Appendix E

Module Implementations

E.1 BSHAM Modules

E.1.1 *absmach* MI

E.1.1.1 Module implementation: `absmach.c`

```
#include "system.h"
#include "absmach.h"

/*****constants*****/

/*****types*****/

/*****module state*****/

static int acc; /*accumulator*/
static int pc; /*program counter*/
static int mem[AM_MEMSIZ]; /*memory*/

/*****local functions*****/
/*out := (state invariant holds =>
*    (an exception is specified in Execution-phase Exception Table of RS
*         => the associated exception identifier
*    | true => AM_NORMAL))
*/
static am_stat execexc()
{
     sy_instr cmd;
     int op;

     if (SY_OPO(mem[pc]))
          return(AM_NORMAL);
```

```
     if (SY_OP1(mem[pc])) {
          cmd = (sy_instr)mem[pc];
          if (pc < AM_MEMSIZ-1) {
               op = mem[pc+1];
               if (cmd == SY_LOADCON)
                    return(AM_NORMAL);
               /*we know that cmd != SY_LOADCON*/
               if (op >= 0 && op <= AM_MEMSIZ-1) {
                    if (cmd == SY_ADD) {
                         if (acc+mem[op] <= AM_MAXINT)
                              return(AM_NORMAL);
                         else
                              return(AM_ARITHEXC);
                    } else if (cmd == SY_SUBTRACT) {
                         if (acc-mem[op] >= 0)
                              return(AM_NORMAL);
                         else
                              return(AM_ARITHEXC);
                    } else
                         return(AM_NORMAL);
               }
               /*we know that op not in shamaddrT*/
               return(AM_ADDREXC);
          }
          /*we know that pc == AM_MEMSIZ-1*/
          return(AM_NOOPEXC);
     }
     /*we know that mem[pc] not in objectT*/
     return(AM_OBJECTEXC);
}

/*****access routines*****/

void am_s_init()
{
     int i;

     acc = 0;
     pc = 0;
     for (i = 0; i < AM_MEMSIZ; i++)
          mem[i] = 0;
     return;
}

void am_s_acc(i)
int i;
{
     if (i < 0 || i > AM_MAXINT) {
```

```
          am_int();
          return;
     }
     acc = i;
     return;
}

int am_g_acc()
{
     return(acc);
}

void am_s_pc(a)
int a;
{
     if (a < 0 || a > AM_MEMSIZ-1) {
          am_addr();
          return;
     }
     pc = a;
     return;
}

int am_g_pc()
{
     return(pc);
}

void am_s_mem(a,i)
int a,i;
{
     if (a < 0 || a > AM_MEMSIZ-1) {
          am_addr();
          return;
     }
     if (i < 0 || i > AM_MAXINT) {
          am_int();
          return;
     }
     mem[a] = i;
     return;
}

int am_g_mem(a)
int a;
{
     if (a < 0 || a > AM_MEMSIZ-1) {
          am_addr();
```

```
          return(0);
     }
     return(mem[a]);
}

am_stat am_sg_exec()
{
     am_stat stat;
     sy_instr cmd;
     int op;

     stat = execexc();
     if (stat != AM_NORMAL)
          return(stat);
     cmd = (sy_instr)mem[pc];
     if (cmd == SY_HALT)
          return(AM_HALT);
     if (cmd == SY_PRINT) {
          pc = (pc+1) % AM_MEMSIZ;
          return(AM_PRINT);
     }
     op = mem[pc+1];
     switch (cmd) {
     case SY_LOAD:
          acc = mem[op];
          pc = (pc+2) % AM_MEMSIZ;
          break;
     case SY_STORE:
          mem[op] = acc;
          pc = (pc+2) % AM_MEMSIZ;
          break;
     case SY_ADD:
          acc = acc+mem[op];
          pc = (pc+2) % AM_MEMSIZ;
          break;
     case SY_SUBTRACT:
          acc = acc-mem[op];
          pc = (pc+2) % AM_MEMSIZ;
          break;
     case SY_BRANCH:
          pc = op;
          break;
     case SY_BRANCHZERO:
          if (acc == 0)
               pc = op;
          else if (acc > 0)
               pc = (pc+2) % AM_MEMSIZ;
          break;
```

```
     case SY_BRANCHPOS:
          if (acc > 0)
               pc = op;
          else if (acc == 0)
               pc = (pc+2) % AM_MEMSIZ;
          break;
     case SY_LOADCON:
          acc = op;
          pc = (pc+2) % AM_MEMSIZ;
          break;
     }
     return(AM_NORMAL);
}

void am_g_dump()
{
     int id;

     printf ("acc=%d!pc=%d!\n",acc,pc);
     for (id = 0; id < AM_MEMSIZ; id++) {
          if (id%10 == 0)
               printf("%d: ",id);
          printf("%d ",mem[id]);
          if ((id+1)%10 == 0)
               printf("\n");
     }
}
```

