Appendix A

Requirements Specifications

A.1 BSHAM Requirements Specification

A.1.1 Overview

A.1.1.1 System overview

SHAM, the Strooper-Hoffman Abstract Machine, provides an interpreter for a toy assembly language. The underlying machine has only two registers and performs arithmetic on unsigned decimal integers. Ten instructions are provided. SHAM operates in a load-and-go fashion; in response to a single user command, a file of assembler instructions is translated to object code, loaded into main memory, and executed.

There are two versions of the SHAM system. BSHAM, the batch version, is specified in this document. ISHAM, the interactive version, is specified separately.

A.1.1.2 Hardware and software environment

BSHAM runs on Sun/3 and Sun/4 workstations running SunOS. It is implemented in the C programming language and requires the UNIX/C standard libraries [55].

A.1.1.3 Notation

All identifiers are shown in *italics*. The names of constants and abbreviations are all uppercase. The others are all lowercase, except for types, whose names end in 'T'.

A.1.1.4 Document overview

The details of BSHAM operation are presented in the sections below. Section A.1.2 declares the environment variables. Section A.1.3 describes how to invoke BSHAM from the UNIX shell and contains the finite state machines (FSMs) that specify how BSHAM loads and executes source programs. Sections A.1.4–A.1.6 declare a collection of constants, types, and functions used throughout this document. Section A.1.7 lists the changes to BSHAM likely to be requested in the future. Section A.1.8 contains two sample BSHAM programs; Section A.1.9 contains tables that specify the details of the BSHAM syntax, semantics, and exceptions.

A.1.2 Environment variables

A.1.2.1 Input variables

$srcfil : string$
The file name passed on the command line.

A.1.2.2 Output variables

$stdout : string$
UNIX stdout.

A.1.3 State machine

BSHAM behavior is specified using two FSMs: one for each of the load and execution phases. The load-phase FSM reads the source program a line at a time, and loads the object code version into BSHAM's main memory. Exception messages are issued as needed. If the load phase is exception-free, then the execution-phase FSM begins running. It continues until a *HALT* instruction is reached, or an exception occurs.

A.1.3.1 Command-line invocation

BSHAM is invoked by typing

`bsham` *srcfil*

on the command line. Input is read from *srcfil* and output is written to *stdout*.
If the *srcfil* argument is not present, $excmsg(NOFILEXC, 0, \texttt{""})$ is written to *stdout*. If *srcfil* is unreadable (or does not exist)

$$excmsg(FILSYSEXC, 0, srcfil)$$

is written to *stdout*. If there are any command-line exceptions, BSHAM execution terminates.

Inputs
Each input is a line from *srcfil*, read in the order it appears in *srcfil*.
Outputs
Normal-case output and exception messages are written to *stdout*.
States
mem : *sequence* $[0..MEMSIZ - 1]$ *of* $shamintegerT$
Initial state
Every element of mem is set to 0.
Transitions and outputs
For line L, with line number n:
if the Load-phase Exc. Table (Table A.3) specifies an exception then
write the specified message to *stdout*
else
if no previous line had an exception then
if there is room in mem then
load the object code form of L into mem
else
write $excmsg(NOMEMEXC, n, "")$ to *stdout*

Figure A.1 Load-phase FSM

A.1.3.2 Load phase

The instructions and their arguments are shown in Table A.1. The first column contains the instruction mnemonic used in this document. Column two contains the string that must be used in source files read by BSHAM. Column three contains the object-code form generated by BSHAM. The last column shows the type of the instruction operand, if any. For an instruction with mnemonic I, $I.source$ and $I.object$ refer to I's source code string and object code integer, respectively.

At load time, the contents of *srcfil* are scanned a line at a time, converted to object code form, and loaded into main memory. Each line in *srcfil* must contain exactly one BSHAM instruction. Input lines must not exceed $MAXLINLEN$ characters—BSHAM behavior is unpredictable on longer lines. On each input line, tokens must be separated by one or more blanks. Object code instructions are loaded contiguously, beginning at address 0. Instructions without operands occupy a single memory location. Instructions with an operand occupy two consecutive memory locations; the instruction code is in the first location and the operand in the second. The load actions are described in detail in the FSM in Figure A.1. If there are any load exceptions, BSHAM execution terminates at the end of the load phase.

