编译原理实验报告

熊永琦 16340258

2018 年 12 月 27 日

本项目所有代码均已在 我的 Github 仓库 开源

1 实验内容

1. 在计算机上实现 PL0 语言的编译程序
2. 扩展 PL0 语言的功能, 并在计算机上实现

1.1 实验一要求

1. 在 PASCAL 系统上运行 PL0 编译程序
2. 在 PASCAL 系统中, 为 PL0 的编译程序建立输入文件和输出文件
3. PL0 的编译程序运行时, 通过输入文件输入 PL0 源程序, 在输出文件中产生源程序的中间代码, 然后运行该中间代码, 在输出文件中产生运行数据

1.2 实验二要求

1. 在 PL0 语言中增加 Read 和 Write 语句
2. 修改 PL0 编译程序，使得 PL0 源程序可以使用 Read 和 Write 语句，从文件 (或键盘) 输入数据，并可以向文件 (或屏幕) 写数据

2 实验一报告

2.1 令编译程序正常运行

首先，我们修改了 PL0 编译程序，使得其能够通过 Free Pascal Compiler 的编译并正常运行。为了达到这个目标，我们订正了 PL0 中多处语法错误，并且改换了部分 PL0 操作

符，使得其源程序和编译器的代码能够使用标准 ASCII 编码空间覆盖。在编译程序中主要存在以下几种错误：

1. 变量命名混乱：许多变量直接使用了 Pascal 语言的保留字作为变量名，导致编译时出现语法错误。
2. 使用不合法的 ASCII 字符作为 PL0 逻辑操作符：编译程序选用 ≥、≤、≠ 作为 PL0 中的逻辑判断符号，由于 FPC 不能够识别这些非标准 ASCII 的字符，从而产生报错。
3. 不合法的跨子程序跳转：当读取数据产生错误时，编译程序会尝试跳转到 Label 99 的代码段来进行错误处理，然而这种跨子程序的跳转是不合法的，无法通过 FPC 编译。

对于上面三种问题，我们首先使用 &、% 两个符号替代了小于等于号和不等号，从而在保证使用单个字符作为操作符的前提下完成了 PL0 的逻辑操作符实现。然后我们将所有保留字的变量名替换为带有 m_ 前缀的变量名，从而规避了命名冲突带来的错误。最后，我们将 Label 99 处的代码段替换到所有跳转到它的代码中，就规避了最后一个问题。至此，编译程序已经可以正常运行。

2.2 编译程序的文件读写

原本的编译程序通过键盘写入源程序，并且将编译结果直接打印到屏幕上，这使得我们在进行代码文件的编译和结果验证的时候遇到了许多不便，因而添加一套文件读写系统对于编译程序是有必要的。我们首先在全局变量生命了四个文件（分别是源程序输入、源程序输出、中间代码输出、运行时结果输出）路径和四个文件类型，然后我们在运行时要求用户通过键盘输入四个文件的路径来指定这四个路径。

在程序开始运行时，我们通过 assign 命令打开这几个文件，并通过 reset 和 rewrite 命令初始化这四个文件。然后我们将所有的 read 和 write 由键盘/屏幕改定向到文件，就实现了文件读写部分的编码。关于此部分的代码和程序的输出结果，您可以在 我的 Github 仓库 里找到，他们分别是 os、oi、or 三个文件，您也可以在随附的附录中找到相应的内容。

3 实验二报告

3.1 为编译程序添加键盘读、屏幕写命令的支持

为了在 PL0 中使用 I/O 的功能（而非单纯执行写死的程序），我们需要首先修改编译程序，使得其能够正常编译和解释执行带有 I/O 代码的 PL0 程序。我们首先定义两个函数 Read() 和 Write()，他们都接收不定数量的变量作为参数。其中，Read 将会接收键盘的数个输入，并将他们的值赋予给参数表内的变量；Write 则会将参数表内变量的值打印在屏幕上。

为了实现对这两个函数的支持，我们首先需要在保留字枚举和函数枚举中添加相应的值。然后，我们在字典、词典和助记符表中分别加入这两个函数的对应字符串。这样就可以在 getsym 时正常解析到这两个函数了。接下来，我们改写句法分析和翻译部分的代码，在 switch case 中加入这两个语义的判断并执行相应的操作即可。

3.2 为 PL0 程序添加键盘读、屏幕写功能

只需要简单地将原本的赋值改为 I/O 语句即可。我们在这里使用 Read(x) Read(y) 来从键盘获取两个值填入到变量中，使用 Write(x, y, z) 来将输入的值和结果值输出。为了方便测试，我们将输入输出分别重定向到 stdin 和 stdout 两个文件。这一部分的代码和输出结果均可以在 我的 Github 仓库 里找到，您也可以在随附的附录中找到相应的内容。