E.1.1.2 Default exception handlers: `absmach_e.c`

```
#include "system.h"
#include "absmach.h"

void am_addr()
{
     fprintf(sy_excfilp,"Exception am_addr occurred\n");
}

void am_int()
{
     fprintf(sy_excfilp,"Exception am_int occurred\n");
}
```

E.1.2 *exec* MI

E.1.2.1 Module implementation: `exec.c`

```
#include "system.h"
```

```
#include "absmach.h"
#include "exec.h"
#ifdef ISHAM
#include "keybdin.h"
#include "scngeom.h"
#include "scndr.h"
#endif

/*****constants*****/

#define STEP 's'
#define EXIT 'e'

/*****types*****/

/*****module state*****/

/*****local functions*****/

/*buf := the exception message corresponding to exception identifier
*    excid and program counter pc
*/
void excmsg(excid,pc,buf)
am_stat excid;
int pc;
char *buf;
{
     char tmpbuf[80];

     sprintf(buf,"Execution exception at %d. ",pc);
     switch (excid) {
     case AM_ADDREXC:
          sprintf(tmpbuf,"Illegal operand: %d",am_g_mem(pc+1));
          break;
     case AM_ARITHEXC:
          sprintf(tmpbuf,"Arithmetic overflow");
          break;
     case AM_NOOPEXC:
          sprintf(tmpbuf,"Operand not accessible");
          break;
     case AM_OBJECTEXC:
          sprintf(tmpbuf,"Illegal instruction: %d",am_g_mem(pc));
          break;
     }
     strcat(buf,tmpbuf);
}

/*****access routines*****/
```

```
void ex_s_init()
{
     return;
}

void ex_s_exec()
{
     am_stat stat;
     char buf[80];
#ifdef ISHAM
     char ch;
     int oldpc;
#endif

     am_s_acc(0);
     am_s_pc(0);
#ifdef BSHAM
     stat = am_sg_exec();
     while (stat == AM_NORMAL || stat == AM_PRINT) {
          if (stat == AM_PRINT)
               printf("%d\n",am_g_acc());
          stat = am_sg_exec();
     }
     if (stat != AM_HALT) {
          excmsg(stat,am_g_pc(),buf);
          printf("%s\n",buf);
     }
#else
     /*clear screen, display constants*/
     sd_s_clrscn();
     sd_s_con();
     /*display initial values*/
     sd_s_mem();
     sd_s_acc();
     sd_s_pc();
     /*highlight current instruction*/
     sd_s_hlt(am_g_pc(),1);
     ch = ki_sg_next();
     while (ch != EXIT) {
          if (ch == STEP) {
               oldpc = am_g_pc();
               stat = am_sg_exec();
               if (stat != AM_PRINT && stat != AM_NORMAL &&
                   stat != AM_HALT) {
                    excmsg(stat,am_g_pc(),buf);
                    sd_s_msg(buf);
               } else if (stat == AM_HALT) {
```

```
                    sd_s_msg("HALT instruction reached");
               } else {
                    /*update screen*/
                    sd_s_msg("");
                    if (stat == AM_PRINT)
                         sd_s_prt(am_g_acc());
                    sd_s_mem();
                    sd_s_acc();
                    sd_s_pc();
                    /*update highlighting of cursor*/
                    sd_s_hlt(oldpc,0);
                    sd_s_hlt(am_g_pc(),1);
               }
          } else
               sd_s_msg(
                    "Illegal keyboard entry: type \"s\" or \"e\".");
          ch = ki_sg_next();
     }
     /*clear screen*/
     sd_s_clrscn();
#endif
}
```

E.1.2.2 Default exception handlers

There are no exceptions for *exec*.