Inputs
None.
Outputs
Normal-case output and exception messages are written to *stdout*.
States
$mem : sequence\ [0..MEMSIZ - 1]\ of\ shamintegerT$
$acc : shamintegerT$
$pc : shamaddrT$
Initial state
$mem, acc, pc :=$ (the final value from the load phase FSM), $0, 0$
Transitions and outputs
for the instruction beginning at $mem[pc]$:
if the Execution-phase Exc. Table (Table A.4) specifies an exception then
write the specified message to *stdout*
terminate SHAM
else if $mem[pc] = HALT.object$ then
terminate SHAM
else
if $mem[pc] = PRINT.object$ then
write to $stdout : acc \mid\mid$ newline
modify *mem*, *acc*, and *pc* as shown in the Language Semantics Table

Figure A.2 BSHAM execution-phase FSM

A.1.3.3 Execution phase

The execution phase is based on Table A.2, the BSHAM Language Semantics table. This table specifies the effect of each exception-free BSHAM instruction on the values of *mem*, *acc*, and *pc*. The FSM itself is straightforward (see Figure A.2). The execution phase consumes no input; all the required information has already been loaded into *mem*, *pc*, and *acc*. The FSM executes the instructions in $mem[pc]$ until an exception occurs or $mem[pc] = HALT.object$.

A.1.4 Constants

Name	Value
MAXLINLEN	100
MAXINT	999
MEMSIZ	100

A.1.5 Types

$shamaddrT = [0..MEMSIZ - 1]$
$shamintegerT = [0..MAXINT]$

$$\begin{aligned}
sourceT = \{&LOAD.source, STORE.source, ADD.source, SUBTRACT.source,\\
&BRANCH.source, BRANCHZERO.source, BRANCHPOS.source,\\
&LOADCON.source, PRINT.source, HALT.source\}
\end{aligned}$$
$op0sourceT = \{HALT.source, PRINT.source\}$
$op1sourceT = sourceT - op0sourceT$

$$\begin{aligned}
objectT = \{&LOAD.object, STORE.object, ADD.object, SUBTRACT.object,\\
&BRANCH.object, BRANCHZERO.object, BRANCHPOS.object,\\
&LOADCON.object, PRINT.object, HALT.object\}
\end{aligned}$$
$op0objectT = \{HALT.object, PRINT.object\}$
$op1objectT = objectT - op0objectT$

$$\begin{aligned}
excidT = \{&FILSYSEXC, NOFILEXC,\\
&BLANKLINEXC, MISSINGOPEXC, NOMEMEXC,\\
&\quad OPFMTEXC, SOURCEEXC,\\
&ADDREXC, ARITHEXC, NOOPEXC, OBJECTEXC\}
\end{aligned}$$

A.1.6 Functions

$excmsg : excidT \times integer \times string \rightarrow string$

if id **is**	**then** $excmsg(id, loc, tok)$ **is**
Command-line messages	
$FILSYSEXC$	`Command line error. Cannot open file:` tok
$NOFILEXC$	`Command line error. No file name specified`
Load-phase messages	
$BLANKLINEXC$	`Load exception at` loc`. Blank line illegal`
$MISSINGOPEXC$	`Load exception at` loc`. Operand missing`
$NOMEMEXC$	`Load exception at` loc`. Program too large`
$OPFMTEXC$	`Load exception at` loc`. Illegal operand:` tok
$SOURCEEXC$	`Load exception at` loc`. Illegal instruction:` tok
Execution-phase messages	
$ADDREXC$	`Execution exception at` loc`. Illegal operand:` tok
$ARITHEXC$	`Execution exception at` loc`. Arithmetic overflow`
$NOOPEXC$	`Execution exception at` loc`. Operand not accessible`
$OBJECTEXC$	`Execution exception at` loc`. Illegal instruction:` tok

A.1.7 Expected changes

Input/output format

- Command-line parameters besides $srcfil$.
- Different input format: new tokens, delimiters, and instruction formats.
- Handle overlength lines robustly.

Abstract machine

- Change in word size, number of words in main memory.
- New or extended data types, especially signed integers.
- More registers, e.g., index registers.

- More or different SHAM instructions.
- More addressing modes.
- Symbolic data and branch addresses.

Platform

- Different operating system: other UNIX platforms or MS-DOS.

Exception handling

- Limits on the number of exceptions reported or instructions executed.
- Changes in the conditions defining exceptions and in the message text.

