata /= 2;
1378              rrt->putf32t(i, fdata);
1379          } break;
1380      }
1381      case CFLTS____:
1382      {
1383          for (int i = 0; i < NSLOT(uint_032t); i++) {
1384              float fdata = rra->getf32t(i);
1385              for (int j = 0; j < 173 - (sint_032t) imm; j++)
```

```
1386                 fdata *= 2;
1387             if (fdata > 0x7fffffffll)
1388                 rrt->put032t(i, 0x7fffffff);
1389
1390             else if (fdata < -0x8000000011)
1391                 rrt->put032t(i, 0x80000000);
1392
1393             else
1394                 rrt->put032t(i, (uint_032t) ((sint_032t) fdata));
1395         }
1396         break;
1397     }
1398     case CUFLT____:
1399     {
1400         for (int i = 0; i < NSLOT(uint_032t); i++) {
1401             float fdata = (float) rra->get032t(i);
1402             for (int j = 0; j < 155 - (sint_032t) imm; j++)
1403                 fdata /= 2;
1404             rrt->putf32t(i, fdata);
1405         } break;
1406     }
1407     case CFLTU____:
1408     {
1409         for (int i = 0; i < NSLOT(uint_032t); i++) {
1410             float fdata = rra->getf32t(i);
1411             for (int j = 0; j < 173 - (sint_032t) imm; j++)
1412                 fdata *= 2;
1413             if (fdata > 0xffffffffll)
1414                 rrt->put032t(i, 0xffffffff);
1415
1416             else if (fdata < 0)
1417                 rrt->put032t(i, 0);
1418
1419             else
1420                 rrt->put032t(i, (uint_032t) fdata);
1421         }
1422         break;
1423     }
1424     case FRDS_____:
1425         for (int i = 0; i < NSLOT(uint_064t); i++)
1426             rrt->putf32t(i * 2, (float) rra->getf64t(i));
1427         break;
1428     case FESD_____:
1429         for (int i = 0; i < NSLOT(uint_064t); i++)
1430             rrt->putf64t(i, (double) rra->getf32t(i * 2));
1431         break;
1432     case FCEQ_____:
1433         for (int i = 0; i < NSLOT(uint_032t); i++)
1434             rrt->put032t(i, (rra->getf32t(i) == rrb->getf32t(i))
1435                          ? 0xffffffff : 0x00000000);
1436         break;
1437     case FCMEQ____:
1438         for (int i = 0; i < NSLOT(uint_032t); i++)
1439             rrt->put032t(i,
1440                          (fabs(rra->getf32t(i)) ==
1441                           fabs(rrb->
1442                                getf32t(i)))  ? 0xffffffff : 0x00000000);
1443         break;
1444     case FCGT_____:
1445         for (int i = 0; i < NSLOT(uint_032t); i++)
1446             rrt->put032t(i, (rra->getf32t(i) > rrb->getf32t(i))
1447                          ? 0xffffffff : 0x00000000);
1448         break;
1449     case FCMGT____:
1450         for (int i = 0; i < NSLOT(uint_032t); i++)
1451             rrt->put032t(i,
1452                          (fabs(rra->getf32t(i)) >
1453                           fabs(rrb->
1454                                getf32t(i)))  ? 0xffffffff : 0x00000000);
1455         break;
1456     case FSCRWR___:
1457         as->fpscr->write(rra->get128t());
1458         break;
1459     case FSCRRD___:
1460         rrt->put128t(as->fpscr->read());
1461         break;
1462
```



```
1463         /********/
1464     case NOP______:
1465         break;
1466     default:
1467         printf("!! %5x:", (uint) inst->pc);
1468         for (int i = 0; i < 4; i++)
1469             printf(" %02x", (uint) ((inst->ir >> (24 - i * 8)) & 0xff));
1470         printf("\n%s is not supported yet. Stop.\n", inst->getopname());
1471         inst->nextstate = SPE_STOP;
1472         as->stop_reason = SR_ERROR;
1473         break;
1474     }
1475     if (chip->debug_mode == 2)
1476         inst->exprinttail();
1477 }
1478
1479 /*************************************************************************/
1480 LocalStore::LocalStore(ullint size)
1481 {
1482     int n = size / sizeof(uint_032t);
1483     mem = new uint_032t[n];
1484 }
1485
1486 /*************************************************************************/
1487 LocalStore::~LocalStore()
1488 {
1489     delete[] mem;
1490 }
1491
1492 /*************************************************************************/
1493 void LocalStore::read1b(uint_032t addr, uint_008t * data)
1494 {
1495     *data = (mem[addr / sizeof(uint_032t)] >>
1496              (24 - (addr % sizeof(uint_032t)) * 8)) & 0xff;
1497 }
1498
1499 /*************************************************************************/
1500 void LocalStore::read4b(uint_032t addr, uint_032t *data)
1501 {
1502     *data = mem[addr / sizeof(uint_032t)];
1503 }
1504
1505 /*************************************************************************/
1506 void LocalStore::readnb(uint_032t addr, int size, uint_032t data[])
1507 {
1508     int i, j, index;
1509     uint_032t temp = 0;
1510     uint_008t c;
1511
1512     index = 0;
1513     for (i = 0; i < size; i += sizeof(uint_032t)) {
1514         read4b(addr + i, &data[index]);
1515         index++;
1516     }
1517     for (j = 0; i + j < size; j++) {
1518         read1b(addr + i + j, &c);
1519         temp = (temp << 8) || c;
1520     }
1521     if (j != 0)
1522         data[index] = temp << (32 - j * 8);
1523 }
1524
1525 /*************************************************************************/
1526 void LocalStore::write1b(uint_032t addr, uint_008t data)
1527 {
1528     int offset = (24 - (addr % sizeof(uint_032t)) * 8);
1529     uint_032t ins = (uint_032t) data << offset;
1530     uint_032t mask = 0xff << offset;
1531     mem[addr / sizeof(uint_032t)] =
1532         ins | (mem[addr / sizeof(uint_032t)] & ~mask);
1533 }
1534
1535 /*************************************************************************/
1536 void LocalStore::write4b(uint_032t addr, uint_032t data)
1537 {
1538     mem[addr / sizeof(uint_032t)] = data;
1539 }
```

```
1540
1541  /*****************************************************************/
1542  void LocalStore::writenb(uint_032t addr, int size, uint_032t data[])
1543  {
1544      int i, j, index;
1545
1546      index = 0;
1547      for (i = 0; i < size; i += sizeof(uint_032t)) {
1548          write4b(addr + i, data[index]);
1549          index++;
1550      }
1551      for (j = 0; i + j < size; j++)
1552          write1b(addr + i + j, (data[index] >> (24 - j * 8)) & 0xff);
1553  }
1554
1555  /*****************************************************************/
1556  void LocalStore::print()
1557  {
1558      uint_032t data;
1559      uint_032t check;
1560      uint_032t off;
1561      uint_032t nulstart = 0;
1562      char buf[256];
1563      char temp[10];
1564      printf("[LOCAL STORE(SIZE: %d)]\n", (int)LS_SIZE);
1565      for (off = 0; off < LS_SIZE; off += 16 * sizeof(uint_032t)) {
1566          buf[0] = '\0';
1567          check = 0;
1568          for (uint i = 0; i < 16; i++) {
1569              read4b(off + i * sizeof(uint_032t), &data);
1570              sprintf(temp, "%08x ", (uint) data);
1571              strcat(buf, temp);
1572              check = check | data;
1573          }
1574          if (check == 0) {
1575              if (nulstart == 0)
1576                  nulstart = off;
1577          } else {
1578              if (nulstart != 0) {
1579                  printf("\n%8x..%8x: null\n\n", (uint) nulstart,
1580                         (uint) off - 1);
1581                  nulstart = 0;
1582              }
1583              printf("%8x: %s\n", (uint) off, buf);
1584          }
1585      }
1586      if (nulstart != 0)
1587          printf("\n%8x..%8x: null\n\n", (uint) nulstart,
1588                 (uint)(off - sizeof(uint_032t)));
1589      return;
1590  }
1591
1592  /*****************************************************************/
1593  Register::Register(Chip * chip)
1594  {
1595      this->chip = chip;
1596      for (uint i = 0; i < NSLOT(uint_064t); i++)
1597          reg.i064t[i] = 0;
1598  }
1599
1600  /*****************************************************************/
1601  void Register::print()
1602  {
1603      printf("0x");
1604      for (uint j = 0; j < NSLOT(uint_032t); j++)
1605          printf("%08x", (uint) get032t(j));
1606      printf("\n");
1607  }
1608
1609  /*****************************************************************/
1610  uint_008t Register::get008t(int offset)
1611  {
1612      return reg.i008t[slot(offset, NSLOT(uint_008t))];
1613  }
1614
1615  /*****************************************************************/
1616  uint_016t Register::get016t(int offset)
```



```
1617  {
1618      return reg.i016t[slot(offset, NSLOT(uint_016t))];
1619  }
1620
1621  /*****************************************************************/
1622  uint_032t Register::get032t(int offset)
1623  {
1624      return reg.i032t[slot(offset, NSLOT(uint_032t))];
1625  }
1626
1627  /*****************************************************************/
1628  uint_064t Register::get064t(int offset)
1629  {
1630      return reg.i064t[slot(offset, NSLOT(uint_064t))];
1631  }
1632
1633  /*****************************************************************/
1634  uint_128t Register::get128t()
1635  {
1636      return *(uint_128t *) & reg;
1637  }
1638
1639  /*****************************************************************/
1640  flot_032t Register::getf32t(int offset)
1641  {
1642      return reg.f032t[slot(offset, NSLOT(uint_032t))];
1643  }
1644
1645  /*****************************************************************/
1646  flot_064t Register::getf64t(int offset)
1647  {
1648      return reg.f064t[slot(offset, NSLOT(uint_064t))];
1649  }
1650
1651  /*****************************************************************/
1652  void Register::put008t(int offset, uint_008t data)
1653  {
1654      reg.i008t[slot(offset, NSLOT(uint_008t))] = data;
1655  }
1656
1657  /*****************************************************************/
1658  void Register::put016t(int offset, uint_016t data)
1659  {
1660      reg.i016t[slot(offset, NSLOT(uint_016t))] = data;
1661  }
1662
1663  /*****************************************************************/
1664  void Register::put032t(int offset, uint_032t data)
1665  {
1666      reg.i032t[slot(offset, NSLOT(uint_032t))] = data;
1667  }
1668
1669  /*****************************************************************/
1670  void Register::put064t(int offset, uint_064t data)
1671  {
1672      reg.i064t[slot(offset, NSLOT(uint_064t))] = data;
1673  }
1674
1675  /*****************************************************************/
1676  void Register::put128t(uint_128t data)
1677  {
1678      reg = *(regdata_t *) & data;
1679  }
1680
1681  /*****************************************************************/
1682  void Register::putf32t(int offset, flot_032t data)
1683  {
1684      reg.f032t[slot(offset, NSLOT(uint_032t))] = data;
1685  }
1686
1687  /*****************************************************************/
1688  void Register::putf64t(int offset, flot_064t data)
1689  {
1690      reg.f064t[slot(offset, NSLOT(uint_064t))] = data;
1691  }
1692
1693  /*****************************************************************/
```

```
1694  Fpscr::Fpscr(Chip *)
1695  {
1696      this->chip = chip;
1697      // enable = 00000f07 00003f07 00003f07 00000f07
1698      for (uint i = 0; i < NSLOT(uint_064t); i++) {
1699          reg.i064t[i] = 0;
1700          enable.i032t[i * 3] = 0x00000f07;
1701          enable.i032t[i + 1] = 0x00003f07;
1702      }
1703  }
1704
1705  /************************************************************************/
1706  uint_128t Fpscr::read()
1707  {
1708      regdata_t data;
1709      for (uint i = 0; i < NSLOT(uint_064t); i++)
1710          data.i064t[i] = reg.i064t[i] & enable.i064t[i];
1711      return *(uint_128t *) & data;
1712  }
1713
1714  /************************************************************************/
1715  void Fpscr::write(uint_128t data)
1716  {
1717      regdata_t temp = *(regdata_t *) & data;
1718      for (uint i = 0; i < NSLOT(uint_064t); i++)
1719          reg.i064t[i] = temp.i064t[i];
1720  }
1721
1722  /************************************************************************/
1723  void Fpscr::bitset(int bit)
1724  {
1725      uint_064t temp = 1 << (63 - (bit % 64));
1726      int index = slot(bit / 64, NSLOT(uint_064t));
1727      reg.i064t[index] = reg.i064t[index] | temp;
1728  }
1729
1730  /************************************************************************/
1731  void Fpscr::twobitwrite(int bit, int data)
1732  {
1733      uint_064t temp = (data & 0x3) << (62 - (bit % 64));
1734      uint_064t mask = 0x3 << (62 - (bit % 64));
1735      int index = slot(bit / 64, NSLOT(uint_064t));
1736      reg.i064t[index] = temp | (~mask & reg.i064t[index]);
1737  }
1738
1739  /************************************************************************/
1740  SpeArchstate::SpeArchstate(Chip * chip)
1741  {
1742      reg = new Register *[NREG];
1743      for (int i = 0; i < NREG; i++)
1744          reg[i] = new Register(chip);
1745      fpscr = new Fpscr(chip);
1746
1747      this->chip = chip;
1748      pc = 0;
1749      lslr = LS_SIZE - 1;
1750      state = SPE_RUN;
1751  }
1752
1753  /************************************************************************/
1754  SpeArchstate::~SpeArchstate()
1755  {
1756      pc = 0;
1757      delete[]reg;
1758      reg = NULL;
1759      state = SPE_STOP;
1760  }
1761
1762  /************************************************************************/
1763  SpeSimstate::SpeSimstate()
1764  {
1765      instcnt = 0;
1766      stall_reason = 0;
1767  }
1768
1769  /************************************************************************/
1770  Spe::Spe(Chip * chip, int id)
```



```
1771  {
1772      this->chip = chip;
1773      chip->spe[id - 1] = this;
1774      ls = new LocalStore(LS_SIZE);
1775      chi = new ChannelInterface();
1776      as = new SpeArchstate(chip);
1777      ss = new SpeSimstate();
1778      spu = new Spu(chip, this);
1779      mfc = new Mfc(chip, this);
1780      this->unitid = id;
1781  }
1782
1783  /************************************************************************/
1784  Spe::~Spe()
1785  {
1786      delete ls;
1787      ls = NULL;
1788      delete chi;
1789      chi = NULL;
1790      delete as;
1791      as = NULL;
1792      delete ss;
1793      ss = NULL;
1794      delete spu;
1795      spu = NULL;
1796      delete mfc;
1797      mfc = NULL;
1798  }
1799
1800  /** printf impremention ***************************************************/
1801  /************************************************************************/
1802  enum {
1803      SPEFUNC_PUTCHAR = 24,
1804      SPEFUNC_PUTS = 25,
1805      SPEFUNC_PRINTF = 37,
1806
1807      AL_032 = 4,
1808      AL_064 = 8,
1809      AL_F64 = 80,
1810      AL_NON = 0,
1811      AL_STR = -1,
1812      DUMMY = 1,
1813  };
1814
1815  /************************************************************************/
1816  char *spegetstring(Spe * spe, uint_032t addr)
1817  {
1818      int length = 0;
1819      uint_008t c;
1820      char *result;
1821
1822      do {
1823          spe->ls->read1b((addr + length) & spe->as->lslr, &c);
1824          length++;
1825      } while (c != '\0');
1826      result = new char[length];
1827      for (int i = 0; i < length; i++) {
1828          spe->ls->read1b((addr + i) & spe->as->lslr, &c);
1829          result[i] = (char) c;
1830      } return result;
1831  }
1832
1833  /************************************************************************/
1834  inline void skipchar(char *targ, char *fmt, int *index)
1835  {
1836      while (strchr(targ, (int) *(fmt + *index)))
1837          (*index)++;
1838  }
1839
1840  /************************************************************************/
1841  int spevprintf(Spe * spe, char *fmt, uint_032t addr)
1842  {
1843      int total, index, head, arglen;
1844      char *temp = new char[strlen(fmt) + 1];
1845      total = 0;
1846      for (index = 0; *(fmt + index) != '\0'; index++) {
1847          if (*(fmt + index) == '%') {
```

```
1848              head = index;
1849              index++;
1850              strcpy(temp, fmt);
1851              skipchar("#0- +'I", fmt, &index);
1852              skipchar("0123456789*.", fmt, &index);
1853              if (strchr("cdipouxX", (int) *(fmt + index))) {
1854                  arglen = AL_032;
1855              } else if (strchr("eEfFgG", (int) *(fmt + index))) {
1856                  arglen = AL_F64;
1857              } else if (*(fmt + index) == 'h') {
1858                  arglen = AL_032;
1859              } else if (*(fmt + index) == 'l') {
1860                  index++;
1861                  if (strchr("eEfFgG", (int) *(fmt + index))) {
1862                      arglen = AL_F64;
1863                  } else if (*(fmt + index) == 'l') {
1864                      arglen = AL_064;
1865                      index++;
1866                  } else {
1867                      arglen = AL_032;
1868                  }
1869              } else if (*(fmt + index) == 's') {
1870                  arglen = AL_STR;
1871              } else {
1872                  arglen = AL_NON;
1873              }
1874              *(temp + index + 1) = '\0';
1875              if (arglen == AL_032) {
1876                  uint_032t arg_032;
1877                  spe->ls->read4b(addr, &arg_032);
1878                  total += printf(temp + head, arg_032, DUMMY, DUMMY);
1879                  addr += 16;
1880              } else if ((arglen == AL_064) || (arglen == AL_F64)) {
1881                  united_064t arg_064;
1882                  uint_032t high, low;
1883                  spe->ls->read4b(addr, &high);
1884                  spe->ls->read4b(addr + 4, &low);
1885                  arg_064.lli = ((uint_064t) high << 32) | low;
1886                  if (arglen == AL_064) {
1887                      total += printf(temp + head, arg_064.lli, DUMMY, DUMMY);
1888                  } else {
1889                      float arg_flt = (float) arg_064.dbl;
1890                      total += printf(temp + head, arg_flt, DUMMY, DUMMY);
1891                  }
1892                  addr += 16;
1893              } else if (arglen == AL_STR) {
1894                  uint_032t arg_addr;
1895                  spe->ls->read4b(addr, &arg_addr);
1896                  arg_addr = arg_addr & spe->as->lslr;
1897                  char *arg_str = spegetstring(spe, arg_addr);
1898                  total += printf(temp + head, arg_str, DUMMY, DUMMY);
1899                  delete[]arg_str;
1900                  addr += 16;
1901              } else {
1902                  total += printf(temp + head, DUMMY, DUMMY);
1903              }
1904          } else {
1905              putchar((int) *(fmt + index));
1906              total++;
1907          }
1908      }
1909      delete[]temp;
1910      return total;
1911  }
1912
1913  /*****************************************************************/
1914  void spefunccall(Spe * spe)
1915  {
1916      uint_032t op, func, addr, arg, ret;
1917      spe->ls->read4b(spe->as->pc, &op);
1918      spe->as->pc += 4;
1919      func = op >> 24;
1920      addr = op & spe->as->lslr;
1921      spe->ls->read4b(addr, &arg);
1922      if (func == SPEFUNC_PUTCHAR) {
1923              // putchar()
1924          ret = putchar((int) arg);
```