E.1.3 *load* MI

E.1.3.1 Module implementation: `load.c`

```
#include "system.h"
#include "load.h"
#include "absmach.h"
#include "token.h"

/*****constants*****/

/*****types*****/

typedef enum
     {OPFMTEXC,SOURCEEXC,BLANKLINEXC,MISSINGOPEXC,NOMEMEXC} excid_t;

/*****module state*****/

/*****local functions*****/
/*out,instr :=
*    ((exists i)(s = i.source) => true,i
```

```
*    | true => false,SY_HALT)
*/
static int getinstr(s,instr)
char s[];
sy_instr *instr;
{
     static struct {
          char *src;
          sy_instr instr;
     } tbl[] = {
          {"load",SY_LOAD},
          {"store",SY_STORE},
          {"add",SY_ADD},
          {"sub",SY_SUBTRACT},
          {"br",SY_BRANCH},
          {"brz",SY_BRANCHZERO},
          {"brp",SY_BRANCHPOS},
          {"loadcon",SY_LOADCON},
          {"print",SY_PRINT},
          {"halt",SY_HALT},
          {NULL,SY_HALT} /*terminator*/
     };
     int i;

     i = 0;
     while (tbl[i].src) {
          if (!strcmp(s,tbl[i].src)) {
               *instr = tbl[i].instr;
               return(1);
          }
          i++;
     }
     *instr = SY_HALT;
     return(0);
}

/*write to stdout the message corresponding to the exception identifier
*    in the first argument, with line number lin, and token tok
*/
static void excmsg(excid,lin,tok)
excid_t excid;
int lin;
char tok[];
{
     switch (excid) {
     case BLANKLINEXC:
          printf("Load exception at %d. Blank line illegal\n",lin);
          break;
```

```
     case MISSINGOPEXC:
          printf("Load exception at %d. Operand missing\n",lin);
          break;
     case NOMEMEXC:
          printf("Load exception at %d. Program too large\n",lin);
          break;
     case OPFMTEXC:
          printf("Load exception at %d. Illegal operand: %s\n",lin,tok);
          break;
     case SOURCEEXC:
          printf("Load exception at %d. Illegal instruction: %s\n",
               lin,tok);
          break;
     }
}

/*(line number lin, consisting of the string buf, contains an error =>
*    write the appropriate message to stdout
*    out := LD_ERROR
* | true =>
*    *instr := SHAM instruction in buf
*    *arg := argument, if any, to the instruction
*    out := LD_NORMAL)
*/
static ld_stat parse(buf,lin,instr,arg)
char buf[];
int lin,*arg;
sy_instr *instr;
{
     tk_valtyp tok;

     tk_s_str(buf);
     if (tk_g_end()) {
          excmsg(BLANKLINEXC,lin,"");
          return(LD_ERROR);
     }
     /*we know that there is at least one token*/
     tk_sg_next(&tok);
     if (getinstr(tok.val,instr)) {
          if (SY_OP0(*instr))
               return(LD_NORMAL);
          /*we know that SY_OP1(*instr)*/
          if (tk_g_end()) {
               excmsg(MISSINGOPEXC,lin,"");
               return(LD_ERROR);
          }
          /*we know that there is more than one token*/
          tk_sg_next(&tok);
```

```
          if (tok.typ != TK_INT) {
               excmsg(OPFMTEXC,lin,tok.val);
               return(LD_ERROR);
          }
          *arg = atoi(tok.val);
          if (*instr == SY_LOADCON) {
               if (*arg >= 0 && *arg <= AM_MAXINT)
                    return(LD_NORMAL);
               else {
                    excmsg(OPFMTEXC,lin,tok.val);
                    return(LD_ERROR);
               }
          } else {
               if (*arg >= 0 && *arg < AM_MEMSIZ)
                    return(LD_NORMAL);
               else {
                    excmsg(OPFMTEXC,lin,tok.val);
                    return(LD_ERROR);
               }
          }
     }
     /*we know that tok.val not in sourceT*/
     excmsg(SOURCEEXC,lin,tok.val);
     return(LD_ERROR);
}

/*****access routines*****/

void ld_s_init()
{
     return;
}

ld_stat ld_sg_load(f)
ld_filptr f;
{
     char buf[TK_MAXSTRLEN+2]; /*extra characters needed for fgets*/
     int index,lin,arg;
     ld_stat stat;
     sy_instr instr;

     index = 0;
     lin = 0;
     stat = LD_NORMAL;
     while (fgets(buf,TK_MAXSTRLEN+2,f) != NULL) {
          /*fgets stores newline character, which must be eliminated*/
          buf[strlen(buf)-1] = '\0';
          lin++;
```

```
          if (parse(buf,lin,&instr,&arg) == LD_ERROR)
               stat = LD_ERROR;
          else if (stat == LD_NORMAL) {
               if (index < AM_MEMSIZ-1 ||
                    (index == AM_MEMSIZ-1 && SY_OPO(instr))) {
                    am_s_mem(index++,(int)instr);
                    if (SY_OP1(instr))
                         am_s_mem(index++,arg);
               } else {
                    excmsg(NOMEMEXC,lin,"");
                    stat = LD_ERROR;
               }
          }
     }
     return(stat);
}
```

