CSC4510 Programming Language Design and Translation

The PAL Machine

Object Code Format

The object code to be produced by your compiler is the machine language of the "PAL-machine". Object code instructions are to be written to a text file, one instruction per line, each using one of the following formats:

<function code> <integer number> <integer number> <comment>
<function code> <integer number> <real number> <comment>
<function code> <integer number> <string> <comment>

The function code is a three-character code occupying the first three character positions of the line, designating which instruction is intended. The other fields of the instruction must be separated by at least one blank character. Integer and real numbers follow the syntax of corresponding lexical elements in typical programming languages. Strings are delimited by single quote characters. Single quote characters embedded within the string are not permitted. The comment is optional and is any sequence of characters.

The PAL-machine is a high-level, language-oriented computer that uses a "tagged" architecture. It includes the concept of a "variable" (implemented as a level difference and a displacement) and each location in memory has a type ("tag"). The possible types are "bool", "real", "int", "string" and "undef".

PAL-machine Instructions

MST	**L**	**0**	mark the stack
CAL	**M**	**A**	procedure call
INC	**0**	**I**	increment top-of-stack pointer
JIF	**0**	**A**	jump if false to address A
JMP	**0**	**A**	jump to address A
LCI	**0**	**I**	load integer constant onto the stack
LCR	**0**	**R**	load real constant onto the stack
LCS	**0**	**S**	load string literal onto the stack
LDA	**L**	**D**	load the absolute address of the variable onto the stack
LDI	**0**	**0**	load the value stored at the address indicated by the value at the top of stack
LDV	**L**	**D**	load the value of a variable onto the stack
LDU	**0**	**0**	load an undefined or void value
OPR	**0**	**I**	execute operation I
RDI	**L**	**D**	read a value into an integer variable
RDR	**L**	**D**	read a value into a real variable
STI	**0**	**0**	load top of stack -1 into variable at address top of stack
STO	**L**	**D**	store into a variable
SIG	**0**	**I**	raise signal I
REH	**0**	**A**	register exception handler at address A

where "**A**" is an address in the instruction store
"**D**" is a displacement in the memory store
"**I**" is an integer number
"**L**" is a level difference
"**M**" is the number of parameters
"**R**" is a real number
"**S**" is a string

The operations which may be executed by the OPR instruction are as follows:

0 procedure return
1 function return
2 negate
3 addition
4 subtraction
5 multiplication
6 division
7 exponentiation
8 string concatenation
9 odd
10 == (test for equality)
11 /= (test for inequality)
12 <
13 >=
14 >
15 <=
16 logical complement (not)
17 true
18 false
19 eof
20 write the integer, real or string value on top of the stack to the output file
21 terminate the current line of the output file
22 swap the top two elements of the stack
23 duplicate the element on top of the stack
24 drop the element on top of the stack
25 integer-to-real conversion
26 real-to-integer conversion
27 integer-to-string conversion
28 real-to-string conversion
29 logical and
30 logical or
31 is(exception) – raises the exception matching the integer at the top of the stack

The explanations for each of the instructions are as follows:

MST L 0
Mark the stack frame.
Used in calling a procedure, the stack is firstly marked, then parameters are loaded onto it and finally the call is executed.
L is the level difference between the call to the procedure and the declaration of the procedure.
For example, in the following

```
procedure A is begin ... end;
procedure B is begin ... A ... end;
```

the level difference is 1 between the call to A and the declaration of A, however with

```
procedure A is
        procedure B is begin ... end;
begin ... B ... end;
```

the level difference is 0 as procedure B is declared at the same level that it is called from.

CAL M A
The new base is calculated from the current top-of-stack minus the parameters already on the stack, where M is the number of parameters. The return address is stored in the frame base and the program counter jumps to the instruction at address A.

INC 0 I
Increment the top-of-stack pointer.
The top-of-stack pointer is incremented by the integer I. Any stack positions skipped through the increment are given the type undefined.
This is generally used to allocate for variables.

JIF 0 A
Jump if false to address A.
If the element on top of stack is a bool
Then
    If the value on top of stack if false
    Then program register is assigned to be A
    If the program register is not inside the range of instructions
    Then an appropriate error message is issued and the program is halted.
Else an appropriate error message is issued.

JMP 0 A
Jump to address A.
The program register is assigned to be the integer value A.
If program register is outside the instruction range
Then an appropriate error message is issued and the program is halted.
"JMP 0 0" is the only way for a program to terminate normally.

LCI 0 I
Load integer constant onto stack.
Top-of-stack register is incremented by 1.
The new element at the top-of-stack is assigned to be of type integer and is given the integer value I.

