define.h

```
/**** Yet Another Alpha Processor Simulator: SimAlpha by Kenji KISE */
/*********************************************************************/
#include <time.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <stdint.h>
#include <signal.h>
#include <fcntl.h>
#include <termios.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/resource.h>

/* 1. Type Definition                                                */
/*********************************************************************/
typedef uint32_t INST_TYPE;
typedef uint64_t ADDR_TYPE;
typedef uint64_t DATA_TYPE;


/* 2. Constant Definition                                            */
/*********************************************************************/

#define VER "SimAlpha release version 1.0.0 2002-06-28"

#define DEF_FILE_NAME    "aout.txt"

#define BLOCK_SIZE       0x00002000    /* 8KB page size        */
#define BLOCK_TABLE_SIZE 0x00080000    /* size of page table   */
#define BLOCK_MASK       0x00001fff    /* mask of the page     */
#define BLOCK_MASK_BIT   13            /* mask bit of the page */
#define DATA_T_SIZE      8             /* size of data_t.value */

#define CPU_FEATURE_MASK 0x0000000000000107ull /* for   AMASK op */
#define IMPLEMENTATION_VERSION 2               /* for IMPLVER op */

#define FD_MAPPING_MAX 128                  /* syscall read & write */

#define OSF_SIGCONTEXT_SIZE 568 /* size in byte */
#define OSF_FSTAT_SIZE       80 /* size in byte */
#define OSF_RUSAGE_SIZE      72 /* size in byte */

#define OPTION_MAX          30 /* max of options */

/** for systemcall implementation in syscall.cc                 **/
/** I don't implement all, but something used for SPEC CPU2000. **/
#define ASYS_exit               1
#define ASYS_read               3
#define ASYS_write              4
#define ASYS_close              6
#define ASYS_unlink            10
#define ASYS_brk               17
#define ASYS_lseek             19
#define ASYS_getpid            20
#define ASYS_getuid            24
#define ASYS_open              45
#define ASYS_getgid            47
```



```
#define ASYS_sigprocmask       48
#define ASYS_ioctl             54
#define ASYS_getpagesize       64
#define ASYS_stat              67
#define ASYS_mmap              71
#define ASYS_munmap            73
#define ASYS_fstat             91
#define ASYS_fcntl             92
#define ASYS_sigreturn        103
#define ASYS_gettimeofday     116
#define ASYS_getrusage        117
#define ASYS_rename           128
#define ASYS_getrlimit        144
#define ASYS_setrlimit        145
#define ASYS_sigaction        156
#define ASYS_getdirentries    159
#define ASYS_UNIMPLEMENT      226
#define ASYS_osf_getsysinfo   256
#define ASYS_osf_setsysinfo   257

/* Table C-6: Opcode Summary (Alpha Architecture Handbook V.4)   */
#define OP_PAL  0x00  /* PALcode instruction(CALL_PAL)           */
#define OP_LDA  0x08  /* load address instruction                */
#define OP_LDAH 0x09  /* load address high instruction           */
#define OP_INTA 0x10  /* Integer arithmetic instruction          */
#define OP_INTL 0x11  /* Integer logical instruction             */
#define OP_INTS 0x12  /* Integer shift instruction               */
#define OP_INTM 0x13  /* Integer multiply instruction            */
#define OP_ITFP 0x14  /* Integer to floating-point register move */
#define OP_FLTV 0x15  /* VAX floating-point instruction          */
#define OP_FLTI 0x16  /* IEEE floating-point instruction         */
#define OP_FLTL 0x17  /* Floating-point instruction              */
#define OP_MISC 0x18  /* Miscellaneous instruction               */
#define OP_JSR  0x1a  /* Jump instruction                        */
#define OP_FPTI 0x1c  /* Floating-point to integer register move */

#define BIT0    0x0000000000000001ull
#define BIT1    0x0000000000000002ull
#define BIT2    0x0000000000000004ull
#define BIT3    0x0000000000000008ull
#define BIT4    0x0000000000000010ull
#define BIT5    0x0000000000000020ull
#define BIT6    0x0000000000000040ull
#define BIT7    0x0000000000000080ull
#define BIT8    0x0000000000000100ull
#define BIT9    0x0000000000000200ull
#define BIT10   0x0000000000000400ull
#define BIT11   0x0000000000000800ull
#define BIT12   0x0000000000001000ull
#define BIT15   0x0000000000008000ull
#define BIT20   0x0000000000100000ull
#define BIT31   0x0000000080000000ull
#define BIT63   0x8000000000000000ull

#define EXTND8  0xffffffffffffff00ull
#define EXTND16 0xffffffffffff0000ull
#define EXTND21 0xffffffffffe00000ull
#define EXTND32 0xffffffff00000000ull
#define MASK16  0x000000000000ffffull /* mask 16bit */
#define MASK21  0x00000000001fffffull /* mask 21bit */
#define MASK32  0x00000000ffffffffull /* mask 32bit */

#define ZAP0    0xffffffffffffff00ull
#define ZAP1    0xffffffffffff00ffull
#define ZAP2    0xffffffffff00ffffull
#define ZAP3    0xffffffff00ffffffull
```

```
#define ZAP4    0xffffff00ffffffffull
#define ZAP5    0xffff00ffffffffffull
#define ZAP6    0xff00ffffffffffffull
#define ZAP7    0x00ffffffffffffffull

/** for instruction.cc **/
#define MSK1 0x3e
#define MSK2 0x3c
#define MSK3 0x38
#define MSK4 0x30


/* 4. structure and class definition                                    */
/************************************************************************/

struct osf_statbuf{ /*** Used in syscall.c SYS_fstat ***/
  uint32_t st_dev;
  uint32_t st_ino;
  uint32_t st_mode;
  uint16_t st_nlink;
  uint16_t pad0;
  uint32_t st_uid;
  uint32_t st_gid;
  uint32_t st_rdev;
  uint32_t pad1;
  uint64_t st_size;
  uint32_t st_atime;
  uint32_t st_spare1;
  uint32_t st_mtime;
  uint32_t st_spare2;
  uint32_t st_ctime;
  uint32_t st_spare3;
  uint32_t st_blksize;
  uint32_t st_blocks;
  uint32_t st_gennum;
  uint32_t st_spare4;
};

class system_config{
 public:
  char program_name[512];
  uint64_t end_tc;
  int verbose_interval;
  int debug_flag;
  int syscall_flag;
  int fd_stdin_flag;
  int fd_stdin; /* file descripter for STDIN */
  system_config(char *, char**);
};


class evaluation_result{
 public:
  uint64_t retired_inst;
  int used_memory_block;
  time_t time_begin;     /* start time stamp */
  struct timeval tp;     /* start time stamp */
  struct timezone tzp;   /* start time stamp */
  evaluation_result();
};


class data_t{
  uint64_t value;
 public:
  int cmov;
```



```
  uint64_t ld();
  int   st(uint64_t);
  int init(uint64_t);
};


class architecture_state{
 public:
  data_t pc;    /* program counter      */
  data_t r[32]; /* general purpose regs */
  data_t f[32]; /* floating point  regs */
  architecture_state(system_config *,
                     evaluation_result *);
};


class main_memory{
  evaluation_result *e;
  data_t *block_table[BLOCK_TABLE_SIZE];
  data_t *allocblock(data_t *);
 public:
  void ld_8byte(data_t *, data_t *);
  void st_8byte(data_t *, data_t *, DATA_TYPE);
  main_memory(evaluation_result *);
};


class memory_system{
  evaluation_result *e;
  class main_memory *mm;
  void ld_8byte(data_t *, data_t *);
  void st_8byte(data_t *, data_t *, DATA_TYPE);
 public:
  void ld_inst(data_t *, INST_TYPE *);
  void ld_nbyte(int, data_t *, data_t *);
  void st_nbyte(int, data_t *, data_t *);
  ~memory_system();
  memory_system(system_config *,
                evaluation_result *);
};


class system_manager{
  class evaluation_result *e;
  class system_config     *ms;
  architecture_state      *as;
  memory_system           *mem;
  /* File Descriptor Mapping Table */
  DATA_TYPE fd_table[FD_MAPPING_MAX];
  /* Unique Data for rduniq & wruniq inst */
  DATA_TYPE uniq;
 public:
  int running;    /* chip is running ? */
  void execute_pal(class instruction *);
  int translate_fd(DATA_TYPE);
  system_manager(architecture_state *,
                 memory_system *,
                 system_config *,
                 evaluation_result *);
};


class debug{
  class evaluation_result *e;
  class system_config     *ms;
  architecture_state      *as;
```

```
262    memory_system           *mem;
263   public:
264    debug(architecture_state *, memory_system *,
265          struct system_config *, evaluation_result *);
266    void debugmode();
267  };
268
269
270  class instruction{
271    evaluation_result  *e;
272    architecture_state *as;
273    system_manager     *sys;
274    memory_system      *mem;
275
276    INST_TYPE ir; /* 32bit instruction code  */
277    int Op;       /* Opcode field            */
278    int RA;       /* Ra field of the inst    */
279    int RB;       /* Rb field of the inst    */
280    int RC;       /* Rc field of the inst    */
281    int ST;       /* store inst ?            */
282    int LD;       /* load inst ?             */
283    int LA;       /* load address inst ?     */
284    int BR;       /* branch inst ?           */
285    int Ai;       /* Rav is immediate ?      */
286    int Bi;       /* Rbv is immediate ?      */
287    int Af;       /* Rav from floating-reg ? */
288    int Bf;       /* Rbv from floating-reg ? */
289    int WF;       /* Write to the f-reg ?    */
290    int WB;       /* Writeback reg index     */
291    data_t  Npc;  /* Update PC or PC + 4     */
292    data_t  Imm;  /* immediate               */
293    data_t  Adr;  /* load & store address    */
294    data_t  Rav;  /* Ra                      */
295    data_t  Rbv;  /* Rb                      */
296    data_t  Rcv;  /* Rc                      */
297   public:
298    int Fetch(data_t *);
299    int Fetch(data_t *, INST_TYPE);
300    int Slot();
301    int Rename();
302    int Issue();
303    int RegisterRead();
304    int Execute(data_t *);
305    int Memory();
306    int WriteBack();
307
308    INST_TYPE get_ir();
309    int data_ld(data_t *, data_t *);
310    int data_st(data_t *, data_t *);
311    instruction(architecture_state *,
312                memory_system *,
313                system_manager *,
314                system_config *,
315                evaluation_result *);
316  };
317
318
319  class simple_chip{
320    system_config      *sc;
321    evaluation_result  *e;