```
1925      } else if (func == SPEFUNC_PUTS) {
1926              // puts()
1927          char *temp = spegetstring(spe, arg & spe->as->lslr);
1928          ret = puts(temp);
1929          delete[]temp;
1930      } else if (func == SPEFUNC_PRINTF) {
1931              // vprintf()
1932          char *fmt = spegetstring(spe, arg & spe->as->lslr);
1933          addr += 16;
1934          spe->ls->read4b(addr, &arg);
1935          ret = spevprintf(spe, fmt, arg & spe->as->lslr);
1936          delete[]fmt;
1937      } else {
1938          printf("## SPE %d unhandled function call(#%d). skip.\n",
1939                  spe->unitid - 1, (int) func);
1940          ret = arg;
1941      } spe->ls->write4b(addr, ret);
1942  }
1943
1944  /*****************************************************************/
```

file name: spuinst.cc

```
  1  /**************************************************************************/
  2  /*  SimCell: Simulator of Cell/B.E.              Arch Lab. TOKYO TECH */
  3  /**************************************************************************/
  4  #include "define.h"
  5
  6  /**************************************************************************/
  7  SpuInst::SpuInst(Chip * chip, Spe * spe)
  8  {
  9      this->chip = chip;
 10      this->spe = spe;
 11      this->as = spe->as;
 12      this->iinfo = chip->iinfo;
 13      rra = NULL;
 14      rrb = NULL;
 15      rrc = NULL;
 16      rrt = new Register(chip);
 17  }
 18
 19  /**************************************************************************/
 20  SpuInst::~SpuInst()
 21  {
 22      delete rra;
 23      rra = NULL;
 24      delete rrb;
 25      rrb = NULL;
 26      delete rrc;
 27      rrc = NULL;
 28      delete rrt;
 29      rrt = NULL;
 30  }
 31
 32  /**************************************************************************/
 33  void SpuInst::exprinthead()
 34  {
 35      printf("\n[id %d / inst %lld]\n", spe->unitid, spe->ss->instcnt);
 36      printf("%5x:", (uint) pc);
 37      for (int i = 0; i < 4; i++)
 38          printf(" %02x", (uint) ((ir >> (24 - i * 8)) & 0xff));
 39      printf("  %s\n", getopname());
 40      if (type & IT_R_I7)
 41          printf("imm7 = %d(0x%x)\n", (int) exts32(imm, 7), imm);
 42      else if (type & IT_R_I8)
 43          printf("imm8 = %d(0x%x)\n", (int) exts32(imm, 8), imm);
 44      else if (type & IT_R_I10)
 45          printf("imm10 = %d(0x%x)\n", (int) exts32(imm, 10), imm);
 46      else if (type & IT_R_I16)
 47          printf("imm16 = %d(0x%x)\n", (int) exts32(imm, 16), imm);
 48      else if (type & IT_R_I18)
 49          printf("imm18 = %d(0x%x)\n", (int) exts32(imm, 18), imm);
 50      if (type & IT_R_CA)
 51          printf("ca = %d(0x%x)\n", ca, ca);
 52      if ((type & IT_R_RO1) || (type & IT_R_RO2))
 53          printf("ro = 0x%05x\n", ro);
 54      if (type & IT_R_RA) {
 55          printf("ra = r%3d: ", ra);
 56          rra->print();
 57      }
 58      if (type & IT_R_RB) {
 59          printf("rb = r%3d: ", rb);
 60          rrb->print();
 61      }
 62      if (type & IT_R_RC) {
 63          printf("rc = r%3d: ", rc);
 64          rrc->print();
 65      }
 66      if ((type & IT_R_RT) || (type & IT_W_RT)) {
 67          printf("rt = r%3d: ", rt);
 68          if (type & IT_R_RT)
 69              rrt->print();
 70          else
 71              printf("0x--------------------------------\n");
 72      }
 73  }
 74
 75  /**************************************************************************/
 76  void SpuInst::exprinttail()
```



```
 77  {
 78      if (type & IT_W_RT) {
 79          printf("          => ");
 80          rrt->print();
 81      }
 82  }
 83
 84  /**************************************************************************/
 85  int SpuInst::isload()
 86  {
 87      return ((op >= LQD______) && (op <= LQR______));
 88  }
 89
 90  /**************************************************************************/
 91  int SpuInst::isstore()
 92  {
 93      return ((op >= STQD_____) && (op <= STQR_____));
 94  }
 95
 96  /**************************************************************************/
 97  int SpuInst::isbranch()
 98  {
 99      return ((op >= BR_______) && (op <= BIHNZ____));
100  }
101
102  /**************************************************************************/
103  int SpuInst::ishintbranch()
104  {
105      return ((op >= HBR______) && (op <= HBRR_____));
106  }
107
108  /**************************************************************************/
109  char *SpuInst::getopname()
110  {
111      return iinfo->getopname(op);
112  }
113
114  /**************************************************************************/
115  void SpuInst::clear()
116  {
117      rra = NULL;
118      rrb = NULL;
119      rrc = NULL;
120      targ = NOADDR;
121      pc = NOADDR;
122      nextpc = NOADDR;
123      nextstate = SPE_RUN;
124  }
125
126  /**************************************************************************/
127  void SpuInst::decode()
128  {
129      op = iinfo->getinstid(ir >> 21);
130      spuinstattr_t attr = iinfo->getinstattr(op);
131      type = attr.type;
132      pipe = attr.pipe;
133      latency = attr.latency;
134      stall = attr.stall;
135      nextpc = pc + 4;
136
137      rt = ir & 0x7f;
138      ra = (ir >> 7) & 0x7f;
139      rb = (ir >> 14) & 0x7f;
140      if (type & IT_R_RC) {
141          rc = rt;
142          rt = (ir >> 21) & 0x7f;
143      }
144
145      if (type & IT_W_RT)
146          depend = rt;
147      else
148          depend = 128;
149
150      if (type & IT_R_I18)
151          imm = (ir >> 7) & 0x3ffff;
152      else if (type & IT_R_I16)
153          imm = (ir >> 7) & 0xffff;
```

```
154      else if (type & IT_R_I10)
155          imm = (ir >> 14) & 0x3ff;
156      else if (type & IT_R_I8)
157          imm = (ir >> 14) & 0xff;
158      else if (type & IT_R_I7)
159          imm = (ir >> 14) & 0x7f;
160
161      ca = ra;
162
163      if (type & IT_R_RO1)
164          ro = pc + exts32(((ir & 0x7f) | ((ir >> 7) & 0x180)) << 2, 11);
165      else if (type & IT_R_RO2)
166          ro = pc + exts32(((ir & 0x7f) | ((ir >> 16) & 0x180)) << 2, 11);
167
168      if (type & IT_R_RA)
169          rra = as->reg[ra];
170      if (type & IT_R_RB)
171          rrb = as->reg[rb];
172      if (type & IT_R_RC)
173          rrc = as->reg[rc];
174      if (type & IT_R_RT)
175          rrt->put128t(as->reg[rt]->get128t());
176  }
177
178  /*************************************************************************/
179  int SpuInstinfo::predecode(int ir)
180  {
181      int opc;
182
183      // opcode = 11bit
184      opc = ir & 0x7ff;
185      if      (opc == 0x1c4) return LQX______;
186      else if (opc == 0x144) return STQX_____;
187      else if (opc == 0x1f4) return CBD______;
188      else if (opc == 0x1d4) return CBX______;
189      else if (opc == 0x1f5) return CHD______;
190      else if (opc == 0x1d5) return CHX______;
191      else if (opc == 0x1f6) return CWD______;
192      else if (opc == 0x1d6) return CWX______;
193      else if (opc == 0x1f7) return CDD______;
194      else if (opc == 0x1d7) return CDX______;
195      else if (opc == 0x0c8) return AH_______;
196      else if (opc == 0x0c0) return A________;
197      else if (opc == 0x048) return SFH______;
198      else if (opc == 0x040) return SF_______;
199      else if (opc == 0x340) return ADDX_____;
200      else if (opc == 0x0c2) return CG_______;
201      else if (opc == 0x342) return CGX______;
202      else if (opc == 0x341) return SFX______;
203      else if (opc == 0x042) return BG_______;
204      else if (opc == 0x343) return BGX______;
205      else if (opc == 0x3c4) return MPY______;
206      else if (opc == 0x3cc) return MPYU_____;
207      else if (opc == 0x3c5) return MPYH_____;
208      else if (opc == 0x3c7) return MPYS_____;
209      else if (opc == 0x3c6) return MPYHH____;
210      else if (opc == 0x346) return MPYHHA___;
211      else if (opc == 0x3ce) return MPYHHU___;
212      else if (opc == 0x34e) return MPYHHAU__;
213      else if (opc == 0x2a5) return CLZ______;
214      else if (opc == 0x2b4) return CNTB_____;
215      else if (opc == 0x1b6) return FSMB_____;
216      else if (opc == 0x1b5) return FSMH_____;
217      else if (opc == 0x1b4) return FSM______;
218      else if (opc == 0x1b2) return GBB______;
219      else if (opc == 0x1b1) return GBH______;
220      else if (opc == 0x1b0) return GB_______;
221      else if (opc == 0x0d3) return AVGB_____;
222      else if (opc == 0x053) return ABSDB____;
223      else if (opc == 0x253) return SUMB_____;
224      else if (opc == 0x2b6) return XSBH_____;
225      else if (opc == 0x2ae) return XSHW_____;
226      else if (opc == 0x2a6) return XSWD_____;
227      else if (opc == 0x0c1) return AND______;
228      else if (opc == 0x2c1) return ANDC_____;
229      else if (opc == 0x041) return OR_______;
230      else if (opc == 0x2c9) return ORC______;
```



```
231      else if (opc == 0x1f0) return ORX______;
232      else if (opc == 0x241) return XOR______;
233      else if (opc == 0x0c9) return NAND_____;
234      else if (opc == 0x049) return NOR______;
235      else if (opc == 0x249) return EQV______;
236      else if (opc == 0x05f) return SHLH_____;
237      else if (opc == 0x07f) return SHLHI____;
238      else if (opc == 0x05b) return SHL______;
239      else if (opc == 0x07b) return SHLI_____;
240      else if (opc == 0x1db) return SHLQBI___;
241      else if (opc == 0x1fb) return SHLQBII__;
242      else if (opc == 0x1df) return SHLQBY___;
243      else if (opc == 0x1ff) return SHLQBYI__;
244      else if (opc == 0x1cf) return SHLQBYBI_;
245      else if (opc == 0x05c) return ROTH_____;
246      else if (opc == 0x07c) return ROTHI____;
247      else if (opc == 0x058) return ROT______;
248      else if (opc == 0x078) return ROTI_____;
249      else if (opc == 0x1dc) return ROTQBY___;
250      else if (opc == 0x1fc) return ROTQBYI__;
251      else if (opc == 0x1cc) return ROTQBYBI_;
252      else if (opc == 0x1d8) return ROTQBI___;
253      else if (opc == 0x1f8) return ROTQBII__;
254      else if (opc == 0x05d) return ROTHM____;
255      else if (opc == 0x07d) return ROTHMI___;
256      else if (opc == 0x059) return ROTM_____;
257      else if (opc == 0x079) return ROTMI____;
258      else if (opc == 0x1dd) return ROTQMBY__;
259      else if (opc == 0x1fd) return ROTQMBYI_;
260      else if (opc == 0x1cd) return ROTQMBYBI;
261      else if (opc == 0x1d9) return ROTQMBI__;
262      else if (opc == 0x1f9) return ROTQMBII_;
263      else if (opc == 0x05e) return ROTMAH___;
264      else if (opc == 0x07e) return ROTMAHI__;
265      else if (opc == 0x05a) return ROTMA____;
266      else if (opc == 0x07a) return ROTMAI___;
267      else if (opc == 0x3d8) return HEQ______;
268      else if (opc == 0x258) return HGT______;
269      else if (opc == 0x2d8) return HLGT_____;
270      else if (opc == 0x3d0) return CEQB_____;
271      else if (opc == 0x3c8) return CEQH_____;
272      else if (opc == 0x3c0) return CEQ______;
273      else if (opc == 0x250) return CGTB_____;
274      else if (opc == 0x248) return CGTH_____;
275      else if (opc == 0x240) return CGT______;
276      else if (opc == 0x2d0) return CLGTB____;
277      else if (opc == 0x2c8) return CLGTH____;
278      else if (opc == 0x2c0) return CLGT_____;
279      else if (opc == 0x1a8) return BI_______;
280      else if (opc == 0x1aa) return IRET_____;
281      else if (opc == 0x1ab) return BISLED___;
282      else if (opc == 0x1a9) return BISL_____;
283      else if (opc == 0x128) return BIZ______;
284      else if (opc == 0x129) return BINZ_____;
285      else if (opc == 0x12a) return BIHZ_____;
286      else if (opc == 0x12b) return BIHNZ____;
287      else if (opc == 0x1ac) return HBR______;
288      else if (opc == 0x2c4) return FA_______;
289      else if (opc == 0x2cc) return DFA______;
290      else if (opc == 0x2c5) return FS_______;
291      else if (opc == 0x2cd) return DFS______;
292      else if (opc == 0x2c6) return FM_______;
293      else if (opc == 0x2ce) return DFM______;
294      else if (opc == 0x35c) return DFMA_____;
295      else if (opc == 0x35e) return DFNMS____;
296      else if (opc == 0x35d) return DFMS_____;
297      else if (opc == 0x35f) return DFNMA____;
298      else if (opc == 0x1b8) return FREST____;
299      else if (opc == 0x1b9) return FRSQEST__;
300      else if (opc == 0x3d4) return FI_______;
301      else if (opc == 0x3b9) return FRDS_____;
302      else if (opc == 0x3b8) return FESD_____;
303      else if (opc == 0x3c3) return DFCEQ____;
304      else if (opc == 0x3cb) return DFCMEQ___;
305      else if (opc == 0x2c3) return DFCGT____;
306      else if (opc == 0x2cb) return DFCMGT___;
307      else if (opc == 0x3bf) return DFTSV____;
```

```
308       else if (opc == 0x3c2) return FCEQ_____;
309       else if (opc == 0x3ca) return FCMEQ____;
310       else if (opc == 0x2c2) return FCGT_____;
311       else if (opc == 0x2ca) return FCMGT____;
312       else if (opc == 0x3ba) return FSCRWR___;
313       else if (opc == 0x398) return FSCRRD___;
314       else if (opc == 0x000) return STOP_____;
315       else if (opc == 0x140) return STOPD____;
316       else if (opc == 0x001) return LNOP_____;
317       else if (opc == 0x201) return NOP______;
318       else if (opc == 0x002) return SYNC_____;
319       else if (opc == 0x003) return DSYNC____;
320       else if (opc == 0x00c) return MFSPR____;
321       else if (opc == 0x10c) return MTSPR____;
322       else if (opc == 0x00d) return RDCH_____;
323       else if (opc == 0x10d) return WRCH_____;
324       else if (opc == 0x00f) return RCHCNT___;
325
326       // opcode = 10bit
327       opc = (ir >> 1) & 0x3ff;
328       if       (opc == 0x1da) return CSFLT____;
329       else if (opc == 0x1d8) return CFLTS____;
330       else if (opc == 0x1db) return CUFLT____;
331       else if (opc == 0x1d9) return CFLTU____;
332
333       // opcode = 9bit
334       opc = (ir >> 2) & 0x1ff;
335       if       (opc == 0x061) return LQA______;
336       else if (opc == 0x067) return LQR______;
337       else if (opc == 0x041) return STQA_____;
338       else if (opc == 0x047) return STQR_____;
339       else if (opc == 0x083) return ILH______;
340       else if (opc == 0x082) return ILHU_____;
341       else if (opc == 0x081) return IL_______;
342       else if (opc == 0x0c1) return IOHL_____;
343       else if (opc == 0x065) return FSMBI____;
344       else if (opc == 0x064) return BR_______;
345       else if (opc == 0x060) return BRA______;
346       else if (opc == 0x066) return BRSL_____;
347       else if (opc == 0x062) return BRASL____;
348       else if (opc == 0x042) return BRNZ_____;
349       else if (opc == 0x040) return BRZ______;
350       else if (opc == 0x046) return BRHNZ____;
351       else if (opc == 0x044) return BRHZ_____;
352
353       // opcode = 8bit
354       opc = (ir >> 3) & 0xff;
355       if       (opc == 0x34) return LQD______;
356       else if (opc == 0x24) return STQD_____;
357       else if (opc == 0x1d) return AHI______;
358       else if (opc == 0x1c) return AI_______;
359       else if (opc == 0x0d) return SFHI_____;
360       else if (opc == 0x0c) return SFI______;
361       else if (opc == 0x74) return MPYI_____;
362       else if (opc == 0x75) return MPYUI____;
363       else if (opc == 0x16) return ANDBI____;
364       else if (opc == 0x15) return ANDHI____;
365       else if (opc == 0x14) return ANDI_____;
366       else if (opc == 0x06) return ORBI_____;
367       else if (opc == 0x05) return ORHI_____;
368       else if (opc == 0x04) return ORI______;
369       else if (opc == 0x46) return XORBI____;
370       else if (opc == 0x45) return XORHI____;
371       else if (opc == 0x44) return XORI_____;
372       else if (opc == 0x7f) return HEQI_____;
373       else if (opc == 0x4f) return HGTI_____;
374       else if (opc == 0x5f) return HLGTI____;
375       else if (opc == 0x7e) return CEQBI____;
376       else if (opc == 0x7d) return CEQHI____;
377       else if (opc == 0x7c) return CEQI_____;
378       else if (opc == 0x4e) return CGTBI____;
379       else if (opc == 0x4d) return CGTHI____;
380       else if (opc == 0x4c) return CGTI_____;
381       else if (opc == 0x5e) return CLGTBI___;
382       else if (opc == 0x5d) return CLGTHI___;
383       else if (opc == 0x5c) return CLGTI____;
384
```