E.1.3.2 Default exception handlers: `load_e.c`

```
#include "system.h"
#include "load.h"

void ld_fil()
{
     fprintf(sy_excfilp,"Exception ld_fil occurred\n");
}
```

E.1.4 *sham* MI

E.1.4.1 Module implementation: `sham.c`

```
#include "system.h"
#include "absmach.h"
#include "token.h"
#include "exec.h"
#include "load.h"
#ifdef ISHAM
#include "keybdin.h"
#include "scngeom.h"
#include "scnstr.h"
#include "scndr.h"
#endif

/*****constants*****/

/*****types*****/

/*****variables*****/
```

```
FILE *sy_excfilp;

/*****local functions*****/

/*****main*****/

main(argc,argv)
int argc;
char *argv[];
{
     FILE *fp;

     /*check command line arguments*/
     if (argc == 1) {
          printf("Command line error. ");
          printf("No file name specified\n");
          return(0);
     } else {
          fp = fopen(argv[1],"r");
          if (fp == NULL) {
               printf("Command line error. ");
               printf("Cannot open file: %s\n",argv[1]);
               return(0);
          }
     }

     /*initialize exception file pointer*/
#ifdef BSHAM
     sy_excfilp = stdout;
#else
     sy_excfilp = fopen(SY_EXCFIL,"a");
#endif

     am_s_init();
     tk_s_init();
     ld_s_init();
     ex_s_init();

     if (ld_sg_load(fp) == LD_NORMAL) {
#ifdef ISHAM
          /*initialize curses*/
          initscr();

          /*initialize keyboard and screen handling*/
          ki_s_init();
          sg_s_init();
          ss_s_init();
```

```
          sd_s_init();
#endif

          ex_s_exec();

#ifdef ISHAM
          /*terminate keyboard and screen handling*/
          ss_s_end();
          ki_s_end();

          /*terminate curses*/
          endwin();
#endif
     }

#ifdef ISHAM
     /*close exception file*/
     fclose(sy_excfilp);
#endif
     return(0);
}
```

E.1.4.2 Default exception handlers

There are no exceptions for *sham*.

E.1.5 *token* MI

E.1.5.1 Module implementation: `token.c`

```
#include <ctype.h>
#include "system.h"
#include "token.h"

/*****constants*****/

/*****types*****/

/*****module state*****/

static char buf[TK_MAXSTRLEN+2]; /*scanned string; space for sentinel*/
static int cur; /*current char in buf*/

/*****local functions*****/

/*****access routines*****/

void tk_s_init()
```

```
{
     buf[0] = '\0';
     cur = 0;
}

void tk_s_str(s)
char *s;
{
     if (strlen(s) > TK_MAXSTRLEN) {
          tk_maxlen();
          return;
     }
     while (*s == ' ') /*skip over leading blanks*/
          s++;
     strcpy(buf,s); /*copy in what remains*/
     if (*s != '\0')
          strcat(buf," "); /*add trailing blank as sentinel*/
     cur = 0;
}

void tk_sg_next(valtyp)
tk_valtyp *valtyp;
{
     enum {START,INT,ID,ERR,END} state; /*lexical analyzer state*/
     int tokstart,tokend,toklen;
     int i;

     if (buf[cur] == '\0') {
          tk_end();
          valtyp->val[0] = '\0';
          valtyp->typ = TK_BADTOK;
          return;
     }
     tokstart = cur; /*needed later to save value of token*/
     state = START;
     while (state != END) {
          switch (state) {
          case START:
               if (isalpha(buf[cur])) {
                    state = ID;
                    cur++;
               } else if (isdigit(buf[cur])) {
                    state = INT;
                    cur++;
               } else {
                    state = ERR;
                    cur++;
               }
```

```
            break;
        case ID:
            if (buf[cur] == ' ') {
                state = END;
                tokend = cur-1;
                valtyp->typ = TK_ID;
            } else if (isalnum(buf[cur]))
                cur++;
            else {
                state = ERR;
                cur++;
            }
            break;
        case INT:
            if (buf[cur] == ' ') {
                state = END;
                tokend = cur-1;
                valtyp->typ = TK_INT;
            } else if (isdigit(buf[cur]))
                cur++;
            else {
                state = ERR;
                cur++;
            }
            break;
        case ERR:
            if (buf[cur] == ' ') {
                state = END;
                tokend = cur-1;
                valtyp->typ = TK_BADTOK;
            } else
                ++cur;
            break;
        }
    }
    /*check maximum lengths*/
    switch (valtyp->typ) {
    case TK_ID:
        if (tokend-tokstart+1 > TK_MAXIDLEN)
            valtyp->typ = TK_BADTOK;
        break;
    case TK_INT:
        if (tokend-tokstart+1 > TK_MAXINTLEN)
            valtyp->typ = TK_BADTOK;
        break;
    }
    /*copy token to valtyp*/
    toklen = tokend-tokstart+1;
```

```
     for (i = 0; i < toklen; i++)
          valtyp->val[i] = buf[tokstart+i];
     valtyp->val[toklen] = '\0';
     /*skip over blanks preceding next token*/
     while (buf[cur] == ' ')
          cur++;
}

/*boolean*/ int tk_g_end()
{
     return(buf[cur] == '\0');
}

void tk_g_dump()
{
     printf("cur=%d!\n",cur);
     printf("buf=!%s!\n",buf);
}
```

