Appendix F

Test Plans and Implementations

F.1 BSHAM Modules

F.1.1 *absmach* TP and TI

F.1.1.1 Test plan

assumptions

  AM_MEMSIZ > 2
  AM_MAXINT $\geq 2 \times$ AM_MEMSIZ

test environment

  PGMGEN driver
  no stubs

test case selection strategy

  special values
    module state
      based of content of *mem*, *pc*, and *acc*, test for:
      run-time exceptions
        all exceptions for all instructions at least once
      AM_HALT
        HALT instruction with $pc \in \{0, \texttt{AM\_MEMSIZ} - 1\}$
      AM_PRINT
        PRINT instruction with $pc \in \{0, \texttt{AM\_MEMSIZ} - 1\}$
      AM_NORMAL
        all instructions with one operand and $pc = \texttt{AM\_MEMSIZ} - 2$
        all instructions with address argument:
          interval rule on address: $[0, \texttt{AM\_MEMSIZ}]$
        each instruction at least once

access routine parameters
- `am_s_acc`(i), `am_s_pc`(a), `am_s_mem`(a, i), and `am_g_mem`(a):
  - interval rule for a: $[0,$ `AM_MEMSIZ`$]$
  - interval rule for i: $[0,$ `AM_MAXINT`$]$

test cases
- exceptions
  - `am_s_mem` and `am_g_mem`
    - generate `am_addr`
  - `am_s_acc`, `am_s_pc`, and `am_s_mem`
    - generate `am_int`
- normal case
  - `am_s_acc` and `am_g_acc`
    - `am_g_acc` after `am_s_init` and after `am_s_acc`
  - `am_s_pc` and `am_g_pc`
    - `am_g_pc` after `am_s_init` and after `am_s_pc`
  - `am_s_mem` and `am_g_mem`
    - `am_g_mem` after `am_s_init` and after `am_s_mem`
  - `am_sg_exec`
    - all special module states
    - in each case, check *pc* and *acc* afterwards
    - for commands that alter *mem*, check *mem* afterwards

test implementation strategy

statement coverage measured using the UNIX utility *tcov*

F.1.1.2 Test implementation

PGMGEN script: `absmach.script`

```
module
     am_

accprogs
     <s_init,s_acc,g_acc,s_pc,g_pc,s_mem,g_mem,sg_exec>

exceptions
     <addr,int>

globcod
{%
#include "system.h"
#include "absmach.h"

int cmd;
%}

cases

/*****exceptions*****/
```

```
/*addr*/
<s_init().s_pc(-1000), addr, dc, dc, dc>
<s_init().s_pc(-1), addr, dc, dc, dc>
<s_init().s_pc(AM_MEMSIZ), addr, dc, dc, dc>
<s_init().s_pc(2*AM_MEMSIZ), addr, dc, dc, dc>
<s_init().s_mem(-1000,0), addr, dc, dc, dc>
<s_init().s_mem(-1,0), addr, dc, dc, dc>
<s_init().s_mem(AM_MEMSIZ,0), addr, dc, dc, dc>
<s_init().s_mem(2*AM_MEMSIZ,0), addr, dc, dc, dc>
<s_init().g_mem(-1000), addr, dc, dc, dc>
<s_init().g_mem(-1), addr, dc, dc, dc>
<s_init().g_mem(AM_MEMSIZ), addr, dc, dc, dc>
<s_init().g_mem(2*AM_MEMSIZ), addr, dc, dc, dc>

/*int*/
<s_init().s_acc(-1000), int, dc, dc, dc>
<s_init().s_acc(-1), int, dc, dc, dc>
<s_init().s_acc(AM_MAXINT+1), int, dc, dc, dc>
<s_init().s_acc(2*AM_MAXINT), int, dc, dc, dc>
<s_init().s_mem(0,-1000), int, dc, dc, dc>
<s_init().s_mem(0,-1), int, dc, dc, dc>
<s_init().s_mem(0,AM_MAXINT+1), int, dc, dc, dc>
<s_init().s_mem(0,2*AM_MAXINT), int, dc, dc, dc>

/*****normal case*****/
/*s_acc and g_acc*/
<s_init(), noexc, g_acc(), 0, int>
<s_init().s_acc(0), noexc, g_acc(), 0, int>
<s_init().s_acc(AM_MAXINT/2), noexc, g_acc(), AM_MAXINT/2, int>
<s_init().s_acc(AM_MAXINT), noexc, g_acc(), AM_MAXINT, int>

/*s_pc and g_pc*/
<s_init(), noexc, g_pc(), 0, int>
<s_init().s_pc(0), noexc, g_pc(), 0, int>
<s_init().s_pc(AM_MEMSIZ/2), noexc, g_pc(), AM_MEMSIZ/2, int>
<s_init().s_pc(AM_MEMSIZ-1), noexc, g_pc(), AM_MEMSIZ-1, int>

/*s_mem and g_mem*/
<s_init(), noexc, g_mem(0), 0, int>
<s_init(), noexc, g_mem(AM_MEMSIZ/2), 0, int>
<s_init(), noexc, g_mem(AM_MEMSIZ-1), 0, int>
<s_init().s_mem(0,0), noexc, g_mem(0), 0, int>
<s_init().s_mem(0,AM_MAXINT/2), noexc, g_mem(0), AM_MAXINT/2, int>
<s_init().s_mem(0,AM_MAXINT), noexc, g_mem(0), AM_MAXINT, int>
<s_init().s_mem(AM_MEMSIZ/2,3), noexc, g_mem(AM_MEMSIZ/2), 3, int>
<s_init().s_mem(AM_MEMSIZ-1,3), noexc, g_mem(AM_MEMSIZ-1), 3, int>

/*sg_exec: run-time exceptions*/
```

```
<s_init.s_mem(0,10), noexc, sg_exec(), AM_OBJECTEXC, int>
<s_init.s_mem(0,AM_MAXINT), noexc, sg_exec(), AM_OBJECTEXC, int>

{% for (cmd = 0; cmd <= 7; cmd++) %}
     <s_init.s_pc(AM_MEMSIZ-1).s_mem(AM_MEMSIZ-1,cmd),
          noexc, sg_exec(), AM_NOOPEXC, int>

{% for (cmd = 0; cmd <= 6; cmd++) { %}
     <s_init.s_mem(0,cmd).s_mem(1,AM_MEMSIZ),
          noexc, sg_exec(), AM_ADDREXC, int>
     <s_init.s_mem(0,cmd).s_mem(1,2*AM_MEMSIZ),
          noexc, sg_exec(), AM_ADDREXC, int>
{% } %}

<s_init.s_mem(0,SY_ADD).s_mem(1,2).s_mem(2,1).s_acc(AM_MAXINT),
     noexc, sg_exec(), AM_ARITHEXC, int>
<s_init.s_mem(0,SY_ADD).s_mem(1,2).s_mem(2,AM_MAXINT).s_acc(1),
     noexc, sg_exec(), AM_ARITHEXC, int>
<s_init.s_mem(0,SY_ADD).s_mem(1,2).s_mem(2,AM_MAXINT).
     s_acc(AM_MAXINT), noexc, sg_exec(), AM_ARITHEXC, int>

<s_init.s_mem(0,SY_SUBTRACT).s_mem(1,2).s_mem(2,1).s_acc(0),
     noexc, sg_exec(), AM_ARITHEXC, int>
<s_init.s_mem(0,SY_SUBTRACT).s_mem(1,2).s_mem(2,AM_MAXINT).
     s_acc(AM_MAXINT-1), noexc, sg_exec(), AM_ARITHEXC, int>
<s_init.s_mem(0,SY_SUBTRACT).s_mem(1,2).s_mem(2,AM_MAXINT).
     s_acc(0), noexc, sg_exec(), AM_ARITHEXC, int>

/*sg_exec: AM_HALT*/
<s_init().s_mem(0,SY_HALT), noexc, sg_exec(), AM_HALT, int>
< , noexc, g_pc(), 0, int>
< , noexc, g_acc(), 0, int>
<s_init().s_pc(AM_MEMSIZ-1).s_mem(AM_MEMSIZ-1,SY_HALT),
     noexc, sg_exec(), AM_HALT, int>
< , noexc, g_pc(), AM_MEMSIZ-1, int>
< , noexc, g_acc(), 0, int>

/*sg_exec: AM_PRINT*/
<s_init().s_mem(0,SY_PRINT), noexc, sg_exec(), AM_PRINT, int>
< , noexc, g_pc(), 1, int>
< , noexc, g_acc(), 0, int>
<s_init().s_pc(AM_MEMSIZ-1).s_mem(AM_MEMSIZ-1,SY_PRINT),
     noexc, sg_exec(), AM_PRINT, int>
< , noexc, g_pc(), 0, int>
< , noexc, g_acc(), 0, int>

/*sg_exec: AM_NORMAL*/
/*maximal pc*/
```

```
{% for (cmd = 0; cmd <= 7; cmd++) { %}
     <s_init().s_pc(AM_MEMSIZ-2).s_mem(AM_MEMSIZ-2,cmd),
          noexc, sg_exec(), AM_NORMAL, int>
     < ,noexc, g_pc(), 0, int>
{% } %}

/*full range of addresses*/
{% for (cmd = 0; cmd <= 6; cmd++) { %}
     <s_init().s_pc(1).s_mem(1,cmd).s_mem(2,0),
          noexc, sg_exec(), AM_NORMAL, int>
     <s_init().s_mem(0,cmd).s_mem(1,AM_MEMSIZ/2),
          noexc, sg_exec(), AM_NORMAL, int>
     <s_init().s_mem(0,cmd).s_mem(1,AM_MEMSIZ-1),
          noexc, sg_exec(), AM_NORMAL, int>
{% } %}

/*load*/
<s_init().s_mem(0,SY_LOAD).s_mem(1,2).s_mem(2,7),
     noexc, sg_exec(), AM_NORMAL, int>
< , noexc, g_pc(), 2, int>
< , noexc, g_acc(), 7, int>

/*store*/
<s_init().s_mem(0,SY_STORE).s_mem(1,2).s_acc(7),
     noexc, sg_exec(), AM_NORMAL, int>
< , noexc, g_pc(), 2, int>
< , noexc, g_acc(), 7, int>
< , noexc, g_mem(2), 7, int>

/*add*/
<s_init().s_mem(0,SY_ADD).s_mem(1,2).s_mem(2,AM_MAXINT-7).s_acc(7),
     noexc, sg_exec(), AM_NORMAL, int>
< , noexc, g_pc(), 2, int>
< , noexc, g_acc(), AM_MAXINT, int>

/*subtract*/
<s_init().s_mem(0,SY_SUBTRACT).s_mem(1,2).s_mem(2,7).s_acc(7),
     noexc, sg_exec(), AM_NORMAL, int>
< , noexc, g_pc(), 2, int>
< , noexc, g_acc(), 0, int>

/*branch*/
<s_init().s_mem(0,SY_BRANCH).s_mem(1,7), noexc, sg_exec(), AM_NORMAL, int>
< , noexc, g_pc(), 7, int>
< , noexc, g_acc(), 0, int>

/*branchzero*/
<s_init().s_mem(0,SY_BRANCHZERO).s_mem(1,7),
```

```
     noexc, sg_exec(), AM_NORMAL, int>
< , noexc, g_pc(), 7, int>
< , noexc, g_acc(), 0, int>
<s_init().s_mem(0,SY_BRANCHZERO).s_mem(1,7).s_acc(1),
     noexc, sg_exec(), AM_NORMAL, int>
< , noexc, g_pc(), 2, int>
< , noexc, g_acc(), 1, int>

/*branchpos*/
<s_init().s_mem(0,SY_BRANCHPOS).s_mem(1,7),
     noexc, sg_exec(), AM_NORMAL, int>
< , noexc, g_pc(), 2, int>
< , noexc, g_acc(), 0, int>
<s_init().s_mem(0,SY_BRANCHPOS).s_mem(1,7).s_acc(1),
     noexc, sg_exec(), AM_NORMAL, int>
< , noexc, g_pc(), 7, int>
< , noexc, g_acc(), 1, int>

/*loadcon*/
<s_init().s_mem(0,SY_LOADCON).s_mem(1,7),
     noexc, sg_exec(), AM_NORMAL, int>
< , noexc, g_pc(), 2, int>
< , noexc, g_acc(), 7, int>
```

