รายการอ้างอิง

ไฟล์ข้อมูลโปรแกรมคอมพิวเตอร์

สมภพ คำนูณเศรษฐ์ และ รศ.ดร.สุริยัน ติษยาธิคม. โปรแกรมจำลองซีพียู 6502. ห้องปฏิบัติการไมโครคอมพิวเตอร์ ภาควิชาวิศวกรรมไฟฟ้า จุฬาลงกรณ์มหาวิทยาลัย,2532.

ภาษาไทย

สมภพ คำนูณเศรษฐ์ และ รศ.ดร.สุริยัน ติษยาธิคม. โปรแกรมจำลองซีพียู 6502. การประชุมทางวิชาการวิศวกรรมไฟฟ้า 8 สถาบัน ครั้งที่ 12 (2532)

ยืน ภู่วรวรรณ และ วัฒนา เชียงกูล. ไมโครโปรเซสเซอร์ไมโครคอมพิวเตอร์ ซีเอ็ดยูเคชั่น,2525.

บุญเลิศ เอี่ยมทัศนา. การโปรแกรมแบบโอโอพีด้วยเทอร์โบปาสคาล ซีเอ็ดยูเคชั่น

สุรศักดิ์ สงวนพงษ์ แอดวานซ์เทอร์โบปาสคาล ซีเอ็ดยูเคชั่น,2532.

ภาษาอังกฤษ

Amsterdam, J. Building a computer in software. Byte (October 1985)

————. An assembler for VM2. Byte (November 1985)

Borland International. Turbo Pascal 5.5 USA.

Corcoran, P. Simulator generator system IEEE PROC. Vol.128 (March 1981)

Daley,H.O. Fundamentals of Microprocessors CBS publishing.

Greenfield,S.E. The Architecture of Microcomputers Winthrop.

Ismal,A.R. , and Rooney V.M. Microprocessor hardware and software concept Macmillan.

Kain,R.Y. Computer architecture, software and hardware Vol.1 Prentice-Hall.

รายการอ้างอิง(ต่อ)

Lane,A. Simulating a microprocessor. Byte (August 1987)

Leventhal,L.A. Introduction to microprocessors (software, hardware, programing) Prentice-Hall.

. ,and Saville W. Z80 Assembly Language Subroutines Osborne/McGraw-Hill.

McIntire,T.C. Software Interpreters for Microcomputers John Wiley & Sons.

Mottola,R. Assembly Language Programming for the Apple II Osborne/McGraw-Hill.

Schildt,H. Advanced Turbo Pascal Programming and Techniques 2nd ed. McGraw-Hill Inc,1988.

Tabak,D. RISC architecture Research Studies Press.