E.1.5.2 Default exception handlers: `token_e.c`

```
#include "system.h"
#include "token.h"

void tk_maxlen()
{
     fprintf(sy_excfilp,"Exception tk_maxlen occurred\n");
}

void tk_end()
{
     fprintf(sy_excfilp,"Exception tk_end occurred\n");
}
```

E.2 ISHAM Modules

E.2.1 *keybdin* MI

E.2.1.1 Module implementation: `keybdin.c`

```
#include <curses.h>
#include "system.h"
#include "keybdin.h"

/*****constants*****/

/*****types*****/
```

```
/*****module state*****/

/*****local functions*****/

/*****access routines*****/

void ki_s_init()
{
     cbreak();
     noecho();
}

char ki_sg_next()
{
     return(getch());
}

void ki_s_end()
{
     echo();
     nocbreak();
}
```

E.2.1.2 Default exception handlers

There are no exceptions for *keybdin*.

E.2.2 *scndr* MI

E.2.2.1 Module implementation: `scndr.c`

```
#include "system.h"
#include "absmach.h"
#include "scngeom.h"
#include "scnstr.h"
#include "scndr.h"

/*****constants*****/

/*****types*****/

/*****module state*****/

/*****local functions*****/

#define FLD(f,t,r,c) (f.nam = t, f.row = r, f.col = c)

/*print constant field with fieldname t, row r, and column c*/
```

```
static void prtcon(t,r,c)
sg_fldnam t;
int r,c;
{
     sg_fld f;

     FLD(f,t,r,c);
     ss_s_str(sg_g_row(f),sg_g_col(f),sg_g_val(f));
}

/*pad s on the left with blanks to length l
*    truncate s to l if longer than l
*/
static void rjust(s,l)
char *s;
int l;
{
     int shift;

     s[l] = '\0';
     shift = l-strlen(s);
     while (--l >= shift)
          s[l] = s[l-shift];
     while (l >= 0)
          s[l--] = ' ';
}

/*pad s on the right with blanks to length l
*    truncate s to l if longer than l
*/
static void ljust(s,l)
char *s;
int l;
{
     int i;

     for (i = strlen(s); i < l; i++)
          s[i] = ' ';
     s[l] = '\0';
}

/*****access routines*****/

void sd_s_init()
{
     /*do nothing*/
}
```

```
void sd_s_clrscn()
{
     ss_s_clrscn();
     ss_s_ref();
}

void sd_s_con()
{
     int i;

     prtcon(SG_SCNTTL,0,0);
     prtcon(SG_MEMTTL1,0,0);
     prtcon(SG_MEMTTL2,0,0);
     for (i = 0; i < 10; i++) {
          prtcon(SG_MEMCOLHDR,0,i);
          prtcon(SG_MEMROWHDR,i,0);
     }
     prtcon(SG_PCTTL,0,0);
     prtcon(SG_ACCTTL,0,0);
     prtcon(SG_PRTTTL,0,0);
     prtcon(SG_PROMPTTTL,0,0);
     prtcon(SG_MSGTTL,0,0);
     ss_s_ref();
}

void sd_s_mem()
{
     sg_fld f;
     int r,c;
     char s[SG_NUMCOL+1];

     for (r = 0; r < 10; r++) {
          for (c = 0; c < 10; c++) {
               FLD(f,SG_MEM,r,c);
               sprintf(s,"%d",am_g_mem(10*r+c));
               rjust(s,sg_g_len(f));
               ss_s_str(sg_g_row(f),sg_g_col(f),s);
          }
     }
     ss_s_ref();
}

void sd_s_pc()
{
     sg_fld f;
     char s[SG_NUMCOL+1];

     FLD(f,SG_PC,0,0);
```

```
     sprintf(s,"%d",am_g_pc());
     rjust(s,sg_g_len(f));
     ss_s_str(sg_g_row(f),sg_g_col(f),s);
     ss_s_ref();
}

void sd_s_acc()
{
     sg_fld f;
     char s[SG_NUMCOL+1];

     FLD(f,SG_ACC,0,0);
     sprintf(s,"%d",am_g_acc());
     rjust(s,sg_g_len(f));
     ss_s_str(sg_g_row(f),sg_g_col(f),s);
     ss_s_ref();
}

void sd_s_prt(val)
int val;
{
     sg_fld f;
     char s[SG_NUMCOL+1];

     FLD(f,SG_PRT,0,0);