322    debug              *deb;
323    system_manager     *sys;
324    instruction        *p;
325   public:
326    memory_system      *mem;
327    architecture_state *as;
```



```
328    simple_chip(char *, char **);
329    ~simple_chip();
330    int step();
331  };
332
333
334  struct alpha_stat{  /*** for syscall.cc ***/
335    uint64_t  st_dev;
336    uint32_t  st_ino;
337     int32_t  __pad1;
338    uint32_t  st_mode;
339    uint32_t  st_nlink;
340    uint32_t  st_uid;
341    uint32_t  st_gid;
342    uint64_t  st_rdev;
343     int64_t  st_size;
344     int64_t  st_atime;
345     int64_t  st_mtime;
346     int64_t  st_ctime;
347    uint32_t  st_blocks;
348     int32_t  __pad2;
349    uint32_t  st_blksize;
350    uint32_t  st_flags;
351    uint32_t  st_gen;
352     int32_t  __pad3;
353     int64_t  __unused[4];
354  };
355
356
357  /* 5. function prototypes                                              */
358  /***********************************************************************/
359  extern void print_evaluation_result(struct evaluation_result *,
360                                      struct system_config *,
361                                      system_manager *);
362  extern void usage();
363  extern uint64_t map_s(unsigned int);
364  extern INST_TYPE cmov_ir_create(INST_TYPE);
365  extern int ALU(INST_TYPE, data_t *, data_t *, data_t *);
366  extern int BRU(INST_TYPE, data_t *, data_t *, data_t *, data_t *);
367
368  extern "C" int simple_cpu(int, uint64_t *); /**** C interface *****/
```

sim.cc

```
369  /**** Yet Another Alpha Processor Simulator in C++ by Kenji KISE ****/
370  /***********************************************************************/
371  #include "define.h"
372
373  int main(int argc, char **argv){
374    if(argc==1) usage();
375    char *p    = argv[argc-1]; /* program name */
376    char **opt = argv;         /* options      */
377
378    simple_chip *chip =  new simple_chip(p, opt);
379    while(chip->step());
380    delete chip;
381
382    return 0;
383  }
```

chip.cc

```
384  /**** Yet Another Alpha Simulator: SimAlpha        by Kenji KISE ****/
385  /***********************************************************************/
386  #include "define.h"
387
```

```
388 inline void house_keeper(system_manager *sys, system_config *ms,
389                          evaluation_result *e, debug *deb){
390   if(ms->debug_flag) deb->debugmode();
391   if(ms->verbose_interval &&
392      (e->retired_inst % ms->verbose_interval)==0){
393     fprintf(stdout, "."); fflush(stdout);
394   }
395   if(ms->end_tc!=0 && e->retired_inst>=ms->end_tc)
396     sys->running = 0;
397 }
398
399
400 simple_chip::simple_chip(char *prog, char **opt){
401   sc  = new system_config(prog, opt);
402   e   = new evaluation_result;
403   as  = new architecture_state(sc, e);
404   mem = new memory_system(sc, e);
405   deb = new debug(as, mem, sc, e);
406   sys = new system_manager(as, mem, sc, e);
407   p   = new instruction(as, mem, sys, sc, e);
408 }
409
410
411 simple_chip::~simple_chip(){   /** destructor **/
412   print_evaluation_result(e, sc, sys);
413   delete p;
414   delete sys;
415   delete deb;
416   delete mem;
417   delete as;
418   delete e;
419   delete sc;
420 }
421
422
423 inline int execute_cmovb(instruction *p, architecture_state *as){
424   INST_TYPE ir=cmov_ir_create(p->get_ir());
425   if(ir){ /** CMOVb inst, see README.txt **/
426     p->Fetch(&as->pc, ir);
427     p->Slot();
428     p->Rename();
429     p->Issue();
430     p->RegisterRead();
431     p->Execute(&as->pc);
432     p->Memory();
433     p->WriteBack();
434   }
435   return 0;
436 }
437
438
439 int simple_chip::step(){
440   p->Fetch(&as->pc);    /* pipeline stage 0 */
441   p->Slot();            /* pipeline stage 1 */
442   p->Rename();          /* pipeline stage 2 */
443   p->Issue();           /* pipeline stage 3 */
444   p->RegisterRead();    /* pipeline stage 4 */
445   p->Execute(&as->pc);  /* pipeline stage 5 */
446   p->Memory();          /* pipeline stage 6 */
447   p->WriteBack();
448
449   /* split a conditional move,see README.txt */
450   execute_cmovb(p, as);
451
452   e->retired_inst++;
453   house_keeper(sys, sc, e, deb);
```



```
454
455   return sys->running;
456 }

instruction.cc

457 /**** Yet Another Alpha Processor Simulator in C++ by Kenji KISE ****/
458 /*********************************************************************/
459 #include "define.h"
460
461 /*********************************************************************/
462 inline int nbyte(INST_TYPE ir){
463   int Op = (ir>>26) & 0x3F;
464
465   int n  =  (Op==0x0e || Op==0x0a) ? 1 :
466     (Op==0x0d || Op==0x0c) ? 2 :
467     (Op==0x22 || Op==0x26 || Op==0x28 || Op==0x2a
468      || Op==0x2c || Op==0x2e) ? 4 : 8;
469   return n;
470 }
471
472 int instruction::data_st(data_t *adr, data_t *dat){
473   int Op = (ir>>26) & 0x3F;
474   if(Op==0x26){ /***** STS *****/
475     DATA_TYPE Rav_t;
476     DATA_TYPE tmp = dat->ld();
477     uint32_t int_tmp = 0;
478     int_tmp = 0;
479     int_tmp |= ((tmp >> 29) & 0x3FFFFFFFul);  // Set 30 bit
480     int_tmp |= ((tmp >> 32) & 0xc0000000ul);  // Set  2 bit
481     memcpy(&Rav_t, &int_tmp, 4);
482     dat->st(Rav_t & 0xffffffffull);
483   }
484
485   mem->st_nbyte(nbyte(ir), adr, dat);
486   return 0;
487 }
488
489 int instruction::data_ld(data_t *adr, data_t *dat){
490
491   mem->ld_nbyte(nbyte(ir), adr, dat);
492
493   int Op = (ir>>26) & 0x3F;
494   if(Op==0x28 || Op==0x2a){ /**** LDL & LDL_L ****/
495     DATA_TYPE Rcv_t = dat->ld();
496     if(Rcv_t & BIT31) Rcv_t|=EXTND32;
497     dat->st(Rcv_t);
498   }
499   if(Op==0x22){ /**** LDS ****/
500     DATA_TYPE Rcv_t = dat->ld();
501     DATA_TYPE Tmp = Rcv_t;
502     Rcv_t = (Tmp >> 31) & 0x1ull;                    // Set  1 bit
503     Rcv_t = (Rcv_t << 11) | (map_s(Tmp) & 0x7ffull); // Set 11 bit
504     Rcv_t = (Rcv_t << 23) | (Tmp & 0x7FFFFull);      // Set 23 bit
505     Rcv_t =  Rcv_t << 29;                            // Set 29 bit
506     dat->st(Rcv_t);
507   }
508   return 0;
509 }
510
511 /*********************************************************************/
512 instruction::instruction(architecture_state *a, memory_system *m,
513                          system_manager *s, system_config *sc,
514                          evaluation_result *et){
515   as  = a;
516   mem = m;
```

```
517    sys = s;
518    e   = et;
519  }
520
521
522  INST_TYPE instruction::get_ir(){
523    return ir;
524  }
525
526
527  int instruction::Fetch(data_t *pc){
528    mem->ld_inst(pc, &ir);
529    Npc.init(pc->ld() + 4);
530    return 0;
531  }
532
533  /** CMOVb case, npc and pc has the same value **/
534  int instruction::Fetch(data_t *pc, INST_TYPE ir_t){/*** CMOVb ***/
535    ir = ir_t;
536    Npc.init(pc->ld());
537    return 0;
538  }
539
540  /** Decode IR **/
541  int instruction::Slot(){
542    Op  = (ir>>26) &  0x3F;
543    RA  = (ir>>21) &  0x1F;
544    RB  = (ir>>16) &  0x1F;
545    RC  = (ir    ) &  0x1F;
546    WF  = ((Op&MSK2)==0x14 || (Op&MSK2)==0x20);
547    LA  = (Op==0x08 || Op==0x09);
548    LD  = (Op==0x0a || Op==0x0b || Op==0x0c ||
549           (Op&MSK2)==0x20 || (Op&MSK2)==0x28);
550    ST  = (Op==0x0d || Op==0x0e || Op==0x0f ||
551           (Op&MSK2)==0x24 || (Op&MSK2)==0x2c);
552    BR  = ((Op&MSK4)==0x30);
553    WB  = (LD || (Op&MSK2)==0x08 || Op==0x1a ||
554           Op==0x30 || Op==0x34) ? RA :
555          ((Op&MSK3)==0x10 || Op==0x1c) ? RC : 31;
556    Af  = (Op==0x15 || Op==0x16 || Op==0x17 ||
557           Op==0x1c ||
558           (Op&MSK2)==0x24 || (Op&MSK3)==0x30);
559    Bf  = ((Op&MSK2)==0x14);
560    Ai = (Op==0x08 || Op==0x09 || LD);
561    Bi = (BR || (Op&MSK2)==0x10 && (ir & BIT12));
562
563    /** For the CMOV Split Code (CMOV1) **/
564    if(cmov_ir_create(ir)){
565      RB = RC;
566      Bi = 0;
567    }
568    return 0;
569  }
570
571
572  int instruction::Rename(){
573    return 0;
574  }
575
576  /** genarate the immediate data Imm **/
577  int instruction::Issue(){
578    DATA_TYPE Lit, D16, D21, tmp, d21e, d16e;
579    d21e = ((ir & MASK21) | EXTND21) << 2;
580    d16e  = (ir & MASK16) | EXTND16;
581
582    Lit = (ir>>13) & 0xFF;
```



```
583    D21 = (ir & BIT20) ? d21e : (ir&MASK21)<<2;
584    D16 = (ir & BIT15) ? d16e : (ir&MASK16);
585    if(Op==0x09) D16 = (D16 << 16);
586
587    tmp = (LA||LD||ST) ? D16 : (BR) ? D21 : Lit;
588    Imm.init(tmp);
589    return 0;
590  }
591
592  /*** copy data to Rav and Rbv ***/
593  int instruction::RegisterRead(){
594    Rav = Ai ? Imm : Af ? as->f[RA] : as->r[RA];
595    Rbv = Bi ? Imm : Bf ? as->f[RB] : as->r[RB];
596    return 0;
597  }
598
599  /*** Updata Rcv, Adr and Tpc ***/
600  int instruction::Execute(data_t *Tpc){
601    /*** Update Rcv ***/
602    if(BR || Op==OP_JSR){
603      Rcv=Npc;
604    }
605    else if(!LD){
606      ALU(ir, &Rav, &Rbv, &Rcv);
607    }
608
609    /*** Update Adr ***/
610    Adr.init(0);
611    if(LD || ST){
612      ALU(ir, &Imm, &Rbv, &Adr);
613    }
614
615    /*** Update Tpc ***/
616    *Tpc = Npc;
617    if(Op==OP_JSR){
618      *Tpc = Rbv;
619      Tpc->st(Tpc->ld() & ~3ull);
620    }
621    if(BR){
622      BRU(ir, &Rav, &Rbv, &Npc, Tpc);
623    }
624    return 0;
625  }
626
627  /*** store Rav to mem or load to Rcv ***/
628  int instruction::Memory(){
629    if(ST) data_st(&Adr, &Rav);
630    if(LD) data_ld(&Adr, &Rcv);
631    return 0;
632  }
633
634  /** PAL or copy Rcv into the regfile **/
635  int instruction::WriteBack(){
636    if(Op==OP_PAL){
637      sys->execute_pal(this);
638    }
639
640    if(!WF && WB!=31) as->r[WB] = Rcv;
641    if( WF && WB!=31) as->f[WB] = Rcv;
642    return 0;
643  }
```

memory.cc

```
644  /**** Yet Another Alpha Processor Simulator in C++ by Kenji KISE ****/
645  /*********************************************************************/
```

```
#include "define.h"

/*
Initialize the virtual memory space, data file looks below.
Line must begin by '@' in order to indicate the valid data.
