

Nov 05, 08 18:55

code.txt

Page 1/62

```

file name: Makefile
1 #####
2 ## SimMips: Simple Computer Simulator of MIPS Arch Lab. TOKYO TECH ##
3 #####
4 CC      = g++
5 OFLAG   = -O3 -Wall
6 LFLAG   = -lncurses
7 DEBUG   = -g
8
9 TARGET  = SimMips
10 HEADER  = define.h
11 SOURCE  = main.cc board.cc memory.cc simloader.cc mips.cc mipsinst.cc cp0.cc device.cc
12 OBJECT  = $(SOURCE:.cc=.o)
13 #####
14 all:
15     $(MAKE) $(TARGET)
16 #####
17 $(TARGET): $(SOURCE) $(HEADER) Makefile
18     $(CC) $(OFLAG) -o $@ $(SOURCE) $(LFLAG)
19 #####
20 debug:
21     $(CC) $(DEBUG) -o $(TARGET) $(SOURCE) $(LFLAG)
22 #####
23 .SUFFIXES :
24 .SUFFIXES : .o .cc
25
26 .cc.o:
27     $(CC) $(OFLAG) -c $<
28
29 $(OBJECT) : $(HEADER) Makefile
30 #####
31 wc:
32     wc -l $(HEADER) $(SOURCE)
33
34 indent:
35     indent -kr -ts4 -nut main.cc
36
37 text:
38     cats Makefile > code.txt
39     echo -e "\nFile Organization by [wc *.h *.cc]" >> code.txt
40     echo -e "  lines words bytes" >> code.txt
41     wc $(HEADER) $(SOURCE) >> code.txt
42     echo -e "\f" >> code.txt
43     cats -f $(HEADER) $(SOURCE) >> code.txt
44     a2ps --medium=a4 -f 6.5 code.txt -o code.ps
45     ps2pdf13 -sPAPERSIZE=a4 code.ps
46     rm -f code.txt code.ps
47
48 cflow:
49     cflow *.cc
50
51 clean:
52     rm -f *.o *.~ *.exe $(TARGET) code.cc code.ps code.pdf
53 #####
54 run:
55     ./$(TARGET) test/qsort
56 runlinux:
57     ./$(TARGET) -M test/mem_gemu.txt test/vmlinux-2.6.18-3-gemu
58 runmieru:
59     ./$(TARGET) -M test/mem_mieru.txt test/tokei_mieru

```

File Organization by [wc *.h *.cc]

	lines	words	bytes
744	1643	17945	define.h
21	49	663	main.cc
622	1715	17590	board.cc
297	874	8731	memory.cc
227	671	6300	simloader.cc
899	2743	25776	mips.cc
767	2712	22781	mipsinst.cc
309	1004	10110	cp0.cc
536	1490	14681	device.cc
4422	12901	124577	total

Nov 05, 08 18:55

code.txt

Page 2/62

```

file name: define.h
1 /*****
2  * SimMips: Simple Computer Simulator of MIPS Arch Lab. TOKYO TECH *
3  *****/
4 #define L_NAME "SimMips: Simple Computer Simulator of MIPS"
5 #define L_VER "Version 0.5.0 2008-11-05"
6
7 /*****/
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11 #include <math.h>
12 #include <unistd.h>
13 #include <fcntl.h>
14 #include <termios.h>
15 #include <signal.h>
16 #include <elf.h>
17 #include <sys/stat.h>
18 #include <sys/time.h>
19
20 /*****/
21 typedef unsigned int uint;
22 typedef signed long long llint;
23 typedef unsigned long long ullint;
24
25 typedef signed char      int008_t;
26 typedef signed short     int016_t;
27 typedef signed int       int032_t;
28 typedef signed long long int064_t;
29 typedef unsigned char    uint008_t;
30 typedef unsigned short   uint016_t;
31 typedef unsigned int     uint032_t;
32 typedef unsigned long long uint064_t;
33
34 /*****/
35 #define DELETE(obj) {delete obj; obj = NULL;}
36 #define DELETE_ARRAY(arr) {delete[] arr; arr = NULL;}
37
38 /*****/
39 enum {
40     CPU_STOP = 0,
41     CPU_START = 1,
42     CPU_IF = 1,
43     CPU_ID = 2,
44     CPU_RF = 3,
45     CPU_EX = 4,
46     CPU_MS = 5,
47     CPU_MR = 6,
48     CPU_WB = 7,
49     CPU_WAIT = 100,
50     CPU_ERROR = -1,
51
52     MAX_CYCLE_DEF = 0x7fffffff,
53     MAX_DEBUG_MODE = 4,
54     DEB_RESULT = 1,
55     DEB_INST = 2,
56     DEB_REG = 3,
57     DEB_EXCTLB = 4,
58
59     NREG = 32,
60     NCREG = 256,
61     TLB_ENTRY = 16,
62     MNEMONIC_BUF_SIZE = 128,
63     INST_CODE_NUM = 107,
64
65     MEM_SIZE_DEF = 0x08000000, // 128MiB
66     PAGE_SIZE = 0x00001000, // 4KiB
67     KSEG0_MIN = 0x80000000,
68     KSEG1_MIN = 0xa0000000,
69     KSEG2_MIN = 0xc0000000,
70     UNMAP_MASK = 0x1fffffff,
71 };
72
73 /* board.cc *****/
74 class Chip;
75 class Mips;

```

Nov 05, 08 18:55

code.txt

Page 3/62

```

76 class MipsCp0;
77 class MainMemory;
78 class MemoryController;
79 class MemoryMap;
80
81 /*****
82 class ttyControl {
83     private:
84         struct termios oldterm;
85         int oldfflg;
86         int valid;
87     public:
88         ttyControl();
89         ~ttyControl();
90 };
91
92 /*****
93 class Board {
94     private:
95         ullint maxcycle;
96         char *binfile, *memfile;
97         ttyControl *tttyc;
98
99         void usage();
100         ullint atoi_postfix(const char *);
101         void checkarg(int, char**);
102         int loadrawfile(char *, uint032_t);
103         char *getlinehead(char *, FILE *);
104         FILE *openmemfile();
105         void setdefaultmap();
106         int setmemorymap();
107         int setinitialdata();
108         void printresult();
109
110     public:
111         int debug_mode, imix_mode, multicycle, use_cp0, use_tttyc;
112         Chip *chip;
113         MemoryMap *mmap;
114
115         Board();
116         ~Board();
117         ullint gettime();
118         int siminit(char *);
119         int siminit(int, char **);
120         void exec();
121 };
122
123 /*****
124 class Chip {
125     private:
126         Board *board;
127         MemoryMap *mmap;
128
129     public:
130         ullint cycle, maxcycle;
131         int ready;
132
133         Mips *mips;
134         MipsCp0 *cp0;
135         MemoryController *mc;
136
137         Chip(Board *, int, int);
138         ~Chip();
139
140         int step_funct();
141         int step_multi();
142         int getState();
143 };
144
145 /* mipsinst.cc *****/
146 class MipsInst {
147     private:
148         char mnemonic[MNEMONIC_BUF_SIZE];
149         char *instname;
150
151     public:
152         uint032_t ir, op, pc;

```

Nov 05, 08 18:55

code.txt

Page 4/62

```

153     uint attr;
154     uint opcode, rs, rt, rd, shamt, funct, imm, addr;
155     uint code_l, code_s, sel;
156     uint latency;
157
158     MipsInst();
159     void decode();
160     char *getinstname();
161     char *getmnemonic();
162     void clearmnemonic();
163 };
164
165 /* mips.cc *****/
166 class MipsArchstate {
167     public:
168         uint032_t pc, delay_npc;
169         uint032_t r[NREG];
170         uint032_t hi, lo;
171
172         MipsArchstate();
173         void print();
174 };
175
176 /*****
177 class MipsSimstate {
178     public:
179         ullint inst_count;
180         ullint imix[INST_CODE_NUM];
181
182         MipsSimstate();
183         void print();
184 };
185
186 /*****
187 class Mips {
188     private:
189         Board *board;
190         Chip *chip;
191         MemoryController *mc;
192         MipsCp0 *cp0;
193
194         uint032_t rrs, rrt, rrd, rhi, rlo;
195         uint032_t npc, vaddr;
196         uint064_t paddr;
197         int cond;
198         int mcid;
199         int exc_occure, exc_code;
200         uint wait_cycle;
201
202         void fetch();
203         void decode();
204         void regfetch();
205         void execute();
206         void memsend();
207         void memreceive();
208         void writeback();
209         void setnpc();
210         void exception(int);
211         void proceedstate();
212         void syscall();
213
214     public:
215         MipsArchstate *as;
216         MipsSimstate *ss;
217         MipsInst *inst;
218         int state;
219
220         Mips(Board *, Chip *);
221         ~Mips();
222
223         int step_funct();
224         int step_multi();
225         int running();
226 };
227
228 /* cp0.cc *****/
229 class MipsTlbEntry {

```

Nov 05, 08 18:55

code.txt

Page 5/62

```

230 public:
231     uint vpn2;
232     uint asid;
233     uint pagemask;
234     uint pageshift;
235     uint global;
236     uint pfn[2];
237     uint valid[2];
238     uint dirty[2];
239     uint cache[2];
240
241     MipsTlbEntry();
242     void print();
243 };
244
245 /*****
246 class MipsCp0 {
247     private:
248         Board *board;
249         Chip *chip;
250         MipsTlbEntry tlb[TLB_ENTRY];
251         uint032_t r[NCREG];
252         int counter, divisor;
253         int gettlbentry(uint032_t);
254         void regprint();
255         void tlbprint();
256     public:
257         MipsCp0(Board *, Chip *, int);
258         void step();
259         void tlbread();
260         void tlbwrite(int);
261         void tlblookup();
262         uint032_t readreg(int);
263         void writereg(int, uint032_t);
264         void modifyreg(int, uint032_t, uint032_t);
265         int getphaddr(uint032_t, uint064_t *, int);
266         uint032_t doexception(int, uint032_t, uint032_t, int);
267         void setinterrupt(int);
268         void clearinterrupt(int);
269         int checkinterrupt();
270         void print();
271 };
272
273
274 /* device.cc *****/
275 class MMDevice {
276     public:
277         virtual ~MMDevice() {}
278
279         virtual void init(Board *) {}
280         virtual void fini() {}
281         virtual void step() {}
282         virtual void readlb(const uint032_t, uint008_t*) {}
283         virtual void read2b(const uint032_t, uint016_t*) {}
284         virtual void read4b(const uint032_t, uint032_t*) {}
285         virtual void read8b(const uint032_t, uint064_t*) {}
286         virtual void writelb(const uint032_t, const uint008_t) {}
287         virtual void write2b(const uint032_t, const uint016_t) {}
288         virtual void write4b(const uint032_t, const uint032_t) {}
289         virtual void write8b(const uint032_t, const uint064_t) {}
290         virtual void print() {}
291 };
292
293 /*****
294 class IntController {
295     private:
296         MipsCp0 *cp0;
297         uint008_t imr[2], irr[2], isr[2];
298         int tobe_read[2], init_mode[2];
299
300         int checkaddr(uint032_t);
301         void recalcirq();
302     public:
303         IntController(MipsCp0 *);
304
305         void readlb(const uint032_t, uint008_t *);

```

Nov 05, 08 18:55

code.txt

Page 6/62

```

307     void writelb(const uint032_t, const uint008_t);
308     void setinterrupt(int);
309     void clearinterrupt(int);
310 };
311
312 /*****
313 class SerialIO {
314     private:
315         IntController *pic;
316         uint008_t ier, iir, lcr, mcr, scr;
317         int currentchar;
318         uint divisor;
319         int counter;
320
321         int charavail();
322         void recalcirq();
323     public:
324         SerialIO(IntController *);
325
326         void step();
327         void readlb(const uint032_t, uint008_t *);
328         void writelb(const uint032_t, const uint008_t);
329 };
330
331 /*****
332 class IsaIO : public MMDevice {
333     private:
334         IntController *pic;
335         SerialIO *sio;
336     public:
337         IsaIO();
338         ~IsaIO();
339
340         void init(Board *);
341         void step();
342         void readlb(const uint032_t, uint008_t *);
343         void writelb(const uint032_t, const uint008_t);
344 };
345
346 /*****
347 class MieruIO : public MMDevice {
348     private:
349         Board *board;
350         int lcd_width, lcd_height, cursorx, cursory;
351         char lcdbuf[100];
352         int lcdindex;
353
354         void lcd_ttyopen();
355         void lcd_ttyclose();
356         void lcd_setcolor(int);
357         void lcd_cls();
358         int lcd_printf(char *, ...);
359         void lcd_nextline();
360
361     public:
362         void init(Board *);
363         void fini();
364         void readlb(const uint032_t, uint008_t *);
365         void read4b(const uint032_t, uint032_t *);
366         void writelb(const uint032_t, const uint008_t);
367         void write4b(const uint032_t, const uint032_t);
368 };
369
370
371 /* memory.cc *****/
372 class MainMemory : public MMDevice {
373     private:
374         uint032_t mem_size, npage;
375         uint032_t *pagetable;
376         int *external;
377         uint032_t *newpage(const uint032_t);
378         uint032_t *getrealaddr(const uint032_t);
379     public:
380         MainMemory(uint032_t);
381         ~MainMemory();
382         void readlb(const uint032_t, uint008_t*);

```

Nov 05, 08 18:55

code.txt

Page 7/62

```

384 void read2b(const uint032_t, uint016_t*);
385 void read4b(const uint032_t, uint032_t*);
386 void read8b(const uint032_t, uint064_t*);
387 void readnb(const uint032_t, int, uint008_t*);
388 void write1b(const uint032_t, const uint008_t);
389 void write2b(const uint032_t, const uint016_t);
390 void write4b(const uint032_t, const uint032_t);
391 void write8b(const uint032_t, const uint064_t);
392 void writenb(const uint032_t, int, uint008_t*);
393 uint032_t *setpageentry(const uint032_t, uint032_t*);
394 void print();
395 };
396
397 /*****/
398 enum {
399     NUM_MCINST = 2,
400     MC_BUFFERMODE = 0,
401     MC_THROUGHMODE = 1,
402
403     MCI_FAILURE = -1,
404     MCI_NONE = 0,
405     MCI_PEND = 1,
406     MCI_FINISH = 2,
407 };
408 /*****/
409 class MemoryMap {
410 public:
411     uint032_t addr;
412     uint032_t size;
413     MMDevice *dev;
414     MemoryMap *next;
415
416     MemoryMap();
417     ~MemoryMap();
418 };
419
420 /*****/
421 class McInst {
422 public:
423     int state;
424     int op;
425     uint064_t addr;
426     uint032_t size;
427     uint008_t data008;
428     uint016_t data016;
429     uint032_t data032;
430     uint064_t data064;
431
432     McInst();
433 };
434
435 /*****/
436 class MemoryController {
437 private:
438     MemoryMap *mmap;
439     int mode;
440     int head, tail;
441
442 public:
443     McInst inst[NUM_MCINST];
444
445     MemoryController(MemoryMap *, int);
446     int enqueue(uint064_t, uint032_t, void *);
447     void step();
448     void print();
449 };
450
451 /* simloader.cc *****/
452 typedef struct {
453     uint032_t addr;
454     uint008_t data;
455 } memtab_t;
456
457 /*****/
458 typedef enum {
459     ST_NOTYPE,
460     ST_FUNC,

```

Nov 05, 08 18:55

code.txt

Page 8/62

```

461 ST_OBJECT,
462 } symtype_t;
463
464 /*****/
465 typedef struct {
466     uint032_t addr;
467     symtype_t type;
468     char *name;
469 } symtab_t;
470
471 /*****/
472 class SimLoader {
473 public:
474     memtab_t *memtab;
475     symtab_t *symtab;
476
477     int fileident;
478     int filetype;
479     int endian;
480     int archtype;
481     int dynamic;
482     uint032_t entry;
483     uint032_t stackptr;
484     int memtabnum;
485     int symtabnum;
486
487     SimLoader();
488     ~SimLoader();
489     int loadfile(char*);
490     int checkfile(const char *);
491     int loadelf32(const char *);
492 };
493
494 /* Exception Code Definition *****/
495 /*****/
496 enum {
497     EXC_INT__ = 0, // Interrupt
498     EXC_MOD__ = 1, // Modifying read-only page
499     EXC_TLBL__ = 2, // TLB miss load
500     EXC_TLBS__ = 3, // TLB miss store
501     EXC_ADEL__ = 4, // Address error load
502     EXC_ADES__ = 5, // Address error store
503     EXC_IBE__ = 6, // Bus error inst
504     EXC_DBE__ = 7, // Bus error data
505     EXC_SYSCALL = 8, // System call
506     EXC_BP__ = 9, // Breakpoint
507     EXC_RI__ = 10, // Illegal instruction
508     EXC_CPU__ = 11, // Coprocessor unavailable
509     EXC_OV__ = 12, // Overflow
510     EXC_TRAP__ = 13, // Trap instruction
511     // # Codes below are NOT architectually defined
512     EXC_TLBRFL = 0x100, // TLB Refill flag (for TLBL, TLBS)
513     EXC_CPU1__ = 0x200, // CP1 unavailable? (for CPU)
514 };
515
516 /* MIPS Register Name Definition *****/
517 /*****/
518 enum {
519     REG_V0 = 2,
520     REG_A0 = 4,
521     REG_A1 = 5,
522     REG_A2 = 6,
523     REG_A3 = 7,
524     REG_T9 = 25,
525     REG_GP = 28,
526     REG_SP = 29,
527     REG_RA = 31,
528 };
529
530 /*****/
531 const char regname[NREG + 1][3] =
532 { "zr", "at", "v0", "v1", "a0", "a1", "a2", "a3",
533   "t0", "t1", "t2", "t3", "t4", "t5", "t6", "t7",
534   "s0", "s1", "s2", "s3", "s4", "s5", "s6", "s7",
535   "t8", "t9", "k0", "k1", "gp", "sp", "fp", "ra", "pc" };
536
537 /* CP0 Register Definition *****/