ภาคผนวก

ภาคผนวก ก

Program Listing

```
{ ----------SIMULATOR MAIN ------------ }
PROGRAM SIMMAIN;
USES Crt,Dos;
{$I root1.pas }
{$I gnrobj.pas }
{$I regold.pas }
{$I itp.pas }
{$I disp.pas }
{$I ins.pas }
{$I asmobj.pas }
{$I seqobj.pas }

VAR  Aec,Tec,Rec:word;

PROCEDURE InitSim;
var i:word;
begin
     For i:=0 to $FF Do begin
         New(Mem[i]);
         FillChar(Mem[i]^,sizeof(Mem[i]^),0); end;
     InitRegister;  PC:=0;  SP:=0;
     InitAssembler; ec:=1; Aec:=1; Tec:=1; Rec:=1;
     LineNo:=1;  MyMemoryCounter:=0; CurrentStoreMemory:=0;
end;
PROCEDURE AssemblerCommand;
var  A1: word;  e: integer;  Stop: boolean;
begin
     SetWindow(RstWd);
     Writeln('-----------ASSEMBLE-------------');
     post:= 2;
     HesToWord(NextWd,A1,e); { Str2v(NextWd,A1,e);}
     if e=0 then LC:= A1;
     Stop:= False;   ec:= Aec;
     Repeat
       ReadCommandLine(ByteToHes(Hi(LC))+ByteToHes(Lo(LC))+': ');
       if CommandLine <> ^M then begin
          AsmOneIns(CommandLine,e,Stop);
```

```
          if e<>0 then   begin
             ErrorMode(e); Writeln('Error at line ',ExeTable[ec]^.line); end;
          EndEC:= ec; Aec:= ec;   end;
     Until (CommandLine = ^M) or Stop;
end;
PROCEDURE LoadAssemblyCommand;
var  pfi: text; Name: string; Erln: word;
begin
     SetCursorOff;  SetWindow(RstWd);
     Writeln('----------------LOADING-------------------');
     post:=2;  Name:= NextWd;  Erln:=0;
     if Name <> '' then  begin   Name:='a:'+Name;
         assign(pfi,Name);  {$I-}  reset(pfi);  {$I+}
         If IOresult = 0 Then begin
             LC:= PC;
             AsmPass1(pfi,Erln);   Close(pfi);
             if Erln = 0 then  AsmPass2(Erln);
             SetWindow(RstWd);
             if Erln = 0 then Writeln('Load Completed.')
             else  Writeln('Error at line ',Erln,'. INITIALIZE !');
                               end
         Else ErrorMode(-9);
                          end;
     SetCursorOn;
end;
PROCEDURE TraceCommand;
var  A1,Tn,step: word; Stop: boolean; ch:char; e,RunErr:integer;
begin
     SetCursorOff;  SetWindow(RstWd);
     Writeln('-------------TRACE---------------');
     post:=2;
     HesToWord(NextWd,A1,e); if e = -1 then Tn:=1
                              else Tn:= A1;
     Stop:= False;  Step:=0;   ec:= Tec;   RunErr:= 0;
     Repeat
         ExecuteOneMacro(Stop,RunErr);  Tec:= ec;
         if KeyPressed then begin ch:=Readkey; if ch=^C then Stop:=true; end;
```

```
        if stop and (RunErr<>0) then  begin
            ErrorMode(RunErr);
            writeln('Error at line..',ExeTable[ec]^.line,'. INITIALIZE !');
                                          end;
        step:= step+1;
    Until (step >= Tn) or Stop;
    if (ec = EndEC) or Stop then Tec:=1;
    SetCursorOn;
end;
PROCEDURE RunCommand;
var Breakpoint: word; Stop:boolean; ch:char; e,RunErr:integer;
begin
    SetCursorOn;  SetWindow(RstWd);
    Writeln('---------------RUN----------------');
    post:=2;
    HesToWord(NextWd,BreakPoint,e); if e= -1 then BreakPoint:=0;
    Stop:= False;  ec:=Rec;  RunErr:= 0;
    Repeat
       ExecuteOneMacro(Stop,RunErr);
       if Keypressed then begin
          ch:=Readkey;
          if ch=^C then begin
             Stop:=true; SetWindow(RstWd);
             Writeln('--------------USER BREAK----------------');
                        end;  end;
    Until (ec=EndEC) or Stop or (PC=BreakPoint);
    if stop and (RunErr<>0) then begin
       ErrorMode(RunErr);
       writeln('Error at line..',ExeTable[ec]^.line,'. INITIALIZE !'); end;
    if (ec=EndEC) or Stop then begin Rec:=1; Tec:=1; end;
    SetCursorOn;
end;
PROCEDURE InitializeCommand;
Var i:byte;
begin
    SetCursorOff;  SetWindow(RstWd);
    Writeln('-----------INITIALIZE--------------');
```

```
     DisposeExeTable;
     For i:=0 to $FF Do
         FillChar(Mem[i]^,sizeof(Mem[i]^),0);
     PC:=0;  SP:=0;
     if HaveR8 then for i:=1 to r8num do R8T[i]^.clear;
     if HaveR16 then for i:=1 to r16num do R16T[i]^.clear;
     for i:=0 to 10 do F1T[i]^.Resetf;
     SymTable.Init;  InitExeTable;  New(oprs1); New(oprs2);
     ec:=1; Aec:=1; Tec:=1; Rec:=1;
     LineNo:=1;
     SetCursorOn;
end;
PROCEDURE HelpCommand;
begin
     SetCursorOff;  SetWindow(RstWd);
     Writeln('--------------------COMMAND-------------------');
     Writeln('A [from]                   -Assemble');
     Writeln('D [from] [off]             -Dump memory');
     Writeln('F [from] [to] [N1]..[N10] -Fill memory with Nn');
     Writeln('G [break point]            -Run');
     Writeln('H                          -This menu!');
     Writeln('I                          -Initialize');
     Writeln('L [file]                   -Load assembly code');
     Writeln('Q                          -Quit');
     Writeln('R [off]                    -Change register');
     Writeln('T [N]                      -Trace N steps');
     SetCursorOn;
end;
PROCEDURE ShowRegisterCommand;
Const     Posit: array[1..43,0..1] of byte =
((6,1),(6,2),(6,3),(6,4),(6,5),(6,6),(6,7),(6,8),(6,9),(6,10),(6,11),(6,12),
(6,13),(6,14),(6,15),(14,1),(14,2),(14,3),(14,4),(14,5),(14,6),(14,7),(14,8),
(14,9),(14,10),(14,11),(14,12),(14,13),(14,14),(14,15),(22,1),(22,2),(25,5),
(25,6),(25,7),(25,8),(25,9),(25,10),(25,11),(25,12),(25,13),(25,14),(25,15));
Var
hex: string; I: word;  j,c,r,A1: byte;  Err:integer;  ch:char;
Ad: array[1..32] of str4;
```

```
begin
SetWindow(RstWd); writeln('--------------REGISTER----------------');
post:= 2;   hex:= NextWd;
if hex= 'OFF' then begin
    SetWindow(RegWd); clrscr; WdData.Wd[RegWd,6]:= 0; exit; end;
WdData.Wd[RegWd,6]:= 1;
ShowRegisterWindow;
if haver8 then
For j:=1 to R8num do Ad[j]:= ByteToHes(R8T[j]^.v);
if haver16 then
For r:=1 to R16num Do begin  j:= r+15;
    Ad[j]:= ByteToHes(R16T[r]^.hb.v)+ByteToHes(R16T[r]^.lb.v); end;
Ad[31]:= ByteToHes(Hi(PC))+ByteToHes(Lo(PC));
Ad[32]:= ByteToHes(Hi(SP))+ByteToHes(Lo(SP));
j:=1;  r:=1;
if haver8 then  c:=0 else if haver16 then c:=1;
REPEAT
   j:= r + 15 * c;
   gotoxy(Posit[j,0],Posit[j,1]);
   ch:= Readkey;
   if KeyPressed and (ch= #27) then ch:= Readkey;
   case ch of
   ^I,#77: case c of
           0: if haver16 then c:=1 else c:=2;
           1: c:=2;  end;
   #75: case c of
        2: if haver16 then c:=1 else c:=0;
        1: if haver8 then c:=0;  end;
   #72: if r>1 then r:= r-1;
   #80: case c of
        0: if r<R8num then r:= r+1;
        1: if r<R16num then r:= r+1;
        2: if r<13 then r:= r+1;    end;
   ^M: begin
       case j of
       1..32: begin
```

```
            hex:= Ad[j];  ReadLine(Posit[j,0],Posit[j,1],hex); Ad[j]:= hex;
            HesToWord(hex,I,Err);
            if Err = 0 then begin
              case j of
              1..15: if haver8 then R8T[r]^.v:= I;
              16..30:if haver16 then R16T[r]^.putw(I);
              31:      PC:= I;
              32:      SP:= I;
              end; Updateflag; end;
            end;
       33..43: begin
            F1T[r-3]^.invertf; UpdateFreg;  end;
       end; end;
    end;
    SetCursorOff;  ShowRegisterWindow;  SetCursorOn;
UNTIL  ch = #27;
end;
PROCEDURE DumpMemoryCommand;
Var  A1,A2,Mad:word;  ch:char; e,R,x,y:integer; Hex:string;
begin
     SetWindow(RstWd);
     writeln('-------------------MEMORY-------------------');
     post:=2; Hex:= NextWd;
     if Hex='OFF' then begin
        SetWindow(MemWd); clrscr; WdData.Wd[MemWd,6]:=0; Exit; end;
     WdData.Wd[MemWd,6]:= 1;
     HesToWord(Hex,A1,e); if e=0 then MyMemoryCounter:= A1;
     WdData.Wd[MemWd,6]:= 1; CurrentStoreMemory:= MyMemoryCounter;
     DumpMemoryWindow;
     R:= MyMemoryCounter Mod 8;
     Repeat
       if (A2<>MyMemoryCounter) then begin
         A2:= MyMemoryCounter; CurrentStoreMemory:= MyMemoryCounter;
         SetCursorOff;
         DumpMemoryWindow;
         SetCursorOn;                    end;
       x:= (R Mod 8)*3+6;
```

```
        y:= R Div 8+1;
        gotoxy(x,y);
        ch:= Readkey;
        if KeyPressed and (ch= #27) then ch:= ReadKey;
        Case ch of
        #72: if R>=8  then R:=R-8 else MyMemoryCounter:=MyMemoryCounter-8;
        #80: if R<=39 then R:=R+8 else MyMemoryCounter:=MyMemoryCounter+8;
        #73: MyMemoryCounter:= MyMemoryCounter-48;
        #81: MyMemoryCounter:= MyMemoryCounter+48;
        #75: if R>=1 then R:=R-1
             else begin MyMemoryCounter:=MyMemoryCounter-8; R:=7; end;
        #77: if R<=46 then R:=R+1
             else begin MyMemoryCounter:=MyMemoryCounter+8; R:=40; end;
        ^M : begin
             Mad:= MyMemoryCounter+R;
             Hex:= ByteToHes(Mem[Hi(Mad)]^[Lo(Mad)]);
             Readline(x,y,Hex);
             HesToWord(Hex,A1,e);
             if (e=0) and (A1<256) then
                Mem[Hi(Mad)]^[Lo(Mad)]:= A1;
             end;
        end;
      Until ch = #27;
      CurrentStoreMemory:= 0;
end;
PROCEDURE FillMemoryCommand;
Var i,j,k,A1,A2,A3:word; ch:char; e:integer;
    Ax: array[0..9] of word;
begin
     SetCursorOff; SetWindow(RstWd);
     writeln('------------------FILL MEM-------------------');
     post:=2;
     HesToWord(NextWd,A1,e); if e<>0 then Exit;
     HesToWord(NextWd,A2,e); if e<>0 then Exit;
     j:=0;
     Repeat
       HesToWord(NextWd,Ax[j],e); j:=j+1;
```

```
     Until (e<>0) or (j>9);
     if e>0 then Exit;
     j:=j-1;
     k:=0; A3:=0;
     For i:=A1 to A2 Do begin
        Mem[Hi(i)]^[Lo(i)]:= Ax[k]; k:=(k+1) Mod j;
        if (i>=MyMemoryCounter) and (i<=MyMemoryCounter+48) then A3:=i;
                        end;
     CurrentStoreMemory:= A3;
     DumpMemoryWindow;
     SetCursorOn;
end;

VAR    ch:Char;  Quit:boolean;      hpnt:pointer;
BEGIN                      Mark(hpnt);
ClrScr;  NormVideo;
InitSim;  Setrmin;
SetWindow(CmdWd);  Writeln('READY');
LC:= PC;
Quit:= False;
Repeat
     ReadCommandLine('-');
     Ch:= Upcase(CommandLine[1]);
     case ch of
     'R': ShowRegisterCommand;
     'T': TraceCommand;
     'Q': Quit:= True;
     'A': AssemblerCommand;
     'D': DumpMemoryCommand;
     'F': FillMemoryCommand;
     'G': RunCommand;
     'L': LoadAssemblyCommand;
     'H': HelpCommand;
     'I': InitializeCommand;
     ^M : ;
     else    ErrorMode(-10);
     end;
```

```
Until  Quit;                  Release(hpnt);
END.□
```

```
{ --------------CPU Control and Execute------------------ }
Var  Js,JnotF :boolean; oprnd1,oprnd2: word;  bts: byte;


PROCEDURE SearchLC(svl:word);
begin
     While (svl>ExeTable[ec]^.Location)and(ec<EndEC) do ec:=ec+1;
     While (svl<ExeTable[ec]^.Location)and(ec>1) do ec:=ec-1;
     If svl=ExeTable[ec]^.Location  Then JnotF:= false
     Else   JnotF:= true;
end;


PROCEDURE DoIns1(s:byte; var qt:boolean);
var xPC: word;
begin
     case s of
     0: PC:= PC+1;
     1: begin SearchLC(oprnd1); Js:=true; end;
2: begin xPC:= PC+bts;  PushW(xPC); SearchLC(oprnd1); Js:=true; end;
     3: begin PullW(xPC); SearchLC(xPC); Js:=true; end;
     4: qt:= true;
     end;
end;


PROCEDURE Jpointer(k:char; n:byte);
var xpc: word;
begin
     GetValue(k,n,xpc); SearchLC(xpc); Js:= true;
end;


PROCEDURE DoIns2(s:byte; k:char; n:byte; var bd:integer);
begin
     case s of
     5:  itpAShiftRight(k,n,bd);
     6:  itpClear(k,n,bd);
     7:  itpComplement(k,n,bd);
     8:  itpDecAdj(k,n,bd);
     9:  itpDec(k,n,bd);
```

```
      10:  itpInc(k,n,bd);
      11:  itpInput(k,n,bd);
      12:  itpInvert(k,n,bd);
      14:  itpLoad(k,n,bd);
      15:  itpLShiftRight(k,n,bd);
      16:  itpOutput(k,n,bd);
      17:  itpPull(k,n,bd);
      18:  itpPull6502(k,n,bd);
      19:  itpPush(k,n,bd);
      20:  itpPush6502(k,n,bd);
      21:  itpReset(k,n,bd);
      22:  itpRotLeft(k,n,bd);
      23:  itpRoLcarry(k,n,bd);
      24:  itpRotRight(k,n,bd);
      25:  itpRoRcarry(k,n,bd);
      26:  itpSet(k,n,bd);
      27:  itpShiftLeft(k,n,bd);
      28:  itpStore(k,n,bd);
      29:  itpTestParity(k,n,bd);
      30:  itpTestSignZero(k,n,bd);
      31:  itpTestZero(k,n,bd);
      32:  itpTwoComp(k,n,bd);      49: itpTestSign(k,n,bd);
       else  ErrorMode(-11);
       end;
end;
PROCEDURE DOIns3(s:byte; ka,kb:char; na,nb:byte; var bd:integer);
begin
     case  s of
     33: itpAdc(ka,kb,na,nb,bd);
     34: itpAdd(ka,kb,na,nb,bd);
     35: itpAnd(ka,kb,na,nb,bd);
     36: itpCompare(ka,kb,na,nb,bd);
     37: itpCp6502(ka,kb,na,nb,bd);
     38: itpExchange(ka,kb,na,nb,bd);
     39: itpInbyPtr(ka,kb,na,nb,bd);
     40: itpLdbyPtr(ka,kb,na,nb,bd);
     41: itpOr(ka,kb,na,nb,bd);
```

```
      42: itpOutbyPtr(ka,kb,na,nb,bd);
      43: itpReplacefv(ka,kb,na,nb,bd);
      44: itpSbc(ka,kb,na,nb,bd);
      45: itpStbyPtr(ka,kb,na,nb,bd);
      46: itpSub(ka,kb,na,nb,bd);
      47: itpTransfer(ka,kb,na,nb,bd);
      48: itpXor(ka,kb,na,nb,bd);
      else  ErrorMode(-11);
      end;
end;

PROCEDURE DoNormIns(ei:ExeInsT; var bad:integer);
begin
      If ei.exk = 2 Then  DoIns2(ei.v2,ei.spc.t,ei.spc.i,bad)
      Else  if ei.exk = 3 then
               DoIns3(ei.v3,ei.spc1.t,ei.spc2.t,ei.spc1.i,ei.spc2.i,bad);
end;

FUNCTION StTrue1(c:condT):boolean;
var  b:boolean;  n:word;
begin
      If c.tp = 'f' Then begin
          Getvalue(c.tp,c.id,b);
          if (c.st xor b)  then StTrue1:= False
          else   StTrue1:= True;
                        end
      Else begin
           GetValue(c.tp,c.id,n);
           if ((c.st) xor (n<>0)) then StTrue1:= False
           else  StTrue1:= True;
           end;
end;

FUNCTION StTrue2(c1,c2:condT; l:boolean):boolean;
var sa,sb:boolean;
begin
      sa:= stTrue1(c1);  sb:= stTrue1(c2);
```

```
     if l then StTrue2:= (sa and sb)
     else  StTrue2:= (sa or sb);
end;

PROCEDURE ExecuteOneMacro(var Stp:boolean; var exerr:integer);
Var  i,mn,x,ii,i1,i2: byte;   tt,t1,t2: char;   mds:str8; err:integer;
begin
  With  ExeTable[ec]^  Do  Begin
     PC := Location;   mn:= instruction.micron;
     oprnd1:= operand1;  oprnd2:= operand2;  bts:= instruction.bytes;
     mds:= instruction.genericmode;
     Js:= False;      JNotF:= False;
With instruction Do  Begin
     If (gins[1].exk in [1..3,5..7])   Then  begin
        ADDRESSING(mds,oprnd1,oprnd2);   i:=1;
        while (i <= mn) and (Not Js)  Do  begin
case gins[i].exk of
1: x:= gins[i].v1;
2: begin  x:= gins[i].v2;  tt:= gins[i].spc.t;  ii:= gins[i].spc.i;  end;
3: begin  x:= gins[i].v3;  t1:= gins[i].spc1.t; i1:= gins[i].spc1.i;
                           t2:= gins[i].spc2.t; i2:= gins[i].spc2.i; end;
5: x:= gins[i].v5;
6: begin  x:= gins[i].v6;  tt:= gins[i].spc6.t; ii:= gins[i].spc6.i; end;
7: begin  x:= gins[i].v7;  t1:= gins[i].spca.t; i1:= gins[i].spca.i;
                           t2:= gins[i].spcb.t; i2:= gins[i].spcb.i; end;
end;
        case gins[i].exk of
        1: DoIns1(x,stp);
        2: If x=13  Then Jpointer(tt,ii)
           Else  DoIns2(x,tt,ii,exerr);
        3: DoIns3(x,t1,t2,i1,i2,exerr);
        5: If StTrue1(gins[i].icond5) Then DoIns1(x,stp);
        6: If StTrue1(gins[i].icond6) Then begin
               if x=13  then Jpointer(tt,ii)
               else  DoIns2(x,tt,ii,exerr);      end;
        7: If StTrue1(gins[i].icond7) Then DoIns3(x,t1,t2,i1,i2,exerr);
        end;    i:= i+1;                  end; { while }
```

```
                                             end
     Else begin
          If gins[1].exk = 4 Then  begin
             While  stTrue1(gins[1].ucond)  Do  begin
                ADDRESSING(mds,oprnd1,oprnd2);
                for i:=2 to mn do begin DOnormINS(gins[i],err);
                    if err<>0 then exerr:= err;   end;
                                                   end;
                                   end
          Else if gins[1].exk = 8 then
               While stTrue2(gins[1].uconda,gins[1].ucondb,gins[1].lg) Do
               begin Addressing(mds,oprnd1,oprnd2);
                    for i:=2 to mn do begin DoNormIns(gins[i],err);
                       if err<>0 then exerr:= err;  end;
               end;
          end;         End; End;
   ShowRegisterWindow; { DumpMemoryWindow; }  { ************** }
   If Not Js and Not Stp Then    ec:= ec+1
   Else if JNotF then  exerr:= -13;
   if exerr<>0 then stp:= true;
 end;□
```

```
{ ----------------Assembler------------------}
TYPE
SymT  =  record  SymName : string[8];  SymValue : word;  end;
SymTableO  =  OBJECT
                 Sym : array[1..100] of SymT;
                 SymNo : word;
