Отчет к лабораторной работе №2

Круглов В.А.

ClassesUnit.pas

```
unit ClassesUnit;

interface

type
  Symbol = record
    SymbolName: String;
    ParamValue: longword;
  end;
  Literal = record
    LiteralName: String;
    LiteralValue: Variant;
  end;

  TASMCommandM = class
  private
    CommandName: String;
    CommandLength: byte;
    CommandParams: longword;
  public
    constructor Create(aName: String; aParams: longword; aLength: byte); overload;
    function    GetLength: byte; inline;
    function    GetHash: byte; inline;
    function    Compare(aName: String; aParam: longword): boolean; inline;
    class function Hash(aName: String; aParams: longword): byte;
  end;

  TASMCommandP = class
  private
    CommandName: String;
    CommandAction: byte;
    CommandParams: longword;
  public
    constructor Create(aName: String; aParams: longword; aAction: byte); overload;
    function    GetAction: byte; inline;
    function    GetHash: byte; inline;
    function    GetParams: longword; inline;
    function    Compare(aName: String): boolean; inline;
    class function Hash(aName: String): byte; inline;
  end;

  TMCommandHashTable = class
  private
    Table: array of TASMCommandM;
    Names: array of String;
    Size: byte;
  public
    constructor Create(aSize: byte); overload;
    procedure AddCommand(aName: String; aParams: longword; aLength: byte);
    function  SearchCommand(aName: String; aParams: longword): TASMCommandM;
    function  isCommand(aName: String): boolean;
  end;

  TPCommandHashTable = class
  private
    Table: array of TASMCommandP;
    Names: array of String;
    Size: byte;
  public
    constructor Create(aSize: byte); overload;
    procedure AddCommand(aName: String; aParams: longword; aLength: byte);
    function  SearchCommand(aName: String): TASMCommandP;
    function  isCommand(aName: String): boolean;
  end;

implementation

uses SysUtils;

{ TASMCommand }

function TASMCommandM.Compare(aName: String; aParam: longword): boolean;
begin
  if CommandParams xor aParam = 0 then
    Result:=AnsiSameStr(aName, CommandName)
  else
    Result:=false;
end;

constructor TASMCommandM.Create(aName: String; aParams: longword; aLength: byte);
begin
  Create;
  CommandName:=aName;
  CommandParams:=aParams;
  CommandLength:=aLength;
end;

function TASMCommandM.GetHash: byte;
begin
  Result:=Hash(CommandName,CommandParams);
end;

function TASMCommandM.GetLength: byte;
begin
  Result:=CommandLength;
end;

class function TASMCommandM.Hash(aName: String; aParams: longword): byte;
var
  i,hashc: Integer;
  hashs: int64;
begin
  hashc:=(aParams div $00010000) xor (aParams and $FFFF);
  hashs:=0;
  for i := 1 to Length(aName) do
    Inc(hashs,byte(aName[i]) xor hashc);
  i:=2;
  while hashs>$FF do
  begin
    hashs:=hashs div i;
    inc(i);
  end;
  Result:=hashs xor hashc;
end;

{ TCommandHashTable }

procedure TMCommandHashTable.AddCommand(aName: String; aParams: longword; aLength: byte);
var
  ASMCommand: TASMCommandM;
  h: byte;
begin
  ASMCommand:=TASMCommandM.Create(aName, aParams, aLength);
  h:=ASMCommand.GetHash mod Size;
  while Table[h]<>nil do
  begin
    Inc(h);
    if h>=Size then h:=0;
  end;
  Table[h]:=ASMCommand;
  h:=0;
  while Names[h]<>'' do
  begin
    Inc(h);
  end;
  Names[h]:=aName;
end;

constructor TMCommandHashTable.Create(aSize: byte);
var
  i: Integer;
begin
  Create;
  SetLength(Table,aSize);
  SetLength(Names,aSize);
  for i := 0 to aSize - 1 do
  begin
    Table[i]:=nil;
    Names[i]:='';
  end;
  Size:=aSize;
end;

function TMCommandHashTable.isCommand(aName: String): boolean;
var
  i: byte;
begin
  Result:=false;
  i:=0;
  while Names[i]<>'' do
  begin
    if AnsiSameStr(aName, Names[i]) then
    begin
      Result:=true;
      Exit;
    end;
    Inc(i);
  end;
end;

function TMCommandHashTable.SearchCommand(aName: String; aParams: longword): TASMCommandM;
var
  h: byte;
  ASMCommand: TASMCommandM;
begin
  h:=TASMCommandM.Hash(aName, aParams) mod Size;
  ASMCommand:=Table[h];
  while (ASMCommand<>nil)and(not ASMCommand.Compare(aName, aParams)) do
  begin
    Inc(h);
    if h>=Size then h:=0;
    ASMCommand:=Table[h];
  end;
  Result:=ASMCommand;
end;

{ TASMCommandP }

function TASMCommandP.Compare(aName: String): boolean;
begin
  Result:=AnsiSameStr(aName,CommandName);
end;

constructor TASMCommandP.Create(aName: String; aParams: longword; aAction: byte);
begin
  Create;
  CommandName:=aName;
  CommandAction:=aAction;
  CommandParams:=aParams;
end;
```

```
function TASMCommandP.GetHash: byte;
begin
  Result:=Hash(CommandName);
end;

function TASMCommandP.GetParams: longword;
begin
  Result:=CommandParams;
end;

function TASMCommandP.GetAction: byte;
begin
  Result:=CommandAction;
end;

class function TASMCommandP.Hash(aName: String): byte;
begin
  Result:=TASMCommandM.Hash(aName, $00000000);
end;

{ TPCommandHashTable }

procedure TPCommandHashTable.AddCommand(aName: String; aParams: longword; aLength: byte);
var
  ASMCommand: TASMCommandP;
  h: byte;
begin
  ASMCommand:=TASMCommandP.Create(aName, aParams, aLength);
  h:=ASMCommand.GetHash mod Size;
  while Table[h]<>nil do
  begin
    Inc(h);
    if h>=Size then h:=0;
  end;
  Table[h]:=ASMCommand;
  h:=0;
  while Names[h]<>'' do
  begin
    Inc(h);
  end;
  Names[h]:=aName;
end;

constructor TPCommandHashTable.Create(aSize: byte);
var
  i: Integer;
begin
  Create;
  SetLength(Table,aSize);
  SetLength(Names,aSize);
  for i := 0 to aSize - 1 do
  begin
    Table[i]:=nil;
    Names[i]:='';
  end;
  Size:=aSize;
end;

function TPCommandHashTable.isCommand(aName: String): boolean;
var
  i: byte;
begin
  Result:=false;
  i:=0;
  while Names[i]<>'' do
  begin
    if AnsiSameStr(aName, Names[i]) then
    begin
      Result:=true;
      Exit;
    end;
    Inc(i);
  end;
end;

function TPCommandHashTable.SearchCommand(aName: String): TASMCommandP;
var
  h: byte;
  ASMCommand: TASMCommandP;
begin
  h:=TASMCommandP.Hash(aName) mod Size;
  ASMCommand:=Table[h];
  while (ASMCommand<>nil)and(not ASMCommand.Compare(aName)) do
  begin
    Inc(h);
    if h>=Size then h:=0;
    ASMCommand:=Table[h];
  end;
  Result:=ASMCommand;
end;

end.
```