@11ff97000 00000003
@11ff97008 1ff97138
*/

/*************************************************************************/
inline uint32_t MM_TABLE_INDEX(uint32_t addr){
  return (addr >> BLOCK_MASK_BIT);
}

inline uint32_t gen_tag(uint32_t adr){
  return (adr  & 0xfffffc0);
}

/*************************************************************************/
void memory_system::ld_inst(data_t *a, INST_TYPE *ir){
  data_t d;
  mm->ld_8byte(a, &d);
  if(a->ld()%8==0) *ir =  d.ld()      & 0xffffffff;
  if(a->ld()%8==4) *ir = (d.ld()>>32) & 0xffffffff;
}


void memory_system::ld_8byte(data_t *a, data_t *d){
  mm->ld_8byte(a, d);
}


void memory_system::st_8byte(data_t *a, data_t *d, DATA_TYPE msk){
  mm->st_8byte(a, d, msk);
}


memory_system::~memory_system(){
}


memory_system::memory_system(system_config *sc,
                             evaluation_result *ev){
  e = ev;
  mm = new main_memory(ev);

  char *prog_name = sc->program_name;

  /*** Read memory image from the file ***/
  FILE *fp;
  if((fp = fopen(prog_name, "r")) == NULL) {
    fprintf(stderr, "Bad file name: %s\n", prog_name);
    exit(1);
  }
  char buf[4096];
  fgets(buf, 4096, fp); /*** File Format Check ***/
  if(strncmp(buf+3, "SimAlpha", 8)){
    fprintf(stderr, "%s: Not a SimAlpha data file.\n", prog_name);
    exit(1);
  }
  while(!feof(fp)){
    fgets(buf, 4096, fp);
    if(*buf=='@'){
      DATA_TYPE dat;
      ADDR_TYPE adr;
      sscanf(buf+1, "%llx %llx\n", &adr, &dat);
```



```
      data_t a,d;
      a.init(adr);
      d.init(dat);
      mm->st_8byte(&a, &d, 0);
    }
  }
  fclose(fp);
}


void memory_system::ld_nbyte(int n, data_t *a, data_t *d){
  if(a->ld()%n!=0) printf("*** ld_nbyte n=%d miss-alignment.\n", n);

  ld_8byte(a, d); /** Type Conversion **/

  int offset = a->ld() & 7;
  switch(n){
  case 1: {
    DATA_TYPE data= (d->ld() >> (offset * 8)) & 0xfflu;
    d->st(data);
    break;
  }
  case 2: {
    DATA_TYPE data= (d->ld() >> (offset * 8)) & 0xffffllu;
    d->st(data);
    break;
  }
  case 4: {
    DATA_TYPE data= (d->ld() >> (offset * 8)) & 0xffffffffllu;
    d->st(data);
    break;
  }
  case 8: {
    break;
  }
  default: printf("Case %d, Error in load_nbyte\n", n);
    exit(1);
  }
}


void memory_system::st_nbyte(int n, data_t *a, data_t *d){
  if(a->ld()%n!=0) printf("*** st_nbyte n=%d miss-alignment.\n", n);

  int offset = a->ld() & 7;
  DATA_TYPE mask = 0;

  switch(n){
  case 1: {
    mask = ~(0xfflu << offset*8);
    DATA_TYPE data = (d->ld() & 0xfflu) << offset*8;
    d->st(data);
    break;
  }
  case 2: {
    mask = ~(0xffffllu << offset*8);
    DATA_TYPE data = (d->ld() & 0xffffllu) << offset*8;
    d->st(data);
    break;
  }
  case 4: {
    mask = ~(0xffffffffllu << offset*8);
    DATA_TYPE data = (d->ld() & 0xffffffffllu) << offset*8;
    d->st(data);
    break;
  }
```

```
   778    case 8: {
   779      mask = 0;
   780      break;
   781    }
   782    default: printf("Case %d, Error in store_nbyte\n", n);
   783      exit(1);
   784    }
   785
   786    st_8byte(a, d, mask); /** Type Conversion **/
   787  }
   788
   789  /*********************************************************************/
   790  main_memory::main_memory(evaluation_result *ev){
   791    e = ev;
   792    for(int i=0; i<BLOCK_TABLE_SIZE; i++) block_table[i]=NULL;
   793  }
   794
   795  data_t *main_memory::allocblock(data_t *a){
   796    data_t *ret=(data_t *)calloc(BLOCK_SIZE/DATA_T_SIZE,
   797                                 sizeof(class data_t));
   798    block_table[MM_TABLE_INDEX(a->ld())]=ret;
   799    if(ret==NULL){
   800      printf("** Error: memory allocation in allocblock.\n");
   801      exit(0);
   802    }
   803    e->used_memory_block++;
   804    return ret;
   805  }
   806
   807  void main_memory::ld_8byte(data_t *a, data_t *d){
   808    ADDR_TYPE adr = a->ld() & ~7;
   809    data_t *ptr = block_table[MM_TABLE_INDEX(adr)];
   810    unsigned int offset = (adr & BLOCK_MASK)/DATA_T_SIZE;
   811    if(ptr==NULL) ptr=allocblock(a);
   812    *d = *(ptr + offset); /** COPY **/
   813  }
   814
   815  void main_memory::st_8byte(data_t *a, data_t *d, DATA_TYPE msk){
   816    ADDR_TYPE adr = a->ld() & ~7;
   817    data_t *ptr = block_table[MM_TABLE_INDEX(adr)];
   818    unsigned int offset = (adr & BLOCK_MASK)/DATA_T_SIZE;
   819    if(ptr==NULL) ptr=allocblock(a);
   820    (ptr + offset)->st( ((ptr + offset)->ld() & msk) | d->ld() );
   821  }
```

etc.cc

```
   822  /*********************************************************************/
   823  /**** Yet Another Alpha Processor Simulator in C++ by Kenji KISE ****/
   824  /*********************************************************************/
   825  #include "define.h"
   826
   827  /*********************************************************************/
   828  void usage(){
   829    char UsageMessage[] =  "\
   830  Usage: SimAlpha [-option] command\n\
   831   -d: enter a debug mode\n\
   832   -c: print systemcall trace\n\
   833   -v[num]: verbose mode. prints '.' by VERBOSE_INTERVAL\n\
   834   -e[num][m]: end simulation at num code run. m means million\n\
   835   -i[file_name]: use file_name as STDIN input file\n\
   836  ";
   837    printf("%s\n", VER);
   838    printf("%s", UsageMessage); exit(0);
   839  }
   840
```



```
   841  /* C interface                                                       */
   842  /*********************************************************************/
   843  int simple_cpu(int job, uint64_t *p){
   844    static simple_chip *chip;
   845
   846    switch(job){
   847    case 1:{
   848      static char *opt[OPTION_MAX];
   849      char *buf;
   850      for(int i=0; i<OPTION_MAX; i++) opt[i]=NULL;
   851      FILE *fp;
   852      if((fp=fopen("SimAlpha_Option.txt","r"))!=NULL){
   853        printf("*** Use SimAlpha_Option.txt\n");
   854        int i=0;
   855        while(!feof(fp)){
   856          buf = (char *)malloc(256);
   857          fscanf(fp, "%s", buf);
   858          if(buf[0]=='-'){
   859            opt[i++] = buf;
   860            i++;
   861          }
   862        }
   863      }
   864      chip = new simple_chip(DEF_FILE_NAME, opt);
   865    }
   866    break;
   867    case 2:{
   868      *(p++) = chip->as->pc.ld();
   869      int running = chip->step();
   870
   871      for(int i=0; i<31; i++)
   872        *(p++)=chip->as->r[i].ld();
   873      for(int i=0; i<31; i++)
   874        *(p++)=chip->as->f[i].ld();
   875
   876      if(running==0) delete chip;
   877      break;
   878    }
   879    default:
   880      printf("Warning: %d siple_cpu.\n", job);
   881    }
   882    return 0;
   883  }
   884
   885  /* map_s is used in LDS & ITOFS instruction                          */
   886  /* input of 8bit and output of 11bit                                 */
   887  /*********************************************************************/
   888  uint64_t map_s(unsigned int int_tmp){
   889    unsigned int i8 = (int_tmp >> 23) & 0xFF;
   890
   891    /** Next line should be valid, SimpleScalar has a bug ? **/
   892    // if(i8==0xFF)  return 0x00000000000007FFull;
   893    if(i8==0x00)  return 0x0000000000000000ull;
   894
   895    if(i8 & 0x80) return 0x0000000000000400ull | (i8 & 0x7F);
   896    else          return 0x0000000000000380ull | (i8 & 0x7F);
   897
   898    printf("Warning In load_store.c, map_s(). \n");
   899  }
   900
   901  /* If IR is not the cmov, return 0. Otherwise return IR of CMOVb.    */
   902  /*********************************************************************/
   903  INST_TYPE cmov_ir_create(INST_TYPE ir){
   904    int Op     = (ir>>26) & 0x3F;
   905    int func7  = (ir>>5)  & 0x7f;
   906    int func11 = (ir>>5)  & 0x7FF;
```

```
907
908    switch(Op){
909    case 0x11:
910      switch(func7){
911      case 0x14:
912      case 0x16:
913      case 0x24:
914      case 0x26:
915      case 0x44:
916      case 0x46:
917      case 0x64:
918      case 0x66:
919        {
920          INST_TYPE ret;
921          int Op  = (ir>>26) &  0x3F;
922          int Lit = (ir>>13) &  0xFF;
923          int RC  = (ir    ) &  0x1F;
924          int Fnc = 0x6e; /** This is assigned by Kenji KISE. **/
925          ret = (Op << 26) | (RC<<21) | (Lit<<13) | (ir & BIT12)
926            | (Fnc<<5) | RC;
927          return ret;
928        }
929      }
930
931    case 0x17:
932      switch(func11){
933      case 0x02a:
934      case 0x02b:
935      case 0x02c:
936      case 0x02d:
937      case 0x02e:
938      case 0x02f:
939        {
940          INST_TYPE ret;
941          int Op  = (ir>>26) &  0x3F;
942          int RB  = (ir>>16) &  0x1F;
943          int RC  = (ir    ) &  0x1F;
944          int Fnc = 0x333; /** This is assigned by Kenji KISE. **/
945          ret = (Op << 26) | (RC<<21) | (RB<<16) | (Fnc<<5) | RC;
946          return ret;
947        }
948      }
949    }
950    return 0;
951  }
952
953  /* Methods for data_t                                                  */
954  /**********************************************************************/
955  int data_t::init(uint64_t d){ /*** generate the new data ***/
956    value = d;
957    cmov  = 0;
958    return 0;
959  }
960
961  /**********************************************************************/
962  uint64_t data_t::ld(){      /*** read  the value of the data ***/
963    return value;
964  }
965
966  /**********************************************************************/
967  int data_t::st(uint64_t d){ /*** write the value of the data ***/
968    value = d;
969    return 0;
970  }
971
972  /*
```



```
973   * Initialize the registers, data file looks below.
974   * Line must begin by "\@" in order to indicate the valid data.
975   * /@reg 28 0000000000000000
976   * /@reg 29 0000000140023e90
977   */
978  /**********************************************************************/
979  architecture_state::architecture_state(system_config *sc,
980                                         evaluation_result *e){
981    for(int i=0; i<32; i++) r[i].init(0);
982    for(int i=0; i<32; i++) f[i].init(0);
983
984    char *program_name = sc->program_name;
985    FILE *fp;
986    if((fp = fopen(program_name, "r")) == NULL) {
987      fprintf(stderr, "Bad file name: %s\n", program_name);
988      exit(1);
989    }
990
991    int i;
992    DATA_TYPE dat;
993    char buf[4096];
994    fgets(buf, 4096, fp); /*** File Format Check ***/
995    if(strncmp(buf+3, "SimAlpha", 8)){
996      fprintf(stderr, "%s: Not a SimAlpha data file.\n", program_name);
997      exit(1);
998    }
999
1000   while(!feof(fp)){
1001     fgets(buf, 4096, fp);
1002
1003     if(*buf=='/' && *(buf+1)=='@'){
1004       sscanf(buf+6,"%d %llx\n", &i, &dat);
1005       if(i==32) pc.init(dat); else r[i].init(dat);
1006     }
1007     if(i==32) break;
1008   }
1009   fclose(fp);
1010 }
1011
1012 /**********************************************************************/
1013 void print_evaluation_result(struct evaluation_result *e,
1014                              struct system_config *ms,
1015                              system_manager *sys){
1016   struct timeval tp;
1017   struct timezone tzp;
1018   gettimeofday(&tp, &tzp);
1019
1020   uint64_t run_usec = (uint64_t)(tp.tv_sec - e->tp.tv_sec) * 1000000ull
1021     + (uint64_t)tp.tv_usec - (uint64_t)e->tp.tv_usec;
1022   fflush(stdout);
1023   fflush(stderr);
1024   int minute = (run_usec / 1000000) / 60;
1025   int second = (run_usec / 1000000) % 60;
1026   fprintf(stderr,"\n==================================");
1027   fprintf(stderr,"================================\n");
1028   fprintf(stderr,"=== %s\n", VER);
1029   char buf[256];
1030   getcwd(buf, 256);
1031   fprintf(stderr, "=== cwd: %s\n", buf);
1032   fprintf(stderr,"=== %5qd million code(%13qd code)",
1033           e->retired_inst/1000000, e->retired_inst);
1034   fprintf(stderr,"  %6.3f MIPS\n",
1035           (double)e->retired_inst/(double)run_usec);