```

Nov 05, 08 18:55

code.txt

Page 9/62

```

538 /*****
539 enum {
540     CP0_INDEX__ = 0,
541     CP0_RANDOM__ = 1,
542     CP0_ENTRYLO0 = 2,
543     CP0_ENTRYLO1 = 3,
544     CP0_CONTEXT__ = 4,
545     CP0_PAGEMASK = 5,
546     CP0_WIRED__ = 6,
547     CP0_BADVADDR = 8,
548     CP0_COUNT__ = 9,
549     CP0_ENTRYHI__ = 10,
550     CP0_COMPARE__ = 11,
551     CP0_SR__ = 12,
552     CP0_CAUSE__ = 13,
553     CP0_EPC__ = 14,
554     CP0_PRID__ = 15,
555     CP0_CONFIG__ = 16,
556     CP0_CONFIG1__ = 48,
557
558     SR_EXL_SH = 1,
559     SR_KSU_SH = 3,
560     SR_BEV_SH = 22,
561     SR_EXL_MASK = 0x1,
562     SR_ERLEXL_MASK = 0x3,
563     SR_KSU_MASK = 0x3,
564     SR_BEV_MASK = 0x1,
565     SR_IE_MASK = 0x1,
566
567     CAUSE_EXC_SH = 2,
568     CAUSE_IP_SH = 8,
569     CAUSE_IV_SH = 23,
570     CAUSE_CE_SH = 28,
571     CAUSE_BD_SH = 31,
572     CAUSE_EXC_MASK = 0x1f,
573     CAUSE_IP_MASK = 0xff,
574     CAUSE_IV_MASK = 0x1,
575     CAUSE_CE_MASK = 0x3,
576     CAUSE_BD_MASK = 0x1,
577
578     TLB_VPAGE_SH = 13,
579     TLB_VPAGE_LOWER = 0x1fff,
580     TLB_VPAGE_MASK = 0x7ffff,
581     TLB_PPAGE_SH = 12,
582     CONT_BADV_SH = 4,
583     CONT_BADV_MASK = TLB_VPAGE_MASK,
584 };
585
586 /* System Call Definition *****/
587 /*****
588 enum {
589     SYS_EXIT = 4001,
590     SYS_WRITE = 4004,
591     SYS_IOCTL = 4054,
592 };
593
594 /* Instruction Definition *****/
595 /*****
596 enum {
597     NOP_____,
598     SSNOP_____,
599     SLL_____,
600     SRL_____,
601     SRA_____,
602     SLLV_____,
603     SRLV_____,
604     SRAV_____,
605     JR_____,
606     JR_HB_____,
607     JALR_____,
608     JALR_HB_____,
609     MOVZ_____,
610     MOVN_____,
611     SYSCALL_____,
612     BREAK_____,
613     SYNC_____,

```

Nov 05, 08 18:55

code.txt

Page 10/62

```

615     MFHI_____,
616     MTHI_____,
617     MFLO_____,
618     MTLO_____,
619     MULT_____,
620     MULTU_____,
621     DIV_____,
622     DIVU_____,
623     ADD_____,
624     ADDU_____,
625     SUB_____,
626     SUBU_____,
627     AND_____,
628     OR_____,
629     XOR_____,
630     NOR_____,
631     SLT_____,
632     SLTU_____,
633     TGE_____,
634     TGEU_____,
635     TLT_____,
636     TLTU_____,
637     TEQ_____,
638     TNE_____,
639     BLTZ_____,
640     BGEZ_____,
641     BLTZL_____,
642     BGEZL_____,
643     TGEI_____,
644     TGEIU_____,
645     TLTI_____,
646     TLTIU_____,
647     TEQI_____,
648     TNEI_____,
649     BLTZAL_____,
650     BGEZAL_____,
651     BLTZALL_____,
652     BGEZALL_____,
653     J_____,
654     JAL_____,
655     BEQ_____,
656     BNE_____,
657     BLEZ_____,
658     BGTZ_____,
659     ADDI_____,
660     ADDIU_____,
661     SLTI_____,
662     SLTIU_____,
663     ANDI_____,
664     ORI_____,
665     XORI_____,
666     LUI_____,
667     MFC0_____,
668     CFC0_____,
669     MTC0_____,
670     TLBR_____,
671     TLBWI_____,
672     TLBWR_____,
673     TLBP_____,
674     ERET_____,
675     WAIT_____,
676     BEQL_____,
677     BNEL_____,
678     BLEZL_____,
679     BGTZL_____,
680     MADD_____,
681     MADDU_____,
682     MUL_____,
683     MSUB_____,
684     MSUBU_____,
685     CLZ_____,
686     CLO_____,
687     LB_____,
688     LH_____,
689     LWL_____,
690     LW_____,
691     LBU_____,

```

Nov 05, 08 18:55

code.txt

Page 11/62

```

692     LHU_____,
693     LWR_____,
694     SB_____,
695     SH_____,
696     SWL_____,
697     SW_____,
698     SWR_____,
699     CACHE_____,
700     LL_____,
701     PREF_____,
702     SC_____,
703     FLOAT_OPS,
704     UNDEFINED,
705 };
706
707 /*****
708 enum {
709     READ_NONE = 0x0,
710     READ_RS = 0x1,
711     READ_RT = 0x2,
712     READ_RD = 0x4,
713     READ_HI = 0x8,
714     READ_LO = 0x10,
715     READ_HILO = 0x18,
716     WRITE_NONE = 0x0,
717     WRITE_RS = 0x100,
718     WRITE_RT = 0x200,
719     WRITE_RD = 0x400,
720     WRITE_HI = 0x800,
721     WRITE_LO = 0x1000,
722     WRITE_HILO = 0x1800,
723     WRITE_RD_COND = 0x4000,
724     WRITE_RRA = 0x8000,
725     LOAD_1B = 0x10000,
726     LOAD_2B = 0x20000,
727     LOAD_4B_ALIGN = 0x40000,
728     LOAD_4B_UNALIGN = 0x80000,
729     LOAD_4B = 0xc0000,
730     LOAD_ANY = 0xf0000,
731     STORE_1B = 0x100000,
732     STORE_2B = 0x200000,
733     STORE_4B_ALIGN = 0x400000,
734     STORE_4B_UNALIGN = 0x800000,
735     STORE_4B = 0xc00000,
736     STORE_ANY = 0xf00000,
737     LOADSTORE = 0xff0000,
738     LOADSTORE_4B_UNALIGN = 0x880000,
739     BRANCH = 0x1000000,
740     BRANCH_LIKELY = 0x2000000,
741     BRANCH_ERET = 0x4000000, // BRANCH_NODELAY (eret only has)
742 };
743
744 /*****/

```

Nov 05, 08 18:55

code.txt

Page 12/62

```

file name: main.cc
1  /*****
2  * SimMips: Simple Computer Simulator of MIPS   Arch Lab. TOKYO TECH *
3  *****/
4  #include "define.h"
5
6  /*****/
7  int main(int argc, char *argv[])
8  {
9      printf("## %s %s\n", L_NAME, L_VER);
10
11      Board *board = new Board();
12
13      // execute if successfully initialized
14      if (board->siminit(argc, argv) == 0)
15          board->exec();
16
17      DELETE(board);
18      return 0;
19  }
20
21 /*****/

```

Nov 05, 08 18:55

code.txt

Page 13/62

```

file name: board.cc
1  /*****
2  * SimMips: Simple Computer Simulator of MIPS Arch Lab. TOKYO TECH *
3  *****/
4  #include "define.h"
5
6  #define MEM_HEADER "SimMips_Machine_Setting"
7
8  enum {
9      SR_DEF      = 0x10000000,
10     PAGEMASK_DEF = 0x00001fff,
11     PRID_DEF     = 0x00018001,
12     CONFIG_DEF   = 0x80000082,
13     CONFIG1_DEF  = 0x1ed96c80,
14
15     RUNNING = 0,
16     HALT_CYCLE = 1,
17     HALT_MIPS = 2,
18     HALT_INT = 3,
19
20     HEAD_SIZE = 128,
21 };
22
23 /*****/
24 volatile sig_atomic_t recieve_int = 0;
25
26 void sigint_handler(int signum)
27 {
28     recieve_int = 1;
29 }
30
31 /*****/
32 ttyControl::ttyControl()
33 {
34     struct termios newterm;
35     int newfflg;
36
37     tcgetattr(STDIN_FILENO, &oldterm);
38     oldfflg = fcntl(STDIN_FILENO, F_GETFL);
39
40     if ((oldterm.c_lflag & (ICANON | ECHO)) == 0) {
41         valid = 0;
42         return;
43     }
44
45     newterm = oldterm;
46     newterm.c_lflag &= ~(ICANON | ECHO);
47     newterm.c_cc[VMIN] = 0;
48     newterm.c_cc[VTIME] = 0;
49     newfflg = oldfflg | O_NONBLOCK;
50
51     tcsetattr(STDIN_FILENO, TCSANOW, &newterm);
52     fcntl(STDIN_FILENO, F_SETFL, newfflg);
53     valid = 1;
54 }
55
56 /*****/
57 ttyControl::~ttyControl()
58 {
59     if (valid) {
60         tcsetattr(STDIN_FILENO, TCSANOW, &oldterm);
61         fcntl(STDIN_FILENO, F_SETFL, oldfflg);
62     }
63 }
64
65 /*****/
66 Board::Board()
67 {
68     debug_mode = multicycle = imix_mode = use_cp0 = use_ttyc = 0;
69     maxcycle = MAX_CYCLE_DEF;
70     binfile = memfile = NULL;
71     ttyc = NULL;
72     mmap = NULL;
73 }
74
75 /*****/
76 Board::~Board()

```

Nov 05, 08 18:55

code.txt

Page 14/62

```

77 {
78     DELETE(ttyc);
79     DELETE(chip);
80     DELETE(mmap);
81 }
82
83 /*****/
84 void Board::usage()
85 {
86     char usagemessage[] = "\n";
87     -e[num][kmg]: stop simulation after num cycles executed\n
88     -d[level]: debug mode\n
89     -i: put instruction mix after simulation\n
90     -m: use multi-cycle execution model\n
91     -M [filename]: specify machine setting file\n
92     \n";
93
94     printf("Usage: simmips [-options] object_file_name\n");
95     printf("%s", usagemessage);
96 }
97
98 /*****/
99 ullint Board::atoi_postfix(const char *nptr)
100 {
101     char *endptr;
102     ullint result = strtoll(nptr, &endptr, 10);
103     if ((*endptr == 'k') || (*endptr == 'K'))
104         result *= 1000;
105     else if ((*endptr == 'm') || (*endptr == 'M'))
106         result *= 1000000;
107     else if ((*endptr == 'g') || (*endptr == 'G'))
108         result *= 1000000000;
109     return result;
110 }
111
112 /*****/
113 void Board::checkarg(int argc, char **argv)
114 {
115     int bin_index = -1;
116     int num;
117     char *opt;
118
119     for (int i = 1; (opt = argv[i]) != NULL; i++) {
120         if (opt[0] != '-') {
121             if (bin_index != -1) {
122                 fprintf(stderr, "## multiple binary files(%s, %s)\n",
123                     argv[bin_index], opt);
124                 usage();
125                 return;
126             }
127             bin_index = i;
128             continue;
129         }
130         switch (opt[1]) {
131             case 'd':
132                 num = atoi(&opt[2]);
133                 if (num > MAX_DEBUG_MODE) {
134                     fprintf(stderr, "## debug mode %d not available\n", num);
135                     return;
136                 }
137                 debug_mode = num;
138                 break;
139             case 'e':
140                 maxcycle = atoi_postfix(&opt[2]);
141                 if (!maxcycle)
142                     maxcycle = MAX_CYCLE_DEF;
143                 break;
144             case 'i':
145                 imix_mode = 1;
146                 break;
147             case 'm':
148                 multicycle = 1;
149                 break;
150             case 'M':
151                 if (memfile) {
152                     fprintf(stderr, "## multiple -M options\n");
153                     return;
154                 }

```

Nov 05, 08 18:55

code.txt

Page 15/62

```

154     }
155     if ((memfile = argv[++i]) == NULL) {
156         fprintf(stderr, "## -M option: no file specified\n");
157         return;
158     }
159     break;
160     default:
161         fprintf(stderr, "## -%c: invalid option\n", opt[1]);
162         usage();
163         return;
164     }
165 }
166 if (bin_index == -1) {
167     usage();
168     return;
169 }
170
171 if (debug_mode)
172     printf("## debug mode %d.\n", debug_mode);
173
174 binfile = argv[bin_index];
175 }
176
177 /*****
178 int Board::loaddrawfile(char *filename, uint032_t addr)
179 {
180     FILE *fp;
181     uint008_t temp;
182
183     if ((fp = fopen(filename, "rb")) == NULL) {
184         fprintf(stderr, "## can't open file: %s\n", filename);
185         return 1;
186     }
187     while (fread(&temp, sizeof(temp), 1, fp)) {
188         chip->mc->enqueue(addr, 1, &temp);
189         chip->mc->step();
190         addr++;
191     }
192     if (!feof(fp)) {
193         fprintf(stderr, "## can't load file: %s\n", filename);
194         return 1;
195     }
196     fclose(fp);
197     return 0;
198 }
199
200 /*****
201 char* Board::getlinehead(char *dest, FILE *fp)
202 {
203     memset(dest, '\0', HEAD_SIZE + 1);
204     char *result = fgets(dest, HEAD_SIZE, fp);
205     int len = strlen(dest);
206     if (dest[len - 1] != '\n') {
207         if (dest[len - 1] == '\r')
208             dest[len - 1] = '\0';
209         char poi[HEAD_SIZE];
210         do
211             if (fgets(poi, HEAD_SIZE, fp) == NULL)
212                 break;
213         while (poi[strlen(poi) - 1] != '\n');
214     } else {
215         dest[len - 1] = '\0';
216         if (dest[len - 2] == '\r')
217             dest[len - 2] = '\0';
218     }
219     return result;
220 }
221
222 /*****
223 FILE *Board::openmemfile()
224 {
225     FILE *fp;
226     char line[HEAD_SIZE + 1];
227     line[HEAD_SIZE] = '\0';
228
229     if ((fp = fopen(memfile, "r")) == NULL) {
230         fprintf(stderr, "## can't open file: %s\n", memfile);

```

Nov 05, 08 18:55

code.txt

Page 16/62

```

231         return NULL;
232     }
233     getlinehead(line, fp);
234     if (strncmp(line, MEM_HEADER, strlen(MEM_HEADER)) != 0) {
235         fprintf(stderr, "## not a machine setting file: %s\n", memfile);
236         fclose(fp);
237         return NULL;
238     }
239     return fp;
240 }
241
242 /*****
243 void Board::setdefaultmap()
244 {
245     mmap = new MemoryMap();
246     mmap->addr = 0;
247     mmap->size = MEM_SIZE_DEF;
248     mmap->dev = new MainMemory(MEM_SIZE_DEF);
249 }
250
251 /*****
252 int Board::setmemorymap()
253 {
254     FILE *fp;
255     char line[HEAD_SIZE + 1];
256     uint032_t addr, size;
257     int linecnt = 1;
258     char *endptr;
259     MemoryMap *now = NULL;
260     line[HEAD_SIZE] = '\0';
261
262     if (!memfile) {
263         setdefaultmap();
264         return 0;
265     }
266     if ((fp = openmemfile()) == NULL)
267         return 1;
268
269     while (getlinehead(line, fp) != NULL) {
270         linecnt++;
271         if (strncmp(line, "@map", 4))
272             continue;
273
274         addr = strtoul(&line[5], &endptr, 16);
275         size = strtoul(&endptr[1], &endptr, 16);
276         if (!size) {
277             fprintf(stderr, "## %s:%d: invalid syntax. skip.\n",
278                     memfile, linecnt);
279             continue;
280         }
281         if (!now)
282             now = (mmap = new MemoryMap());
283         else
284             now = (now->next = new MemoryMap());
285         now->addr = addr;
286         now->size = size;
287
288         if (strcmp(&endptr[1], "MAIN_MEMORY") == 0) {
289             now->dev = new MainMemory(size);
290         } else if (strcmp(&endptr[1], "ISA_IO") == 0) {
291             now->dev = new IsaIO();
292             use_cp0 = use_ttyc = 1;
293         } else if (strcmp(&endptr[1], "ISA_BUS") == 0) {
294             now->dev = new MMDevice();
295         } else if (strcmp(&endptr[1], "MIERU_IO") == 0) {
296             now->dev = new MieruIO();
297             use_ttyc = 1;
298         }
299     }
300     if (!now)
301         setdefaultmap();
302     fclose(fp);
303     return 0;
304 }
305
306 /*****
307 int Board::setinitialdata()

```


Nov 05, 08 18:55

code.txt

Page 17/62

```

308 {
309     FILE *fp;
310     char line[HEAD_SIZE + 1];
311     int linecnt = 1, regnum;
312     uint032_t addr, regval;
313     char *endptr;
314     line[HEAD_SIZE] = '\0';
315
316     if (!memfile)
317         return 0;
318
319     if ((fp = openmemfile()) == NULL)
320         return 1;
321
322     while (getlinehead(line, fp) != NULL) {
323         linecnt++;
324         if (line[0] != '@')
325             continue;
326
327         if (strncmp(&line[1], "reg", 3) == 0) {
328             regnum = strtol(&line[6], &endptr, 10);
329             if (!regnum)
330                 for (int i = 1; i < NREG + 1; i++)
331                     if (strncmp(&line[6], regname[i],
332                             strlen(regname[i])) == 0) {
333                         regnum = i;
334                         endptr = &line[6 + strlen(regname[i])];
335                         break;
336                     }
337             if (endptr[0] != '=') {
338                 fprintf(stderr, "## %s:%d: invalid syntax. skip.\n",
339                         memfile, linecnt);
340                 continue;
341             } else {
342                 regval = strtoul(&endptr[1], NULL, 0);
343             }
344             if ((!regnum) && (regnum >= NREG + 1)) {
345                 fprintf(stderr, "## %s:%d: invalid register. skip.\n",
346                         memfile, linecnt);
347             } else if (regnum != NREG) {
348                 chip->mips->as->r[regnum] = regval;
349             } else {
350                 chip->mips->as->pc = regval;
351             }
352             if (strncmp(&line[1], "mem", 3) == 0) {
353                 addr = strtoul(&line[5], &endptr, 16);
354                 loadrawfile(&endptr[1], addr);
355             } else if (strncmp(&line[1], "map", 3) != 0) {
356                 fprintf(stderr, "## %s:%d: unknown command. skip.\n",
357                         memfile, linecnt);
358             }
359         }
360         fclose(fp);
361         return 0;
362     }
363
364     /*****
365     ullint Board::gettime()
366     {
367         static ullint start = 0;
368         ullint tm;
369         struct timeval tv;
370
371         gettimeofday(&tv, NULL);
372         tm = tv.tv_sec * 1000000ull + tv.tv_usec;
373         if (start == 0)
374             start = tm;
375         return tm - start;
376     }
377
378     /*****
379     int Board::siminit(char *arg)
380     {
381         if (arg == NULL)
382             return 1;
383         if (arg[0] == '\0')
384             return 1;