Interactive driver: `absmach_i.c`

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

#define QUIT 0
#define S_INIT 1
#define S_PC 2
#define G_PC 3
#define S_ACC 4
#define G_ACC 5
#define S_MEM 6
#define G_MEM 7
#define SG_EXEC 8
#define G_DUMP 9

#define BUFLEN 80

FILE *sy_excfilp = stderr;

int nextcall()
{
     int reply;
     char s[81];
```

```
    do {
        printf("\nEnter command:\n");
        printf("\t0:quit\n");
        printf("\t1:s_init\n");
        printf("\t2:s_pc\n");
        printf("\t3:g_pc\n");
        printf("\t4:s_acc\n");
        printf("\t5:g_acc\n");
        printf("\t6:s_mem\n");
        printf("\t7:g_mem\n");
        printf("\t8:sg_exec\n");
        printf("\t9:g_dump:");
        gets(s);
        if (sscanf(s,"%d",&reply) != 1)
            reply = -1; /*user error - stay in loop*/;
    } while (reply < 0 || reply > G_DUMP);
    return(reply);
}

int readint(msg)
char *msg;
{
    int reply,found;
    char s[BUFLEN];

    found = 0;
    while (!found) {
        printf(msg);
        gets(s);
        if (sscanf(s,"%d",&reply) == 1)
            found = 1;
    }
    return (reply);
}

main()
{
    int reply,i1,i2;

    while ((reply=nextcall()) != QUIT) {
        switch(reply) {
        case S_INIT:
            am_s_init();
            break;
        case S_PC:
            i1 = readint("Enter pc:");
            am_s_pc(i1);
```

```
                break;
            case G_PC:
                i1 = am_g_pc();
                printf("returns %d\n",i1);
                break;
            case S_ACC:
                i1 = readint("Enter acc:");
                am_s_acc(i1);
                break;
            case G_ACC:
                i1 = am_g_acc();
                printf("returns %d\n",i1);
                break;
            case S_MEM:
                i1 = readint("Enter addr:");
                i2 = readint("Enter val:");
                am_s_mem(i1,i2);
                break;
            case G_MEM:
                i1 = readint("Enter addr:");
                i2 = am_g_mem(i1);
                printf("returns %d\n",i2);
                break;
            case SG_EXEC:
                i1 = (int)am_sg_exec();
                printf("returns %d\n",i1);
                break;
            case G_DUMP:
                am_g_dump();
                break;
            }
        }
        return(0);
}
```

F.1.2 *exec* TP

F.1.2.1 Test plan

assumptions

test environment

test case selection strategy

test implementation strategy

considerations

exec testing performed during SHAM system testing

F.1.3 *load* TP and TI

F.1.3.1 Test plan

assumptions

- `AM_MAXINT` = 999
- `AM_MEMSIZ` = 100

test environment

- *sham* Coordinator used as driver
- stubs for *absmach* and *exec*, production code for *sham* and *token*
- input stored in files
- output saved in files, checked with delta testing
- directory structure:
  - `load/`
    - `input/` - test cases stored one per file
    - `exp/` - expected results of test case (same file name)
    - `act/` - actual results of test case (same file name)

test case selection strategy

- special values
  - module state
    - none
  - access routine parameters
    - input file for `ld_sg_load`:
      - every load-time exception for every instruction
      - every SHAM instruction at least once
        - interval rule for instructions with an operand
      - completely fill up memory
- test cases
  - load-time exceptions
    - `ldexc1`: all load-time exceptions except $NOMEMEXC$
    - `ldexc2`: $NOMEMEXC$
  - normal case
    - `instr`: every SHAM instruction
    - `fill`: completely fill up memory

test implementation strategy

- stubs print out name and parameters of access routines
- target `runtest` in `Makefile`
  - for each file f in `input/`
    - `bsham` f `>act/`f
    - `diff act/`f `exp/`f
- statement coverage measured using the UNIX utility *tcov*

F.1.3.2 Test implementation

absmach **stubs:** `absmach_s.c`

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