1036   fprintf(stderr,"=== SimAlpha takes %d min %d second (%qd usec)\n",
1037           minute, second, run_usec);
1038   fprintf(stderr,"=== SimAlpha starts at %s", ctime(&e->time_begin));
```

```
1039    fprintf(stderr,"=== Memory %dKB used (block size %dKB, %d blocks)\n",
1040            BLOCK_SIZE/1024 * e->used_memory_block,
1041            BLOCK_SIZE/1024, e->used_memory_block);
1042    fflush(stderr);
1043  }
1044
1045  /*************************************************************************/
1046  system_config::system_config(char *program, char **option){
1047    memset(this, 0, sizeof(system_config));
1048    strcpy(program_name, program);
1049
1050    if(option==NULL) return;
1051    for(int i=0; option[i]!=NULL; i++){
1052      char opt[1024];
1053      strcpy(opt, option[i]);
1054      if(opt[0]!='-') continue;
1055      switch(opt[1]){
1056      case 'c': syscall_flag = 1; break;
1057      case 'd': debug_flag   = 1; break;
1058      case 'v':
1059        verbose_interval =atoi(&opt[2]);
1060        if(verbose_interval == 0)   verbose_interval  =  100000;
1061        if(opt[strlen(opt)-1]=='m') verbose_interval *= 1000000;
1062        printf("*** verbose interval: %d\n", verbose_interval);
1063        break;
1064      case 'e':
1065        end_tc =atoi(&opt[2]);
1066        if(opt[strlen(opt)-1]=='m') end_tc *= 1000000;
1067        printf("*** Finish at %qd code execution.\n",
1068               end_tc);
1069        break;
1070      case 'i':
1071        fd_stdin_flag = 1;
1072        fd_stdin = open(&opt[2], O_RDONLY);
1073        printf("*** SimAlpha: use [%s] as STDIN.\n", &opt[2]);
1074        if(fd_stdin<0){
1075          printf("Can not open file: %s\n", &opt[2]);
1076          exit(1);
1077        }
1078        break;
1079      default:
1080        printf("*** Error: [%s] Wrong option!!\n\n", opt);
1081        usage();
1082        exit(1);
1083      }
1084    }
1085  }
1086
1087  /*************************************************************************/
1088  evaluation_result::evaluation_result(){
1089    memset(this, 0, sizeof(class evaluation_result));
1090
1091    time_begin = time(0);
1092    gettimeofday(&tp, &tzp);
1093  }
```

debug.cc

```
1094  /**** Yet Another Alpha Processor Simulator in C++ by Kenji KISE ****/
1095  /*************************************************************************/
1096  #include "define.h"
1097
1098  /*
1099   * convert hexadecimal value into decimal value
1100   ************************************************/
1101  uint64_t get_hex(char *buf){
```



```
1102    uint64_t sum  = 0;
1103
1104    int num = strlen(buf);
1105    if(num>16){
1106      printf("Error! hex() can treat within 16digit numbers.\n");
1107      exit(1);
1108    }
1109
1110    for(int i=0; i<num; i++){
1111      uint64_t temp = 0;
1112      char t = *((char *)(buf+(num-i-1)));
1113      switch(t){
1114      case '0': temp = 0; break;
1115      case '1': temp = 1; break;
1116      case '2': temp = 2; break;
1117      case '3': temp = 3; break;
1118      case '4': temp = 4; break;
1119      case '5': temp = 5; break;
1120      case '6': temp = 6; break;
1121      case '7': temp = 7; break;
1122      case '8': temp = 8; break;
1123      case '9': temp = 9; break;
1124      case 'a': temp = 10;break;
1125      case 'b': temp = 11;break;
1126      case 'c': temp = 12;break;
1127      case 'd': temp = 13;break;
1128      case 'e': temp = 14;break;
1129      case 'f': temp = 15;break;
1130      }
1131      sum += temp * (uint64_t)(1llu << 4*i);
1132    }
1133    return sum;
1134  }
1135
1136  /*************************************************************************/
1137  void print_register_name(int i){
1138    switch(i){
1139    case 0:   printf("v0    "); break;
1140    case 1:   printf("t0    "); break;
1141    case 2:   printf("t1    "); break;
1142    case 3:   printf("t2    "); break;
1143    case 4:   printf("t3    "); break;
1144    case 5:   printf("t4    "); break;
1145    case 6:   printf("t5    "); break;
1146    case 7:   printf("t6    "); break;
1147    case 8:   printf("t7    "); break;
1148    case 9:   printf("s0    "); break;
1149    case 10:  printf("s1    "); break;
1150    case 11:  printf("s2    "); break;
1151    case 12:  printf("s3    "); break;
1152    case 13:  printf("s4    "); break;
1153    case 14:  printf("s5    "); break;
1154    case 15:  printf("fp    "); break;
1155    case 16:  printf("a0    "); break;
1156    case 17:  printf("a1    "); break;
1157    case 18:  printf("a2    "); break;
1158    case 19:  printf("a3    "); break;
1159    case 20:  printf("a4    "); break;
1160    case 21:  printf("a5    "); break;
1161    case 22:  printf("t8    "); break;
1162    case 23:  printf("t9    "); break;
1163    case 24:  printf("t10   "); break;
1164    case 25:  printf("t11   "); break;
1165    case 26:  printf("ra    "); break;
1166    case 27:  printf("t12   "); break;
1167    case 28:  printf("at    "); break;
```

```
1168    case 29:  printf("gp    "); break;
1169    case 30:  printf("sp    "); break;
1170    case 31:  printf("zero  "); break;
1171    case 32:  printf("pc    "); break;
1172    case 33:  printf("vfp   "); break;
1173    }
1174  }
1175
1176  /*********************************************************************/
1177  void print_status(ADDR_TYPE pc, char *buf, data_t *reg, uint64_t tc){
1178    for(int i=0; i<32; i++){
1179      print_register_name(i);
1180      printf("r%-2d      0x%-6qx %-8qd\n", i, reg[i].ld(), reg[i].ld());
1181    }
1182
1183    printf("fpcr           0x%-6qx %-8qd\n", 0ull, 0ull);
1184    printf("pc             0x%-6qx %-8qd\n", pc,   pc  );
1185    printf("vfp            0x%-6qx %-8qd\n", 0ull, 0ull);
1186  }
1187
1188
1189  /*************************************************************/
1190  debug::debug(architecture_state *as_t, memory_system *mem_t,
1191               system_config *ms_t, evaluation_result *e_t){
1192    as  =  as_t;
1193    mem = mem_t;
1194    ms  =  ms_t;
1195    e   =   e_t;
1196  }
1197
1198  void debug::debugmode(){
1199    int i,j;
1200    char t_buf[256];
1201    char t_temp[256];
1202    static int stop = 1;
1203
1204    char *buf;
1205    char *temp;
1206
1207    static unsigned long long stop_count = 0;
1208    static unsigned long long stop_address = 0xFFFFFFFFFFFFFFFFull;
1209    unsigned long long address;
1210
1211    buf = t_buf;
1212    temp = t_temp;
1213
1214    if(stop_count==e->retired_inst){
1215      stop = 1;
1216    }
1217
1218    if(stop_address==as->pc.ld()){
1219      stop = 1;
1220    }
1221
1222    if(stop==0) return;
1223
1224    if(stop==1){
1225      data_t d;
1226      INST_TYPE ir;
1227      mem->ld_nbyte(4, &as->pc, &d);
1228      ir = d.ld() & 0xffffffff;
1229
1230      data_t ir_t;
1231      mem->ld_nbyte(4, &as->pc, &ir_t);
1232      printf("TC:%qd  pc:%qx  ir:%08qx\n",
1233             e->retired_inst, as->pc.ld(), ir_t.ld());
```



```
1234
1235      print_status(as->pc.ld(), NULL, as->r, e->retired_inst);
1236
1237      printf("(db)");
1238      fgets(buf,256, stdin);
1239
1240      while(*buf != (char)NULL){
1241
1242        if(!strncmp(buf,"quit",4)) exit(1);
1243        if(!strncmp(buf,"h",1)){ /* help */
1244          printf("  command\n");
1245          printf("  c followd by number:  stop specified tc.\n");
1246          printf("  s followd by address: stop specified address.\n");
1247          printf("  x followd by address: examine specified address.\n");
1248          printf("  quit: quit simulate.\n");
1249        }
1250
1251        if(*(buf)=='\n') break;
1252
1253        if(*(buf)=='x'){
1254          buf++;
1255          address = get_hex(buf);
1256          address = address >> 4; /* I don't know why? */
1257
1258          for(j=0; j<40; j++){
1259            data_t tmp;
1260            printf("0x%x%x: ",(int)(address >> 32),(int)(address & MASK32));
1261            for(i=0; i<2; i++){
1262              data_t adr;
1263              adr.st(address);
1264              mem->ld_nbyte(8, &adr, &tmp);
1265              printf("0x%08x%08x ", (int)(tmp.ld() >> 32),
1266                     (int)(tmp.ld() & MASK32));
1267              address +=8;
1268            }
1269            printf("\n");
1270          }
1271        }
1272
1273        if(*(buf)=='c'){ /* stop by total count */
1274          buf++;
1275          stop_count = atol(buf); /* for int use atoi */
1276          printf("stop_tc = %qd\n", stop_count);
1277          stop=0;
1278          break;
1279        }
1280
1281        if(*(buf)=='s'){ /* stop by address */
1282          buf++;
1283          stop_address  = get_hex(buf);
1284          stop_address  = stop_address >> 4;
1285          printf("stop_address = 0x%qx\n", stop_address);
1286          stop=0;
1287          break;
1288        }
1289
1290        printf("\n(db)");
1291        fgets(buf,256,stdin);
1292      }
1293    }
1294  }
```

syscall.cc

```
1295  /**** Yet Another Alpha Simulator: SimAlpha        by Kenji KISE ****/
1296  /*********************************************************************/
```

```
#include "define.h"

/*************************************************************************/
int system_manager::translate_fd(DATA_TYPE fd){
  int fd_t = -1;
  for(int i=0; i<FD_MAPPING_MAX; i++){
    if(fd_table[i] == fd) {fd_t = i; break;}
  }
  if(fd_t==-1){printf("*** Error in FD mapping\n"); exit(1);}
  return fd_t;
}

/*************************************************************************/
system_manager::system_manager(architecture_state *as_t,
                               memory_system *mem_t,
                               struct system_config *ms_t,
                               struct evaluation_result *e_t){
  as  =  as_t;
  mem = mem_t;
  ms  =  ms_t;
  e   =   e_t;

  uniq     = 0;
  running  = 1;

  for(int i=0; i<FD_MAPPING_MAX; i++) fd_table[i]= i+1;

  /*** This part should be modified for each application. ***/
  /*** fd_table[6]=7; maps real fd(6) to logical fd(7).   ***/
  fd_table[0] = 0; // STDIN
  fd_table[1] = 1; // STDOUT
  fd_table[2] = 2; // STDERR
}

/*************************************************************************/
void system_manager::execute_pal(class instruction *p){
  INST_TYPE ir = p->get_ir();
  int64_t ret=0;
  int F = ms->syscall_flag;

  if((ir & 0xFFFF)!=0x0083){
    switch((ir & 0xFFFF)){
    case 0x009e: // rdunique(Read Unique Value)
      as->r[0].init(uniq);
      if(F) printf("%8qd: call_pal rduniq = %llx\n",
                   e->retired_inst, uniq);
      return;
    case 0x009f: // wrunique(Write Unique Value)
      uniq  = as->r[16].ld();
      if(F) printf("%8qd: call_pal wrruniq = %llx\n",
                   e->retired_inst, as->r[16].ld());
      return;
    default:
      if(F){
        printf("*** PAL (ir & 0xFFFF)!=0x0083  Function %x",
               (ir & 0xffff));
        printf("  TC:%qd default. %llx\n",
              e->retired_inst, as->r[0].ld());
      }
      return;
    }
  }

  if(as->r[0].ld()==0){
    as->r[19].init(0);
    return;
```



```
  }


  switch(as->r[0].ld()){
    /*************************************************************/
  case ASYS_UNIMPLEMENT:
    {
      ret = -1;
      as->r[0].init(0);
      as->r[19].init(ret);
      if(F) printf("%8qd: SYS_226 = %lld\n",
                   e->retired_inst, ret);
      break;
    }
    /*************************************************************/
  case ASYS_unlink:
    {
      char buf[512]; /** file name **/
      for(int i=0; i<=512; i++){
        data_t d;
        data_t adr;
        adr.init(as->r[16].ld()+i);
        mem->ld_nbyte(1, &adr, &d);
        buf[i] = d.ld();
        if(buf[i]=='\0') break;
      }
      buf[1]='S';
      buf[2]='A';

      ret = unlink(buf);
      if(ret==-1){
        as->r[0].init(errno);
        as->r[19].init(ret);
      }
      else{
        as->r[19].init(0);
      }

      if(F) printf("%8qd: unlink(%s) = %qd\n",
                   e->retired_inst, buf, ret);