Main.pas

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, StdCtrls, Buttons, ExtCtrls, ClassesUnit;

const
  cDerictive = $00;
  cMCommand = $0F;
  cPCommand = $F0;
  cSymbol = $FE;
  cUnknown = $FF;

type
  TMainForm = class(TForm)
    GridPanel1: TGridPanel;
    Panel1: TPanel;
    Panel3: TPanel;
    btnOpenFile: TSpeedButton;
    eFileName: TEdit;
    mFile: TMemo;
    Panel4: TPanel;
    btnAction: TButton;
    GridPanel2: TGridPanel;
    Panel2: TPanel;
    Label1: TLabel;
    lvCommand: TListView;
    Panel5: TPanel;
    Label2: TLabel;
    lvDerective: TListView;
    Panel6: TPanel;
    Label3: TLabel;
    lvSymbol: TListView;
    Panel7: TPanel;
    Label4: TLabel;
    lvLiteral: TListView;
    OpenDialog: TOpenDialog;
    procedure btnOpenFileClick(Sender: TObject);
    procedure btnActionClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    MOT: TMCommandHashTable;
    POT: TPCommandHashTable;
    ReserveTable: array [0..34] of Symbol;
    SymbolTable: array [0..$FF] of Symbol;
    LiteralTable: array [0..$FF] of Literal;
    SymbolCounter: byte;
    LiteralCounter: byte;
    function AddSymbol(aSymbol: String; Value: longword = $00): boolean;
    function SetSymbol(aSymbol: String; Value: longword; sType: byte): boolean;
    function GetSymbol(aSymbol: String): longword;
    function IsSymbol(aSymbol: String): boolean; inline;
    function Analyze(aString: String): byte;
    function GetParam(pString: String): longword;
    class function Min(i1,i2: integer): integer; inline;
    class function VarTypeToStr(basicType: integer): String;
    procedure ShowCommand(Command: String; Params: longword; aLength: byte);
    procedure ShowPCommand(Command: String; aAction: byte);
    procedure AddLiteral(aValue: Variant);
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.dfm}