```
#include "absmach.h"

/*****module state*****/
static int mem[AM_MEMSIZ];

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

void am_s_init()
{
}

void am_s_mem(a,i)
int a,i;
{
     mem[a] = i;
}

int am_g_mem(a)
int a;
{
     return(mem[a]);
}
```

exec **stubs:** `exec_s.c`

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

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

void ex_s_init()
{
}

void ex_s_exec()
{
     int i;

     for (i = 0; i < AM_MEMSIZ; i++) {
          printf("%4d",am_g_mem(i));
          if (i % 10 == 9)
               printf("\n");
     }
}
```

Input file: `input/fill`

```
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
store 2
```

```
store 2
store 2
store 2
```

Expected output: `exp/fill`

```
   1    2    1    2    1    2    1    2    1    2
   1    2    1    2    1    2    1    2    1    2
   1    2    1    2    1    2    1    2    1    2
   1    2    1    2    1    2    1    2    1    2
   1    2    1    2    1    2    1    2    1    2
   1    2    1    2    1    2    1    2    1    2
   1    2    1    2    1    2    1    2    1    2
   1    2    1    2    1    2    1    2    1    2
   1    2    1    2    1    2    1    2    1    2
   1    2    1    2    1    2    1    2    1    2
```

Input file: `input/instr`

```
load 0
load 50
load 99
store 0
store 50
store 99
add 0
add 50
add 99
sub 0
sub 50
sub 99
br 0
br 50
br 99
brz 0
brz 50
brz 99
brp 0
brp 50
brp 99
loadcon 0
loadcon 500
loadcon 999
print
halt
```

Expected output: `exp/instr`

```
   0   0    0  50    0  99    1   0    1  50
   1  99    2   0    2  50    2  99    3   0
   3  50    3  99    4   0    4  50    4  99
   5   0    5  50    5  99    6   0    6  50
   6  99    7   0    7 500    7 999    8   9
   0   0    0   0    0   0    0   0    0   0
   0   0    0   0    0   0    0   0    0   0
   0   0    0   0    0   0    0   0    0   0
   0   0    0   0    0   0    0   0    0   0
   0   0    0   0    0   0    0   0    0   0
```

Input file: `input/ldexc1`

```
xxx
load
store
add
sub
br
brz
brp
loadcon
load x
load 100
load 200
store x
store 100
store 200
add x
add 100
add 200
sub x
sub 100
sub 200
br x
br 100
br 200
brz x
brz 100
brz 200
brp x
brp 100
brp 200
loadcon x
loadcon 1000
loadcon 2000
```

Expected output: `exp/ldexc1`

```
Load exception at 1. Blank line illegal
Load exception at 2. Illegal instruction: xxx
Load exception at 3. Operand missing
Load exception at 4. Operand missing
Load exception at 5. Operand missing
Load exception at 6. Operand missing
Load exception at 7. Operand missing
Load exception at 8. Operand missing
Load exception at 9. Operand missing
Load exception at 10. Operand missing
Load exception at 11. Illegal operand: x
Load exception at 12. Illegal operand: 100
Load exception at 13. Illegal operand: 200
Load exception at 14. Illegal operand: x
Load exception at 15. Illegal operand: 100
Load exception at 16. Illegal operand: 200
Load exception at 17. Illegal operand: x
Load exception at 18. Illegal operand: 100
Load exception at 19. Illegal operand: 200
Load exception at 20. Illegal operand: x
Load exception at 21. Illegal operand: 100
Load exception at 22. Illegal operand: 200
Load exception at 23. Illegal operand: x
Load exception at 24. Illegal operand: 100
Load exception at 25. Illegal operand: 200
Load exception at 26. Illegal operand: x
Load exception at 27. Illegal operand: 100
Load exception at 28. Illegal operand: 200
Load exception at 29. Illegal operand: x
Load exception at 30. Illegal operand: 100
Load exception at 31. Illegal operand: 200
Load exception at 32. Illegal operand: x
Load exception at 33. Illegal operand: 1000
Load exception at 34. Illegal operand: 2000
```

Input file: `input/ldexc2`

```
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
```

```
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
load 0
print
```

Expected output: `exp/ldexc2`

```
Load exception at 51. Program too large
```

F.1.4 *sham* TP

F.1.4.1 Test plan

assumptions

test environment

test case selection strategy

test implementation strategy

considerations
sham module testing performed during SHAM system testing

F.1.5 *token* TP and TI

F.1.5.1 Test plan

assumptions
`TK_MAXSTRLEN` ≥ 9
`TK_MAXIDLEN` ≥ 3
`TK_MAXINTLEN` ≥ 3

test environment
PGMGEN driver
no stubs

test case selection strategy
- special values
  - module state
    - number of tokens: $\{0, 1, 3\}$
    - types of token:
      - `TK_ID` - minimum and maximum length
      - `TK_INT` - minimum and maximum length
      - `TK_BADTOK` -
        - overlength `TK_ID` and `TK_INT` tokens
        - tokens that are "almost" legal `TK_ID` or `TK_INT` tokens
  - access routine parameters
    - `tk_s_str`(s):
      - interval rule on $|s|$: $[0,$ `TK_MAXSTRLEN`$]$
      - number of blanks before/after tokens: $\{0, 1, 3\}$
- test cases
  - exceptions
    - `tk_s_str`
      - generate `tk_maxlen`
    - `tk_sg_next`
      - generate `tk_end` for special number of tokens
  - normal

tk_s_str(s)
special values $|s|$
special values number of blanks
check using tk_sg_next
special number of tokens
check using tk_g_end
special token types
check token value and type using tk_sg_next

test implementation strategy

mkvaltyp(v, t): returns $\langle v, t\rangle$ as a structure of type tk_valtyp
cmp_valtyp(a, e) and prt_valtyp(a, e): cmp_ and prt_ functions for tk_valtyp
mkstring(n): returns a string of n *'s
must be able to return a string much longer than TK_MAXSTRLEN
mkid(n): returns a string with one alphabetic by $n - 1$ alphanumerics
mkint(n): returns a string of n digits
statement coverage measured using the UNIX utility *tcov*