```
385       // opcode = 7bit
386       opc = (ir >> 4) & 0x7f;
387       if       (opc == 0x21) return ILA______;
388       else if (opc == 0x08) return HBRA_____;
389       else if (opc == 0x09) return HBRR_____;
390
391       // opcode = 4bit
392       opc = (ir >> 7) & 0xf;
393       if       (opc == 0xc) return MPYA_____;
394       else if (opc == 0x8) return SELB_____;
395       else if (opc == 0xb) return SHUFB____;
396       else if (opc == 0xe) return FMA______;
397       else if (opc == 0xd) return FNMS_____;
398       else if (opc == 0xf) return FMS______;
399
400       return UNDEF____;
401   }
402
403   /*************************************************************************/
404   void SpuInstinfo::setattr()
405   {
406       id_to_attr[LQD______].name    = "lqd";
407       id_to_attr[LQD______].type    = IT_W_RT | IT_R_RA | IT_R_I10;
408       id_to_attr[LQD______].pipe    = PIPE_ODD;
409       id_to_attr[LQD______].latency = 6;
410       id_to_attr[LQD______].stall   = 0;
411       id_to_attr[LQX______].name    = "lqx";
412       id_to_attr[LQX______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
413       id_to_attr[LQX______].pipe    = PIPE_ODD;
414       id_to_attr[LQX______].latency = 6;
415       id_to_attr[LQX______].stall   = 0;
416       id_to_attr[LQA______].name    = "lqa";
417       id_to_attr[LQA______].type    = IT_W_RT | IT_R_I16;
418       id_to_attr[LQA______].pipe    = PIPE_ODD;
419       id_to_attr[LQA______].latency = 6;
420       id_to_attr[LQA______].stall   = 0;
421       id_to_attr[LQR______].name    = "lqr";
422       id_to_attr[LQR______].type    = IT_W_RT | IT_R_I16;
423       id_to_attr[LQR______].pipe    = PIPE_ODD;
424       id_to_attr[LQR______].latency = 6;
425       id_to_attr[LQR______].stall   = 0;
426       id_to_attr[STQD_____].name    = "stqd";
427       id_to_attr[STQD_____].type    = IT_R_RA | IT_R_RT | IT_R_I10;
428       id_to_attr[STQD_____].pipe    = PIPE_ODD;
429       id_to_attr[STQD_____].latency = 6;
430       id_to_attr[STQD_____].stall   = 0;
431       id_to_attr[STQX_____].name    = "stqx";
432       id_to_attr[STQX_____].type    = IT_R_RA | IT_R_RB | IT_R_RT;
433       id_to_attr[STQX_____].pipe    = PIPE_ODD;
434       id_to_attr[STQX_____].latency = 6;
435       id_to_attr[STQX_____].stall   = 0;
436       id_to_attr[STQA_____].name    = "stqa";
437       id_to_attr[STQA_____].type    = IT_R_RT | IT_R_I16;
438       id_to_attr[STQA_____].pipe    = PIPE_ODD;
439       id_to_attr[STQA_____].latency = 6;
440       id_to_attr[STQA_____].stall   = 0;
441       id_to_attr[STQR_____].name    = "stqr";
442       id_to_attr[STQR_____].type    = IT_R_RT | IT_R_I16;
443       id_to_attr[STQR_____].pipe    = PIPE_ODD;
444       id_to_attr[STQR_____].latency = 6;
445       id_to_attr[STQR_____].stall   = 0;
446       id_to_attr[CBD______].name    = "cbd";
447       id_to_attr[CBD______].type    = IT_W_RT | IT_R_RA | IT_R_I7;
448       id_to_attr[CBD______].pipe    = PIPE_ODD;
449       id_to_attr[CBD______].latency = 4;
450       id_to_attr[CBD______].stall   = 0;
451       id_to_attr[CBX______].name    = "cbx";
452       id_to_attr[CBX______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
453       id_to_attr[CBX______].pipe    = PIPE_ODD;
454       id_to_attr[CBX______].latency = 4;
455       id_to_attr[CBX______].stall   = 0;
456       id_to_attr[CHD______].name    = "chd";
457       id_to_attr[CHD______].type    = IT_W_RT | IT_R_RA | IT_R_I7;
458       id_to_attr[CHD______].pipe    = PIPE_ODD;
459       id_to_attr[CHD______].latency = 4;
460       id_to_attr[CHD______].stall   = 0;
461       id_to_attr[CHX______].name    = "chx";
```

```
462     id_to_attr[CHX______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
463     id_to_attr[CHX______].pipe    = PIPE_ODD;
464     id_to_attr[CHX______].latency = 4;
465     id_to_attr[CHX______].stall   = 0;
466     id_to_attr[CWD______].name    = "cwd";
467     id_to_attr[CWD______].type    = IT_W_RT | IT_R_RA | IT_R_I7;
468     id_to_attr[CWD______].pipe    = PIPE_ODD;
469     id_to_attr[CWD______].latency = 4;
470     id_to_attr[CWD______].stall   = 0;
471     id_to_attr[CWX______].name    = "cwx";
472     id_to_attr[CWX______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
473     id_to_attr[CWX______].pipe    = PIPE_ODD;
474     id_to_attr[CWX______].latency = 4;
475     id_to_attr[CWX______].stall   = 0;
476     id_to_attr[CDD______].name    = "cdd";
477     id_to_attr[CDD______].type    = IT_W_RT | IT_R_RA | IT_R_I7;
478     id_to_attr[CDD______].pipe    = PIPE_ODD;
479     id_to_attr[CDD______].latency = 4;
480     id_to_attr[CDD______].stall   = 0;
481     id_to_attr[CDX______].name    = "cdx";
482     id_to_attr[CDX______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
483     id_to_attr[CDX______].pipe    = PIPE_ODD;
484     id_to_attr[CDX______].latency = 4;
485     id_to_attr[CDX______].stall   = 0;
486     id_to_attr[ILH______].name    = "ilh";
487     id_to_attr[ILH______].type    = IT_W_RT | IT_R_I16;
488     id_to_attr[ILH______].pipe    = PIPE_EVEN;
489     id_to_attr[ILH______].latency = 2;
490     id_to_attr[ILH______].stall   = 0;
491     id_to_attr[ILHU_____].name    = "ilhu";
492     id_to_attr[ILHU_____].type    = IT_W_RT | IT_R_I16;
493     id_to_attr[ILHU_____].pipe    = PIPE_EVEN;
494     id_to_attr[ILHU_____].latency = 2;
495     id_to_attr[ILHU_____].stall   = 0;
496     id_to_attr[IL_______].name    = "il";
497     id_to_attr[IL_______].type    = IT_W_RT | IT_R_I16;
498     id_to_attr[IL_______].pipe    = PIPE_EVEN;
499     id_to_attr[IL_______].latency = 2;
500     id_to_attr[IL_______].stall   = 0;
501     id_to_attr[ILA______].name    = "ila";
502     id_to_attr[ILA______].type    = IT_W_RT | IT_R_I18;
503     id_to_attr[ILA______].pipe    = PIPE_EVEN;
504     id_to_attr[ILA______].latency = 2;
505     id_to_attr[ILA______].stall   = 0;
506     id_to_attr[IOHL_____].name    = "iohl";
507     id_to_attr[IOHL_____].type    = IT_W_RT | IT_R_RT | IT_R_I16;
508     id_to_attr[IOHL_____].pipe    = PIPE_EVEN;
509     id_to_attr[IOHL_____].latency = 2;
510     id_to_attr[IOHL_____].stall   = 0;
511     id_to_attr[FSMBI____].name    = "fsmbi";
512     id_to_attr[FSMBI____].type    = IT_W_RT | IT_R_I16;
513     id_to_attr[FSMBI____].pipe    = PIPE_ODD;
514     id_to_attr[FSMBI____].latency = 4;
515     id_to_attr[FSMBI____].stall   = 0;
516     id_to_attr[AH_______].name    = "ah";
517     id_to_attr[AH_______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
518     id_to_attr[AH_______].pipe    = PIPE_EVEN;
519     id_to_attr[AH_______].latency = 2;
520     id_to_attr[AH_______].stall   = 0;
521     id_to_attr[AHI______].name    = "ahi";
522     id_to_attr[AHI______].type    = IT_W_RT | IT_R_RA | IT_R_I10;
523     id_to_attr[AHI______].pipe    = PIPE_EVEN;
524     id_to_attr[AHI______].latency = 2;
525     id_to_attr[AHI______].stall   = 0;
526     id_to_attr[A________].name    = "a";
527     id_to_attr[A________].type    = IT_W_RT | IT_R_RA | IT_R_RB;
528     id_to_attr[A________].pipe    = PIPE_EVEN;
529     id_to_attr[A________].latency = 2;
530     id_to_attr[A________].stall   = 0;
531     id_to_attr[AI_______].name    = "ai";
532     id_to_attr[AI_______].type    = IT_W_RT | IT_R_RA | IT_R_I10;
533     id_to_attr[AI_______].pipe    = PIPE_EVEN;
534     id_to_attr[AI_______].latency = 2;
535     id_to_attr[AI_______].stall   = 0;
536     id_to_attr[SFH______].name    = "sfh";
537     id_to_attr[SFH______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
538     id_to_attr[SFH______].pipe    = PIPE_EVEN;
```



```
539     id_to_attr[SFH______].latency = 2;
540     id_to_attr[SFH______].stall   = 0;
541     id_to_attr[SFHI_____].name    = "sfhi";
542     id_to_attr[SFHI_____].type    = IT_W_RT | IT_R_RA | IT_R_I10;
543     id_to_attr[SFHI_____].pipe    = PIPE_EVEN;
544     id_to_attr[SFHI_____].latency = 2;
545     id_to_attr[SFHI_____].stall   = 0;
546     id_to_attr[SF_______].name    = "sf";
547     id_to_attr[SF_______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
548     id_to_attr[SF_______].pipe    = PIPE_EVEN;
549     id_to_attr[SF_______].latency = 2;
550     id_to_attr[SF_______].stall   = 0;
551     id_to_attr[SFI______].name    = "sfi";
552     id_to_attr[SFI______].type    = IT_W_RT | IT_R_RA | IT_R_I10;
553     id_to_attr[SFI______].pipe    = PIPE_EVEN;
554     id_to_attr[SFI______].latency = 2;
555     id_to_attr[SFI______].stall   = 0;
556     id_to_attr[ADDX_____].name    = "addx";
557     id_to_attr[ADDX_____].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RT;
558     id_to_attr[ADDX_____].pipe    = PIPE_EVEN;
559     id_to_attr[ADDX_____].latency = 2;
560     id_to_attr[ADDX_____].stall   = 0;
561     id_to_attr[CG_______].name    = "cg";
562     id_to_attr[CG_______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
563     id_to_attr[CG_______].pipe    = PIPE_EVEN;
564     id_to_attr[CG_______].latency = 2;
565     id_to_attr[CG_______].stall   = 0;
566     id_to_attr[CGX______].name    = "cgx";
567     id_to_attr[CGX______].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RT;
568     id_to_attr[CGX______].pipe    = PIPE_EVEN;
569     id_to_attr[CGX______].latency = 2;
570     id_to_attr[CGX______].stall   = 0;
571     id_to_attr[SFX______].name    = "sfx";
572     id_to_attr[SFX______].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RT;
573     id_to_attr[SFX______].pipe    = PIPE_EVEN;
574     id_to_attr[SFX______].latency = 2;
575     id_to_attr[SFX______].stall   = 0;
576     id_to_attr[BG_______].name    = "bg";
577     id_to_attr[BG_______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
578     id_to_attr[BG_______].pipe    = PIPE_EVEN;
579     id_to_attr[BG_______].latency = 2;
580     id_to_attr[BG_______].stall   = 0;
581     id_to_attr[BGX______].name    = "bgx";
582     id_to_attr[BGX______].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RT;
583     id_to_attr[BGX______].pipe    = PIPE_EVEN;
584     id_to_attr[BGX______].latency = 2;
585     id_to_attr[BGX______].stall   = 0;
586     id_to_attr[MPY______].name    = "mpy";
587     id_to_attr[MPY______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
588     id_to_attr[MPY______].pipe    = PIPE_EVEN;
589     id_to_attr[MPY______].latency = 7;
590     id_to_attr[MPY______].stall   = 0;
591     id_to_attr[MPYU_____].name    = "mpyu";
592     id_to_attr[MPYU_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
593     id_to_attr[MPYU_____].pipe    = PIPE_EVEN;
594     id_to_attr[MPYU_____].latency = 7;
595     id_to_attr[MPYU_____].stall   = 0;
596     id_to_attr[MPYI_____].name    = "mpyi";
597     id_to_attr[MPYI_____].type    = IT_W_RT | IT_R_RA | IT_R_I10;
598     id_to_attr[MPYI_____].pipe    = PIPE_EVEN;
599     id_to_attr[MPYI_____].latency = 7;
600     id_to_attr[MPYI_____].stall   = 0;
601     id_to_attr[MPYUI____].name    = "mpyui";
602     id_to_attr[MPYUI____].type    = IT_W_RT | IT_R_RA | IT_R_I10;
603     id_to_attr[MPYUI____].pipe    = PIPE_EVEN;
604     id_to_attr[MPYUI____].latency = 7;
605     id_to_attr[MPYUI____].stall   = 0;
606     id_to_attr[MPYA_____].name    = "mpya";
607     id_to_attr[MPYA_____].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RC;
608     id_to_attr[MPYA_____].pipe    = PIPE_EVEN;
609     id_to_attr[MPYA_____].latency = 7;
610     id_to_attr[MPYA_____].stall   = 0;
611     id_to_attr[MPYH_____].name    = "mpyh";
612     id_to_attr[MPYH_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
613     id_to_attr[MPYH_____].pipe    = PIPE_EVEN;
614     id_to_attr[MPYH_____].latency = 7;
615     id_to_attr[MPYH_____].stall   = 0;
```

```
616     id_to_attr[MPYS_____].name    = "mpys";
617     id_to_attr[MPYS_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
618     id_to_attr[MPYS_____].pipe    = PIPE_EVEN;
619     id_to_attr[MPYS_____].latency = 7;
620     id_to_attr[MPYS_____].stall   = 0;
621     id_to_attr[MPYHH____].name    = "mpyhh";
622     id_to_attr[MPYHH____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
623     id_to_attr[MPYHH____].pipe    = PIPE_EVEN;
624     id_to_attr[MPYHH____].latency = 7;
625     id_to_attr[MPYHH____].stall   = 0;
626     id_to_attr[MPYHHA___].name    = "mpyhha";
627     id_to_attr[MPYHHA___].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RT;
628     id_to_attr[MPYHHA___].pipe    = PIPE_EVEN;
629     id_to_attr[MPYHHA___].latency = 7;
630     id_to_attr[MPYHHA___].stall   = 0;
631     id_to_attr[MPYHHU___].name    = "mpyhhu";
632     id_to_attr[MPYHHU___].type    = IT_W_RT | IT_R_RA | IT_R_RB;
633     id_to_attr[MPYHHU___].pipe    = PIPE_EVEN;
634     id_to_attr[MPYHHU___].latency = 7;
635     id_to_attr[MPYHHU___].stall   = 0;
636     id_to_attr[MPYHHAU__].name    = "mpyhhau";
637     id_to_attr[MPYHHAU__].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RT;
638     id_to_attr[MPYHHAU__].pipe    = PIPE_EVEN;
639     id_to_attr[MPYHHAU__].latency = 7;
640     id_to_attr[MPYHHAU__].stall   = 0;
641     id_to_attr[CLZ______].name    = "clz";
642     id_to_attr[CLZ______].type    = IT_W_RT | IT_R_RA;
643     id_to_attr[CLZ______].pipe    = PIPE_EVEN;
644     id_to_attr[CLZ______].latency = 2;
645     id_to_attr[CLZ______].stall   = 0;
646     id_to_attr[CNTB_____].name    = "cntb";
647     id_to_attr[CNTB_____].type    = IT_W_RT | IT_R_RA;
648     id_to_attr[CNTB_____].pipe    = PIPE_EVEN;
649     id_to_attr[CNTB_____].latency = 4;
650     id_to_attr[CNTB_____].stall   = 0;
651     id_to_attr[FSMB_____].name    = "fsmb";
652     id_to_attr[FSMB_____].type    = IT_W_RT | IT_R_RA;
653     id_to_attr[FSMB_____].pipe    = PIPE_ODD;
654     id_to_attr[FSMB_____].latency = 4;
655     id_to_attr[FSMB_____].stall   = 0;
656     id_to_attr[FSMH_____].name    = "fsmh";
657     id_to_attr[FSMH_____].type    = IT_W_RT | IT_R_RA;
658     id_to_attr[FSMH_____].pipe    = PIPE_ODD;
659     id_to_attr[FSMH_____].latency = 4;
660     id_to_attr[FSMH_____].stall   = 0;
661     id_to_attr[FSM______].name    = "fsm";
662     id_to_attr[FSM______].type    = IT_W_RT | IT_R_RA;
663     id_to_attr[FSM______].pipe    = PIPE_ODD;
664     id_to_attr[FSM______].latency = 4;
665     id_to_attr[FSM______].stall   = 0;
666     id_to_attr[GBB______].name    = "gbb";
667     id_to_attr[GBB______].type    = IT_W_RT | IT_R_RA;
668     id_to_attr[GBB______].pipe    = PIPE_ODD;
669     id_to_attr[GBB______].latency = 4;
670     id_to_attr[GBB______].stall   = 0;
671     id_to_attr[GBH______].name    = "gbh";
672     id_to_attr[GBH______].type    = IT_W_RT | IT_R_RA;
673     id_to_attr[GBH______].pipe    = PIPE_ODD;
674     id_to_attr[GBH______].latency = 4;
675     id_to_attr[GBH______].stall   = 0;
676     id_to_attr[GB_______].name    = "gb";
677     id_to_attr[GB_______].type    = IT_W_RT | IT_R_RA;
678     id_to_attr[GB_______].pipe    = PIPE_ODD;
679     id_to_attr[GB_______].latency = 4;
680     id_to_attr[GB_______].stall   = 0;
681     id_to_attr[AVGB_____].name    = "avgb";
682     id_to_attr[AVGB_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
683     id_to_attr[AVGB_____].pipe    = PIPE_EVEN;
684     id_to_attr[AVGB_____].latency = 4;
685     id_to_attr[AVGB_____].stall   = 0;
686     id_to_attr[ABSDB____].name    = "absdb";
687     id_to_attr[ABSDB____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
688     id_to_attr[ABSDB____].pipe    = PIPE_EVEN;
689     id_to_attr[ABSDB____].latency = 4;
690     id_to_attr[ABSDB____].stall   = 0;
691     id_to_attr[SUMB_____].name    = "sumb";
692     id_to_attr[SUMB_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
```