procedure TMainForm.AddLiteral(aValue: Variant);
var
  i: byte;
  ListItem: TListItem;
begin
  if LiteralCounter > 0 then
    for i := 0 to LiteralCounter - 1 do
      if LiteralTable[i].LiteralValue=aValue then
        Exit;
  LiteralTable[LiteralCounter].LiteralName := '~'+
    VarTypeToStr(VarType(aValue) and VarTypeMask) + IntToHex(LiteralCounter, 2);
  LiteralTable[LiteralCounter].LiteralValue := aValue;
  ListItem := lvLiteral.Items.Add;
  ListItem.Caption := LiteralTable[LiteralCounter].LiteralName;
  case (VarType(aValue) and VarTypeMask) of
    2..3,17..20:
      ListItem.SubItems.Add(IntToHex(aValue, 8)+' h');
  end;
  Inc(LiteralCounter);
end;

function TMainForm.AddSymbol(aSymbol: String; Value: longword): boolean;
var
  i: byte;
begin
  Result:=false;
  if SymbolCounter > 0 then
    for i := 0 to SymbolCounter - 1 do
      if AnsiSameStr(SymbolTable[i].SymbolName, aSymbol) then
        Exit;
  SymbolTable[SymbolCounter].SymbolName := aSymbol;
  SymbolTable[SymbolCounter].ParamValue := Value;
  Inc(SymbolCounter);
  Result := true;
end;

function TMainForm.SetSymbol(aSymbol: String; Value: longword;
  sType: byte): boolean;
var
  ListItem: TListItem;
  i: byte;
begin
  Result := true;
  if SymbolCounter > 0 then
    for i := 0 to SymbolCounter - 1 do
      if AnsiSameStr(SymbolTable[i].SymbolName, aSymbol) then
      begin
        SymbolTable[i].ParamValue := Value;
        ListItem := lvSymbol.Items.Add;
        ListItem.Caption := aSymbol;
        case sType of
          $00:
            ListItem.SubItems.Add('$ ' + IntToHex(Value, 8));
          $01:
            ListItem.SubItems.Add('R ' + IntToHex(Value, 8));
          $02:
            ListItem.SubItems.Add('# ' + IntToHex(Value, 8));
        end;
        Exit;
      end;
  Result := false;
end;

procedure TMainForm.ShowCommand(Command: String; Params: longword; aLength: byte);
var
  ListItem: TlistItem;
begin
  ListItem:=lvCommand.Items.Add;
  ListItem.Caption:=Command;
  ListItem.SubItems.Add(IntToHex(Params, 8));
  ListItem.SubItems.Add('$ '+IntToHex(aLength, 2));
end;

procedure TMainForm.ShowPCommand(Command: String; aAction: byte);
var
  ListItem: TlistItem;
begin
  ListItem:=lvDerective.Items.Add;
  ListItem.Caption:=Command;
  case aAction of
    $01: ListItem.SubItems.Add('Initiation');
    $00: ListItem.SubItems.Add('real p-Com');
  end;
end;

class function TMainForm.VarTypeToStr(basicType: integer): String;
begin
  case basicType of
    varEmpty     : Result := 'varEmpty';
    varNull      : Result := 'varNull';
    varSmallInt  : Result := 'varSmallInt';
    varInteger   : Result := 'varInteger';
    varSingle    : Result := 'varSingle';
    varDouble    : Result := 'varDouble';
    varCurrency  : Result := 'varCurrency';
    varDate      : Result := 'varDate';
    varOleStr    : Result := 'varOleStr';
    varDispatch  : Result := 'varDispatch';
    varError     : Result := 'varError';
    varBoolean   : Result := 'varBoolean';
    varVariant   : Result := 'varVariant';
    varUnknown   : Result := 'varUnknown';
    varByte      : Result := 'varByte';
    varWord      : Result := 'varWord';
    varLongWord  : Result := 'varLongWord';
    varInt64     : Result := 'varInt64';
    varStrArg    : Result := 'varStrArg';
    varString    : Result := 'varString';
    varAny       : Result := 'varAny';
    varTypeMask  : Result := 'varTypeMask';
  end;
end;
```

