EXPERIMENTAL operating system

SYSTEM PROGRAMMER'S LANGUAGE (SPL)

Download as PDF

Introduction

SPL or *System Programmer's Language* is an untyped programming language designed for implementation of an operating system on XSM *(eXperimental String Machine)* architecture. The language is minimalistic and consists only of basic constructs required for the implementation. Programming using SPL requires a basic understanding of the underlying XSM architecture and operating system concepts.

top ↑

Lexical Elements

Comments and White Spaces

SPL allows only single line comments. Comments start with the character sequence // and stop at the end of the line. White spaces in the program including tabs, newline and horizontal spaces are ignored.

top ↑

Keywords

The following are the reserved words in SPL and it cannot be used as identifiers.

```
alias     else      if     store   while
define    endif     ireturn break continue
do        endwhile  load   then    read
print     breakpoint halt  inline
```

top ↑

Operators and Delimiters

The following are the operators and delimiters in SPL

```
( )   ;    [   ]   /   *   +   -  %
> <  >=  <=  !=  ==  =  &&  ||  !
```

top ↑

Registers

SPL allows the use of 30 registers for various operations. (`R0-R7, S0 - S15, BP, SP, IP, PTBR, PTLR, EFR`)

top ↑

Identifiers

Identifiers are used as symbolic names for constants and aliases for registers. Identifiers should start with an alphabet but may contain alphabets, digits and/or underscore (_). No other special characters are allowed in identifiers.

Examples: `var1`, `new_page`
Invalid identifiers include `9blocks` , `$n` etc.

top ↑

Literals

Integer and String literals are permitted in SPL. An integer literal is a sequence of digits representing an integer. Negative integers are represented with a negative sign preceding the sequence of digits.
A string literal is a sequence of characters which are enclosed within double quotes (" "). e.g. `"alice"`

top ↑

Register Set

SPL doesn't allow the use of declared variables. Instead a fixed set of registers is provided. The register set in SPL contains 30 registers. There is a direct mapping between these registers and the machine registers in XSM.

Name	Register

Program Registers	R0 - R7
Kernel Registers	S0 - S15
Base Pointer	BP
Stack Pointer	SP
Instruction Pointer	IP
Page Table Base Register	PTBR
Page Table Length Register	PTLR
Exception Flag Register	EFR

top ↑

Aliasing

Any register can be referred to by using a different name. A name is assigned to a particular register using the alias keyword. Each register can be assigned to only one alias at any particular point of time. However, a register can be reassigned to a different alias at a later point. Aliasing can also be done inside the if and while block. However, an alias defined within the if and while blocks will only be valid within the block. No two registers can have the same alias name simultaneously.

top ↑

Constants

Symbolic names can be assigned to values using the define keyword. Unlike aliasing, two or more names can be assigned to the same value. A constant can only be defined once in a program.

top ↑

Predefined Constants

SPL provides a set of predefined constants. These predefined constants can be assigned to different values explicitly by the user using define keyword. These constants are mostly starting addresses of various OS components in the memory. The predefined set of constants provided in SPL are

Name	Default Value
SCRATCHPAD	512
PAGE_TABLE	1024
MEM_LIST	1280
FILE_TABLE	1344
READY_LIST	1536

FAT	2560
DISK_LIST	3072
EX_HANDLER	3584
T_INTERRUPT	4608
INTERRUPT	5632
USER_PROG	12800

top ↑

Expressions

An expression specifies the computation of a value by applying operators to operands. SPL supports arithmetic and logical expressions.

top ↑

Arithmetic Expressions

Registers, constants, and 2 or more arithmetic expressions connected using arithmetic operators are categorized as arithmetic expressions. SPL provides five arithmetic operators, viz., +, -, *, / (Integer Division) and % (Modulo operator) through which arithmetic expressions may be combined. Expression syntax and semantics are similar to standard practice in programming languages and normal rules of precedence, associativity and paranthesization hold.

Examples:

```
(5*x) + 3
10 % 4
```

top ↑

Logical Expressions

Logical expressions may be formed by combining arithmetic expressions using relational operators. The relational operators supported by SPL are `>, <, >=, <=, !=, ==`

Standard meanings apply to these operators. A relational operator will take in two arguments and return 1 if the relation is valid and 0 otherwise.