     sprintf(s,"%d",val);
     rjust(s,sg_g_len(f));
     ss_s_str(sg_g_row(f),sg_g_col(f),s);
     ss_s_ref();
}

void sd_s_msg(msg)
char *msg;
{
     sg_fld f;
     char s[SG_NUMCOL+1];

     FLD(f,SG_MSG,0,0);
     strcpy(s,msg);
     ljust(s,sg_g_len(f));
     ss_s_str(sg_g_row(f),sg_g_col(f),s);
     ss_s_ref();
}

void sd_s_hlt(a,f)
int a,f;
{
     sg_fld fld;
```

```
     FLD(fld,SG_MEM,a/10,a%10);
     ss_s_hlt(sg_g_row(fld),sg_g_col(fld),sg_g_len(fld),f);
     ss_s_ref();
}
```

E.2.2.2 Default exception handlers

There are no exceptions for *scndr*.

E.2.3 *scngeom* MI

E.2.3.1 Module implementation: `scngeom.c`

```
#include "system.h"
#include "scngeom.h"

/*****constants*****/

/*NOTE: because the variable below is initialized and never changed
*it is listed as a constant.
*/

static struct {
     int row,col,len;
     char *val;
} fldtbl[] = {
     /*SG_MEM*/          {5,15,3,""},
     /*SG_PC*/      {17,45,2,""},
     /*SG_ACC*/          {18,44,3,""},
     /*SG_PRT*/          {19,44,3,""},
     /*SG_MSG*/          {23,9,63,""},

     /*SG_SCNTTL*/       {0,33,4,"SHAM"},
     /*SG_MEMTTL1*/      {5,0,4,"Main"},
     /*SG_MEMTTL2*/      {6,0,7,"memory:"},
     /*SG_MEMCOLHDR*/    {3,15,3,""},   /*depends on row*/
     /*SG_MEMROWHDR*/    {5,9,3,""},    /*depends on column*/
     /*SG_PCTTL*/        {17,24,19,"   Program counter:"},
     /*SG_ACCTTL*/       {18,24,19,"       Accumulator:"},
     /*SG_PRTTTL*/       {19,24,19,"Last value printed:"},
     /*SG_PROMPTTTL*/    {22,0,46,
          "Enter command: \"s\" to single step; \"e\" to exit"},
     /*SG_MSGTTL*/       {23,0,8,"Message:"}
};

/*****types*****/
```

```
/*****module state*****/

/*****local functions*****/

/*out := (fld is a legal field identifier)*/
static int legfld(fld)
sg_fld fld;
{
     int maxrow,maxcol;

     if (fld.nam == SG_MEM || fld.nam == SG_MEMROWHDR)
          maxrow = 9;
     else
          maxrow = 0;
     if (fld.nam == SG_MEM || fld.nam == SG_MEMCOLHDR)
          maxcol = 9;
     else
          maxcol = 0;
     return(
          (int)fld.nam >= 0 && (int)fld.nam <= (int)SG_MSGTTL &&
          fld.row >= 0 && fld.row <= maxrow &&
          fld.col >= 0 && fld.col <= maxcol
     );
}

/*****access routines*****/

void sg_s_init()
{
     /*do nothing*/
}

int sg_g_legfld(fld)
sg_fld fld;
{
     return(legfld(fld));
}

int sg_g_row(fld)
sg_fld fld;
{
     if (!legfld(fld)) {
          sg_badfld();
          return(0);
     }
     return(fldtbl[(int)fld.nam].row+fld.row);
}
```

```
int sg_g_col(fld)
sg_fld fld;
{
     if (!legfld(fld)) {
          sg_badfld();
          return(0);
     }
     return(fldtbl[(int)fld.nam].col+fld.col*6);
}

int sg_g_len(fld)
sg_fld fld;
{
     if (!legfld(fld)) {
          sg_badfld();
          return(0);
     }
     return(fldtbl[(int)fld.nam].len);
}

char *sg_g_val(fld)
sg_fld fld;
{
     char str[4];

     if (!legfld(fld)) {
          sg_badfld();
          return(NULL);
     }
     if (fld.nam == SG_MEMCOLHDR) {
          sprintf(str,"%3d",fld.col);
          return(str);
     } else if (fld.nam == SG_MEMROWHDR) {
          sprintf(str,"%3d",10*fld.row);
          return(str);
     }
     return(fldtbl[(int)fld.nam].val);
}
```