Github 仓库地址：https://github.com/siskonemilia/PL0_Compiler

Appendices

A Experiment 1

A.1 编译程序 compiler.pas

```
Program PL0 ;
{带有代码生成的PL0编译程序}

Const
  norw = 11; {保留字的个数}
  txmax = 100; {标识符表长度}
  nmax = 14; {数字的最大位数}
  al = 10; {标识符的长度}
  amax = 2047; {最大地址}
  levmax = 3; {程序体嵌套的最大深度}
  cxmax = 200; {代码数组的大小}

Type
  symbol = (nul, ident, number, plus, minus, times, slash,
            oddsym, eql, neq, lss, leq, gtr, geq, lparen, rparen,
            comma, semicolon, period, becomes, beginsym,
            endsym, ifsym, thensym, whilesym, dosym,
            callsym, constsym, varsym, procsym );
  alfa = packed array [1..al] Of char;
  m_object = (constant, variable, m_procedure);
  symset = set Of symbol;
  fct = (lit, opr, lod, sto, cal, int, jmp, jpc); {functions}
  instruction = packed Record
    f : fct; {功能码}
    l : 0..levmax; {相对层数}
    a : 0..amax; {相对地址}
  End;
```

```
{LIT 0,a : 取常数a
  OPR 0,a : 执行运算a
  LOD l,a : 取层差为l的层  相对地址为a的变量
  STO l,a : 存到层差为l的层  相对地址为a的变量
  CAL l,a : 调用层差为l的过程
  INT 0,a : t寄存器增加a
  JMP 0,a : 转移到指令地址a处
  JPC 0,a : 条件转移到指令地址a处 }

Var
  ch : char; {最近读到的字符}
  sym : symbol; {最近读到的符号}
  id : alfa; {最近读到的标识符}
  num : integer; {最近读到的数}
  cc : integer; {当前行的字符计数}
  ll : integer; {当前行的长度}
  kk, err : integer;
  cx : integer; {代码数组的当前下标}
  line : array [1..81] Of char;
  a : alfa;
  code : array [0..cxmax] Of instruction;
  word : array [1..norw] Of alfa;
  wsym : array [1..norw] Of symbol;
  ssym : array [char] Of symbol;
  mnemonic : array [fct] Of packed array [1..5] Of char;
  declbegsys, statbegsys, facbegsys : symset;
  table : array [0..txmax] Of
          Record
            name : alfa;
            Case kind : m_object Of
              constant : (val : integer);
              variable, m_procedure : (level, adr : integer)
          End;
  sfin, sfout, ifout, rfout: text;
  sfinp, sfoutp, ifoutp, rfoutp: string;
Procedure error (n : integer);
Begin
```

```
  writeln(sfout, '****', '␣' : cc-1, '↑', n : 2);
  err := err + 1
End {error};
Procedure getsym;

Var  i, j, k : integer;
Procedure getch;
Begin
  If cc = ll Then
    Begin
      If eof(sfin) Then
        Begin
          write('PROGRAM␣INCOMPLETE');
          close(sfin);
          close(sfout);
          close(ifout);
          close(rfout);
          exit;
        End;
      ll := 0;
      cc := 0;
      write(sfout, cx : 5, '␣');
      While Not eoln(sfin) Do
        Begin
          ll := ll + 1;
          read(sfin, ch);
          write(sfout, ch);
          line[ll] := ch
        End;
      writeln(sfout);
      readln(sfin);
      ll := ll + 1;
      line[ll] := '␣';
    End;
  cc := cc + 1;
  ch := line[cc]
End {getch};
```

```
Begin {getsym}
  While ch = '␣' Do
    getch;
  If ch In ['a'..'z'] Then
    Begin {标识符或保留字}
      k := 0;
      Repeat
        If k < al Then
          Begin
            k := k + 1;
            a[k] := ch
          End;
        getch
      Until Not (ch In ['a'..'z', '0'..'9']);
      If k >= kk  Then kk := k
      Else
        Repeat
          a[kk] := '␣';
          kk := kk−1
        Until kk = k;
      id := a;
      i := 1;
      j := norw;
      Repeat
        k := (i+j) Div 2;
        If id <= word[k] Then j := k−1;
        If id >= word[k] Then i := k + 1
      Until i > j;
      If i−1 > j Then sym := wsym[k]
      Else sym := ident
    End
  Else
    If ch In ['0'..'9'] Then
      Begin {数字}
        k := 0;
        num := 0;
        sym := number;
        Repeat
          num := 10*num + (ord(ch)-ord('0'));
          k := k + 1;
          getch;
        Until Not (ch In ['0'..'9']);
        If k > nmax Then error(30)
      End
  Else
    If ch = ':' Then
      Begin
        getch;
        If ch = '=' Then
          Begin
            sym := becomes;
            getch
          End
        Else  sym := nul;
      End
  Else
    Begin
      sym := ssym[ch];
      getch
    End
End {getsym};
Procedure  gen(x : fct; y, z : integer);
Begin
  If cx > cxmax Then
    Begin
      write('PROGRAM␣TOO␣LONG');
      close(sfin);
      close(sfout);
      close(ifout);
      close(rfout);
      exit;
    End;
  With code[cx] Do
    Begin
```

```
      f := x;
      l := y;
      a := z
    End;
  cx := cx + 1
End {gen};
Procedure test(s1, s2 : symset; n : integer);
Begin
  If Not (sym In s1) Then
    Begin
      error(n);
      s1 := s1 + s2;
      While Not (sym In s1) Do
        getsym
    End
End {test};
Procedure block(lev, tx : integer; fsys : symset);

Var
  dx : integer; {本过程数据空间分配下标}
  tx0 : integer; {本过程标识表起始下标}
  cx0 : integer; {本过程代码起始下标}
Procedure enter(k : m_object);
Begin {把m_object填入符号表中}
  tx := tx +1;
  With table[tx] Do
    Begin
      name := id;
      kind := k;
      Case k Of
        constant :
                  Begin
                    If num > amax Then
                      Begin
                        error(30);
                        num := 0
                      End;
                    val := num
                  End;
        variable :
                  Begin
                    level := lev;
                    adr := dx;
                    dx := dx +1;
                  End;
        m_procedure : level := lev
      End
    End
End {enter};
Function position(id : alfa) : integer;

Var i : integer;
Begin {在标识符表中查标识符id}
  table[0].name := id;
  i := tx;
  While table[i].name <> id Do
    i := i-1;
  position := i
End {position};
Procedure constdeclaration;
Begin
  If sym = ident Then
    Begin
      getsym;
      If sym In [eql, becomes] Then
        Begin
          If sym = becomes Then error(1);
          getsym;
          If sym = number Then
            Begin
              enter(constant);
              getsym
            End
          Else error(2)
        End
      Else error(3)
    End
  Else error(4)
End {constdeclaration};
Procedure vardeclaration;
Begin
  If sym = ident Then
    Begin
      enter(variable);
      getsym
    End
  Else error(4)
End {vardeclaration};
Procedure listcode;

Var i : integer;
Begin {列出本程序体生成的代码}
  For i := cx0 To cx-1 Do
    With code[i] Do
      writeln(ifout, i, mnemonic[f] : 5, l : 3, a : 5)
End {listcode};
Procedure statement(fsys : symset);

Var i, cx1, cx2 : integer;
Procedure expression(fsys : symset);

Var addop : symbol;
Procedure term(fsys : symset);

Var mulop : symbol;
Procedure factor(fsys : symset);

Var i : integer;
Begin
  test(facbegsys, fsys, 24);
  While sym In facbegsys Do
```

```
    Begin
      If sym = ident Then
        Begin
          i := position(id);
          If i = 0 Then error(11)
          Else
            With table[i] Do
              Case kind Of
                constant : gen(lit, 0, val);
                variable : gen(lod, lev-level, adr);
                m_procedure : error(21)
              End;
          getsym
        End
      Else
        If sym = number Then
          Begin
            If num > amax Then
              Begin
                error(30);
                num := 0
              End;
            gen(lit, 0, num);
            getsym
          End
      Else
        If sym = lparen Then
          Begin
            getsym;
            expression([rparen]+fsys);
            If sym = rparen Then getsym
            Else error(22)
          End;
      test(fsys, [lparen], 23)
    End
End {factor};
Begin {term}
```

```
  factor(fsys+[times, slash]);
  While sym In [times, slash] Do
    Begin
      mulop := sym;
      getsym;
      factor(fsys+[times, slash]);
      If mulop = times Then gen(opr, 0, 4)
      Else gen(opr, 0, 5)
    End
End {term};
Begin {expression}
  If sym In [plus, minus] Then
    Begin
      addop := sym;
      getsym;
      term(fsys+[plus, minus]);
      If addop = minus Then gen(opr, 0, 1)
    End
  Else term(fsys+[plus, minus]);
  While sym In [plus, minus] Do
    Begin
      addop := sym;
      getsym;
      term(fsys+[plus, minus]);
      If addop = plus Then gen(opr, 0, 2)
      Else gen(opr, 0, 3)
    End
End {expression};
Procedure condition(fsys : symset);

Var relop : symbol;
Begin
  If sym = oddsym Then
    Begin
      getsym;
      expression(fsys);
      gen(opr, 0, 6)
    End
  Else
    Begin
      expression([eql, neq, lss, gtr, leq, geq] + fsys);
      If Not (sym In [eql, neq, lss, leq, gtr, geq]) Then
        error(20)
      Else
        Begin
          relop := sym;
          getsym;
          expression(fsys);
          Case relop Of
            eql : gen(opr, 0, 8);
            neq : gen(opr, 0, 9);
            lss : gen(opr, 0, 10);
            geq : gen(opr, 0, 11);
            gtr : gen(opr, 0, 12);