```

Nov 05, 08 18:55

code.txt

Page 18/62

```

385
386     int num, argc = 0;
387     int arglen = strlen(arg);
388     char *head = new char[arglen + 1];
389     memset(head, 0, arglen + 1);
390     char *temp = new char[arglen + 1];
391     strcpy(temp, arg);
392     char flag = ' ';
393
394     for (int i = 0; i < arglen; i++) {
395         if ((temp[i] == '"') || (temp[i] == '\')) {
396             if (flag == temp[i]) {
397                 temp[i] = '\0';
398                 flag = ' ';
399             } else if ((flag != '"') && (flag != '\')) {
400                 head[i + 1] = 1;
401                 argc++;
402                 flag = temp[i];
403             }
404         } else if ((temp[i] == ' ') && (flag == '\0')) {
405             temp[i] = '\0';
406             flag = ' ';
407         } else if ((temp[i] != ' ') && (flag == ' ')) {
408             head[i] = 1;
409             argc++;
410             flag = '\0';
411         }
412     }
413     if ((flag == '"') || (flag == '\')) {
414         fprintf(stderr, "## checkarg(): syntax error\n");
415         return 1;
416     }
417     char **argv = new char*[argc + 2];
418     num = 1;
419     for (int i = 0; i < arglen; i++) {
420         if (head[i]) {
421             argv[num] = &temp[i];
422             num++;
423         }
424     }
425     argv[0] = argv[argc + 1] = NULL;
426
427     int ret = siminit(argc, argv);
428
429     DELETE_ARRAY(head);
430     DELETE_ARRAY(temp);
431     DELETE_ARRAY(argv);
432     return ret;
433 }
434
435     /*****
436     int Board::siminit(int argc, char **argv)
437     {
438         struct sigaction sa;
439
440         checkarg(argc, argv);
441         if (!binfile)
442             return 1;
443         if (setmemorymap())
444             return 1;
445         chip = new Chip(this, use_cp0, multicycle);
446
447         // load ELF image
448         SimLoader *ld = new SimLoader();
449         if (ld->loadfile(binfile))
450             return 1;
451         if ((ld->archtype != EM_MIPS) ||
452             (ld->filetype != ET_EXEC)) { // || (ld->dynamic)) {
453             printf("## ERROR: improper binary: %s\n", binfile);
454             return 1;
455         }
456
457         // set registers' default value
458         chip->mips->as->pc = ld->entry;
459         chip->mips->as->r[REG_SP] = ((ld->stackptr) ? ld->stackptr :
460                                     MEM_SIZE_DEF - 0x100);
461         if (!use_cp0)

```

Nov 05, 08 18:55

code.txt

Page 19/62

```

462 chip->mips->as->r[REG_T9] = chip->mips->as->pc;
463 for (int i = 0; i < ld->syntabnum; i++)
464     if (strcmp(ld->syntab[i].name, "_gp") == 0) {
465         chip->mips->as->r[REG_GP] = ld->syntab[i].addr;
466         break;
467     }
468 if (use_cp0) {
469     chip->cp0->writereg(CP0_SR_____, SR_DEF);
470     chip->cp0->writereg(CP0_PAGEMASK, PAGEMASK_DEF);
471     chip->cp0->writereg(CP0_PRID____, PRID_DEF);
472     chip->cp0->writereg(CP0_CONFIG___, CONFIG_DEF);
473     chip->cp0->writereg(CP0_CONFIG1_, CONFIG1_DEF);
474 }
475 if (setinitialdata())
476     return 1;
477
478 // write ELF image to memory
479 if (use_cp0) {
480     for (int i = 0; i < ld->memtabnum; i++) {
481         if ((ld->memtab[i].addr < KSEG0_MIN) ||
482             (ld->memtab[i].addr >= KSEG2_MIN)) {
483             printf("## ERROR: load to unmapped segment: 0x%08x\n",
484                 ld->memtab[i].addr);
485             return 1;
486         } else {
487             ld->memtab[i].addr &= UNMAP_MASK;
488         }
489     }
490 }
491 for (int i = 0; i < ld->memtabnum; i++) {
492     chip->mc->enqueue(ld->memtab[i].addr, 1, &ld->memtab[i].data);
493     chip->mc->step();
494 }
495 DELETE(ld);
496
497 // set up console and signal
498 if (use_ttyc)
499     ttyc = new ttyControl();
500 sa.sa_handler = sigint_handler;
501 sigemptyset(&sa.sa_mask);
502 sa.sa_flags = 0;
503 sigaction(SIGINT, &sa, NULL);
504
505 for (MemoryMap *temp = mmap; temp != NULL; temp = temp->next)
506     temp->dev->init(this);
507
508 // now, the chip is ready for execution
509 chip->maxcycle = maxcycle;
510 chip->mips->state = CPU_START;
511 chip->ready = 1;
512 return 0;
513 }
514
515 /*****
516 void Board::exec()
517 {
518     gettimeofday();
519     if (multicycle) {
520         while (chip->getstate() == RUNNING)
521             chip->step_multi();
522     } else {
523         while (chip->getstate() == RUNNING)
524             chip->step_func();
525     }
526     printresult();
527 }
528
529 /*****
530 void Board::printresult() {
531     double simtime = (double) gettimeofday() / 1000000.0;
532
533     for (MemoryMap *temp = mmap; temp != NULL; temp = temp->next)
534         temp->dev->fini();
535
536     if (chip->getstate() == HALT_CYCLE)
537         printf("\n## cycle count reaches the limit\n");
538

```

Nov 05, 08 18:55

code.txt

Page 20/62

```

539 else if (chip->getstate() == HALT_MIPS)
540     printf("\n## cpu stopped\n");
541 else // (recieve_int)
542     printf("\n## interrupt\n");
543 printf("## cycle count: %llu\n", chip->cycle);
544 printf("## inst count: %llu\n", chip->mips->ss->inst_count);
545 printf("## simulation time: %8.3f\n", simtime);
546 printf("## mips: %4.3f\n",
547     chip->mips->ss->inst_count / (simtime * 1000000.0));
548
549 if (debug_mode == DEB_RESULT) {
550     chip->mips->as->print();
551     if (chip->cp0)
552         chip->cp0->print();
553     chip->mc->print();
554 }
555 if (imix_mode)
556     chip->mips->ss->print();
557 }
558
559 /*****
560 Chip::Chip(Board *board, int use_cp0, int multi)
561 {
562     cycle = 0;
563     ready = 0;
564
565     this->board = board;
566     this->mmap = board->mmap;
567     if (use_cp0)
568         cp0 = new MipsCp0(board, this, (multi) ? 3 : 1);
569     else
570         cp0 = NULL;
571     mc = new MemoryController(mmap,
572         (multi) ? MC_BUFFERMODE : MC_THROUGHMODE);
573     mips = new Mips(board, this);
574 }
575
576 /*****
577 Chip::~Chip()
578 {
579     DELETE(mips);
580     DELETE(mc);
581     DELETE(cp0);
582 }
583
584 /*****
585 int Chip::step_func()
586 {
587     cycle++;
588     int ret = mips->step_func();
589     if (cp0)
590         cp0->step();
591     for (MemoryMap *temp = mmap; temp != NULL; temp = temp->next)
592         temp->dev->step();
593     return ret;
594 }
595
596 /*****
597 int Chip::step_multi()
598 {
599     cycle++;
600     int ret = mips->step_multi();
601     if (cp0)
602         cp0->step();
603     for (MemoryMap *temp = mmap; temp != NULL; temp = temp->next)
604         temp->dev->step();
605     mc->step();
606     return ret;
607 }
608
609 /*****
610 int Chip::getstate()
611 {
612     if (cycle >= maxcycle)
613         return HALT_CYCLE;
614     else if (!mips->running())
615         return HALT_MIPS;

```

Nov 05, 08 18:55

code.txt

Page 21/62

```

616     else if (recieve_int)
617         return HALT_INT;
618     else
619         return RUNNING;
620 }
621
622 /*****

```

Nov 05, 08 18:55

code.txt

Page 22/62

```

file name: memory.cc
1  /*****
2  * SimMips: Simple Computer Simulator of MIPS Arch Lab. TOKYO TECH *
3  *****/
4  #include "define.h"
5
6  enum {
7      MCO_READ = 0,
8      MCO_WRITE = 1,
9  };
10
11 /*****/
12 MainMemory::MainMemory(uint032_t mem_size)
13 {
14     npage = (mem_size + PAGE_SIZE - 1) / PAGE_SIZE;
15     mem_size = npage * PAGE_SIZE;
16     pagetable = new uint032_t*[npage];
17     external = new int[npage];
18     for (uint i = 0; i < npage; i++) {
19         pagetable[i] = NULL;
20         external[i] = 0;
21     }
22 }
23
24 /*****/
25 MainMemory::~MainMemory()
26 {
27     for (uint i = 0; i < npage; i++)
28         if (!external[i]) {
29             DELETE_ARRAY(pagetable[i]);
30         }
31     DELETE_ARRAY(pagetable);
32     DELETE_ARRAY(external);
33 }
34
35 /*****/
36 uint032_t* MainMemory::setpageentry(uint032_t addr, uint032_t *array)
37 {
38     if (addr >= mem_size)
39         return NULL;
40     if (!external[addr / PAGE_SIZE])
41         DELETE_ARRAY(pagetable[addr / PAGE_SIZE]);
42     pagetable[addr / PAGE_SIZE] = array;
43     external[addr / PAGE_SIZE] = 1;
44     return array;
45 }
46
47 /*****/
48 inline uint032_t* MainMemory::newpage(uint032_t addr)
49 {
50     uint032_t *page = new uint032_t[PAGE_SIZE / sizeof(uint032_t)];
51     for (uint i = 0; i < PAGE_SIZE / sizeof(uint032_t); i++)
52         page[i] = 0;
53     pagetable[addr / PAGE_SIZE] = page;
54     return page;
55 }
56
57 /*****/
58 inline uint032_t* MainMemory::getrealaddr(uint032_t addr)
59 {
60     uint032_t *page = pagetable[addr / PAGE_SIZE];
61     if (page == NULL)
62         page = newpage(addr);
63     page += (addr % PAGE_SIZE) / sizeof(uint032_t);
64     return page;
65 }
66
67 /*****/
68 void MainMemory::readlb(uint032_t addr, uint008_t *data)
69 {
70     uint032_t temp;
71     int offset = (addr & 0x3) * 8;
72     read4b(addr, &temp);
73     *data = (temp >> offset) & 0xff;
74 }
75
76 /*****/

```

Nov 05, 08 18:55

code.txt

Page 23/62

```

77 void MainMemory::read2b(uint032_t addr, uint016_t *data)
78 {
79     uint032_t temp;
80     int offset = (addr & 0x2) * 8;
81     read4b(addr, &temp);
82     *data = (temp >> offset) & 0xffff;
83 }
84
85 /*****/
86 void MainMemory::read4b(uint032_t addr, uint032_t *data)
87 {
88     uint032_t *mem = getrealaddr(addr);
89     *data = *mem;
90 }
91
92 /*****/
93 void MainMemory::read8b(uint032_t addr, uint064_t *data)
94 {
95     addr = addr & ~0x7;
96     uint032_t temp_h, temp_l;
97     read4b(addr, &temp_l);
98     read4b(addr + 4, &temp_h);
99     *data = ((uint064_t) temp_h << 32) || temp_l;
100 }
101
102 /*****/
103 void MainMemory::readnb(uint032_t addr, int size, uint008_t *data)
104 {
105     for (int i = 0; i < size; i++, addr++, data++)
106         read1b(addr, data);
107 }
108
109 /*****/
110 void MainMemory::write1b(uint032_t addr, uint008_t data)
111 {
112     int offset = (addr & 0x3) * 8;
113     uint032_t ins = (uint032_t) data << offset;
114     uint032_t mask = 0xff << offset;
115     uint032_t temp;
116     read4b(addr, &temp);
117     temp = ins | (temp & ~mask);
118     write4b(addr, temp);
119 }
120
121 /*****/
122 void MainMemory::write2b(uint032_t addr, uint016_t data)
123 {
124     int offset = (addr & 0x2) * 8;
125     uint032_t ins = (uint032_t) data << offset;
126     uint032_t mask = 0xffff << offset;
127     uint032_t temp;
128     read4b(addr, &temp);
129     temp = ins | (temp & ~mask);
130     write4b(addr, temp);
131 }
132
133 /*****/
134 void MainMemory::write4b(uint032_t addr, uint032_t data)
135 {
136     uint032_t *mem = getrealaddr(addr);
137     *mem = data;
138 }
139
140 /*****/
141 void MainMemory::write8b(uint032_t addr, uint064_t data)
142 {
143     addr = addr & ~0x7;
144     write4b(addr, (uint032_t) data);
145     write4b(addr + 4, (uint032_t) (data >> 32));
146 }
147
148 /*****/
149 void MainMemory::writenb(uint032_t addr, int size, uint008_t *data)
150 {
151     for (int i = 0; i < size; i++, addr++, data++)
152         write1b(addr, *data);
153 }

```

Nov 05, 08 18:55

code.txt

Page 24/62

```

154
155 /*****/
156 void MainMemory::print()
157 {
158     uint032_t *page;
159     for (uint i = 0; i < npage; i++) {
160         page = pagetable[i];
161         if (page != NULL) {
162             printf("[MEMORY BLOCK: 0x%08x]", i);
163             for (uint j = 0; j < PAGE_SIZE / sizeof(uint032_t); j++) {
164                 if ((j % 4) == 0)
165                     printf("\n%08x: ", (uint)
166                         (i * PAGE_SIZE + j * sizeof(uint032_t)));
167                 printf("%02x%02x%02x%02x ",
168                     page[j] & 0xff, (page[j] >> 8) & 0xff,
169                     (page[j] >> 16) & 0xff, (page[j] >> 24) & 0xff);
170             }
171             printf("\n\n");
172         }
173     }
174 }
175
176 /*****/
177 MemoryMap::MemoryMap()
178 {
179     addr = size = 0;
180     dev = NULL;
181     next = NULL;
182 }
183
184 /*****/
185 MemoryMap::~MemoryMap()
186 {
187     DELETE(dev);
188     DELETE(next);
189 }
190
191 /*****/
192 McInst::McInst()
193 {
194     state = MCI_NONE;
195     data008 = 0;
196     data016 = 0;
197     data032 = 0;
198     data064 = 0;
199 }
200
201 /*****/
202 MemoryController::MemoryController(MemoryMap *mmmap, int mode)
203 {
204     this->mmmap = mmmap;
205     this->mode = mode;
206     head = tail = 0;
207 }
208
209 /*****/
210 int MemoryController::enqueue(uint064_t addr, uint032_t size, void *data)
211 {
212     int ret = head;
213     McInst *ih = &inst[head];
214     if ((tail - head + NUM_MCINST) % NUM_MCINST == 1)
215         return -1;
216
217     ih->state = MCI_PENDING;
218     ih->op = (data) ? MCO_WRITE : MCO_READ;
219     ih->addr = addr;
220     ih->size = size;
221
222     if (data) {
223         if (size == 1)
224             ih->data008 = *(uint008_t *) data;
225         else if (size == 2)
226             ih->data016 = *(uint016_t *) data;
227         else if (size == 4)
228             ih->data032 = *(uint032_t *) data;
229         else
230             return -1;

```

Nov 05, 08 18:55

code.txt

Page 25/62

```

231     }
232
233     head = (head == NUM_MCINST - 1) ? 0 : head + 1;
234     if (mode == MC_THROUGHMODE)
235         step();
236     return ret;
237 }
238
239 /*****
240 void MemoryController::step()
241 {
242     McInst *it = &inst[tail];
243     MemoryMap *temp;
244
245     if (it->state != MCI_PEND)
246         return;
247
248     for (temp = mmap; temp != NULL; temp = temp->next) {
249         if (it->addr - temp->addr >= temp->size)
250             continue;
251         uint032_t addr = (uint032_t) it->addr - temp->addr;
252         if (it->op == MCO_READ) {
253             // Memory Read
254             if (it->size == 1)
255                 temp->dev->read1b(addr, &it->data008);
256             else if (it->size == 2)
257                 temp->dev->read2b(addr, &it->data016);
258             else if (it->size == 4)
259                 temp->dev->read4b(addr, &it->data032);
260             else if (it->size == 8)
261                 temp->dev->read8b(addr, &it->data064);
262             else
263                 it->state = MCI_FAILURE;
264         } else {
265             // Memory Write
266             if (it->size == 1)
267                 temp->dev->writelb(addr, it->data008);
268             else if (it->size == 2)
269                 temp->dev->write2b(addr, it->data016);
270             else if (it->size == 4)
271                 temp->dev->write4b(addr, it->data032);
272             else if (it->size == 8)
273                 temp->dev->write8b(addr, it->data064);
274             else
275                 it->state = MCI_FAILURE;
276         }
277         break;
278     }
279     if (temp == NULL) {
280         it->state = MCI_FAILURE;
281         // for debug
282         fprintf(stderr, "!! OUT OF MEMORY RANGE: 0x%08llx\n", it->addr);
283     }
284     if (it->state != MCI_FAILURE)
285         it->state = MCI_FINISH;
286
287     tail = (tail == NUM_MCINST - 1) ? 0 : tail + 1;
288 }
289
290 /*****
291 void MemoryController::print()
292 {
293     for (MemoryMap *temp = mmap; temp != NULL; temp = temp->next)
294         temp->dev->print();
295 }
296
297 /*****/