            Procedure Init;
            Procedure Store(Symin:string; Vin:word; Var SE:integer);
            Procedure Search(Symin:string; Var Vout:word; Var SE:integer);
            end;

InstrPtr  =  ^EachInstr;
EachInstr =  record InstrData  : ExeInsRecT;
                    Next       : InstrPtr;   end;
ITableO  =  object
                  InstrTable : array[1..1000] of InstrPtr;
           Procedure Init;
           Function Key(keystr :string):byte;
           Procedure InsertInstr(Datain:ExeInsRecT);
           Procedure ReadInstrFi;
           Procedure Search(mnein,modein:string;
                        var exeireco:ExeInsRecT;var Ie:integer);
           end;

ANALZO = object
           CommentSep, OperandSep :Char;
           Procedure Init (CmtSepIn,OprSepIn:Char);
           Procedure NextWord(Sin:string; var p:byte; var sout:string);
           Procedure Tokenize(Sin:string;
              var Labelo,Mneo,Operf1o,Operf2o: string);
           Procedure SepOpr(var Oone,Otwo:string);
           end;

   MdrecT = record pat: str8;  md:array[0..4] of str8;
                   oa: array[1..12] of str5;      end;
   ar20ofMdrecT = array[1..20] of MdrecT;
 AddrMdO = OBJECT
            Amp: ^ar20ofMdrecT;   NoAm: byte;
```

```
            Procedure Init;
            Procedure Search(oprin,OPin:string;
                var Mout,oper1,oper2:string;var er:integer);
            end;

ExeTableT = RECORD  Line: word; {AsmIns: string;} Location: word;
                           Instruction: ExeInsRecT;
                           Operand1,Operand2: word;    end;
PntExeTableT = ^ExeTableT;

arofstr15 = array[1..500] of string[15];

CONST TokSep : set of char =  [' ',^I,^M,':'];
VAR   Analz: AnalzO; SymTable: SymTableO; AddrMd: AddrMdO;
      Itable: ITableO;  HaveMneImp,Resinoprf: boolean; OPRSEP: char;
      oprs1,oprs2: ^arofstr15;   arec:arecT;
      ec,EndEC,LC,LineNo : word;
      ExeTable: array[1..500] of PntExeTableT;

Procedure SymTableO.Init;
BEGIN
SymNo := 1;    FillChar(Sym,sizeof(Sym),0);
END;
Procedure SymTableO.Store(Symin:string; Vin:word; VAR SE:integer);
VAR I:BYTE;
BEGIN
i := 0;   SE := 0;
While (i < SymNo) and (Sym[i].SymName <> Symin) do i:=i+1;
If i = SymNo Then  begin
     With Sym[SymNO] Do begin
     SymName := Symin; SymValue := Vin;  end;
     SymNo := SymNo + 1;  end
Else SE := 1;
END;
Procedure SymTableO.Search(Symin:string; VAR Vout:word; VAR SE:integer);
VAR   I:BYTE;
BEGIN
i := 0;   SE := -2;    Vout := 0;
```

```
While  (i<SymNo) and (Sym[i].SymName <> Symin) do i:=i+1;
If Sym[i].SymName = Symin Then  begin
     Vout := Sym[I].SymValue;  SE:=0;   end;
END;


Procedure ITableO.Init;
BEGIN
FillChar(InstrTable,sizeof(InstrTable),0);
END;
Function ITableO.Key(Keystr:string):byte;
CONST MAXKEY = 255;
VAR I,SUM : WORD;
BEGIN
Sum := 0;
for i:=1 to length(keystr) do Sum:= Sum+(Ord(keystr[i])-Ord('A'))*i;
Key := Sum MOD Maxkey;
END;
Procedure ITableO.InsertInstr(Datain:ExeInsRecT);
VAR  INDEX :WORD;  DATAPTR , PTR : INSTRPTR;
BEGIN
Index := Key(Datain.Mnemonic+Datain.Mode);
New(DataPtr);
DataPtr^.Next := Nil;
DataPtr^.InstrData := Datain;
Ptr := InstrTable[Index];
If Ptr = Nil Then InstrTable[Index] := DataPtr
Else begin
     while Ptr^.Next <> Nil do Ptr := Ptr^.Next;
     Ptr^.Next := DataPtr;  end;
END;
Procedure ITableO.ReadInstrFi;
CONST InstrFiName: string = 'a:eins.dat';
VAR  FV : FILE OF ExeInsRecT;  FL : ExeInsRecT;
BEGIN
Assign(FV,InstrFiName);
{$I-}  Reset(FV);  {$I+}
if ioresult<>0 then begin
   EdittedIfi2ExeIfi;  reset(fv);  end;
```

```
While Not EOF(FV) Do begin
      Read(FV,FL);
      InsertInstr(FL);  end;
Close(FV);
END;
Procedure ITableO.Search(mnein,modein:string; var exeireco:ExeInsRecT;
                        Var Ie:integer);
VAR INDEX:WORD;  PTR:INSTRPTR;
BEGIN
Index := Key(mnein+modein);
Ptr := InstrTAble[Index];  Ie:= -6;
While  (Ptr <> Nil) and (ie<>0) Do begin
      If (Ptr^.InstrData.Mnemonic = mnein)and(Ptr^.InstrData.Mode=modein)
      Then  Ie:=0
      Else Ptr:=Ptr^.Next;          end;
If Ie=0 Then   exeireco:= Ptr^.instrData;
END;

Procedure AnalzO.Init (CmtSepIn,OprSepIn:Char);
begin
CommentSep := cmtsepin; OperandSep := oprsepin;
TokSep := TokSep + [commentsep];
end;
Procedure AnalzO.Nextword(Sin:string; var p:byte; var sout:string);
begin  sout := '';
       If Sin[p] <> ^M Then  begin
            While Sin[p] in TokSep Do begin
                 if sin[p]=commentsep then exit;
                 p:= p+1;             end;
            while Not (Sin[p] in TokSep) Do
                  begin sout:=sout+upcase(sin[p]); p:=p+1; end; end;
end;
Procedure AnalzO.SepOpr(var Oone,Otwo:string);
var e:integer;  p,l:byte;
begin
      l := length(Oone);  p:=1;
      while Not(Oone[p] in TokSep) and Not(Oone[p] = OperandSep) do p:=p+1;
      if Oone[p] = OperandSep then  begin
```

```
        Otwo := copy(Oone,p+1,l-p);
        delete(Oone,p,l-p+1);      end;
end;
Procedure AnalzO.Tokenize(Sin:string; var Labelo,Mneo,Operf1o,Operf2o:string);
var post:byte;
begin
      Labelo := ''; Mneo := '';  operf1o := '';  operf2o := '';
      post :=1;
      If sin <> '' Then begin
           sin := sin+^M;
           if Not (sin[post] in TokSep) then
              Nextword(sin,post,Labelo);
           Nextword(sin,post,Mneo);  if Mneo = '' then exit;
           Nextword(sin,post,Operf1o);
           if (Operf1o[1] <> '') and (operandsep<>'') then
                  SepOpr(Operf1o,Operf2o);
      end;
end;

PROCEDURE SearchReserved(sin:string; var r:integer);
var  i:byte;
begin
     i:=1;
     while (sin<>arec.rs[i]) and (i<40) do i:=i+1;
     if sin = arec.rs[i]  then  r:= 0
     else  r:= -3;
end;
PROCEDURE SearchImpMne(sin:string; var r:integer);
var i:byte;
begin
     i:=1;
     while (sin<>arec.mimp[i]) and (i<10) do i:=i+1;
     if sin = arec.mimp[i]  then r:= 0
     else  r:= -4;
end;

PROCEDURE Getpattern(opri:string; var patterno,ar1,ar2:string);
Var l,i,post:integer; ar:array[1..2] of string;
```

```
    rg1,rg2:integer; idx:boolean; operator:char;
const Sep :set of char =['(',')',',','#',' ',^I,^M];


begin
OPRI:= OPRI+^M;
post:= 1; patterno:= '';
for i:=1 to 2 do ar[i]:='';   i:=1;
While OPRI[POST]<>^M Do  begin
    while opri[post] in Sep do  begin
        IF NOT(OPRI[POST] IN [' ',^I]) THEN
          patterno:= patterno+opri[post];
          post:=post+1;  end;
    while not (opri[post] in Sep)  do  begin
          ar[i]:= ar[i]+opri[post];
          post:= post+1;               end;
    i:=i+1;       end;
ar1:=ar[1];  ar2:=ar[2];  for i:=1 to 2 do ar[i]:='';
post:=1;  l:= length(ar1);
while (post<=l) and not(ar1[post] in ['+','-']) do post:=post+1;
If ar1[post] in ['+','-']  Then  begin
     ar[1] := copy(ar1,1,post-1);  operator:= ar1[post];
     ar[2] := copy(ar1,post+1,l-post);
     SearchReserved(ar[1],rg1);  SearchReserved(ar[2],rg2);
     if (rg1=0) or (rg2=0) then begin
         if (rg1=0) and (rg2<>0) then begin
             patterno:= patterno+ar[1];  ar1:=ar[2];  end
         else begin patterno:= patterno+ar[2];  ar1:=ar[1];  end;
         if operator = '-' then ar1:= concat('0-',ar1);
                                     end;
                                end
Else begin
     SearchReserved(ar1,rg1);  SearchReserved(ar2,rg2);
     if (rg1=0) and (rg2<>0) then begin
         patterno:= patterno+ar1;  ar1:= ar2;  ar2:='';  end
     else if rg2=0 then begin
             patterno:= patterno+ar2; ar2:='';  end;
     end;
end;
```

```
PROCEDURE ManageOper(oper:string; var operV:word; var res:integer);
VAR L,I:BYTE; OPERATOR:CHAR; TWO:STRING; VALUETWO:WORD; SYM:BOOLEAN;
BEGIN
l := length(oper);  i:=1;   operator:=#0;   operV:=0;
While (i<l) and not(oper[i] in ['+','-']) do i:=i+1;
If oper[i] in ['+','-'] Then  begin
      operator :=oper[i];
      Two :=copy(oper,i+1,l-i);
      Str2V(Two,ValueTwo,res);
      if res <> 0 then Exit;
      delete(oper,i,l-i+1);   end;
Sym := CheckSym(oper);
If Sym Then  SymTable.Search(oper,operV,res)
Else  Str2V(oper,operV,res);
If res = 0 Then
     if operator <> '' then   begin
     case operator of
     '+': operV := operV + ValueTwo;
     '-': begin  if Valuetwo <= operV then  operV := operV - ValueTwo
                 else operv:= operv+256-valuetwo;  end;
     end;
                             end;
END;

procedure AddrMdO.Init;
Const mfn: string = 'a:mode.dat';
Var  mf:File of MdrecT;  i:byte;
Begin
     assign(mf,mfn);  {i-}  reset(mf);  {i+}
     if ioresult=0 then begin
     GetMem(Amp,20*sizeof(MdrecT));    i:=1;
     While Not Eof(mf) Do  begin
         Read(mf,Amp^[i]);  i:=i+1;  end;  close(mf); NoAm:=i-1;
                    end
     else begin writeln(mfn,' Not Found!'); exit; end;
End;
PROCEDURE
```

```
AddrMdO.Search(oprin,OPin:string;var Mout,oper1,oper2:string;var er:integer);
Var pattern: string;  Voper:word;  i,j:byte;
Begin
     GetPattern(oprin,pattern,oper1,oper2);
     i:=1;  er:= -5;  Mout:= '';
     While (i<NoAm) and (pattern<>Amp^[i].pat) do i:=i+1;
     If pattern = Amp^[i].pat  Then  begin
With Amp^[i] Do BEGIN
         er:=0;   Mout:= md[0];
         if Mout = 'MNE' then  begin
             j:=1;
             while (j<12) and (OPin <> oa[j]) do j:=j+1;
             if OPin = oa[j]  then  Mout:= md[1]
             else Mout:= md[2];
                                  end;
         if Mout = 'OPERAND'  then  begin
             ManageOper(oper1,Voper,er);
             if er=0 then begin
                 if Voper<256 then Mout:= md[3]
                 else  Mout:= md[4]; end
             else  er:= -5;
                                       end; END; end;
End;

Procedure InitExeTable;
Var i: integer;
Begin
     for i:=1 to 500 do ExeTable[i]:= Nil;
End;

PROCEDURE InitAssembler;
Const afn: string = 'a:uasm.dat';
Var  af:File of arecT;  chn :array[1..4] of char;  i:byte;
Begin
     assign(af,afn); reset(af); read(af,arec); close(af);
     Analz.Init(arec.cms,arec.ops);   OPRSEP:= arec.ops;
     if arec.mimp[1] = '' then Havemneimp:= False
     else    Havemneimp:=True;
```

```
     if arec.rs[1]='' then  Resinoprf:= False  else Resinoprf:=True;
     For i:=1 to 4 Do    VarToReg(arec.idxreg[i],chn[i],xn[i]);
     AddrMd.Init;
     Itable.Init;  Itable.ReadInstrFi;
     SymTable.Init;
     InitExeTable;
     New(oprs1);  New(oprs2);
End;

PROCEDURE Equate(lb,opr:string; var r:integer);
var equvalue :word;
begin
     Str2v(opr,equvalue,r);
     if r=0  then  SymTable.Store(lb,equvalue,r);
end;
PROCEDURE DefStorage(lb,m,opr:string; var r:integer);
var literal :word;
begin
     If lb <> '' Then  SymTable.Store(lb,LC,r);
     if r<>0 then Exit;
     Str2v(opr,literal,r);
     if r<>0 then Exit;
     Mem[Hi(LC)]^[Lo(LC)]:= Lo(Literal);  LC:= LC+1;
     if m = 'DW' then begin
         Mem[Hi(LC)]^[Lo(LC)]:= Hi(Literal); LC:=LC+1; end;
end;
PROCEDURE ResStorage(lb,opr:string; var r:integer);
var bt:word;
begin
     if lb <> '' then  SymTable.Store(lb,LC,r);
     if r <> 0 then Exit;
     Str2v(opr,bt,r);
     if r <> 0 then Exit;
     LC:= LC+bt;
end;
PROCEDURE Origin(opr:string; var r:integer);
var startad :word;
begin
```

```
     Str2v(opr,startad,r);
     if r = 0  then  LC:=startad;
end;

PROCEDURE DisposeExeTable;
Var  i,n: integer;
begin
      i:=1;
      While ExeTable[i] <> Nil Do begin
         Dispose(ExeTable[i]); i:=i+1;  end;  n:=i-1;
      for i:=1 to n do ExeTable[i]:= Nil;
      Dispose(oprs1);   Dispose(oprs2);
end;

PROCEDURE AsmOneIns(OneIns:string; var e:integer; var stp:boolean);
Var s1,s2 :string;             PSEUDOI: BOOLEAN;
    InsData : ExeInsRecT;
    lab,mne,mde,operf1,operf2: string;    exepoint: PntExeTableT;
begin
e:= 0;  stp:= false;
If OneIns<>'' Then
begin   ANALZ.TOKENIZE(OneIns,lab,mne,operf1,operf2);
pseudoi:= false;
       if mne='EQU' then BEGIN  PSEUDOI:= TRUE;
            Equate(lab,operf1,e); END;
       if (mne='DB') or (mne='DW') then  BEGIN  PSEUDOI:= TRUE;
            DefStorage(lab,mne,operf1,e);      END;
       if (mne='DS') or (mne='RS') then  BEGIN  PSEUDOI:= TRUE;
            ResStorage(lab,operf1,e);          END;
       if mne='ORG' then  BEGIN  PSEUDOI:= TRUE; Origin(operf1,e); END;
       if mne='END' then  BEGIN  PSEUDOI:= TRUE; Stp:= true; END;
       if e<>0 then    stp:=true
       else  if  pseudoi   then       begin
             writeln(lineNo:3,')     :',OneIns);
             lineNo:= lineNo+1;        end;
IF  NOT PSEUDOI  THEN
BEGIN
If lab <> '' Then  begin
```

```
    SymTable.Store(lab,LC,e);
    if e<>0 then  Stp:=true;   end;
If Not Stp Then
begin {1}
IF  MNE <> '' THEN
begin {2}
if operf1='' then  mde:='IMP'
else begin
      if operf2='' then begin   { ONE OPERAND }
         if havemneimp then  begin
            SearchImpMne(mne,e);
            if e=0 then begin
                mne:=mne+operf1; mde:='IMP'; end;
                              end;
         if (e<>0) or (Not havemneimp) then  if resinoprf  then  begin
                  SearchReserved(operf1,e);
                  if e=0 then begin
                     mne:=mne+operf1; mde:='IMP'; end;
                                                             end;
         if (e<>0) or Not(resinoprf) then
             AddrMd.Search(operf1,mne,mde,s1,s2,e);
                        end
       else begin                           { TWO OPERANDS }
            if havemneimp then begin
                SearchImpMne(mne,e);
                if e=0 then begin
                    mne:=mne+operf1+operf2; mde:='IMP'; end;
                               end;
            if (e<>0) or (Not havemneimp) then if resinoprf  then  begin
                SearchReserved(operf1,e);
                if e=0 then begin
                   SearchReserved(operf2,e);
                   if e=0 then begin
                       mne:=mne+operf1+operf2; mde:='IMP' end
                   else begin
                       AddrMd.Search(operf2,mne,mde,s1,s2,e);
                       mne:=mne+operf1;   end;
                            end
```

```
                else  begin
                   SearchReserved(operf2,e);
                   if e=0 then begin
                       AddrMd.Search(operf1,mne,mde,s1,s2,e);
                       mne:=mne+'_'+operf2;  end
                   else begin
                       operf1:= operf1+OPRSEP+operf2; operf2:='';
                       AddrMd.Search(operf1,mne,mde,s1,s2,e); end;
                       end;
                                               end else e:= -5;
            end;
      end;
      if e<>0 then   Stp:=true;
If Not Stp Then  begin  setwindow(RstWd); WRITELN(MNE,' ',MDE);
      ITable.Search(mne,mde,InsData,e);
      if e=0 then begin      New(exepoint);
          with exepoint^ do  begin
          line:=LineNo;   operand1:=0;  operand2:=0;
          Location:=LC; Instruction:=InsData;
          if mde<> 'IMP' then  begin
              ManageOper(s1,operand1,e);
              if (e=0) and (s2<>'') then ManageOper(s2,operand2,e);
              if e<>0 then e:= -7;  end;
Writeln(lineNo:3,') ',ByteToHes(Hi(LC))+ByteToHes(Lo(LC)),':',OneIns);
                              end;
          ExeTable[ec] := exepoint; oprs1^[ec]:=s1; oprs2^[ec]:=s2;
          LC:= LC+InsData.bytes;
          ec:= ec+1;
          lineNo:= lineNo+1;
                  end  { e=0 }
      else    Stp:=true;
                   end;  end; {2}  end; {1}
END;  end;  end;

PROCEDURE ASMPASS1(var fv:text; var errline:word);
var Stop: boolean;  aer:integer;  Oneline: string;
begin
     Stop:= false;  lineNo:= 1;  ec:= 1;  LC:= PC;  aer:=0;
```

```
     While Not Eof(fv) and Not Stop Do begin
        Readln(fv,Oneline);
        AsmOneIns(Oneline,aer,Stop);
                                        end;
     EndEC:= ec;
     if (aer<>0) and Stop then begin Errline:= LineNo; ErrorMode(aer); end;
end;

PROCEDURE ASMPASS2(var errline:word);
var  stop:boolean;  aer:integer;  s1,s2:string;
begin
Stop:= false;  aer:=0;    ec:=1;
While  (ec < EndEc) and Not Stop Do  begin
    with ExeTable[ec]^ do begin
    If Instruction.mode <> 'IMP' Then begin
         s1:= oprs1^[ec];  s2:= oprs2^[ec];
         ManageOper(s1,operand1,aer);
         if (aer=0) and (s2<>'') then ManageOper(s2,operand2,aer);
         if aer<>0 then  begin  Errormode(aer);
             Stop:=true; Errline:=line;   end;