```
693     id_to_attr[SUMB_____].pipe    = PIPE_EVEN;
694     id_to_attr[SUMB_____].latency = 4;
695     id_to_attr[SUMB_____].stall   = 0;
696     id_to_attr[XSBH_____].name    = "xsbh";
697     id_to_attr[XSBH_____].type    = IT_W_RT | IT_R_RA;
698     id_to_attr[XSBH_____].pipe    = PIPE_EVEN;
699     id_to_attr[XSBH_____].latency = 2;
700     id_to_attr[XSBH_____].stall   = 0;
701     id_to_attr[XSHW_____].name    = "xshw";
702     id_to_attr[XSHW_____].type    = IT_W_RT | IT_R_RA;
703     id_to_attr[XSHW_____].pipe    = PIPE_EVEN;
704     id_to_attr[XSHW_____].latency = 2;
705     id_to_attr[XSHW_____].stall   = 0;
706     id_to_attr[XSWD_____].name    = "xswd";
707     id_to_attr[XSWD_____].type    = IT_W_RT | IT_R_RA;
708     id_to_attr[XSWD_____].pipe    = PIPE_EVEN;
709     id_to_attr[XSWD_____].latency = 2;
710     id_to_attr[XSWD_____].stall   = 0;
711     id_to_attr[AND______].name    = "and";
712     id_to_attr[AND______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
713     id_to_attr[AND______].pipe    = PIPE_EVEN;
714     id_to_attr[AND______].latency = 2;
715     id_to_attr[AND______].stall   = 0;
716     id_to_attr[ANDC_____].name    = "andc";
717     id_to_attr[ANDC_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
718     id_to_attr[ANDC_____].pipe    = PIPE_EVEN;
719     id_to_attr[ANDC_____].latency = 2;
720     id_to_attr[ANDC_____].stall   = 0;
721     id_to_attr[ANDBI____].name    = "andbi";
722     id_to_attr[ANDBI____].type    = IT_W_RT | IT_R_RA | IT_R_I10;
723     id_to_attr[ANDBI____].pipe    = PIPE_EVEN;
724     id_to_attr[ANDBI____].latency = 2;
725     id_to_attr[ANDBI____].stall   = 0;
726     id_to_attr[ANDHI____].name    = "andhi";
727     id_to_attr[ANDHI____].type    = IT_W_RT | IT_R_RA | IT_R_I10;
728     id_to_attr[ANDHI____].pipe    = PIPE_EVEN;
729     id_to_attr[ANDHI____].latency = 2;
730     id_to_attr[ANDHI____].stall   = 0;
731     id_to_attr[ANDI_____].name    = "andi";
732     id_to_attr[ANDI_____].type    = IT_W_RT | IT_R_RA | IT_R_I10;
733     id_to_attr[ANDI_____].pipe    = PIPE_EVEN;
734     id_to_attr[ANDI_____].latency = 2;
735     id_to_attr[ANDI_____].stall   = 0;
736     id_to_attr[OR_______].name    = "or";
737     id_to_attr[OR_______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
738     id_to_attr[OR_______].pipe    = PIPE_EVEN;
739     id_to_attr[OR_______].latency = 2;
740     id_to_attr[OR_______].stall   = 0;
741     id_to_attr[ORC______].name    = "orc";
742     id_to_attr[ORC______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
743     id_to_attr[ORC______].pipe    = PIPE_EVEN;
744     id_to_attr[ORC______].latency = 2;
745     id_to_attr[ORC______].stall   = 0;
746     id_to_attr[ORBI_____].name    = "orbi";
747     id_to_attr[ORBI_____].type    = IT_W_RT | IT_R_RA | IT_R_I10;
748     id_to_attr[ORBI_____].pipe    = PIPE_EVEN;
749     id_to_attr[ORBI_____].latency = 2;
750     id_to_attr[ORBI_____].stall   = 0;
751     id_to_attr[ORHI_____].name    = "orhi";
752     id_to_attr[ORHI_____].type    = IT_W_RT | IT_R_RA | IT_R_I10;
753     id_to_attr[ORHI_____].pipe    = PIPE_EVEN;
754     id_to_attr[ORHI_____].latency = 2;
755     id_to_attr[ORHI_____].stall   = 0;
756     id_to_attr[ORI______].name    = "ori";
757     id_to_attr[ORI______].type    = IT_W_RT | IT_R_RA | IT_R_I10;
758     id_to_attr[ORI______].pipe    = PIPE_EVEN;
759     id_to_attr[ORI______].latency = 2;
760     id_to_attr[ORI______].stall   = 0;
761     id_to_attr[ORX______].name    = "orx";
762     id_to_attr[ORX______].type    = IT_W_RT | IT_R_RA;
763     id_to_attr[ORX______].pipe    = PIPE_ODD;
764     id_to_attr[ORX______].latency = 4;
765     id_to_attr[ORX______].stall   = 0;
766     id_to_attr[XOR______].name    = "xor";
767     id_to_attr[XOR______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
768     id_to_attr[XOR______].pipe    = PIPE_EVEN;
769     id_to_attr[XOR______].latency = 2;
```

```
770     id_to_attr[XOR_____].stall   = 0;
771     id_to_attr[XORBI____].name    = "xorbi";
772     id_to_attr[XORBI____].type    = IT_W_RT | IT_R_RA | IT_R_I10;
773     id_to_attr[XORBI____].pipe    = PIPE_EVEN;
774     id_to_attr[XORBI____].latency = 2;
775     id_to_attr[XORBI____].stall   = 0;
776     id_to_attr[XORHI____].name    = "xorhi";
777     id_to_attr[XORHI____].type    = IT_W_RT | IT_R_RA | IT_R_I10;
778     id_to_attr[XORHI____].pipe    = PIPE_EVEN;
779     id_to_attr[XORHI____].latency = 2;
780     id_to_attr[XORHI____].stall   = 0;
781     id_to_attr[XORI_____].name    = "xori";
782     id_to_attr[XORI_____].type    = IT_W_RT | IT_R_RA | IT_R_I10;
783     id_to_attr[XORI_____].pipe    = PIPE_EVEN;
784     id_to_attr[XORI_____].latency = 2;
785     id_to_attr[XORI_____].stall   = 0;
786     id_to_attr[NAND_____].name    = "nand";
787     id_to_attr[NAND_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
788     id_to_attr[NAND_____].pipe    = PIPE_EVEN;
789     id_to_attr[NAND_____].latency = 2;
790     id_to_attr[NAND_____].stall   = 0;
791     id_to_attr[NOR______].name    = "nor";
792     id_to_attr[NOR______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
793     id_to_attr[NOR______].pipe    = PIPE_EVEN;
794     id_to_attr[NOR______].latency = 2;
795     id_to_attr[NOR______].stall   = 0;
796     id_to_attr[EQV______].name    = "eqv";
797     id_to_attr[EQV______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
798     id_to_attr[EQV______].pipe    = PIPE_EVEN;
799     id_to_attr[EQV______].latency = 2;
800     id_to_attr[EQV______].stall   = 0;
801     id_to_attr[SELB_____].name    = "selb";
802     id_to_attr[SELB_____].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RC;
803     id_to_attr[SELB_____].pipe    = PIPE_EVEN;
804     id_to_attr[SELB_____].latency = 2;
805     id_to_attr[SELB_____].stall   = 0;
806     id_to_attr[SHUFB____].name    = "shufb";
807     id_to_attr[SHUFB____].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RC;
808     id_to_attr[SHUFB____].pipe    = PIPE_ODD;
809     id_to_attr[SHUFB____].latency = 4;
810     id_to_attr[SHUFB____].stall   = 0;
811     id_to_attr[SHLH_____].name    = "shlh";
812     id_to_attr[SHLH_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
813     id_to_attr[SHLH_____].pipe    = PIPE_EVEN;
814     id_to_attr[SHLH_____].latency = 4;
815     id_to_attr[SHLH_____].stall   = 0;
816     id_to_attr[SHLHI____].name    = "shlhi";
817     id_to_attr[SHLHI____].type    = IT_W_RT | IT_R_RA | IT_R_I7;
818     id_to_attr[SHLHI____].pipe    = PIPE_EVEN;
819     id_to_attr[SHLHI____].latency = 4;
820     id_to_attr[SHLHI____].stall   = 0;
821     id_to_attr[SHL______].name    = "shl";
822     id_to_attr[SHL______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
823     id_to_attr[SHL______].pipe    = PIPE_EVEN;
824     id_to_attr[SHL______].latency = 4;
825     id_to_attr[SHL______].stall   = 0;
826     id_to_attr[SHLI_____].name    = "shli";
827     id_to_attr[SHLI_____].type    = IT_W_RT | IT_R_RA | IT_R_I7;
828     id_to_attr[SHLI_____].pipe    = PIPE_EVEN;
829     id_to_attr[SHLI_____].latency = 4;
830     id_to_attr[SHLI_____].stall   = 0;
831     id_to_attr[SHLQBI___].name    = "shlqbi";
832     id_to_attr[SHLQBI___].type    = IT_W_RT | IT_R_RA | IT_R_RB;
833     id_to_attr[SHLQBI___].pipe    = PIPE_ODD;
834     id_to_attr[SHLQBI___].latency = 4;
835     id_to_attr[SHLQBI___].stall   = 0;
836     id_to_attr[SHLQBII__].name    = "shlqbii";
837     id_to_attr[SHLQBII__].type    = IT_W_RT | IT_R_RA | IT_R_I7;
838     id_to_attr[SHLQBII__].pipe    = PIPE_ODD;
839     id_to_attr[SHLQBII__].latency = 4;
840     id_to_attr[SHLQBII__].stall   = 0;
841     id_to_attr[SHLQBY___].name    = "shlqby";
842     id_to_attr[SHLQBY___].type    = IT_W_RT | IT_R_RA | IT_R_RB;
843     id_to_attr[SHLQBY___].pipe    = PIPE_ODD;
844     id_to_attr[SHLQBY___].latency = 4;
845     id_to_attr[SHLQBY___].stall   = 0;
846     id_to_attr[SHLQBYI__].name    = "shlqbyi";
```



```
847     id_to_attr[SHLQBYI__].type    = IT_W_RT | IT_R_RA | IT_R_I7;
848     id_to_attr[SHLQBYI__].pipe    = PIPE_ODD;
849     id_to_attr[SHLQBYI__].latency = 4;
850     id_to_attr[SHLQBYI__].stall   = 0;
851     id_to_attr[SHLQBYBI_].name    = "shlqbybi";
852     id_to_attr[SHLQBYBI_].type    = IT_W_RT | IT_R_RA | IT_R_RB;
853     id_to_attr[SHLQBYBI_].pipe    = PIPE_ODD;
854     id_to_attr[SHLQBYBI_].latency = 4;
855     id_to_attr[SHLQBYBI_].stall   = 0;
856     id_to_attr[ROTH_____].name    = "roth";
857     id_to_attr[ROTH_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
858     id_to_attr[ROTH_____].pipe    = PIPE_EVEN;
859     id_to_attr[ROTH_____].latency = 4;
860     id_to_attr[ROTH_____].stall   = 0;
861     id_to_attr[ROTHI____].name    = "rothi";
862     id_to_attr[ROTHI____].type    = IT_W_RT | IT_R_RA | IT_R_I7;
863     id_to_attr[ROTHI____].pipe    = PIPE_EVEN;
864     id_to_attr[ROTHI____].latency = 4;
865     id_to_attr[ROTHI____].stall   = 0;
866     id_to_attr[ROT______].name    = "rot";
867     id_to_attr[ROT______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
868     id_to_attr[ROT______].pipe    = PIPE_EVEN;
869     id_to_attr[ROT______].latency = 4;
870     id_to_attr[ROT______].stall   = 0;
871     id_to_attr[ROTI_____].name    = "roti";
872     id_to_attr[ROTI_____].type    = IT_W_RT | IT_R_RA | IT_R_I7;
873     id_to_attr[ROTI_____].pipe    = PIPE_EVEN;
874     id_to_attr[ROTI_____].latency = 4;
875     id_to_attr[ROTI_____].stall   = 0;
876     id_to_attr[ROTQBY___].name    = "rotqby";
877     id_to_attr[ROTQBY___].type    = IT_W_RT | IT_R_RA | IT_R_RB;
878     id_to_attr[ROTQBY___].pipe    = PIPE_ODD;
879     id_to_attr[ROTQBY___].latency = 4;
880     id_to_attr[ROTQBY___].stall   = 0;
881     id_to_attr[ROTQBYI__].name    = "rotqbyi";
882     id_to_attr[ROTQBYI__].type    = IT_W_RT | IT_R_RA | IT_R_I7;
883     id_to_attr[ROTQBYI__].pipe    = PIPE_ODD;
884     id_to_attr[ROTQBYI__].latency = 4;
885     id_to_attr[ROTQBYI__].stall   = 0;
886     id_to_attr[ROTQBYBI_].name    = "rotqbybi";
887     id_to_attr[ROTQBYBI_].type    = IT_W_RT | IT_R_RA | IT_R_RB;
888     id_to_attr[ROTQBYBI_].pipe    = PIPE_ODD;
889     id_to_attr[ROTQBYBI_].latency = 4;
890     id_to_attr[ROTQBYBI_].stall   = 0;
891     id_to_attr[ROTQBI___].name    = "rotqbi";
892     id_to_attr[ROTQBI___].type    = IT_W_RT | IT_R_RA | IT_R_RB;
893     id_to_attr[ROTQBI___].pipe    = PIPE_ODD;
894     id_to_attr[ROTQBI___].latency = 4;
895     id_to_attr[ROTQBI___].stall   = 0;
896     id_to_attr[ROTQBII__].name    = "rotqbii";
897     id_to_attr[ROTQBII__].type    = IT_W_RT | IT_R_RA | IT_R_I7;
898     id_to_attr[ROTQBII__].pipe    = PIPE_ODD;
899     id_to_attr[ROTQBII__].latency = 4;
900     id_to_attr[ROTQBII__].stall   = 0;
901     id_to_attr[ROTHM____].name    = "rothm";
902     id_to_attr[ROTHM____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
903     id_to_attr[ROTHM____].pipe    = PIPE_EVEN;
904     id_to_attr[ROTHM____].latency = 4;
905     id_to_attr[ROTHM____].stall   = 0;
906     id_to_attr[ROTHMI___].name    = "rothmi";
907     id_to_attr[ROTHMI___].type    = IT_W_RT | IT_R_RA | IT_R_I7;
908     id_to_attr[ROTHMI___].pipe    = PIPE_EVEN;
909     id_to_attr[ROTHMI___].latency = 4;
910     id_to_attr[ROTHMI___].stall   = 0;
911     id_to_attr[ROTM_____].name    = "rotm";
912     id_to_attr[ROTM_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
913     id_to_attr[ROTM_____].pipe    = PIPE_EVEN;
914     id_to_attr[ROTM_____].latency = 4;
915     id_to_attr[ROTM_____].stall   = 0;
916     id_to_attr[ROTMI____].name    = "rotmi";
917     id_to_attr[ROTMI____].type    = IT_W_RT | IT_R_RA | IT_R_I7;
918     id_to_attr[ROTMI____].pipe    = PIPE_EVEN;
919     id_to_attr[ROTMI____].latency = 4;
920     id_to_attr[ROTMI____].stall   = 0;
921     id_to_attr[ROTQMBY__].name    = "rotqmby";
922     id_to_attr[ROTQMBY__].type    = IT_W_RT | IT_R_RA | IT_R_RB;
923     id_to_attr[ROTQMBY__].pipe    = PIPE_ODD;
```