The relational operators can also be applied to strings. `<, >, <=, >=` compares two strings lexicographically. `!=` and == checks for equality in the case of strings. e.g.:

```
"adam" < "apple" // This returns 1
"hansel" == "gretel" // This returns 0
```

Logical expressions themselves may be combined using logical operators, && (logical and) , || (logical or) and ! (not).

top ↑

Addressing Expressions

Memory of the machine can be directly accessed in an SPL program. A word in the memory is accessed by specifying the addressing element, i.e. memory location within [ ]. This corresponds to the value stored in the given address. An arithmetic expression or an addressing expression can be used to specify the address.
Examples of addressing expressions:
`[1024], [S3], [S5+[S7]+128], [FAT + S2]` etc.

top ↑

Statements

Statements control the execution of the program. All statements in SPL are terminated with a semicolon ;

top ↑

Define Statement

Define statement is used to define a symbolic name for a value. Define statements should be used before any other statement in an SPL program. The keyword define is used to associate a literal to a symbolic name.

```
    define constant name value;
```

```
  define DISK_BLOCK 437;
```

top ↑

Alias Statement

An alias statement is used to associate a register with a name. Alias statements can be used anywhere in the program.

```
    alias alias_name register_name ;
```

```
  alias counter S0;
```

top ↑

Breakpoint Statement

The Breakpoint statement is used to debug the program. The program when run in debug mode pauses the execution at this instruction.

```
    breakpoint;
```

This instruction translates to BRKP machine instruction.

top ↑

Assignment Statement

The SPL assignment statement assigns the value of an expression or value stored in a memory address to a register or a memory address. = is the assignment operator used in SPL. The operand on the right hand side of the operator is assigned to the left hand side. The general syntax is as follows

```
    Register / Alias / [Address] = Register / Number /
String / Expression / [Address] ;
```

```
S0 = S2 * 10 + 5;
counter = counter + 1;
[PTBR + 3] = [1024] + 10;
S1 = "hello world";
```

top ↑

If Statment

If statements specify the conditional execution of two branches according to the value of a logical expression. If the expression evaluates to 1, the if branch is executed, otherwise the else branch is executed. The else part is optional. The general syntax is as follows

```
if (logical expression) then
   statements;
else
   statements;
endif;
```

top ↑

While Statement

While statement iteratively executes a set of statements based on a condition. The condition is defined using a logical expression. The statements are iteratively executed as long as the condition is true.

```
while (logical expression) do
   statements;
endwhile;
```

top ↑

Break Statement

Break statement is a statement which is used in a while loop block. This statement stops the execution of the loop in which it is used and passes the control of execution to the next statement after the loop. This

statement cannot be used anywhere else other than while loop. The syntax is as follows

```
break ;
```

top ↑

Continue Statement

Continue statement is a statement which is also used only in a while loop block. This statement skips the current iteration of the loop and passes the control to the next iteration after checking the loop condition. The syntax is as follows

```
continue ;
```

top ↑

ireturn Statement

ireturn statement or the Interrupt Return statement is used to pass control from kernel mode to user mode. **`ireturn;`** The ireturn is generally used at the end of an interrupt code.

This instruction translates to IRET machine instruction.

top ↑

Read/Print Statements

The read and print statements are used as standard input and output statements. The read statement reads a value from the standard input device and stores it in a register.

NOTE: String read or printed must not exceed 10 characters

The print statement outputs value of a register or an integer/string literal or value of a memory location.

```
read Register;
print Register / Number / String / Expression /
[Address];
```

top ↑

Load/Store Statement

Loading and storing between filesystem and memory is accomplished using load and store statements in SPL. load statement loads the block specified by *block_number* from the disk to the the page specified by the *page_number* in the memory. store statement stores the page specified by *page_number* in the memory to the the block specified by the *block_number* in the disk. The *page_number* and *block_number* can be specified using arithmetic expressions.

```
load (page_number, block_number);
store (page_number, block_number);
```

top ↑

Halt Statement

The Halt statement is used to halt the machine.

```
halt;
```

This instruction translates to HALT machine instruction.

top ↑

Inline Statement

The inline statement is used give XSM machine instructions directly within an SPL program.

```
inline "MACHINE INSTRUCTION";
```

e.g. `inline "JMP 11776";`

top ↑