```

Nov 05, 08 18:55

code.txt

Page 26/62

```

file name: simloader.cc
1  /*****
2  * SimMips: Simple Computer Simulator of MIPS Arch Lab. TOKYO TECH *
3  *****/
4  #include "define.h"
5
6  enum {
7      FI_NONE = 0x0,
8      FI_ELF = 0x1,
9      FI_COFF = 0x2,
10     FI_BIT32 = 0x4,
11     FI_BIT64 = 0x8,
12     FI_LITTLE = 0x10,
13     FI_BIG = 0x20,
14     FI_ELF32LSB = 0x15,
15     FI_ELF32MSB = 0x25,
16     FI_ELF64LSB = 0x19,
17     FI_ELF64MSB = 0x29,
18 };
19
20 /*****
21 SimLoader::SimLoader()
22 {
23     memtab = NULL;
24     symtab = NULL;
25     fileident = FI_NONE;
26     filetype = 0;
27     endian = 0;
28     archtype = 0;
29     memtabnum = 0;
30     symtabnum = 0;
31     entry = 0;
32     stackptr = 0;
33     dynamic = 0;
34 }
35
36 SimLoader::~SimLoader()
37 {
38     if (memtab != NULL) {
39         delete [] memtab;
40         memtab = NULL;
41     }
42     if (symtab != NULL) {
43         for (int i = 0; i < symtabnum; i++)
44             delete [] symtab[i].name;
45         delete [] symtab;
46         symtab = NULL;
47     }
48 }
49
50 /*****
51 int SimLoader::checkfile(const char *file)
52 {
53     unsigned char e_ident[EI_NIDENT];
54     fileident = FI_NONE;
55
56     /* read magic */
57     memcpy(e_ident, file, EI_NIDENT);
58     if (e_ident[EI_MAG0] != ELFMAG0 || e_ident[EI_MAG1] != ELFMAG1 ||
59         e_ident[EI_MAG2] != ELFMAG2 || e_ident[EI_MAG3] != ELFMAG3) {
60         fprintf(stderr, "## ERROR: file is not ELF format.");
61         return(1);
62     }
63     fileident |= FI_ELF;
64
65     /* check class */
66     switch (e_ident[EI_CLASS]) {
67     case ELFCLASS32:
68         fileident |= FI_BIT32;
69         break;
70     case ELFCLASS64:
71         fileident |= FI_BIT64;
72         fprintf(stderr, "## ERROR: ELF64 is not supported.");
73         return(1);
74         break;
75     case ELFCLASSNONE:
76     default:

```

Nov 05, 08 18:55

code.txt

Page 27/62

```

77     fprintf(stderr, "## ERROR: ELFCLASS none.");
78     return(1);
79     break;
80 }
81
82 /* check endian */
83 switch (e_ident[EI_DATA]) {
84 case ELFDATA2LSB:
85     fileident |= FI_LITTLE;
86     break;
87 case ELFDATA2MSB:
88     fileident |= FI_BIG;
89     break;
90 case ELFDATANONE:
91 default:
92     fprintf(stderr, "## ERROR: ELFDATA none.");
93     return(1);
94     break;
95 }
96 return(0);
97 }
98
99 /*****
100 int SimLoader::loadelf32(const char *file)
101 {
102     Elf32_Ehdr *ehdr;
103     Elf32_Shdr *shdr, *shstr, *str = NULL, *sym = NULL;
104     Elf32_Sym *symp;
105
106     /* read ELF Header */
107     ehdr = (Elf32_Ehdr *)file;
108     filetype = ehdr->e_type;
109     archtype = ehdr->e_machine;
110     entry = ehdr->e_entry;
111
112     /* read Section Header */
113     memtabnum = 0;
114     shstr = (Elf32_Shdr *) (file + ehdr->e_shoff +
115                             ehdr->e_shentsize * ehdr->e_shstrndx);
116     for (int i = 0; i < ehdr->e_shnum; i++) {
117         shdr = (Elf32_Shdr *) (file + ehdr->e_shoff +
118                               ehdr->e_shentsize * i);
119         char *sname = (char *) (file + shstr->sh_offset + shdr->sh_name);
120         if (strcmp(sname, ".strtab") == 0)
121             str = shdr;
122         if (strcmp(sname, ".stack") == 0)
123             stackptr = shdr->sh_addr;
124         if (shdr->sh_type == SHT_SYMTAB)
125             sym = shdr;
126         if (shdr->sh_type == SHT_DYNAMIC)
127             dynamic = 1;
128         if (shdr->sh_flags & SHF_ALLOC) {
129             memtabnum += shdr->sh_size;
130         }
131     }
132
133     /* read Symbol table */
134     symtabnum = 0;
135     if (sym) {
136         for (unsigned int i = 0; i < sym->sh_size / sym->sh_entsize; i++) {
137             symp = (Elf32_Sym *) (file + sym->sh_offset +
138                                  sym->sh_entsize * i);
139             if (!symp->st_name)
140                 continue;
141             symtabnum++;
142         }
143     }
144
145     /* write to memtab */
146     memtab = new memtab_t[memtabnum];
147     int count = 0;
148     for (int i = 0; i < ehdr->e_shnum; i++) {
149         shdr = (Elf32_Shdr *) (file + ehdr->e_shoff +
150                               ehdr->e_shentsize * i);
151         if (shdr->sh_flags & SHF_ALLOC) {
152             for (unsigned int j = 0; j < shdr->sh_size; j++) {
153                 memtab[count].addr = shdr->sh_addr + j;

```

Nov 05, 08 18:55

code.txt

Page 28/62

```

154         memtab[count].data = (shdr->sh_type & SHT_NOBITS) ? 0 :
155             *(uint008_t *) (file + shdr->sh_offset + j);
156         count++;
157     }
158 }
159
160 /* write to symtab */
161 if (!sym || !str)
162     return(0);
163
164 symtab = new symtab_t[symtabnum];
165 count = 0;
166 for (unsigned int i = 0; i < sym->sh_size / sym->sh_entsize; i++) {
167     symp = (Elf32_Sym *) (file + sym->sh_offset +
168                           sym->sh_entsize * i);
169     if (!symp->st_name)
170         continue;
171     if (ELF32_ST_TYPE(symp->st_info) == STT_FUNC)
172         symtab[count].type = ST_FUNC;
173     else if (ELF32_ST_TYPE(symp->st_info) == STT_OBJECT)
174         symtab[count].type = ST_OBJECT;
175     else
176         symtab[count].type = ST_NOTYPE;
177     symtab[count].addr = symp->st_value;
178     char *symname = (char *) (file + str->sh_offset + symp->st_name);
179     int symlen = strlen(symname) + 1;
180     symtab[count].name = new char[symlen];
181     strncpy(symtab[count].name, symname, symlen);
182     count++;
183 }
184 return(0);
185 }
186
187 /*****
188 int SimLoader::loadfile(char *filename)
189 {
190     int fd = 0;
191     struct stat sb;
192     char *file;
193
194     /* open and read file */
195     fd = open(filename, O_RDONLY);
196     if (fd == -1) {
197         fprintf(stderr, "## ERROR: Can't open file. (%s)\n", filename);
198         return(1);
199     }
200     fstat(fd, &sb);
201     file = new char[sb.st_size];
202     if (read(fd, file, sb.st_size) < sb.st_size) {
203         fprintf(stderr, "## ERROR: Can't read file. (%s)\n", filename);
204         return(1);
205     }
206     close(fd);
207     if (checkfile(file))
208         return(1);
209
210     switch (fileident) {
211     case FI_ELF32LSB:
212         if (loadelf32(file))
213             return(1);
214         endian = FI_LITTLE;
215         break;
216     default:
217         fprintf(stderr, "## ERROR: non supported file type.\n");
218         return(1);
219         break;
220     }
221     delete [] file;
222     file = NULL;
223     return(0);
224 }
225
226 /*****

```

Nov 05, 08 18:55

code.txt

Page 29/62

```

file name: mips.cc
1  /*****
2  * SimMips: Simple Computer Simulator of MIPS   Arch Lab. TOKYO TECH *
3  *****/
4  #include "define.h"
5
6  #define SGN(x) ((x) & 0x80000000)
7  inline uint032_t exts32(uint032_t x, int y);
8
9  /*****/
10 Mips::Mips(Board *board, Chip *chip)
11 {
12     this->board = board;
13     this->chip = chip;
14     this->mc = chip->mc;
15     this->cp0 = chip->cp0;
16     as = new MipsArchstate();
17     ss = new MipsSimstate();
18     inst = new MipsInst();
19     state = CPU_STOP;
20     exc_occur = 0;
21     wait_cycle = 0;
22 }
23
24 /*****/
25 Mips::~Mips()
26 {
27     DELETE(as);
28     DELETE(inst);
29 }
30
31 /*****/
32 int Mips::step_func()
33 {
34     if (!running())
35         return (state != CPU_ERROR) ? 0 : -1;
36     if (wait_cycle) {
37         wait_cycle--;
38         return 0;
39     }
40
41     uint064_t inst_last = ss->inst_count;
42     if (cp0)
43         if (cp0->checkinterrupt())
44             exception(EXC_INT__);
45
46     if (state == CPU_WAIT)
47         return 0;
48
49     fetch();
50     decode();
51     regfetch();
52     execute();
53     if (inst->attr & LOADSTORE) {
54         memsend();
55         memreceive();
56     }
57     writeback();
58     setnpc();
59     return (state != CPU_ERROR) ?
60         (int) (ss->inst_count - inst_last) : -1;
61 }
62
63 /*****/
64 int Mips::step_multi()
65 {
66     if (!running())
67         return (state != CPU_ERROR) ? 0 : -1;
68     if (wait_cycle) {
69         wait_cycle--;
70         return 0;
71     }
72
73     uint064_t inst_last = ss->inst_count;
74     if (cp0)
75         if (cp0->checkinterrupt())
76             exception(EXC_INT__);

```

Nov 05, 08 18:55

code.txt

Page 30/62

```

77
78     if (state == CPU_IF) {
79         fetch();
80     } else if (state == CPU_ID) {
81         decode();
82     } else if (state == CPU_RF) {
83         regfetch();
84     } else if (state == CPU_EX) {
85         execute();
86     } else if (state == CPU_MS) {
87         memsend();
88     } else if (state == CPU_MR) {
89         memreceive();
90     } else if (state == CPU_WB) {
91         writeback();
92         setnpc();
93     }
94     proceedstate();
95     return (state != CPU_ERROR) ?
96         (int) (ss->inst_count - inst_last) : -1;
97 }
98
99 /*****/
100 inline void Mips::fetch()
101 {
102     if ((exc_occur) || (!running()))
103         return;
104
105     uint064_t addr = (uint064_t) as->pc;
106     int ret;
107
108     ss->inst_count++;
109     inst->pc = as->pc;
110     inst->clearmnemonic();
111     if (cp0) {
112         if ((ret = cp0->getphaddr(inst->pc, &addr, 0)) != 0) {
113             vaddr = inst->pc;
114             exception(ret);
115             return;
116         }
117     }
118     mcid = mc->enqueue(addr, 4, NULL);
119     if (mcid < 0) {
120         printf("## fetch failure 0x%08x\n", inst->pc);
121         state = CPU_ERROR;
122     }
123 }
124
125 /*****/
126 inline void Mips::decode()
127 {
128     if ((exc_occur) || (!running()))
129         return;
130
131     if (mc->inst[mcid].state == MCI_FAILURE) {
132         printf("## fetch failure 0x%08x\n", inst->pc);
133         state = CPU_ERROR;
134         return;
135     }
136     inst->ir = mc->inst[mcid].data032;
137     inst->decode();
138
139     if ((board->debug_mode == DEB_INST) ||
140         (board->debug_mode == DEB_REG))
141         printf("[%011d] %08x: %s\n",
142             ss->inst_count, inst->pc, inst->getmnemonic());
143
144     if (board->imix_mode)
145         ss->imix[inst->op]++;
146 }
147
148 /*****/
149 inline void Mips::regfetch()
150 {
151     if ((exc_occur) || (!running()))
152         return;
153 }

```

Nov 05, 08 18:55

code.txt

Page 31/62

```

154 // NOTE: semantic 'if's are omitted for speedup
155 // if (inst->attr & READ_RS)
156 //   rrs = as->r[inst->rs];
157 // if (inst->attr & READ_RT)
158 //   rrt = as->r[inst->rt];
159 // if (inst->attr & READ_RD)
160 //   rrd = as->r[inst->rd];
161 // if (inst->attr & READ_HI)
162 //   rhi = as->hi;
163 // if (inst->attr & READ_LO)
164 //   rlo = as->lo;
165
166 if (board->debug_mode == DEB_REG) {
167   printf(" ");
168   if ((inst->attr & READ_RS) && (inst->rs))
169     printf("%s>%08x ", regname[inst->rs], rrs);
170   if ((inst->attr & READ_RT) && (inst->rt))
171     printf("%s>%08x ", regname[inst->rt], rrt);
172   if ((inst->attr & READ_RD) && (inst->rd))
173     printf("%s>%08x ", regname[inst->rd], rrd);
174   if (inst->attr & READ_HI)
175     printf("$hi>%08x ", rhi);
176   if (inst->attr & READ_LO)
177     printf("$lo>%08x ", rlo);
178 }
179 }
180
181 /*****
182 inline void Mips::execute()
183 {
184   if ((exc_occur) || (!running()))
185     return;
186   int ret;
187
188   switch (inst->op) {
189   case NOP____:
190   case SSNOP____:
191   case WAIT____:
192     // do nothing
193     break;
194   case SLL____:
195     rrd = rrt << inst->shamt;
196     break;
197   case SRL____:
198     rrd = rrt >> inst->shamt;
199     break;
200   case SRA____:
201     rrd = exts32(rrt >> inst->shamt, 32 - inst->shamt);
202     break;
203   case SLLV____:
204     rrd = rrt << (rrs % 32);
205     break;
206   case SRLV____:
207     rrd = rrt >> (rrs % 32);
208     break;
209   case SRAV____:
210     rrd = exts32(rrt >> (rrs % 32), 32 - (rrs % 32));
211     break;
212   case JR____:
213   case JR_HB____:
214     npc = rrs;
215     cond = 1;
216     break;
217   case JALR____:
218   case JALR_HB____:
219     rrd = inst->pc + 8;
220     npc = rrs;
221     cond = 1;
222     break;
223   case MOVZ____:
224     rrd = rrs;
225     cond = (rrt == 0);
226     break;
227   case MOVN____:
228     rrd = rrs;
229     cond = (rrt != 0);
230     break;

```

Nov 05, 08 18:55

code.txt

Page 32/62

```

231 case SYSCALL____:
232   if (cp0)
233     exception(EXC_SYSCALL);
234   else
235     syscall();
236   break;
237 case BREAK____:
238   exception(EXC_BP____);
239   break;
240 case SYNC____:
241   // LD/ST barrier, nothing to do for functional simulator
242   break;
243 case MFHI____:
244   rrd = rhi;
245   break;
246 case MTHI____:
247   rhi = rrs;
248   break;
249 case MFLO____:
250   rrd = rlo;
251   break;
252 case MTLO____:
253   rlo = rrs;
254   break;
255 case MULT____:
256   {
257     int064_t temp = (int064_t) rrs * (int064_t) rrt;
258     rhi = (uint032_t) (temp >> 32);
259     rlo = (uint032_t) (temp & 0xffffffff);
260     break;
261   }
262 case MULTU____:
263   {
264     uint064_t temp = (uint064_t) rrs * (uint064_t) rrt;
265     rhi = (uint032_t) (temp >> 32);
266     rlo = (uint032_t) (temp & 0xffffffff);
267     break;
268   }
269 case DIV____:
270   rlo = (uint032_t) ((int032_t) rrs / (int032_t) rrt);
271   rhi = (uint032_t) ((int032_t) rrs % (int032_t) rrt);
272   break;
273 case DIVU____:
274   rlo = rrs / rrt;
275   rhi = rrs % rrt;
276   break;
277 case ADD____:
278   rrd = rrs + rrt;
279   if ((SGN(rrs) == SGN(rrt)) && (SGN(rrs) != SGN(rrd)))
280     exception(EXC_OV____);
281   break;
282 case ADDU____:
283   rrd = rrs + rrt;
284   break;
285 case SUB____:
286   rrd = rrs - rrt;
287   if ((SGN(rrs) != SGN(rrt)) && (SGN(rrs) != SGN(rrd)))
288     exception(EXC_OV____);
289   break;
290 case SUBU____:
291   rrd = rrs - rrt;
292   break;
293 case AND____:
294   rrd = rrs & rrt;
295   break;
296 case OR____:
297   rrd = rrs | rrt;
298   break;
299 case XOR____:
300   rrd = rrs ^ rrt;
301   break;
302 case NOR____:
303   rrd = ~(rrs | rrt);
304   break;
305 case SLT____:
306   rrd = ((int032_t) rrs < (int032_t) rrt) ? 1 : 0;
307   break;

```


Nov 05, 08 18:55

code.txt

Page 33/62

```

308 case SLTU____:
309     rrd = (rrs < rrt) ? 1 : 0;
310     break;
311 case TGE____:
312     if ((int032_t) rrs >= (int032_t) rrt)
313         exception(EXC_TRAP__);
314     break;
315 case TGEU____:
316     if (rrs >= rrt)
317         exception(EXC_TRAP__);
318     break;
319 case TLT____:
320     if ((int032_t) rrs < (int032_t) rrt)
321         exception(EXC_TRAP__);
322     break;
323 case TLTU____:
324     if (rrs < rrt)
325         exception(EXC_TRAP__);
326     break;
327 case TEQ____:
328     if (rrs == rrt)
329         exception(EXC_TRAP__);
330     break;
331 case TNE____:
332     if (rrs != rrt)
333         exception(EXC_TRAP__);
334     break;
335 case BLTZ____:
336 case BLTZL____:
337 case BLTZAL____:
338 case BLTZALL____:
339     npc = inst->pc + (exts32(inst->imm, 16) << 2) + 4;
340     cond = ((int032_t) rrs < 0);
341     break;
342 case BGEZ____:
343 case BGEZL____:
344 case BGEZAL____:
345 case BGEZALL____:
346     npc = inst->pc + (exts32(inst->imm, 16) << 2) + 4;
347     cond = ((int032_t) rrs >= 0);
348     break;
349 case TGEI____:
350     if ((int032_t) rrs >= (int032_t) exts32(inst->imm, 16))
351         exception(EXC_TRAP__);
352     break;
353 case TGEIU____:
354     if (rrs >= exts32(inst->imm, 16))
355         exception(EXC_TRAP__);
356     break;
357 case TLTU____:
358     if ((int032_t) rrs < (int032_t) exts32(inst->imm, 16))
359         exception(EXC_TRAP__);
360     break;
361 case TLTIU____:
362     if (rrs < exts32(inst->imm, 16))
363         exception(EXC_TRAP__);
364     break;
365 case TEQI____:
366     if (rrs == exts32(inst->imm, 16))
367         exception(EXC_TRAP__);
368     break;
369 case TNEI____:
370     if (rrs != exts32(inst->imm, 16))
371         exception(EXC_TRAP__);
372     break;
373 case J____:
374 case JAL____:
375     npc = (inst->pc & 0xf0000000) | (inst->addr << 2);
376     cond = 1;
377     break;
378 case BEQ____:
379 case BEQL____:
380     npc = inst->pc + (exts32(inst->imm, 16) << 2) + 4;
381     cond = (rrs == rrt);
382     break;
383 case BNE____:
384 case BNEL____:

```

Nov 05, 08 18:55

code.txt

Page 34/62

```

385     npc = inst->pc + (exts32(inst->imm, 16) << 2) + 4;
386     cond = (rrs != rrt);
387     break;
388 case BLEZ____:
389 case BLEZL____:
390     npc = inst->pc + (exts32(inst->imm, 16) << 2) + 4;
391     cond = ((int032_t) rrs <= 0);
392     break;
393 case BGTZ____:
394 case BGTZL____:
395     npc = inst->pc + (exts32(inst->imm, 16) << 2) + 4;
396     cond = ((int032_t) rrs > 0);
397     break;
398 case ADDI____:
399     rrt = rrs + exts32(inst->imm, 16);
400     if ((SGN(rrs) == SGN(exts32(inst->imm, 16))) &&
401         (SGN(rrt) != SGN(exts32(inst->imm, 16))))
402         exception(EXC_OV__);
403     break;
404 case ADDIU____:
405     rrt = rrs + exts32(inst->imm, 16);
406     break;
407 case SLTI____:
408     rrt = ((int032_t) rrs < (int032_t) exts32(inst->imm, 16)) ? 1 : 0;
409     break;
410 case SLTIU____:
411     rrt = (rrs < exts32(inst->imm, 16)) ? 1 : 0;
412     break;
413 case ANDI____:
414     rrt = rrs & inst->imm;
415     break;
416 case ORI____:
417     rrt = rrs | inst->imm;
418     break;
419 case XORI____:
420     rrt = rrs ^ inst->imm;
421     break;
422 case LUI____:
423     rrt = inst->imm << 16;
424     break;
425 case MFC0____:
426     if (cp0)
427         rrt = cp0->readreg(inst->rd + inst->sel * 32);
428     else
429         rrt = 0;
430     break;
431 case CFC0____:
432     // cfc0 is defined but not used(really?)
433     rrt = 0;
434     break;
435 case MTC0____:
436     if (cp0)
437         cp0->writereg(inst->rd + inst->sel * 32, rrt);
438     break;
439 case TLBR____:
440     if (cp0)
441         cp0->tlbread();
442     break;
443 case TLBWI____:
444     if (cp0)
445         cp0->tlbwrite(0);
446     break;
447 case TLBWR____:
448     if (cp0)
449         cp0->tlbwrite(1);
450     break;
451 case TLBP____:
452     if (cp0)
453         cp0->tlblookup();
454     break;
455 case ERET____:
456     if (cp0) {
457         npc = cp0->readreg(CP0_EPC__);
458         cond = 1;
459     } else {
460         cond = 0;
461     }

```

Nov 05, 08 18:55

code.txt

Page 35/62

```

462     break;
463 case MADD____:
464 {
465     int064_t temp = ((int064_t) rhi << 32) | rlo;
466     temp += (int064_t) rrs * (int064_t) rrt;
467     rhi = (uint032_t) (temp >> 32);
468     rlo = (uint032_t) (temp & 0xffffffff);
469     break;
470 }
471 case MADDU____:
472 {
473     int064_t temp = ((int064_t) rhi << 32) | rlo;
474     temp += (int064_t) rrs * (int064_t) rrt;
475     rhi = (uint032_t) (temp >> 32);
476     rlo = (uint032_t) (temp & 0xffffffff);
477     break;
478 }
479 case MUL____:
480 {
481     int064_t temp = (int064_t) rrs * (int064_t) rrt;
482     rrd = (uint032_t) (temp & 0xffffffff);
483     break;
484 }
485 case MSUB____:
486 {
487     int064_t temp = ((int064_t) rhi << 32) | rlo;
488     temp -= (int064_t) rrs * (int064_t) rrt;
489     rhi = (uint032_t) (temp >> 32);
490     rlo = (uint032_t) (temp & 0xffffffff);
491     break;
492 }
493 case MSUBU____:
494 {
495     int064_t temp = ((int064_t) rhi << 32) | rlo;
496     temp -= (int064_t) rrs * (int064_t) rrt;
497     rhi = (uint032_t) (temp >> 32);
498     rlo = (uint032_t) (temp & 0xffffffff);
499     break;
500 }
501 case CLZ____:
502     for (rrd = 0; rrd < 32; rrd++)
503         if (rrs & (0x80000000u >> rrd))
504             break;
505     break;
506 case CLO____:
507     for (rrd = 0; rrd < 32; rrd++)
508         if (!(rrs & (0x80000000u >> rrd)))
509             break;
510     break;
511 case LB____:
512 case LBU____:
513 case LWL____:
514 case LWR____:
515     vaddr = rrs + exts32(inst->imm, 16);
516     if (!cp0)
517         paddr = (uint064_t) vaddr;
518     else if ((ret = cp0->getphaddr(vaddr, &paddr, 0)) != 0)
519         exception(ret);
520     break;
521 case SB____:
522 case SWL____:
523 case SWR____:
524     vaddr = rrs + exts32(inst->imm, 16);
525     if (!cp0)
526         paddr = (uint064_t) vaddr;
527     else if ((ret = cp0->getphaddr(vaddr, &paddr, 1)) != 0)
528         exception(ret);
529     break;
530 case LH____:
531 case LHU____:
532     vaddr = rrs + exts32(inst->imm, 16);
533     if (!cp0)
534         paddr = (uint064_t) vaddr;
535     else if (vaddr & 0x1)
536         exception(EXC_ADEL__);
537     else if ((ret = cp0->getphaddr(vaddr, &paddr, 0)) != 0)
538         exception(ret);

```

Nov 05, 08 18:55

code.txt

Page 36/62

```

539     break;
540 case SH____:
541     vaddr = rrs + exts32(inst->imm, 16);
542     if (!cp0)
543         paddr = (uint064_t) vaddr;
544     else if (vaddr & 0x1)
545         exception(EXC_ADEL__);
546     else if ((ret = cp0->getphaddr(vaddr, &paddr, 1)) != 0)
547         exception(ret);
548     break;
549 case LW____:
550 case LL____:
551     vaddr = rrs + exts32(inst->imm, 16);
552     if (!cp0)
553         paddr = (uint064_t) vaddr;
554     else if (vaddr & 0x3)
555         exception(EXC_ADEL__);
556     else if ((ret = cp0->getphaddr(vaddr, &paddr, 0)) != 0)
557         exception(ret);
558     break;
559 case SW____:
560 case SC____:
561     vaddr = rrs + exts32(inst->imm, 16);
562     if (!cp0)
563         paddr = (uint064_t) vaddr;
564     else if (vaddr & 0x3)
565         exception(EXC_ADEL__);
566     else if ((ret = cp0->getphaddr(vaddr, &paddr, 1)) != 0)
567         exception(ret);
568     break;
569 case CACHE____:
570     // Nothing to do so far
571     break;
572 case PREF____:
573     // Prefetch instruction, not implemented
574     break;
575 case FLOAT_OPS:
576     // floating operation, trapped if cp0 is enabled
577     if (cp0)
578         exception(EXC_CPU__ | EXC_CPU1__);
579     else {
580         printf("## Floating Instruction ! %08x: %08x\n",
581             inst->pc, inst->ir);
582         state = CPU_ERROR;
583     }
584     break;
585 default:
586     printf("## Undefined Opcode ! %08x: %08x\n",
587         inst->pc, inst->ir);
588     state = CPU_ERROR;
589     break;
590 }
591
592 wait_cycle = inst->latency - 1;
593 }
594
595 /*****
596 inline void Mips::memsend()
597 {
598     if ((exc_occure) || (!running()))
599         return;
600
601     if (inst->attr & LOAD_1B) {
602         mcid = mc->enqueue(paddr, 1, NULL);
603     } else if (inst->attr & LOAD_2B) {
604         mcid = mc->enqueue(paddr, 2, NULL);
605     } else if (inst->attr & LOAD_4B_ALIGN) {
606         mcid = mc->enqueue(paddr, 4, NULL);
607     } else if (inst->attr & STORE_1B) {
608         uint008_t temp = (uint008_t) rrt;
609         mcid = mc->enqueue(paddr, 1, &temp);
610     } else if (inst->attr & STORE_2B) {
611         uint016_t temp = (uint016_t) rrt;
612         mcid = mc->enqueue(paddr, 2, &temp);
613     } else if (inst->attr & STORE_4B_ALIGN) {
614         mcid = mc->enqueue(paddr, 4, &rrt);
615     } else if (inst->attr & LOADSTORE_4B_UNALIGN) {

```

Nov 05, 08 18:55

code.txt

Page 37/62

```

616         mcid = mc->enqueue(paddr & ~0x3, 4, NULL);
617     }
618     if (mcid < 0)
619         exception(EXC_DBE____);
620 }
621
622 /*****
623 inline void Mips::memreceiue()
624 {
625     if ((exc_occure) || (!running()))
626         return;
627
628     if (mc->inst[mcid].state == MCI_FAILURE) {
629         exception(EXC_DBE____);
630         return;
631     }
632     McInst *im = &mc->inst[mcid];
633
634     if (inst->attr & LOAD_1B) {
635         rrt = ((inst->op == LBU____) ?
636             im->data008 : exts32(im->data008, 8));
637     } else if (inst->attr & LOAD_2B) {
638         rrt = ((inst->op == LHU____) ?
639             im->data016 : exts32(im->data016, 16));
640     } else if (inst->attr & LOAD_4B_ALIGN) {
641         rrt = im->data032;
642     } else if (inst->attr & LOAD_4B_UNALIGN) {
643         if (inst->op == LWR____) {
644             int shamt = (vaddr & 0x3) * 8;
645             uint032_t mask = 0xffffffff >> shamt;
646             rrt = ((im->data032 >> shamt) & mask) | (rrt & ~mask);
647         } else { // LWL____
648             int shamt = 24 - (vaddr & 0x3) * 8;
649             uint032_t mask = 0xffffffff << shamt;
650             rrt = ((im->data032 << shamt) & mask) | (rrt & ~mask);
651         }
652     } else if (inst->attr & STORE_4B_UNALIGN) {
653         // TODO: failure in store cannot be trapped
654         if (inst->op == SWR____) {
655             int shamt = (vaddr & 0x3) * 8;
656             uint032_t mask = 0xffffffff << shamt;
657             uint032_t temp = (((rrt << shamt) & mask) |
658                             (im->data032 & ~mask));
659             mc->enqueue(paddr & ~0x3, 4, &temp);
660         } else { // SWL____
661             int shamt = 24 - (vaddr & 0x3) * 8;
662             uint032_t mask = 0xffffffff >> shamt;
663             uint032_t temp = (((rrt >> shamt) & mask) |
664                             (im->data032 & ~mask));
665             mc->enqueue(paddr & ~0x3, 4, &temp);
666         }
667     } else if (inst->op == SC____) {
668         rrt = 1;
669     }
670 }
671
672 /*****
673 inline void Mips::writeback()
674 {
675     if ((exc_occure) || (!running()))
676         return;
677     if (inst->attr & WRITE_RS)
678         as->r[inst->rs] = rrs;
679     if (inst->attr & WRITE_RT)
680         as->r[inst->rt] = rrt;
681     if (inst->attr & WRITE_RD)
682         as->r[inst->rd] = rrd;
683     if (inst->attr & WRITE_HI)
684         as->hi = rhi;
685     if (inst->attr & WRITE_LO)
686         as->lo = rlo;
687     if ((inst->attr & WRITE_RD_COND) && (cond))
688         as->r[inst->rd] = rrd;
689     if (inst->attr & WRITE_RRA)
690         as->r[REG_RA] = inst->pc + 8;
691
692     if (board->debug_mode == DEB_REG) {

```

Nov 05, 08 18:55

code.txt

Page 38/62

```

693     if ((inst->attr & WRITE_RS) && (inst->rs))
694         printf("%s<%08x ", regname[inst->rs], rrs);
695     if ((inst->attr & WRITE_RT) && (inst->rt))
696         printf("%s<%08x ", regname[inst->rt], rrt);
697     if ((inst->attr & WRITE_RD) && (inst->rd))
698         printf("%s<%08x ", regname[inst->rd], rrd);
699     if (inst->attr & WRITE_HI)
700         printf("%hi<%08x ", rhi);
701     if (inst->attr & WRITE_LO)
702         printf("%lo<%08x ", rlo);
703     if ((inst->attr & WRITE_RD_COND) && (inst->rd) && (cond))
704         printf("%s<%08x ", regname[inst->rd], rrd);
705     if (inst->attr & WRITE_RRA)
706         printf("$ra<%08x ", inst->pc + 8);
707     printf("\n");
708 }
709
710     as->r[0] = 0; // yes, register $zero is always zero
711 }
712
713 /*****
714 inline void Mips::setnpc()
715 {
716     if (!running())
717         return;
718
719     if (exc_occure) {
720         as->pc = cp0->doexception(exc_code, as->pc,
721                                 vaddr, (as->delay_npc) ? 1 : 0);
722         as->delay_npc = 0;
723     } else if (as->delay_npc) {
724         as->pc = as->delay_npc;
725         as->delay_npc = 0;
726     } else if ((inst->attr & BRANCH) || (inst->attr & BRANCH_LIKELY)
727               && (cond)) {
728         as->pc += 4;
729         as->delay_npc = npc;
730         if (!npc) {
731             printf("## Branch to zero. stop.\n");
732             state = CPU_ERROR;
733         }
734     } else if ((inst->attr & BRANCH_ERET) && (cond)) {
735         cp0->modifyreg(CP0_SR____, 0, 0x2);
736         as->pc = npc;
737         if (!npc) {
738             printf("## Branch to zero. stop.\n");
739             state = CPU_ERROR;
740         }
741     } else if ((inst->attr & BRANCH_LIKELY) && (!cond)) {
742         as->pc += 8;
743     } else {
744         as->pc += 4;
745     }
746
747     if ((state != CPU_ERROR) && (inst->op == WAIT____) && (!exc_occure))
748         state = CPU_WAIT;
749
750     exc_occure = 0;
751 }
752
753 /*****
754 void Mips::exception(int code)
755 {
756     if (!cp0)
757         return;
758     if (exc_occure)
759         return;
760
761     exc_occure = 1;
762     exc_code = code;
763     if (state == CPU_WAIT)
764         state = CPU_WB;
765     if (board->debug_mode == DEB_EXCTLB)
766         printf("## exception %#d (PC=0x%08x vaddr=0x%08x)\n",
767             code, as->pc, vaddr);
768 }
769

```

Nov 05, 08 18:55

code.txt

Page 39/62

```

770 /*****/
771 void Mips::syscall()
772 {
773     switch (as->r[REG_V0]) {
774     case SYS_EXIT:
775         state = CPU_STOP;
776         break;
777     case SYS_WRITE:
778         if (as->r[REG_A0] == STDOUT_FILENO) {
779             for (uint i = 0; i < as->r[REG_A2]; i++) {
780                 int mcid = mc->enqueue(as->r[REG_A1] + i, 1, NULL);
781                 if (mcid < 0)
782                     break;
783                 mc->step();
784                 if (mc->inst[mcid].state == MCI_FINISH)
785                     putchar((int) mc->inst[mcid].data008);
786             }
787             as->r[REG_V0] = as->r[REG_A2];
788             as->r[REG_A3] = 0;
789             break;
790         case SYS_IOCTL:
791             as->r[REG_V0] = as->r[REG_A3] = 0;
792             break;
793         default:
794             printf("## unknown syscall #%d (see asm/unistd.h). Skip.\n",
795                 as->r[REG_V0]);
796             as->r[REG_V0] = as->r[REG_A3] = 0;
797             break;
798     }
799 }
800 }
801 /*****/
802 void Mips::proceedstate()
803 {
804     if (!running())
805         return;
806     else if (state == CPU_EX)
807         state = (inst->attr & LOADSTORE) ? CPU_MS : CPU_WB;
808     else if (state == CPU_WB)
809         state = CPU_IF;
810     else if (state != CPU_WAIT)
811         state++;
812 }
813 }
814 /*****/
815 int Mips::running()
816 {
817     return (state > 0);
818 }
819 }
820 /*****/
821 MipsArchstate::MipsArchstate()
822 {
823     for (int i = 0; i < NREG; i++)
824         r[i] = 0;
825     hi = lo = 0;
826     pc = delay_npc = 0;
827 }
828 }
829 /*****/
830 void MipsArchstate::print()
831 {
832     for (int i = 0; i < NREG / 8; i++) {
833         for (int j = 0; j < 8; j++)
834             printf("%s", regname[i * 8 + j]);
835         printf("\n");
836         for (int j = 0; j < 8; j++)
837             printf("%08x ", r[i * 8 + j]);
838         printf("\n");
839     }
840     printf("pc      hi      lo\n");
841     printf("%08x %08x %08x\n", pc, hi, lo);
842 }
843 }
844 /*****/
845 MipsSimstate::MipsSimstate()

```

Nov 05, 08 18:55

code.txt

Page 40/62

```

847 {
848     inst_count = 0;
849     for (int i = 0; i < INST_CODE_NUM; i++)
850         imix[i] = 0;
851 }
852 /*****/
853 void MipsSimstate::print()
854 {
855     ullint temp[INST_CODE_NUM];
856     ullint max, lastmax;
857     MipsInst *inst = new MipsInst();
858     int num = 0, index = 0;
859     memcpy(temp, imix, sizeof(imix));
860     printf("[[Instruction Statistics]]\n");
861     lastmax = 0;
862     for (int i = 0; i < INST_CODE_NUM; i++) {
863         max = 0;
864         for (int j = 0; j < INST_CODE_NUM; j++)
865             if (max < temp[j]) {
866                 max = temp[j];
867                 index = j;
868             }
869         if (max == 0)
870             break;
871         temp[index] = 0;
872         if (max != lastmax) {
873             lastmax = max;
874             num = i + 1;
875         }
876         inst->op = index;
877         printf("[%3d] %-9s%11lld (%7.3f%%)\n",
878             num, inst->getinstname(), max,
879             (double) max / inst_count * 100.0);
880     }
881     printf("[---] Total      %11lld\n", inst_count);
882 }
883 /*****/
884 inline uint032_t exts32(uint032_t x, int y)
885 {
886     if (y == 32)
887         return x;
888     uint032_t temp = 0xffffffff << y;
889     if (x & (1 << (y - 1)))
890         return (temp | (x & ~temp));
891     else
892         return (x & ~temp);
893 }
894 /*****/