            leq : gen(opr, 0, 13);
          End
        End
    End
End {condition};
Begin {statement}
  If sym = ident Then
    Begin
      i := position(id);
      If i = 0 Then error(11)
      Else
        If table[i].kind <> variable Then
          Begin {对非变量赋值}
            error(12);
            i := 0;
          End;
      getsym;
      If sym = becomes Then getsym
      Else error(13);
      expression(fsys);
      If i <> 0 Then
        With table[i] Do
          gen(sto, lev-level, adr)
    End
  Else
    If sym = callsym Then
      Begin
        getsym;
        If sym <> ident Then error(14)
        Else
          Begin
            i := position(id);
            If i = 0 Then error(11)
            Else
              With table[i] Do
                If kind = m_procedure Then
                  gen(cal, lev-level, adr)
                Else error(15);
            getsym
          End
      End
  Else
    If sym = ifsym Then
      Begin
        getsym;
        condition([thensym, dosym]+fsys);
        If sym = thensym Then getsym
        Else error(16);
        cx1 := cx;
        gen(jpc, 0, 0);
        statement(fsys);
        code[cx1].a := cx
      End
  Else
    If sym = beginsym Then
      Begin
        getsym;
```

```
        statement([semicolon, endsym]+fsys);
        While sym In [semicolon]+statbegsys Do
          Begin
            If sym = semicolon Then getsym
            Else error(10);
            statement([semicolon, endsym]+fsys)
          End;
        If sym = endsym Then getsym
        Else error(17)
      End
  Else
    If sym = whilesym Then
      Begin
        cx1 := cx;
        getsym;
        condition([dosym]+fsys);
        cx2 := cx;
        gen(jpc, 0, 0);
        If sym = dosym Then getsym
        Else error(18);
        statement(fsys);
        gen(jmp, 0, cx1);
        code[cx2].a := cx
      End;
  test(fsys, [ ], 19)
End {statement};
Begin {block}
  dx := 3;
  tx0 := tx;
  table[tx].adr := cx;
  gen(jmp, 0, 0);
  If lev > levmax Then error(32);
  Repeat
    If sym = constsym Then
      Begin
        getsym;
        Repeat
      constdeclaration;
      While sym = comma Do
        Begin
          getsym;
          constdeclaration
        End;
      If sym = semicolon Then getsym
      Else error(5)
    Until sym <> ident
  End;
If sym = varsym Then
  Begin
    getsym;
    Repeat
      vardeclaration;
      While sym = comma Do
        Begin
          getsym;
          vardeclaration
        End;
      If sym = semicolon Then getsym
      Else error(5)
    Until sym <> ident;
  End;
While sym = procsym Do
  Begin
    getsym;
    If sym = ident Then
      Begin
        enter(m_procedure);
        getsym
      End
    Else error(4);
    If sym = semicolon Then getsym
    Else error(5);
    block(lev+1, tx, [semicolon]+fsys);
    If sym = semicolon Then
```

```
          Begin
            getsym;
            test(statbegsys+[ident, procsym], fsys, 6)
          End
        Else error(5)
      End;
    test(statbegsys+[ident], declbegsys, 7)
  Until Not (sym In declbegsys);
  code[table[tx0].adr].a := cx;
  With table[tx0] Do
    Begin
      adr := cx; {代码开始地址}
    End;
  cx0 := cx;
  gen(int, 0, dx);
  statement([semicolon, endsym]+fsys);
  gen(opr, 0, 0); {生成返回指令}
  test(fsys, [ ], 8);
  listcode;
End {block};
Procedure interpret;

Const stacksize = 500;

Var p, b, t : integer;
  {程序地址寄存器, 基地址寄存器,栈顶地址寄存器}
  i : instruction; {指令寄存器}
  s : array [1..stacksize] Of integer; {数据存储栈}
Function base(l : integer) : integer;

Var b1 : integer;
Begin
  b1 := b; {顺静态链求层差为l的层的基地址}
  While l > 0 Do
    Begin
      b1 := s[b1];
      l := l-1
    End;
  base := b1
End {base};
Begin
  writeln('START PL/0');
  t := 0;
  b := 1;
  p := 0;
  s[1] := 0;
  s[2] := 0;
  s[3] := 0;
  Repeat
    i := code[p];
    p := p+1;
    With i Do
      Case f Of
        lit :
              Begin
                t := t+1;
                s[t] := a
              End;
        opr : Case a Of {运算}
                0 :
                    Begin {返回}
                      t := b-1;
                      p := s[t+3];
                      b := s[t+2];
                    End;
                1 : s[t] := -s[t];
                2 :
                    Begin
                      t := t-1;
                      s[t] := s[t] + s[t+1]
                    End;
                3 :
                    Begin
                      t := t-1;
                      s[t] := s[t]-s[t+1]
                    End;
                4 :
                    Begin
                      t := t-1;
                      s[t] := s[t] * s[t+1]
                    End;
                5 :
                    Begin
                      t := t-1;
                      s[t] := s[t] Div s[t+1]
                    End;
                6 : s[t] := ord(odd(s[t]));
                8 :
                    Begin
                      t := t-1;
                      s[t] := ord(s[t] = s[t+1])
                    End;
                9:
                   Begin
                     t := t-1;
                     s[t] := ord(s[t] <> s[t+1])
                   End;
                10 :
                     Begin
                       t := t-1;
                       s[t] := ord(s[t] < s[t+1])
                     End;
                11:
                    Begin
                      t := t-1;
                      s[t] := ord(s[t] >= s[t+1])
                    End;
                12 :
                     Begin
                       t := t-1;
                       s[t] := ord(s[t] > s[t+1])
```

```
                    End;
                13 :
                    Begin
                      t := t-1;
                      s[t] := ord(s[t] <= s[t+1])
                    End;
              End;
        lod :
              Begin
                t := t + 1;
                s[t] := s[base(l) + a]
              End;
        sto :
              Begin
                s[base(l) + a] := s[t];
                writeln(rfout, s[t]);
                t := t-1
              End;
        cal :
              Begin {generate new block mark}
                s[t+1] := base( l );
                s[t+2] := b;
                s[t+3] := p;
                b := t+1;
                p := a
              End;
        int : t := t + a;
        jmp : p := a;
        jpc :
              Begin
                If s[t] = 0 Then p := a;
                t := t-1
              End
      End {with, case}
  Until p = 0;
  write('END␣PL/0');
End {interpret};
```

```
Begin  {主程序}
  writeln('Type␣in␣the␣path␣to␣your␣source␣code:␣');
  readln(sfinp);
  writeln('Type␣in␣path␣of␣the␣file␣to␣output␣source␣program␣at');
  readln(sfoutp);
  writeln('Type␣in␣path␣of␣the␣file␣to␣output␣intermediate␣program␣at');
  readln(ifoutp);
  writeln('Type␣in␣path␣of␣the␣file␣to␣output␣runtime␣data␣at');
  readln(rfoutp);
  assign(sfin, sfinp);
  assign(sfout, sfoutp);
  assign(ifout, ifoutp);
  assign(rfout, rfoutp);
  reset(sfin);
  rewrite(sfout);
  rewrite(ifout);
  rewrite(rfout);
  For ch := 'A' To ';' Do
    ssym[ch] := nul;

  word[1] := 'begin␣␣␣␣␣';
  word[2] := 'call␣␣␣␣␣␣';
  word[3] := 'const␣␣␣␣␣';
  word[4] := 'do␣␣␣␣␣␣␣␣';
  word[5] := 'end␣␣␣␣␣␣␣';
  word[6] := 'if␣␣␣␣␣␣␣␣';
  word[7] := 'odd␣␣␣␣␣␣␣';
  word[8] := 'procedure␣';
  word[9] := 'then␣␣␣␣␣␣';
  word[10] := 'var␣␣␣␣␣␣␣';
  word[11] := 'while␣␣␣␣␣';

  wsym[1] := beginsym;
  wsym[2] := callsym;
  wsym[3] := constsym;
  wsym[4] := dosym;
```

```
wsym[5] := endsym;
wsym[6] := ifsym;
wsym[7] := oddsym;
wsym[8] := procsym;
wsym[9] := thensym;
wsym[10] := varsym;
wsym[11] := whilesym;

ssym['+'] := plus;
ssym['-'] := minus;
ssym['*'] := times;
ssym['/'] := slash;
ssym['('] := lparen;
ssym[')'] := rparen;
ssym['='] := eql;
ssym[','] := comma;
ssym['.'] := period;
ssym['&'] := neq;
ssym['<'] := lss;
ssym['>'] := gtr;
ssym[';'] := semicolon;
ssym['%'] := leq;

mnemonic[lit] := 'LIT␣␣';
mnemonic[opr] := 'OPR␣␣';
mnemonic[lod] := 'LOD␣␣';
mnemonic[sto] := 'STO␣␣';
mnemonic[cal] := 'CAL␣␣';
mnemonic[int] := 'INT␣␣';
mnemonic[jmp] := 'JMP␣␣';
mnemonic[jpc] := 'JPC␣␣';

declbegsys := [constsym, varsym, procsym];
statbegsys := [beginsym, callsym, ifsym, whilesym];
facbegsys := [ident, number, lparen];
err := 0;
cc := 0;
```

```
   cx := 0;
   ll := 0;
   ch := '␣';
   kk := al;
   getsym;
   block(0, 0, [period]+declbegsys+statbegsys);
   If sym <> period Then error(9);
   If err = 0 Then interpret
   Else write('ERRORS␣IN␣PL/0␣PROGRAM');
   writeln;
   close(sfin);
   close(sfout);
   close(ifout);
   close(rfout);
   exit;
End.
```