                                       end;
                           end;
     ec:= ec+1;                       end;
 end;□
```

```
{ ---------------Ins. Data Preparation------------------}
TYPE  Charset= set of char;   arr40ofstr= array[1..40] of string;

  specT = record  t:char;  i:byte;  end;
  condT = record  tp:char; id:byte; st:boolean;  end;
  ExeInsT = RECORD
      case exk:byte of
      1: (v1: byte);
      2: (v2: byte; spc: specT);
      3: (v3: byte; spc1,spc2: specT);
      4: (ucond: condT);
      5: (icond5: condT; v5: byte);
      6: (icond6: condT; v6: byte; spc6: specT);
      7: (icond7: condT; v7: byte; spca,spcb: specT);
      8: (uconda,ucondb: condT; lg: boolean);
      end;

ExeInsRecT = record  opcode: str12;  mnemonic :str12;  mode:str8;
                     Bytes : byte;   Genericmode :str8;
                     gins : array[1..15] of ExeInsT; micron:byte; end;

PROCEDURE VarToReg(vs:string; var ch:char; var nm:byte);
begin
     IF HAVER8 THEN SearchR8name(vs,ch,nm);
     if (ch='x') AND HAVER16 then SearchR16name(vs,ch,nm);
     if ch='x' then SearchF1name(vs,ch,nm);
     if ch='x' then SearchGname(vs,ch,nm);
end;
PROCEDURE SearchGr1Ins(s:string; var vout:byte; var r:integer);
begin
     r:=0;
     case s[1] of
     'J': if s='JUMP' then  vout:=1
          else if s='JUMPSUB'  then  vout:=2;
     'N': if s='NOP'  then  vout:=0;
     'R': if s='RETSUB'  then  vout:=3;
     'S': if s='STOP' then  vout:=4;
     else    r:= -11;
```

```
    end;
end;
PROCEDURE SearchGr2Ins(s:string; var vout:byte; var r:integer);
begin
    r:=0;
    case s[1] of
    'A': if s= 'ASHR' then vout:= 5;
    'C': if s= 'CLEAR' then  vout:= 6
         else  if s= 'COMPLEMENT' then vout:= 7;
    'D': if s= 'DECADJ'  then  vout:= 8
         else if s= 'DECREMENT'  then vout:= 9;
    'I': if s= 'INCREMENT' then  vout:= 10
         else if s= 'INPUT' then vout:= 11
              else if s= 'INVERT' then vout:= 12;
    'J': if s= 'JUMPTO'  then  vout:= 13;
    'L': if s= 'LOAD' then  vout:= 14
         else if s= 'LSHR'  then vout:= 15;
    'O': if s= 'OUTPUT' then vout:= 16;
    'P': case s[3] of
         'L': if s= 'PULL' then vout:= 17
              else if s= 'PULL65' then vout:= 18;
         'S': if s= 'PUSH' then vout:= 19
              else if s= 'PUSH65' then vout:= 20;
         end;
    'R': case s[3] of
         'S': if s= 'RESET'  then vout:= 21;
         'L': if s= 'ROL'    then vout:= 22
              else if s= 'ROLCARRY'  then vout:= 23;
         'R': if s= 'ROR' then  vout:= 24
              else if s= 'RORCARRY'  then  vout:= 25;
         end;
    'S': if s= 'SET' then vout:= 26
         else if s= 'SHL' then vout:= 27
              else if s= 'STORE'  then vout:= 28;
    'T': case s[5] of
         'P': if s= 'TESTP'  then vout:= 29;
         'S': if s= 'TESTSZ' then vout:= 30
              else if s='TESTSIGN' then vout:=49;
```

```
            'Z': if s= 'TESTZ'  then vout:= 31;
            'O': if s= 'TWOCOMP' then vout:= 32;
            end;
        else   r:= -11;
        end;
end;
PROCEDURE SearchGr3Ins(s:string; var vout:byte; var r:integer);
begin
     r:=0;
     case s[1] of
     'A': if s= 'ADC' then vout:= 33
          else if s= 'ADD' then vout:= 34
               else if s= 'AND' then vout:= 35;
     'C': if s= 'COMPARE' then vout:= 36
          else if s= 'CP65' then vout:= 37;
     'E': if s= 'EXCHANGE' then vout:= 38;
     'I': if s= 'INBYPTR' then vout:= 39;
     'L': if s= 'LOADBYPTR' then vout:= 40;
     'O': if s= 'OR' then vout:= 41
          else if s= 'OUTBYPTR' then vout:= 42;
     'R': if s= 'REPLACE' then vout:= 43;
     'S': if s= 'SBC' then vout:= 44
          else if s= 'STOREBYPTR' then vout:= 45
               else if s= 'SUB' then vout:= 46;
     'T': if s= 'TRANSFER' then vout:= 47;
     'X': if s= 'XOR' then vout:= 48;
     else  r:= -11;
     end;
end;
PROCEDURE Token(s:string;sep:CharSet;n:byte;var tk:arr40ofstr);
Var i,p:byte;
Begin
     if s<>'' then begin  s:=s+^M;
        for i:=1 to n do tk[i] := ''; i:=1; p:=1;
        While s[p] <> ^M Do begin
          while (s[p] in sep) do p:=p+1;
          while Not(s[p] in sep) do begin
             tk[i] := tk[i]+s[p];  p:=p+1; end;
```

```
        i:=i+1;          end;    end;
End;
PROCEDURE TokenOneIns(ist:str40; var inso:ExeInsT; var e:integer);
const InsTkSep :set of char =[' ',',',':','=','/',^I,^M];
var T:arr40ofstr;  i,n:byte;
begin
    If ist<>'' Then BEGIN
      Token(ist,InsTkSep,7,T);
      i:=1; while T[i]<>'' do i:=i+1;  n:=i-1;
      With inso Do Begin
      exk:=n;
      if (n=7) and (T[1]='WHILE') then exk:=8;
      case exk of
      1: SearchGr1Ins(T[1],v1,e);
      2:begin SearchGr2Ins(T[1],v2,e); VarToReg(T[2],spc.t,spc.i); end;
      3:begin SearchGr3Ins(T[1],v3,e); VarToReg(T[2],spc1.t,spc1.i);
        VarToReg(T[3],spc2.t,spc2.i);   end;
      4:begin VarToReg(T[2],ucond.tp,ucond.id);
        if (T[3]='SET') OR (T[3]='NONZERO') then ucond.st:=true
        else  ucond.st:=false;   end;
      5:begin VarToReg(T[2],icond5.tp,icond5.id);
        if (T[3]='SET') OR (T[3]='NONZERO') then icond5.st:=true
        else  icond5.st:=false;
        SearchGr1Ins(T[5],v5,e);  end;
      6:begin  VarToReg(T[2],icond6.tp,icond6.id);
        if (T[3]='SET') OR (T[3]='NONZERO') then icond6.st:=true
        else icond6.st:= false;
        SearchGr2Ins(T[5],v6,e);  VarToReg(T[6],spc6.t,spc6.i);end;
      7:begin  VarToReg(T[2],icond7.tp,icond7.id);
        if (T[3]='SET') OR (T[3]='NONZERO') then icond7.st:=true
        else icond7.st:= false;
        SearchGr3Ins(T[5],v7,e);
        VarToReg(T[6],spca.t,spca.i); VarToReg(T[7],spcb.t,spcb.i); end;
      8:begin
        VarToReg(T[2],uconda.tp,uconda.id);
        VarToReg(T[5],ucondb.tp,ucondb.id);
        if (T[3]='SET') OR (T[3]='NONZERO') then uconda.st:=true
        else uconda.st:=false;
```

```
          if (T[6]='SET') OR (T[6]='NONZERO') then ucondb.st:=true
          else ucondb.st:=false;
          if T[4]='AND' then  lg:=true
          else  lg:=false;              end;
        end;          End;
                      END;
end;
PROCEDURE Edittedifi2Exeifi;
Type ar20ofstr40 = array[1..20] of str40;
Const ifn: string = 'a:uins.dat';  eifin:string = 'a:eins.dat';
Var  insfi:File of ar20ofstr40;   irec:ar20ofstr40;
     exeirec: ExeInsRecT;      r:integer;   i,j:byte;
     ExeInsfi: file of ExeInsRecT;
begin     R:=0;
     assign(insfi,ifn);  reset(insfi);
     assign(ExeInsfi,eifin);  rewrite(Exeinsfi);
     While Not Eof(insfi) Do begin
        Read(insfi,irec);
        With exeirec Do Begin
        opcode:= irec[1];  mnemonic:= irec[2]; mode:= irec[3];
        Val(irec[4],bytes,r);  genericmode:= irec[5];
        i:=1;   j:=i+5;
        While (j<=20) and (irec[j]<>'') Do  begin
           TokenOneIns(irec[j],exeirec.gins[i],r);
           if r<>0 then begin writeln('Unknown Instruction!'); exit; end;
           i:=i+1;  j:=j+1;    end;  micron:= i-1; End;
        Write(Exeinsfi,exeirec);  end;
      close(insfi);  close(Exeinsfi);
 end;□
```

```
{ ----------------Ins. Interpreter-----------------}
Var xn: array[1..4] of byte;

PROCEDURE Addressing(m:str8; opn1,opn2:word);
Var  disp: integer;   TAd: word;
begin
     case m[2] of
     'B': if m= 'ABS' then EA:= opn1;
     'M': if m= 'IMM' then IMMDAT:= opn1;
     'O': if m= 'IO'  then IOAd:= opn1 mod 256;
     'E': ;
     'D': If m= 'IDR' then LoadWptr(EA,opn1)
   Else   case m[4] of
            'A':if (m='IDXA') or (m='IDXAIMM') then
                begin if opn1>127 then  disp:= opn1-256
                      else   disp:= opn1;
                      EA:= R16T[xn[1]]^.getw + disp;
                      if m= 'IDXAIMM' then   IMMDAT:= opn2 mod 256;
                end;
            'B':if (m='IDXB') or (m='IDXBIMM') then
                begin if opn1>127 then  disp:= opn1-256
                      else   disp:= opn1;
                      EA:= R16T[xn[2]]^.getw + disp;
                      if m= 'IDXBIMM' then   IMMDAT:= opn2 mod 256;
                end;
            '1':if m= 'IDX1' then  EA:= R8T[xn[3]]^.v + (opn1 mod 256)
                else if m= 'IDX1IDR' then begin
                     EA:= R8T[xn[3]]^.v + opn1;
                     EA:= EA mod 256;  LoadW(TAd); EA:= TAd; end;
            '2':if m= 'IDX2' then
                     EA:= R8T[xn[4]]^.v + (opn1 mod 256);
            'I':if m= 'IDRIDX2' then begin
                     LoadWptr(EA,opn1);
                     EA:= EA + R8T[xn[4]]^.v;  end;
            end;
      end;
 end;
```

```
procedure itpLoad    (tx:char;nx:byte; var e:integer);
begin
     case tx of
     'r':   R8T[nx]^.Load;
     'h':   R16T[nx]^.hb.Load;
     'l':   R16T[nx]^.lb.Load;
     'w':   R16T[nx]^.Load;
     's':   LoadW(SP);
     'p':   LoadW(PC);
     else  e:= -12;
     end;
     if (tx = ftp) and (nx = fid) then  Updateflag;
end;
procedure itpStore   (tx:char;nx:byte; var e:integer);
begin
     case tx of
     'r': R8T[nx]^.Store;
     'w': R16T[nx]^.Store;
     'h': R16T[nx]^.hb.Store;
     'l': R16T[nx]^.lb.Store;
     's': StoreW(SP);
     'p': StoreW(PC);
     else  e:= -12;  end;
end;
procedure itpPush    (tx:char;nx:byte; var e:integer);
var  m:byte;
begin
     case tx of
     'r': R8T[nx]^.push;
     'w': R16T[nx]^.push;
     'h': R16T[nx]^.hb.push;
     'l': R16T[nx]^.lb.push;
     'p': PushW(PC);
else  e:= -12;     end;
end;
procedure itpPull    (tx:char;nx:byte; var e:integer);
begin
     case tx of
```

```
     'r': R8T[nx]^.pull;
     'w': R16T[nx]^.pull;
     'h': R16T[nx]^.hb.pull;
     'l': R16T[nx]^.lb.pull;
     'p': PullW(PC);
else  e:= -12;     end;
     if (tx = ftp) and (nx = fid) then  Updateflag;
end;
procedure itpPush6502(tx:char;nx:byte; var e:integer);
begin
     case tx of
     'r': R8T[nx]^.push65;
     'w': begin R16T[nx]^.hb.push65;  R16T[nx]^.lb.push65; end;
     'h': R16T[nx]^.hb.push65;
     'l': R16T[nx]^.lb.push65;
     'p': Push6502W(PC);
else  e:= -12;     end;
end;
procedure itpPull6502(tx:char;nx:byte; var e:integer);
begin
     case tx of
     'r': R8T[nx]^.pull65;
     'w': begin R16T[nx]^.lb.pull65;  R16T[nx]^.hb.pull65;  end;
     'h': R16T[nx]^.hb.pull65;
     'l': R16T[nx]^.lb.pull65;
     'p': Pull6502W(PC);
else  e:= -12;     end;
     if (tx = ftp) and (nx = fid) then  Updateflag;
end;
procedure itpClear   (tx:char;nx:byte; var e:integer);
begin
     case tx of
     'r': R8T[nx]^.Clear;
     'w': R16T[nx]^.Clear;
     'h': R16T[nx]^.hb.Clear;
     'l': R16T[nx]^.lb.Clear;
else  e:= -12;     end;
     if (tx = ftp) and (nx = fid) then  Updateflag;
```

```
end;
procedure itpTestZero(tx:char;nx:byte; var e:integer);
begin
     case tx of
     'r': R8T[nx]^.TestZ;
     'h': R16T[nx]^.hb.TestZ;
     'l': R16T[nx]^.lb.TestZ;
     'w': with R16T[nx]^ do Zero:= ((hb.v=0) and (lb.v=0)) ;
else  e:= -12;      end; UpdateFreg;
end;
procedure itpTestSignZero (tx:char;nx:byte; var e:integer);
begin
     case tx of
     'r': R8T[nx]^.TestSZ;
     'h': R16T[nx]^.hb.TestSZ;
     'l': R16T[nx]^.lb.TestSZ;
else  e:= -12;      end;
     UpdateFreg;
end;
procedure itpTestSign (tx:char; nx:byte; var e:integer);
begin
     case tx of
     'r':  R8T[nx]^.TestS;
     'h':  R16T[nx]^.hb.TestS;
     'l':  R16T[nx]^.lb.TestS;
     'w':  R16T[nx]^.hb.TestS;
else  e:= -12;      end;  UpdateFreg;
end;
procedure itpTestParity (tx:char;nx:byte; var e:integer);
begin
     case tx of
     'r': R8T[nx]^.TestP;
     'h': R16T[nx]^.hb.TestP;
     'l': R16T[nx]^.lb.TestP;
else  e:= -12;      end;
     UpdateFreg;
end;
procedure itpSet     (tx:char;nx:byte; var e:integer);
```

```
begin
     if tx='f' then begin
        FlT[nx]^.setf;
        UpdateFreg;  end  else  e:= -12;
end;
procedure itpReset   (tx:char;nx:byte; var e:integer);
begin
     if tx='f' then begin
        FlT[nx]^.resetf;
        UpdateFreg;  end  else  e:= -12;
end;
procedure itpInvert  (tx:char;nx:byte; var e:integer);
begin
     if tx='f' then begin
        FlT[nx]^.invertf;
        UpdateFreg;  end  else  e:= -12;
end;
procedure itpInc     (tx:char;nx:byte; var e:integer);
var m,sh,sl:byte;  sold:word;
begin
     If (tx=ftp) and (nx=fid) Then  begin e:= -12; exit; end;
     case tx of
     'r': R8T[nx]^.Inc;
     'w': R16T[nx]^.Inc;
     'h': R16T[nx]^.hb.Inc;
     'l': R16T[nx]^.lb.Inc;
     'm': begin m:= Mem[Hi(EA)]^[Lo(EA)]; AddReg(m,1,false);
                Mem[Hi(EA)]^[Lo(EA)] := m;              end;
     's': case nx of
          0: IncW(sp);
        1,2: begin sh:= Hi(sp); sl:= Lo(sp);  sold:= sp;
             if nx=1 then Addreg(sh,1,false) else Addreg(sl,1,false);
             sp:= sh shl 8 + sl;  TestZero(sp); TestSignW(sp);
             Testoverflow(sp,sold);  end;
        end;
     end;   UpdateFreg;
end;
procedure itpDec     (tx:char;nx:byte; var e:integer);
```

```
var m,sh,sl:byte;  sold:word;
begin
     If (tx=ftp) and (nx=fid) Then  begin  e:= -12; exit; end;
     case tx of
     'r': R8T[nx]^.Dec;
     'w': R16T[nx]^.Dec;
     'h': R16T[nx]^.hb.Dec;
     'l': R16T[nx]^.lb.Dec;
     'm': begin m:= Mem[Hi(EA)]^[Lo(EA)]; SubReg(m,1,false);
                Mem[Hi(EA)]^[Lo(EA)] := m;              end;
     's': case nx of
          0: DecW(sp);
        1,2: begin sh:= Hi(sp); sl:= Lo(sp);  sold:= sp;
             if nx=1 then Subreg(sh,1,false) else Subreg(sl,1,false);
             sp:= sh shl 8 + sl;  TestZero(sp); TestSignW(sp);
             Testoverflow(sp,sold);  end;
        end;
     end;  UpdateFreg;
end;
procedure itpComplement (tx:char;nx:byte; var e:integer);
var m:byte;
begin
     If (tx=ftp) and (nx=fid) Then  begin  e:= -12;  exit;  end;
     case tx of
     'r': R8T[nx]^.Complement;
     'h': R16T[nx]^.hb.Complement;
     'l': R16T[nx]^.lb.Complement;
     'm': begin m:= Mem[Hi(EA)]^[Lo(EA)]; ComplementReg(m);
                Mem[Hi(EA)]^[Lo(EA)] := m;              end;
     end;
end;
procedure itpTwocomp (tx:char;nx:byte; var e:integer);
var m:byte;
begin
     If (tx=ftp) and (nx=fid) Then  begin  e:= -12;  exit;  end;
     case tx of
     'r': R8T[nx]^.TwoComplement;
     'h': R16T[nx]^.hb.TwoComplement;
```

```
     'l': R16T[nx]^.lb.TwoComplement;
     'm': begin m:= Mem[Hi(EA)]^[Lo(EA)]; TwoCompReg(m);
                Mem[Hi(EA)]^[Lo(EA)] := m;              end;
     end;  UpdateFreg;
end;
procedure itpDecAdj  (tx:char;nx:byte; var e:integer);
var m:byte;
begin
     If (tx=ftp) and (nx=fid) Then  begin  e:= -12; exit; end;
     case tx of
     'r': R8T[nx]^.DecimalAdj;
     'h': R16T[nx]^.hb.DecimalAdj;
     'l': R16T[nx]^.lb.DecimalAdj;
     'm': begin m:= Mem[Hi(EA)]^[Lo(EA)]; DecAdjust(m);
                Mem[Hi(EA)]^[Lo(EA)] := m;              end;
     end;   UpdateFreg;
end;
procedure itpShiftLeft (tx:char;nx:byte; var e:integer);
var m:byte;
begin
     If (tx=ftp) and (nx=fid) Then  begin  e:= -12; exit; end;
     case tx of
     'r': R8T[nx]^.ShiftL;
     'h': R16T[nx]^.hb.ShiftL;
     'l': R16T[nx]^.lb.ShiftL;
     'm': begin m:= Mem[Hi(EA)]^[Lo(EA)]; ShiftLeft(m);
                Mem[Hi(EA)]^[Lo(EA)] := m;              end;
     end;  UpdateFreg;
end;
procedure itpAshiftRight (tx:char;nx:byte; var e:integer);
var m:byte;
begin
     If (tx=ftp) and (nx=fid) Then  begin e:= -12; exit; end;
     case tx of
     'r': R8T[nx]^.AshiftR;
     'h': R16T[nx]^.hb.AshiftR;
     'l': R16T[nx]^.lb.AshiftR;
     'm': begin m:= Mem[Hi(EA)]^[Lo(EA)]; AshiftRight(m);
```

```
                Mem[Hi(EA)]^[Lo(EA)] := m;              end;
     end; UpdateFreg;
end;
procedure itpLshiftRight (tx:char;nx:byte; var e:integer);
var m:byte;
begin
     If (tx=ftp) and (nx=fid) Then  begin e:= -12; exit; end;
     case tx of
     'r': R8T[nx]^.LshiftR;
     'h': R16T[nx]^.hb.LshiftR;
     'l': R16T[nx]^.lb.LshiftR;
     'm': begin m:= Mem[Hi(EA)]^[Lo(EA)]; LshiftRight(m);
                Mem[Hi(EA)]^[Lo(EA)] := m;              end;
     end; UpdateFreg;
end;
procedure itpRotLeft (tx:char;nx:byte; var e:integer);
var m:byte;
begin
     If (tx=ftp) and (nx=fid) Then  begin e:= -12; exit; end;
     case tx of
     'r': R8T[nx]^.RoL;
     'h': R16T[nx]^.hb.RoL;
     'l': R16T[nx]^.lb.RoL;
     'm': begin m:= Mem[Hi(EA)]^[Lo(EA)]; RotLeft(m);
                Mem[Hi(EA)]^[Lo(EA)] := m;              end;
     end;  UpdateFreg;
end;