```
924       id_to_attr[ROTQMBY__].latency = 4;
925       id_to_attr[ROTQMBY__].stall   = 0;
926       id_to_attr[ROTQMBYI_].name    = "rotqmbyi";
927       id_to_attr[ROTQMBYI_].type    = IT_W_RT | IT_R_RA | IT_R_I7;
928       id_to_attr[ROTQMBYI_].pipe    = PIPE_ODD;
929       id_to_attr[ROTQMBYI_].latency = 4;
930       id_to_attr[ROTQMBYI_].stall   = 0;
931       id_to_attr[ROTQMBYBI].name    = "rotqmbybi";
932       id_to_attr[ROTQMBYBI].type    = IT_W_RT | IT_R_RA | IT_R_RB;
933       id_to_attr[ROTQMBYBI].pipe    = PIPE_ODD;
934       id_to_attr[ROTQMBYBI].latency = 4;
935       id_to_attr[ROTQMBYBI].stall   = 0;
936       id_to_attr[ROTQMBI__].name    = "rotqmbi";
937       id_to_attr[ROTQMBI__].type    = IT_W_RT | IT_R_RA | IT_R_RB;
938       id_to_attr[ROTQMBI__].pipe    = PIPE_ODD;
939       id_to_attr[ROTQMBI__].latency = 4;
940       id_to_attr[ROTQMBI__].stall   = 0;
941       id_to_attr[ROTQMBII_].name    = "rotqmbii";
942       id_to_attr[ROTQMBII_].type    = IT_W_RT | IT_R_RA | IT_R_I7;
943       id_to_attr[ROTQMBII_].pipe    = PIPE_ODD;
944       id_to_attr[ROTQMBII_].latency = 4;
945       id_to_attr[ROTQMBII_].stall   = 0;
946       id_to_attr[ROTMAH___].name    = "rotmah";
947       id_to_attr[ROTMAH___].type    = IT_W_RT | IT_R_RA | IT_R_RB;
948       id_to_attr[ROTMAH___].pipe    = PIPE_EVEN;
949       id_to_attr[ROTMAH___].latency = 4;
950       id_to_attr[ROTMAH___].stall   = 0;
951       id_to_attr[ROTMAHI__].name    = "rotmahi";
952       id_to_attr[ROTMAHI__].type    = IT_W_RT | IT_R_RA | IT_R_I7;
953       id_to_attr[ROTMAHI__].pipe    = PIPE_EVEN;
954       id_to_attr[ROTMAHI__].latency = 4;
955       id_to_attr[ROTMAHI__].stall   = 0;
956       id_to_attr[ROTMA____].name    = "rotma";
957       id_to_attr[ROTMA____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
958       id_to_attr[ROTMA____].pipe    = PIPE_EVEN;
959       id_to_attr[ROTMA____].latency = 4;
960       id_to_attr[ROTMA____].stall   = 0;
961       id_to_attr[ROTMAI___].name    = "rotmai";
962       id_to_attr[ROTMAI___].type    = IT_W_RT | IT_R_RA | IT_R_I7;
963       id_to_attr[ROTMAI___].pipe    = PIPE_EVEN;
964       id_to_attr[ROTMAI___].latency = 4;
965       id_to_attr[ROTMAI___].stall   = 0;
966       id_to_attr[HEQ_____].name     = "heq";
967       id_to_attr[HEQ_____].type     = IT_R_RA | IT_R_RB;
968       id_to_attr[HEQ_____].pipe     = PIPE_EVEN;
969       id_to_attr[HEQ_____].latency  = 0;
970       id_to_attr[HEQ_____].stall    = 0;
971       id_to_attr[HEQI____].name     = "heqi";
972       id_to_attr[HEQI____].type     = IT_R_RA | IT_R_I10;
973       id_to_attr[HEQI____].pipe     = PIPE_EVEN;
974       id_to_attr[HEQI____].latency  = 0;
975       id_to_attr[HEQI____].stall    = 0;
976       id_to_attr[HGT_____].name     = "hgt";
977       id_to_attr[HGT_____].type     = IT_R_RA | IT_R_RB;
978       id_to_attr[HGT_____].pipe     = PIPE_EVEN;
979       id_to_attr[HGT_____].latency  = 0;
980       id_to_attr[HGT_____].stall    = 0;
981       id_to_attr[HGTI____].name     = "hgti";
982       id_to_attr[HGTI____].type     = IT_R_RA | IT_R_I10;
983       id_to_attr[HGTI____].pipe     = PIPE_EVEN;
984       id_to_attr[HGTI____].latency  = 0;
985       id_to_attr[HGTI____].stall    = 0;
986       id_to_attr[HLGT____].name     = "hlgt";
987       id_to_attr[HLGT____].type     = IT_R_RA | IT_R_RB;
988       id_to_attr[HLGT____].pipe     = PIPE_EVEN;
989       id_to_attr[HLGT____].latency  = 0;
990       id_to_attr[HLGT____].stall    = 0;
991       id_to_attr[HLGTI___].name     = "hlgti";
992       id_to_attr[HLGTI___].type     = IT_R_RA | IT_R_I10;
993       id_to_attr[HLGTI___].pipe     = PIPE_EVEN;
994       id_to_attr[HLGTI___].latency  = 0;
995       id_to_attr[HLGTI___].stall    = 0;
996       id_to_attr[CEQB____].name     = "ceqb";
997       id_to_attr[CEQB____].type     = IT_W_RT | IT_R_RA | IT_R_RB;
998       id_to_attr[CEQB____].pipe     = PIPE_EVEN;
999       id_to_attr[CEQB____].latency  = 2;
1000      id_to_attr[CEQB____].stall    = 0;
```



```
1001      id_to_attr[CEQBI___].name     = "ceqbi";
1002      id_to_attr[CEQBI___].type     = IT_W_RT | IT_R_RA | IT_R_I10;
1003      id_to_attr[CEQBI___].pipe     = PIPE_EVEN;
1004      id_to_attr[CEQBI___].latency  = 2;
1005      id_to_attr[CEQBI___].stall    = 0;
1006      id_to_attr[CEQH____].name     = "ceqh";
1007      id_to_attr[CEQH____].type     = IT_W_RT | IT_R_RA | IT_R_RB;
1008      id_to_attr[CEQH____].pipe     = PIPE_EVEN;
1009      id_to_attr[CEQH____].latency  = 2;
1010      id_to_attr[CEQH____].stall    = 0;
1011      id_to_attr[CEQHI___].name     = "ceqhi";
1012      id_to_attr[CEQHI___].type     = IT_W_RT | IT_R_RA | IT_R_I10;
1013      id_to_attr[CEQHI___].pipe     = PIPE_EVEN;
1014      id_to_attr[CEQHI___].latency  = 2;
1015      id_to_attr[CEQHI___].stall    = 0;
1016      id_to_attr[CEQ_____].name     = "ceq";
1017      id_to_attr[CEQ_____].type     = IT_W_RT | IT_R_RA | IT_R_RB;
1018      id_to_attr[CEQ_____].pipe     = PIPE_EVEN;
1019      id_to_attr[CEQ_____].latency  = 2;
1020      id_to_attr[CEQ_____].stall    = 0;
1021      id_to_attr[CEQI____].name     = "ceqi";
1022      id_to_attr[CEQI____].type     = IT_W_RT | IT_R_RA | IT_R_I10;
1023      id_to_attr[CEQI____].pipe     = PIPE_EVEN;
1024      id_to_attr[CEQI____].latency  = 2;
1025      id_to_attr[CEQI____].stall    = 0;
1026      id_to_attr[CGTB____].name     = "cgtb";
1027      id_to_attr[CGTB____].type     = IT_W_RT | IT_R_RA | IT_R_RB;
1028      id_to_attr[CGTB____].pipe     = PIPE_EVEN;
1029      id_to_attr[CGTB____].latency  = 2;
1030      id_to_attr[CGTB____].stall    = 0;
1031      id_to_attr[CGTBI___].name     = "cgtbi";
1032      id_to_attr[CGTBI___].type     = IT_W_RT | IT_R_RA | IT_R_I10;
1033      id_to_attr[CGTBI___].pipe     = PIPE_EVEN;
1034      id_to_attr[CGTBI___].latency  = 2;
1035      id_to_attr[CGTBI___].stall    = 0;
1036      id_to_attr[CGTH____].name     = "cgth";
1037      id_to_attr[CGTH____].type     = IT_W_RT | IT_R_RA | IT_R_RB;
1038      id_to_attr[CGTH____].pipe     = PIPE_EVEN;
1039      id_to_attr[CGTH____].latency  = 2;
1040      id_to_attr[CGTH____].stall    = 0;
1041      id_to_attr[CGTHI___].name     = "cgthi";
1042      id_to_attr[CGTHI___].type     = IT_W_RT | IT_R_RA | IT_R_I10;
1043      id_to_attr[CGTHI___].pipe     = PIPE_EVEN;
1044      id_to_attr[CGTHI___].latency  = 2;
1045      id_to_attr[CGTHI___].stall    = 0;
1046      id_to_attr[CGT_____].name     = "cgt";
1047      id_to_attr[CGT_____].type     = IT_W_RT | IT_R_RA | IT_R_RB;
1048      id_to_attr[CGT_____].pipe     = PIPE_EVEN;
1049      id_to_attr[CGT_____].latency  = 2;
1050      id_to_attr[CGT_____].stall    = 0;
1051      id_to_attr[CGTI____].name     = "cgti";
1052      id_to_attr[CGTI____].type     = IT_W_RT | IT_R_RA | IT_R_I10;
1053      id_to_attr[CGTI____].pipe     = PIPE_EVEN;
1054      id_to_attr[CGTI____].latency  = 2;
1055      id_to_attr[CGTI____].stall    = 0;
1056      id_to_attr[CLGTB___].name     = "clgtb";
1057      id_to_attr[CLGTB___].type     = IT_W_RT | IT_R_RA | IT_R_RB;
1058      id_to_attr[CLGTB___].pipe     = PIPE_EVEN;
1059      id_to_attr[CLGTB___].latency  = 2;
1060      id_to_attr[CLGTB___].stall    = 0;
1061      id_to_attr[CLGTBI__].name     = "clgtbi";
1062      id_to_attr[CLGTBI__].type     = IT_W_RT | IT_R_RA | IT_R_I10;
1063      id_to_attr[CLGTBI__].pipe     = PIPE_EVEN;
1064      id_to_attr[CLGTBI__].latency  = 2;
1065      id_to_attr[CLGTBI__].stall    = 0;
1066      id_to_attr[CLGTH___].name     = "clgth";
1067      id_to_attr[CLGTH___].type     = IT_W_RT | IT_R_RA | IT_R_RB;
1068      id_to_attr[CLGTH___].pipe     = PIPE_EVEN;
1069      id_to_attr[CLGTH___].latency  = 2;
1070      id_to_attr[CLGTH___].stall    = 0;
1071      id_to_attr[CLGTHI__].name     = "clgthi";
1072      id_to_attr[CLGTHI__].type     = IT_W_RT | IT_R_RA | IT_R_I10;
1073      id_to_attr[CLGTHI__].pipe     = PIPE_EVEN;
1074      id_to_attr[CLGTHI__].latency  = 2;
1075      id_to_attr[CLGTHI__].stall    = 0;
1076      id_to_attr[CLGT____].name     = "clgt";
1077      id_to_attr[CLGT____].type     = IT_W_RT | IT_R_RA | IT_R_RB;
```

```
1078      id_to_attr[CLGT_____].pipe    = PIPE_EVEN;
1079      id_to_attr[CLGT_____].latency = 2;
1080      id_to_attr[CLGT_____].stall   = 0;
1081      id_to_attr[CLGTI____].name    = "clgti";
1082      id_to_attr[CLGTI____].type    = IT_W_RT | IT_R_RA | IT_R_I10;
1083      id_to_attr[CLGTI____].pipe    = PIPE_EVEN;
1084      id_to_attr[CLGTI____].latency = 2;
1085      id_to_attr[CLGTI____].stall   = 0;
1086      id_to_attr[BR_______].name    = "br";
1087      id_to_attr[BR_______].type    = IT_R_I16;
1088      id_to_attr[BR_______].pipe    = PIPE_ODD;
1089      id_to_attr[BR_______].latency = 0;
1090      id_to_attr[BR_______].stall   = 0;
1091      id_to_attr[BRA______].name    = "bra";
1092      id_to_attr[BRA______].type    = IT_R_I16;
1093      id_to_attr[BRA______].pipe    = PIPE_ODD;
1094      id_to_attr[BRA______].latency = 0;
1095      id_to_attr[BRA______].stall   = 0;
1096      id_to_attr[BRSL_____].name    = "brsl";
1097      id_to_attr[BRSL_____].type    = IT_W_RT | IT_R_I16;
1098      id_to_attr[BRSL_____].pipe    = PIPE_ODD;
1099      id_to_attr[BRSL_____].latency = 0;
1100      id_to_attr[BRSL_____].stall   = 0;
1101      id_to_attr[BRASL____].name    = "brasl";
1102      id_to_attr[BRASL____].type    = IT_W_RT | IT_R_I16;
1103      id_to_attr[BRASL____].pipe    = PIPE_ODD;
1104      id_to_attr[BRASL____].latency = 0;
1105      id_to_attr[BRASL____].stall   = 0;
1106      id_to_attr[BI_______].name    = "bi";
1107      id_to_attr[BI_______].type    = IT_R_RA;
1108      id_to_attr[BI_______].pipe    = PIPE_ODD;
1109      id_to_attr[BI_______].latency = 0;
1110      id_to_attr[BI_______].stall   = 0;
1111      id_to_attr[IRET_____].name    = "iret";
1112      id_to_attr[IRET_____].type    = IT_R_RA;
1113      id_to_attr[IRET_____].pipe    = PIPE_ODD;
1114      id_to_attr[IRET_____].latency = 0;
1115      id_to_attr[IRET_____].stall   = 0;
1116      id_to_attr[BISLED___].name    = "bisled";
1117      id_to_attr[BISLED___].type    = IT_W_RT | IT_R_RA;
1118      id_to_attr[BISLED___].pipe    = PIPE_ODD;
1119      id_to_attr[BISLED___].latency = 4;
1120      id_to_attr[BISLED___].stall   = 0;
1121      id_to_attr[BISL_____].name    = "bisl";
1122      id_to_attr[BISL_____].type    = IT_W_RT | IT_R_RA;
1123      id_to_attr[BISL_____].pipe    = PIPE_ODD;
1124      id_to_attr[BISL_____].latency = 4;
1125      id_to_attr[BISL_____].stall   = 0;
1126      id_to_attr[BRNZ_____].name    = "brnz";
1127      id_to_attr[BRNZ_____].type    = IT_R_RT | IT_R_I16;
1128      id_to_attr[BRNZ_____].pipe    = PIPE_ODD;
1129      id_to_attr[BRNZ_____].latency = 0;
1130      id_to_attr[BRNZ_____].stall   = 0;
1131      id_to_attr[BRZ______].name    = "brz";
1132      id_to_attr[BRZ______].type    = IT_R_RT | IT_R_I16;
1133      id_to_attr[BRZ______].pipe    = PIPE_ODD;
1134      id_to_attr[BRZ______].latency = 0;
1135      id_to_attr[BRZ______].stall   = 0;
1136      id_to_attr[BRHNZ____].name    = "brhnz";
1137      id_to_attr[BRHNZ____].type    = IT_R_RT | IT_R_I16;
1138      id_to_attr[BRHNZ____].pipe    = PIPE_ODD;
1139      id_to_attr[BRHNZ____].latency = 0;
1140      id_to_attr[BRHNZ____].stall   = 0;
1141      id_to_attr[BRHZ_____].name    = "brhz";
1142      id_to_attr[BRHZ_____].type    = IT_R_RT | IT_R_I16;
1143      id_to_attr[BRHZ_____].pipe    = PIPE_ODD;
1144      id_to_attr[BRHZ_____].latency = 0;
1145      id_to_attr[BRHZ_____].stall   = 0;
1146      id_to_attr[BIZ______].name    = "biz";
1147      id_to_attr[BIZ______].type    = IT_R_RA | IT_R_RT;;
1148      id_to_attr[BIZ______].pipe    = PIPE_ODD;
1149      id_to_attr[BIZ______].latency = 0;
1150      id_to_attr[BIZ______].stall   = 0;
1151      id_to_attr[BINZ_____].name    = "binz";
1152      id_to_attr[BINZ_____].type    = IT_R_RA | IT_R_RT;;
1153      id_to_attr[BINZ_____].pipe    = PIPE_ODD;
1154      id_to_attr[BINZ_____].latency = 0;
```