A.2 输入源程序 source.pl0

```
const  m = 7, n = 85;
var  x, y, z, q, r;
procedure  multiply;
  var  a, b;
  begin  a := x;  b := y;  z := 0;
while b > 0 do
begin
  if odd b then z := z + a;
  a := 2*a ;  b := b/2 ;
end
  end;
procedure  divide;
  var  w;
  begin  r := x;  q := 0;  w := y;
while w % r do w := 2*w ;
while w > y do
begin  q := 2*q;  w := w/2;
  if w % r then
  begin  r := r-w;  q := q+1 end
end
  end;
procedure  gcd;
  var  f, g ;
  begin  f := x;  g := y;
while f & g do
begin
  if f < g then g := g-f;
  if g < f then f := f-g;
end;
z := f
  end;
begin
  x := m;  y := n;  call multiply;
  x := 25;  y:= 3;  call divide;
  x := 84;  y := 36;  call gcd;
end.
```

A.3 输出源程序 os

```
  0 const  m = 7, n = 85;
  1 var  x, y, z, q, r;
  1 procedure  multiply;
  1   var  a, b;
  2   begin  a := x;  b := y;  z := 0;
  9 while b > 0 do
 13 begin
 13   if odd b then z := z + a;
 20   a := 2*a ;  b := b/2 ;
 28 end
 28   end;
 30 procedure  divide;
 30   var  w;
 31   begin  r := x;  q := 0;  w := y;
 38 while w % r do w := 2*w ;
 47 while w > y do
 51 begin  q := 2*q;  w := w/2;
 59   if w % r then
 62   begin  r := r-w;  q := q+1 end
 71 end
 71   end;
 73 procedure  gcd;
 73   var  f, g ;
 74   begin  f := x;  g := y;
 79 while f & g do
 83 begin
 83   if f < g then g := g-f;
 91   if g < f then f := f-g;
 99 end;
100 z := f
101   end;
103 begin
104   x := m;  y := n;  call multiply;
109   x := 25;  y:= 3;  call divide;
114   x := 84;  y := 36;  call gcd;
119 end.
```

A.4 输出中间代码 oi

```
2INT     0    5
3LOD     1    3
4STO     0    3
5LOD     1    4
6STO     0    4
7LIT     0    0
8STO     1    5
9LOD     0    4
10LIT     0    0
11OPR     0   12
12JPC     0   29
13LOD     0    4
14OPR     0    6
15JPC     0   20
16LOD     1    5
17LOD     0    3
18OPR     0    2
19STO     1    5
20LIT     0    2
21LOD     0    3
22OPR     0    4
23STO     0    3
24LOD     0    4
25LIT     0    2
26OPR     0    5
27STO     0    4
28JMP     0    9
29OPR     0    0
31INT     0    4
32LOD     1    3
33STO     1    7
34LIT     0    0
35STO     1    6
36LOD     1    4
37STO     0    3
38LOD     0    3
```

```
39LOD    1     7
40OPR    0    13
41JPC    0    47
42LIT    0     2
43LOD    0     3
44OPR    0     4
45STO    0     3
46JMP    0    38
47LOD    0     3
48LOD    1     4
49OPR    0    12
50JPC    0    72
51LIT    0     2
52LOD    1     6
53OPR    0     4
54STO    1     6
55LOD    0     3
56LIT    0     2
57OPR    0     5
58STO    0     3
59LOD    0     3
60LOD    1     7
61OPR    0    13
62JPC    0    71
63LOD    1     7
64LOD    0     3
65OPR    0     3
66STO    1     7
67LOD    1     6
68LIT    0     1
69OPR    0     2
70STO    1     6
71JMP    0    47
72OPR    0     0
74INT    0     5
75LOD    1     3
76STO    0     3
```

```
77LOD    1    4
78STO    0    4
79LOD    0    3
80LOD    0    4
81OPR    0    9
82JPC    0  100
83LOD    0    3
84LOD    0    4
85OPR    0   10
86JPC    0   91
87LOD    0    4
88LOD    0    3
89OPR    0    3
90STO    0    4
91LOD    0    4
92LOD    0    3
93OPR    0   10
94JPC    0   99
95LOD    0    3
96LOD    0    4
97OPR    0    3
98STO    0    3
99JMP    0   79
100LOD    0    3
101STO    1    5
102OPR    0    0
103INT    0    8
104LIT    0    7
105STO    0    3
106LIT    0   85
107STO    0    4
108CAL    0    2
109LIT    0   25
110STO    0    3
111LIT    0    3
112STO    0    4
113CAL    0   31
```

```
114LIT    0   84
115STO    0    3
116LIT    0   36
117STO    0    4
118CAL    0   74
119OPR    0    0
```