LCR 0 R
Load real constant onto stack.
Top-of-stack register is incremented by 1.
The new element at the top-of-stack is assigned to be of type real and is given the real value R.

LCS 0 S
Load string literal onto the stack.
Top-of-stack register is incremented by 1.
The new element at the top-of-stack is assigned to be of type string and is given the string value S.

LDA L D
Load the address of a variable onto the top of the stack.
Top-of-stack register is incremented by 1.
The absolute address of the variable at stack location level difference L and displacement D is loaded into the new top-of-stack position.

LDI 0 0
Load the value stored at the address indicated by the value at the top of stack.
The value of the variable, whose address is specified in the top-of-stack position, is loaded into the new top-of-stack position. The top-of-stack register remains unchanged.

LDV L D
Load the value of a variable onto the top of the stack.
Top-of-stack register is incremented by 1.
The variable at stack location level difference L and displacement D is loaded into the new top-of-stack position.

LDU 0 0
Load an undefined or void value onto the top of the stack.
Top-of-stack register is incremented by 1.

OPR 0 I

Value of I

0 {procedure return}
Top of stack is returned to the current base - 1. The program counter is set to the return address which is stored in the procedure stack frame and the base is set to the value of base before the procedure call.

1 {function return}
The value on the top of stack is assumed to be the function result.
Top of stack is returned to the current base. The program counter is set to the return address which is stored in the procedure stack frame and the base is set to the value of base before the procedure call. The function result is placed in the new top of stack.

2 {negate}

If the value on top-of-stack is an integer or real
Then the value on the top-of-stack is replaced by that value negated.

3 {addition}

If the elements which occupy the top and next to top positions on the stack are not both of type integer or real
Then an error message is issued and the program is halted
Else the value on top-of-stack is added to the value on top-of-stack - 1.
Both of these values are removed from the stack and the result is placed on the new top-of-stack.

4 {subtraction}

If the elements which occupy the top and next to top positions on the stack are not both of type integer or real
Then an error message is issued and the program is halted
Else the value on top-of-stack is subtracted from the value on top-of-stack - 1.
Both of these values are removed from the stack and the result is placed on the new top-of-stack.

5 {multiplication}

If the elements which occupy the top and next to top positions on the stack are not both of type integer or real
Then an error message is issued and the program is halted
Else the value on top-of-stack is multiplied by the value on top-of-stack - 1.
Both of these values are removed from the stack and the result is placed on the new top-of-stack.

6 {division}

If the elements which occupy the top and next to top positions on the stack are not both of type integer or real
Then an error message is issued and the program is halted
Else
    If the value on top-of-stack is not 0.0
    Then the value on top-of-stack - 1 is divided by the value on top-of-stack.
Both of these values are removed from the stack and the result is placed on the new top-of-stack.

7 {exponentiation}

Applies to the two elements occupying the top two positions of the stack.
The value on top-of-stack must be an integer and that at the next-to-top position may be an integer or real; the type of the latter determines the type of the result.
If the types of the top two elements on the stack are not consistent with this
Then an error message is issued and the program is halted
Else the value at the next-to-top position is raised to the power given by the value on top-of-stack.
Both of these values are removed from the stack and the result is placed on the new top-of-stack.

8 {string concatenation}

If the value on top-of-stack is a string, and the value at top-of-stack – 1 is a string
Then the string at the top-of-stack position is appended to the end of the string and the top-of-stack – 1 position.
    Both of these values are removed from the stack and the result is placed on the new top-of-stack,
Else an error message is issued and the program is halted.

9 **{odd}**

If the value on the top-of-stack is an integer
Then
    If the value on top-of-stack is odd
    Then replace the top-of-stack element with the boolean value *true*
    Else replace the top-of-stack with the boolean value *false*
Else an error message is issued, and the program is halted.

10 **{=}**

Applies to the elements that occupy the next-to-top and top positions on the stack.
If these elements are both of type integer or real
Then the element on the top of the stack is removed from the stack and the next-to-top element is replaced by the boolean element *true* or *false*.

11 **{/=}**

Applies to the elements that occupy the next-to-top and top positions on the stack.
If these elements are both of type integer or real
Then the top-of-stack element is removed from the stack and the next-to-top element is replaced by the boolean element *true* or *false*.

12 **{<}**

Applies to the elements that occupy the next-to-top and top positions on the stack.
If these elements are both of type integer or real
Then the top-of-stack element is removed from the stack and the next-to-top element is replaced by the boolean element *true* or *false*.

13 **{>=}**

Applies to the elements that occupy the next-to-top and top positions on the stack.
If these elements are both of type integer or real
Then the top-of-stack element is removed from the stack and the next-to-top element is replaced by the boolean element *true* or *false*.