A.1.8 Sample programs

Calculate 2 + 2 and display the result

```
loadcon 2
store 8
add 8
print
halt
```

Calculate $\Sigma_{i=0}^{n} i$ and display the result

```
loadcon 5          initial 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
```

A.1.9 Tables

Table A.1 Language syntax table

Mnemonic	*I.source*	*I.object*	**Operand type**
	Memory access		
LOAD	`load`	0	*shamaddrT*
STORE	`store`	1	*shamaddrT*
	Arithmetic		
ADD	`add`	2	*shamaddrT*
SUBTRACT	`sub`	3	*shamaddrT*
	Branch		
BRANCH	`br`	4	*shamaddrT*
BRANCHZERO	`brz`	5	*shamaddrT*
BRANCHPOS	`brp`	6	*shamaddrT*
	Miscellaneous		
LOADCON	`loadcon`	7	*shamintegerT*
PRINT	`print`	8	
HALT	`halt`	9	

Table A.2 Language semantics table ($op = mem[pc+1]$)

Instruction at $mem[pc]$	**Effect on** mem**,** acc**, and** pc
Memory access	
LOAD.object	$acc, pc := mem[op], (pc+2) \bmod MEMSIZ$
STORE.object	$mem[op], pc := acc, (pc+2) \bmod MEMSIZ$
Arithmetic	
ADD.object	$acc, pc := acc + mem[op], (pc+2) \bmod MEMSIZ$
SUBTRACT.object	$acc, pc := acc - mem[op], (pc+2) \bmod MEMSIZ$
Branch	
BRANCH.object	$pc := op$
BRANCHZERO.object	$pc := (acc = 0 \Rightarrow op$ $\mid acc > 0 \Rightarrow (pc+2) \bmod MEMSIZ)$
BRANCHPOS.object	$pc := (acc > 0 \Rightarrow op$ $\mid acc = 0 \Rightarrow (pc+2) \bmod MEMSIZ)$
Miscellaneous	
LOADCON.object	$acc, pc := op, (pc+2) \bmod MEMSIZ$
PRINT.object	$pc := (pc+1) \bmod MEMSIZ$
HALT.object	no change to acc, pc, mem

Table A.3 Load-phase exception table

Let L be the current line, with line number n (numbered one-relative). Let $T_1, T_2, \ldots, T_K$ be the tokens in L.

Condition	Message
$K = 0$ (L is blank)	$excmsg(BLANKLINEEXC, n, "")$
$K > 0$	
$T_1 \in op0sourceT$	**Normal case**
$T_1 \in op1sourceT$	
$K = 1$	$excmsg(MISSINGOPEXC, n, "")$
$K > 1$	
$T_1 = LOADCON.source$	
$T_2 \in shamintegerT$	**Normal case**
$T_2 \notin shamintegerT$	$excmsg(OPFMTEXC, n, T_2)$
$T_1 \neq LOADCON.source$	
$T_2 \in shamaddrT$	**Normal case**
$T_2 \notin shamaddrT$	$excmsg(OPFMTEXC, n, T_2)$
$T_1 \notin sourceT$	$excmsg(SOURCEEXC, n, T_1)$

Table A.4 Execution-phase exception table

Let $i = mem[pc]$ and $op = mem[pc + 1]$

Condition	Message
$i \in op0objectT$	**Normal case**
$i \in op1objectT$	
$pc \in [0..MEMSIZ - 2]$	
$i = LOADCON.object$	**Normal case**
$i \neq LOADCON.object$	
$op \in shamaddrT$	
$i = ADD.object$	
$acc + mem[op] \in shamintegerT$	**Normal case**
$acc + mem[op] \notin shamintegerT$	$excmsg(ARITHEXC, pc, "")$
$i = SUBTRACT.object$	
$acc - mem[op] \in shamintegerT$	**Normal case**
$acc - mem[op] \notin shamintegerT$	$excmsg(ARITHEXC, pc, "")$
$true$	**Normal case**
$op \notin shamaddrT$	$excmsg(ADDREXC, pc, op)$
$pc = MEMSIZ - 1$	$excmsg(NOOPEXC, pc, "")$
$i \notin objectT$	$excmsg(OBJECTEXC, pc, i)$

A.2 ISHAM Requirements Specification

A.2.1 Overview

A.2.1.1 System overview

This document specifies the behavior of ISHAM, the interactive version of SHAM. The ISHAM and BSHAM load phases are identical, as are the language syntax and semantics, but the execution phases differ in two ways. In ISHAM:

1. Object code execution is "single-stepped" under user control.
2. Output is through a formatted screen, with main memory and the registers displayed, and updated after each instruction execution.

A.2.1.2 Hardware and software environment

The *curses* function library is required to perform output to the terminal screen.

A.2.1.3 Notation

Nothing is added to the BSHAM Requirements Specification.

A.2.1.4 Document overview

Because ISHAM and BSHAM have much in common, this document is written as an addendum to the BSHAM Requirements Specification, describing only the differences between ISHAM and BSHAM. Section A.2.2 declares the new environment variables: *stdin* to model keyboard input and *scn* to model the terminal screen. A detailed format is provided to precisely describe screen updates. Section A.2.3 specifies the ISHAM execution-phase FSM; the BSHAM load-phase FSM is unchanged. Sections A.2.4–A.2.6 declare the constants, types, and functions used throughout this document. There are two new constants and no new types or functions. Section A.2.7 lists the changes to ISHAM likely to be requested in the future.