```
function TMainForm.Analyze(aString: String): byte;
var i: byte;
begin
  Result:=cSymbol;
  if MOT.isCommand(aString) then
    Result:=cMCommand;
  if POT.isCommand(aString) then
    Result:=cPCommand;
  if aString[1]='$' then
    Result:=cDerictive;
  if aString[1] in ['A'..'Z'] then
    for i := 2 to Length(aString) do
      if not (aString[i] in ['A'..'Z','0'..'9','.']) then
        Result:=cUnknown;
end;

procedure TMainForm.btnActionClick(Sender: TObject);
var
  i, j, pComm, pMet, p, pSpace, pTab: Integer;
  s, ss, sp: string;
  params,tmp: longword;
  counter: int64;
  ASMCommandM: TASMCommandM;
  ASMCommandP: TASMCommandP;
begin
  if mFile.Lines.Count = 0 then Exit;
  Counter:=0;
  SymbolCounter:=1;
  for i := 0 to mFile.Lines.Count - 1 do
  begin
    s:=UpperCase(Trim(mFile.Lines.Strings[i]));
    pComm:=Pos(';',s);
    if pComm>0 then
      s:=Trim(Copy(s,1,pComm-1));
    pMet:=Pos(':',s);
    if pMet>0 then
    begin
      ss:=Trim(Copy(s,1,pMet-1));
      s:=Trim(Copy(s,pMet+1,Length(s)-(pMet-1)));
      if not AddSymbol(ss,$80000000) then
      begin
        MessageDlg('Duplicate symbol: ' + ss, mtError, [mbOk], 0);
        Exit;
      end;
    end;
  end;
  for i := 0 to mFile.Lines.Count - 1 do
  begin
    s:=UpperCase(Trim(mFile.Lines.Strings[i]));
    pComm:=Pos(';',s);
    if pComm>0 then
      s:=Trim(Copy(s,1,pComm-1));
    pMet:=Pos(':',s);
    if pMet>0 then
    begin
      ss:=Trim(Copy(s,1,pMet-1));
      s:=Trim(Copy(s,pMet+1,Length(s)-(pMet-1)));
      if not SetSymbol(ss, counter + $80000000, $00) then
      begin
        MessageDlg('Unknown error with ''' + ss+''' symbol', mtError, [mbOk], 0);
        Exit;
      end;
    end;
    pSpace:=Pos(' ',s);
    pTab:=Pos(#$09,s);
    p:=Min(pSpace,pTab);
    if (p>0)or(Length(s)>0) then
    begin
      if p>0 then
        ss:=Trim(Copy(s,1,p-1))
```

```
else
  ss:=s;
case Analyze(ss) of
  cDerictive: ;
  cMCommand:
    begin
      if p>0 then
        s:=Trim(Copy(s,p+1,Length(s)-(p-1)))
      else
        s:='';
      params:=0;
      j:=Pos(',',s);
      while j>0 do
      begin
        sp:=Trim(Copy(s,1,j-1));
        s:=Trim(Copy(s,j+1,Length(s)-(j-1)));
        Inc(params,GetParam(sp));
        j:=Pos(',',s);
      end;
      Inc(params,GetParam(s));
      s:='';
      ASMCommandM:=MOT.SearchCommand(ss,params and $F00FFFFF);
      if ASMCommandM<>nil then
      begin
        Inc(Counter,ASMCommandM.GetLength);
        ShowCommand(ss,params,ASMCommandM.GetLength);
      end
      else
      begin
        MessageDlg('Unknown command params: '+IntToHex(params,8),mtError,[mbOk],0);
        Exit;
      end;
    end;
  cPCommand:
    begin
      ShowPCommand(ss,0);
    end;
  cSymbol:
    begin
      if p=0 then
      begin
        MessageDlg('Unknown command found: '+s,mtError,[mbOk],0);
        Exit;
      end;
      s:=Trim(Copy(s,p+1,Length(s)-(p-1)));
      pSpace:=Pos(' ',s);
      pTab:=Pos(#$09,s);
      j:=Min(pSpace,pTab);
      if j=0 then
      begin
        MessageDlg('Uncorrect params list for P-command: '+s,mtError,[mbOk],0);
        Exit;
      end;
      sp:=Trim(Copy(s,1,j-1));
      s:=Trim(Copy(s,j+1,Length(s)-(j-1)));
      ASMCommandP:=POT.SearchCommand(sp);
      if ASMCommandP=nil then
      begin
        MessageDlg('Unknown P-command: '+sp,mtError,[mbOk],0);
        Exit;
      end;
      if ASMCommandP.GetAction<>1 then
      begin
        MessageDlg('Uncorrect format for P-command: '+sp,mtError,[mbOk],0);
        Exit;
      end;
      if Length(s)=0 then
      begin
        MessageDlg('Unknown error for P-command: '+sp,mtError,[mbOk],0);
        Exit;
```