A.5 输出中间结果 or

```
7
85
7
85
0
7
14
42
28
21
35
56
10
112
5
147
224
2
448
1
595
896
0
25
3
25
0
3
6
12
24
48
0
24
1
1
```

2

12

4

6

8

3

84

36

84

36

48

12

24

12

12

B Experiment 2

B.1 编译程序 compiler.pas

```
Program  PL0 ;
{带有代码生成的PL0编译程序}

Const
  norw = 13; {保留字的个数}
  txmax = 100; {标识符表长度}
  nmax = 14; {数字的最大位数}
  al = 10; {标识符的长度}
  amax = 2047; {最大地址}
  levmax = 3; {程序体嵌套的最大深度}
  cxmax = 200; {代码数组的大小}

Type
  symbol = (nul, ident, number, plus, minus, times, slash, oddsym,
            eql, neq, lss, leq, gtr, geq, lparen, rparen, comma,
            semicolon, period, becomes, beginsym, endsym, ifsym,
            thensym, whilesym, dosym, callsym, constsym, varsym,
            procsym, redsym, wrtsym);
  alfa = packed array [1..al] Of char;
  m_object = (constant, variable, m_procedure);
  symset = set Of symbol;
  fct = (lit, opr, lod, sto, cal, int, jmp, jpc, red, wrt); {func}
  instruction = packed Record
    f : fct;  {功能码}
    l : 0..levmax; {相对层数}
    a : 0..amax; {相对地址}
  End;

{LIT 0,a : 取常数a
  OPR 0,a : 执行运算a
  LOD l,a : 取层差为l的层  相对地址为a的变量
  STO l,a : 存到层差为l的层  相对地址为a的变量
  CAL l,a : 调用层差为l的过程
  INT 0,a : t寄存器增加a
  JMP 0,a : 转移到指令地址a处
  JPC 0,a : 条件转移到指令地址a处 }

Var
  ch : char; {最近读到的字符}
  sym : symbol; {最近读到的符号}
  id : alfa; {最近读到的标识符}
  num : integer; {最近读到的数}
  cc : integer; {当前行的字符计数}
  ll : integer; {当前行的长度}
  kk, err : integer;
  cx : integer; {代码数组的当前下标}
  line : array [1..81] Of char;
  a : alfa;
  code : array [0..cxmax] Of instruction;
  word : array [1..norw] Of alfa;
  wsym : array [1..norw] Of symbol;
  ssym : array [char] Of symbol;
  mnemonic : array [fct] Of packed array [1..5] Of char;
  declbegsys, statbegsys, facbegsys : symset;
  table : array [0..txmax] Of
          Record
            name : alfa;
            Case kind : m_object Of
              constant : (val : integer);
              variable, m_procedure : (level, adr : integer)
          End;
  sfin, sfout, ifout, rfout: text;
  sfinp, sfoutp, ifoutp, rfoutp: string;
Procedure error (n : integer);
Begin
  writeln(sfout, '****', '␣' : cc-1, '↑', n : 2);
  err := err + 1
End {error};
Procedure getsym;

Var  i, j, k : integer;
```

```
Procedure getch;
Begin
  If cc = ll Then
    Begin
      If eof(sfin) Then
        Begin
          write('PROGRAM␣INCOMPLETE');
          close(sfin);
          close(sfout);
          close(ifout);
          close(rfout);
          exit;
        End;
      ll := 0;
      cc := 0;
      write(sfout, cx : 5, '␣');
      While Not eoln(sfin) Do
        Begin
          ll := ll + 1;
          read(sfin, ch);
          write(sfout, ch);
          line[ll] := ch
        End;
      writeln(sfout);
      readln(sfin);
      ll := ll + 1;
      line[ll] := '␣';
    End;
  cc := cc + 1;
  ch := line[cc]
End {getch};
Begin {getsym}
  While ch = '␣' Do
    getch;
  If ch In ['a'..'z'] Then
    Begin {标识符或保留字}
      k := 0;
      Repeat
        If k < al Then
          Begin
            k := k + 1;
            a[k] := ch
          End;
        getch
      Until Not (ch In ['a'..'z', '0'..'9']);
      If k >= kk  Then kk := k
      Else
        Repeat
          a[kk] := '␣';
          kk := kk-1
        Until kk = k;
      id := a;
      i := 1;
      j := norw;
      Repeat
        k := (i+j) Div 2;
        If id <= word[k] Then j := k-1;
        If id >= word[k] Then i := k + 1
      Until i > j;
      If i-1 > j Then sym := wsym[k]
      Else sym := ident
    End
  Else
    If ch In ['0'..'9'] Then
      Begin {数字}
        k := 0;
        num := 0;
        sym := number;
        Repeat
          num := 10*num + (ord(ch)-ord('0'));
          k := k + 1;
          getch;
        Until Not (ch In ['0'..'9']);
        If k > nmax Then error(30)
```

```
         End
   Else
      If ch = ':' Then
         Begin
            getch;
            If ch = '=' Then
               Begin
                  sym := becomes;
                  getch
               End
            Else  sym := nul;
         End
   Else
      Begin
         sym := ssym[ch];
         getch
      End
End {getsym};
Procedure  gen(x : fct; y, z : integer);
Begin
   If cx > cxmax Then
      Begin
         write('PROGRAM␣TOO␣LONG');
         close(sfin);
         close(sfout);
         close(ifout);
         close(rfout);
         exit;
      End;
   With code[cx] Do
      Begin
         f := x;
         l := y;
         a := z
      End;
   cx := cx + 1
End {gen};
```

```
Procedure test(s1, s2 : symset; n : integer);
Begin
  If Not (sym In s1) Then
    Begin
      error(n);
      s1 := s1 + s2;
      While Not (sym In s1) Do
        getsym
    End
End {test};
Procedure block(lev, tx : integer; fsys : symset);

Var
  dx : integer; {本过程数据空间分配下标}
  tx0 : integer; {本过程标识表起始下标}
  cx0 : integer; {本过程代码起始下标}
Procedure enter(k : m_object);
Begin {把m_object填入符号表中}
  tx := tx +1;
  With table[tx] Do
    Begin
      name := id;
      kind := k;
      Case k Of
        constant :
                  Begin
                    If num > amax Then
                      Begin
                        error(30);
                        num := 0
                      End;
                    val := num
                  End;
        variable :
                  Begin
                    level := lev;
                    adr := dx;
                    dx := dx +1;
                  End;
        m_procedure : level := lev
      End
    End
End {enter};
Function position(id : alfa) : integer;

Var i : integer;
Begin {在标识符表中查标识符id}
  table[0].name := id;
  i := tx;
  While table[i].name <> id Do
    i := i-1;
  position := i
End {position};
Procedure constdeclaration;
Begin
  If sym = ident Then
    Begin
      getsym;
      If sym In [eql, becomes] Then
        Begin
          If sym = becomes Then error(1);
          getsym;
          If sym = number Then
            Begin
              enter(constant);
              getsym
            End
          Else error(2)
        End
      Else error(3)
    End
  Else error(4)
End {constdeclaration};
Procedure vardeclaration;
```

```
Begin
  If sym = ident Then
    Begin
      enter(variable);