F.1.5.2 Test implementation

PGMGEN script: token.script

```
module
     tk_

accprogs
     <s_init,s_str,g_end,sg_next>

exceptions
     <end,maxlen>

globcod
{%
#include "system.h"
#include "token.h"

/***tk_valtyp functions: creation; pgmgen cmp_, prt_*/

static tk_valtyp valtyp1,valtyp2;
static tk_valtyp *vtp = &valtyp1;

tk_valtyp *mkvaltyp(val,typ)
char *val;
tk_toktyp typ;
{
     strcpy(valtyp2.val,val);
     valtyp2.typ = typ;
     return(&valtyp2);
}
```

```
int cmp_valtyp(actvtp,expvtp)
tk_valtyp *actvtp,*expvtp;
{
     if (!strcmp(actvtp->val,expvtp->val))
          return(actvtp->typ == expvtp->typ);
     else
          return(0);
}

void prt_valtyp(actvtp,expvtp)
tk_valtyp *actvtp,*expvtp;
{
     printf("Expected value:<%s,%d>. Actual value:<%s,%d>\n",
          expvtp->val,expvtp->typ,actvtp->val,actvtp->typ);
}
static char s[TK_MAXSTRLEN+2];
char *mkstr(n)
int n;
{
     int i;

     for (i = 0; i < n; i++)
          s[i] = '*';
     s[n] = '\0';
     return(s);
}

char *mkid(n)
int n;
{
     int i;

     for (i = 0; i < n; i++)
          s[i] = 'a';
     s[n] = '\0';
     return(s);
}

char *mkint(n)
int n;
{
     int i;

     for (i = 0; i < n; i++)
          s[i] = '9';
     s[n] = '\0';
     return(s);
```

```
}
%}

cases

/*****exceptions*****/
/*maxlen*/
<s_init().s_str(mkstr(TK_MAXSTRLEN+1)), maxlen, dc, dc, dc>
<s_init().s_str(mkstr(2*TK_MAXSTRLEN)), maxlen, dc, dc, dc>

/*end*/
<s_init().sg_next(vtp), end, dc, dc, dc>
<s_init().s_str("").sg_next(vtp), end, dc, dc, dc>
<s_init().s_str("abc").sg_next(vtp).sg_next(vtp), end, dc, dc, dc>
<s_init().s_str("a b c").sg_next(vtp).sg_next(vtp).sg_next(vtp).
     sg_next(vtp), end, dc, dc, dc>

/*****normal case*****/
/*s_str - length of string*/
<s_init().s_str(""), noexc, dc, dc, dc>
<s_init().s_str(mkstr(TK_MAXSTRLEN)), noexc, dc, dc, dc>

/*s_str - blanks before and after token*/
<s_init().s_str("abc").sg_next(vtp),
     noexc, vtp, mkvaltyp("abc",TK_ID), valtyp>
<s_init().s_str(" abc ").sg_next(vtp),
     noexc, vtp, mkvaltyp("abc",TK_ID), valtyp>
<s_init().s_str(" abc").sg_next(vtp),
     noexc, vtp, mkvaltyp("abc",TK_ID), valtyp>
<s_init().s_str("abc ").sg_next(vtp),
     noexc, vtp, mkvaltyp("abc",TK_ID), valtyp>
<s_init().s_str("   abc   ").sg_next(vtp),
     noexc, vtp, mkvaltyp("abc",TK_ID), valtyp>
<s_init().s_str("abc def").sg_next(vtp),
     noexc, vtp, mkvaltyp("abc",TK_ID), valtyp>
<s_init().s_str("abc def").sg_next(vtp).sg_next(vtp),
     noexc, vtp, mkvaltyp("def",TK_ID), valtyp>
<s_init().s_str("abc   def").sg_next(vtp),
     noexc, vtp, mkvaltyp("abc",TK_ID), valtyp>
<s_init().s_str("abc   def").sg_next(vtp).sg_next(vtp),
     noexc, vtp, mkvaltyp("def",TK_ID), valtyp>

/*special number of tokens*/
<s_init(), noexc, g_end(), 1, bool>
<s_init().s_str(""), noexc, g_end(), 1, bool>
<s_init().s_str("abc"), noexc, g_end(), 0, bool>
<s_init().s_str("abc").sg_next(vtp), noexc, g_end(), 1, bool>
<s_init().s_str("a b c").sg_next(vtp).sg_next(vtp),
```

```
     noexc, g_end(), 0, bool>
<s_init().s_str("a b c").sg_next(vtp).sg_next(vtp).sg_next(vtp),
     noexc, g_end(), 1, bool>

/*special token types: TK_ID*/
<s_init().s_str("a").sg_next(vtp),
     noexc, vtp, mkvaltyp("a",TK_ID), valtyp>
<s_init().s_str(mkid(TK_MAXIDLEN)).sg_next(vtp),
     noexc, vtp, mkvaltyp(mkid(TK_MAXIDLEN),TK_ID), valtyp>

/*special token types: TK_INT*/
<s_init().s_str("1").sg_next(vtp),
     noexc, vtp, mkvaltyp("1",TK_INT), valtyp>
<s_init().s_str(mkint(TK_MAXINTLEN)).sg_next(vtp),
     noexc, vtp, mkvaltyp(mkint(TK_MAXINTLEN),TK_INT), valtyp>

/*special token types: TK_BADTOK - overlength*/
/*TK_ID*/
<s_init().s_str(mkid(TK_MAXIDLEN+1)).sg_next(vtp),
     noexc, vtp, mkvaltyp(mkid(TK_MAXIDLEN+1),TK_BADTOK), valtyp>
/*TK_INT*/
<s_init().s_str(mkint(TK_MAXINTLEN+1)).sg_next(vtp),
     noexc, vtp, mkvaltyp(mkint(TK_MAXINTLEN+1),TK_BADTOK), valtyp>

/*special token types: TK_BADTOK - almost legal id or int*/
/*bad characters*/
<s_init().s_str("!").sg_next(vtp),
     noexc, vtp, mkvaltyp("!",TK_BADTOK), valtyp>
/*TK_ID*/
<s_init().s_str("!bc").sg_next(vtp),
     noexc, vtp, mkvaltyp("!bc",TK_BADTOK), valtyp>
<s_init().s_str("a!c").sg_next(vtp),
     noexc, vtp, mkvaltyp("a!c",TK_BADTOK), valtyp>
<s_init().s_str("ab!").sg_next(vtp),
     noexc, vtp, mkvaltyp("ab!",TK_BADTOK), valtyp>
/*TK_INT*/
<s_init().s_str("!23").sg_next(vtp),
     noexc, vtp, mkvaltyp("!23",TK_BADTOK), valtyp>
<s_init().s_str("1!3").sg_next(vtp),
     noexc, vtp, mkvaltyp("1!3",TK_BADTOK), valtyp>
<s_init().s_str("12!").sg_next(vtp),
     noexc, vtp, mkvaltyp("12!",TK_BADTOK), valtyp>
```

Interactive driver: `token_i.c`

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

```
#define QUIT 0
#define S_INIT 1
#define S_STR 2
#define G_END 3
#define SG_NEXT 4
#define G_DUMP 5

FILE *sy_excfilp = stderr;

int nextcall()
{
     int reply;
     char s[81];

     do {
          printf("\nEnter call name:\n");
          printf("\t0:quit\n");
          printf("\t1:s_init\n");
          printf("\t2:s_str\n");
          printf("\t3:g_end\n");
          printf("\t4:sg_next\n");
          printf("\t5:g_dump: ");
          gets(s);
          if (sscanf(s,"%d",&reply) != 1)
               reply = -1; /* user error - stay in loop */
     } while (reply < 0 || reply > G_DUMP);

     return(reply);
}

main()
{
     int reply;
     char s[80];

     tk_valtyp valtyp;

     while ((reply=nextcall()) != QUIT) {
          switch(reply) {
          case S_INIT:
               tk_s_init();
               break;
          case S_STR:
               printf("Enter string:");
               gets(s);
               tk_s_str(s);
               break;
```

```
          case SG_NEXT:
               tk_sg_next(&valtyp);
               printf("val=%s!typ=%d\n",valtyp.val,valtyp.typ);
               break;
          case G_END:
               reply = tk_g_end();
               printf("returns!%d!\n",reply);
               break;
          case G_DUMP:
               tk_g_dump();
          }
     }
     return(0);
}
```

F.2 ISHAM Modules

F.2.1 *keybdin* TP and TI

F.2.1.1 Test plan

assumptions

test environment
- `keybdin_i` customized interactive driver
- no stubs

test case selection strategy
- special values
  - module state
    - none
  - access routine parameters
    - return value for `ki_sg_next`:
      - all valid ISHAM commands
      - at least one invalid ISHAM command
- test cases
  - enter special values one at a time

test implementation strategy
- `keybdin_i`:
  - repetitively wait for input command
  - if command is `'q'` then quit
  - else print command
- statement coverage measured using the UNIX utility *tcov*

F.2.1.2 Test implementation

Customized interactive driver: `keybdin_i.c`

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

main()
{
     char ch;

     /*initialize curses*/
     initscr();
     clear();
     refresh();

     ki_s_init();
     move(0,0);
     addstr("Enter character ('q' to quit).");
     move(1,0);
     addstr("Character entered: ");
     refresh();
     ch = ki_sg_next();
     while (ch != 'q') {
          move(1,strlen("Character entered: "));
          addch(ch);
          refresh();
          ch = ki_sg_next();
     }
     ki_s_end();