14 **{>}**

Applies to the elements that occupy the next-to-top and top positions on the stack.
If these elements are both of type integer or real
Then the top-of-stack element is removed from the stack and the next-to-top element is replaced by the boolean element *true* or *false*.

15 **{<=}**

Applies to the elements that occupy the next-to-top and top positions on the stack.
If these elements are both of type integer or real
Then the top-of-stack element is removed from the stack and the next-to-top element is replaced by the boolean element *true* or *false*.

16 **{logical complement (not)}**

If the element which occupies the top position on the stack is a bool
Then the element is replaced on the top of the stack by its logical complement
Else an appropriate error message is issued and the program is halted.

17 **{true}**

Put the boolean element *true* on top of the stack.

18 **{false}**
Put the boolean operator *false* on the top of the stack.

19 **{eof}**
If end of file has been reached
Then put the boolean value *true* on the top of the stack
Else put the boolean value *false* on the top of the stack.

20 **{write the integer, real, or string value on top of the stack to the output file}**
If the element on top of stack is of type integer, bool, real or string
Then it is removed from the top of the stack and written to output
Else (it is of type bool or undefined) an appropriate error message is issued, and the program is halted.

21 **{terminate the current line of the output file}**
A newline command is written to output.

22 **{swap the top two elements of the stack}**
The elements on top of the stack and top of stack -1 are swapped.

23 **{duplicate the element on top of the stack}**
The element on top of the stack is duplicated and the new element is placed on top of the stack.

24 **{drop the element on top of the stack}**
The element on top of the stack is dropped from the stack.

25 **{integer-to-real conversion}**
If the value on top-of-stack is an integer
Then this value is replaced on the stack by the integer to real conversion of this value
Else an error message is issued and the program is halted.

26 **{real-to-integer conversion}**
If the value on top-of-stack is a real
Then this value is replaced on the stack by the real to integer conversion of this value
Else an error message is issued and the program is halted.

27 **{integer-to-string conversion}**
If the value on top-of-stack is an integer
Then this value is replaced on the stack by the integer to string conversion of this value
Else an error message is issued and the program is halted.

28 **{real-to-string conversion}**
If the value on top-of-stack is a real
Then this value is replaced on the stack by the real to string conversion of this value
Else an error message is issued and the program is halted.

29 **{logical and}**
If the elements which occupy the top two positions on the stack are bool
Then the elements are replaced by the logical *and* of the boolean values
Else an appropriate error message is issued and the program is halted.

30 {logical or}
If the elements which occupy the top two positions on the stack are bool
Then the elements are replaced by the logical *or* of the boolean values
Else an appropriate error message is issued and the program is halted.

31 {is(exception)}
The element on the top of the stack is an integer that represents an exception type.
This function pops the value and pushes a boolean value indicating if the current exception type matches.

RDI L D
Read a value into an integer variable.
At the stack location level difference L and displacement D, assign the type of the element to be int, and assign the value to be that of the integer value read in from the next line of the input file.
If eof was read
Then an appropriate error message is issued and the program is halted.

RDR L D
Read a value into a real variable.
At the stack location level difference L and displacement D, assign the type of the element to be real, and assign the value to be that of the real or integer value read in from the next line of the input file.
If eof was read
Then an appropriate error message is issued and the program is halted.

STI 0 0
Load the element in top-of-stack - 1 into the variable at the address specified by the element on the top-of-stack.
The top two elements are removed from the stack.

STO L D
Store into a variable.
Load the element on the top-of-stack into the stack location level difference L and displacement D and remove that element from the top of the stack.
If they are not of precisely the same type an error message is issued and the program is halted.

SIG 0 I
Raise a signal/exception.
This instruction causes the entire run–time stack to be searched looking for an exception handler which deals with the particular exception indicated. If a handler is found, all activation records down to the frame that contains the handler are discarded. Control is transferred to the exception handler.
The following exceptions are defined:

0 Re-raise the present active signal
This will cause the current signal to be re–raised and the current block to terminate.

1 Program Abort
This signal will cause termination of the program. Cannot be intercepted.

2 No return in function
No “return” value was generated for this function.

3 Type mis-match in input

RDI or RDR found a value which was neither integer nor real.

4 Attempt to read past end of file

End of file was true before RDI or RDR were executed.

REH 0 A

Register exception handler at address A

Address “A” is stored in the stack mark as the exception handling code for this stack frame. A list of built–in exceptions is given in the description of the “SIG” instruction. An address of zero indicates that no exception handler is registered.

Michael Oudshoorn
August 13, 2023