E.2.3.2 Default exception handlers: `scngeom_e.c`

```
#include "system.h"
#include "scngeom.h"

void sg_badfld()
{
     fprintf(sy_excfilp,"Exception sg_badfld occurred\n");
}
```

E.2.4 *scnstr* MI

E.2.4.1 Module implementation: `scnstr.c`

```
#include <curses.h>
#include "system.h"
#include "scnstr.h"

/*****constants*****/

/*****types*****/

/*****module state*****/

/*****local functions*****/

/*instr(r,c,l,s)
*    load into s the string at positions (r,c) thru (r,c+l-1)
*/
static void instr(r,c,l,s)
int r,c,l;
char *s;
{
     int i;

     for (i = 0; i < l; i++) {
          move(r,c+i);
          s[i] = inch();
     }
     s[l] = '\0';
}

/*****access routines*****/

void ss_s_init()
{
     /*do nothing*/
}

void ss_s_clrscn()
{
     clear();
}

void ss_s_str(r,c,s)
int r,c;
char *s;
{
     if (r < 0 || r >= SS_NUMROW) {
```

```
          ss_row();
          return;
     } else if (c < 0 || c >= SS_NUMCOL) {
          ss_col();
          return;
     } else if (strlen(s) > SS_NUMCOL-c) {
          ss_len();
          return;
     }
     if (strlen(s) > 0) { /*handle the 0-length case neutrally*/
          move(r,c);
          addstr(s);
     }
}

void ss_s_hlt(r,c,l,f)
int r,c,l,f;
{
     char s[SS_NUMCOL+1];

     if (r < 0 || r >= SS_NUMROW) {
          ss_row();
          return;
     } else if (c < 0 || c >= SS_NUMCOL) {
          ss_col();
          return;
     } else if (l < 0 || l > SS_NUMCOL-c) {
          ss_len();
          return;
     }
     if (l > 0) {
          instr(r,c,l,s);
          if (f) {
               standout();
               move(r,c);
               addstr(s);
               standend();
          } else {
               move(r,c);
               addstr(s);
          }
     }
}

void ss_s_cur(r,c)
int r,c;
{
     if (r < 0 || r >= SS_NUMROW) {
```

```
          ss_row();
          return;
     } else if (c < 0 || c >= SS_NUMCOL) {
          ss_col();
          return;
     }
     move(r,c);
}

void ss_s_ref()
{
     refresh();
}

void ss_s_end()
{
     /*do nothing*/
}
```

E.2.4.2 Default exception handlers: `scnstr_e.c`

```
#include "system.h"
#include "scnstr.h"

void ss_row()
{
     fprintf(sy_excfilp,"Exception ss_row occurred\n");
}

void ss_col()
{
     fprintf(sy_excfilp,"Exception ss_col occurred\n");
}

void ss_len()
{
     fprintf(sy_excfilp,"Exception ss_len occurred\n");
}
```