```
1155      id_to_attr[BINZ_____].stall   = 0;
1156      id_to_attr[BIHZ_____].name    = "bihz";
1157      id_to_attr[BIHZ_____].type    = IT_R_RA | IT_R_RT;;
1158      id_to_attr[BIHZ_____].pipe    = PIPE_ODD;
1159      id_to_attr[BIHZ_____].latency = 0;
1160      id_to_attr[BIHZ_____].stall   = 0;
1161      id_to_attr[BIHNZ____].name    = "bihnz";
1162      id_to_attr[BIHNZ____].type    = IT_R_RA | IT_R_RT;;
1163      id_to_attr[BIHNZ____].pipe    = PIPE_ODD;
1164      id_to_attr[BIHNZ____].latency = 0;
1165      id_to_attr[BIHNZ____].stall   = 0;
1166      id_to_attr[HBR______].name    = "hbr";
1167      id_to_attr[HBR______].type    = IT_R_RA | IT_R_RO1;
1168      id_to_attr[HBR______].pipe    = PIPE_ODD;
1169      id_to_attr[HBR______].latency = 15;
1170      id_to_attr[HBR______].stall   = 0;
1171      id_to_attr[HBRA_____].name    = "hbra";
1172      id_to_attr[HBRA_____].type    = IT_R_I16 | IT_R_RO2;
1173      id_to_attr[HBRA_____].pipe    = PIPE_ODD;
1174      id_to_attr[HBRA_____].latency = 15;
1175      id_to_attr[HBRA_____].stall   = 0;
1176      id_to_attr[HBRR_____].name    = "hbrr";
1177      id_to_attr[HBRR_____].type    = IT_R_I16 | IT_R_RO2;
1178      id_to_attr[HBRR_____].pipe    = PIPE_ODD;
1179      id_to_attr[HBRR_____].latency = 15;
1180      id_to_attr[HBRR_____].stall   = 0;
1181      id_to_attr[FA_______].name    = "fa";
1182      id_to_attr[FA_______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1183      id_to_attr[FA_______].pipe    = PIPE_EVEN;
1184      id_to_attr[FA_______].latency = 6;
1185      id_to_attr[FA_______].stall   = 0;
1186      id_to_attr[DFA______].name    = "dfa";
1187      id_to_attr[DFA______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1188      id_to_attr[DFA______].pipe    = PIPE_EVEN;
1189      id_to_attr[DFA______].latency = 7;
1190      id_to_attr[DFA______].stall   = 6;
1191      id_to_attr[FS_______].name    = "fs";
1192      id_to_attr[FS_______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1193      id_to_attr[FS_______].pipe    = PIPE_EVEN;
1194      id_to_attr[FS_______].latency = 6;
1195      id_to_attr[FS_______].stall   = 0;
1196      id_to_attr[DFS______].name    = "dfs";
1197      id_to_attr[DFS______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1198      id_to_attr[DFS______].pipe    = PIPE_EVEN;
1199      id_to_attr[DFS______].latency = 7;
1200      id_to_attr[DFS______].stall   = 6;
1201      id_to_attr[FM_______].name    = "fm";
1202      id_to_attr[FM_______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1203      id_to_attr[FM_______].pipe    = PIPE_EVEN;
1204      id_to_attr[FM_______].latency = 6;
1205      id_to_attr[FM_______].stall   = 0;
1206      id_to_attr[DFM______].name    = "dfm";
1207      id_to_attr[DFM______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1208      id_to_attr[DFM______].pipe    = PIPE_EVEN;
1209      id_to_attr[DFM______].latency = 7;
1210      id_to_attr[DFM______].stall   = 6;
1211      id_to_attr[FMA______].name    = "fma";
1212      id_to_attr[FMA______].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RC;
1213      id_to_attr[FMA______].pipe    = PIPE_EVEN;
1214      id_to_attr[FMA______].latency = 6;
1215      id_to_attr[FMA______].stall   = 0;
1216      id_to_attr[DFMA_____].name    = "dfma";
1217      id_to_attr[DFMA_____].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RT;
1218      id_to_attr[DFMA_____].pipe    = PIPE_EVEN;
1219      id_to_attr[DFMA_____].latency = 7;
1220      id_to_attr[DFMA_____].stall   = 6;
1221      id_to_attr[FNMS_____].name    = "fnms";
1222      id_to_attr[FNMS_____].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RC;
1223      id_to_attr[FNMS_____].pipe    = PIPE_EVEN;
1224      id_to_attr[FNMS_____].latency = 6;
1225      id_to_attr[FNMS_____].stall   = 0;
1226      id_to_attr[DFNMS____].name    = "dfnms";
1227      id_to_attr[DFNMS____].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RT;
1228      id_to_attr[DFNMS____].pipe    = PIPE_EVEN;
1229      id_to_attr[DFNMS____].latency = 7;
1230      id_to_attr[DFNMS____].stall   = 6;
1231      id_to_attr[FMS______].name    = "fms";
```

```
1232      id_to_attr[FMS_______].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RC;
1233      id_to_attr[FMS_______].pipe    = PIPE_EVEN;
1234      id_to_attr[FMS_______].latency = 6;
1235      id_to_attr[FMS_______].stall   = 0;
1236      id_to_attr[DFMS______].name    = "dfms";
1237      id_to_attr[DFMS______].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RT;
1238      id_to_attr[DFMS______].pipe    = PIPE_EVEN;
1239      id_to_attr[DFMS______].latency = 7;
1240      id_to_attr[DFMS______].stall   = 6;
1241      id_to_attr[DFNMA_____].name    = "dfnma";
1242      id_to_attr[DFNMA_____].type    = IT_W_RT | IT_R_RA | IT_R_RB | IT_R_RT;
1243      id_to_attr[DFNMA_____].pipe    = PIPE_EVEN;
1244      id_to_attr[DFNMA_____].latency = 7;
1245      id_to_attr[DFNMA_____].stall   = 6;
1246      id_to_attr[FREST_____].name    = "frest";
1247      id_to_attr[FREST_____].type    = IT_W_RT | IT_R_RA;
1248      id_to_attr[FREST_____].pipe    = PIPE_ODD;
1249      id_to_attr[FREST_____].latency = 4;
1250      id_to_attr[FREST_____].stall   = 0;
1251      id_to_attr[FRSQEST___].name    = "frsqest";
1252      id_to_attr[FRSQEST___].type    = IT_W_RT | IT_R_RA;
1253      id_to_attr[FRSQEST___].pipe    = PIPE_ODD;
1254      id_to_attr[FRSQEST___].latency = 4;
1255      id_to_attr[FRSQEST___].stall   = 0;
1256      id_to_attr[FI________].name    = "fi";
1257      id_to_attr[FI________].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1258      id_to_attr[FI________].pipe    = PIPE_EVEN;
1259      id_to_attr[FI________].latency = 7;
1260      id_to_attr[FI________].stall   = 0;
1261      id_to_attr[CSFLT_____].name    = "csflt";
1262      id_to_attr[CSFLT_____].type    = IT_W_RT | IT_R_RA | IT_R_I8;
1263      id_to_attr[CSFLT_____].pipe    = PIPE_EVEN;
1264      id_to_attr[CSFLT_____].latency = 7;
1265      id_to_attr[CSFLT_____].stall   = 0;
1266      id_to_attr[CFLTS_____].name    = "cflts";
1267      id_to_attr[CFLTS_____].type    = IT_W_RT | IT_R_RA | IT_R_I8;
1268      id_to_attr[CFLTS_____].pipe    = PIPE_EVEN;
1269      id_to_attr[CFLTS_____].latency = 7;
1270      id_to_attr[CFLTS_____].stall   = 0;
1271      id_to_attr[CUFLT_____].name    = "cuflt";
1272      id_to_attr[CUFLT_____].type    = IT_W_RT | IT_R_RA | IT_R_I8;
1273      id_to_attr[CUFLT_____].pipe    = PIPE_EVEN;
1274      id_to_attr[CUFLT_____].latency = 7;
1275      id_to_attr[CUFLT_____].stall   = 0;
1276      id_to_attr[CFLTU_____].name    = "cfltu";
1277      id_to_attr[CFLTU_____].type    = IT_W_RT | IT_R_RA | IT_R_I8;
1278      id_to_attr[CFLTU_____].pipe    = PIPE_EVEN;
1279      id_to_attr[CFLTU_____].latency = 7;
1280      id_to_attr[CFLTU_____].stall   = 0;
1281      id_to_attr[FRDS______].name    = "frds";
1282      id_to_attr[FRDS______].type    = IT_W_RT | IT_R_RA;
1283      id_to_attr[FRDS______].pipe    = PIPE_EVEN;
1284      id_to_attr[FRDS______].latency = 7;
1285      id_to_attr[FRDS______].stall   = 6;
1286      id_to_attr[FESD______].name    = "fesd";
1287      id_to_attr[FESD______].type    = IT_W_RT | IT_R_RA;
1288      id_to_attr[FESD______].pipe    = PIPE_EVEN;
1289      id_to_attr[FESD______].latency = 7;
1290      id_to_attr[FESD______].stall   = 6;
1291      id_to_attr[DFCEQ_____].name    = "dfceq";
1292      id_to_attr[DFCEQ_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1293      id_to_attr[DFCEQ_____].pipe    = PIPE_EVEN;
1294      id_to_attr[DFCEQ_____].latency = 7;
1295      id_to_attr[DFCEQ_____].stall   = 6;
1296      id_to_attr[DFCMEQ____].name    = "dfcmeq";
1297      id_to_attr[DFCMEQ____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1298      id_to_attr[DFCMEQ____].pipe    = PIPE_EVEN;
1299      id_to_attr[DFCMEQ____].latency = 7;
1300      id_to_attr[DFCMEQ____].stall   = 6;
1301      id_to_attr[DFCGT_____].name    = "dfcgt";
1302      id_to_attr[DFCGT_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1303      id_to_attr[DFCGT_____].pipe    = PIPE_EVEN;
1304      id_to_attr[DFCGT_____].latency = 7;
1305      id_to_attr[DFCGT_____].stall   = 6;
1306      id_to_attr[DFCMGT____].name    = "dfcmgt";
1307      id_to_attr[DFCMGT____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1308      id_to_attr[DFCMGT____].pipe    = PIPE_EVEN;
```



```
1309      id_to_attr[DFCMGT____].latency = 7;
1310      id_to_attr[DFCMGT____].stall   = 6;
1311      id_to_attr[DFTSV_____].name    = "dftsv";
1312      id_to_attr[DFTSV_____].type    = IT_W_RT | IT_R_RA | IT_R_I7;
1313      id_to_attr[DFTSV_____].pipe    = PIPE_EVEN;
1314      id_to_attr[DFTSV_____].latency = 7;
1315      id_to_attr[DFTSV_____].stall   = 6;
1316      id_to_attr[FCEQ______].name    = "fceq";
1317      id_to_attr[FCEQ______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1318      id_to_attr[FCEQ______].pipe    = PIPE_EVEN;
1319      id_to_attr[FCEQ______].latency = 2;
1320      id_to_attr[FCEQ______].stall   = 0;
1321      id_to_attr[FCMEQ_____].name    = "fcmeq";
1322      id_to_attr[FCMEQ_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1323      id_to_attr[FCMEQ_____].pipe    = PIPE_EVEN;
1324      id_to_attr[FCMEQ_____].latency = 2;
1325      id_to_attr[FCMEQ_____].stall   = 0;
1326      id_to_attr[FCGT______].name    = "fcgt";
1327      id_to_attr[FCGT______].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1328      id_to_attr[FCGT______].pipe    = PIPE_EVEN;
1329      id_to_attr[FCGT______].latency = 2;
1330      id_to_attr[FCGT______].stall   = 0;
1331      id_to_attr[FCMGT_____].name    = "fcmgt";
1332      id_to_attr[FCMGT_____].type    = IT_W_RT | IT_R_RA | IT_R_RB;
1333      id_to_attr[FCMGT_____].pipe    = PIPE_EVEN;
1334      id_to_attr[FCMGT_____].latency = 2;
1335      id_to_attr[FCMGT_____].stall   = 0;
1336      id_to_attr[FSCRWR____].name    = "fscrwr";
1337      id_to_attr[FSCRWR____].type    = IT_R_RA;
1338      id_to_attr[FSCRWR____].pipe    = PIPE_EVEN;
1339      id_to_attr[FSCRWR____].latency = 7;
1340      id_to_attr[FSCRWR____].stall   = 0;
1341      id_to_attr[FSCRRD____].name    = "fscrrd";
1342      id_to_attr[FSCRRD____].type    = IT_W_RT;
1343      id_to_attr[FSCRRD____].pipe    = PIPE_EVEN;
1344      id_to_attr[FSCRRD____].latency = 7;
1345      id_to_attr[FSCRRD____].stall   = 6;
1346      id_to_attr[STOP______].name    = "stop";
1347      id_to_attr[STOP______].type    = IT_NONE;
1348      id_to_attr[STOP______].pipe    = PIPE_ODD;
1349      id_to_attr[STOP______].latency = 4;
1350      id_to_attr[STOP______].stall   = 0;
1351      id_to_attr[STOPD_____].name    = "stopd";
1352      id_to_attr[STOPD_____].type    = IT_NONE;
1353      id_to_attr[STOPD_____].pipe    = PIPE_ODD;
1354      id_to_attr[STOPD_____].latency = 4;
1355      id_to_attr[STOPD_____].stall   = 0;
1356      id_to_attr[LNOP______].name    = "lnop";
1357      id_to_attr[LNOP______].type    = IT_NONE;
1358      id_to_attr[LNOP______].pipe    = PIPE_ODD;
1359      id_to_attr[LNOP______].latency = 0;
1360      id_to_attr[LNOP______].stall   = 0;
1361      id_to_attr[NOP_______].name    = "nop";
1362      id_to_attr[NOP_______].type    = IT_NONE;
1363      id_to_attr[NOP_______].pipe    = PIPE_EVEN;
1364      id_to_attr[NOP_______].latency = 0;
1365      id_to_attr[NOP_______].stall   = 0;
1366      id_to_attr[SYNC______].name    = "sync";
1367      id_to_attr[SYNC______].type    = IT_NONE;
1368      id_to_attr[SYNC______].pipe    = PIPE_ODD;
1369      id_to_attr[SYNC______].latency = 0;
1370      id_to_attr[SYNC______].stall   = 0;
1371      id_to_attr[DSYNC_____].name    = "dsync";
1372      id_to_attr[DSYNC_____].type    = IT_NONE;
1373      id_to_attr[DSYNC_____].pipe    = PIPE_ODD;
1374      id_to_attr[DSYNC_____].latency = 0;
1375      id_to_attr[DSYNC_____].stall   = 0;
1376      id_to_attr[MFSPR_____].name    = "mfspr";
1377      id_to_attr[MFSPR_____].type    = IT_W_RT;
1378      id_to_attr[MFSPR_____].pipe    = PIPE_ODD;
1379      id_to_attr[MFSPR_____].latency = 6;
1380      id_to_attr[MFSPR_____].stall   = 0;
1381      id_to_attr[MTSPR_____].name    = "mtspr";
1382      id_to_attr[MTSPR_____].type    = IT_R_RT;
1383      id_to_attr[MTSPR_____].pipe    = PIPE_ODD;
1384      id_to_attr[MTSPR_____].latency = 6;
1385      id_to_attr[MTSPR_____].stall   = 0;
```

```
1386      id_to_attr[RDCH_____].name    = "rdch";
1387      id_to_attr[RDCH_____].type    = IT_W_RT | IT_R_CA;
1388      id_to_attr[RDCH_____].pipe    = PIPE_ODD;
1389      id_to_attr[RDCH_____].latency = 6;
1390      id_to_attr[RDCH_____].stall   = 0;
1391      id_to_attr[WRCH_____].name    = "wrch";
1392      id_to_attr[WRCH_____].type    = IT_R_RT | IT_R_CA;
1393      id_to_attr[WRCH_____].pipe    = PIPE_ODD;
1394      id_to_attr[WRCH_____].latency = 0;
1395      id_to_attr[WRCH_____].stall   = 0;
1396      id_to_attr[RCHCNT___].name    = "rchcnt";
1397      id_to_attr[RCHCNT___].type    = IT_W_RT | IT_R_CA;
1398      id_to_attr[RCHCNT___].pipe    = PIPE_ODD;
1399      id_to_attr[RCHCNT___].latency = 6;
1400      id_to_attr[RCHCNT___].stall   = 0;
1401  }
1402
1403  /*************************************************************************/
1404  int SpuInstinfo::getinstid(int op)
1405  {
1406      return op_to_id[op];
1407  }
1408
1409  /*************************************************************************/
1410  spuinstattr_t SpuInstinfo::getinstattr(int id)
1411  {
1412      return id_to_attr[id];
1413  }
1414
1415  /*************************************************************************/
1416  char *SpuInstinfo::getopname(int id)
1417  {
1418      return id_to_attr[id].name;
1419  }
1420
1421  /*************************************************************************/
1422  SpuInstinfo::SpuInstinfo(Chip *chip)
1423  {
1424      chip->iinfo = this;
1425      for (uint i = 0; i < IINFO_OP_NUM; i++)
1426          op_to_id[i] = predecode(i);
1427      for (uint i = 0; i < IINFO_ID_NUM; i++) {
1428          id_to_attr[i].type = 0;
1429          id_to_attr[i].pipe = 0;
1430          id_to_attr[i].latency = 0;
1431          id_to_attr[i].stall = 0;
1432          id_to_attr[i].name = "undefined";
1433      }
1434      setattr();
1435  }
1436
1437  /*************************************************************************/
```


file name: eib.cc