     /*terminate curses*/
     clear();
     refresh();
     endwin();
     return(0);
}
```

F.2.2 *scndr* TP and TI

F.2.2.1 Test plan

assumptions

`SY_MAXINT` ≥ 990

test environment

`scndr_pic` customized driver

stubs for *absmach*, production code for *scngeom* and *scnstr*

test case selection strategy

special values

module state
ISHAM screen, with the following values
$MEM(r, c) = 10 \times (10 \times r + c)$
$MEM(0, 0)$ and $MEM(9, 9)$ highlighted
$PC = 11$
$ACC = 222$
$PRT = 333$
$MSG =$ "123456789012345678901234567890"
access routine parameters
none
test cases
exceptions
no exception testing done
normal case
ISHAM screen

test implementation strategy
stubs for *absmach*
only get calls implemented
return values indicated above for ISHAM screen
`scndr_pic`: display ISHAM screen
wait until return is hit
clear screen and exit
during execution, no exceptions should be recorded in the file `SHAM.excfil`
statement coverage measured using the UNIX utility *tcov*

considerations
The normal-case testing of *scngeom* is included in the testing of *scndr*.

F.2.2.2 Test implementation

Customized driver: `scndr_pic.c`

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

FILE *sy_excfilp;

main()
{
     char buf[80];

     /*initialize exception file pointer*/
     sy_excfilp = fopen(SY_EXCFIL,"a");

     /*initialize curses*/
```

```
     initscr();

     /*initialize modules*/
     ss_s_init();
     sg_s_init();
     sd_s_init();

     /*create screen*/
     sd_s_clrscn();
     sd_s_con();
     sd_s_mem();
     sd_s_pc();
     sd_s_acc();
     sd_s_prt(333);
     sd_s_msg("123456789012345678901234567890");

     sd_s_hlt(0,1);
     sd_s_hlt(50,1);
     sd_s_hlt(50,0);
     sd_s_hlt(99,1);

     /*leave picture up until return*/
     gets(buf);

     sd_s_clrscn();
     ss_s_end();

     /*terminate curses*/
     endwin();

     /*close exception file*/
     fclose(sy_excfilp);

     return(0);
}
```

absmach **stubs:** `absmach_s.c`

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

/*access routines*/

int am_g_acc()
{
     return(222);
}
```

```
int am_g_pc()
{
     return(11);
}

int am_g_mem(a)
int a;
{
     return(10*a);
}
```

F.2.3 *scngeom* TP and TI

F.2.3.1 Test plan

assumptions

test environment
- PGMGEN driver
- no stubs

test case selection strategy
- special values
  - module state
    - none
  - access routine parameters
    - `sg_g_row`
      - all field names
      - interval rule for row and column numbers
    - `sg_g_col`, `sg_g_len`, and `sg_g_val`
      - one illegal field
- test cases
  - exceptions
    - `sg_g_row`, `sg_g_col`, `sg_g_len`, and `sg_g_val`
      - generate `sg_badfld`
  - normal
    - none

test implementation strategy
- statement coverage measured using the UNIX utility *tcov*
  - 100% statement coverage for exception-code

considerations
- Only exception testing is included.
- Normal-case testing is performed while testing *scndr*.

F.2.3.2 Test implementation

PGMGEN script: scngeom.script

```
module
     sg_

accprogs
     <s_init,g_legfld,g_row,g_col,g_len,g_val>

exceptions
     <badfld>

globcod
{%
#include "system.h"
#include "scngeom.h"

sg_fld fld;
int nam;

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

cases

/*****exceptions for g_row, g_col, g_len, and g_val only*****/
/*****normal case handled elsewhere*****/
/*g_row*/
{%
for (nam = 0; nam <= SG_MSGTTL; nam++) {
%}
     <s_init().FLD(fld,nam,-1,0).g_row(fld), badfld, dc, dc, dc>
     <s_init().FLD(fld,nam,-100,0).g_row(fld), badfld, dc, dc, dc>
     <s_init().FLD(fld,nam,0,-1).g_row(fld), badfld, dc, dc, dc>
     <s_init().FLD(fld,nam,0,-100).g_row(fld), badfld, dc, dc, dc>
{%
     if (nam == SG_MEM || nam == SG_MEMROWHDR) {
%}
          <s_init().FLD(fld,nam,10,0).g_row(fld), badfld, dc, dc, dc>
{%
     } else {
%}
          <s_init().FLD(fld,nam,1,0).g_row(fld), badfld, dc, dc, dc>
{%
     }
%}
     <s_init().FLD(fld,nam,100,0).g_row(fld), badfld, dc, dc, dc>
{%
```

```
     if (nam == SG_MEM || nam == SG_MEMCOLHDR) {
%}
          <s_init().FLD(fld,nam,0,10).g_row(fld), badfld, dc, dc, dc>
{%
     } else {
%}
          <s_init().FLD(fld,nam,0,1).g_row(fld), badfld, dc, dc, dc>
{%
     }
%}
     <s_init().FLD(fld,nam,0,100).g_row(fld), badfld, dc, dc, dc>
{%
}
%}

/*g_col, g_len, and g_val*/
<s_init().FLD(fld,SG_MEM,-1,0).g_col(fld), badfld, dc, dc, dc>
<s_init().FLD(fld,SG_MEM,-1,0).g_len(fld), badfld, dc, dc, dc>
<s_init().FLD(fld,SG_MEM,-1,0).g_val(fld), badfld, dc, dc, dc>

/*****g_legfld normal case*****/
{%
for (nam = 0; nam <= SG_MSGTTL; nam++) {
%}
     <s_init().FLD(fld,nam,-1,0), noexc, g_legfld(fld), 0, bool>
     <s_init().FLD(fld,nam,0,-1), noexc, g_legfld(fld), 0, bool>
     <s_init().FLD(fld,nam,0,0), noexc, g_legfld(fld), 1, bool>
{%
}
%}
<s_init().FLD(fld,SG_MEM,10,0), noexc, g_legfld(fld), 0, bool>
<s_init().FLD(fld,SG_MEM,0,10), noexc, g_legfld(fld), 0, bool>
<s_init().FLD(fld,SG_MEM,9,9), noexc, g_legfld(fld), 1, bool>
<s_init().FLD(fld,SG_MEMROWHDR,10,0), noexc, g_legfld(fld), 0, bool>
<s_init().FLD(fld,SG_MEMROWHDR,9,0), noexc, g_legfld(fld), 1, bool>
<s_init().FLD(fld,SG_MEMCOLHDR,0,10), noexc, g_legfld(fld), 0, bool>
<s_init().FLD(fld,SG_MEMCOLHDR,0,9), noexc, g_legfld(fld), 1, bool>
```

Interactive driver: scngeom_i.c

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

#define QUIT 0
#define S_INIT 1
#define G_LEGFLD 2
#define G_ROW 3
#define G_COL 4
```

```
#define G_LEN 5
#define G_VAL 6

#define BUFLEN 80

FILE *sy_excfilp = stderr;

int nextcall()
{
     int reply;
     char s[81];

     do {
          printf("\nEnter command:\n");
          printf("\t0:quit\n");
          printf("\t1:s_init\n");
          printf("\t2:g_legfld\n");
          printf("\t3:g_row\n");
          printf("\t4:g_col\n");
          printf("\t5:g_len\n");
          printf("\t6:g_val:");
          gets(s);
          if (sscanf(s,"%d",&reply) != 1)
               reply = -1; /*user error - stay in loop*/;
     } while (reply < 0 || reply > G_VAL);
     return(reply);
}

int readint(msg)
char *msg;
{
     int reply,found;
     char s[BUFLEN];

     found = 0;
     while (!found) {
          printf(msg);
          gets(s);
          if (sscanf(s,"%d",&reply) == 1)
               found = 1;
     }
     return (reply);
}

sg_fld readfld()
{
     sg_fld fld;
```

```
    fld.nam = (sg_fldnam)readint("Enter field name:");
    fld.row = readint("Enter row:");
    fld.col = readint("Enter column:");
    return(fld);
}

main()
{
    int reply;

    while ((reply=nextcall()) != QUIT) {
        switch(reply) {
        case S_INIT:
            sg_s_init();
            break;
        case G_LEGFLD:
            printf("returns %d\n",sg_g_legfld(readfld()));
            break;
        case G_ROW:
            printf("returns %d\n",sg_g_row(readfld()));
            break;
        case G_COL:
            printf("returns %d\n",sg_g_col(readfld()));
            break;
        case G_LEN:
            printf("returns %d\n",sg_g_len(readfld()));
            break;
        case G_VAL:
            printf("returns !%s!\n",sg_g_val(readfld()));
            break;
        }
    }
    return(0);
}
```