procedure itpRotRight(tx:char;nx:byte; var e:integer);
var m:byte;
begin
     If (tx=ftp) and (nx=fid) Then begin e:= -12; exit; end;
     case tx of
     'r': R8T[nx]^.RoR;
     'h': R16T[nx]^.hb.RoR;
     'l': R16T[nx]^.lb.RoR;
     'm': begin m:= Mem[Hi(EA)]^[Lo(EA)]; RotRight(m);
                Mem[Hi(EA)]^[Lo(EA)] := m;              end;
     end;  UpdateFreg;
```

```
end;
procedure itpROLcarry(tx:char;nx:byte; var e:integer);
var m:byte;
begin
     If (tx=ftp) and (nx=fid) Then begin e:= -12;  exit; end;
     case tx of
     'r': R8T[nx]^.ROLcarry;
     'h': R16T[nx]^.hb.ROLcarry;
     'l': R16T[nx]^.lb.ROLcarry;
     'm': begin m:= Mem[Hi(EA)]^[Lo(EA)]; ROL_carry(m);
                Mem[Hi(EA)]^[Lo(EA)] := m;              end;
     end;  UpdateFreg;
end;
procedure itpRORcarry(tx:char;nx:byte; var e:integer);
var m:byte;
begin
     If (tx=ftp) and (nx=fid) Then begin e:= -12; exit; end;
     case tx of
     'r': R8T[nx]^.RORcarry;
     'h': R16T[nx]^.hb.RORcarry;
     'l': R16T[nx]^.lb.RORcarry;
     'm': begin m:= Mem[Hi(EA)]^[Lo(EA)]; ROR_carry(m);
                Mem[Hi(EA)]^[Lo(EA)] := m;              end;
     end;  UpdateFreg;
end;
procedure itpLdbyPtr (tx,ty:char; nx,ny:byte;var e:integer);
var ptr:word;
begin
     GetValue(ty,ny,ptr);
     case tx of
     'r':  R8T[nx]^.LdbyPtr(ptr);
     'w':  R16T[nx]^.LdbyPtr(ptr);
     'h':  R16T[nx]^.hb.LdbyPtr(ptr);
     'l':  R16T[nx]^.lb.LdbyPtr(ptr);
     's':  LoadWptr(sp,ptr);
     'p':  LoadWptr(pc,ptr);
     else  e:= -12;
     end;
```

```
     if (tx = ftp) and (nx = fid) then  Updateflag;
end;
procedure itpStbyPtr (tx,ty:char; nx,ny:byte;var e:integer);
var ptr:word;
begin
     GetValue(ty,ny,ptr);
     case tx of
     'r':  R8T[nx]^.StbyPtr(ptr);
     'w':  R16T[nx]^.StbyPtr(ptr);
     'h':  R16T[nx]^.hb.StbyPtr(ptr);
     'l':  R16T[nx]^.lb.StbyPtr(ptr);
     's': StoreWptr(sp,ptr);
     'p': StoreWptr(pc,ptr);
     else  e:= -12;
     end;
end;
procedure itpTransfer(tx,ty:char; nx,ny:byte;var e:integer);
var rv:byte;  rw:word;
begin
     if (tx='f') or (ty='f') then  e:= -12;
     case tx of
     'r': begin Getvalue(ty,ny,rv); R8T[nx]^.putv(rv);  end;
     'w': begin Getvalue(ty,ny,rw); R16T[nx]^.putw(rw);  end;
     'h': begin Getvalue(ty,ny,rv); R16T[nx]^.hb.putv(rv);  end;
     'l': begin Getvalue(ty,ny,rv); R16T[nx]^.lb.putv(rv);  end;
     'p': begin Getvalue(ty,ny,rw); pc:= rw; end;
     's': case nx of
          0: begin Getvalue(ty,ny,rw); SP:= rw;  end;
          1: begin Getvalue(ty,ny,rv); SP:= rv shl 8 + Lo(SP); end;
          2: begin Getvalue(ty,ny,rv); SP:= Hi(SP) shl 8 + rv; end; end;
     end;
     if (tx=ftp) and (nx=fid) then Updateflag;
end;
procedure itpExchange(tx,ty:char; nx,ny:byte;var e:integer);
var  r1,r2:byte;  w1,w2:word;
begin
     if (tx='f') or (ty='f') then  e:= -12;
     case tx of
```

```
      'r': begin getvalue(tx,nx,r1);  getvalue(ty,ny,r2);
                 R8T[nx]^.putv(r2);  R8T[ny]^.putv(r1); end;
      'w': begin getvalue(tx,nx,w1);  getvalue(ty,ny,w2);
                 R16T[nx]^.putw(w2);
                 if ty='w' then R16T[ny]^.putw(w1)
                 else if ty='s' then   sp:= w1;   end;
      'h': begin getvalue(tx,nx,r1);  getvalue(ty,ny,r2);
                 R16T[nx]^.hb.putv(r2);
                 if ty ='h' then R16T[ny]^.hb.putv(r1)
                 else if ty='l' then  R16T[ny]^.lb.putv(r1) end;
      'l': begin getvalue(tx,nx,r1);  getvalue(ty,ny,r2);
                 R16T[nx]^.lb.putv(r2);
                 if ty ='h' then R16T[ny]^.hb.putv(r1)
                 else if ty='l' then  R16T[ny]^.lb.putv(r1) end;
      's': begin
               getvalue(tx,nx,w1);  getvalue(ty,ny,w2);
               sp:= w2;
               if ty='w' then R16T[ny]^.putw(w1);  end;
      end;
      if (tx=ftp) and (nx=fid) then Updateflag;
      if (ty=ftp) and (ny=fid) then Updateflag;
end;
procedure itpAdd     (tx,ty:char; nx,ny:byte;var e:integer);
var  rv:byte;  rw:word;
begin
if (tx=ftp) and (nx=fid) then begin e:= -12; exit; end;
case tx of
'r': begin getvalue(ty,ny,rv); R8T[nx]^.add(rv); end;
'h': begin getvalue(ty,ny,rv); R16T[nx]^.hb.add(rv);  end;
'l': begin getvalue(ty,ny,rv); R16T[nx]^.lb.add(rv);  end;
'w': begin getvalue(ty,ny,rw); R16T[nx]^.add(rw); end;
's': begin getvalue(ty,ny,rw); AddW(sp,rw); end;
'p': begin getvalue(ty,ny,rw); AddW(pc,rw); end;
end;   UpdateFreg;
end;
procedure itpAdc     (tx,ty:char; nx,ny:byte;var e:integer);
var  rv:byte;  rw:word;
begin
```

```
if (tx=ftp) and (nx=fid) then begin e:= -12; exit; end;
case tx of
'r': begin getvalue(ty,ny,rv); R8T[nx]^.adc(rv); end;
'h': begin getvalue(ty,ny,rv); R16T[nx]^.hb.adc(rv);   end;
'l': begin getvalue(ty,ny,rv); R16T[nx]^.lb.adc(rv);   end;
'w': begin getvalue(ty,ny,rw); R16T[nx]^.adc(rw); end;
's': begin getvalue(ty,ny,rw); AdcW(sp,rw); end;
'p': begin getvalue(ty,ny,rw); AdcW(pc,rw); end;
end;   UpdateFreg;
end;
procedure itpSub     (tx,ty:char; nx,ny:byte;var e:integer);
var  rv:byte;  rw:word;
begin
if (tx=ftp) and (nx=fid) then begin e:= -12; exit; end;
case tx of
'r': begin getvalue(ty,ny,rv); R8T[nx]^.sub(rv); end;
'h': begin getvalue(ty,ny,rv); R16T[nx]^.hb.sub(rv);  end;
'l': begin getvalue(ty,ny,rv); R16T[nx]^.lb.sub(rv);  end;
'w': begin getvalue(ty,ny,rw); R16T[nx]^.sub(rw); end;
's': begin getvalue(ty,ny,rw); SubW(sp,rw);  end;
'p': begin getvalue(ty,ny,rw); SubW(pc,rw);  end;
end;    UpdateFreg;
end;
procedure itpSbc     (tx,ty:char; nx,ny:byte;var e:integer);
var  rv:byte;  rw:word;
begin
if (tx=ftp) and (nx=fid) then begin e:= -12; exit; end;
case tx of
'r': begin getvalue(ty,ny,rv); R8T[nx]^.sbc(rv); end;
'h': begin getvalue(ty,ny,rv); R16T[nx]^.hb.sbc(rv);  end;
'l': begin getvalue(ty,ny,rv); R16T[nx]^.lb.sbc(rv);  end;
'w': begin getvalue(ty,ny,rw); R16T[nx]^.sbc(rw); end;
's': begin getvalue(ty,ny,rw); SbcW(sp,rw); end;
'p': begin getvalue(ty,ny,rw); SbcW(pc,rw); end;
end;    UpdateFreg;
end;
procedure itpCompare (tx,ty:char; nx,ny:byte;var e:integer);
var  rv:byte;
```

```
begin
if (tx=ftp) and (nx=fid) then begin e:= -12; exit; end;
case tx of
'r': begin getvalue(ty,ny,rv); R8T[nx]^.compare(rv); end;
'h': begin getvalue(ty,ny,rv); R16T[nx]^.hb.compare(rv); end;
'l': begin getvalue(ty,ny,rv); R16T[nx]^.lb.compare(rv); end;
end;    UpdateFreg;
end;
procedure itpCp6502  (tx,ty:char; nx,ny:byte;var e:integer);
var  rv:byte;
begin
if (tx=ftp) and (nx=fid) then begin e:= -12; exit; end;
case tx of
'r': begin getvalue(ty,ny,rv); R8T[nx]^.cp6502(rv); end;
'h': begin getvalue(ty,ny,rv); R16T[nx]^.hb.cp6502(rv); end;
'l': begin getvalue(ty,ny,rv); R16T[nx]^.lb.cp6502(rv); end;
end;    UpdateFreg;
end;
procedure itpAnd     (tx,ty:char; nx,ny:byte;var e:integer);
var  rv:byte;
begin
if (tx=ftp) and (nx=fid) then begin e:= -12; exit; end;
case tx of
'r': begin getvalue(ty,ny,rv); R8T[nx]^.andlg(rv); end;
'h': begin getvalue(ty,ny,rv); R16T[nx]^.hb.andlg(rv);  end;
'l': begin getvalue(ty,ny,rv); R16T[nx]^.lb.andlg(rv);  end;
end;    UpdateFreg;
end;
procedure itpOr      (tx,ty:char; nx,ny:byte;var e:integer);
var  rv:byte;
begin
if (tx=ftp) and (nx=fid) then begin e:= -12; exit; end;
case tx of
'r': begin getvalue(ty,ny,rv); R8T[nx]^.orlg(rv); end;
'h': begin getvalue(ty,ny,rv); R16T[nx]^.hb.orlg(rv);  end;
'l': begin getvalue(ty,ny,rv); R16T[nx]^.lb.orlg(rv);  end;
end;    UpdateFreg;
end;
```

```
procedure itpXor     (tx,ty:char; nx,ny:byte;var e:integer);
var  rv:byte;
begin
if (tx=ftp) and (nx=fid) then begin e:= -12; exit; end;
case tx of
'r': begin getvalue(ty,ny,rv); R8T[nx]^.xorlg(rv); end;
'h': begin getvalue(ty,ny,rv); R16T[nx]^.hb.xorlg(rv);  end;
'l': begin getvalue(ty,ny,rv); R16T[nx]^.lb.xorlg(rv);  end;
end;    UpdateFreg;
end;
procedure itpReplacefv  (tx,ty:char; nx,ny:byte;var e:integer);
var  bv:boolean;
begin
      If (tx='f') and (ty='f') Then begin
           Getvalue(ty,ny,bv);
           F1T[nx]^.Putfv(bv);
           UpdateFreg;             end  else e:= -12;
end;
procedure itpInput   (tx:char; nx:byte; var e:integer);
begin
     if (tx=ftp) and (nx=fid) then begin e:= -12; exit; end;
     case tx of
     'r': R8T[nx]^.input;
     'h': R16T[nx]^.hb.input;
     'l': R16T[nx]^.lb.input;
     end;
end;
procedure itpOutput  (tx:char; nx:byte;var e:integer);
begin
     if (tx=ftp) and (nx=fid) then begin e:= -12; exit; end;
     case tx of
     'r': R8T[nx]^.output;
     'h': R16T[nx]^.hb.output;
     'l': R16T[nx]^.lb.output;      end;
end;
procedure itpInbyPtr (tx,ty:char; nx,ny:byte;var e:integer);
var   rv:byte;
begin
```

```
     if ((tx=ftp)and(nx=fid)) or ((ty=ftp)and(ny=fid)) then begin
        e:= -12;  exit; end;
     Getvalue(ty,ny,rv);
     case tx of
     'r': R8T[nx]^.InbyPtr(rv);
     'h': R16T[nx]^.hb.InbyPtr(rv);
     'l': R16T[nx]^.lb.InbyPtr(rv);
     end;
end;
procedure itpOutbyPtr(tx,ty:char; nx,ny:byte;var e:integer);
var   rv:byte;
begin
     if ((tx=ftp)and(nx=fid)) or ((ty=ftp)and(ny=fid)) then begin
         e:= -12;  exit;   end;
     Getvalue(ty,ny,rv);
     case tx of
     'r': R8T[nx]^.OutbyPtr(rv);
     'h': R16T[nx]^.hb.OutbyPtr(rv);
     'l': R16T[nx]^.lb.OutbyPtr(rv);
     end;
end;□
```

```
{ -----------------Register--------------------}
TYPE
     r8p = ^r8o;
     r8o = OBJECT  n: str4;  v:byte;
        procedure Init(ni:str4);
        function Getv:byte;        procedure Putv(vx:byte);
        procedure Clear;           procedure DecimalAdj;
        procedure Complement;      procedure Twocomplement;
procedure TestSZ; procedure TestP; procedure TestS; procedure TestZ;
        procedure Load;            procedure Store;
        procedure Input;           procedure Output;
        procedure Push;            procedure Pull;
        procedure Push65;        procedure Pull65;
        procedure Inc;             procedure Dec;
        procedure ShiftL;
        procedure AshiftR;     procedure LshiftR;
        procedure RoL;         procedure RoR;
        procedure ROLcarry;        procedure RORcarry;
        procedure InbyPtr(vx:byte);   procedure OutbyPtr(vx:byte);
        procedure Add(vx:byte);       procedure Adc(vx:byte);
        procedure Sub(vx:byte);       procedure Sbc(vx:byte);
        procedure Sbc6502(vx:byte);   procedure Cp6502(vx:byte);
        procedure Compare(vx:byte);   procedure ANDlg(vx:byte);
        procedure ORlg(vx:byte);        procedure XORlg(vx:byte);
        procedure LdbyPtr(px:word);    procedure StbyPtr(px:word);   end;
    r16p = ^r16o;
    r16o = OBJECT  wn:str4;   hb,lb:r8o;
        procedure Init(wni,hni,lni:str4);
        function Getw:word;
        procedure Putw(wx:word);
        procedure Load;               procedure Store;
        procedure Push;               procedure Pull;
        procedure Inc;                procedure Dec;
        procedure Clear;
        procedure LdbyPtr(p:word);    procedure StbyPtr(p:word);
        procedure Add(wx:word);       procedure Adc(wx:word);
        procedure Sub(wx:word);       procedure Sbc(wx:word);    end;
    flp = ^flo;
```

```
    flo = OBJECT
                fn:str2; gn:str10; fv:boolean;
        procedure Init(fni:str2; gni:str10);
        function Getfv:boolean;       procedure Putfv(fvx:boolean);
        procedure Setf;  procedure Resetf;   procedure Invertf;       end;

    WrecT= record  w,h,l: str4;  end;
    fT= record  sn: str2;  gn: str10;   end;
FrecT= record  rt:char; rn:byte;  FregN: str4; f:array[0..10] of fT; end;
    arecT= record  cms,ops: char;  mimp: array[1..10] of str6;
                rs: array[1..40] of str4;
                idxreg: array[1..4] of str4;          end;
ar15ofr8p  = array[1..17] of r8p;
ar15ofr16p = array[1..17] of r16p;
ar010offlp = array[0..10] of flp;
R8AP = ar15ofr8p;   R16AP = ar15ofr16p;
FlAP = ar010offlp;



VAR R8T: R8AP;  R16T: R16AP;  FlT: FlAP; R8Num,R16Num: byte;

    Fname:str4; Ftp:char; Fid:byte;
    haver8,haver16: boolean;

procedure r8o.Init(ni:str4);
begin  n:=ni;  v:=0;  end;
function r8o.Getv:byte;
begin  Getv:=v;       end;
procedure r8o.Putv(vx:byte);
begin  v:=vx;         end;
procedure r8o.Clear;
begin  v:=0;          end;
procedure r8o.DecimalAdj;
begin   DecAdjust(v);  end;
procedure r8o.Complement;
begin  ComplementReg(v); end;
procedure r8o.Twocomplement;
```

```
begin  TwocompReg(v);  end;
procedure r8o.TestSZ;
begin  TestSignZero(v);  end;
procedure r8o.TestS;
begin  TestSign(v); end;
procedure r8o.TestZ;
begin  TestZero(v); end;
procedure r8o.TestP;
begin  TestParity(v);  end;
procedure r8o.Load;
begin  LoadB(v); end;
procedure r8o.Store;
begin  StoreB(v);  end;
procedure r8o.Input;
begin  v:= IOspace[IOad];  end;
procedure r8o.Output;
begin  IOspace[IOad] := v; end;
procedure r8o.Push;
begin  pushB(v);  end;
procedure r8o.Pull;
begin  pullB(v);  end;
procedure r8o.Push65;
begin  push6502(v); end;
procedure r8o.Pull65;
begin  pull6502(v); end;
procedure r8o.Inc;
begin  AddReg(v,1,false); end;
procedure r8o.Dec;
begin  SubReg(v,1,false); end;
procedure r8o.ShiftL;
begin  ShiftLeft(v);  end;
procedure r8o.AshiftR;
begin  AshiftRight(v); end;
procedure r8o.LshiftR;
begin  LshiftRight(v); end;
procedure r8o.RoL;
begin  RotLeft(v); end;
procedure r8o.RoR;
```

```
begin  RotRight(v) end;
procedure r8o.ROLcarry;
begin  ROL_carry(v); end;
procedure r8o.RORcarry;
begin  ROR_carry(v); end;
procedure r8o.InbyPtr(vx:byte);
begin   v:= IOspace[vx]; end;
procedure r8o.OutbyPtr(vx:byte);
begin  IOspace[vx] := v; end;
procedure r8o.Add(vx:byte);
begin  AddReg(v,vx,false); end;
procedure r8o.Adc(vx:byte);
begin  AddReg(v,vx,true);  end;
procedure r8o.Sub(vx:byte);
begin  SubReg(v,vx,false); end;
procedure r8o.Sbc(vx:byte);
begin  SubReg(v,vx,true); end;
procedure r8o.Compare(vx:byte);
begin  CompareReg(v,vx); end;
      procedure r8o.Sbc6502(vx:byte);
      begin Carry:=Not(carry); SubReg(v,vx,true); Carry:=Not(carry); end;
      procedure r8o.Cp6502(vx:byte);
      begin Carry:=Not(carry); CompareReg(v,vx); Carry:=Not(carry); end;
procedure r8o.ANDlg(vx:byte);
begin  andlogic(v,vx);  end;
procedure r8o.ORlg(vx:byte);
begin  orlogic(v,vx); end;
procedure r8o.XORlg(vx:byte);
begin  xorlogic(v,vx); end;
procedure r8o.LdbyPtr(px:word);
begin  LoadBptr(v,px); end;
procedure r8o.StbyPtr(px:word);
begin  StoreBptr(v,px);  end;
{--------------------------------------------------}
procedure r16o.Init(wni,hni,lni:str4);
begin  wn:=wni;   hb.init(hni);  lb.init(lni); end;
function r16o.Getw:word;
begin  getw:= hb.v shl 8 + lb.v;  end;
```

```
procedure r16o.Putw(wx:word);
begin  hb.v:= Hi(wx);  lb.v:= Lo(wx); end;
procedure r16o.Clear;
begin  hb.clear;  lb.clear;  end;
procedure r16o.Load;
begin  lb.Load;  EA:=EA+1;  hb.Load;  end;
procedure r16o.Store;
begin  lb.Store;  EA:=EA+1;  hb.Store;  end;
procedure r16o.Push;
begin  hb.push;  lb.push;  end;
procedure r16o.Pull;
begin  lb.pull;  hb.pull;  end;
procedure r16o.Inc;
begin  lb.add(1);  hb.adc(0);  end;
procedure r16o.Dec;
begin  lb.sub(1);  hb.sbc(0);  end;
procedure r16o.LdbyPtr(p:word);
begin lb.LdByPtr(p); p:=p+1; hb.LdbyPtr(p);  end;
procedure r16o.StbyPtr(p:word);
begin lb.StbyPtr(p); p:=p+1; hb.StbyPtr(p); end;
procedure r16o.Add(wx:word);
var l,h:byte;
begin  l:=Lo(wx);  h:=Hi(wx);
       lb.add(l);  hb.adc(h);
end;
procedure r16o.Adc(wx:word);
var l,h:byte;
begin  l:=Lo(wx);  h:=Hi(wx);
       lb.adc(l);  hb.adc(h);
end;
procedure r16o.Sub(wx:word);
var  l,h:byte;
begin  l:= Lo(wx);   h:= Hi(wx);
       lb.sub(l);  hb.sbc(h);
end;
procedure r16o.Sbc(wx:word);
var  l,h:byte;
begin  l:= Lo(wx);  h:= Hi(wx);
```

```
       lb.sbc(l);  hb.sbc(h);
end;
{----------------------------------------------------}
procedure flo.Init(fni:str2; gni:str10);
var  b:boolean;
begin fn:=fni;  gn:=gni;
      if gn = '' then  fv:= false
      else  ProcessGFlag(gn,'r',b);
end;
function flo.Getfv:boolean;
var b:boolean;
begin  if gn = '' then  getfv:= fv
       else   begin ProcessGFlag(gn,'o',b);  getfv:=b; end;
end;
procedure flo.Putfv(fvx:boolean);
begin if gn = '' then  fv:= fvx
      else  ProcessGFlag(gn,'i',fvx);
end;
procedure flo.Setf;
var b:boolean;
begin  if  gn=''  then  fv:= true
       else   ProcessGFlag(gn,'s',b);
end;
procedure flo.Resetf;
var b:boolean;
begin  if gn=''  then  fv:= false
       else   ProcessGFlag(gn,'r',b);
end;
procedure flo.Invertf;
var b:boolean;