```
            end;
            params:= ASMCommandP.GetParams;
            case params of
              $00000000:
                begin
                  params:=params or GetParam(s);
                  s:='';
                  if (not AddSymbol(ss)) or (not SetSymbol(ss,params,$01)) then
                  begin
                    MessageDlg('Error on symbol: ' + ss, mtError, [mbOk], 0);
                    Exit;
                  end;
                end;
              $80000000:
                begin
                  if s[Length(s)]='H' then
                  begin
                    Move(s[1],s[2],Length(s));
                    s[1]:='$';
                  end;
                  params:=params or StrToInt(s);
                  s:='';
                  if (not AddSymbol(ss)) or (not SetSymbol(ss,params,$02)) then
                  begin
                    MessageDlg('Error on symbol: ' + ss, mtError, [mbOk], 0);
                    Exit;
                  end;
                  Inc(counter);
                end;
            end;
            ShowPCommand(sp,ASMCommandP.GetAction);
          end;
      else
        begin
          MessageDlg('Uncorrect symbol or unknown command: ' + ss, mtError, [mbOk], 0);
          Exit;
        end;
      end;
    end;
    //mFile.Lines.Strings[i]:=s;
  end;
end;

procedure TMainForm.btnOpenFileClick(Sender: TObject);
begin
  if OpenDialog.Execute then
  begin
    eFileName.Text := OpenDialog.FileName;
    mFile.Lines.LoadFromFile(OpenDialog.FileName);
    lvCommand.Clear;
    lvSymbol.Clear;
  end;
end;

procedure TMainForm.FormCreate(Sender: TObject);
var
  i: byte;
begin
  ReserveTable[00].SymbolName:='';      ReserveTable[00].ParamValue:=$00000000;
  ReserveTable[01].SymbolName:='A';     ReserveTable[01].ParamValue:=$00000001;
  ReserveTable[02].SymbolName:='R0';    ReserveTable[02].ParamValue:=$00000002;
  ReserveTable[03].SymbolName:='R1';    ReserveTable[03].ParamValue:=$00100002;
  ReserveTable[04].SymbolName:='R2';    ReserveTable[04].ParamValue:=$00200002;
  ReserveTable[05].SymbolName:='R3';    ReserveTable[05].ParamValue:=$00400002;
  ReserveTable[06].SymbolName:='R4';    ReserveTable[06].ParamValue:=$00800002;
  ReserveTable[07].SymbolName:='R5';    ReserveTable[07].ParamValue:=$01000002;
  ReserveTable[08].SymbolName:='R6';    ReserveTable[08].ParamValue:=$02000002;
  ReserveTable[09].SymbolName:='R7';    ReserveTable[09].ParamValue:=$04000002;
  ReserveTable[10].SymbolName:='PSW';   ReserveTable[10].ParamValue:=$00000004;
  ReserveTable[11].SymbolName:='BUS';   ReserveTable[11].ParamValue:=$00000008;
```