      getsym
    End
  Else error(4)
End {vardeclaration};
Procedure  listcode;

Var  i : integer;
Begin  {列出本程序体生成的代码}
  For i := cx0 To cx-1 Do
    With code[i] Do
      writeln(ifout, i, mnemonic[f] : 5, l : 3, a : 5)
End {listcode};
Procedure  statement(fsys : symset);

Var  i, cx1, cx2 : integer;
Procedure  expression(fsys : symset);

Var  addop : symbol;
Procedure  term(fsys : symset);

Var  mulop : symbol;
Procedure  factor(fsys : symset);

Var i : integer;
Begin
  test(facbegsys, fsys, 24);
  While sym In facbegsys Do
    Begin
      If sym = ident Then
        Begin
          i := position(id);
          If i = 0 Then error(11)
          Else
```

```
            With table[i] Do
              Case kind Of
                constant : gen(lit, 0, val);
                variable : gen(lod, lev-level, adr);
                m_procedure : error(21)
              End;
          getsym
        End
      Else
        If sym = number Then
          Begin
            If num > amax Then
              Begin
                error(30);
                num := 0
              End;
            gen(lit, 0, num);
            getsym
          End
      Else
        If sym = lparen Then
          Begin
            getsym;
            expression([rparen]+fsys);
            If sym = rparen Then getsym
            Else error(22)
          End;
      test(fsys, [lparen], 23)
    End
End {factor};
Begin {term}
  factor(fsys+[times, slash]);
  While sym In [times, slash] Do
    Begin
      mulop := sym;
      getsym;
      factor(fsys+[times, slash]);
```

```
        If mulop = times Then gen(opr, 0, 4)
        Else gen(opr, 0, 5)
      End
End {term};
Begin {expression}
  If sym In [plus, minus] Then
    Begin
      addop := sym;
      getsym;
      term(fsys+[plus, minus]);
      If addop = minus Then gen(opr, 0, 1)
    End
  Else term(fsys+[plus, minus]);
  While sym In [plus, minus] Do
    Begin
      addop := sym;
      getsym;
      term(fsys+[plus, minus]);
      If addop = plus Then gen(opr, 0, 2)
      Else gen(opr, 0, 3)
    End
End {expression};
Procedure condition(fsys : symset);

Var relop : symbol;
Begin
  If sym = oddsym Then
    Begin
      getsym;
      expression(fsys);
      gen(opr, 0, 6)
    End
  Else
    Begin
      expression([eql, neq, lss, gtr, leq, geq] + fsys);
      If Not (sym In [eql, neq, lss, leq, gtr, geq]) Then
        error(20)
      Else
        Begin
          relop := sym;
          getsym;
          expression(fsys);
          Case relop Of
            eql : gen(opr, 0, 8);
            neq : gen(opr, 0, 9);
            lss : gen(opr, 0, 10);
            geq : gen(opr, 0, 11);
            gtr : gen(opr, 0, 12);
            leq : gen(opr, 0, 13);
          End
        End
    End
End {condition};
Begin {statement}
  If sym = ident Then
    Begin
      i := position(id);
      If i = 0 Then error(11)
      Else
        If table[i].kind <> variable Then
          Begin {对非变量赋值}
            error(12);
            i := 0;
          End;
      getsym;
      If sym = becomes Then getsym
      Else error(13);
      expression(fsys);
      If i <> 0 Then
        With table[i] Do
          gen(sto, lev-level, adr)
    End
  Else
    If sym = callsym Then
      Begin
        getsym;
        If sym <> ident Then error(14)
        Else
          Begin
            i := position(id);
            If i = 0 Then error(11)
            Else
              With table[i] Do
                If kind = m_procedure Then
                  gen(cal, lev-level, adr)
                Else error(15);
            getsym
          End
      End
  Else
    If sym = ifsym Then
      Begin
        getsym;
        condition([thensym, dosym]+fsys);
        If sym = thensym Then getsym
        Else error(16);
        cx1 := cx;
        gen(jpc, 0, 0);
        statement(fsys);
        code[cx1].a := cx
      End
  Else
    If sym = beginsym Then
      Begin
        getsym;
        statement([semicolon, endsym]+fsys);
        While sym In [semicolon]+statbegsys Do
          Begin
            If sym = semicolon Then getsym
            Else error(10);
            statement([semicolon, endsym]+fsys)
          End;
        If sym = endsym Then getsym
        Else error(17)
      End
  Else
    If sym = whilesym Then
      Begin
        cx1 := cx;
        getsym;
        condition([dosym]+fsys);
        cx2 := cx;
        gen(jpc, 0, 0);
        If sym = dosym Then getsym
        Else error(18);
        statement(fsys);
        gen(jmp, 0, cx1);
        code[cx2].a := cx
      End
  Else
    If sym = redsym Then
      Begin
        getsym;
        If sym = lparen Then
          Repeat
            getsym;
            If sym = ident Then
              Begin
                i := position(id);
                If i = 0 Then error(11)
                Else If table[i].kind <> variable Then
                         Begin
                           error(12);
                           i := 0
                         End
                Else With table[i] Do
                         gen(red,lev-level,adr)
              End
              Else error(4);
              getsym;
            Until sym <> comma
          Else error(40);
          If sym <> rparen Then error(22);
          getsym
        End
    Else
      If sym = wrtsym Then
        Begin
          getsym;
          If sym = lparen
            Then
            Begin
              Repeat
                getsym;
                expression([rparen,comma]+fsys);
                gen(wrt,0,0);
              Until sym <> comma;
              If sym <> rparen
                Then error(22);
              getsym
            End
          Else error(40)
        End;
    test(fsys, [ ], 19)
  End {statement};
  Begin {block}
    dx := 3;
    tx0 := tx;
    table[tx].adr := cx;
    gen(jmp, 0, 0);
    If lev > levmax Then error(32);
    Repeat
      If sym = constsym Then
        Begin
          getsym;
      Repeat
        constdeclaration;
        While sym = comma Do
          Begin
            getsym;
            constdeclaration
          End;
        If sym = semicolon Then getsym