begin  if gn=''  then  fv:= Not(fv)
       else   ProcessGflag(gn,'n',b);
end;

PROCEDURE SearchR8name (namein:string; var k:char; var c:byte);
var i:byte;
begin
     i:=1;
```

```
     while (i<R8num) and (namein <> R8T[i]^.n) do i:=i+1;
     if namein = R8T[i]^.n  then  begin
         k:='r';  c:=i;            end
     else  k:='x';
end;
PROCEDURE SearchR16name(namein:string; var k:char; var c:byte);
var  i:byte;
begin
     k:= 'x';  c:= 0;   i:= 1;
     With R16T[i]^  Do  Begin
     while (i<R16num) and (namein<>wn)and(namein<>hb.n)and(namein<>lb.n)
         do i:=i+1;
if namein = wn  then  begin k:= 'w'; c:= i; end
else if namein = hb.n  then  begin k:= 'h'; c:= i; end
     else if namein = lb.n  then  begin k:= 'l'; c:= i; end;  End;
end;

PROCEDURE InitRegister;
CONST rfn: string = 'a:r8.dat'; wfn: string = 'a:r16.dat';
      ffn: string = 'a:flg.dat';
Var rrec:str4;   i:byte;  wrec:WrecT;  frec:FrecT;
    rf: File of str4; wf: File of WrecT; ff: File of FrecT;
Begin      ftp:= 'x';
     assign(rf,rfn); {$I-}  reset(rf); {$I+}
If IORESULT = 0  Then Begin    haver8:= true;
     i:=1;   reset(rf);
     While Not Eof(rf) Do  begin
         Read(rf,rrec);  New(R8T[i]); R8T[i]^.init(rrec);
         i:=i+1;
                           end;
     close(rf);   R8num:= i-1+2;
     New(R8T[R8num-1]); R8T[R8num-1]^.Init('T1');
     New(R8T[R8num]);   R8T[R8num]^.Init('T2');
                    End
Else haver8:= false;
     assign(wf,wfn);  {$I-}  reset(wf);  {$I+}
If IOResult = 0  Then  Begin    haver16:= true;
     i:=1;  reset(wf);
```

```
     While Not Eof(wf)  Do begin
         Read(wf,wrec);  New(R16T[i]);
         R16T[i]^.init(wrec.w,wrec.h,wrec.l);
         i:=i+1;            end;
     close(wf);    R16num:= i-1+2;
     New(R16T[R16num-1]); R16T[R16num-1]^.Init('T3','T3H','T3L');
     New(R16T[R16num]);   R16T[R16num]^.Init('T4','T4H','T4L');
                         End
Else haver16:= false;
     assign(ff,ffn);  {$i-} reset(ff); {$i+}
If ioresult=0 Then Begin  reset(ff);
     read(ff,frec);   close(ff);
     Ftp:= frec.rt;   Fid:= frec.rn; Fname:= frec.FregN;
     For i:=0 to 10  do  begin New(F1T[i]);
     F1T[i]^.init(frec.f[i].sn,frec.f[i].gn); end;
                    End
Else begin writeln('ERROR...NO FLAG DATA FILE!'); exit; end;
if haver8  then    SearchR8name(Fname,Ftp,Fid);
if (ftp='x') and haver16  then  SearchR16name(Fname,ftp,fid);
     if ftp='x' then writeln('ERROR...NO FLAG REGISTER!');
End;


PROCEDURE SearchF1name (namein:string; var k:char; var c:byte);
var i:byte;
begin
     i:=0;
     While (i<10) and (namein<>F1T[i]^.fn)  do i:=i+1;
     if namein = F1T[i]^.fn  then  begin
         k:='f';  c:=i;            end
     else  k:='x';
end;
PROCEDURE SearchGname  (namein:string; var k:char; var c:byte);
var notnum:boolean;  e:integer; wout:word;
begin
     k:='x';  c:=0;
     notnum := CheckSym(namein);
     If notnum Then begin
         case namein[1] of
```

```
        'S' : if namein='SP' then  k:='s'
              else if namein='SPH' then begin k:='s'; c:=1; end
                  else if namein='SPL' then begin k:='s'; c:=2; end;
        'P' : if namein='PC' then  k:='p';
        'M' : if namein='MEM'  then  k:='m';
        'I' : if namein= 'IMMBYTE'  then  k:='a'
              else if namein= 'IMMWORD' then  k:='b';
        end;       end
    Else begin
         Str2v(namein,wout,e);
         if e=0 then begin k:= 'c'; c:= wout; end; end;
end;
PROCEDURE GetValue (t:char; n:word; var vout);
begin
     case t of
     'r': Byte(vout):= R8T[n]^.v;
     'w': Word(vout):= R16T[n]^.getw;
     'h': Byte(vout):= R16T[n]^.hb.v;
     'l': Byte(vout):= R16T[n]^.lb.v;
     'f': Boolean(vout):= F1T[n]^.getfv;
     'm': Byte(vout):= Mem[Hi(EA)]^[Lo(EA)];
     'c': Word(vout):= n;
     's': case n of
          0: Word(vout):= SP;
          1: Byte(vout):= Hi(SP);
          2: Byte(vout):= Lo(SP); end;
     'p': Word(vout):= PC;
     'a': Byte(vout):= IMMDAT MOD 256;
     'b': Word(vout) := IMMDAT;
     end;
end;
PROCEDURE UpdateFReg;
Var a:byte;
begin      a:=0;
     if(F1T[0]^.getfv) then a:=a or $01;
     if(F1T[1]^.getfv) then a:=a or $02;
     if(F1T[2]^.getfv) then a:=a or $04;
     if(F1T[3]^.getfv) then a:=a or $08;
```

```
     if(F1T[4]^.getfv) then a:=a or $10;
     if(F1T[5]^.getfv) then a:=a or $20;
     if(F1T[6]^.getfv) then a:=a or $40;
     if(F1T[7]^.getfv) then a:=a or $80;
     case ftp of
     'r': R8T[fid]^.putv(a);
     'h': R16T[fid]^.hb.putv(a);
     'l': R16T[fid]^.lb.putv(a);  end;
end;
PROCEDURE Updateflag;
Var a,i:byte;  b:array[0..7] of boolean;
begin
     case ftp of 'r': a:= R8T[fid]^.getv;
                 'h': a:= R16T[fid]^.hb.getv;
                 'l': a:= R16T[fid]^.lb.getv; end;
     b[0] := ((a and $01)>0);     b[1] := ((a and $02)>0);
     b[2] := ((a and $04)>0);     b[3] := ((a and $08)>0);
     b[4] := ((a and $10)>0);     b[5] := ((a and $20)>0);
     b[6] := ((a and $40)>0);     b[7] := ((a and $80)>0);

     for i:=1 to 7 do F1T[i]^.putfv(b[i]);
end;□
```

```
{ ---------------Generic CPU---------------------}
TYPE  MemPage = array[0..$FF] of byte;

Var   Mem: array[0..$FF] of ^MemPage;
      IOspace: MemPage;
      carry,sign,zero,overflow,halfcarry,parity,decimal: boolean;
      PC,SP,EA,IMMDAT: word;   IOAd: byte;

PROCEDURE ProcessGFlag(gnx:str10; p:char; var bx:boolean);
begin
     case gnx[1] of
     'C':     case p of
              's': carry:= true;
              'r': carry:= false;
              'n': carry:= Not(carry);
              'i': carry:= bx;
              'o': bx:= carry;         end;
     'S':     case p of
              's': sign:= true;
              'r': sign:= false;
              'n': sign:= Not(sign);
              'i': sign:= bx;
              'o': bx:= sign;          end;
     'Z':     case p of
              's': zero:= true;
              'r': zero:= false;
              'n': zero:= Not(zero);
              'i': zero:= bx;
              'o': bx:= zero;          end;
     'O':     case p of
              's': overflow:= true;
              'r': overflow:= false;
              'n': overflow:= Not(overflow);
              'i': overflow:= bx;
              'o': bx:= overflow;      end;
     'H':     case p of
              's': halfcarry:= true;
              'r': halfcarry:= false;
```

```
                'n': halfcarry:= Not(halfcarry);
                'i': halfcarry:= bx;
                'o': bx:= halfcarry;    end;
       'P':     case p of
                's': parity:= true;
                'r': parity:= false;
                'n': parity:= Not(parity);
                'i': parity:= bx;
                'o': bx:= parity;       end;
       'D':     case p of
                's': decimal:= true;
                'r': decimal:= false;
                'n': decimal:= Not(decimal);
                'i': decimal:= bx;
                'o': bx:= decimal;      end;
     end;
end;
PROCEDURE PushB(value:byte);
begin
     SP:= SP-1;
     Mem[Hi(SP)]^[Lo(SP)] := value;
end;
procedure Push6502(value:byte);
begin
     Mem[hi(sp)]^[lo(sp)] := value;   sp:=sp-1;
end;
PROCEDURE PullB(var value:byte);
begin
     value:= Mem[Hi(SP)]^[Lo(SP)];
     SP:= SP+1;
end;
procedure Pull6502(var value:byte);
begin
     sp:= sp+1;   value:= Mem[Hi(sp)]^[Lo(sp)];
end;
PROCEDURE PushW(vl:word);
begin
     PushB(Hi(vl));  PushB(Lo(vl));
```

```
end;
procedure Push6502W(vl:word);
begin
     Push6502(Hi(vl)); Push6502(Lo(vl));
end;
PROCEDURE PullW(var value:word);
var hb,lb:byte;
begin
     PullB(lb);  PullB(hb);
     value:= hb SHL 8 + lb;
end;
procedure Pull6502W(var value:word);
var hb,lb:byte;
begin
     Pull6502(lb); Pull6502(hb); value:= hb shl 8+lb;
end;
procedure LoadB(var a:byte);
begin
     a:= Mem[Hi(EA)]^[Lo(EA)];
end;
procedure LoadW(var b:word);
var h,l:byte;
begin
     LoadB(l);  EA:=EA+1;  LoadB(h);
     b:= h shl 8+l;
end;
procedure StoreB(a:byte);
begin
     Mem[Hi(EA)]^[Lo(EA)] := a;
end;
procedure StoreW(b:word);
var i:byte;
begin
     i:= Lo(b);  StoreB(i);  EA:=EA+1;
     i:= Hi(b);  StoreB(i);
end;
procedure LoadBptr(var vl:byte; p:word);
begin
```

```
     v1:= mem[hi(p)]^[lo(p)];
end;
procedure StoreBptr(v1:byte; p:word);
begin
     mem[hi(p)]^[lo(p)] := v1;
end;
procedure LoadWptr(var v1:word; pt:word);
var h,l:byte;
begin
     LoadBptr(l,pt);  pt:=pt+1;  LoadBptr(h,pt);
     v1:= h shl 8+l;
end;
procedure StoreWptr(v1,pt:word);
var h,l:byte;
begin
     l:=Lo(v1);  h:=Hi(v1);
     StoreBptr(l,pt); pt:=pt+1; StoreBptr(h,pt);
end;

PROCEDURE TestSignZero(i:byte);
begin
     Sign := ( i > 127 );  Zero := ( i = 0 );
end;
procedure Testsign(i:byte);
begin  Sign:= (i>127); end;
procedure Testzero(i:word);
begin  Zero:= (i=0);   end;
procedure TestSignW(i:word);
begin  Sign:= (i>32767); end;
PROCEDURE TestOverflow(v1,v2:byte);
begin
     v1 := v1 XOR v2;  Overflow := ((v1 AND $80) > 0);
end;
procedure TestOverflowW(v1,v2:word);
begin  v1:= v1 Xor v2;  Overflow := ((v1 and $8000) > 0); end;
PROCEDURE TestHalfcarry(v1,v2:byte);
begin
     v1 := v1 AND $0F;  v2 := v2 AND $0F;  v1 := v1+v2;
```

```
     if v1 > 15  then  Halfcarry:= true
     else  Halfcarry:= false;
end;
PROCEDURE TestParity(i:byte);
Var  b0,b1,b2,b3,b4,b5,b6,b7 : boolean;
begin
     b0 := ((i AND $01) > 0);
     b1 := ((i AND $02) > 0);
     b2 := ((i AND $04) > 0);
     b3 := ((i AND $08) > 0);
     b4 := ((i AND $10) > 0);
     b5 := ((i AND $20) > 0);
     b6 := ((i AND $40) > 0);
     b7 := ((i AND $80) > 0);
b0 := Not( b0 XOR b1 );        b2 := Not( b2 XOR b3 );
b4 := Not( b4 XOR b5 );        b6 := Not( b6 XOR b7 );
     b0 := Not( b0 XOR b2 );        b4 := Not( b4 XOR b6 );
     Parity := Not( b0 XOR b4);
end;
PROCEDURE Addreg(var vd:byte; i:byte; wcy:boolean);
Var R:word; T:byte;
begin
     If wcy Then  R:= vd+i+ord(carry)
     Else  R:= vd+i;
     If Decimal  Then  begin
          T:= (vd mod 16)+(i mod 16);
          if wcy then T:= T+ord(carry);
          if T>9 Then  begin
             R:= R+6;  Halfcarry:= true;  end
          else  Halfcarry:= false;
          T := R div 16;
          if T>9  then  R:= R+6*16;
                           end
     Else begin
          T:= R mod 256;  TestOverflow(vd,T); TestHalfcarry(vd,i);
          end;
     carry := ((R div 256) > 0);  vd:= R mod 256;
     TestSignZero(vd);
```

```
end;
PROCEDURE Subreg(var vd:byte; i:byte; wcy:boolean);
Var R:word; v1,i1:byte;
begin
     If Decimal Then begin
         v1:= vd mod 16; i1:= i mod 16;
         if wcy then    i1:= i1+ord(carry);
         if v1 < i1  then  begin
              v1:= v1+10;  Halfcarry:= true;  end
         else  Halfcarry:= false;
         v1:= v1-i1;
         vd:= vd div 16;  i:= (i div 16)+ord(Halfcarry);
         if vd<i  then begin
              vd:= vd+10;   carry:= true;  end
         else  carry:= false;
         vd:= vd-i;
         vd:= (vd*16+v1) mod 256;  end
     Else begin
          R:=vd;
          if wcy  then i:= i+ord(carry);
          if R<i  then  begin
               carry:=true;  R:= R+256;  end
          else   carry:=false;
          v1:=vd;   vd:=R-i;
          if i<>0 then i:=256-i;    { make i twocomplement }
          TestHalfcarry(v1,i);  TestOverflow(v1,vd);
          end;
     TestSignZero(vd);
end;
PROCEDURE CompareReg(a,b:byte);
Var R:word;
begin
     R:= a;
     if R<b  then begin R:=R+256; carry:=true;  end
     else  carry:= false;
     R:= R-b;  b:=256-b;
     TestHalfcarry(a,b);  TestOverflow(a,R); TestSignZero(R);
end;
```

```
PROCEDURE ComplementReg(var i:byte);
begin
     i:= Not i;
end;
PROCEDURE TwoCompReg(var i:byte);
Var R:word;  T:byte;
begin
     T:= i;
     if i <> 0 then i:= 256-i;
     TestOverflow(T,i);  TestSignZero(i);
end;
PROCEDURE DecAdjust(var a:byte);
Var a0:word; t:byte;
begin
     a0:= a;  t:= a mod 16;
     if t>9  then begin a0:=a0+6; Halfcarry:=true;  end
     else  Halfcarry:= false;
     t:= a0 div 16;
     if t>9  then a0:= a0+6*16;
     carry := ((a0 div 256) > 0);
     a:= a0 mod 256;
     TestSignZero(a);  TestParity(a);
end;
PROCEDURE ANDLogic(var v1:byte; v2:byte);
begin
     v1:= v1 AND v2;
     TestSignZero(v1);  TestParity(v1);
end;
PROCEDURE ORLogic(var v1:byte; v2:byte);
begin
     v1:= v1 OR v2;
     TestSignZero(v1);  TestParity(v1);
end;
PROCEDURE XORLogic(var v1:byte; v2:byte);
begin
     v1:= v1 XOR v2;
     TestSignZero(v1);  TestParity(v1);
end;
```

```
PROCEDURE ShiftLeft(var a:byte);
begin
     carry:= (a > 127);
     a:= a SHL 1;    TestSignZero(a);  TestParity(a);
end;
PROCEDURE AshiftRight(var a:byte);
Var t:byte;
begin
     carry:= Odd(a);
     t:= a AND $80;   a:= a SHR 1;  a:= a OR t;
     TestSignZero(a);   TestParity(a);
end;
PROCEDURE LshiftRight(var a:byte);
begin
     carry:= Odd(a);  a:= a SHR 1;
     TestSignZero(a);   TestParity(a);
end;
PROCEDURE RotRight(var a:byte);
begin
     carry:= Odd(a);
     a:= a SHR 1;
     if carry  then a:= a+128;
     TestSignZero(a);   TestParity(a);
end;
PROCEDURE RotLeft(var a:byte);
begin
     carry:= (a > 127);  a:= a SHL 1;
     if carry  then  a:= a+1;
     TestSignZero(a);   TestParity(a);
end;
PROCEDURE ROR_Carry(var a:byte);
Var a1:boolean;
begin
     a1:= Odd(a);
     a:= (a SHR 1) + ord(carry)*12Û¥-□+@     □□□□□-□□□□□□□□□□□□□□□¦
Zero(a);   TestParity(a);
end;
PROCEDURE ROL_Carry(var a:byte);
Var a1:boolean;
```

```
begin
     a1:= (a > 127);
     a:= (a SHL 1) + ord(carry);  carry:= a1;
     TestSignZero(a);   TestParity(a);
end;

procedure AddRegW(var vd:word; i:word; wc:boolean);
Var vdl,vdh,il,ih:byte; old:word;
begin
     vdl:= Lo(vd);  il:= Lo(i);
     vdh:= Hi(vd);  ih:= Hi(i);
     old:= vd;
     Addreg(vdl,il,wc);
     Addreg(vdh,ih,true); vd:= vdh shl 8 + vdl;
     TestZero(vd);  TestSignW(vd);
     TestoverflowW(vd,old);
end;
procedure SubRegW(var vd:word; i:word; wc:boolean);
Var vdl,vdh,il,ih:byte; old:word;
begin
     vdl:= Lo(vd);  il:= Lo(i);
     vdh:= Hi(vd);  ih:= Hi(i);
     old:= vd;
     Subreg(vdl,il,wc);  Subreg(vdh,ih,true);
     vd:= vdh shl 8+vdl;
     TestZero(vd);    TestSignW(vd);  TestoverflowW(vd,old);
end;
procedure AddW(var va:word; vi:word);
begin
     AddRegW(va,vi,false);
end;
procedure AdcW(var va:word; vi:word);
begin
     AddRegW(va,vi,true);
end;
procedure SubW(var va:word; vi:word);
begin
     SubRegW(va,vi,false);
```

```
    SubRegW(va,vi,false);
end;
procedure SbcW(var va:word; vi:word);
begin
    SubRegW(va,vi,true);
end;
procedure IncW(var vi:word);
begin
    AddRegW(vi,1,false);
end;
procedure DecW(var vi:word);
begin
    SubRegW(vi,1,false);
end;
```

```
{ ----------------------Display--------------------}
TYPE WdType= (CmdWd,RstWd,RegWd,MemWd,IOWd);
     WdArray= array [WdType,0..6] of byte;
VAR CommandLine: string;  post,rmin: byte;
    MyMemoryCounter,CurrentStoreMemory: word;

CONST WindowOn = 1;
      WdData: record CurrWd: WdType; Wd: WdArray  end =
      ( CurrWd: CmdWd; Wd: ((1,22,80,25,1,1,1),
                             (1,1,49,20,1,1,1),
                             (56,1,80,15,1,1,1),
                             (53,16,80,22,1,1,1),
                             (50,1,54,15,1,1,1)));

PROCEDURE SetWindow(Wd:WdType);
begin
     WdData.Wd[WdData.CurrWd,4]:= wherex;
     WdData.Wd[WdData.CurrWd,5]:= wherey;
Window(WdData.Wd[Wd,0],WdData.Wd[Wd,1],WdData.Wd[Wd,2],WdData.Wd[Wd,3]);
     WdData.CurrWd:= Wd;
     Gotoxy(WdData.Wd[WdData.CurrWd,4],WdData.Wd[WdData.CurrWd,5]);
end;

PROCEDURE Errormode(i:integer);
begin
     SetWindow(RstWd);
     case i of
     -1: Writeln('Operand Error.');
     -2: Writeln('Symbol Error.');
     -3: Writeln('Reserved word Error.');
     -4: Writeln('Implied mnemonic Error.');
     -5: Writeln('Mode Error.');
     -6: Writeln('Assemble Error.');
     -7: Writeln('Cannot Find Operand Value.');
     -9: Writeln('No File Found.');
    -10: Writeln('Unknown Command.');
    -11: Writeln('Micro-Ins Error.');
    -12: Writeln('Var Not Match Ins.');
```

```
    -13: Writeln('Jump Not Found.');
    end;
end;

FUNCTION  NextWd:string;
Const Seperator: set of char = [',','(',')',';',':','#',' ',^I,^M];
Var A:string;
begin
     A:= '';
     if CommandLine[post] <> ^M then begin
        while CommandLine[post] in Seperator do post:= post+1;
        While Not(CommandLine[post] in Seperator) Do begin
            A:= A+Upcase(CommandLine[post]);