```
ReserveTable[12].SymbolName:='TF';    ReserveTable[12].ParamValue:=$00000010;
ReserveTable[13].SymbolName:='P1';    ReserveTable[13].ParamValue:=$00100020;
ReserveTable[14].SymbolName:='P2';    ReserveTable[14].ParamValue:=$00200020;
ReserveTable[15].SymbolName:='P4';    ReserveTable[15].ParamValue:=$00400020;
ReserveTable[16].SymbolName:='P5';    ReserveTable[16].ParamValue:=$00800020;
ReserveTable[17].SymbolName:='P6';    ReserveTable[17].ParamValue:=$01000020;
ReserveTable[18].SymbolName:='P7';    ReserveTable[18].ParamValue:=$02000020;
ReserveTable[19].SymbolName:='C';     ReserveTable[19].ParamValue:=$00000040;
ReserveTable[20].SymbolName:='T';     ReserveTable[20].ParamValue:=$00000080;
ReserveTable[21].SymbolName:='CNT';   ReserveTable[21].ParamValue:=$00000100;
ReserveTable[22].SymbolName:='TCNT';  ReserveTable[22].ParamValue:=$00000200;
ReserveTable[23].SymbolName:='RB0';   ReserveTable[23].ParamValue:=$00000400;
ReserveTable[24].SymbolName:='RB1';   ReserveTable[24].ParamValue:=$00000800;
ReserveTable[25].SymbolName:='MB0';   ReserveTable[25].ParamValue:=$00001000;
ReserveTable[26].SymbolName:='MB1';   ReserveTable[26].ParamValue:=$00002000;
ReserveTable[27].SymbolName:='I';     ReserveTable[27].ParamValue:=$00004000;
ReserveTable[28].SymbolName:='TCNTI'; ReserveTable[28].ParamValue:=$00008000;
ReserveTable[29].SymbolName:='F0';    ReserveTable[29].ParamValue:=$00010000;
ReserveTable[30].SymbolName:='F1';    ReserveTable[30].ParamValue:=$00020000;
ReserveTable[31].SymbolName:='@*';    ReserveTable[31].ParamValue:=$00040000;
ReserveTable[32].SymbolName:='#*';    ReserveTable[32].ParamValue:=$00080000;
ReserveTable[33].SymbolName:='NOT';   ReserveTable[33].ParamValue:=$00000000;
ReserveTable[34].SymbolName:='*';     ReserveTable[34].ParamValue:=$70000000;
for i := 0 to $FF do
begin
  SymbolTable[i].SymbolName:='';
  SymbolTable[i].ParamValue:=$00000000;
end;
MOT := TMCommandHashTable.Create($EF);
MOT.AddCommand('ADD', $00000003, 1);
MOT.AddCommand('ANL', $80080020, 2);
MOT.AddCommand('CALL',$80000000, 2);
MOT.AddCommand('CLR', $00000001, 1);
MOT.AddCommand('CLR', $00020000, 1);
MOT.AddCommand('CPL', $00000001, 1);
MOT.AddCommand('CPL', $00020000, 1);
MOT.AddCommand('DEC', $00000002, 1);
MOT.AddCommand('DJNZ',$80000002, 2);
MOT.AddCommand('EN',  $00004000, 1);
MOT.AddCommand('EN',  $00008000, 1);
MOT.AddCommand('JC',  $80000000, 2);
MOT.AddCommand('JF1', $80000000, 2);
MOT.AddCommand('JMP', $80000000, 2);
MOT.AddCommand('JNZ', $80000000, 2);
MOT.AddCommand('JZ',  $80000000, 2);
MOT.AddCommand('IN',  $00000021, 1);
MOT.AddCommand('INC', $00000002, 1);
MOT.AddCommand('MOV', $80080002, 2);
MOT.AddCommand('MOV', $80080001, 2);
MOT.AddCommand('MOV', $800C0002, 2);
MOT.AddCommand('MOV', $00000081, 1);
MOT.AddCommand('MOV', $00040003, 1);
MOT.AddCommand('MOV', $00000003, 1);
MOT.AddCommand('MOVD',$00000021, 1);
MOT.AddCommand('MOVD',$10000021, 1);
MOT.AddCommand('ORL', $00000003, 1);
MOT.AddCommand('ORL', $80080020, 2);
MOT.AddCommand('RET', $00000000, 1);
MOT.AddCommand('RETR',$00000000, 1);
MOT.AddCommand('RL',  $00000001, 1);
MOT.AddCommand('RR',  $00000001, 1);
MOT.AddCommand('SEL', $00000800, 1);
MOT.AddCommand('SEL', $00000400, 1);
MOT.AddCommand('STRT',$00000080, 1);
MOT.AddCommand('SWAP',$00000001, 1);
MOT.AddCommand('XRL', $00000003, 1);
MOT.AddCommand('XRL', $80080001, 2);
MOT.AddCommand('END', $00000000, 1);
MOT.AddCommand('DS',  $80000000, 2);
POT:=TPCommandHashTable.Create($0F);
```