```

Nov 05, 08 18:55

code.txt

Page 41/62

```

file name: mipsinst.cc
1  /*****
2  * SimMips: Simple Computer Simulator of MIPS   Arch Lab. TOKYO TECH *
3  *****/
4  #include "define.h"
5
6  inline uint032_t exts32(uint032_t x, int y);
7
8  /*****/
9  MipsInst::MipsInst()
10 {
11     ir = 0;
12     op = UNDEFINED;
13     clearmnemonic();
14 }
15
16 /*****/
17 void MipsInst::decode()
18 {
19     opcode = (ir >> 26) & 0x3f;
20     rs      = (ir >> 21) & 0x1f;
21     rt      = (ir >> 16) & 0x1f;
22     rd      = (ir >> 11) & 0x1f;
23     shamt   = (ir >> 6) & 0x1f;
24     funct   = ir & 0x3f;
25     imm     = ir & 0xffff;
26     addr    = ir & 0x3fffffff;
27     code_l  = (ir >> 6) & 0xffff;
28     code_s  = (ir >> 16) & 0x3ff;
29     sel     = ir & 0x7;
30
31     op = UNDEFINED;
32     attr = READ_NONE | WRITE_NONE;
33     latency = 1;
34
35     switch (opcode) {
36     case 0:
37         switch (funct) {
38         case 0:
39             if ((rt | rd | shamt) == 0) {
40                 op = NOP;
41             } else if (((rt | rd) == 0) && (shamt == 1)) {
42                 op = SSNOP;
43             } else {
44                 op = SLL;
45                 attr = READ_RT | WRITE_RD;
46             }
47             break;
48         case 2:
49             op = SRL;
50             attr = READ_RT | WRITE_RD;
51             break;
52         case 3:
53             op = SRA;
54             attr = READ_RT | WRITE_RD;
55             break;
56         case 4:
57             op = SLLV;
58             attr = READ_RS | READ_RT | WRITE_RD;
59             break;
60         case 6:
61             op = SRLV;
62             attr = READ_RS | READ_RT | WRITE_RD;
63             break;
64         case 7:
65             op = SRAV;
66             attr = READ_RS | READ_RT | WRITE_RD;
67             break;
68         case 8:
69             if (shamt == 0) {
70                 op = JR;
71                 attr = BRANCH | READ_RS;
72             } else if (shamt == 16) {
73                 op = JR_HB;
74                 attr = BRANCH | READ_RS;
75             }
76             break;

```

Nov 05, 08 18:55

code.txt

Page 42/62

```

77     case 9:
78         if (shamt == 0) {
79             op = JALR;
80             attr = BRANCH | READ_RS | WRITE_RD;
81         } else if (shamt == 16) {
82             op = JALR_HB;
83             attr = BRANCH | READ_RS | WRITE_RD;
84         }
85         break;
86     case 10:
87         op = MOVZ;
88         attr = READ_RS | READ_RT | WRITE_RD_COND;
89         break;
90     case 11:
91         op = MOVN;
92         attr = READ_RS | READ_RT | WRITE_RD_COND;
93         break;
94     case 12:
95         op = SYSCALL;
96         break;
97     case 13:
98         op = BREAK;
99         break;
100    case 15:
101        op = SYNC;
102        break;
103    case 16:
104        op = MFHI;
105        attr = READ_HI | WRITE_RD;
106        break;
107    case 17:
108        op = MTHI;
109        attr = READ_RS | WRITE_HI;
110        break;
111    case 18:
112        op = MFLO;
113        attr = READ_LO | WRITE_RD;
114        break;
115    case 19:
116        op = MTLO;
117        attr = READ_RS | WRITE_LO;
118        break;
119    case 24:
120        op = MULT;
121        attr = READ_RS | READ_RT | WRITE_HILO;
122        break;
123    case 25:
124        op = MULTU;
125        attr = READ_RS | READ_RT | WRITE_HILO;
126        break;
127    case 26:
128        op = DIV;
129        attr = READ_RS | READ_RT | WRITE_HILO;
130        break;
131    case 27:
132        op = DIVU;
133        attr = READ_RS | READ_RT | WRITE_HILO;
134        break;
135    case 32:
136        op = ADD;
137        attr = READ_RS | READ_RT | WRITE_RD;
138        break;
139    case 33:
140        op = ADDU;
141        attr = READ_RS | READ_RT | WRITE_RD;
142        break;
143    case 34:
144        op = SUB;
145        attr = READ_RS | READ_RT | WRITE_RD;
146        break;
147    case 35:
148        op = SUBU;
149        attr = READ_RS | READ_RT | WRITE_RD;
150        break;
151    case 36:
152        op = AND;
153        attr = READ_RS | READ_RT | WRITE_RD;

```

Nov 05, 08 18:55

code.txt

Page 43/62

```

154         break;
155     case 37:
156         op = OR_____;
157         attr = READ_RS | READ_RT | WRITE_RD;
158         break;
159     case 38:
160         op = XOR_____;
161         attr = READ_RS | READ_RT | WRITE_RD;
162         break;
163     case 39:
164         op = NOR_____;
165         attr = READ_RS | READ_RT | WRITE_RD;
166         break;
167     case 42:
168         op = SLT_____;
169         attr = READ_RS | READ_RT | WRITE_RD;
170         break;
171     case 43:
172         op = SLTU_____;
173         attr = READ_RS | READ_RT | WRITE_RD;
174         break;
175     case 48:
176         op = TGE_____;
177         attr = READ_RS | READ_RT;
178         break;
179     case 49:
180         op = TGEU_____;
181         attr = READ_RS | READ_RT;
182         break;
183     case 50:
184         op = TLT_____;
185         attr = READ_RS | READ_RT;
186         break;
187     case 51:
188         op = TLTU_____;
189         attr = READ_RS | READ_RT;
190         break;
191     case 52:
192         op = TEQ_____;
193         attr = READ_RS | READ_RT;
194         break;
195     case 54:
196         op = TNE_____;
197         attr = READ_RS | READ_RT;
198         break;
199     }
200     break;
201 case 1:
202     switch (rt) {
203     case 0:
204         op = BLTZ_____;
205         attr = BRANCH | READ_RS;
206         break;
207     case 1:
208         op = BGEZ_____;
209         attr = BRANCH | READ_RS;
210         break;
211     case 2:
212         op = BLTZL_____;
213         attr = BRANCH_LIKELY | READ_RS;
214         break;
215     case 3:
216         op = BGEZL_____;
217         attr = BRANCH_LIKELY | READ_RS;
218         break;
219     case 8:
220         op = TGEI_____;
221         attr = READ_RS;
222         break;
223     case 9:
224         op = TGEIU_____;
225         attr = READ_RS;
226         break;
227     case 10:
228         op = TLTU_____;
229         attr = READ_RS;
230         break;

```

Nov 05, 08 18:55

code.txt

Page 44/62

```

231     case 11:
232         op = TLTU_____;
233         attr = READ_RS;
234         break;
235     case 12:
236         op = TEQI_____;
237         attr = READ_RS;
238         break;
239     case 14:
240         op = TNEI_____;
241         attr = READ_RS;
242         break;
243     case 16:
244         op = BLTZAL_____;
245         attr = BRANCH | READ_RS | WRITE_RRA;
246         break;
247     case 17:
248         op = BGEZAL_____;
249         attr = BRANCH | READ_RS | WRITE_RRA;
250         break;
251     case 18:
252         op = BLTZALL_____;
253         attr = BRANCH_LIKELY | READ_RS | WRITE_RRA;
254         break;
255     case 19:
256         op = BGEZALL_____;
257         attr = BRANCH_LIKELY | READ_RS | WRITE_RRA;
258         break;
259     }
260     break;
261 case 2:
262     op = J_____;
263     attr = BRANCH;
264     break;
265 case 3:
266     op = JAL_____;
267     attr = BRANCH | WRITE_RRA;
268     break;
269 case 4:
270     op = BEQ_____;
271     attr = BRANCH | READ_RS | READ_RT;
272     break;
273 case 5:
274     op = BNE_____;
275     attr = BRANCH | READ_RS | READ_RT;
276     break;
277 case 6:
278     op = BLEZ_____;
279     attr = BRANCH | READ_RS;
280     break;
281 case 7:
282     op = BGTZ_____;
283     attr = BRANCH | READ_RS;
284     break;
285 case 8:
286     op = ADDI_____;
287     attr = READ_RS | WRITE_RT;
288     break;
289 case 9:
290     op = ADDIU_____;
291     attr = READ_RS | WRITE_RT;
292     break;
293 case 10:
294     op = SLTI_____;
295     attr = READ_RS | WRITE_RT;
296     break;
297 case 11:
298     op = SLTIU_____;
299     attr = READ_RS | WRITE_RT;
300     break;
301 case 12:
302     op = ANDI_____;
303     attr = READ_RS | WRITE_RT;
304     break;
305 case 13:
306     op = ORI_____;
307     attr = READ_RS | WRITE_RT;

```

Nov 05, 08 18:55

code.txt

Page 45/62

```

308     break;
309     case 14:
310         op = XORI____;
311         attr = READ_RS | WRITE_RT;
312         break;
313     case 15:
314         op = LUI____;
315         attr = WRITE_RT;
316         break;
317     case 16:
318         switch (rs) {
319             case 0:
320                 op = MFC0____;
321                 attr = READ_NONE | WRITE_RT;
322                 break;
323             case 2:
324                 op = CFC0____;
325                 attr = READ_NONE | WRITE_RT;
326                 break;
327             case 4:
328                 op = MTC0____;
329                 attr = READ_RT | WRITE_NONE;
330                 break;
331             case 16:
332                 if (funct == 1)
333                     op = TLBR____;
334                 else if (funct == 2)
335                     op = TLBWI____;
336                 else if (funct == 6)
337                     op = TLBWR____;
338                 else if (funct == 8)
339                     op = TLBP____;
340                 else if (funct == 24) {
341                     op = ERET____;
342                     attr = BRANCH_ERET;
343                 } else if (funct == 32) {
344                     op = WAIT____;
345                 }
346                 break;
347             }
348         break;
349     case 17:
350         op = FLOAT_OPS;
351         break;
352     case 20:
353         op = BEQL____;
354         attr = BRANCH_LIKELY | READ_RS | READ_RT;
355         break;
356     case 21:
357         op = BNEL____;
358         attr = BRANCH_LIKELY | READ_RS | READ_RT;
359         break;
360     case 22:
361         op = BLEZL____;
362         attr = BRANCH_LIKELY | READ_RS;
363         break;
364     case 23:
365         op = BGTZL____;
366         attr = BRANCH_LIKELY | READ_RS;
367         break;
368     case 28:
369         switch (funct) {
370             case 0:
371                 op = MADD____;
372                 attr = READ_RS | READ_RT | READ_HILO | WRITE_HILO;
373                 break;
374             case 1:
375                 op = MADDDU____;
376                 attr = READ_RS | READ_RT | READ_HILO | WRITE_HILO;
377                 break;
378             case 2:
379                 op = MUL____;
380                 attr = READ_RS | READ_RT | WRITE_RD;
381                 break;
382             case 4:
383                 op = MSUB____;
384                 attr = READ_RS | READ_RT | READ_HILO | WRITE_HILO;

```

Nov 05, 08 18:55

code.txt

Page 46/62

```

385     break;
386     case 5:
387         op = MSUBU____;
388         attr = READ_RS | READ_RT | READ_HILO | WRITE_HILO;
389         break;
390     case 32:
391         op = CLZ____;
392         attr = READ_RS | WRITE_RD;
393         break;
394     case 33:
395         op = CLO____;
396         attr = READ_RS | WRITE_RD;
397         break;
398     }
399     break;
400     case 32:
401         op = LB____;
402         attr = READ_RS | WRITE_RT | LOAD_1B;
403         break;
404     case 33:
405         op = LH____;
406         attr = READ_RS | WRITE_RT | LOAD_2B;
407         break;
408     case 34:
409         op = LWL____;
410         attr = READ_RS | READ_RT | WRITE_RT | LOAD_4B_UNALIGN;
411         break;
412     case 35:
413         op = LW____;
414         attr = READ_RS | WRITE_RT | LOAD_4B_ALIGN;
415         break;
416     case 36:
417         op = LBU____;
418         attr = READ_RS | WRITE_RT | LOAD_1B;
419         break;
420     case 37:
421         op = LHU____;
422         attr = READ_RS | WRITE_RT | LOAD_2B;
423         break;
424     case 38:
425         op = LWR____;
426         attr = READ_RS | READ_RT | WRITE_RT | LOAD_4B_UNALIGN;
427         break;
428     case 40:
429         op = SB____;
430         attr = READ_RS | READ_RT | STORE_1B;
431         break;
432     case 41:
433         op = SH____;
434         attr = READ_RS | READ_RT | STORE_2B;
435         break;
436     case 42:
437         op = SWL____;
438         attr = READ_RS | READ_RT | STORE_4B_UNALIGN;
439         break;
440     case 43:
441         op = SW____;
442         attr = READ_RS | READ_RT | STORE_4B_ALIGN;
443         break;
444     case 46:
445         op = SWR____;
446         attr = READ_RS | READ_RT | STORE_4B_UNALIGN;
447         break;
448     case 47:
449         op = CACHE____;
450         attr = READ_RS;
451         break;
452     case 48:
453         op = LL____;
454         attr = READ_RS | WRITE_RT | LOAD_4B_ALIGN;
455         break;
456     case 51:
457         op = PREF____;
458         attr = READ_RS;
459         break;
460     case 56:
461         op = SC____;

```

Nov 05, 08 18:55

code.txt

Page 47/62

```

462     attr = READ_RS | READ_RT | WRITE_RT | STORE_4B_ALIGN;
463     break;
464     case 49:
465     case 53:
466     case 57:
467     case 61:
468         op = FLOAT_OPS;
469         break;
470     }
471 }
472 /*****
473 void MipsInst::clearmnemonic()
474 {
475     mnemonic[0] = '\0';
476     instname = NULL;
477 }
478
479 /*****
480 char *MipsInst::getinstname()
481 {
482     if (op == NOP_____) instname = "nop";
483     else if (op == SSNOP_____) instname = "ssnop";
484     else if (op == SLL_____) instname = "sll";
485     else if (op == SRL_____) instname = "srl";
486     else if (op == SRA_____) instname = "sra";
487     else if (op == SLLV_____) instname = "sllv";
488     else if (op == SRLV_____) instname = "srlv";
489     else if (op == SRAV_____) instname = "srav";
490     else if (op == JR_____) instname = "jr";
491     else if (op == JR_HB_____) instname = "jr";
492     else if (op == JALR_____) instname = "jalr";
493     else if (op == JALR_HB_____) instname = "jalr";
494     else if (op == MOVZ_____) instname = "movz";
495     else if (op == MOVN_____) instname = "movn";
496     else if (op == SYSCALL_____) instname = "syscall";
497     else if (op == BREAK_____) instname = "break";
498     else if (op == SYNC_____) instname = "sync";
499     else if (op == MFHI_____) instname = "mfhi";
500     else if (op == MTHI_____) instname = "mthi";
501     else if (op == MFLO_____) instname = "mflo";
502     else if (op == MTLO_____) instname = "mtlo";
503     else if (op == MULT_____) instname = "mult";
504     else if (op == MULTU_____) instname = "multu";
505     else if (op == DIV_____) instname = "div";
506     else if (op == DIVU_____) instname = "divu";
507     else if (op == ADD_____) instname = "add";
508     else if (op == ADDU_____) instname = "addu";
509     else if (op == SUB_____) instname = "sub";
510     else if (op == SUBU_____) instname = "subu";
511     else if (op == AND_____) instname = "and";
512     else if (op == OR_____) instname = "or";
513     else if (op == XOR_____) instname = "xor";
514     else if (op == NOR_____) instname = "nor";
515     else if (op == SLT_____) instname = "slt";
516     else if (op == SLTU_____) instname = "sltu";
517     else if (op == TGE_____) instname = "tge";
518     else if (op == TGEU_____) instname = "tgeu";
519     else if (op == TLT_____) instname = "tlt";
520     else if (op == TLTU_____) instname = "tltu";
521     else if (op == TEQ_____) instname = "teq";
522     else if (op == TNE_____) instname = "tne";
523     else if (op == BLTZ_____) instname = "bltz";
524     else if (op == BGEZ_____) instname = "bgez";
525     else if (op == BLTZL_____) instname = "bltzl";
526     else if (op == BGEZL_____) instname = "bgezl";
527     else if (op == TGEI_____) instname = "tgei";
528     else if (op == TGEIU_____) instname = "tgeiu";
529     else if (op == TLTi_____) instname = "tlti";
530     else if (op == TLTiU_____) instname = "tltiu";
531     else if (op == TEQi_____) instname = "teqi";
532     else if (op == TNEi_____) instname = "tnei";
533     else if (op == BLTZAL_____) instname = "bltzal";
534     else if (op == BGEZAL_____) instname = "bgezal";
535     else if (op == BLTZALL_____) instname = "bltzall";
536     else if (op == BGEZALL_____) instname = "bgezall";
537     else if (op == J_____) instname = "j";
538     else if (op == JAL_____) instname = "jal";

```

Nov 05, 08 18:55

code.txt

Page 48/62

```

539     else if (op == BEQ_____) instname = "beq";
540     else if (op == BNE_____) instname = "bne";
541     else if (op == BLEZ_____) instname = "blez";
542     else if (op == BGTZ_____) instname = "bgtz";
543     else if (op == ADDI_____) instname = "addi";
544     else if (op == ADDIU_____) instname = "addiu";
545     else if (op == SLTI_____) instname = "slti";
546     else if (op == SLTIU_____) instname = "sltiu";
547     else if (op == ANDI_____) instname = "andi";
548     else if (op == ORI_____) instname = "ori";
549     else if (op == XORI_____) instname = "xori";
550     else if (op == LUI_____) instname = "lui";
551     else if (op == MFC0_____) instname = "mfc0";
552     else if (op == CFC0_____) instname = "cfc0";
553     else if (op == MTC0_____) instname = "mtc0";
554     else if (op == TLBR_____) instname = "tlbr";
555     else if (op == TLBWI_____) instname = "tlbwi";
556     else if (op == TLBWR_____) instname = "tlbwr";
557     else if (op == TLBP_____) instname = "tlbp";
558     else if (op == ERET_____) instname = "eret";
559     else if (op == WAIT_____) instname = "wait";
560     else if (op == BEQL_____) instname = "beql";
561     else if (op == BNEL_____) instname = "bnel";
562     else if (op == BLEZL_____) instname = "blezl";
563     else if (op == BGTZL_____) instname = "bgtzl";
564     else if (op == MADD_____) instname = "madd";
565     else if (op == MADDU_____) instname = "maddu";
566     else if (op == MUL_____) instname = "mul";
567     else if (op == MSUB_____) instname = "msub";
568     else if (op == MSUBU_____) instname = "msubu";
569     else if (op == CLZ_____) instname = "clz";
570     else if (op == CLO_____) instname = "clo";
571     else if (op == LB_____) instname = "lb";
572     else if (op == LH_____) instname = "lh";
573     else if (op == LWL_____) instname = "lwl";
574     else if (op == LW_____) instname = "lw";
575     else if (op == LBU_____) instname = "lbu";
576     else if (op == LHU_____) instname = "lhu";
577     else if (op == LWR_____) instname = "lwr";
578     else if (op == SB_____) instname = "sb";
579     else if (op == SH_____) instname = "sh";
580     else if (op == SWL_____) instname = "swl";
581     else if (op == SW_____) instname = "sw";
582     else if (op == SWR_____) instname = "swr";
583     else if (op == CACHE_____) instname = "cache";
584     else if (op == LL_____) instname = "ll";
585     else if (op == PREF_____) instname = "pref";
586     else if (op == SC_____) instname = "sc";
587     else if (op == FLOAT_OPS_____) instname = "(FP inst)";
588     else
589         instname = "";
590
591     return instname;
592 }
593
594 /*****
595 char *MipsInst::getmnemonic()
596 {
597     if (mnemonic[0])
598         return mnemonic;
599
600     char *arg;
601     arg = new char[MNEMONIC_BUF_SIZE];
602     arg[0] = '\0';
603     int size = MNEMONIC_BUF_SIZE;
604
605     getinstname();
606
607     switch (op) {
608     case SLL_____:
609     case SRL_____:
610     case SRA_____:
611         snprintf(arg, size, "%s, %s, %d",
612                 regname[rd], regname[rt], shamt);
613         break;
614     case MFHI_____:
615     case MFLO_____:
616         snprintf(arg, size, "%s", regname[rd]);