            post:= post+1;     end;   end;
     NextWd:= A;
end;
PROCEDURE ReadCommandLine(prompt:string);
begin
     SetWindow(CmdWd);
     Write(prompt); Readln(CommandLine); CommandLine:= CommandLine+^M;
     post:=1;
end;
Procedure SetCursor(StartLine,EndLine:byte);
Var R:Registers;
begin
     R.CH:= StartLine And $1F;
     R.CL:= EndLine And $1F;
     R.AH:= $01;
     Intr($10,R);
end;
PROCEDURE SetCursorOn;
begin
     SetCursor(12,13);
end;
PROCEDURE SetCursorOff;
begin
     SetCursor($15,$15);
end;
```

```
PROCEDURE ReadLine(x,y:integer; var s:string);
Var ch:char;  i:integer;  st:boolean;
begin
     gotoxy(x,y);
     TextAttr:= $70;  write(s);  gotoxy(x,y);
     i:=1;  st:= False;
     Repeat
        ch:= Upcase(Readkey);
        if ch = ^M then st:= True
        else if ch in ['0'..'9','A'..'F'] then begin
                 s[i]:= ch;
              Û¥-□+@   □□□□□-□□□□□□□□□□□□□□□
```

```
write(ch);
                if i > length(s) then st:= True;  end
            else if ch = ^H then begin
                    i:= i-1;
                    if i<1 then i:=1;
                    write(ch);   end;
    Until  st;
    gotoxy(x,y);  NormVideo;  Write(s);
end;
PROCEDURE DumpMemoryWindow;
Var  i,A1,A2,E:word;
begin
    if WdData.Wd[MemWd,6]<>WindowOn then Exit;
    A2:= (MyMemoryCounter Div 8)*8;
    if (CurrentStoreMemory < A2) or (CurrentStoreMemory > (A2+48)) then Exit;
    SetWindow(MemWd);
    For A1:=0 to 5 Do begin
        gotoxy(1,A1+1);
        E:= A2+A1*8;
        Write(ByteToHes(Hi(E))+ByteToHes(Lo(E))+':');
        for i:=0 to 7 do
          write(ByteToHes(Mem[Hi(E+i)]^[Lo(E+i)])+' ');
                    end;
end;
PROCEDURE ShowRegisterWindow;
var i:byte;
begin
    if WdData.Wd[RegWd,6] <> WindowOn then Exit;
    SetWindow(RegWd);
if haver8 then
    For i:=1 to R8num Do begin
        gotoxy(1,i); write(R8T[i]^.n:4,' ',ByteToHes(R8T[i]^.v)); end;
if haver16 then
    For i:=1 to R16num Do begin
        gotoxy(9,i);
Write(R16T[i]^.wn:4,' ',ByteToHes(R16T[i]^.hb.v)+ByteToHes(R16T[i]^.lb.v));
                        end;
    gotoxy(19,1); write('PC ',ByteToHes(Hi(PC))+ByteToHes(Lo(PC)));
```

```
    gotoxy(19,2); write('SP ',ByteToHes(Hi(SP))+ByteToHes(Lo(SP)));
    For i:=0 to 10 Do
        if F1T[i]^.fn <> '' then  begin
           gotoxy(22,i+5);
           write(F1T[i]^.fn:2,' ',ORD(F1T[i]^.getfv)); end;
end;
Procedure Setrmin;
BEGIN
If haveR8 and haveR16 Then begin
if R8num < R16num then rmin:= R8num
else rmin:= R16num;         end
Else  if Not haveR8  then  rmin:= R16num
      else  rmin:= R8num;
rmin:= rmin-2;
if rmin > 13 then rmin:= 13;
END;□
```

```
{ ---------------Miscellaneous--------------- }
TYPE  str2= string[2]; str4= string[4]; str5= string[5]; str6= string[6];
str8= string[8]; str10= string[10]; str12= string[12]; str40= string[40];


FUNCTION CheckSym(s:string):boolean;
begin
     if s[1] in ['$','0'..'9'] then  CheckSym:= false
     else  CheckSym:= true;
end;
FUNCTION HesToByte(hex:string):byte;
const HexTable : array['0'..'F'] of byte
               = (0,1,2,3,4,5,6,7,8,9,0,0,0,0,0,0,0,10,11,12,13,14,15);
begin
     HesToByte := HexTable[hex[1]]*16+HexTable[hex[2]];
end;
PROCEDURE HesToWord(x:string; var R:word; var e:integer);
const hex4 : string[4] = '0000';
var   y:string;  i:word;
begin
     if x='' then begin e:= -1; R:= 0; Exit;  end;
     e:=0;  i:=0;
     if length(x) <= 4 then
          y:= copy(hex4,1,(4-length(x))) + x
     else  e:=5;
     While (i<4) and (e=0) Do  begin
          i:=i+1;
          y[i]:= upcase(y[i]);
          if Not(y[i] in ['0'..'9','A'..'F']) then e:=i;  end;
     if e=0 then
          R:= HesToByte(copy(y,1,2))*256 + HesToByte(copy(y,3,2))
     else R:=0;
end;
PROCEDURE Str2v(s:string; var vl:word; var er:integer);
var l:byte;
begin
     l:= length(s);
     if Not ((s[1] = '$') or (s[1] = 'H')) then
         Val(s,vl,er)
```

```
     else begin
          if s[1] in ['$','0'] then begin
              delete(s,1,1); l:=l-1; end;
          if s[1] = 'H' then delete(s,1,1);
          HesToWord(s,vl,er);
          end;
end;

PROCEDURE UpperCase(var s:string);
VAR I:INTEGER;
BEGIN
for i:= 1 to length(s) do s[i] := upcase(s[i]);
END;
FUNCTION  ByteToHes(i:byte):string;
CONST TABLE : ARRAY [0..$0F] OF CHAR = '0123456789ABCDEF';
VAR   H : STRING[2];
BEGIN
      H := TABLE[i SHR 4];
      H := H + TABLE[i AND $0F];
      ByteToHes := H;
END;□
```

```
{ ----------SIMULATOR EDITOR------------- }
PROGRAM SIMEDIT;
{$I ROOTDISP.PAS }
{$I EDr8.PAS }
{$I EDr16.PAS }
{$I EDf1.PAS }
{$I EDasm.PAS }
{$I EDmd.PAS }
{$I EDins.PAS }

VAR ch:char;
BEGIN
Repeat
   clrscr;  gotoxy(12,8);
   write('The simulator is to be configured in the following parts.');
   gotoxy(24,10); write('1. 8-BIT REGISTER');
   gotoxy(24,11); write('2. 16-BIT (8-BIT PAIR) REGISTER');
   gotoxy(24,12); write('3. FLAGS');
   gotoxy(24,13); write('4. ASSEMBLER');
   gotoxy(24,14); write('5. ASSEMBLY ADDRESSING RECOGNITION');
   gotoxy(24,15); write('6. INSTRUCTION SET');
{   gotoxy(24,16); write('7. Prepare Instruction File for Simulator');}
   gotoxy(12,18);
   write('Select the number (enter [q] to quit). ');Readln(ch);
   case ch of
   '1': nameR8;
   '2': USER_R16;
   '3': USER_F1;
   '4': USER_ASM;
   '5': USER_MODE;
   '6': USER_INS;
{   '7': }
   end;
Until (ch = 'q') or (ch = 'Q');
END.□
```

```
CONST  rfn : string = 'a:r8.dat';
TYPE arr15ofstr = array[1..15] of string;
VAR  ra : ^arr15ofstr;

PROCEDURE WriteEscr1;
var r:byte;
begin Highvideo;
     gotoxy(33,5); write('8-BIT REGISTERS');
     For r:=1 to 15 Do begin
         gotoxy(33,r+5);   write(r:2,'.'); end;
end;
PROCEDURE ShowEscrDat1;
var  i:byte;
begin   For i:=1 to 15 Do
          IF ra^[i] <> '' THEN  BEGIN
LowVideo;
          gotoxy(37,i+5);  write(ra^[i]); END;
        TextAttr:= black*16+white;
end;

PROCEDURE nameR8;
VAR  rf : File of str4;  rrec :str4;
     r:byte;   done:boolean;  ch:char;
BEGIN
New(ra); for r:=1 to 15 do ra^[r]:='';
If Not Exist(rfn)  Then   begin
      clrscr;  WriteEscr1;  end
Else  Begin    clrscr;
      writeln('Reading...');
      assign(rf,rfn); reset(rf); r:=1;
      While Not Eof(rf)  Do begin
          Read(rf,rrec);  ra^[r] := rrec;
          r:= r+1;          end;
      close(rf); write('Press ENTER to continue.'); Readln;
      clrscr; WriteEscr1; ShowEscrDat1;  End;
r:=1;   gotoxy(37,r+5);    done:=false;
Repeat
      ReadMapKey(ch);
      case ch of
      ^U: begin if r>1 then r:=r-1;  gotoxy(37,r+5); end;
      ^D: begin if r<15 then r:=r+1;  gotoxy(37,r+5); end;
      ^[: done:=true;
      ^S: begin assign(rf,rfn); Rewrite(rf); r:=1;
                While ra^[r] <> '' Do begin  rrec:=ra^[r];
                    write(rf,rrec);  r:=r+1;  end; close(rf);
                done:=true;
          end;
      ^M: begin
          MakeRevWin(37,r+5,40,r+5); clrscr;
          EditLine(ra^[r],4);  ClrEol; window(1,1,80,25);
          if r<15 then r:=r+1;
          gotoxy(37,r+5);
          end;
      end;
Until done;  Dispose(ra);  Normvideo;
END;□
```

```
CONST   wfn :string = 'a:R16.DAT';
VAR   wa,ha,la : ^arr15ofstr;

PROCEDURE WriteEscr2;
var r: byte;
begin  Highvideo;
     gotoxy(33,4); write('16-BIT REGISTERS');
     gotoxy(33,6); write('Word');
     gotoxy(39,6); write('Hbyte'); gotoxy(45,6); write('Lbyte');
     for r:=1 to 15 do  BEGIN
         gotoxy(30,r+6); write(r:2,'.'); END;
end;
PROCEDURE ShowEscrDat2;
var  r:byte;
begin      LowVideo;
     for r:=1 to 15 do    begin
          gotoxy(33,r+6); write(wa^[r]);
          gotoxy(39,r+6); write(ha^[r]);
          gotoxy(45,r+6); write(la^[r]);  end;
end;

PROCEDURE USER_R16;
VAR   wf:File of WrecT;  wrec:WrecT;
      r,c:byte;  done:boolean;  ch:char;
BEGIN
New(wa);  New(ha);  New(la);
for r:= 1 to 15 Do begin wa^[r]:=''; ha^[r]:=''; la^[r]:=''; end;
If Not Exist(wfn) Then begin clrscr; WriteEscr2; end
Else begin  clrscr; writeln('Reading...');
     assign(wf,wfn); reset(wf);  r:=1;
     While Not Eof(wf) Do begin
         Read(wf,wrec);
         wa^[r]:=wrec.w; ha^[r]:=wrec.h; la^[r]:=wrec.l;
         r:=r+1;  end; close(wf);
     Write('Press ENTER to continue.');  readln;
     clrscr; WriteEscr2;  ShowEscrDat2;  end;
r:=1;  c:=0;  done:=false;  gotoxy(33,r+6);
Repeat
   ReadMapKey(ch);
   case ch of
   ^M: begin
       MakeRevWin(33+6*c,r+6,36+6*c,r+6);  clrscr;
       case c of
       0 : EditLine(wa^[r],4);
       1 : EditLine(ha^[r],4);
       2 : EditLine(la^[r],4); end;
       ClrEol; window(1,1,80,25);
       if c<2 then c:=c+1 else begin c:=0; if r<15 then r:=r+1; end;
       gotoxy(33+6*c,r+6);   end;
   ^L: begin if c>0 then c:=c-1; gotoxy(33+6*c,r+6); end;
   ^R: begin if c<2 then c:=c+1; gotoxy(33+6*c,r+6); end;
   ^U: begin if r>1 then r:=r-1; gotoxy(33+6*c,r+6); end;
   ^D: begin if r<15 then r:=r+1; gotoxy(33+6*c,r+6); end;
   ^[: done := true;
   ^S: begin assign(wf,wfn); rewrite(wf);  r:=1;
             while wa^[r] <> '' Do begin
             wrec.w:=wa^[r]; wrec.h:=ha^[r]; wrec.l:=la^[r];
             write(wf,wrec); r:=r+1; end; close(wf); done:=true;
       end;
   end;
Until done;  Dispose(wa); dispose(ha); dispose(la);  Normvideo;
END;□
```

```
CONST fsx: array[0..2] of byte =(21,23,40);
      fex: array[0..2] of byte =(24,24,49);
      ffn: string = 'a:flg.dat';
TYPE  arr010ofstr =array[0..10] of string;
VAR   sna,gna: ^arr010ofstr;  s0: ^string;

PROCEDURE WriteEscr3;
begin     Highvideo;
     GOTOXY(38,1); writeln('FLAGS');
     writeln('FLAG REGISTER NAME');
     gotoxy(23,3); write('Flag name');
     gotoxy(40,3); writeln('Generic flag');
     writeln('Flag register: bit 0');
     writeln('               bit 1');
     writeln('               bit 2');
     writeln('               bit 3');
     writeln('               bit 4');
     writeln('               bit 5');
     writeln('               bit 6');
     writeln('               bit 7');
     writeln('Other flags:  1');
     writeln('              2');
     writeln('              3');
end;
PROCEDURE ShowEscrDat3;
var n:byte;
begin  Lowvideo;
     gotoxy(fsx[0],2); write(s0^);
     for n:=0 to 10 do begin gotoxy(fsx[1],n+4); write(sna^[n]);
         gotoxy(fsx[2],n+4); write(gna^[n]);  end;
end;

PROCEDURE USER_F1;
VAR   frec: frecT;   ff: File of frecT;
      i,n,y:byte;  ch:char;  done:boolean;
BEGIN
New(sna); New(gna); New(s0);  s0^:='';
for n:=0 to 10 Do begin sna^[n] := ''; gna^[n]:=''; end;
If Not Exist(ffn) Then begin clrscr; writeEscr3;  end
Else  begin
    clrscr; writeln('Reading...');
    assign(ff,ffn); reset(ff);  read(ff,frec);  close(ff);
    s0^ := frec.fregn;
    for n:=0 to 10 do begin sna^[n]:=frec.f[n].sn; gna^[n]:=frec.f[n].gn;end;
    write('Press ENTER to continue.'); readln;
    clrscr;  writeEscr3;   ShowEscrDat3;  end;
i:=0;  y:=2;  done:=false;  gotoxy(fsx[0],2);
Repeat
    ReadMapKey(ch);
    case ch of
    ^M:begin  MakeRevwin(fsx[i],y,fex[i],y); clrscr;
    case i of
    0: EditLine(s0^,4);
    1: EditLine(sna^[n],2);
    2: EditLine(gna^[n],10);  end;
    ClrEol;  Window(1,1,80,25);
    if i=0 then  begin i:=1; n:=0; y:=4; end
    else if i=1  then  i:=2
         else begin i:=1; if n<10 then n:=n+1; y:=n+4; end;
    gotoxy(fsx[i],y);     end;
    ^L:begin if i=2 then i:=1; gotoxy(fsx[i],y); end;
    ^R:begin if i=1 then i:=2; gotoxy(fsx[i],y); end;
    ^U:begin if y>4 then begin y:=y-1; n:=n-1; end
             else begin y:=2; i:=0; end;
             gotoxy(fsx[i],y); end;
    ^D:begin if y=2 then begin y:=4; i:=1; end
             else  if y<14 then begin y:=y+1; n:=n+1; end;
             gotoxy(fsx[i],y); end;
    ^[: done:=true;
     ^S:begin assign(ff,ffn); rewrite(ff);
             With frec Do Begin
               rt:='x';  rn:=0;  fregn:=s0^;
               for n:=0 to 10 do begin
                  f[n].sn:=sna^[n];  f[n].gn:=gna^[n]; end; End;
             write(ff,frec);  close(ff);  done:=true;  end;
     end;
 Until  done;  dispose(sna);  dispose(gna);  dispose(s0);  Normvideo;
 END;□
```

```
CONST asx:array[1..10] of byte =(22,22,4,4,4,4,17,17,17,17);
      aex:array[1..10] of byte =(22,22,78,78,78,78,20,20,20,20);
      ay :array[1..10] of byte =(2,3,5,7,8,9,19,20,21,22);
      afn: string = 'a:uasm.dat';
TYPE arr10ofstr = array [1..10] of string;
VAR  asa : ^arr10ofstr;

PROCEDURE WriteEscr4;
begin
     highvideo;  gotoxy(29,1); writeln('ASSEMBLER AND ADDRESSING');
     writeln('1) Comment separator');
     writeln('2) Operand separator');
     writeln('3) Mnemonics determine implied addressing mode (Type "," between
mnemonics.)');
     writeln;
     writeln('4) Reserved names in the operand fields (Type "," between names.)');
     gotoxy(1,10); writeln('5) Generic addressing modes:-   0 IMP');
writeln('   1 ABS                   5 IDXA                 9 IDX1');
writeln('   2 IMM                   6 IDXAIMM             10 IDX1IDR');
writeln('   3 IO                    7 IDXB                11 IDX2');
writeln('   4 REL                   8 IDXBIMM             12 IDRIDX2');
writeln('                                                 13 IDR');
writeln('   If modes 5-16 are supposed to use, enter the register name for each
of');
writeln('   the selected modes.');
gotoxy(17,18);writeln('register name');
writeln('     mode 5,6');
writeln('     mode 7,8');
writeln('     mode 9,10');
writeln('     mode 11,12');
end;
PROCEDURE ShowEscrDat4;
var i:byte;
begin
     Lowvideo;
     for i:=1 to 10 do begin gotoxy(asx[i],ay[i]); write(asa^[i]); end;
end;

PROCEDURE USER_ASM;
VAR arec:arecT;  af:File of arecT;
    i,c,l,n:byte;  ch:char;  done:boolean;  t : ^arr40ofstr;
BEGIN
new(asa); for i:=1 to 10 do asa^[i]:='';
If Not Exist(afn) Then begin clrscr; writeEscr4; end
Else begin clrscr; writeln('Reading...');
     assign(af,afn); reset(af); read(af,arec); close(af);
     With arec Do Begin
asa^[1]:=cms; asa^[2]:=ops;
if mimp[1]<>'' then begin
   c:=1;
   while (mimp[c]<>'') and (c<=10) do begin
       asa^[3]:=asa^[3]+mimp[c]+','; c:=c+1; end; l:=length(asa^[3]);
   if asa^[3][l] = ',' then delete(asa^[3],l,1);  end;
if rs[1]<>'' then begin  c:=1;
   while (rs[c]<>'') and (c<=40) do begin
       asa^[4]:=asa^[4]+rs[c]+','; c:=c+1; end; l:=length(asa^[4]);
DELETE(ASA^[4],L,1);  L:=L-1;
IF L>75 THEN  begin ASA^[5]:=COPY(ASA^[4],76,L-75); delete(asa^[4],76,l-75);end;
L:=LENGTH(ASA^[5]);
IF L>75 THEN  begin aSA^[6]:=COPY(ASA^[5],76,L-75); delete(asa^[5],76,l-75);end;
                 end;
for i:=1 to 4 do asa^[i+6] := idxreg[i];  End;
    write('Press ENTER to continue.'); readln;
    clrscr;  writeEscr4;   ShowEscrDat4;  end;
i:=1;  done:=false;  gotoxy(asx[i],ay[i]);
Repeat
     ReadMapKey(ch);
     case ch of
     ^M:begin  MakeRevWin(asx[i],ay[i],aex[i],ay[i]); clrscr;
               case i of
               1..2 : n:=1;
               3..6 : n:=75;
               7..10: n:=4;  end;  EditLine(asa^[i],n);
               ClrEol; window(1,1,80,25);
          if i<10 then i:=i+1; gotoxy(asx[i],ay[i]);      end;
    ^U:begin if i>1 then i:=i-1; gotoxy(asx[i],ay[i]); end;
    ^L:begin if i>1 then i:=i-1; gotoxy(asx[i],ay[i]); end;
    ^D:begin if i<10 then i:=i+1; gotoxy(asx[i],ay[i]); end;
    ^R:begin if i<10 then i:=i+1; gotoxy(asx[i],ay[i]); end;
```

```
   ^[: done:=true;
   ^S:begin assign(af,afn); rewrite(af);
With arec Do Begin
if asa^[1] <> '' then cms:=asa^[1][1];
if asa^[2] <> '' then ops:=asa^[2][1];
for i:=1 to 10 do mimp[i]:='';
if asa^[3]<>'' then begin    new(t);
   Token(asa^[3],cmasep,10,t^); for i:=1 to 10 do mimp[i]:=t^[i];
   dispose(t);       end;
for i:=1 to 40 do rs[i]:='';
i:=1;
if asa^[4]<>'' then begin    new(t);
    Token(asa^[4],cmasep,40,t^);
    while (t^[i]<>'') and (i<40) do begin  rs[i]:=t^[i];  i:=i+1; end;
    dispose(t);      end;
if asa^[5]<>'' then begin  new(t);
    Token(asa^[5],cmasep,40,t^);
    n:=1;
    while (t^[n]<>'') and (i<40) do begin  rs[i]:=t^[n]; i:=i+1; n:=n+1; end;
    dispose(t);      end;
if asa^[6]<>'' then begin  new(t);
    Token(asa^[6],cmasep,40,t^);
    n:=1;
    while (t^[n]<>'') and (i<40) do begin  rs[i]:=t^[n]; i:=i+1; n:=n+1; end;
    dispose(t);      end;
for i:=1 to 4 do idxreg[i]:=asa^[i+6];
             End;
            write(af,arec); close(af);
            done:=true; end;
   end;
Until done;  dispose(asa);  Normvideo;
END;
```

