Lexical

1．Token word: if ， else，int， while，for，read ， write

2．special symbol：

single word：+, -, *, /,=，；,！， <,> ，(，) ，{ ， }

double word：<= , >=, ==, !=

3. identifier：begin with a 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 identifier, integer or reserved word between separated by Spaces (at least one), in lexical analysis of oversight.

6. comment：begin form /* and end with */, Not nested.

7. Lexical rules regular grammar

<identifier>→<letter>|<identifier><lettle>|< identifier ><digit>

<number>→<digit>|<number> <digit>

<letter>→→a|b|…|z|A|..|Z

<digit>→0|1|…|9

<singleword>→+| -| *| /|=|; |！ | <|> |(|) |{ | }

<doubleword>→<= |>=| ==| !=

<comment_star>→/*

<comment_end>→*/

Grammar

1. <程序>→{<语句序列>}
   <Programme>→{<StatementList>}
2. <语句序列>→<语句序列><语句>|<语句>
   <StatementList>→<StatementList><Statement>|<Statement>
3. <语句>→
   <声明语句>|<if 语句>|<while 语句>|<for 语句>|<read 语句>|<write 语句>|<表达式语句>
   <Statement>→
   <VarDeclare>|<IfStat>|<WhileStat>|<ForStat>|<ReadStar>|<WriteStar>|<ExpressionStat>
4. <声明语句>→int <变量>|int<变量>=<无符号整数>
   <VarDeclare>→int ID|int ID=NUM
5. <IF 语句>→if (<表达式>) <语句 > [else <语句 >]
   <IfStat>→if (<Expr>) <Statement > [else < Statement >]
6. <while 语句>→while(<表达式>) <语句>
   <WhileStat>→While (<Expr >) < Statement >
7. <for 语句>→for(<表达式>;<表达式>;<表达式>) <语句 >
   <ForStat>→For(<Expr>,<Expr>,<Expr>)<Statement>
8. <write_语句>→write <表达式>;
   <WriteStat>→Write <Expression>;
9. <read_语句>→read <变量>;
   <ReadStat>→Read <ID>;

10. <表达式语句>→<<表达式>;|;
    <ExpressionStat>→<Expression>;|;
11. <表达式>→<标识符>=<布尔表达式>|<布尔表达式>
    <Expr>→ID=<BoolExpr>|<BoolExpr>
12. <布尔表达式>→<算术表达式>|<算术表达式>(>|<|>=|<=|==|!=)<算术表达式>
    <BoolExpr>→<AdditiveExpr>|< AdditiveExpr >(>|<|>=|<=|==|!=)< AdditiveExpr >
13. <算术表达式>→<项>{(+|-)<项>}
    <AdditiveExpr>→<Term>{(+|-)< Term >}
14. <项>→<因子>{(*|/)<因子>}
    < Term >→<Factor>{(*| /)< Factor >}
15. <因子>→(<算术表达式>)|<标识符>|<无符号整数>
    < Factor >→(<AdditiveExpr>)| ID|NUM

Instruction

In order to enhance readability, simplify code generation process, our target platform adopted by the machine is an abstract stack type computer, it with a stack to save the operand, and set the have enough memory space. Abstract machine assembly instructions defined as follows:

NO.	opcode	operand	action
0	Load	d	将 d 中的内容加载到操作数栈。
1	Loadi	N	将常量 N 压入操作数栈
2	Sto	d	将操作数栈栈顶单元内容存入 d，且栈顶单元内容保持不变。
3	Sti	d	将操作数栈栈顶单元内容存入 d，且栈顶单元出栈。
4	Add	-	将次栈顶单元与栈顶单元内容出栈并相加，和置于栈顶。
5	Sub	-	将次栈顶单元减去栈顶单元内容并出栈，差置于栈顶。
6	Mult	-	将次栈顶单元与栈顶单元内容出栈并相乘，积置于栈顶。
7	Div	-	将次栈顶单元与栈顶单元内容出栈并相除，商置于栈顶。
8	jmp	lab	无条件转移到 lab
9	jcxz	lab	检查栈顶单元逻辑值，若为假(0)则转移到 lab
10	Eq	-	将栈顶两单元做等于比较，并将结果真或假(1 或 0)置于栈顶
11	Noteq	-	将栈顶两单元做不等于比较，并将结果真或假(1 或 0)置于栈顶
12	Gt	-	次栈顶大于栈顶操作数，则栈顶置 1，否则置 0
13	Les	-	次栈顶小于栈顶操作数，则栈顶置 1，否则置 0
14	Ge	-	次栈顶大于等于栈顶操作数，则栈顶置 1，否则置 0
15	Le	-	次栈顶小于等于栈顶操作数，则栈顶置 1，否则置 0
16	And	-	将栈顶两单元做逻辑与运算，并将结果真或假(1 或 0)置于栈顶
17	Or	-	将栈顶两单元做逻辑或运算，并将结果真或假(1 或 0)置于栈顶
18	Not	-	将栈顶元素逻辑值取反
19	In	-	从标准输入设备(键盘)读入一个整型数据，并入操作数栈
20	Out	-	将栈顶单元内容出栈，并输出到标准输出设备上(显示器)
21	Stop	-	停止执行