E.3 Demonstration Modules

E.3.1 *stack* MI

E.3.1.1 Module implementation: `stack.c`

```
#include "system.h"
#include "stack.h"
```

```
/*****constants*****/

/*****types*****/

/*****module state*****/

static int stack[PS_MAXSIZ]; /*stack elements*/
static int siz; /*number of elements in stack*/

/*****local functions*****/

/*****access routines*****/

void ps_s_init()
{
     siz = 0;
}

void ps_s_push(x)
int x;
{
     if (siz == PS_MAXSIZ) {
          ps_full();
          return;
     }
     stack[siz++] = x;
}

void ps_s_pop()
{
     if (siz == 0) {
          ps_empty();
          return;
     }
     --siz;
}

int ps_g_top()
{
     if (siz == 0) {
          ps_empty();
          return(0);
     }
     return(stack[siz-1]);
}

int ps_g_depth()
{
```

```
     return(siz);
}

void ps_g_dump()
{
     int i;

     printf("siz=%d\n",siz);
     for (i = 0; i < siz; i++)
          printf("stack[%d]=%d\n",i,stack[i]);
}
```

E.3.1.2 Default exception handlers: `stack_e.c`

```
#include "system.h"
#include "stack.h"

void ps_empty()
{
     fprintf(sy_excfilp,"Exception ps_empty occurred\n");
}

void ps_full()
{
     fprintf(sy_excfilp,"Exception ps_full occurred\n");
}
```

E.3.2 *symtbl* MI

E.3.2.1 Module implementation: `symtbl.c`

```
#include "system.h"
#include "symtbl.h"

/*****constants*****/

#define NOTFOUND -1

/*****types*****/

/*****module state*****/

static struct {
     char sym[ST_MAXSYMLEN+1]; /*symbol value*/
     int loc; /*symbol location*/
} tbl[ST_MAXSYMS]; /*one entry per symbol*/

static int tblcnt; /*number of symbols in tbl*/
```

```
/*****local functions*****/

/*out := (state invariant holds =>
*         ((exists i in [0,tblcnt-1])(s = tbl[i].sym)) => i
*         | true => NOTFOUND))
*/
static int findsym(sym)
char *sym;
{
     int i;

     for (i = 0; i < tblcnt; i++) {
          if (!strcmp(sym,tbl[i].sym))
               return(i);
     }
     return(NOTFOUND);
}

/*****access routines*****/

void st_s_init()
{
     tblcnt = 0;
}

void st_s_add(sym,loc)
char *sym;
int loc;
{
     if (strlen(sym) > ST_MAXSYMLEN) {
          st_maxlen();
          return;
     } else if (findsym(sym) != NOTFOUND) {
          st_exsym();
          return;
     } else if (tblcnt == ST_MAXSYMS) {
          st_full();
          return;
     }
     strcpy(tbl[tblcnt].sym,sym);
     tbl[tblcnt].loc = loc;
     tblcnt++;
}

int st_g_exsym(sym)
char *sym;
{
```

```
     return(findsym(sym) != NOTFOUND);
}

void st_s_loc(sym,loc)
char *sym;
int loc;
{
     int i;

     i = findsym(sym);
     if (i == NOTFOUND) {
          st_notexsym();
          return;
     }
     tbl[i].loc = loc;
}

int st_g_loc(sym)
char *sym;
{
     int i;

     i = findsym(sym);
     if (i == NOTFOUND) {
          st_notexsym();
          return(0);
     }
     return(tbl[i].loc);
}

int st_g_siz()
{
     return(tblcnt);
}

void st_g_dump()
{
     int i;

     printf ("tblcnt=%d!ST_MAXSYMS=%d!\n",tblcnt,ST_MAXSYMS);
     for (i = 0; i < tblcnt; i++)
          printf("tbl[%d].sym=%s!.loc=%d\n",i,tbl[i].sym,tbl[i].loc);
}
```

E.3.2.2 Default exception handlers: `symtbl_e.c`

```
#include "system.h"
#include "symtbl.h"
```

```
void st_exsym()
{
     fprintf(sy_excfilp,"Exception st_exsym occurred\n");
}

void st_maxlen()
{
     fprintf(sy_excfilp,"Exception st_maxlen occurred\n");
}

void st_notexsym()
{
     fprintf(sy_excfilp,"Exception st_notexsym occurred\n");
}

void st_full()
{
     fprintf(sy_excfilp,"Exception st_full occurred\n");
}
```