Sharif University of Technology

Compiler Course Project

Spring 1395

Chapter 5

Code Generator

After preparing scanner and parser, we can continue with code generator *(I won't explain anymore about CG and will continue with implementation details).*

5.1 Virtual Machine

Virtual machine works after the compiler has done its job, and executes the output of the compiler. The virtual machine and the compiler need not to be seperate programs.

5.2 Implementation

The final result of this project is an executable file, say compiler-vm.jar (or compiler-vm.exe). The executable takes two arguments, the first of which is the source file we want to compile and the other is the file the compiled source is written to, and compiles the source code. After the source is compiled, the virtual machine runs the program. So executing the following line should compile and run "`program.L`" and also write the compiled source to "`output.Lm`".

```
java -jar compiler-vm.jar program.L output.Lm
```

5.3 Errors

You will report any errors you see during the compiling process in form of exceptions including semantic errors (just like what you did in scanner and parser).

5.4 Extra Bonus Grade

If you optimize the generated code, you get up to 1 grade (It depends on the level of optimization).

Chapter 6

Virtual Machine

Instructions appear in one line of output each and will have the following format:

[OpCode] [Opr_1] ... [Opr_n]

Where the number of operands (n) is determined for each instruction separately. All of the required instructions in *Decaf* are listed in table below. If you need any instructions that is not listed here, please inform me.

Operator Name	OpCode	# of Operands	Operation
Add	+	3	[Opr1] ← [Opr2] + [Opr3]
Subtract	−	3	[Opr1] ← [Opr2] − [Opr3]
Multiply	*	3	[Opr1] ← [Opr2] * [Opr3]
Divide	/	3	[Opr1] ← [Opr2] / [Opr3]
Mod	%	3	[Opr1] ← [Opr2] % [Opr3]
Logical And	&&	3	[Opr1] ← [Opr2] && [Opr3]
Logical Or	||	3	[Opr1] ← [Opr2] || [Opr3]
Less Than	<	3	[Opr1] ← [Opr2] < [Opr3]
Greater Than	>	3	[Opr1] ← [Opr2] > [Opr3]
Less Than Equal	<=	3	[Opr1] ← [Opr2] <= [Opr3]
Greater Than Equal	>=	3	[Opr1] ← [Opr2] >= [Opr3]
Equal	==	3	[Opr1] ← [Opr2] == [Opr3]
Not Equal	!=	3	[Opr1] ← [Opr2] != [Opr3]
Logical Not	!	2	[Opr1] ← ! [Opr2]
Assignment	:=	2	[Opr1] ← [Opr2]
Jump If True	*jt*	2	if [Opr1]==TRUE then pc ← [Opr2]
Jump	*jmp*	1	pc ← [Opr1]
Write Integer	*wi*	1	{output} ← [Opr1]
Write Float	*wf*	1	{output} ← [Opr1]
Write Boolean	*wb*	1	{output} ← [Opr1]
Write Character	*wc*	1	{output} ← [Opr1]
Read Integer	*ri*	1	{input} →[Opr1]
Read Float	*rf*	1	{input} →[Opr1]
Read Boolean	*rb*	1	{input} →[Opr1]
Read Character	*rc*	1	{input} →[Opr1]

Operator Name	OpCode	# of Operands	Operation
PC Value	:= *pc*	1	[Opr1] ← pc
SP Value	:= *sp*	1	[Opr1] ← sp
Assign SP	*sp* :=	1	sp ← [Opr1]
Return	*ret*	2	pc ← [Opr1] , sp ← [Opr2]

6.1 Operands

Each of the operands will be of the following format:

[Addressing Mode] [Type] [Value]

You have to concatenate their text values in order to obtain the operand. For immediate addressing, value will be the literal except characters. The immediate value for characters should be their ASCII value. In other kinds of addressing, value will be a memory address (integer).

6.2 Addressing Modes

in *Decaf* we will need at most five kind of addressing mode.

Addressing Mode	Text Form
Global Direct	gd_
Global Indirect	gi_
Local Direct	ld_
Local Indirect	li_
Immediate	im_

6.3 Types

Type	Text Form
Integer	i_
Float	f_
Boolean	b_
Char	c_

6.4 Example

In this section you can see some examples from instructions in text form. the white space between operator and operands can be a single space or tab.

```
+   gd_i_12  im_i_5     ld_i_14
wc  im_c_13
*   gi_f_10  im_f_10.5  ld_f_10
```