Lexical

The various components involved in the lexical analysis phase of interpreting a programming language. Each entry describes a specific symbol type and its content or rules.

NO.	Symbol	Content
1	Token word	if, else, int, while, for, read, write
2	Special symbol	Single word: +, -, *, /, =, ;, !, <, >, (,), {, } Double word: <=, >=, ==, !=
3	Identifier	Begin with an alphabet followed by alphabet or number
4	Number	Unsigned number
5	Separator	From the source program in order to correctly identify various characters stream words, symbols, adjacent identifiers, integers or reserved words must be separated by spaces (at least one). This ensures correct lexical analysis.
6	Comment	Begin from /* and end with */, not nested
7	Lexical rules	Regular grammar: <identifier> → <letter> | <identifier><letter> | <identifier><digit> <number> → <digit> | <number><digit> <letter> → a | b | … | z | A | ... | Z <digit> → 0 | 1 | … | 9 <singleword> → + | - | * | / | = | ; | ! | < | > | (|) | { | } <doubleword> → <= | >= | == | != <comment_star> → /* <comment_end> → */

Grammar

The grammar rules used to define the structure of a programming language. Each entry in the table specifies production rules for various language constructs.

NO.	Content
1	<Programme>→{<StatementList>}
2	<StatementList>→<StatementList><Statement>|<Statement>
3	<Statement>→ <VarDeclare>|<IfStat>|<WhileStat>|<ForStat>|<ReadStar>|<WriteStar>|<ExpressionStat>
4	<VarDeclare>→int ID|int ID=NUM
5	<IfStat>→if (<Expr>) <Statement > [else < Statement >]
6	<WhileStat>→While (<Expr >) < Statement >
7	<ForStat>→For(<Expr>,<Expr>,<Expr>)<Statement>
8	<WriteStat>→Write <Expression>;
9	<ReadStat>→Read <ID>;
10	<ExpressionStat>→<Expression>;|;
11	<Expr>→ID=<BoolExpr>|<BoolExpr>
12	<BoolExpr>→<AdditiveExpr>|< AdditiveExpr >(>|<|>=|<=|==|!=)< AdditiveExpr >
13	<AdditiveExpr>→<Term>{(+|-)< Term >}
14	< Term >→<Factor>{(*| /)< Factor >}
15	< Factor >→(<AdditiveExpr>)| ID|NUM

Instruction

In order to enhance readability and simplify the code generation process, our target platform uses an abstract stack-based computer. This machine uses a stack to store operands and has ample memory space. The assembly instructions for this abstract machine are defined as follows:

NO.	Opcode	Operand	Action
0	Load	d	Load the content of d onto the operand stack.
1	Loadi	N	Push the constant N onto the operand stack.
2	Sto	d	Store the content of the top element of the operand stack into d, while keeping the top element unchanged.
3	Sti	d	Store the content of the top element of the operand stack into d, and then pop the top element.
4	Add	-	Pop the second-to-top and the top elements of the operand stack, add their values, and push the result onto the top of the stack.
5	Sub	-	Subtract the top element from the second-to-top element of the operand stack, pop both elements, and push the result onto the top of the stack.
6	Mult	-	Pop the second-to-top and the top elements of the operand stack, multiply their values, and push the product onto the top of the stack.
7	Div	-	Pop the second-to-top and the top elements of the operand stack, divide the second-to-top by the top element, and push the quotient onto the top of the stack.
8	jmp	lab	Unconditionally jump to label lab.
9	jcxz	lab	Check the logical value of the top element of the stack; if it is false (0), jump to label lab.
10	Eq	-	Compare the top two elements of the stack for equality, and push the result (true or false, 1 or 0) onto the top of the stack.
11	Noteq	-	Compare the top two elements of the stack for inequality, and push the result (true or false, 1 or 0) onto the top of the stack.
12	Gt	-	If the second-to-top element is greater than the top element of the stack, push 1 onto the top of the stack; otherwise, push 0.
13	Les	-	If the second-to-top element is less than the top element of the stack, push 1 onto the top of the stack; otherwise, push 0.
14	Ge	-	If the second-to-top element is greater than or equal to the top element of the stack, push 1 onto the top of the stack; otherwise, push 0.
15	Le	-	If the second-to-top element is less than or equal to the top element of the stack, push 1 onto the top of the stack; otherwise, push 0.
16	And	-	Perform a logical AND operation on the top two elements of the stack, and push the result (true or false, 1 or 0) onto the top of the stack.
17	Or	-	Perform a logical OR operation on the top two elements of the stack, and push the result (true or false, 1 or 0) onto the top of the stack.

18	Not	-	Negate the logical value of the top element of the stack.
19	In	-	Read an integer from the standard input device (keyboard) and push it onto the operand stack.
20	Out	-	Pop the top element of the stack and output it to the standard output device (monitor).
21	Stop	-	Halt execution.