F.2.4 *scnstr* TP and TI

F.2.4.1 Test plan

assumptions

SS_NUMCOL ≥ 9
SS_NUMROW > 0

test environment

exception testing with PGMGEN driver
normal case testing performed with `scnstr_pic` customized driver
no stubs

test case selection strategy

special values
- module state
  - none
- access routine parameters
  - `ss_s_str`(r, c, s), `ss_s_hlt`(r, c, l, f), and `ss_s_cur`(r, c)
    - interval rule for r and c
  - `ss_s_str`(r, c, s)
    - interval rule for $|s|$
  - `ss_s_hlt`(r, c, l, f)
    - interval rule for l
    - turn highlighting on and off again

test cases
- exceptions
  - `ss_s_str`, `ss_s_hlt`, and `ss_s_cur`
    - generate `ss_row`, `ss_col`, and `ss_len`
- normal case
  - pattern consisting of
    - letters A, B, C, and D in four corners
    - letter V going down vertically in the middle
    - letter H going across horizontally in the middle
    - string YYYXXXYYY in center with the Y's highlighted
    - cursor in center of the screen

test implementation strategy

`scnstr_pic`:
- display pattern on the screen
- wait until return is hit
- clear screen and exit
- during execution, no exceptions should be recorded in the file `SHAM.excfil`

statement coverage measured using the UNIX utility *tcov*

F.2.4.2 Test implementation

PGMGEN script: `scnstr.script`

```
module
     ss_

accprogs
     <s_init,s_clrscn,s_str,s_hlt,s_cur,s_ref,s_end>

exceptions
     <row,col,len>

globcod
{%
#include "system.h"
#include "scnstr.h"
```

```
/*return an oversize string*/
char s[2*SS_NUMCOL+1];
char *str(l)
int l;
{
     int i;

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

cases

/*****exceptions only - normal case handled elsewhere*****/
/*s_str*/
<s_init().s_str(-100,0,""), row, dc, dc, dc>
<s_init().s_str(-1,0,""), row, dc, dc, dc>
<s_init().s_str(SS_NUMROW,0,""), row, dc, dc, dc>
<s_init().s_str(2*SS_NUMROW,0,""), row, dc, dc, dc>

<s_init().s_str(0,-100,""), col, dc, dc, dc>
<s_init().s_str(0,-1,""), col, dc, dc, dc>
<s_init().s_str(0,SS_NUMCOL,""), col, dc, dc, dc>
<s_init().s_str(0,2*SS_NUMCOL,""), col, dc, dc, dc>

<s_init().s_str(0,0,str(SS_NUMCOL+1)), len, dc, dc, dc>
<s_init().s_str(0,SS_NUMCOL-1,str(2)), len, dc, dc, dc>
<s_init().s_str(0,0,str(2*SS_NUMCOL)), len, dc, dc, dc>

/*s_hlt*/
<s_init().s_hlt(-100,0,0,0), row, dc, dc, dc>
<s_init().s_hlt(-1,0,0,0), row, dc, dc, dc>
<s_init().s_hlt(SS_NUMROW,0,0,0), row, dc, dc, dc>
<s_init().s_hlt(2*SS_NUMROW,0,0,0), row, dc, dc, dc>

<s_init().s_hlt(0,-100,0,0), col, dc, dc, dc>
<s_init().s_hlt(0,-1,0,0), col, dc, dc, dc>
<s_init().s_hlt(0,SS_NUMCOL,0,0), col, dc, dc, dc>
<s_init().s_hlt(0,2*SS_NUMCOL,0,0), col, dc, dc, dc>

<s_init().s_hlt(0,0,-100,0), len, dc, dc, dc>
<s_init().s_hlt(0,0,-1,0), len, dc, dc, dc>
<s_init().s_hlt(0,0,SS_NUMCOL+1,0), len, dc, dc, dc>
<s_init().s_hlt(0,SS_NUMCOL-1,2,0), len, dc, dc, dc>
```

```
<s_init().s_hlt(0,0,2*SS_NUMCOL,0), len, dc, dc, dc>

/*s_cur*/
<s_init().s_cur(-100,0), row, dc, dc, dc>
<s_init().s_cur(-1,0), row, dc, dc, dc>
<s_init().s_cur(SS_NUMROW,0), row, dc, dc, dc>
<s_init().s_cur(2*SS_NUMROW,0), row, dc, dc, dc>

<s_init().s_cur(0,-100), col, dc, dc, dc>
<s_init().s_cur(0,-1), col, dc, dc, dc>
<s_init().s_cur(0,SS_NUMCOL), col, dc, dc, dc>
<s_init().s_cur(0,2*SS_NUMCOL), col, dc, dc, dc>
```

Customized driver: `scnstr_pic.c`

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

FILE *sy_excfilp;

main()
{
     int i;
     char buf[SS_NUMCOL+1];

     /*initialize exception file pointer*/
     sy_excfilp = fopen(SY_EXCFIL,"a");

     /*initialize curses*/
     initscr();

     ss_s_init();
     ss_s_clrscn();

     ss_s_str(0,0,"A");
     ss_s_str(0,SS_NUMCOL-1,"B");
     ss_s_str(SS_NUMROW-1,SS_NUMCOL-1,"C");
     ss_s_str(SS_NUMROW-1,0,"D");

     for (i = 0; i < SS_NUMROW; i++)
          ss_s_str(i,SS_NUMCOL/2,"V");
     for (i = 0; i < SS_NUMCOL; i++)
          buf[i] = 'H';
     buf[SS_NUMCOL] = '\0';
     ss_s_str(SS_NUMROW/2,0,buf);

     ss_s_str(SS_NUMROW/2,SS_NUMCOL/2-4,"YYYXXXYYY");
```

```
    ss_s_hlt(SS_NUMROW/2,SS_NUMCOL/2-4,9,1);
    ss_s_hlt(SS_NUMROW/2,SS_NUMCOL/2-1,3,0);

    ss_s_cur(SS_NUMROW/2,SS_NUMCOL/2);
    ss_s_ref();

    /*leave picture up until return*/
    gets(buf);

    ss_s_clrscn();
    ss_s_ref();
    ss_s_end();

    /*terminate curses*/
    endwin();

    /*close exception file*/
    fclose(sy_excfilp);

    return(0);
}
```

F.3 System Testing

F.3.1 BSHAM system TP and TI

F.3.1.1 Test plan

assumptions

SY_MAXINT = 999
SY_MEMSIZ = 100

test environment

entire BSHAM system
input stored in files
output saved in files, checked with delta testing
directory structure:
- sham/
  - input/ - test cases stored one per file
  - exp/ - expected results of test case (same file name)
  - act/ - actual results of test case (same file name)
- sham/cmdlin.exp - expected results for command-line test cases
- sham/cmdlin.act - actual results for command-line test cases

test case selection strategy

special values
- command-line errors
  - each command-line error once

content of input file
    one load-time exception
    every run-time exception once
    SHAM instructions
        halt with pc = $\{0, \texttt{SY\_MEMSIZ}/2, \texttt{SY\_MEMSIZ} - 1\}$
        print with interval rule on content accumulator
test cases
    command-line errors
        hardcoded in `Makefile`
    load-time exceptions
        `ldexc`: one load-time exception
    run-time exceptions
        `addrexc`: $mem[pc] = ADD.object$, $mem[pc+1] = 100$
        `arithexc`: $acc = 500 + 500$
        `noopexc`: $pc = 99$, $mem[pc] = LOADCON.object$
        `objectexc`: $mem[pc] = 10$
    normal-case
        `halt[1-3]`: *HALT* instruction, check with *PRINT* instruction
        `print`: print special values
        `two+two,sum`: programs from Appendix of RS

test implementation strategy
    target `runtestb` in `Makefile`
        test cases for command-line errors
        for each file f in `input/`
            `bsham` f `>act/`f
            `diff act/`f `exp/`f
    statement coverage for *sham* and *exec* measured using the UNIX utility *tcov*
        100% coverage for statements not associated with ISHAM

F.3.1.2 Test implementation

Command-line test cases input commands

```
bsham >cmdlin.act
bsham foo >>cmdlin.act
```

Command-line test cases expected output: `cmdlin.exp`