```


Nov 05, 08 18:55

code.txt

Page 49/62

```

616         break;
617     case JR_____ :
618     case JR_HB_____ :
619     case MTHI_____ :
620     case MTLO_____ :
621         snprintf(arg, size, "%s", regname[rs]);
622         break;
623     case MULT_____ :
624     case MULTU_____ :
625     case DIV_____ :
626     case DIVU_____ :
627     case TGE_____ :
628     case TGEU_____ :
629     case TLT_____ :
630     case TLTU_____ :
631     case TEQ_____ :
632     case TNE_____ :
633     case MADD_____ :
634     case MADDU_____ :
635     case MSUB_____ :
636     case MSUBU_____ :
637         snprintf(arg, size, "%s, %s", regname[rs], regname[rt]);
638         break;
639     case JALR_____ :
640     case JALR_HB_____ :
641     case CLZ_____ :
642     case CLO_____ :
643         snprintf(arg, size, "%s, %s", regname[rd], regname[rs]);
644         break;
645     case SLLV_____ :
646     case SRLV_____ :
647     case SRAV_____ :
648         snprintf(arg, size, "%s, %s, %s",
649                 regname[rd], regname[rt], regname[rs]);
650         break;
651     case MOVZ_____ :
652     case MOVN_____ :
653     case ADD_____ :
654     case ADDU_____ :
655     case SUB_____ :
656     case SUBU_____ :
657     case AND_____ :
658     case OR_____ :
659     case XOR_____ :
660     case NOR_____ :
661     case SLT_____ :
662     case SLTU_____ :
663     case MUL_____ :
664         snprintf(arg, size, "%s, %s, %s",
665                 regname[rd], regname[rs], regname[rt]);
666         break;
667     case TGEI_____ :
668     case TGEIU_____ :
669     case TLT_____ :
670     case TLTU_____ :
671     case TEQI_____ :
672     case TNEI_____ :
673         snprintf(arg, size, "%s, %d", regname[rs], exts32(imm, 16));
674         break;
675     case ADDI_____ :
676     case ADDIU_____ :
677     case SLTI_____ :
678     case SLTIU_____ :
679         snprintf(arg, size, "%s, %s, %d",
680                 regname[rt], regname[rs], exts32(imm, 16));
681         break;
682     case ANDI_____ :
683     case ORI_____ :
684     case XORI_____ :
685         snprintf(arg, size, "%s, %s, 0x%x",
686                 regname[rt], regname[rs], imm);
687         break;
688     case LUI_____ :
689         snprintf(arg, size, "%s, 0x%x", regname[rt], imm);
690         break;
691     case BLTZ_____ :
692     case BGEZ_____ :

```

Nov 05, 08 18:55

code.txt

Page 50/62

```

693     case BLTZL_____ :
694     case BGEZL_____ :
695     case BLTZALL_____ :
696     case BGEZALL_____ :
697     case BLTZALL_____ :
698     case BGEZALL_____ :
699     case BLEZ_____ :
700     case BGTZ_____ :
701     case BLEZL_____ :
702     case BGTZL_____ :
703         snprintf(arg, size, "%s, %08x",
704                 regname[rs], pc + 4 + (exts32(imm, 16) << 2));
705         break;
706     case BEQ_____ :
707     case BNE_____ :
708     case BEQL_____ :
709     case BNEL_____ :
710         snprintf(arg, size, "%s, %s, %08x", regname[rs], regname[rt],
711                 pc + 4 + (exts32(imm, 16) << 2));
712         break;
713     case J_____ :
714     case JAL_____ :
715         snprintf(arg, size, "%08x",
716                 ((pc + 4) & 0xf0000000) | (addr << 2));
717         break;
718     case SYSCALL_____ :
719         snprintf(arg, size, "0x%x", code_l);
720         break;
721     case BREAK_____ :
722         snprintf(arg, size, "0x%x", code_s);
723         break;
724     case MFC0_____ :
725     case CFC0_____ :
726     case MTC0_____ :
727         snprintf(arg, size, "%s, %d.%d", regname[rt], rd, sel);
728         break;
729     case LB_____ :
730     case LH_____ :
731     case LWL_____ :
732     case LW_____ :
733     case LBU_____ :
734     case LHU_____ :
735     case LWR_____ :
736     case SB_____ :
737     case SH_____ :
738     case SWL_____ :
739     case SW_____ :
740     case SWR_____ :
741     case CACHE_____ :
742     case LL_____ :
743     case PREF_____ :
744     case SC_____ :
745         snprintf(arg, size, "%s, %d(%s)",
746                 regname[rt], exts32(imm, 16), regname[rs]);
747         break;
748     }
749
750     snprintf(mnemonic, size, "%-9s%s", instname, arg);
751     DELETE_ARRAY(arg);
752     return mnemonic;
753 }
754
755 /*****
756 inline uint032_t exts32(uint032_t x, int y)
757 {
758     if (y == 32)
759         return x;
760     uint032_t temp = 0xffffffff << y;
761     if (x & (1 << (y - 1)))
762         return (temp | (x & ~temp));
763     else
764         return (x & ~temp);
765 }
766
767 /*****

```

Nov 05, 08 18:55

code.txt

Page 51/62

```

file name: cp0.cc
1  /*****
2  * SimMips: Simple Computer Simulator of MIPS Arch Lab. TOKYO TECH *
3  *****/
4  #include "define.h"
5
6  #define OX(arg) ((arg) ? 'o' : 'x')
7  #define READBITS(reg, pos) ((r[reg] >> pos ## _SH) & pos ## _MASK)
8  #define MODIFYBITS(reg, pos, code) modifyreg(reg, (code) << pos ## _SH,\
9  pos ## _MASK << pos ## _SH)
10
11 /*****/
12 enum {
13     EXC_HANDLE_BASE_NORM = 0x80000000,
14     EXC_HANDLE_BASE_BEV = 0xbfc00200,
15     EXC_HANDLE_TLB = 0x0,
16     EXC_HANDLE_GENERAL = 0x180,
17     EXC_HANDLE_INT = 0x200,
18
19     TLB_PFN_SH = 6,
20     TLB_CACHE_SH = 3,
21     TLB_DIRTY_SH = 2,
22     TLB_VALID_SH = 1,
23     TLB_CACHE_MASK = 0x7,
24     TLB_DIRTY_MASK = 0x1,
25     TLB_VALID_MASK = 0x1,
26     TLB_GLOBAL_MASK = 0x1,
27     TLB_ASID_MASK = 0xff,
28     PAGESHIFT_MAX = 24,
29
30     COMPARE_CONNECTED = 7,
31 };
32
33 /*****/
34 MipsTlbEntry::MipsTlbEntry()
35 {
36     vpn2 = 0;
37     asid = 0;
38     pagemask = 0;
39     pageshift = 0;
40     global = 0;
41     for (int i = 0; i < 2; i++) {
42         pfn[i] = 0;
43         valid[i] = 0;
44         dirty[i] = 0;
45         cache[i] = 0;
46     }
47 }
48
49 /*****/
50 void MipsTlbEntry::print()
51 {
52     printf("%08x %02x %08x | %c | ", vpn2 << TLB_VPAGE_SH, asid,
53           (pagemask << TLB_VPAGE_SH) | TLB_VPAGE_LOWER, OX(global));
54     for (int i = 0; i < 2; i++)
55         printf("%010llx %c%c%s",
56               (uint064_t) (pfn[i] & ~(pagemask >> 1)) << TLB_PPAGE_SH,
57               OX(valid[i]), OX(dirty[i]), OX(cache[i]),
58               (i == 0) ? " | " : "\n");
59 }
60
61 /*****/
62 MipsCp0::MipsCp0(Board *board, Chip *chip, int divisor)
63 {
64     this->board = board;
65     this->chip = chip;
66     this->divisor = divisor;
67     for (int i = 0; i < NREG; i++)
68         r[i] = 0;
69     counter = 0;
70 }
71
72 /*****/
73 int MipsCp0::gettlbentry(uint032_t page)
74 {
75     uint id = r[CP0_ENTRYHI_] & TLB_ASID_MASK;
76     for (int i = 0; i < TLB_ENTRY; i++) {

```

Nov 05, 08 18:55

code.txt

Page 52/62

```

77     uint032_t mask = ~tlb[i].pagemask;
78     if (((tlb[i].vpn2 & mask) == (page & mask)) &&
79         ((tlb[i].asid == id) || tlb[i].global))
80         return i;
81     }
82     return -1;
83 }
84
85 /*****/
86 void MipsCp0::step()
87 {
88     if ((++counter) < divisor)
89         return;
90     counter = 0;
91
92     r[CP0_COUNT_]++;
93     if (r[CP0_COUNT_] == r[CP0_COMPARE_])
94         setinterrupt(COMPARE_CONNECTED);
95
96     if (r[CP0_RANDOM_] <= r[CP0_WIRED_])
97         r[CP0_RANDOM_] = TLB_ENTRY - 1;
98     else
99         r[CP0_RANDOM_]--;
100 }
101
102 /*****/
103 void MipsCp0::tlbread()
104 {
105     uint x = r[CP0_INDEX_];
106     if (x >= TLB_ENTRY) {
107         printf("!! TLB READ ERROR: index is too large: %d\n", x);
108         exit(0);
109     }
110     r[CP0_ENTRYHI_] = (tlb[x].vpn2 << TLB_VPAGE_SH) | tlb[x].asid;
111     r[CP0_PAGEMASK_] = tlb[x].pagemask << TLB_VPAGE_SH;
112     for (int i = 0; i < 2; i++)
113         r[CP0_ENTRYLO0 + i] = (tlb[x].pfn[i] << TLB_PFN_SH |
114                               tlb[x].cache[i] << TLB_CACHE_SH |
115                               tlb[x].dirty[i] << TLB_DIRTY_SH |
116                               tlb[x].valid[i] << TLB_VALID_SH |
117                               tlb[x].global);
118 }
119
120 /*****/
121 void MipsCp0::tlbwrite(int use_random)
122 {
123     uint x = (uint) (use_random) ? r[CP0_RANDOM_] : r[CP0_INDEX_];
124     if (x >= TLB_ENTRY) {
125         printf("!! TLB WRITE ERROR: index is too large: %d\n", x);
126         exit(0);
127     }
128     tlb[x].vpn2 = r[CP0_ENTRYHI_] >> TLB_VPAGE_SH;
129     tlb[x].asid = r[CP0_ENTRYHI_] & TLB_ASID_MASK;
130     tlb[x].pagemask = 0;
131     tlb[x].pageshift = TLB_PPAGE_SH;
132     int pagemask = r[CP0_PAGEMASK_] >> TLB_VPAGE_SH;
133     for (uint i = 0; i <= PAGESHIFT_MAX - TLB_PPAGE_SH; i += 2) {
134         if (pagemask == (1 << i) - 1) {
135             tlb[x].pagemask = pagemask;
136             tlb[x].pageshift += i;
137             break;
138         }
139     }
140     tlb[x].global = r[CP0_ENTRYLO0] & TLB_GLOBAL_MASK;
141     for (int i = 0; i < 2; i++) {
142         uint032_t entrylo = r[CP0_ENTRYLO0 + i];
143         tlb[x].pfn[i] = entrylo >> TLB_PFN_SH;
144         tlb[x].cache[i] = (entrylo >> TLB_CACHE_SH) & TLB_CACHE_MASK;
145         tlb[x].dirty[i] = (entrylo >> TLB_DIRTY_SH) & TLB_DIRTY_MASK;
146         tlb[x].valid[i] = (entrylo >> TLB_VALID_SH) & TLB_VALID_MASK;
147     }
148     if (board->debug_mode == DEB_EXCTLB) {
149         printf("## TLB Wrote.\n");
150         tlbprint();
151     }
152 }
153

```

Nov 05, 08 18:55

code.txt

Page 53/62

```

154 /*****/
155 void MipsCp0::tlblookup()
156 {
157     r[CP0_INDEX_] = gettlbentry(r[CP0_ENTRYHI_] >> TLB_VPAGE_SH);
158 }
159
160 /*****/
161 uint032_t MipsCp0::readreg(int x)
162 {
163     return r[x];
164 }
165
166 /*****/
167 void MipsCp0::writereg(int x, uint032_t value)
168 {
169     r[x] = value;
170     if (x == CP0_COMPARE_)
171         clearinterrupt(COMPARE_CONNECTED);
172 }
173
174 /*****/
175 void MipsCp0::modifyreg(int x, uint032_t value, uint032_t mask)
176 {
177     r[x] = (value & mask) | (r[x] & ~mask);
178 }
179
180 /*****/
181 int MipsCp0::getphaddr(uint032_t vaddr, uint064_t *paddr, int store)
182 {
183     uint032_t sr = r[CP0_SR_____];
184     int kernel_mode = (((sr >> SR_KSU_SH) & SR_KSU_MASK) == 0) ||
185         (((sr >> SR_EXL_SH) & SR_ERLEXL_MASK) != 0);
186     if (vaddr >= KSEG0_MIN)
187     {
188         if (!kernel_mode) {
189             return (store) ? EXC_ADES_ : EXC_ADEL_;
190         } else if (vaddr < KSEG2_MIN) {
191             *paddr = (uint064_t) vaddr & UNMAP_MASK;
192             return 0;
193         }
194     }
195     int x = gettlbentry(vaddr >> TLB_VPAGE_SH);
196     if (x < 0)
197         return ((store) ? EXC_TLBS_ : EXC_TLBL_) | EXC_TLBREFL;
198     int odd = (vaddr >> tlb[x].pageshift) & 0x1;
199     uint064_t mask, tmp_addr;
200     mask = ~(uint064_t) tlb[x].pagemask << TLB_PPAGE_SH;
201     tmp_addr = (((uint064_t) tlb[x].pfno[odd] << TLB_PPAGE_SH) & mask) |
202         ((uint064_t) vaddr & ~mask);
203     if (!tlb[x].valid[odd])
204         return (store) ? EXC_TLBS_ : EXC_TLBL_;
205     if (!tlb[x].dirty[odd] && store)
206         return EXC_MOD_;
207     *paddr = tmp_addr;
208     return 0;
209 }
210
211 /*****/
212 uint032_t MipsCp0::doexception(int code, uint032_t epc,
213     uint032_t vaddr, int delay)
214 {
215     {
216         uint032_t base, restart_pc;
217         int refill = code & EXC_TLBREFL;
218         int cp1 = code & EXC_CPU1_;
219         code &= CAUSE_EXC_MASK;
220
221         // set cp0 registers
222         MODIFYBITS(CP0_CAUSE_, CAUSE_EXC, code);
223         if (code == EXC_CPU_)
224             MODIFYBITS(CP0_CAUSE_, CAUSE_CE, (cp1 ? 1 : 0);
225         if ((code >= EXC_MOD_) || (code <= EXC_ADES_)) {
226             writereg(CP0_BADVADDR, vaddr);
227             MODIFYBITS(CP0_CONTEXT_, CONT_BADV, vaddr >> TLB_VPAGE_SH);
228             MODIFYBITS(CP0_ENTRYHI_, TLB_VPAGE, vaddr >> TLB_VPAGE_SH);
229         }
230         if (!READBITS(CP0_SR_____, SR_EXL)) {

```

Nov 05, 08 18:55

code.txt

Page 54/62

```

231     writereg(CP0_EPC_____, (delay) ? epc - 4 : epc);
232     MODIFYBITS(CP0_CAUSE_____, CAUSE_BD, (delay) ? 1 : 0);
233 }
234
235 // jump to the address of interrupt handler
236 if (READBITS(CP0_SR_____, SR_BEV))
237     base = EXC_HANDLE_BASE_BEV;
238 else
239     base = EXC_HANDLE_BASE_NORM;
240
241 if (((code == EXC_TLBL_) || (code == EXC_TLBS_))
242     && refill && (!READBITS(CP0_SR_____, SR_EXL)))
243     restart_pc = base + EXC_HANDLE_TLB;
244 else if ((code == EXC_INT_)
245     && READBITS(CP0_CAUSE_____, CAUSE_IV))
246     restart_pc = base + EXC_HANDLE_INT;
247 else
248     restart_pc = base + EXC_HANDLE_GENERAL;
249
250 // set exception bit and restart
251 MODIFYBITS(CP0_SR_____, SR_EXL, 1);
252 return restart_pc;
253 }
254
255 /*****/
256 void MipsCp0::setinterrupt(int num)
257 {
258     r[CP0_CAUSE_] |= 1 << (CAUSE_IP_SH + num);
259 }
260
261 /*****/
262 void MipsCp0::clearinterrupt(int num)
263 {
264     r[CP0_CAUSE_] &= ~(1 << (CAUSE_IP_SH + num));
265 }
266
267 /*****/
268 int MipsCp0::checkinterrupt()
269 {
270     return (((r[CP0_SR_____] & r[CP0_CAUSE_])
271         >> CAUSE_IP_SH) & CAUSE_IP_MASK) &&
272         (((r[CP0_SR_____] >> SR_EXL_SH) & SR_ERLEXL_MASK) == 0) &&
273         (r[CP0_SR_____] & SR_IE_MASK != 0);
274 }
275
276 /*****/
277 void MipsCp0::regprint()
278 {
279     printf("[CP0 Register]\n");
280     for (int i = 0; i < 4; i++) {
281         for (int j = 0; j < 8; j++)
282             printf("cr%.2d ", (i % 4) * 8 + j);
283         printf("\n");
284         for (int j = 0; j < 8; j++)
285             printf("%08x ", r[i * 8 + j]);
286         printf("\n");
287     }
288     printf("cr16.1\n%08x\n", r[CP0_CONFIG1]);
289 }
290
291 /*****/
292 void MipsCp0::tlbprint()
293 {
294     printf("[TLB Entries]\n");
295     printf("    VPN AS      MASK | G |"
296         "    PFNO VDC |    PFN1 VDC\n");
297     for (int i = 0; i < TLB_ENTRY; i++)
298         tlb[i].print();
299     printf("\n");
300 }
301
302 /*****/
303 void MipsCp0::print()
304 {
305     regprint();
306     tlbprint();
307 }