A.2.2 Environment variables

A.2.2.1 Input variables

stdin : *string*
UNIX standard input

A.2.2.2 Output variables

scn : *sequence* [24][80] *of char*
$scn[r][c]$ is the character at screen row r and column c,
with numbering zero-relative and beginning at the upper-left corner.

```
012345678901234567890123456789012345678901234567890123456789012345678901
************************************************************************
                                 SHAM


                 0     1     2     3     4     5     6     7     8     9

Main       0   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM
memory:   10   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM
          20   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM
          30   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM
          40   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM
          50   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM
          60   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM
          70   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM
          80   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM
          90   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM   MEM


                           Program counter:  PC
                               Accumulator: ACC
                        Last value printed: PRT


Enter command: 's' to single step; 'e' to exit
Message: MSG------------------------------------------------------------
************************************************************************
```

Figure A.3 Screen format

We divide *scn*, a 24-by-80 array, into parts, as shown in Figure A.3. The non-blank areas are divided into *screen fields*, either *fixed* or *varying*. The fixed fields are written when ISHAM execution begins and remain unchanged while ISHAM is running. The varying fields may change repeatedly during ISHAM execution. Each varying field has an identifier: *MEM*, *PC*, *ACC*, *PRT*, or *MSG*. The extent of each varying field on the screen is the character positions occupied by the field identifier, and the trailing `-`s if present. When a *MEM*, *PC*, *ACC*, or *PRT* value is shorter than the extent shown, it is right-justified and padded left with blanks; *MSG* values are left-justified and padded right with blanks. Because the *MEM* field occurs 100 times on the screen, a particular *MEM* occurrence is indicated by row and column subscripts, numbered zero-relative, top-down, and left-to-right. For example, $MEM[9,0]$ is the leftmost and lowest occurrence.

A.2.3 State machine

A.2.3.1 Command-line invocation

ISHAM is invoked by typing

`isham` *srcfil*

on the command line. Input is read from *srcfil* and *stdin*, and output is written to *stdout* and *scn*.

If the *srcfil* argument is not present,

$$excmsg(NOFILEXC, 0, \texttt{""})$$

is written to *stdout*. If *srcfil* is unreadable (or does not exist),

$$excmsg(FILSYSEXC, 0, srcfil)$$

is written to *stdout*. If there are any command-line exceptions, ISHAM execution terminates.

A.2.3.2 Load phase

Same as in the BSHAM Requirements Specification.

A.2.3.3 Execution phase

The ISHAM execution-phase behavior is defined by the FSM in Figure A.4.

A.2.4 Constants

Name	**Value**
EXIT	`‘e’`
STEP	`‘s’`
CMDERRMSG	`"Illegal keyboard entry:  type ’s’ or ’e’"`
HALTMSG	`"HALT instruction reached"`

A.2.5 Types

Same as in the BSHAM Requirements Specification.

A.2.6 Functions

Same as in the BSHAM Requirements Specification.

Inputs

Keystrokes from $stdin$.

Outputs

All outputs are to scn and its fields.

States

Same as for the load phase FSM.

Initial state

$mem, acc, pc :=$ (the final value from the load phase FSM), $0, 0$

Transitions and outputs

For each character, c, from $stdin$
if $c = EXIT$ then
    clear scn
    halt ISHAM execution
else if $c = STEP$ then
    if the BSHAM Execution-phase Exception Table specifies
    an exception for $mem[pc]$ then
        $MSG :=$ the specified message
    else if $mem[pc] = HALT.object$ then
        $MSG := HALTMSG$
    else
        $MSG :=$ ""
        if $mem[pc] = PRINT.object$ then
            $PRT := acc$
        modify mem, acc, pc, as per the BSHAM Language Semantics Table
else
    $MSG := CMDERRMSG$

Notes on screen updating:

- Initially and between transitions, ensure that:
    1. The fixed fields shown in the ISHAM screen format are displayed.
    2. MEM, PC, and ACC are such that
       $(\forall r, c \in [0..9])(MEM[r, c] = mem[10 \times r + c]) \ \wedge \ ACC = acc \ \wedge PC = pc$
    3. $MEM[pc/10, pc \ mod \ 10]$ is displayed in inverse video.
- Initially the MSG and PRT fields are blank

Figure A.4 ISHAM execution-phase FSM

A.2.7 Expected changes

1. The field positions and the contents of the fixed fields will change.
2. $MEMSIZ$ will exceed 100 and vertical scrolling will be supported.
3. Different forms of stepping through the instructions will be supported, such as executing a specified number of instructions or executing until a specified instruction is reached.