```
Command line error. No file name specified
Command line error. Cannot open file: foo
```

Input file: `input/addrexc`

```
loadcon 100  % store bad address 100 as operand of add instruction
store 5
add 0
```

Expected output: `exp/addrexc`

```
Execution exception at 4. Illegal operand: 100
```

Input file: `input/arithexc`

```
loadcon 500
store 10
loadcon 500
add 10       % 500+500
```

Expected output: `exp/arithexc`

```
Execution exception at 6. Arithmetic overflow
```

Input file: `input/halt1`

```
halt
```

Expected output: `exp/halt1`

Input file: `input/halt2`

```
loadcon 8    % store "print, halt" in locations 50,51
store 50
loadcon 9
store 51
loadcon 5    % 5 should be printed once
br 50
```

Expected output: `exp/halt2`

```
5
```

Input file: `input/halt3`

```
loadcon 8    % store "print, halt" in locations 98,99
store 98
loadcon 9
store 99
loadcon 5    % 5 should be printed once
br 98
```

Expected output: `exp/halt3`

```
5
```

Input file: `input/ldexc`

```
loadcon 5
xxx
halt
```

Expected output: `exp/ldexc`

```
Load exception at 2. Illegal instruction: xxx
```

Input file: `input/noopexc`

```
loadcon 7    % store "loadcon" in location 99
store 99
br 99        % should cause NOOPERR
```

Expected output: `exp/noopexc`

```
Execution exception at 99. Operand not accessible
```

Input file: `input/objectexc`

```
loadcon 10   % store invalid command 10 in location 4
store 4
```

Expected output: `exp/objectexc`

```
Execution exception at 4. Illegal instruction: 10
```

Input file: `input/print`

```
print
loadcon 999
print
loadcon 500
print
loadcon 0
print
halt
```

Expected output: `exp/print`

```
0
999
500
0
```

Input file: `input/sum`

```
loadcon 5        % value of n
store 40         % location 40: value of n, decremented each iteration
loadcon 0
store 41         % location 41: value of sum
loadcon 1
store 42         % location 42: 1, used for decrementing
load 40
brz 28           % check if 0
add 41           % add to the sum
store 41
load 40          % subtract 1 from n
sub 42
store 40
br 14
load 41          % print value of sum
print
halt
```

Expected output: `exp/sum`

```
15
```

Input file: `input/two+two`

```
loadcon 2
store 20
loadcon 2
add 20
print
halt
```

Expected output: `exp/two+two`

```
4
```

F.3.2 ISHAM system TP and TI

F.3.2.1 Test plan

assumptions

SY_MAXINT = 999
SY_MEMSIZ = 100

test environment

entire ISHAM system
SHAM programs stored in files, ISHAM commands entered manually
output checked manually

directory `sham/input/` contains test cases, one per file

test case selection strategy
- special values
  - one load-time exception
  - one run-time exception
  - user commands
    - one invalid command
    - *STEP* through entire program
    - *EXIT* at beginning, in the middle, and at *HALT* instruction
- test cases
  - `ldexc`: one load-time exception
  - `arithexc`: one run-time exception
  - `two+two,sum`: programs from Appendix of RS

test implementation strategy
- test cases must be run manually
  - run four test cases described above
  - for `aritherr`, step through program until run-time exception
  - for `two+two` and `sum`, step through entire program
  - use `two+two` to test special user commands
- statement coverage for *sham* and *exec* measured using the UNIX utility *tcov*
  - 100% coverage for statements associated with ISHAM
  - *tcov* must be run manually after the test cases

considerations

The testing of the ISHAM version of *exec* is included in the testing of ISHAM.

F.4 Demonstration Modules

F.4.1 *stack* TP and TI

F.4.1.1 Test plan

assumptions

`PS_MAXSIZ` > 2

test environment
- PGMGEN driver
- no stubs

test case selection strategy
- special values
  - module state
    - interval rule on size of stack: $[0,$ `PS_MAXSIZ`$]$
  - access routine parameters
    - none
- test cases

for each of the special module state values,
call **ps_s_push**, **ps_s_pop**, **ps_g_top**, **ps_g_depth**
check exception behavior
after set calls, check get call values

test implementation strategy

`load`(n)

loads stack with $10, 20, \ldots, 10 \times n$

statement coverage measured using the UNIX utility *tcov*

F.4.1.2 Test implementation

PGMGEN script: `stack.script`

```
module
     ps_

accprogs
     <s_init,s_push,s_pop,g_top,g_depth>

exceptions
     <empty,full>

globcod
{%
#include "system.h"
#include "stack.h"

static void load(n)
int n;
{
     int i;

     ps_s_init();
     for (i = 0; i < n; i++)
          ps_s_push((i+1)*10);
}
%}

cases

/*empty stack*/
<load(0).s_push(10), noexc, g_top(), 10, int>
<load(0).s_push(10), noexc, g_depth(), 1, int>
<load(0).s_pop(), empty, dc, dc, dc>
<load(0).g_top(), empty, dc, dc, dc>
<load(0), noexc, g_depth(), 0, int>

/*partially full stack*/
```

```
<load(2).s_push(30), noexc, g_top(), 30, int>
<load(2).s_push(30), noexc, g_depth(), 3, int>
<load(2).s_pop(), noexc, g_top(), 10, int>
<load(2).s_pop(), noexc, g_depth(), 1, int>
<load(2), noexc, g_top(), 20, int>
<load(2), noexc, g_depth(), 2, int>

/*full stack*/
<load(PS_MAXSIZ).s_push(0), full, dc, dc, dc>
<load(PS_MAXSIZ).s_pop(), noexc, g_top(), (PS_MAXSIZ-1)*10, int>
<load(PS_MAXSIZ).s_pop(), noexc, g_depth(), PS_MAXSIZ-1, int>
<load(PS_MAXSIZ), noexc, g_top(), PS_MAXSIZ*10, int>
<load(PS_MAXSIZ), noexc, g_depth(), PS_MAXSIZ, int>
```

Interactive driver: `stack_i.c`

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

#define QUIT 0
#define S_INIT 1
#define S_PUSH 2
#define S_POP 3
#define G_TOP 4
#define G_DEPTH 5
#define G_DUMP 6

#define BUFLEN 80

FILE *sy_excfilp = stderr;

int nextcall()
{
     int reply;
     char s[81];

     do {
          printf("\nEnter command:\n");
          printf("\t0:quit\n");
          printf("\t1:s_init\n");
          printf("\t2:s_push\n");
          printf("\t3:s_pop\n");
          printf("\t4:g_top\n");
          printf("\t5:g_depth\n");
          printf("\t6:g_dump:");
          gets(s);
          if (sscanf(s,"%d",&reply) != 1)
               reply = -1; /*user error - stay in loop*/;
```

```
     } while (reply < 0 || reply > G_DUMP);
     return(reply);
}

int readint(msg)
char *msg;
{
     int reply,found;
     char s[BUFLEN];

     found = 0;
     while (!found) {
          printf(msg);
          gets(s);
          if (sscanf(s,"%d",&reply) == 1)
               found = 1;
     }
     return (reply);
}

main()
{
     int reply,i;

     while ((reply=nextcall()) != QUIT) {
          switch(reply) {
          case S_INIT:
               ps_s_init();
               break;
          case S_PUSH:
               i = readint("Enter element:");
               ps_s_push(i);
               break;
          case S_POP:
               ps_s_pop();
               break;
          case G_TOP:
               i = ps_g_top();
               printf("returns %d\n",i);
               break;
          case G_DEPTH:
               i = ps_g_depth();
               printf("returns %d\n",i);
               break;
          case G_DUMP:
               ps_g_dump();
               break;
          }
```

```
    }
    return(0);
}
```