```
   1  /*************************************************************************/
   2  /*  SimCell: Cell/B.E. Processor Simulator       Arch Lab. TOKYO TECH */
   3  /*************************************************************************/
   4  #include "define.h"
   5
   6  /*************************************************************************/
   7  enum {
   8      CH00_SPU_RDEVENTSTAT____ = 0,
   9      CH01_SPU_WREVENTMASK____ = 1,
  10      CH02_SPU_WREVENTACK_____ = 2,
  11      CH03_SPU_RDSIGNOTIFY1___ = 3,
  12      CH04_SPU_RDSIGNOTIFY2___ = 4,
  13      CH05_RESERVED___________ = 5,
  14      CH06_RESERVED___________ = 6,
  15      CH07_SPU_WRDEC__________ = 7,
  16      CH08_SPU_RDDEC__________ = 8,
  17      CH09_SPU_WRMSSYNCREQ____ = 9,
  18      CH10_RESERVED___________ = 10,
  19      CH11_SPU_RDEVENTMASK____ = 11,
  20      CH12_SPU_RDTAGMASK______ = 12,
  21      CH13_SPU_RDMACHSTAT_____ = 13,
  22      CH14_SPU_WRSRR0_________ = 14,
  23      CH15_SPU_RDSRR0_________ = 15,
  24      CH16_MFC_LSA____________ = 16,
  25      CH17_MFC_EAH____________ = 17,
  26      CH18_MFC_EAL____________ = 18,
  27      CH19_MFC_SIZE___________ = 19,
  28      CH20_MFC_TAGID__________ = 20,
  29      CH21_MFC_CMD____________ = 21,
  30      CH22_MFC_WRTAGMASK______ = 22,
  31      CH23_MFC_WRTAGUPDATE____ = 23,
  32      CH24_MFC_RDTAGSTAT______ = 24,
  33      CH25_MFC_RDLISTSTALLSTAT = 25,
  34      CH26_MFC_WRLISTSTALLACK_ = 26,
  35      CH27_MFC_RDATOMICSTAT___ = 27,
  36      CH28_SPU_WROUTMBOX______ = 28,
  37      CH29_SPU_RDINMBOX_______ = 29,
  38      CH30_SPU_WROUTINTRMBOX__ = 30,
  39
  40      CH_ENTRY01 = 1,
  41      CH_ENTRY04 = 4,
  42      CH_ENTRY16 = 16,
  43      CH_RD_ONLY = 0,
  44      CH_WR_ONLY = 1,
  45      CH_NONBLOCK = 0,
  46      CH_BLOCK = 1,
  47
  48      DMA_GET = 0x40,
  49      DMA_PUT = 0x20,
  50      DMA_SIZE_4BYTE = 4,
  51      DMA_PACKET_SIZE = 128,
  52      DMA_SIZE_MAX = 16384,
  53
  54      MMU_UNITID_UNDEF = -1,
  55      MMU_UNITID_MEM = 0,
  56
  57      MFC_CMD_NONE = 0,
  58      MFC_CMD_EXIST = 1,
  59
  60      DMA_NONE = -1,
  61      DMA_PEND = 0,
  62      DMA_WAIT = 1,
  63      DMA_RUN = 2,
  64      DMA_FINISH = 3,
  65
  66      TAG_UPDATE_IMM = 0,
  67      TAG_UPDATE_ANY = 1,
  68      TAG_UPDATE_ALL = 2,
  69      TAG_UPDATE_UNDEF = 4,
  70
  71      EIB_CMD_NOTACTIVE = 0,
  72      EIB_CMD_ACTIVE = 1,
  73
  74      ENQ_SUCCESS = 0,
  75      ENQ_FAIL = 1,
  76      DEQ_SUCCESS = 0,
```

```
77      DEQ_FAIL = 1,
78  };
79
80  /*************************************************************************/
81  Channel::Channel(int size, int mode, int type) :
82      size(size), blocking_mode(mode), channel_type(type)
83  {
84      message = new uint_032t[size];
85      head = 0;
86      tail = 0;
87      ch_count = (uint_032t) size;
88  }
89
90  /*************************************************************************/
91  Channel::~Channel()
92  {
93      delete[] message;
94  }
95
96  /*************************************************************************/
97  inline bool Channel::empty()
98  {
99      return (ch_count == size);
100 }
101
102 /*************************************************************************/
103 inline bool Channel::full()
104 {
105     return (ch_count == 0);
106 }
107
108 /*************************************************************************/
109 inline uint_032t Channel::capacity()
110 {
111     return ch_count;
112 }
113
114 /*************************************************************************/
115 inline uint_032t Channel::occupancy()
116 {
117     return (size - ch_count);
118 }
119
120 /*************************************************************************/
121 uint_032t Channel::read()
122 {
123     uint_032t data = message[head];
124     if (!empty()) {
125         head = (head + 1) % size;
126         ch_count++;
127     }
128     return data;
129 }
130
131 /*************************************************************************/
132 void Channel::write(uint_032t data)
133 {
134     message[tail] = data;
135     if (!full()) {
136         tail = (tail + 1) % size;
137         ch_count--;
138     }
139 }
140
141 /*************************************************************************/
142 ChannelInterface::ChannelInterface()
143 {
144     ch[CH00_SPU_RDEVENTSTAT____] =
145         new Channel(CH_ENTRY01, CH_BLOCK, CH_RD_ONLY);
146     ch[CH01_SPU_WREVENTMASK____] =
147         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_WR_ONLY);
148     ch[CH02_SPU_WREVENTACK_____] =
149         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_WR_ONLY);
150     ch[CH03_SPU_RDSIGNOTIFY1___] =
151         new Channel(CH_ENTRY01, CH_BLOCK, CH_RD_ONLY);
152     ch[CH04_SPU_RDSIGNOTIFY2___] =
153         new Channel(CH_ENTRY01, CH_BLOCK, CH_RD_ONLY);
```



```
154     ch[CH05_RESERVED___________] = NULL;
155     ch[CH06_RESERVED___________] = NULL;
156     ch[CH07_SPU_WRDEC__________] =
157         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_WR_ONLY);
158     ch[CH08_SPU_RDDEC__________] =
159         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_RD_ONLY);
160     ch[CH09_SPU_WRMSSYNCREQ____] =
161         new Channel(CH_ENTRY01, CH_BLOCK, CH_WR_ONLY);
162     ch[CH10_RESERVED___________] = NULL;
163     ch[CH11_SPU_RDEVENTMASK____] =
164         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_RD_ONLY);
165     ch[CH12_SPU_RDTAGMASK______] =
166         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_RD_ONLY);
167     ch[CH13_SPU_RDMACHSTAT_____] =
168         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_RD_ONLY);
169     ch[CH14_SPU_WRSRR0_________] =
170         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_WR_ONLY);
171     ch[CH15_SPU_RDSRR0_________] =
172         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_RD_ONLY);
173     ch[CH16_MFC_LSA____________] =
174         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_WR_ONLY);
175     ch[CH17_MFC_EAH____________] =
176         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_WR_ONLY);
177     ch[CH18_MFC_EAL____________] =
178         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_WR_ONLY);
179     ch[CH19_MFC_SIZE___________] =
180         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_WR_ONLY);
181     ch[CH20_MFC_TAGID__________] =
182         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_WR_ONLY);
183     ch[CH21_MFC_CMD____________] =
184         new Channel(CH_ENTRY16, CH_BLOCK, CH_WR_ONLY);
185     ch[CH22_MFC_WRTAGMASK______] =
186         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_WR_ONLY);
187     ch[CH23_MFC_WRTAGUPDATE____] =
188         new Channel(CH_ENTRY01, CH_BLOCK, CH_WR_ONLY);
189     ch[CH24_MFC_RDTAGSTAT______] =
190         new Channel(CH_ENTRY01, CH_BLOCK, CH_RD_ONLY);
191     ch[CH25_MFC_RDLISTSTALLSTAT] =
192         new Channel(CH_ENTRY01, CH_BLOCK, CH_RD_ONLY);
193     ch[CH26_MFC_WRLISTSTALLACK_] =
194         new Channel(CH_ENTRY01, CH_NONBLOCK, CH_WR_ONLY);
195     ch[CH27_MFC_RDATOMICSTAT___] =
196         new Channel(CH_ENTRY01, CH_BLOCK, CH_RD_ONLY);
197     ch[CH28_SPU_WROUTMBOX______] =
198         new Channel(CH_ENTRY01, CH_BLOCK, CH_WR_ONLY);
199     ch[CH29_SPU_RDINMBOX_______] =
200         new Channel(CH_ENTRY04, CH_BLOCK, CH_RD_ONLY);
201     ch[CH30_SPU_WROUTINTRMBOX__] =
202         new Channel(CH_ENTRY01, CH_BLOCK, CH_WR_ONLY);
203 }
204
205 /*************************************************************************/
206 ChannelInterface::~ChannelInterface()
207 {
208     for (int i = 0; i < CH_NUM; i++) {
209         delete ch[i];
210         ch[i] = NULL;
211     }
212 }
213
214 /*************************************************************************/
215 bool ChannelInterface::spureadable(int num)
216 {
217     if (ch[num] == NULL)
218         return 0;
219
220     return !(ch[num]->channel_type == CH_WR_ONLY ||
221              (ch[num]->blocking_mode == CH_BLOCK && ch[num]->empty()));
222 }
223
224 /*************************************************************************/
225 bool ChannelInterface::spuwritable(int num)
226 {
227     if (ch[num] == NULL)
228         return 0;
229
230     return !(ch[num]->channel_type == CH_RD_ONLY ||
```

```
231                 (ch[num]->blocking_mode == CH_BLOCK && ch[num]->full()));
232 }
233
234 /*****************************************************************/
235 int ChannelInterface::spureadch(int num, uint_032t * data)
236 {
237     if (!spureadable(num))
238         return 1;
239
240     *data = ch[num]->read();
241     return 0;
242 }
243
244 /*****************************************************************/
245 int ChannelInterface::spuwritech(int num, uint_032t data)
246 {
247     if (!spuwritable(num))
248         return 1;
249
250     ch[num]->write(data);
251     return 0;
252 }
253
254 /*****************************************************************/
255 uint_032t ChannelInterface::spureadchcnt(int num)
256 {
257     if (ch[num] == NULL)
258         return 0;
259     if (ch[num]->blocking_mode == CH_NONBLOCK)
260         return 1;
261     if (ch[num]->channel_type == CH_RD_ONLY)
262         return ch[num]->occupancy();
263     if (ch[num]->channel_type == CH_WR_ONLY)
264         return ch[num]->capacity();
265     return 0;
266 }
267
268 /*****************************************************************/
269 void ChannelInterface::mfcreadch(int num, uint_032t * data)
270 {
271     *data = ch[num]->read();
272 }
273
274 /*****************************************************************/
275 void ChannelInterface::mfcwritech(int num, uint_032t data)
276 {
277     ch[num]->write(data);
278 }
279
280 /*****************************************************************/
281 Mfc::Mfc(Chip * chip, Spe * spe)
282 {
283     this->chip = chip;
284     this->mmu = chip->mmu;
285     this->spe = spe;
286     this->chi = spe->chi;
287
288     head = 0;
289     tail = 0;
290     entry = 0;
291     for (int i = 0; i < MFC_MAX_ENTRY; i++)
292         cmdtag[i].state = DMA_NONE;
293     tagmask = 0;
294     tagupdate = TAG_UPDATE_UNDEF;
295     tagstat = 0;
296     decrementer = 0xffffffff;
297 }
298
299 /*****************************************************************/
300 inline dmacmd_t Mfc::getcmdfromchi()
301 {
302     dmacmd_t cmd;
303     uint_032t classid_cmd;
304
305     chi->mfcreadch(CH16_MFC_LSA____________, &cmd.lsa.physical);
306     chi->mfcreadch(CH18_MFC_EAL____________, &cmd.ea.logical);
307     chi->mfcreadch(CH19_MFC_SIZE____________, &cmd.size);
```



```
308     chi->mfcreadch(CH20_MFC_TAGID___________, &cmd.tagid);
309     chi->mfcreadch(CH21_MFC_CMD_____________, &classid_cmd);
310
311     cmd.classid = classid_cmd >> 16;
312     cmd.cmd = classid_cmd & 0xffff;
313     cmd.lsa.unitid = spe->unitid;
314     return cmd;
315 }
316
317 /*****************************************************************/
318 inline int Mfc::enq_spucmd(dmacmd_t cmd, dmacmdtag_t tag)
319 {
320     if (entry == MFC_MAX_ENTRY)
321         return ENQ_FAIL;
322     else {
323         mfcspucmdq[tail] = cmd;
324         cmdtag[tail] = tag;
325         entry++;
326         tail = (tail + 1) % MFC_MAX_ENTRY;
327         return ENQ_SUCCESS;
328     }
329 }
330
331 /*****************************************************************/
332 inline int Mfc::deq_spucmd()
333 {
334     if (entry == 0)
335         return DEQ_FAIL;
336     else {
337         cmdtag[head].state = DMA_NONE;
338         entry--;
339         head = (head + 1) % MFC_MAX_ENTRY;
340         return DEQ_SUCCESS;
341     }
342 }
343
344 /*****************************************************************/
345 inline bool Mfc::ismemread(dmacmd_t cmd)
346 {
347     return ((cmd.ea.unitid == MMU_UNITID_MEM)
348             && (cmd.cmd == DMA_GET));
349 }
350
351 /*****************************************************************/
352 inline bool Mfc::ismemwrite(dmacmd_t cmd)
353 {
354     return ((cmd.ea.unitid == MMU_UNITID_MEM)
355             && (cmd.cmd == DMA_PUT));
356 }
357
358 /*****************************************************************/
359 inline int Mfc::getdelay(dmacmd_t cmd)
360 {
361     int packet_num = (cmd.size / DMA_PACKET_SIZE);
362     if (ismemread(cmd))
363         return chip->mem_rddelay + packet_num * chip->mem_width;
364     else if (ismemwrite(cmd))
365         return chip->mem_wrdelay + packet_num * chip->mem_width;
366     else
367         return chip->c2c_delay + packet_num * chip->c2c_width;
368 }
369
370 /*****************************************************************/
371 inline void Mfc::makedmacmd()
372 {
373     dmacmd_t cmd = getcmdfromchi();
374     mmu->transaddr(&cmd.ea);
375     dmacmdtag_t tag;
376     tag.delay = getdelay(cmd);
377     tag.state = DMA_PEND;
378
379     enq_spucmd(cmd, tag);
380
381     if (chip->log_mode)
382         for (uint i = 0; i < cmd.size; i += 16)
383             chip->setlogdata(spe->unitid, cmd.lsa.physical + i, MEM_DMA);
384 }
```

```
385
386 /*****************************************************************/
387 inline uint_032t Mfc::maketagstat()
388 {
389     uint_032t stat = tagmask;
390     for (int i = 0; i < entry; i++) {
391         int index = (head + i) % MFC_MAX_ENTRY;
392         switch (cmdtag[index].state) {
393         case DMA_PEND:
394         case DMA_WAIT:
395         case DMA_RUN:
396             stat &= ~(1 << mfcspucmdq[index].tagid);
397             break;
398         case DMA_FINISH:
399         case DMA_NONE:
400             stat |= (1 << mfcspucmdq[index].tagid);
401             break;
402         default:
403             printf("## Mfc::maketagstat(), undefined TagStat.\n");
404             exit(1);
405         }
406     }
407     return (stat & tagmask);
408 }
409
410 /*****************************************************************/
411 inline void Mfc::checkchannel()
412 {
413     if (!chi->ch[CH07_SPU_WRDEC__________]->empty()) {
414         chi->mfcreadch(CH07_SPU_WRDEC__________, &decrementer);
415         chi->mfcwritech(CH08_SPU_RDDEC__________, decrementer);
416     }
417     if (!chi->ch[CH21_MFC_CMD____________]->empty() &&
418         entry < MFC_MAX_ENTRY)
419         makedmacmd();
420     if (!chi->ch[CH22_MFC_WRTAGMASK______]->empty())
421         chi->mfcreadch(CH22_MFC_WRTAGMASK______, &tagmask);
422     if (!chi->ch[CH23_MFC_WRTAGUPDATE____]->empty()) {
423         chi->mfcreadch(CH23_MFC_WRTAGUPDATE____, &tagupdate);
424     }
425 }
426
427 /*****************************************************************/
428 inline void Mfc::checkdma()
429 {
430     for (int i = 0, deqflag = 0; i < entry; i++) {
431         int index = (head + i) % MFC_MAX_ENTRY;
432         switch (cmdtag[index].state) {
433         case DMA_PEND:
434             if (chip->eib->enq_waitcmd(&mfcspucmdq[index], &cmdtag[index])
435                 == ENQ_SUCCESS)
436                 cmdtag[index].state = DMA_WAIT;
437             break;
438         case DMA_WAIT:
439             break;
440         case DMA_RUN:
441             cmdtag[index].delay--;
442             break;
443         case DMA_FINISH:
444             cmdtag[index].state = DMA_NONE;
445             break;
446         case DMA_NONE:
447             if (i == deqflag) {
448                 deq_spucmd();
449                 deqflag++;
450             }
451             break;
452         default:
453             printf("## Mfc::checkdma(), undefined TagStat.\n");
454             exit(1);
455         }
456     }
457 }
458
459 /*****************************************************************/
460 inline void Mfc::updatetagstate()
461 {
```



```
462     switch (tagupdate) {
463     case TAG_UPDATE_UNDEF:
464         break;;
465     case TAG_UPDATE_IMM:
466         tagstat = maketagstat();
467         chi->mfcwritech(CH24_MFC_RDTAGSTAT______, tagstat);
468         tagupdate = TAG_UPDATE_UNDEF;
469         break;
470     case TAG_UPDATE_ANY:
471         tagstat = maketagstat();
472         if (0 < tagstat && tagstat <= tagmask) {
473             chi->mfcwritech(CH24_MFC_RDTAGSTAT______, tagstat);
474             tagupdate = TAG_UPDATE_UNDEF;
475         }
476         break;
477     case TAG_UPDATE_ALL:
478         tagstat = maketagstat();
479         if (tagstat == tagmask) {
480             chi->mfcwritech(CH24_MFC_RDTAGSTAT______, tagstat);
481             tagupdate = TAG_UPDATE_UNDEF;
482         }
483         break;
484     default:
485         printf("## Mfc::updatetagstate(), illegal request.\n");
486         exit(1);
487     }
488 }
489
490 /*****************************************************************/
491 inline void Mfc::decrementerstep()
492 {
493     if (chip->cycle % 40 == 0) {
494         decrementer--;
495         chi->mfcwritech(CH08_SPU_RDDEC__________, decrementer);
496     }
497 }
498
499 /*****************************************************************/
500 void Mfc::step()
501 {
502     checkchannel();
503     checkdma();
504     updatetagstate();
505     decrementerstep();
506 }
507
508 /*****************************************************************/
509 Mmu::Mmu(Chip * chip)
510 {
511     chip->mmu = this;
512     for (int i = 0; i < BLOCK_TABLE_SIZE; i++) {
513         mmutable[i].addr = i;
514         mmutable[i].id = MMU_UNITID_MEM;
515     }
516 }
517
518 /*****************************************************************/
519 void Mmu::transaddr(address_t * addr)
520 {
521     uint_032t block = addr->logical / BLOCK_SIZE;
522     uint_032t offset = addr->logical % BLOCK_SIZE;
523
524     addr->physical = (mmutable[block].addr * BLOCK_SIZE) + offset;
525     addr->unitid = mmutable[block].id;
526 }
527
528 /*****************************************************************/
529 int Mmu::mapping(uint_032t ea, int id, uint_032t addr)
530 {
531     uint_032t block = ea / BLOCK_SIZE;
532     uint_032t block_addr = addr / BLOCK_SIZE;
533
534     mmutable[block].addr = block_addr;
535     mmutable[block].id = id;
536     return 0;
537 }
538
```