        Else error(5)
      Until sym <> ident
    End;
  If sym = varsym Then
    Begin
      getsym;
      Repeat
        vardeclaration;
        While sym = comma Do
          Begin
            getsym;
            vardeclaration
          End;
        If sym = semicolon Then getsym
        Else error(5)
      Until sym <> ident;
    End;
  While sym = procsym Do
    Begin
      getsym;
      If sym = ident Then
        Begin
          enter(m_procedure);
          getsym
        End
      Else error(4);
      If sym = semicolon Then getsym
      Else error(5);
      block(lev+1, tx, [semicolon]+fsys);
```

```
        If sym = semicolon Then
          Begin
            getsym;
            test(statbegsys+[ident, procsym], fsys, 6)
          End
        Else error(5)
      End;
    test(statbegsys+[ident], declbegsys, 7)
  Until Not (sym In declbegsys);
  code[table[tx0].adr].a := cx;
  With table[tx0] Do
    Begin
      adr := cx; {代码开始地址}
    End;
  cx0 := cx;
  gen(int, 0, dx);
  statement([semicolon, endsym]+fsys);
  gen(opr, 0, 0); {生成返回指令}
  test(fsys, [ ], 8);
  listcode;
End {block};
Procedure interpret;

Const stacksize = 500;

Var p, b, t : integer;
  {程序地址寄存器, 基地址寄存器,栈顶地址寄存器}
  i : instruction; {指令寄存器}
  s : array [1..stacksize] Of integer; {数据存储栈}
Function base(l : integer) : integer;

Var b1 : integer;
Begin
  b1 := b; {顺静态链求层差为 l 的层的基地址}
  While l > 0 Do
    Begin
      b1 := s[b1];
```

```
          l := l-1
        End;
    base := b1
End {base};
Begin
    writeln('START␣PL/0');
    t := 0;
    b := 1;
    p := 0;
    s[1] := 0;
    s[2] := 0;
    s[3] := 0;
    Repeat
        i := code[p];
        p := p+1;
        With i Do
            Case f Of
                lit :
                          Begin
                              t := t+1;
                              s[t] := a
                          End;
                opr : Case a Of {运算}
                              0 :
                                    Begin {返回}
                                        t := b-1;
                                        p := s[t+3];
                                        b := s[t+2];
                                    End;
                              1 : s[t] := -s[t];
                              2 :
                                    Begin
                                        t := t-1;
                                        s[t] := s[t] + s[t+1]
                                    End;
                              3 :
                                    Begin
```

```
      t := t-1;
      s[t] := s[t]-s[t+1]
    End;
4 :
    Begin
      t := t-1;
      s[t] := s[t] * s[t+1]
    End;
5 :
    Begin
      t := t-1;
      s[t] := s[t] Div s[t+1]
    End;
6 : s[t] := ord(odd(s[t]));
8 :
    Begin
      t := t-1;
      s[t] := ord(s[t] = s[t+1])
    End;
9:
   Begin
     t := t-1;
     s[t] := ord(s[t] <> s[t+1])
   End;
10 :
     Begin
       t := t-1;
       s[t] := ord(s[t] < s[t+1])
     End;
11:
    Begin
      t := t-1;
      s[t] := ord(s[t] >= s[t+1])
    End;
12 :
     Begin
       t := t-1;
       s[t] := ord(s[t] > s[t+1])
      End;
    13 :
      Begin
       t := t-1;
       s[t] := ord(s[t] <= s[t+1])
      End;
   End;
lod :
   Begin
    t := t + 1;
    s[t] := s[base(l) + a]
   End;
sto :
   Begin
    s[base(l) + a] := s[t];
    writeln(rfout, s[t]);
    t := t-1
   End;
cal :
   Begin {generate new block mark}
    s[t+1] := base( l );
    s[t+2] := b;
    s[t+3] := p;
    b := t+1;
    p := a
   End;
int : t := t + a;
jmp : p := a;
jpc :
   Begin
    If s[t] = 0 Then p := a;
    t := t-1;
   End;
red :
   Begin
    writeln('Running␣program␣ask␣for␣input:␣');
```

```
               readln(s[base(l) + a]);
             End;
        wrt :
             Begin
               writeln(s[t]);
               t := t + 1
             End
      End {with, case}
  Until p = 0;
  write('END␣PL/0');
End {interpret};

Begin  {主程序}
  writeln('Type␣in␣the␣path␣to␣your␣source␣code:␣');
  readln(sfinp);
  writeln('Type␣in␣path␣of␣the␣file␣to␣output␣source␣program␣at');
  readln(sfoutp);
  writeln('Type␣in␣path␣of␣the␣file␣to␣output␣intermediate␣program␣at');
  readln(ifoutp);
  writeln('Type␣in␣path␣of␣the␣file␣to␣output␣runtime␣data␣at');
  readln(rfoutp);
  assign(sfin, sfinp);
  assign(sfout, sfoutp);
  assign(ifout, ifoutp);
  assign(rfout, rfoutp);
  reset(sfin);
  rewrite(sfout);
  rewrite(ifout);
  rewrite(rfout);
  For ch := 'A' To ';' Do
    ssym[ch] := nul;

  word[1] := 'begin␣␣␣␣␣';
  word[2] := 'call␣␣␣␣␣␣';
  word[3] := 'const␣␣␣␣␣';
  word[4] := 'do␣␣␣␣␣␣␣␣';
  word[5] := 'end␣␣␣␣␣␣␣';
```

```
word[6] := 'if␣␣␣␣␣␣␣␣';
word[7] := 'odd␣␣␣␣␣␣␣';
word[8] := 'procedure␣';
word[9] := 'read␣␣␣␣␣␣';
word[10] := 'then␣␣␣␣␣␣';
word[11] := 'var␣␣␣␣␣␣␣';
word[12] := 'while␣␣␣␣␣';
word[13] := 'write␣␣␣␣␣';

wsym[1] := beginsym;
wsym[2] := callsym;
wsym[3] := constsym;
wsym[4] := dosym;
wsym[5] := endsym;
wsym[6] := ifsym;
wsym[7] := oddsym;
wsym[8] := procsym;
wsym[9] := redsym;
wsym[10] := thensym;
wsym[11] := varsym;
wsym[12] := whilesym;
wsym[13] := wrtsym;

ssym['+'] := plus;
ssym['-'] := minus;
ssym['*'] := times;
ssym['/'] := slash;
ssym['('] := lparen;
ssym[')'] := rparen;
ssym['='] := eql;
ssym[','] := comma;
ssym['.'] := period;
ssym['&'] := neq;
ssym['<'] := lss;
ssym['>'] := gtr;
ssym[';'] := semicolon;
ssym['%'] := leq;
```

```
  mnemonic[lit] := 'LIT␣␣';
  mnemonic[opr] := 'OPR␣␣';
  mnemonic[lod] := 'LOD␣␣';
  mnemonic[sto] := 'STO␣␣';
  mnemonic[cal] := 'CAL␣␣';
  mnemonic[int] := 'INT␣␣';
  mnemonic[jmp] := 'JMP␣␣';
  mnemonic[jpc] := 'JPC␣␣';
  mnemonic[red] := 'RED␣␣';
  mnemonic[wrt] := 'WRT␣␣';

  declbegsys := [constsym, varsym, procsym];
  statbegsys := [beginsym, callsym, ifsym, whilesym];
  facbegsys := [ident, number, lparen];
  err := 0;
  cc := 0;
  cx := 0;
  ll := 0;
  ch := '␣';
  kk := al;
  getsym;
  block(0, 0, [period]+declbegsys+statbegsys);
  If sym <> period Then error(9);
  If err = 0 Then interpret
  Else write('ERRORS␣IN␣PL/0␣PROGRAM');
  writeln;
  close(sfin);
  close(sfout);
  close(ifout);
  close(rfout);
  exit;
End.
```