```
  POT.AddCommand('EQU', $80000000, 1);
  POT.AddCommand('REQ', $00000000, 1);
  POT.AddCommand('DEFSEG', $00000000, 0);
  POT.AddCommand('SEG', $00000000, 0);
end;

function TMainForm.GetParam(pString: String): longword;
var
  i: byte;
  sN: string;
  hexFlag: boolean;
begin
  Result:=ReserveTable[00].ParamValue;
  if pString='' then Exit;
  case pString[1] of
    '#','@':
      begin
        Result:=GetParam(Trim(Copy(pString,2,Length(pString)-1)));
        pString:=pString[1]+'*';
      end;
    '0'..'9','A'..'F':
      begin
      hexFlag:=false;
      for i := 1 to Length(pString) do
      begin
        if pString[i] in ['0'..'9','A'..'F','H'] then
        begin
          if pString[i] in ['A'..'F','H'] then
            hexFlag:=true;
        end
        else
          Break;
        if hexFlag then
          if Pos('H',pString)<>Length(pString) then
            Break;
        Move(pString[1],pString[2],Length(pString));
        pString[1]:='$';
        AddLiteral(StrToInt(pString));
        Result:=$80000000;
        Exit;
      end;
      end;
    '+','-': Result:=GetParam(Trim(Copy(pString,2,Length(pString)-1)));
  end;
  for i := 0 to 33 do
    if AnsiSameStr(ReserveTable[i].SymbolName,pString) then
    begin
      Result:=Result or ReserveTable[i].ParamValue;
      Exit;
    end;
  if IsSymbol(pString) then
    Result:=GetSymbol(pString)
  else
  begin
    for i := 1 to Length(pString) do
    if pString[i] in ['-','+',' ',#$09] then
    begin
      Result := GetParam(Trim(Copy(pString,1,i-1))) or
      GetParam(Trim(Copy(pString,i+1,Length(pString)-i)));
      Exit;
    end;
    Result:=ReserveTable[34].ParamValue;
  end;
end;

function TMainForm.GetSymbol(aSymbol: String): longword;
var
  i: byte;
begin
  if SymbolCounter > 0 then
```

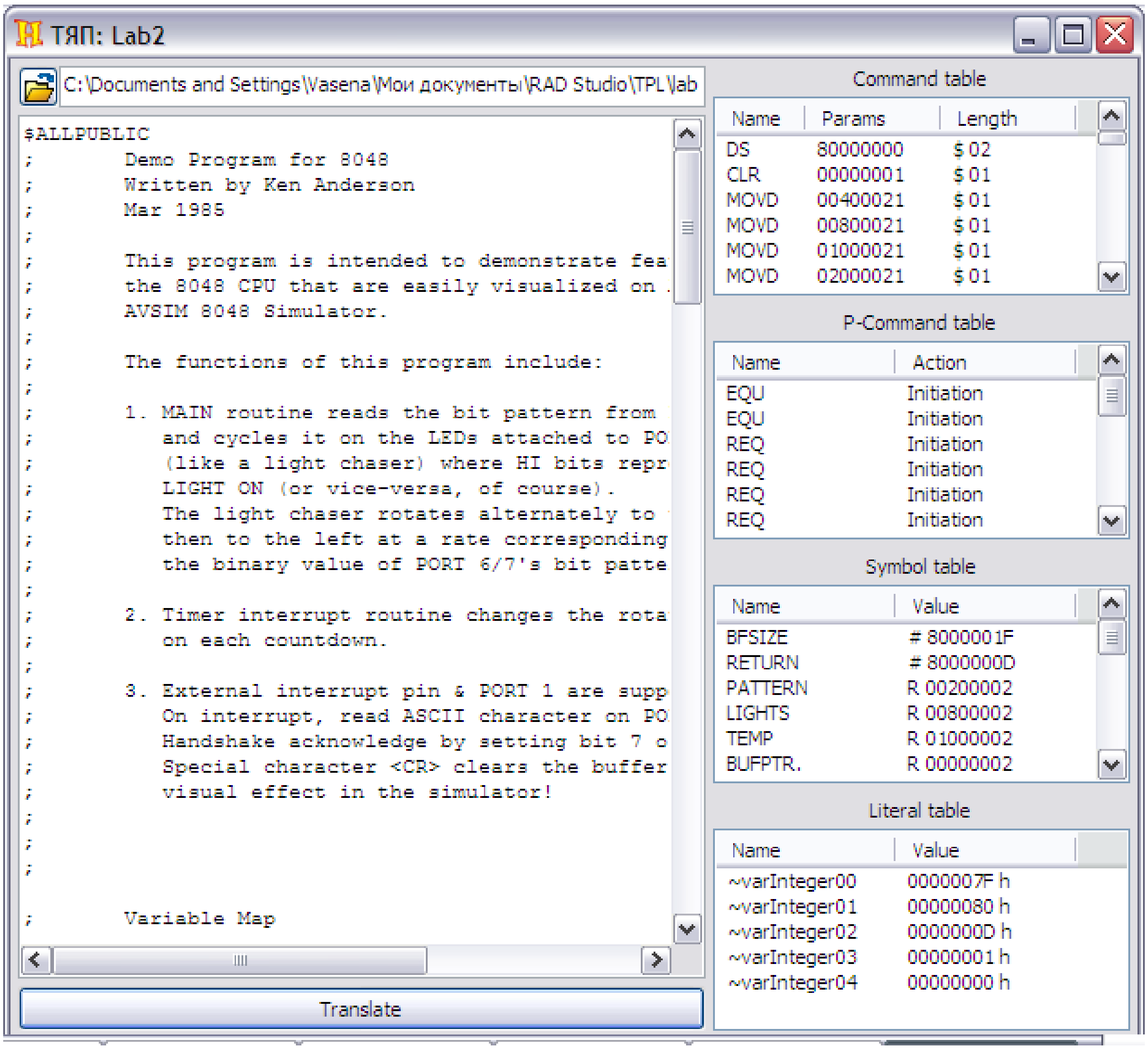
```
    for i := 0 to SymbolCounter - 1 do
      if AnsiSameStr(SymbolTable[i].SymbolName, aSymbol) then
      begin
        if SymbolTable[i].ParamValue and $F0000000 <> 0 then
          Result := SymbolTable[i].ParamValue and $F0000000
        else
          Result := SymbolTable[i].ParamValue;
        Exit;
      end;
  Result := $40000000;
end;