```

Nov 05, 08 18:55

code.txt

Page 55/62

```

308
309  /*****

```

Nov 05, 08 18:55

code.txt

Page 56/62

```

file name: device.cc
1  /*****
2  * SimMips: Simple Computer Simulator of MIPS Arch Lab. TOKYO TECH *
3  *****/
4  #include "define.h"
5  #include <ncurses.h>
6
7  enum {
8      PIC_RD_IRR = 0,
9      PIC_RD_ISR = 1,
10     PIC_RD_BUFIRR = 2,
11     PIC_CONNECTED = 2,
12
13     PIC_PRI_ADDR = 0x20,
14     PIC_SEC_ADDR = 0xa0,
15     PIC_ADDR_RANGE = 2,
16
17     SIO_RBR = 0,
18     SIO_IER = 1,
19     SIO_IIR = 2,
20     SIO_LCR = 3,
21     SIO_MCR = 4,
22     SIO_LSR = 5,
23     SIO_MSR = 6,
24     SIO_SCR = 7,
25
26     SIO_IER_TX = 0x02,
27     SIO_IER_RX = 0x01,
28     SIO_IIR_FIFO = 0xc0,
29     SIO_FIFO_SH = 5,
30     SIO_IIR_TX = 0x02,
31     SIO_IIR_RX = 0x04,
32     SIO_IIR_NP = 0x01,
33     SIO_FCR_FIFO = 0x06,
34     SIO_LCR_DIV = 0x80,
35     SIO_MCR_INT = 0x08,
36     SIO_LSR_VALUE = 0x60,
37     SIO_MSR_VALUE = 0x0a,
38     SIO_CONNECTED = 4,
39
40     SIO_PRI_ADDR = 0x3f8,
41     SIO_ADDR_RANGE = 8,
42
43     SIO_POLL_CYCLE = 0x100,
44
45     MIERU_SW = 0x00,
46     MIERU_LCD = 0x04,
47     MIERU_SEG = 0x08,
48     MIERU_CNT = 0x0c,
49 };
50
51 /* Interrupt Controller, integrating TWO 8259-like PIC */
52 /*****
53 IntController::IntController(MipsCp0 *cp0)
54 {
55     this->cp0 = cp0;
56     for (int i = 0; i < 2; i++) {
57         imr[i] = 0xff;
58         irr[i] = 0;
59         isr[i] = 0;
60         tobe_read[i] = PIC_RD_BUFIRR;
61         init_mode[i] = 0;
62     }
63 }
64
65 /*****
66 int IntController::checkaddr(uint032_t addr)
67 {
68     uint032_t base = addr & ~0x1;
69     if (base == PIC_PRI_ADDR)
70         return 0;
71     else if (base == PIC_SEC_ADDR)
72         return 1;
73     else
74         return -1;
75 }
76 /*****

```

Nov 05, 08 18:55

code.txt

Page 57/62

```

77 void IntController::recalcirq()
78 {
79     // Secondary PIC(Irq 8-15) is connected to IRQ2
80     irr[0] = ((irr[0] & ~(1 << 2)) |
81         ((irr[1] & ~imr[1]) != 0) << 2));
82
83     for (int i = 0; i < 2; i++) {
84         isr[i] = 0;
85         for (int j = 0; j < 8; j++) {
86             if (irr[i] & ~imr[i] & (1 << j)) {
87                 isr[i] = 1 << j;
88                 break;
89             }
90         }
91     }
92     if (isr[0])
93         cp0->setinterrupt(PIC_CONNECTED);
94     else
95         cp0->clearinterrupt(PIC_CONNECTED);
96 }
97
98 /*****
99 void IntController::readlb(uint032_t addr, uint008_t *data)
100 {
101     int ch = checkaddr(addr);
102     if (ch < 0)
103         return;
104
105     if ((addr & 0x1) == 0) {
106         if (tobe_read[ch] == PIC_RD_IRR) {
107             *data = irr[ch];
108         } else if (tobe_read[ch] == PIC_RD_ISR) {
109             *data = isr[ch];
110         } else {
111             // PIC_RD_BUFIRR
112             *data = 0x00;
113             for (int i = 0; i < 8; i++) {
114                 if (irr[ch] & (1 << i)) {
115                     *data = 0x80 + i;
116                     break;
117                 }
118             }
119         } else {
120             *data = imr[ch];
121         }
122     }
123
124 /*****
125 void IntController::writelb(uint032_t addr, uint008_t data)
126 {
127     int ch = checkaddr(addr);
128     if (ch < 0)
129         return;
130
131     if ((addr & 0x1) == 0) {
132         if (data == 0x0a) {
133             tobe_read[ch] = PIC_RD_IRR;
134         } else if (data == 0x0b) {
135             tobe_read[ch] = PIC_RD_ISR;
136         } else if (data == 0x0c) {
137             tobe_read[ch] = PIC_RD_BUFIRR;
138         } else if ((data >= 0x10) && (data <= 0x1f)) {
139             init_mode[ch] = 2 + (data & 0x1);
140         } else if ((data >= 0x20) && (data <= 0x27)) {
141             irr[ch] &= ~isr[ch];
142             isr[ch] = 0;
143             recalcirq();
144         } else if ((data >= 0x60) && (data <= 0x67)) {
145             irr[ch] &= ~(1 << (data & 0xf));
146             isr[ch] &= ~(1 << (data & 0xf));
147             recalcirq();
148         } else {
149             printf("## IntController: undefined op: 0x%02x\n", data);
150         }
151     } else {
152         if (init_mode[ch]) {
153             init_mode[ch]--;

```

Nov 05, 08 18:55

code.txt

Page 58/62

```

154     } else {
155         imr[ch] = data;
156         recalcirq();
157     }
158 }
159
160
161 /****
162 void IntController::setinterrupt(int irq)
163 {
164     irr[irq / 8] |= 1 << (irq % 8);
165     recalcirq();
166 }
167
168 /****
169 void IntController::clearinterrupt(int irq)
170 {
171     irr[irq / 8] &= ~(1 << (irq % 8));
172     recalcirq();
173 }
174
175 /* Serial I/O Controller, simplifying ns16550 */
176 /****
177 SerialIO::SerialIO(IntController *pic)
178 {
179     this->pic = pic;
180     ier = 0;
181     iir = SIO_IIR_NP;
182     lcr = 0;
183     mcr = 0;
184     scr = 0;
185     counter = 0;
186     divisor = 12;
187     currentchar = -1;
188 }
189
190 /****
191 void SerialIO::step()
192 {
193     if (++counter < SIO_POLL_CYCLE)
194         return;
195
196     counter = 0;
197     if (charavail()) {
198         iir |= SIO_IIR_RX;
199         recalcirq();
200     }
201 }
202
203 /****
204 int SerialIO::charavail()
205 {
206     char buf;
207     int ret;
208
209     if (currentchar != -1)
210         return 1;
211     ret = read(STDIN_FILENO, &buf, sizeof(buf));
212     if (ret == 1) {
213         currentchar = ((int) buf) & 0xff;
214         return 1;
215     }
216     return 0;
217 }
218
219 /****
220 void SerialIO::recalcirq()
221 {
222     int int_pend = 0;
223     int_pend = ((ier & SIO_IER_TX) && (iir & SIO_IIR_TX));
224     int_pend = int_pend || ((ier & SIO_IER_RX) && (iir & SIO_IIR_RX));
225     iir = (iir & ~SIO_IIR_NP) | ((int_pend) ? 0 : SIO_IIR_NP);
226     int_pend = int_pend && (mcr & SIO_MCR_INT);
227     if (int_pend)
228         pic->setinterrupt(SIO_CONNECTED);
229     else
230         pic->clearinterrupt(SIO_CONNECTED);

```

Nov 05, 08 18:55

code.txt

Page 59/62

```

231 }
232
233 /*****/
234 void SerialIO::readlb(uint032_t addr, uint008_t *data)
235 {
236     switch(addr) {
237         case SIO_RBR:
238             if (lcr & SIO_LCR_DIV) {
239                 *data = divisor & 0xff;
240             } else {
241                 *data = (uint008_t) currentchar;
242                 currentchar = -1;
243                 iir &= ~SIO_IIR_RX;
244                 recalcirq();
245             }
246             break;
247         case SIO_IER:
248             *data = (lcr & SIO_LCR_DIV) ? divisor >> 8 : ier;
249             break;
250         case SIO_IIR:
251             *data = iir;
252             iir &= ~SIO_IIR_TX;
253             recalcirq();
254             break;
255         case SIO_LCR:
256             *data = lcr;
257             break;
258         case SIO_MCR:
259             *data = mcr;
260             break;
261         case SIO_LSR:
262             *data = SIO_LSR_VALUE + ((iir & SIO_IIR_RX) != 0);
263             break;
264         case SIO_MSR:
265             *data = SIO_MSR_VALUE;
266             break;
267         case SIO_SCR:
268             *data = scr;
269             break;
270     }
271 }
272
273 /*****/
274 void SerialIO::writelb(uint032_t addr, uint008_t data)
275 {
276     switch(addr) {
277         case SIO_RBR:
278             if (lcr & SIO_LCR_DIV) {
279                 divisor = (divisor & 0xff00) | data;
280             } else {
281                 putchar((int) data & 0xff);
282                 fflush(stdout);
283                 iir |= SIO_IIR_TX;
284                 recalcirq();
285             }
286             break;
287         case SIO_IER:
288             if (lcr & SIO_LCR_DIV) {
289                 divisor = (divisor & 0x00ff) | ((uint) data << 8);
290             } else {
291                 if (((ier & SIO_IER_TX) == 0) && ((data & SIO_IER_TX) != 0))
292                     iir |= SIO_IIR_TX;
293                 ier = data;
294                 recalcirq();
295             }
296             break;
297         case SIO_IIR:
298             iir = ((iir & ~SIO_IIR_FIFO) |
299                 ((data & SIO_FCR_FIFO) << SIO_FIFO_SH));
300             break;
301         case SIO_LCR:
302             lcr = data;
303             break;
304         case SIO_MCR:
305             mcr = data;
306             recalcirq();
307             break;

```

Nov 05, 08 18:55

code.txt

Page 60/62

```

308     case SIO_LSR:
309     case SIO_MSR:
310         // lsr, msr are read-only
311         break;
312     case SIO_SCR:
313         scr = data;
314         break;
315 }
316
317
318 /* ISA Bus I/O */
319 /*****/
320 IsaIO::IsaIO()
321 {
322     pic = NULL;
323     sio = NULL;
324 }
325
326 /*****/
327 IsaIO::~IsaIO()
328 {
329     DELETE(pic);
330     DELETE(sio);
331 }
332
333 /*****/
334 void IsaIO::init(Board *board)
335 {
336     pic = new IntController(board->chip->cp0);
337     sio = new SerialIO(pic);
338 }
339
340 /*****/
341 void IsaIO::step()
342 {
343     sio->step();
344 }
345
346 /*****/
347 void IsaIO::readlb(uint032_t addr, uint008_t *data)
348 {
349     if ((addr - PIC_PRI_ADDR < PIC_ADDR_RANGE) ||
350         (addr - PIC_SEC_ADDR < PIC_ADDR_RANGE))
351         pic->readlb(addr, data);
352     else if (addr - SIO_PRI_ADDR < SIO_ADDR_RANGE)
353         sio->readlb(addr - SIO_PRI_ADDR, data);
354     else
355         *data = 0x00;
356 }
357
358 /*****/
359 void IsaIO::writelb(uint032_t addr, uint008_t data)
360 {
361     if ((addr - PIC_PRI_ADDR < PIC_ADDR_RANGE) ||
362         (addr - PIC_SEC_ADDR < PIC_ADDR_RANGE))
363         pic->writelb(addr, data);
364     else if (addr - SIO_PRI_ADDR < SIO_ADDR_RANGE)
365         sio->writelb(addr - SIO_PRI_ADDR, data);
366 }
367
368 /* I/O for MieruPC */
369 /*****/
370 void MieruIO::init(Board *board)
371 {
372     this->board = board;
373     if (!board->debug_mode)
374         lcd_ttyopen();
375 }
376
377 /*****/
378 void MieruIO::fini()
379 {
380     lcd_ttyclose();
381 }
382
383 /*****/
384 void MieruIO::lcd_ttyopen()

```

Nov 05, 08 18:55

code.txt

Page 61/62

```

385 {
386     initscr();
387     start_color();
388
389     if (COLORS >= 8) {
390         init_pair(1, COLOR_YELLOW, COLOR_BLACK);
391         init_pair(2, COLOR_MAGENTA, COLOR_BLACK);
392         init_pair(3, COLOR_RED, COLOR_BLACK);
393         init_pair(4, COLOR_CYAN, COLOR_BLACK);
394         init_pair(5, COLOR_GREEN, COLOR_BLACK);
395         init_pair(6, COLOR_BLUE, COLOR_BLACK);
396         init_pair(7, COLOR_BLACK, COLOR_BLACK);
397     }
398
399     clear();
400     lcd_width = 40;
401     lcd_height = 15;
402     cursorex = cursory = 0;
403
404     for (int i = 0; i < lcd_height; i++) {
405         move(i, lcd_width);
406         printf("#");
407     }
408     move(lcd_height, 0);
409     for (int i = 0; i < lcd_width + 1; i++)
410         printf("#");
411 }
412
413 /*****
414 void MieruIO::lcd_ttyclose()
415 {
416     endwin();
417 }
418 */
419 /*****
420 void MieruIO::lcd_setcolor(int color)
421 {
422     attrset(COLOR_PAIR(((color & 0x80) >> 5) ^
423                         ((color & 0x10) >> 3) ^
424                         ((color & 0x02) >> 1) ^ 0x7));
425 }
426 */
427 /*****
428 void MieruIO::lcd_cls()
429 {
430     int x, y;
431     for (y = 0; y < lcd_height; y++) {
432         move(y, 0);
433         for (x = 0; x < lcd_width; x++)
434             printf(" ");
435     }
436     refresh();
437 }
438 */
439 /*****
440 int MieruIO::lcd_printf(char *fmt, ...)
441 {
442     char buf[256];
443     int i, ret;
444     va_list arg;
445     va_start(arg, fmt);
446
447     move(cursory, cursorex);
448     ret = vsnprintf(buf, 256, fmt, arg);
449     va_end(arg);
450
451     for (i = 0; i < ret; i++) {
452         printf("%c", buf[i]);
453         if (++cursorex >= lcd_width) {
454             cursorex = 0;
455             cursory = (cursory + 1) % lcd_height;
456             move(cursory, cursorex);
457         }
458     }
459     refresh();
460     return ret;
461 }

```

Nov 05, 08 18:55

code.txt

Page 62/62

```

462
463 /*****
464 void MieruIO::lcd_nextline()
465 {
466     cursorex = 0;
467     cursory = (cursory + 1) % lcd_height;
468 }
469 */
470 /*****
471 void MieruIO::read1b(uint032_t addr, uint008_t *data)
472 {
473     if (addr == MIERU_SW) {
474         char swbuf;
475         *data = 0;
476         if (read(STDIN_FILENO, &swbuf, sizeof(swbuf)) != 0)
477             *data = ((swbuf == 'z') ? 4 :
478                     (swbuf == 'x') ? 2 :
479                     (swbuf == 'c') ? 1 : 0);
480     } else if (addr == MIERU_LCD)
481         *data = 1;
482 }
483 */
484 /*****
485 void MieruIO::read4b(uint032_t addr, uint032_t *data)
486 {
487     if (addr == MIERU_CNT)
488         *data = (uint032_t) (board->gettime() / 10);
489 }
490 */
491 /*****
492 void MieruIO::write1b(uint032_t addr, uint008_t data)
493 {
494     if ((addr == MIERU_LCD) && (!board->debug_mode)) {
495         lcdbuf[lcdindex] = data;
496         if (data == '\r') {
497             lcdbuf[lcdindex] = '\0';
498             if (strncmp(lcdbuf, "CS", 2) == 0) {
499                 lcd_setcolor(strtol(lcdbuf + 2, NULL, 16));
500             } else if (strncmp(lcdbuf, "ER", 2) == 0) {
501                 lcd_cls();
502             } else if (strncmp(lcdbuf, "HP", 2) == 0) {
503                 char *endptr;
504                 cursorex = strtol(lcdbuf + 2, &endptr, 10);
505                 cursory = strtol(endptr + 1, NULL, 10);
506             } else if (strncmp(lcdbuf, "HW", 2) == 0) {
507                 lcdbuf[lcdindex - 1] = '\0';
508                 lcd_printf(lcdbuf + 3);
509             } else if (strncmp(lcdbuf, "HR", 2) == 0) {
510                 lcd_nextline();
511             }
512             lcdindex = 0;
513         } else {
514             lcdindex += (lcdindex != 99);
515         }
516     }
517 }
518 */
519 /*****
520 void MieruIO::write4b(uint032_t addr, uint032_t data)
521 {
522     int x7seg[24] = {12,13,13,12,11,11,12,14, 7, 8, 8, 7, 6, 6, 7, 9,
523                     2, 3, 3, 2, 1, 1, 2, 4};
524     int y7seg[24] = {18,19,21,22,21,19,20,22,18,19,21,22,21,19,20,22,
525                     18,19,21,22,21,19,20,22};
526
527     if ((addr == MIERU_SEG) && (!board->debug_mode)) {
528         for (int i = 0; i < 24; i++) {
529             move(y7seg[i], x7seg[i]);
530             printf((data & 0x1) ? "*" : " ");
531             data >>= 1;
532         }
533     }
534 }
535 */
536 /*****

```