B.2 输入源程序 source.pl0

```
var x, y, z, q, r;
procedure multiply;
var a, b;
begin
  a := x;  b := y;  z := 0;
  while b > 0 do
  begin
    if odd b then z := z + a;
    a := 2*a ;  b := b/2 ;
  end
end;
procedure divide;
var w;
begin
  r := x;  q := 0;  w := y;
  while w % r do w := 2*w ;
  while w > y do
  begin
    q := 2*q;
    w := w/2;
    if w % r then
    begin
      r := r-w;
      q := q+1;
    end
  end
end;
procedure gcd;
var f, g ;
begin
  f := x;  g := y;
  while f & g do
  begin
    if f < g then g := g-f;
    if g < f then f := f-g;
  end;
```

```
   z := f
end;
begin
      read(x); read(y);
      call multiply;
      write(x, y, z);
      read(x); read(y);
      call divide;
      write(x, y, q);
      read(x); read(y);
      call gcd;
      write(x, y, z);
end.
```

B.3 标准输入 stdin

```
source.pl0
os
oi
or
7
85
25
3
84
36
```

B.4 输出源程序 os

```
 0 var  x, y, z, q, r;
 1 procedure  multiply;
 1 var  a, b;
 2 begin
 3   a := x;  b := y;  z := 0;
 9   while b > 0 do
13   begin
13     if odd b then z := z + a;
20     a := 2*a ;  b := b/2 ;
28   end
28 end;
30 procedure  divide;
30 var  w;
31 begin
32   r := x;  q := 0;  w := y;
38   while w % r do w := 2*w ;
47   while w > y do
51   begin
51     q := 2*q;
55     w := w/2;
59     if w % r then
62     begin
63       r := r-w;
67       q := q+1;
71     end
71   end
71 end;
73 procedure  gcd;
73 var  f, g ;
74 begin
75   f := x;  g := y;
79   while f & g do
83   begin
83     if f < g then g := g-f;
91     if g < f then f := f-g;
99   end;
```

```
100    z := f
101 end;
103 begin
104     read(x); read(y);
106     call multiply;
107     write(x, y, z);
113     read(x); read(y);
115     call divide;
116     write(x, y, q);
122     read(x); read(y);
124     call gcd;
125     write(x, y, z);
131 end.
```

B.5 输出中间代码 oi

```
2INT     0    5
3LOD     1    3
4STO     0    3
5LOD     1    4
6STO     0    4
7LIT     0    0
8STO     1    5
9LOD     0    4
10LIT     0    0
11OPR     0   12
12JPC     0   29
13LOD     0    4
14OPR     0    6
15JPC     0   20
16LOD     1    5
17LOD     0    3
18OPR     0    2
19STO     1    5
20LIT     0    2
21LOD     0    3
22OPR     0    4
23STO     0    3
24LOD     0    4
25LIT     0    2
26OPR     0    5
27STO     0    4
28JMP     0    9
29OPR     0    0
31INT     0    4
32LOD     1    3
33STO     1    7
34LIT     0    0
35STO     1    6
36LOD     1    4
37STO     0    3
38LOD     0    3
```

```
39LOD   1   7
40OPR   0  13
41JPC   0  47
42LIT   0   2
43LOD   0   3
44OPR   0   4
45STO   0   3
46JMP   0  38
47LOD   0   3
48LOD   1   4
49OPR   0  12
50JPC   0  72
51LIT   0   2
52LOD   1   6
53OPR   0   4
54STO   1   6
55LOD   0   3
56LIT   0   2
57OPR   0   5
58STO   0   3
59LOD   0   3
60LOD   1   7
61OPR   0  13
62JPC   0  71
63LOD   1   7
64LOD   0   3
65OPR   0   3
66STO   1   7
67LOD   1   6
68LIT   0   1
69OPR   0   2
70STO   1   6
71JMP   0  47
72OPR   0   0
74INT   0   5
75LOD   1   3
76STO   0   3
```

```
77LOD     1     4
78STO     0     4
79LOD     0     3
80LOD     0     4
81OPR     0     9
82JPC     0   100
83LOD     0     3
84LOD     0     4
85OPR     0    10
86JPC     0    91
87LOD     0     4
88LOD     0     3
89OPR     0     3
90STO     0     4
91LOD     0     4
92LOD     0     3
93OPR     0    10
94JPC     0    99
95LOD     0     3
96LOD     0     4
97OPR     0     3
98STO     0     3
99JMP     0    79
100LOD     0     3
101STO     1     5
102OPR     0     0
103INT     0     8
104RED     0     3
105RED     0     4
106CAL     0     2
107LOD     0     3
108WRT     0     0
109LOD     0     4
110WRT     0     0
111LOD     0     5
112WRT     0     0
113RED     0     3
```

```
114RED      0     4
115CAL      0    31
116LOD      0     3
117WRT      0     0
118LOD      0     4
119WRT      0     0
120LOD      0     6
121WRT      0     0
122RED      0     3
123RED      0     4
124CAL      0    74
125LOD      0     3
126WRT      0     0
127LOD      0     4
128WRT      0     0
129LOD      0     5
130WRT      0     0
131OPR      0     0
```

B.6 输出中间结果 or

```
7
85
0
7
14
42
28
21
35
56
10
112
5
147
224
2
448
1
595
896
0
25
0
3
6
12
24
48
0
24
1
1
2
12
4
6
```

8

3

84

36

48

12

24

12

12

B.7 标准输出 stdout

```
Type in the path to your source code:
Type in path of the file to output source program at
Type in path of the file to output intermediate program at
Type in path of the file to output runtime data at
START PL/0
Running program ask for input:
Running program ask for input:
7
85
595
Running program ask for input:
Running program ask for input:
25
3
8
Running program ask for input:
Running program ask for input:
84
36
12
END PL/0
```