      break;
    }
    /*************************************************************/
  case ASYS_open:
    {
      int fd_t;

      char buf[512]; /** file name **/
      for(int i=0; i<=512; i++){
        data_t d;
        data_t adr;
        adr.init(as->r[16].ld()+i);
        mem->ld_nbyte(1, &adr, &d);
        buf[i] = d.ld();
        if(buf[i]=='\0') break;
      }
      //      if(as->r[17].ld()==0){ /** RDONLY **/
      if(as->r[17].ld()==0 || as->r[17].ld()==2){ /** RDONLY or RDWR **/
        //      printf("*** %d %d %d \n", O_RDONLY, O_WRONLY, O_RDWR);
        fd_t = open(buf, as->r[17].ld(), as->r[18].ld());/* open() here */
      }
      else{
        buf[1]='S';
        buf[2]='A';
        int flag = as->r[17].ld() | O_CREAT;
        fd_t = open(buf, flag, as->r[18].ld()); /* open() here */
```

```
1429        }
1430
1431        /* mapping from the real to the logical */
1432        if(fd_t==-1){
1433          ret = -1;
1434          as->r[0].init(errno);
1435          as->r[19].init(~0ull);
1436        }
1437        else{
1438          ret = fd_table[fd_t];
1439          as->r[0].init(ret);
1440          as->r[19].init(0);
1441        }
1442
1443        if(F) printf("%8qd: open(\"%s\", 0x%qx, 0x%qx) = logic %qd(real %d)\n",
1444                     e->retired_inst,
1445                     buf, as->r[17].ld(), as->r[18].ld(),
1446                     ret, fd_t);
1447      }
1448      break;
1449      /*************************************************************/
1450    case ASYS_sigreturn: /*** Note that sigreturn updates as->pc ***/
1451      {
1452        uint64_t pt[OSF_SIGCONTEXT_SIZE/8];
1453        char *buf = (char *)pt;
1454
1455        data_t d;
1456        data_t adr;
1457        for(int i=0; i<OSF_SIGCONTEXT_SIZE; i++){
1458          adr.init(as->r[16].ld() + i);
1459          mem->ld_nbyte(1, &adr, &d);
1460          buf[i] = d.ld();
1461        }
1462        for(int i=0; i<31; i++) as->r[i].init(pt[i+4]);
1463        for(int i=0; i<31; i++) as->f[i].init(pt[i+37]);
1464        as->pc.init(pt[2]);  /** Update PC **/
1465
1466        if(F) printf("%8qd: sigreturn(0x%qx), next pc:0x%qx\n",
1467                     e->retired_inst, as->r[16].ld(), as->pc.ld());
1468        break;
1469      }
1470
1471      /*************************************************************/
1472    case ASYS_close:
1473      {
1474        int fd_t = translate_fd(as->r[16].ld());
1475
1476        if(fd_t!=0 && fd_t!=1 && fd_t!=2){
1477          ret = close(fd_t);
1478          as->r[0].init(ret);
1479        }
1480        as->r[19].init(0);
1481        if(F) printf("%8qd: close(%qd(%d)) = %qd\n",
1482                     e->retired_inst, as->r[16].ld(),
1483                     fd_t, ret);
1484        break;
1485      }
1486    case ASYS_getpagesize:
1487      {
1488        ret = 8192; /* getpagesize(); */
1489        as->r[0].init(ret);
1490        as->r[19].init(0);
1491        if(F) printf("%8qd: getpagesize() = 0x%llx\n",
1492                     e->retired_inst, ret);
1493        break;
1494      }
```



```
1495
1496      /*************************************************************/
1497    case ASYS_sigprocmask:
1498      ret = 0;
1499      as->r[0].init(ret);
1500      as->r[19].init(ret);
1501      if(F) printf("%8qd: sigprocmask(%qd,%qx) = %qd\n",
1502                   e->retired_inst, as->r[16].ld(),
1503                   as->r[17].ld(), ret);
1504      break;
1505
1506      /*************************************************************/
1507    case ASYS_getrusage: // Not Checked
1508      {
1509        struct rusage ru;
1510        ret = getrusage((enum __rusage_who)as->r[16].ld(), &ru);
1511
1512        for(int i=0; i<OSF_RUSAGE_SIZE; i++){
1513          data_t adr; adr.init(as->r[17].ld()+i);
1514          data_t dat; dat.init(0);
1515          mem->st_nbyte(1, &adr, &dat);
1516        }
1517
1518        as->r[0].init(ret);
1519        as->r[19].init(0);
1520
1521        if(F) printf("%8qd: getrusage(%qd, %qx) = %qd\n",
1522                     e->retired_inst,
1523                     as->r[16].ld(), as->r[17].ld(), ret);
1524        break;
1525      }
1526
1527      /*************************************************************/
1528    case  ASYS_gettimeofday:
1529      {
1530        as->r[19].init(0);
1531        if(F) printf("%8qd: gettimeofday(%qd,%qx) = %qd\n",
1532                     e->retired_inst, as->r[16].ld(),
1533                     as->r[17].ld(), ret);
1534        break;
1535      }
1536
1537      /*************************************************************/
1538    case ASYS_fcntl:
1539      {
1540        ret = 0;
1541        as->r[0].init(ret);
1542        as->r[19].init(0);
1543        if(F) printf("%8qd: fcntl(%qd,%qx) = %qd\n",
1544                     e->retired_inst, as->r[16].ld(),
1545                     as->r[17].ld(), ret);
1546        break;
1547      }
1548
1549      /*************************************************************/
1550    case ASYS_sigaction:
1551      {
1552        ret = 0;
1553        as->r[0].init(ret);
1554        as->r[18].init(as->r[17].ld());
1555        as->r[19].init(0);
1556        if(F) printf("%8qd: sigaction(%qx,%qx,%qx,%qx) = %qd\n",
1557                     e->retired_inst,
1558                     as->r[16].ld(), as->r[17].ld(),
1559                     as->r[18].ld(), as->r[19].ld(), ret);
1560        break;
```

```
1561      }
1562
1563      /*************************************************************/
1564    case ASYS_osf_getsysinfo:
1565      {
1566        ret = 0;
1567        as->r[0].init(ret);
1568        as->r[19].init(0);
1569        if(F) printf("%8qd: osf_getsysinfo(%qx,%qx,%qx,%qx) = %qd\n",
1570                     e->retired_inst,
1571                     as->r[16].ld(), as->r[17].ld(),
1572                     as->r[18].ld(), as->r[19].ld(), ret);
1573        break;
1574      }
1575
1576      /*************************************************************/
1577    case ASYS_osf_setsysinfo:
1578      {
1579        ret = 0;
1580        as->r[0].init(ret);
1581        //    as->r[20]= ret;
1582        as->r[19].init(0);
1583        if(F) printf("%8qd: osf_setsysinfo(%qx,%qx,%qx,%qx) = %qd\n",
1584                     e->retired_inst,
1585                     as->r[16].ld(), as->r[17].ld(),
1586                     as->r[18].ld(), as->r[19].ld(), ret);
1587        break;
1588      }
1589
1590      /*************************************************************/
1591    case ASYS_getuid:
1592      {
1593        ret = getuid();
1594        as->r[0].init(ret);
1595        as->r[19].init(0);
1596        as->r[20].init(geteuid());
1597        if(F) printf("%8qd: getuid() = %qd\n", e->retired_inst, ret);
1598        break;
1599      }
1600
1601      /*************************************************************/
1602    case ASYS_getgid:
1603      {
1604        ret = getgid();
1605        as->r[0].init(ret);
1606        as->r[19].init(0);
1607        as->r[20].init(getegid());
1608        if(F) printf("%8qd: getgid(0x%qx) = %qd\n", e->retired_inst,
1609                     as->r[16].ld(), ret);
1610        break;
1611      }
1612
1613      /*************************************************************/
1614    case ASYS_brk:
1615      {
1616        ret = as->r[16].ld();
1617        as->r[0].init(ret);
1618        as->r[19].init(0);
1619        if(F) printf("%8qd: brk(0x%qx) = %qx\n", e->retired_inst,
1620                     as->r[16].ld(), ret);
1621        break;
1622      }
1623
1624      /*************************************************************/
1625    case ASYS_getpid:
1626      {
```



```
1627        ret = getpid();
1628        as->r[0].init(ret);
1629        as->r[19].init(0);
1630        if(F) printf("%8qd: getpid() = %qx\n",
1631                     e->retired_inst, ret);
1632        break;
1633      }
1634      /*************************************************************/
1635      /*
1636    case ASYS_stat:
1637      {
1638        if(F) printf("%8qd: stat() = %qx\n",
1639                     e->retired_inst, ret);
1640        break;
1641      }
1642      */
1643      /*************************************************************/
1644    case ASYS_fstat:
1645      {
1646        int fd_t = translate_fd(as->r[16].ld());
1647
1648        struct stat sbuf;
1649        ret = fstat(fd_t,  &sbuf);
1650
1651        struct osf_statbuf asbuf;
1652        memset(&asbuf, 0, sizeof(struct osf_statbuf));
1653        asbuf.st_dev     = sbuf.st_dev;
1654        asbuf.st_ino     = sbuf.st_ino;
1655        asbuf.st_mode    = sbuf.st_mode;
1656        asbuf.st_nlink   = sbuf.st_nlink;
1657        asbuf.st_uid     = sbuf.st_uid;
1658        asbuf.st_gid     = sbuf.st_gid;
1659        asbuf.st_rdev    = sbuf.st_rdev;
1660        asbuf.st_size    = sbuf.st_size;
1661        asbuf.st_atime   = sbuf.st_atime;
1662        asbuf.st_mtime   = sbuf.st_mtime;
1663        asbuf.st_ctime   = sbuf.st_ctime;
1664        asbuf.st_blksize = sbuf.st_blksize;
1665        asbuf.st_blocks  = sbuf.st_blocks;
1666
1667        char *buf = (char *)&asbuf;
1668
1669        for(int i=0; i<OSF_FSTAT_SIZE; i++){
1670          data_t adr;
1671          adr.init(as->r[17].ld()+i);
1672          data_t dat;
1673          dat.init(buf[i]);
1674          mem->st_nbyte(1, &adr, &dat);
1675        }
1676
1677        as->r[0].init(ret);
1678        as->r[19].init(0);
1679        if(F) printf("%8qd: fstat(logic %qd(real %d), 0x%qx) = %qd\n",
1680                     e->retired_inst, as->r[16].ld(),
1681                     fd_t, as->r[17].ld(), ret);
1682        break;
1683      }
1684      /*************************************************************/
1685    case ASYS_ioctl: /** now write 0, should modify **/
1686      {
1687        int fd_t = translate_fd(as->r[16].ld());
1688
1689        switch(as->r[17].ld()){
1690        case 0x40067408:      // OSF_TIOCGRETP
1691          {
1692            struct osf_sgttyb {
```

```
1693              uint8_t  sg_ispeed;     /* input speed */
1694              uint8_t  sg_ospeed;     /* output speed */
1695              uint8_t  sg_erase;      /* erase character */
1696              uint8_t  sg_kill;       /* kill character */
1697              uint16_t sg_flags;      /* mode flags */
1698            };
1699
1700            struct termios lbuf;
1701            ret = tcgetattr(fd_t, &lbuf);
1702
1703            /*** store 6 byte, value of 0 ***/
1704            for(int i=0; i<6; i++){
1705              data_t adr; adr.init(as->r[18].ld()+i);
1706              data_t dat; dat.init(0);
1707              mem->st_nbyte(1, &adr, &dat);
1708            }
1709
1710            if(ret!=-1){
1711              as->r[0].init(ret);
1712              as->r[19].init(0); // A3
1713            }
1714            else{
1715              as->r[0].init(errno);
1716              as->r[19].init(ret);
1717            }
1718            break;
1719          }
1720
1721
1722        default:
1723          ret = 0;
1724          as->r[19].init(ret);
1725          if(F) printf("*** ioctl default\n");
1726          break;
1727        }
1728
1729        if(F)
1730          printf("%8qd: ioctl(logic %qd(real %d), 0x%qx, 0x%qx) = 0x%qx\n",
1731                 e->retired_inst, as->r[16].ld(), fd_t,
1732                 as->r[17].ld(), as->r[18].ld(), ret);
1733        break;
1734      }
1735
1736      /*************************************************************/
1737    case ASYS_read:
1738      {
1739        char *buf = (char *)calloc(1, as->r[18].ld());
1740        if(buf==NULL) printf("*** malloc error in syscall.cc::read\n");
1741
1742        int fd_t = translate_fd(as->r[16].ld());
1743        if(ms->fd_stdin_flag && fd_t==0) fd_t=ms->fd_stdin; /* STDIN */
1744
1745        ret = read(fd_t, buf, as->r[18].ld());
1746
1747        /*      for(int i=0; i<ret; i++){  */  // This is better.
1748        for(int i=0; i<(int)as->r[18].ld(); i++){ // simple scalar imple!