F.4.2 *symtbl* TP and TI

F.4.2.1 Test plan

assumptions

ST_MAXSYMLEN $\geq$ length of ST_MAXSYMS $-$ 1 in string form
ST_MAXSYMS > 0

assumptions

PGMGEN driver
no stubs

test case selection strategy

- special values
  - module state
    - number of symbols in table: $\{0, 1, \texttt{ST\_MAXSYMS}/2, \texttt{ST\_MAXSYMS}\}$
    - symbol length: short, ST_MAXSYMLEN
  - access routine parameters
    - st_s_add: strings of length $\{0, \texttt{ST\_MAXSYMLEN} + 1, 2 \times \texttt{ST\_MAXSYMLEN}\}$
    - st_s_add, st_s_loc, st_g_loc, st_g_exsym: empty string
- test cases
  - exceptions
    - for each special module state
      - add overlength symbols
      - if the table is full
        - add a symbol not in the table
      - set and get locations for symbols not in table
      - add every symbol in the table
  - normal
    - check st_g_exsym for empty string in empty table
    - add the empty string, check and change its location
    - for each special module state
      - check table length
      - check that a very long symbol is not in table
      - for each i in $[0, \texttt{ST\_MAXSYMS} - 1]$
        - if i in $[0, \texttt{t\_siz} - 1]$
          - check $\texttt{t\_sym}(i)$ in table with correct location
          - check st_s_loc resets location
        - else
          - check $\texttt{t\_sym}(i)$ not in table

test implementation strategy

C functions to support iterating over the special module states,
viewed as a sequence:

void `t_init`: initialize to the first state
void `t_next`: load next state
int `t_end`: return *true* if no states remain

C functions to generate and check symbols in current state:
int `t_siz`: number of symbols in current state
char *`t_sym`(i): i-th symbol in current state
int `t_loc`(i): location of i-th symbol in current state
char *`t_mksym`(i, l): string consisting of i converted to ASCII, padded right with *'s to length l

statement coverage measured using the UNIX utility *tcov*

F.4.2.2 Test implementation

PGMGEN script: `symtbl.script`

```
module
     st_

accprogs
     <s_init,s_add,g_exsym,s_loc,g_loc,g_siz>

exceptions
     <exsym,maxlen,notexsym,full>

globcod
{%
#include "system.h"
#include "symtbl.h"

#define T_FILLCHAR '*'

static int i,cur;

static struct {
     int syms; /*number of symbols*/
     int symlen; /*symbol length*/
} tbl[] = {
     {0,0},
     {1,0},
     {1,ST_MAXSYMLEN},
     {ST_MAXSYMS/2,0},
     {ST_MAXSYMS/2,ST_MAXSYMLEN},
     {ST_MAXSYMS,0},
     {ST_MAXSYMS,ST_MAXSYMLEN},
     {-1,0} /*sentinel*/
};

static void t_init()
{
```

```
     cur = -1;
}

static char *t_mksym(i,len)
int i,len;
{
     static char buf[2*ST_MAXSYMLEN+1];
     int j;

     sprintf(buf,"%d",i); /*convert i to ASCII*/
     if (len > strlen(buf)) {
          for (j = strlen(buf); j < len; j++) /*pad right with '*'*/
               buf[j] = T_FILLCHAR;
          buf[len] = '\0'; /*add string terminator*/
     }
     return(buf);
}

static void t_next()
{
     int i;

     cur++;
     st_s_init();
     for (i = 0; i < tbl[cur].syms; i++)
          st_s_add(t_mksym(i,tbl[cur].symlen),10*i);
}

static int t_end()
{
     return(tbl[cur].syms == -1);
}

static int t_siz()
{
     return(tbl[cur].syms);
}

static char *t_sym(i)
int i;
{
     return(t_mksym(i,tbl[cur].symlen));
}

static int t_loc(i)
int i;
{
     return(10*i);
```

```
}
%}

cases

/*****exceptions*****/
{%
t_init();
t_next();
while (!t_end()) {
%}
     /*add overlength symbols*/
     <s_add(t_mksym(0,ST_MAXSYMLEN+1),0), maxlen, dc, dc, dc>
     <s_add(t_mksym(0,2*ST_MAXSYMLEN),0), maxlen, dc, dc, dc>

     /*if the table is full, add a symbol not in the table*/
{%
     if (t_siz() == ST_MAXSYMS)
%}
          <s_add("x",0), full, dc, dc, dc>

     /*set and get locations for symbols not in the table*/
     <s_loc(t_mksym(t_siz(),0),0), notexsym, dc, dc, dc>
     <s_loc("",0), notexsym, dc, dc, dc>
     <g_loc(t_mksym(t_siz(),0)), notexsym, dc, dc, dc>
     <g_loc(""), notexsym, dc, dc, dc>

     /*add every symbol in the table*/
{%
     for (i = 0; i < t_siz(); i++)
%}
          <s_add(t_sym(i),0), exsym, dc, dc, dc>
{%
     t_next();
}
%}

/*****normal case*****/
/*check g_exsym for empty string in empty table*/
<s_init(), noexc, g_exsym(""), 0, bool>
/*add the empty string, check and change its location*/
<s_init().s_add("",10), noexc, g_exsym(""), 1, bool>
< , noexc, g_loc(""), 10, int>
<s_loc("",20), noexc, g_loc(""), 20, int>

{%
t_init();
t_next();
```

```
while (!t_end()) {
%}
     /*check table length*/
     < , noexc, g_siz(), t_siz(), int>
     /*check that a very long symbol is not in table*/
     < , noexc, g_exsym(t_mksym(0,2*ST_MAXSYMLEN)), 0, bool>
{%
     for (i = 0; i < ST_MAXSYMS; i++) {
          if (i < t_siz()) {
%}
               /*check t_sym(i) in table with correct location*/
               < , noexc, g_exsym(t_sym(i)), 1, bool>
               < , noexc, g_loc(t_sym(i)), t_loc(i), int>
               /*check s_loc resets location*/
               <s_loc(t_sym(i),t_loc(-i)), noexc,
                    g_loc(t_sym(i)), t_loc(-i), int>
{%
          } else {
%}
               /*check t_sym(i) not in table*/
               < , noexc, g_exsym(t_sym(i)), 0, bool>
{%
          }
     }
     t_next();
}
%}
```

Interactive driver: `symtbl_i.c`

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

#define QUIT 0
#define S_INIT 1
#define S_ADD 2
#define G_EXSYM 3
#define S_LOC 4
#define G_LOC 5
#define G_SIZ 6
#define G_DUMP 7

#define BUFLEN 80

FILE *sy_excfilp = stderr;

int nextcall()
{
```

```
     int reply;
     char s[81];

     do {
          printf("\nEnter command:\n");
          printf("\t0:quit\n");
          printf("\t1:s_init\n");
          printf("\t2:s_add\n");
          printf("\t3:g_exsym\n");
          printf("\t4:s_loc\n");
          printf("\t5:g_loc\n");
          printf("\t6:g_siz\n");
          printf("\t7:g_dump:");
          gets(s);
          if (sscanf(s,"%d",&reply) != 1)
               reply = -1; /*user error - stay in loop*/;
     } while (reply < 0 || reply > G_DUMP);
     return(reply);
}

int readint(msg)
char *msg;
{
     int reply,found;
     char s[BUFLEN];

     found = 0;
     while (!found) {
          printf(msg);
          gets(s);
          if (sscanf(s,"%d",&reply) == 1)
               found = 1;
     }
     return (reply);
}

main()
{
     int reply,i;
     char s[80];

     while ((reply=nextcall()) != QUIT) {
          switch(reply) {
          case S_INIT:
               st_s_init();
               break;
          case S_ADD:
               printf("Enter sym:");
```

```
                gets(s);
                sscanf(s,"%s",s);
                i = readint("Enter loc:");
                st_s_add(s,i);
                break;
            case G_EXSYM:
                printf("Enter sym:");
                gets(s);
                sscanf(s,"%s",s);
                i = st_g_exsym(s);
                printf("returns %d\n",i);
                break;
            case S_LOC:
                printf("Enter sym:");
                gets(s);
                sscanf(s,"%s",s);
                i = readint("Enter loc:");
                st_s_loc(s,i);
                break;
            case G_LOC:
                printf("Enter sym:");
                gets(s);
                sscanf(s,"%s",s);
                i = st_g_loc(s);
                printf("returns %d\n",i);
                break;
            case G_SIZ:
                i = st_g_siz();
                printf("returns %d\n",i);
                break;
            case G_DUMP:
                st_g_dump();
                break;
            }
        }
        return(0);
}
```