function TMainForm.IsSymbol(aSymbol: String): boolean;
begin
  Result := (GetSymbol(aSymbol) <> $40000000);
end;

class function TMainForm.Min(i1, i2: integer): integer;
begin
  if i1=0 then
    Result:=i2
  else if i2=0 then
    Result:=i1
  else if i1>i2 then
    Result:=i2
  else
    Result:=i1;
end;

end.
```

Результат работы программы


ТЯП: Lab2
C:\Documents and Settings\Vasena\Мои документы\RAD Studio\TPL\lab
Translate

```
$ALLPUBLIC
;       Demo Program for 8048
;       Written by Ken Anderson
;       Mar 1985
;
;       This program is intended to demonstrate fea
;       the 8048 CPU that are easily visualized on .
;       AVSIM 8048 Simulator.
;
;       The functions of this program include:
;
;       1. MAIN routine reads the bit pattern from
;          and cycles it on the LEDs attached to PO
;          (like a light chaser) where HI bits repr
;          LIGHT ON (or vice-versa, of course).
;          The light chaser rotates alternately to
;          then to the left at a rate corresponding
;          the binary value of PORT 6/7's bit patte
;
;       2. Timer interrupt routine changes the rota
;          on each countdown.
;
;       3. External interrupt pin & PORT 1 are supp
;          On interrupt, read ASCII character on PO
;          Handshake acknowledge by setting bit 7 o
;          Special character <CR> clears the buffer
;          visual effect in the simulator!
;
;
;

;       Variable Map
```

Command table

Name	Params	Length
DS	80000000	$ 02
CLR	00000001	$ 01
MOVD	00400021	$ 01
MOVD	00800021	$ 01
MOVD	01000021	$ 01
MOVD	02000021	$ 01

P-Command table

Name	Action
EQU	Initiation
EQU	Initiation
REQ	Initiation
REQ	Initiation
REQ	Initiation
REQ	Initiation

Symbol table

Name	Value
BFSIZE	# 8000001F
RETURN	# 8000000D
PATTERN	R 00200002
LIGHTS	R 00800002
TEMP	R 01000002
BUFPTR.	R 00000002

Literal table

Name	Value
~varInteger00	0000007F h
~varInteger01	00000080 h
~varInteger02	0000000D h
~varInteger03	00000001 h
~varInteger04	00000000 h