1749          DATA_TYPE d = buf[i];
1750          data_t adr; adr.init(as->r[17].ld()+i);
1751          data_t dat; dat.init(d);
1752          mem->st_nbyte(1, &adr, &dat);
1753        }
1754        as->r[0].init(ret);
1755        as->r[19].init(0);
1756        free(buf);
1757
1758        if(F) printf("%8qd: read(logic %qd(real %d),%qx,0x%qx) = 0x%qx\n",
```



```
1759                   e->retired_inst, as->r[16].ld(),
1760                   fd_t, as->r[17].ld(), as->r[18].ld(),
1761                   ret);
1762        break;
1763      }
1764
1765      /*************************************************************/
1766    case ASYS_write:
1767      {
1768        char *buf = (char *)malloc(as->r[18].ld());
1769        if(buf==NULL) printf("*** malloc error in syscall.cc::write\n");
1770
1771        for(int i=0; i<(int)as->r[18].ld(); i++){
1772          data_t d;
1773          data_t adr;
1774          adr.init(as->r[17].ld()+i);
1775          mem->ld_nbyte(1, &adr, &d);
1776          buf[i] = d.ld();
1777        }
1778
1779        int fd_t = translate_fd(as->r[16].ld());
1780
1781        ret = write(fd_t, (void *)buf, (long)as->r[18].ld());
1782        as->r[0].init(ret);
1783        as->r[19].init(0);
1784        free(buf);
1785
1786        if(F) printf("%8qd: write(%qd(%d), 0x%qx, %qd) = %qd\n",
1787                   e->retired_inst, as->r[16].ld(),
1788                   fd_t, as->r[17].ld(), as->r[18].ld(), ret);
1789        break;
1790      }
1791
1792
1793      /*************************************************************/
1794    case ASYS_lseek:
1795      {
1796        int fd_t = translate_fd(as->r[16].ld());
1797        if(ms->fd_stdin_flag && fd_t==0) fd_t=ms->fd_stdin; /* STDIN */
1798
1799        ret = lseek(fd_t, as->r[17].ld(), as->r[18].ld());
1800        if(ret==-1){
1801          as->r[0].init(errno);
1802          as->r[19].init(~0);
1803        }
1804        else{
1805          as->r[0].init(ret);
1806          as->r[19].init(0);
1807        }
1808        if(F) printf("%8qd: lseek(logic %qd(real %d), %qd, %qd) = 0x%qx\n",
1809                   e->retired_inst,
1810                   as->r[16].ld(), fd_t,
1811                   as->r[17].ld(), as->r[18].ld(), ret);
1812        break;
1813      }
1814
1815      /*************************************************************/
1816    case ASYS_getrlimit:
1817      {
1818        ret = 0;
1819        as->r[19].init(ret);
1820        if(F) printf("%8qd: getrlimit(%qd,%qd,%qd) = %qd\n",
1821                   e->retired_inst, as->r[16].ld(),
1822                   as->r[17].ld(), as->r[18].ld(), ret);
1823        break;
1824      }
```

```
1825
1826      /*************************************************************/
1827    case ASYS_setrlimit:
1828      {
1829        ret = 0;
1830        as->r[19].init(ret);
1831        if(F) printf("%8qd: setrlimit(%qd,%qd,%qd) = %qd\n",
1832                     e->retired_inst, as->r[16].ld(),
1833                     as->r[17].ld(), as->r[18].ld(), ret);
1834        break;
1835      }
1836
1837      /*************************************************************/
1838    case ASYS_exit:
1839      {
1840        ret = 0;
1841        running = 0;
1842        if(F) printf("%8qd: exit() = %qd\n", e->retired_inst, ret);
1843        break;
1844      }
1845
1846    default:
1847      {
1848        printf("TC: %qd systemcall not implemented!!: %qd\n" ,
1849               e->retired_inst, as->r[0].ld());
1850        as->r[0].init(0);
1851        as->r[19].init((uint64_t)-1);
1852      }
1853    }
1854
1855    fflush(stdout);
1856  }
```

arithmetic.cc

```
1857  /**** Yet Another Alpha Simulator: SimAlpha        by Kenji KISE ****/
1858  /*********************************************************************/
1859  #include "define.h"
1860
1861  /* 64 bit multiply, Input of A and B, output of High 64b and Low 64b. */
1862  /*********************************************************************/
1863  void mul128(uint64_t a, uint64_t b, uint64_t *rh, uint64_t *rl){
1864    uint64_t ah,al, bh, bl;
1865    uint64_t x,y,z;
1866    int carry=0;  /* has a value of 0, 1 and 2 */
1867
1868    ah = a >> 32;
1869    al = a & 0xFFFFFFFF;
1870
1871    bh = b >> 32;
1872    bl = b & 0xFFFFFFFF;
1873
1874    x = al*bl;
1875    y = (al*bh << 32);
1876    z = x + y;
1877    if(z<x || z<y) carry ++;
1878    x = z;
1879    y = (ah*bl << 32);
1880    z = x + y;
1881    if(z<x || z<y) carry ++;
1882    *rl = z;
1883
1884    *rh = ah*bh+ (al*bh >> 32) + (ah*bl >> 32) + carry;
1885  }
1886
1887  /* For the branch instruction, calculate branch taken or not.        */
```



```
1888  /*********************************************************************/
1889  int Condition(int code, DATA_TYPE Rav){
1890    int taken=0;
1891    double Fav=0.0;
1892    //  signal(SIGFPE, (void *)signal_float_branch);
1893    switch(code){
1894    case 0x0: taken = 1;                break;
1895    case 0x4: taken = 1;                break;
1896    case 0x9: taken = (Rav==0);         break;
1897    case 0xe: taken = ((int64_t)Rav>=0); break;
1898    case 0xf: taken = ((int64_t)Rav>0);  break;
1899    case 0x8: taken = (!(Rav & 1));     break;
1900    case 0xc: taken = (  Rav & 1 );     break;
1901    case 0xb: taken = ((int64_t)Rav<=0); break;
1902    case 0xa: taken = ((int64_t)Rav<0 ); break;
1903    case 0xd: taken = (Rav!=0);         break;
1904
1905    case 0x1: memcpy(&Fav,&Rav,8); taken = (Fav==0.0); break;
1906    case 0x2: memcpy(&Fav,&Rav,8); taken = (Fav< 0.0); break;
1907    case 0x3: memcpy(&Fav,&Rav,8); taken = (Fav<=0.0); break;
1908    case 0x5: memcpy(&Fav,&Rav,8); taken = (Fav!=0.0); break;
1909    case 0x6: memcpy(&Fav,&Rav,8); taken = (Fav>=0.0); break;
1910    case 0x7: memcpy(&Fav,&Rav,8); taken = (Fav> 0.0); break;
1911    default: fprintf(stderr, "*** ex_branch default error!\n");
1912    }
1913    //  printf("code=4'h%x; Rav=64'h%016qx; result=%d; #10;\n",
1914    //     code, Rav, taken);
1915    //  signal(SIGFPE, SIG_IGN);
1916    return taken;
1917  }
1918
1919  /* Jump & Branch Unit, Update TPC                                    */
1920  /*********************************************************************/
1921  int BRU(INST_TYPE ir, data_t *R1, data_t *R2, data_t *npc, data_t *tpc){
1922    int Op = (ir>>26) & 0x3F;
1923    tpc->st(Condition(Op&0xF, R1->ld()) ?
1924            npc->ld() + R2->ld() :  npc->ld());
1925
1926    return 0;
1927  }
1928
1929  /*********************************************************************/
1930  int MUL_value(INST_TYPE ir,
1931                DATA_TYPE Rav, DATA_TYPE Rbv, DATA_TYPE *W){
1932    DATA_TYPE Rcv, tmp2, tmp3;
1933    int func7 = (ir>>5) & 0x7f;
1934
1935    switch(func7)
1936      {
1937      case 0x00: /*__MULL; */
1938        Rcv = (Rav*Rbv) & 0x00000000FFFFFFFFllu;
1939        if(Rcv & BIT31) Rcv |= EXTND32;
1940        break;
1941
1942      case 0x20: /*__MULQ; */
1943        mul128(Rav, Rbv, &tmp2, &tmp3);
1944        Rcv = tmp3;
1945        break;
1946
1947      case 0x30: /*__UMULH; */
1948        mul128(Rav, Rbv, &tmp2, &tmp3);
1949        Rcv = tmp2;
1950        break;
1951
1952      default:
1953        printf("** Warning Code 0x13 default. func11 %x\n",func7);
```

```
1954        Rcv = 0;
1955      }
1956
1957    *W = Rcv;
1958    return 0;
1959  }
1960
1961  /* Rc <- BYTE_ZAP(Rav, byte_mask)                                    */
1962  /*********************************************************************/
1963  uint64_t byte_zap(uint64_t Rav, int byte_mask){
1964    if(byte_mask & BIT0) Rav &=  ZAP0;
1965    if(byte_mask & BIT1) Rav &=  ZAP1;
1966    if(byte_mask & BIT2) Rav &=  ZAP2;
1967    if(byte_mask & BIT3) Rav &=  ZAP3;
1968    if(byte_mask & BIT4) Rav &=  ZAP4;
1969    if(byte_mask & BIT5) Rav &=  ZAP5;
1970    if(byte_mask & BIT6) Rav &=  ZAP6;
1971    if(byte_mask & BIT7) Rav &=  ZAP7;
1972    return Rav;
1973  }
1974
1975
1976  /* Rc <- BYTE_ZAP(Rav, byte_mask)                                    */
1977  /*********************************************************************/
1978  int ShiftUnit_value(INST_TYPE ir,
1979                      DATA_TYPE Rav, DATA_TYPE Rbv, DATA_TYPE *R){
1980    int func7 = (ir>>5) & 0x7f;
1981    DATA_TYPE Rcv=0;
1982    unsigned int byte_mask, byte_loc;
1983
1984    switch(func7){
1985    case 0x34: Rcv=(uint64_t)Rav >> (Rbv & 0x3F); break; // SRL
1986    case 0x39: Rcv=(uint64_t)Rav << (Rbv & 0x3F); break; // SLL
1987    case 0x3c: Rcv=( int64_t)Rav >> (Rbv & 0x3F); break; // SRA
1988
1989    case 0x06: /*__EXTBL; */
1990      Rcv = (uint64_t)Rav >> ((Rbv & 0x7)<<3);
1991      Rcv &= 0x00000000000000FFllu;
1992      break;
1993    case 0x16: /*__EXTWL; */
1994      Rcv = (uint64_t)Rav >> ((Rbv & 0x7)<<3);
1995      Rcv &= 0x000000000000FFFFllu;
1996      break;
1997    case 0x26: /*__EXTLL; */
1998      Rcv = (uint64_t)Rav >> ((Rbv & 0x7)<<3);
1999      Rcv &= 0x00000000FFFFFFFFllu;
2000      break;
2001
2002    case 0x36: /*__EXTQL; */
2003      Rcv = (uint64_t)Rav >> ((Rbv & 0x7)<<3);
2004      break;
2005    case 0x5a: /*__EXTWH; */
2006      Rcv = (uint64_t)Rav << (64-((Rbv & 0x7)<<3));
2007      Rcv &= 0x000000000000FFFFllu;
2008      break;
2009    case 0x6a: /*__EXTLH; */
2010      Rcv = (uint64_t)Rav << (64-((Rbv & 0x7)<<3));
2011      Rcv &= 0x00000000FFFFFFFFllu;
2012      break;
2013    case 0x7a: /*__EXTQH; */
2014      Rcv = (uint64_t)Rav << (64-((Rbv & 0x7)<<3));
2015      break;
2016
2017    case 0x0b: /*__INSBL; */
2018      byte_mask = 0x00000001ull << (Rbv & 0x7);
2019      byte_loc  = ((Rbv & 0x7) << 3);
```



```
2020      Rcv = byte_zap((Rav << byte_loc), ~byte_mask);
2021      break;
2022    case 0x1b: /*__INSWL; */
2023      byte_mask = 0x00000003ull << (Rbv & 0x7);
2024      byte_loc  = ((Rbv & 0x7) << 3);
2025      Rcv = byte_zap((Rav << byte_loc), ~byte_mask);
2026      break;
2027    case 0x2b: /*__INSLL; */
2028      byte_mask = 0x0000000Full << (Rbv & 0x7);
2029      byte_loc  = ((Rbv & 0x7) << 3);
2030      Rcv = byte_zap((Rav << byte_loc), ~byte_mask);
2031      break;
2032    case 0x3b: /*__INSQL; */
2033      byte_mask = 0x000000FFull << (Rbv & 0x7);
2034      byte_loc  = ((Rbv & 0x7) << 3);
2035      Rcv = byte_zap((Rav << byte_loc), ~byte_mask);
2036      break;
2037
2038    case 0x57: /*__INSWH; */