```
539 /*****************************************************************/
540 void Mmu::mapls(uint_032t ea, int id)
541 {
542     int i;
543     for (i = 0; i < LS_SIZE; i += BLOCK_SIZE)
544         mapping(ea + i, id, i);
545 }
546
547 /*****************************************************************/
548 Eib::Eib(Chip * chip)
549 {
550     chip->eib = this;
551     this->chip = chip;
552     this->mem = chip->mem;
553     for (int i = 0; i < MAX_SPES; i++) {
554         if (i < chip->spe_num)
555             this->spe[i] = chip->spe[i];
556         else
557             this->spe[i] = NULL;
558     }
559     for (int i = 0; i < EIB_MAX_ENTRY; i++)
560         valid[i] = EIB_CMD_NOTACTIVE;
561     entry_count = 0;
562     head = 0;
563     tail = 0;
564     transdata_ptr = new uint_032t[DMA_SIZE_MAX / sizeof(uint_032t)];
565 }
566
567 /*****************************************************************/
568 Eib::~Eib()
569 {
570     delete [] transdata_ptr;
571     transdata_ptr = NULL;
572 }
573
574 /*****************************************************************/
575 inline bool Eib::emptyactivecmd()
576 {
577     return (entry_count == 0);
578 }
579
580 /*****************************************************************/
581 inline bool Eib::fullactivecmd()
582 {
583     return (entry_count == EIB_MAX_ENTRY);
584 }
585
586 /*****************************************************************/
587 inline int Eib::enq_waitcmd(dmacmd_t * cmd, dmacmdtag_t * tag)
588 {
589     if (head == (tail + 1) % EIB_WAIT_ENTRY)
590         return ENQ_FAIL;
591     else {
592         waitcmdq[tail].cmd = cmd;
593         waitcmdq[tail].tag = tag;
594         tail = (tail + 1) % EIB_WAIT_ENTRY;
595         return ENQ_SUCCESS;
596     }
597 }
598
599 /*****************************************************************/
600 inline int Eib::deq_waitcmd(eibcmd_t * eibcmd)
601 {
602     if (head == tail)
603         return DEQ_FAIL;
604     else {
605         *eibcmd = waitcmdq[head];
606         head = (head + 1) % EIB_WAIT_ENTRY;
607         return DEQ_SUCCESS;
608     }
609 }
610
611 /*****************************************************************/
612 inline void Eib::checkcmd(dmacmd_t *cmd)
613 {
614     int dma_vio = 0;
615     static int warn = 0;
```



```
616
617     if ((cmd->cmd != DMA_GET) && (cmd->cmd != DMA_PUT)) {
618         printf("## undefined DMA Command Error\n");
619         exit(1);
620     }
621
622     if ((cmd->size == 0) || (cmd->size > DMA_SIZE_MAX)) {
623         dma_vio = 1;
624     } else if (cmd->size < 16) {
625         dma_vio = ((16 % cmd->size != 0) ||
626                    (cmd->ea.physical % 16 != cmd->lsa.physical % 16) ||
627                    (cmd->ea.physical % cmd->size != 0) ||
628                    (cmd->lsa.physical % cmd->size != 0));
629     } else {
630         dma_vio = ((cmd->size % 16 != 0) ||
631                    (cmd->ea.physical % 16 != 0) ||
632                    (cmd->lsa.physical % 16 != 0));
633     }
634     if (dma_vio) {
635         if (!warn) {
636             fprintf(stderr, "## DMA Access Violation Detected.\n");
637             fprintf(stderr, "## EA=%08x LS=%08x SIZE=%5d\n",
638                     (uint) cmd->ea.physical, (uint) cmd->lsa.physical,
639                     (uint) cmd->size);
640             warn = 1;
641             if (!chip->force_mode) {
642                 fprintf(stderr, "## Simulaton aborted.\n");
643                 exit(1);
644             }
645         }
646     }
647 }
648
649 /*****************************************************************/
650 inline void Eib::transdata(dmacmd_t * cmd)
651 {
652     int src_id, dst_id, size = cmd->size;
653     uint_032t src_addr, dst_addr;
654
655     checkcmd(cmd);
656
657     if (cmd->cmd == DMA_GET) {
658         src_addr = cmd->ea.physical;
659         dst_addr = cmd->lsa.physical;
660         src_id = cmd->ea.unitid;
661         dst_id = cmd->lsa.unitid;
662     } else { // if (cmd->cmd == DMA_PUT) {
663         src_addr = cmd->lsa.physical;
664         dst_addr = cmd->ea.physical;
665         src_id = cmd->lsa.unitid;
666         dst_id = cmd->ea.unitid;
667     }
668
669     if (src_id == MMU_UNITID_MEM) {
670         mem->readnb(src_addr, size, transdata_ptr);
671         spe[dst_id - 1]->ls->writenb(dst_addr, size, transdata_ptr);
672     } else if (dst_id == MMU_UNITID_MEM) {
673         spe[src_id - 1]->ls->readnb(src_addr, size, transdata_ptr);
674         mem->writenb(dst_addr, size, transdata_ptr);
675     } else {
676         spe[src_id - 1]->ls->readnb(src_addr, size, transdata_ptr);
677         spe[dst_id - 1]->ls->writenb(dst_addr, size, transdata_ptr);
678     }
679 }
680
681 /*****************************************************************/
682 inline void Eib::comstep()
683 {
684     for (int i = 0; i < EIB_MAX_ENTRY; i++) {
685         if (emptyactivecmd())
686             return;
687         if (valid[i] == EIB_CMD_NOTACTIVE)
688             continue;
689         if (activecmd[i].tag->delay <= 0) {
690             transdata(activecmd[i].cmd);
691             activecmd[i].tag->state = DMA_FINISH;
692             valid[i] = EIB_CMD_NOTACTIVE;
```

```
693             entry_count--;
694         }
695     }
696 }
697
698 /*************************************************************************/
699 inline void Eib::activatecmd()
700 {
701     for (int i = 0; i < EIB_MAX_ENTRY; i++) {
702         if (fullactivecmd())
703             return;
704         if (valid[i] == EIB_CMD_NOTACTIVE) {
705             if (deq_waitcmd(&activecmd[i]) == DEQ_SUCCESS) {
706                 activecmd[i].tag->state = DMA_RUN;
707                 valid[i] = EIB_CMD_ACTIVE;
708                 entry_count++;
709             } else
710                 return;
711         }
712     }
713 }
714
715 /*************************************************************************/
716 void Eib::step()
717 {
718     activatecmd();
719     comstep();
720 }
721
722 /*************************************************************************/
```


file name: etc.cc

```
  1 /*************************************************************************/
  2 /* SimCell: Cell/B.E. Processor Simulator        Arch Lab. TOKYO TECH */
  3 /*************************************************************************/
  4 #include "define.h"
  5
  6 #define IMAGE_HEADER "SimCell_PPE_MEMORY_IMAGE_FILE_VERSION_1.0"
  7 #define LOG_HEADER   "SimCell_LS_LOG_FILE_VERSION_1.0\n"
  8
  9 /*************************************************************************/
 10 ullint gettime()
 11 {
 12     static ullint start = 0;
 13     ullint tm;
 14     struct timeval tv;
 15
 16     gettimeofday(&tv, NULL);
 17     tm = tv.tv_sec * 1000000ull + tv.tv_usec;
 18     if (start == 0)
 19         start = tm;
 20     return tm - start;
 21 }
 22
 23 /*************************************************************************/
 24 char *getlinehead(char *dest, FILE *fp)
 25 {
 26     char *result = fgets(dest, 64, fp);
 27     if (dest[strlen(dest) - 1] != '\n') {
 28         char poi[64];
 29         do
 30             if (fgets(poi, 64, fp) == NULL)
 31                 break;
 32         while (poi[strlen(poi) - 1] != '\n');
 33     }
 34     return result;
 35 }
 36
 37 /*************************************************************************/
 38 void readmemimage(Chip *chip, uint_032t *map_addr)
 39 {
 40     FILE *fp;
 41     char line[65];
 42     int linecnt = 1;
 43     char *endptr;
 44     int regspe = 0, regnum = 0;
 45     char op[4];
 46     uint_032t arg1, arg2;
 47
 48     MainMemory *mem = chip->mem;
 49     Spe **spe = chip->spe;
 50
 51     line[64] = op[3] = '\0';
 52
 53     if (chip->mem_imagefile == NULL)
 54         return;
 55     if ((fp = fopen(chip->mem_imagefile, "r")) == NULL) {
 56         fprintf(stderr, "## can't open file: %s\n", chip->mem_imagefile);
 57         exit(1);
 58     }
 59     getlinehead(line, fp);
 60     if (strncmp(line, IMAGE_HEADER, strlen(IMAGE_HEADER)) != 0) {
 61         fprintf(stderr, "## not a memory image file: %s\n",
 62                 chip->mem_imagefile);
 63         exit(1);
 64     }
 65
 66     while (getlinehead(line, fp) != NULL) {
 67         linecnt++;
 68         // read a line from file
 69         if (line[0] != '@')
 70             continue;
 71         memcpy(op, &line[1], 3);
 72         arg1 = strtoul(&line[5], &endptr, 16);
 73         if (endptr)
 74             arg2 = strtoul(&endptr[1], NULL, 16);
 75         else
 76             arg2 = 0;
```

```
77          if (strcmp(op, "rgi") == 0) {
78              regspe = 0;
79              if (arg1 > (uint) chip->spe_num) {
80                  fprintf(stderr, "## %s:%d: invalid SPE number. skip.\n",
81                          chip->mem_imagefile, linecnt);
82                  continue;
83              } else if (arg2 >= NREG) {
84                  fprintf(stderr, "## %s:%d: too big register number. skip.\n",
85                          chip->mem_imagefile, linecnt);
86                  continue;
87              }
88              regspe = arg1;
89              regnum = arg2;
90          } else if (strcmp(op, "rgv") == 0) {
91              if (!regspe) {
92                  fprintf(stderr, "## %s:%d: use @rgi first. skip.",
93                          chip->mem_imagefile, linecnt);
94                  continue;
95              }
96              spe[regspe - 1]->as->reg[regnum]->put032t(0, arg1);
97              spe[regspe - 1]->as->reg[regnum]->put032t(1, arg2);
98              regspe = 0;
99          } else if (strcmp(op, "mal") == 0) {
100             if ((arg1 > (uint) chip->spe_num) || (arg1 == 0)) {
101                 fprintf(stderr, "## %s:%d: invalid SPE number. skip.\n",
102                         chip->mem_imagefile, linecnt);
103                 continue;
104             }
105             spe[arg1 - 1]->chi->mfcwritech(29, arg2);
106         } else if (strncmp(op, "map", 3) == 0) {
107             if ((arg1 > (uint) chip->spe_num) || (arg1 == 0)) {
108                 fprintf(stderr, "## %s:%d: invalid SPE number. skip.\n",
109                         chip->mem_imagefile, linecnt);
110                 continue;
111             }
112             map_addr[arg1 - 1] = arg2;
113         } else if (strncmp(op, "mem", 3) == 0) {
114             mem->write4b(arg1, arg2);
115         } else if (strncmp(op, "end", 3) == 0) {
116             break;
117         } else {
118             fprintf(stderr, "## %s:%d: undefined command. skip.",
119                     chip->mem_imagefile, linecnt);
120         }
121     }
122     fclose(fp);
123 }
124
125 /**************************************************************************/
126 void verbose(Chip *chip)
127 {
128     static ullint comma_count = 0;
129     ullint inst_count = 0;
130     Spe **spe = chip->spe;
131
132     for (int i = 0; i < chip->spe_num; i++)
133         inst_count += spe[i]->ss->instcnt;
134
135     inst_count /= chip->verb_cycle;
136     for (; comma_count < inst_count; comma_count++) {
137         printf(".");
138         fflush(stdout);
139     }
140 }
141
142 /**************************************************************************/
143 void cyclelog(Chip *chip)
144 {
145     Spe **spe = chip->spe;
146
147     if (chip->debug_mode == 3) {
148         printf("[[cycle %d]]\n", (int) chip->cycle);
149         for (int i = 0; i < chip->spe_num; i++) {
150             printf("[SPU %d]\n", i + 1);
151             spe[i]->spu->printregall();
152             printf("pc                0x%x\n\n",
153                    (uint) (spe[i]->as->pc - 4));
```



```
154         }
155     }
156
157     if ((chip->log_mode) && (chip->cycle % chip->log_cycle == 0))
158         chip->writelogdata();
159
160     if ((chip->cycle % 0x1000 == 0) && (chip->verb_cycle))
161         verbose(chip);
162 }
163
164 /**************************************************************************/
165 ullint atoi_postfix(const char *nptr)
166 {
167     char *endptr;
168     ullint result = strtoll(nptr, &endptr, 10);
169     if ((*endptr == 'k') || (*endptr == 'K'))
170         result *= 1000;
171     else if ((*endptr == 'm') || (*endptr == 'M'))
172         result *= 1000000;
173     else if ((*endptr == 'g') || (*endptr == 'G'))
174         result *= 1000000000;
175     return result;
176 }
177
178 /**************************************************************************/
179 Chip::Chip()
180 {
181     cycle = 0;
182     max_cycle = MAX_CYCLE_DEF;
183     verb_cycle = 0;
184     log_cycle = 0;
185     running_core = 0;
186     spe_num = 1;
187     mem_rddelay = MEM_RDDELAY_DEF;
188     mem_wrdelay = MEM_WRDELAY_DEF;
189     mem_width = MEM_WIDTH_DEF;
190     c2c_delay = C2C_DELAY_DEF;
191     c2c_width = C2C_WIDTH_DEF;
192     log_mode = 0;
193     logpage = 0;
194     logdata = NULL;
195     logfile = NULL;
196     mem_imagefile = NULL;
197     debug_mode = 0;
198     force_mode = 0;
199     big_endian = checkendian();
200 }
201
202 /**************************************************************************/
203 bool Chip::checkendian()
204 {
205     union {
206         unsigned short sh;
207         unsigned char ch[2];
208     } test;
209     test.sh = 0xfeff;
210     return (test.ch[0] == 0xfe);
211 }
212
213 /**************************************************************************/
214 void Chip::setlogmode()
215 {
216     if (log_mode)
217         return;
218
219     log_mode = 1;
220     if ((logfile = fopen(LOGMODE_FILE, "wb")) == NULL) {
221         fprintf(stderr, "can't open file: %s\n", LOGMODE_FILE);
222         exit(1);
223     }
224     fprintf(logfile, LOG_HEADER);
225     fwrite(&spe_num, sizeof(int), 1, logfile);
226     fseek(logfile, 64, SEEK_SET);
227 }
228
229 /**************************************************************************/
230 void Chip::initlogmode()
```

```
231 {
232     logdata = new int **[spe_num];
233     for (int i = 0; i < spe_num; i++) {
234         logdata[i] = new int *[MEM_ACCESS_KIND];
235         for (int j = 0; j < MEM_ACCESS_KIND; j++) {
236             logdata[i][j] = new int[LS_SIZE >> 4];
237             for (int k = 0; k < LS_SIZE >> 4; k++)
238                 logdata[i][j][k] = 0;
239         }
240     }
241 }
242
243 /*********************************************************************/
244 void Chip::setlogdata(int spe, int addr, int type)
245 {
246     spe--;
247     addr = addr >> 4;
248     logdata[spe][type][addr]++;
249 }
250
251 /*********************************************************************/
252 void Chip::writelogdata()
253 {
254     char exist[MEM_ACCESS_KIND][LS_SIZE >> 12];
255     char data[0x100];
256
257     for (int spe = 0; spe < spe_num; spe++) {
258         for (int i = 0; i < MEM_ACCESS_KIND; i++) {
259             for (int block = 0; block < LS_SIZE >> 12; block++) {
260                 int temp = 0;
261                 int off = block << 8;
262                 for (int j = 0; j < 0x100; j++)
263                     temp |= logdata[spe][i][off + j];
264                 exist[i][block] = (temp != 0);
265             }
266         }
267         fwrite(exist, sizeof(exist), 1, logfile);
268         for (int i = 0; i < MEM_ACCESS_KIND; i++) {
269             for (int block = 0; block < LS_SIZE >> 12; block++) {
270                 if (exist[i][block]) {
271                     int off = block << 8;
272                     for (int j = 0; j < 0x100; j++)
273                         data[j] = ((logdata[spe][i][off + j] < 0x100) ?
274                                    logdata[spe][i][off + j] : 0xff);
275                     fwrite(data, sizeof(data), 1, logfile);
276                 }
277             }
278         }
279         for (int i = 0; i < LS_SIZE >> 4; i++)
280             logdata[spe][MEM_DMA][i] = 0;
281     }
282     logpage++;
283 }
284
285 /*********************************************************************/
286 void Chip::finilogmode()
287 {
288     fseek(logfile, 36, SEEK_SET);
289     fwrite(&logpage, sizeof(int), 1, logfile);
290     fclose(logfile);
291     for (int i = 0; i < spe_num; i++) {
292         for (int j = 0; j < MEM_ACCESS_KIND; j++) {
293             delete[] logdata[i][j];
294             logdata[i][j] = NULL;
295         }
296         delete[] logdata[i];
297         logdata[i] = NULL;
298     }
299     delete[] logdata;
300     logdata = NULL;
301     logfile = NULL;
302     log_mode = 0;
303 }
304
305 /*********************************************************************/
```