2039      byte_mask = ((0x00000003ull << (Rbv & 0x7))>>8);
2040      byte_loc  = (64 - ((Rbv & 0x7)<<3)) & 0x3f;
2041      Rcv = byte_zap((Rav >> byte_loc), ~byte_mask);
2042      break;
2043    case 0x67: /*__INSLH; */
2044      byte_mask = ((0x0000000Full << (Rbv & 0x7))>>8);
2045      byte_loc  = (64 - ((Rbv & 0x7)<<3)) & 0x3f;
2046      Rcv = byte_zap((Rav >> byte_loc), ~byte_mask);
2047      break;
2048    case 0x77: /*__INSQH; */
2049      byte_mask = ((0x000000FFull << (Rbv & 0x7))>>8);
2050      byte_loc  = (64 - ((Rbv & 0x7)<<3)) & 0x3f;
2051      Rcv = byte_zap((Rav >> byte_loc), ~byte_mask);
2052      break;
2053
2054    case 0x02: /*__MSKBL; */
2055      byte_mask = 0x00000001ull << (Rbv & 0x7);
2056      Rcv = byte_zap(Rav, byte_mask);
2057      break;
2058    case 0x12: /*__MSKWL; */
2059      byte_mask = 0x00000003ull << (Rbv & 0x7);
2060      Rcv = byte_zap(Rav, byte_mask);
2061      break;
2062    case 0x22: /*__MSKLL; */
2063      byte_mask = 0x0000000Full << (Rbv & 0x7);
2064      Rcv = byte_zap(Rav, byte_mask);
2065      break;
2066    case 0x32: /*__MSKQL; */
2067      byte_mask = 0x000000FFull << (Rbv & 0x7);
2068      Rcv = byte_zap(Rav, byte_mask);
2069      break;
2070    case 0x52: /*__MSKWH; */
2071      byte_mask = (0x00000003ull << (Rbv & 0x7)) >> 8;
2072      Rcv = byte_zap(Rav, byte_mask);
2073      break;
2074    case 0x62: /*__MSKLH; */
2075      byte_mask = (0x0000000Full << (Rbv & 0x7)) >> 8;
2076      Rcv = byte_zap(Rav, byte_mask);
2077      break;
2078    case 0x72: /*__MSKQH; */
2079      byte_mask = (0x000000FFull << (Rbv & 0x7)) >> 8;
2080      Rcv = byte_zap(Rav, byte_mask);
2081      break;
2082
2083    case 0x30: /*__ZAP; */
2084      Rcv = byte_zap(Rav,  Rbv);
2085      break;
```

```
2086    case 0x31: /*__ZAPNOT; */
2087      Rcv = byte_zap(Rav, ~Rbv);
2088      break;
2089
2090    default:
2091      printf("In logic.cc Warning Code 0x12 defalult. func11 0x%u\n",
2092             func7);
2093      Rcv = 0;
2094    }
2095
2096    *R = Rcv;
2097    return 0;
2098  }
2099
2100
2101
2102
2103  /* Integer Arithmetic and Logic Unit                                 */
2104  /*********************************************************************/
2105  int ALU_value(INST_TYPE ir, DATA_TYPE Rav, int Racv,
2106                DATA_TYPE Rbv, DATA_TYPE *w, int *cmov){
2107    DATA_TYPE Rcv=0;
2108    int cm = 0;
2109
2110    int Op = (ir>>26) & 0x3F;
2111    int func7 = (ir>>5) & 0x7F;
2112
2113    switch(Op){
2114    case 0x0a: Rcv = Rav + Rbv; break; /**** LD ****/
2115    case 0x0b: Rcv = (Rav + Rbv) & ~7; break;
2116    case 0x0c: Rcv = Rav + Rbv; break;
2117    case 0x20: Rcv = Rav + Rbv; break;
2118    case 0x21: Rcv = Rav + Rbv; break;
2119    case 0x22: Rcv = Rav + Rbv; break;
2120    case 0x23: Rcv = Rav + Rbv; break;
2121    case 0x28: Rcv = Rav + Rbv; break;
2122    case 0x29: Rcv = Rav + Rbv; break;
2123    case 0x2a: Rcv = Rav + Rbv; break;
2124    case 0x2b: Rcv = Rav + Rbv; break;
2125
2126    case 0x0d: Rcv = Rav + Rbv; break; /**** ST ****/
2127    case 0x0e: Rcv = Rav + Rbv; break;
2128    case 0x0f: Rcv = (Rav + Rbv) & ~7; break;
2129    case 0x24: Rcv = Rav + Rbv; break;
2130    case 0x25: Rcv = Rav + Rbv; break;
2131    case 0x26: Rcv = Rav + Rbv; break;
2132    case 0x27: Rcv = Rav + Rbv; break;
2133    case 0x2c: Rcv = Rav + Rbv; break;
2134    case 0x2d: Rcv = Rav + Rbv; break;
2135    case 0x2e: Rcv = Rav + Rbv; break;
2136    case 0x2f: Rcv = Rav + Rbv; break;
2137
2138    case 0x08: Rcv = Rav + Rbv; break; /* LDA  */
2139    case 0x09: Rcv = Rav + Rbv; break; /* LDAH */
2140
2141    case OP_INTA:
2142      switch(func7){
2143      case 0x02:
2144      case 0x0b:
2145      case 0x22:
2146      case 0x2b: Rav = (Rav << 2); break;
2147      case 0x12:
2148      case 0x1b:
2149      case 0x32:
2150      case 0x3b: Rav = (Rav << 3); break;
2151      }
```



```
2152
2153      switch(func7)
2154        {
2155        case 0x00: Rcv = Rav + Rbv; break;
2156        case 0x20: Rcv = Rav + Rbv; break;
2157        case 0x02: Rcv = Rav + Rbv; break;
2158        case 0x22: Rcv = Rav + Rbv; break;
2159        case 0x12: Rcv = Rav + Rbv; break;
2160        case 0x32: Rcv = Rav + Rbv; break;
2161
2162        case 0x09: Rcv = Rav - Rbv; break;
2163        case 0x29: Rcv = Rav - Rbv; break;
2164        case 0x0b: Rcv = Rav - Rbv; break;
2165        case 0x2b: Rcv = Rav - Rbv; break;
2166        case 0x1b: Rcv = Rav - Rbv; break;
2167        case 0x3b: Rcv = Rav - Rbv; break;
2168
2169        case 0x2d: Rcv = ((int64_t)Rav==(int64_t)Rbv); break;
2170        case 0x4d: Rcv = ((int64_t)Rav< (int64_t)Rbv); break;
2171        case 0x6d: Rcv = ((int64_t)Rav<=(int64_t)Rbv); break;
2172        case 0x3d: Rcv = (Rav <= Rbv); break;
2173        case 0x1d: Rcv = (Rav <  Rbv); break;
2174
2175        case 0x0f: /*__CMPBGE; */
2176          Rcv=0;
2177          if((Rav & ~ZAP0) >= (Rbv & ~ZAP0)) Rcv |= BIT0;
2178          if((Rav & ~ZAP1) >= (Rbv & ~ZAP1)) Rcv |= BIT1;
2179          if((Rav & ~ZAP2) >= (Rbv & ~ZAP2)) Rcv |= BIT2;
2180          if((Rav & ~ZAP3) >= (Rbv & ~ZAP3)) Rcv |= BIT3;
2181          if((Rav & ~ZAP4) >= (Rbv & ~ZAP4)) Rcv |= BIT4;
2182          if((Rav & ~ZAP5) >= (Rbv & ~ZAP5)) Rcv |= BIT5;
2183          if((Rav & ~ZAP6) >= (Rbv & ~ZAP6)) Rcv |= BIT6;
2184          if((Rav & ~ZAP7) >= (Rbv & ~ZAP7)) Rcv |= BIT7;
2185          break;
2186
2187        default: printf("Warning in 0x10 arithmetic\n");  Rcv=0;
2188        }
2189
2190      switch(func7){
2191      case 0x00:
2192      case 0x09:
2193      case 0x02:
2194      case 0x12:
2195      case 0x0b:
2196      case 0x1b: if(Rcv & BIT31) Rcv |= EXTND32; else Rcv &= MASK32;
2197        break;
2198      }
2199      break;
2200
2201    case OP_INTL:
2202      switch(func7){
2203      case 0x08:
2204      case 0x28:
2205      case 0x48: Rbv = ~Rbv; break;
2206      }
2207
2208      switch(func7){
2209      case 0x00: Rcv = Rav & Rbv; break;
2210      case 0x08: Rcv = Rav & Rbv; break;
2211      case 0x20: Rcv = Rav | Rbv; break;
2212      case 0x28: Rcv = Rav | Rbv; break;
2213      case 0x40: Rcv = Rav ^ Rbv; break;
2214      case 0x48: Rcv = Rav ^ Rbv; break;
2215
2216      case 0x14: Rcv=Rbv; cm=(  Rav & 1);       break;
2217      case 0x16: Rcv=Rbv; cm=(!(Rav & 1));      break;
```

```
2218      case 0x24: Rcv=Rbv; cm=(Rav==0);          break;
2219      case 0x26: Rcv=Rbv; cm=(Rav!=0);          break;
2220      case 0x44: Rcv=Rbv; cm=((int64_t)Rav< 0); break;
2221      case 0x46: Rcv=Rbv; cm=((int64_t)Rav>=0); break;
2222      case 0x64: Rcv=Rbv; cm=((int64_t)Rav<=0); break;
2223      case 0x66: Rcv=Rbv; cm=((int64_t)Rav> 0); break;
2224
2225      case 0x61: Rcv=Rbv & ~CPU_FEATURE_MASK;    break; /* AMASK   */
2226      case 0x6c: Rcv=IMPLEMENTATION_VERSION;     break; /* IMPLVER */
2227
2228        /** CMOVb: This code is assigned by Kenji KISE **/
2229      case 0x6e: Rcv = (Racv) ? Rbv : Rav;   break;
2230
2231      default:
2232        printf("Warning 0x11 Func:0x%x Default Case.\n",func7);
2233        Rcv=0; break;
2234      }
2235      break;
2236    }
2237
2238    *w    = Rcv;
2239    *cmov = cm;
2240    return 0;
2241  }
2242
2243  /* Floating Arithmetic and Logic Unit                                 */
2244  /**********************************************************************/
2245  int FALU_value(INST_TYPE ir, DATA_TYPE Rav, int Racv,
2246                 DATA_TYPE Rbv, DATA_TYPE *w, int *cmov){
2247
2248    DATA_TYPE Rcv=0;
2249
2250    int code   = (ir>>26) & 0x3F;
2251    int func11 = (ir>>5) & 0x7FF;
2252
2253    double Fav   = 0.0, Fbv   = 0.0, Fcv   = 0.0;
2254
2255    signal(SIGFPE, SIG_IGN); // Ignore Floagint Exceptions
2256
2257    memcpy(&Fav, &Rav, 8);
2258    memcpy(&Fbv, &Rbv, 8);
2259
2260    switch(code){
2261
2262    case OP_ITFP: /*** Integer to floating-point register move ***/
2263      switch(func11){
2264      case 0x024: // ITOFT
2265        //      if(Rav==0) Rav=0xaae4d; /* ????? */
2266        memcpy(&Fcv, &Rav, 8);
2267        break;
2268
2269      case 0x004: // ITOFS
2270        {
2271          DATA_TYPE tmp = 0;
2272          tmp = ((Rav & BIT31) << 32) |
2273            (map_s(Rav) << 52) |
2274            ((Rav & 0x7fffff) << 29);
2275          memcpy(&Fcv, &tmp, 8);
2276        }
2277        break;
2278
2279      case 0x0ab: // SQRTT
2280        {
2281          Fcv = sqrt(Fbv);
2282        }
2283        break;
```



```
2284
2285      default:
2286        printf("Warning 0x14 Floating Default. func11 %x \n", func11);
2287        exit(1);
2288        Fcv=0.0;
2289      }
2290      memcpy(&Rcv, &Fcv, 8);
2291      break;
2292
2293    case OP_FPTI: /*** Floating-point to integer register move ***/
2294      switch(func11){
2295      case 0x000: // SEXTB(Sign Extend Byte): Rc <= SEXT(Rbv[7:0])
2296        Rcv = (Rbv & BIT7) ? Rbv |= EXTND8 : (Rbv & 0xffull);
2297        break;
2298      case 0x001: // SEXTW(Sign Extend Word): Rc <= SEXT(Rbv[15:0])
2299        Rcv = (Rbv & BIT15) ? Rbv |= EXTND16 : (Rbv & 0xffffull);
2300        break;
2301      case 0x070: // FTOIT: Rc <= Fav
2302        Rcv = Rav;
2303        break;
2304      case 0x078: // FTOIS: Rc <= Fav
2305        Rcv = ((Rav>>32) & 0xc0000000) | ((Rav >> 29) & 0x3fffffff);
2306        if(Rav & BIT63) Rcv |= EXTND32;
2307        break;
2308
2309      default:
2310        printf("Warning 0x1c Floating Default. func11 %x \n", func11);
2311        exit(1);
2312      }
2313      break;
2314
2315    case OP_FLTV: /*** VAX floating-point instruction ***/
2316      printf("Warning 0x15 Floating Default. func11 %x \n", func11);
2317      exit(1);
2318
2319    case OP_FLTI: { /*** IEEE floating-point instruction ***/
2320        switch(func11){
2321
2322        case 0x080:
2323        case 0x000:
2324        case 0x040:
2325        case 0x0c0:
2326        case 0x180:
2327        case 0x100:
2328        case 0x140:
2329        case 0x1c0:
2330        case 0x580:
2331        case 0x500:
2332        case 0x540:
2333        case 0x5c0:
2334        case 0x780:
2335        case 0x700:
2336        case 0x740:
2337        case 0x7c0: Fcv = Fav + Fbv; break; break; // ADDS
2338
2339        case 0x0a0:
2340        case 0x020:
2341        case 0x060:
2342        case 0x0e0:
2343        case 0x1a0:
2344        case 0x120:
2345        case 0x160:
2346        case 0x1e0:
2347        case 0x5a0:
2348        case 0x520:
2349        case 0x560:
```

```
2350        case 0x5e0:
2351        case 0x7a0:
2352        case 0x720:
2353        case 0x760:
2354        case 0x7e0: Fcv = Fav + Fbv; break; // ADDT
2355
2356        case 0x0a5:
2357        case 0x5a5: // CMPTEQ
2358          Fcv = (Fav==Fbv) ? 2.0 : 0.0; break;
2359
2360        case 0x0a6:
2361        case 0x5a6: // CMPTLT
2362          Fcv = (Fav< Fbv) ? 2.0 : 0.0; break;
2363
2364        case 0x0a7:
2365        case 0x5a7: // CMPTLE
2366          Fcv = (Fav<=Fbv) ? 2.0 : 0.0; break;
2367
2368        case 0x0a4:
2369        case 0x5a4: // CMPTUN
2370          Fcv = (isnan(Fav) || isnan(Fbv)) ? 2.0 : 0.0; break;
2371
2372        case 0x0bc:
2373        case 0x03c:
2374        case 0x07c:
2375        case 0x0fc:
2376        case 0x7bc:
2377        case 0x73c:
2378        case 0x77c:
2379        case 0x7fc: // CVTQS
2380          Fcv = (double)(int64_t)Rbv;
2381          break;
2382
2383        case 0x0be:
2384        case 0x03e:
2385        case 0x07e:
2386        case 0x0fe:
2387        case 0x7be:
2388        case 0x73e:
2389        case 0x77e:
2390        case 0x7fe: // CVTQT
2391          Fcv = (double)(int64_t)Rbv;
2392          //      printf("*** CVTQT  Fcv %f Rbv %llx \n", Fcv, Rbv);
2393          break;
2394
2395        case 0x0ac:
2396        case 0x02c:
2397        case 0x06c:
2398        case 0x0ec:
2399        case 0x1ac:
2400        case 0x12c:
2401        case 0x16c:
2402        case 0x1ec:
2403        case 0x5ac:
2404        case 0x52c:
2405        case 0x56c:
2406        case 0x5ec:
2407        case 0x7ac:
2408        case 0x72c:
2409        case 0x76c:
2410        case 0x7ec: // CVTTS
2411          {
2412            memcpy(&Fcv, &Rbv, 8);
2413            float tmp = Fcv;
2414            Fcv = tmp;
2415          }
```



```
2416          break;
2417
2418        case 0x083:
2419        case 0x003:
2420        case 0x043:
2421        case 0x0c3:
2422        case 0x183:
2423        case 0x103:
2424        case 0x143:
2425        case 0x1c3:
2426        case 0x583:
2427        case 0x503:
2428        case 0x543:
2429        case 0x5c3:
2430        case 0x783:
2431        case 0x703:
2432        case 0x743:
2433        case 0x7c3: // DIVS
2434          if(Fbv==0.0){
2435            Fcv = 0.0;
2436            break;
2437          }
2438          Fcv = Fav / Fbv;
2439          break;
2440
2441        case 0x0a3:
2442        case 0x023:
2443        case 0x063:
2444        case 0x0e3:
2445        case 0x1a3:
2446        case 0x123:
2447        case 0x163:
2448        case 0x1e3:
2449        case 0x5a3:
2450        case 0x523:
2451        case 0x563:
2452        case 0x5e3:
2453        case 0x7a3:
2454        case 0x723:
2455        case 0x763:
2456        case 0x7e3: // DIVT
2457          if(Fbv==0.0){
2458            Fcv = 0.0;
2459            break;
2460          }
2461          Fcv = (double)(Fav / Fbv);
2462          break;
2463
2464        case 0x082:
2465        case 0x002:
2466        case 0x042:
2467        case 0x0c2:
2468        case 0x182:
2469        case 0x102:
2470        case 0x142:
2471        case 0x1c2:
2472        case 0x582:
2473        case 0x502:
2474        case 0x542:
2475        case 0x5c2:
2476        case 0x782:
2477        case 0x702:
2478        case 0x742:
2479        case 0x7c2: // MULS
2480          Fcv = Fav * Fbv;   break;
2481
```

```
2482        case 0x0a2:
2483        case 0x022:
2484        case 0x062:
2485        case 0x0e2:
2486        case 0x1a2:
2487        case 0x122:
2488        case 0x162:
2489        case 0x1e2:
2490        case 0x5a2:
2491        case 0x522:
2492        case 0x562:
2493        case 0x5e2:
2494        case 0x7a2:
2495        case 0x722:
2496        case 0x762:
2497        case 0x7e2: // MULT
2498          Fcv = Fav * Fbv;   break;
2499
2500        case 0x081:
2501        case 0x001:
2502        case 0x041:
2503        case 0x0c1:
2504        case 0x181:
2505        case 0x101:
2506        case 0x141:
2507        case 0x1c1:
2508        case 0x581:
2509        case 0x501:
2510        case 0x541:
2511        case 0x5c1:
2512        case 0x781:
2513        case 0x701:
2514        case 0x741:
2515        case 0x7c1: // SUBS
2516          Fcv = Fav - Fbv; break;
2517
2518        case 0x0a1:
2519        case 0x021:
2520        case 0x061:
2521        case 0x0e1:
2522        case 0x1a1:
2523        case 0x121:
2524        case 0x161:
2525        case 0x1e1:
2526        case 0x5a1:
2527        case 0x521:
2528        case 0x561:
2529        case 0x5e1:
2530        case 0x7a1:
2531        case 0x721:
2532        case 0x761:
2533        case 0x7e1: // SUBT
2534          Fcv = Fav - Fbv; break;
2535
2536        case 0x0af:
2537        case 0x02f:
2538        case 0x1af:
2539        case 0x12f:
2540        case 0x5af:
2541        case 0x52f:
2542        case 0x7af:
2543        case 0x72f:
2544        case 0x0ef:
2545        case 0x1ef:
2546        case 0x5ef:
2547        case 0x7ef:
```



```
2548        case 0x06f:
2549        case 0x16f:
2550        case 0x56f:
2551        case 0x76f: // CVTTQ
2552          {
2553            int64_t tmp;
2554            tmp = (int64_t)Fbv;
2555            memcpy(&Fcv, &tmp, 8);
2556          }
2557          break;
2558
2559        default:
2560          printf("Warning 0x16 Floating Default. func11 %x \n", func11);
2561          exit(1);
2562          Fcv=0.0;
2563        }
2564        memcpy(&Rcv, &Fcv, 8);
2565
2566        break;
2567      }
2568
2569
2570    case OP_FLTL: /*** Floating-point instruction ***/
2571      {
2572        switch(func11){
2573        case 0x020: /*__CPYS;  Copy Sign */
2574          {
2575            uint64_t tmp=0;
2576            tmp |= (Rav &  BIT63);
2577            tmp |= (Rbv & ~BIT63);
2578            memcpy(&Fcv, &tmp, 8);
2579          }
2580          /*
2581          tmp  = Rav;
2582          tmp2 = Rbv;
2583          tmp2 = ((tmp2 << 1) >> 1);
2584          if(tmp & BIT63) tmp2 |= BIT63;
2585          memcpy(&Fcv, &tmp2, 8);
2586          */
2587          break;
2588
2589        case 0x021: /*__CPYSN; Copy Sign Negative */
2590          {
2591            uint64_t tmp=0;
2592            tmp |= (~Rav &  BIT63);
2593            tmp |= ( Rbv & ~BIT63);
2594            memcpy(&Fcv, &tmp, 8);
2595          }
2596          break;
2597
2598        case 0x022: /*__CPYSE; Copy Sign and Exponent */
2599          {
2600            uint64_t tmp=0;
2601            tmp |= (Rav &  0xFFF0000000000000llu);
2602            tmp |= (Rbv & ~0xFFF0000000000000llu);
2603            memcpy(&Fcv, &tmp, 8);
2604          }
2605          /*
2606          tmp  = Rav;
2607          tmp2 = Rbv;
2608          tmp  &= 0xFFF0000000000000llu;
2609          tmp2 &= 0x000FFFFFFFFFFFFFllu;
2610          tmp |= tmp2;
2611          memcpy(&Fcv, &tmp, 8);
2612          */
2613          break;
```

```
2614
2615        case 0x010: /*__CVTLQ; */
2616          {
2617            uint64_t tmp = 0;
2618            tmp  = 0;
2619            tmp |= ((Rbv>>32) & 0xc0000000ull);
2620            tmp |= ((Rbv>>29) & 0x3fffffffull);
2621            if(tmp & BIT31) tmp |= EXTND32;
2622            memcpy(&Fcv, &tmp, 8);
2623          }
2624          break;
2625
2626        case 0x030: /*__CVTQL; */
2627          {
2628            DATA_TYPE tmp = 0;
2629            /* Next line is needed. SimpleScalar has a bug?  */
2630            tmp |=  (((Rbv << 32) & 0xc000000000000000ull));
2631            tmp |=  (((Rbv << 29) & 0x07fffffe0000000ull));
2632            memcpy(&Fcv, &tmp, 8);
2633            break;
2634          }
2635
2636        case 0x24: /* __MT_FPCR */
2637          /* Move Fav to Floating-point Control Register
2638             Return 0.0 for the simplicity.
2639          */
2640          Fcv=0.0; break;
2641
2642        case 0x25: /* __MF_FPCR */
2643          /* Move from Floating-point Control Register to Fa
2644             Return 0.0 for the simplicity.
2645          */
2646          Fcv=0.0; break;
2647
2648        case 0x02a:/*__FCMOVEQ; */
2649          Fcv = Fbv; *cmov = (Fav==0.0);
2650          break;
2651
2652        case 0x02b:/*__FCMOVNE; */
2653          Fcv = Fbv; *cmov=(Fav!=0.0);
2654          break;
2655
2656          /** CMOVb: This is assigned by Kenji KISE. **/
2657        case 0x333:
2658          Fcv = (Racv) ? Fbv : Fav;
2659          //      printf("** Condition %d  Fb Fa %qx %qx\n", R1->cmov, Fbv, Fav
);
2660          break;
2661
2662        default:
2663          printf("Warning Floating 0x17 code default: func11 0x%x\n", func11);
2664          exit(1);
2665          Fcv=0.0; break;
2666        }
2667        memcpy(&Rcv, &Fcv, 8);
2668        break;
2669      }
2670    }
2671
2672    *w = Rcv;
2673    return 0;
2674  }
2675
2676  /* calculate Rcv                                                      */
2677  /*********************************************************************/
2678  int ALU(INST_TYPE ir, data_t *R1, data_t *R2, data_t *Rw){
```



```
2679    DATA_TYPE Rcv =0;
2680    DATA_TYPE Rav =R1->ld();
2681    DATA_TYPE Rbv =R2->ld();
2682    int Racm = R1->cmov;
2683    int cm=0;
2684
2685    int Op = (ir>>26) & 0x3F;
2686
2687    switch(Op){
2688    case OP_JSR:
2689
2690    case 0x30: /** BR **/
2691    case 0x31:
2692    case 0x32:
2693    case 0x33:
2694    case 0x34:
2695    case 0x35:
2696    case 0x36:
2697    case 0x37:
2698    case 0x38:
2699    case 0x39:
2700    case 0x3a:
2701    case 0x3b:
2702    case 0x3c:
2703    case 0x3d:
2704    case 0x3e:
2705    case 0x3f:    Rcv = Rbv; break;
2706
2707    case OP_PAL : Rcv=0; break;
2708    case OP_MISC: Rcv=0; break;
2709    case OP_INTS: ShiftUnit_value(ir, Rav, Rbv, &Rcv); break;
2710    case OP_INTM: MUL_value(ir, Rav, Rbv, &Rcv); break;
2711    case OP_LDA :
2712    case OP_LDAH:
2713    case OP_INTA:
2714    case OP_INTL: ALU_value(ir, Rav, Racm, Rbv, &Rcv, &cm); break;
2715    case OP_ITFP:
2716    case OP_FLTV:
2717    case OP_FLTI:
2718    case OP_FLTL:
2719    case OP_FPTI: FALU_value(ir, Rav, Racm, Rbv, &Rcv, &cm); break;
2720    default:      ALU_value(ir, Rav, Racm, Rbv, &Rcv, &cm); break;
2721    }
2722
2723    Rw->cmov = cm;
2724    Rw->st(Rcv);
2725
2726    return 0;
2727  }
```