□

```
CONST msx:array[1..7] of byte= (11,14,27,40,53,66,3);
      mex:array[1..7] of byte= (18,21,34,47,60,73,79);
      my :array[1..7] of byte= (4,5,5,5,5,5,7);

TYPE  E22recT = record p: string;
                        m: array[2..6] of string;
                        oc: string;  end;
      arr20ofE22recT = array[1..20] of E22recT;
VAR   E22p :^arr20ofE22recT;


PROCEDURE Writeitem(p:byte);
VAR i,n,y:byte;
begin   Highvideo;
     for i:=1 to 5 Do begin n:=i+5*(p-1); y:=4+4*(i-1);
        gotoxy(1,y); write(n);  end;
end;
PROCEDURE DelItem(p:byte);
var i,n,y:byte;
begin
     for i:=1 to 5 Do begin n:=i+5*(p-1); y:=4+4*(i-1);
        window(1,y,2,y); clrscr; end;  window(1,1,80,25);
end;
PROCEDURE WriteEscr5;
var j,y1,y2,y3:byte;
begin  Highvideo;
     gotoxy(26,2); write('USER-DEFINED ADDRESSING MODES');
     For j:=1 to 5 Do begin
        y1:=4+4*(j-1); y2:=5+4*(j-1); y3:=6+4*(j-1);
        gotoxy(3,y1); write('Pattern');
        gotoxy(11,y2); write('Md');
        gotoxy(24,y2); write('Mo'); gotoxy(37,y2); write('Mn');
        gotoxy(50,y2); write('Mb'); gotoxy(63,y2); write('Mw');
        gotoxy(3,y3); write('Opcode');  end;
end;
PROCEDURE ShowEscrDat5(p:byte);
var   i,j,n,y1,y2,y3:byte;
begin
     LowVideo;
     For j:=1 to 5 Do begin
       n:=j+5*(p-1);
       y1:=my[1]+4*(j-1); y2:=my[2]+4*(j-1); y3:=my[7]+4*(j-1);
       gotoxy(msx[1],y1); write(E22p^[n].p);
       for i:=2 to 6 do begin
          gotoxy(msx[i],y2); write(E22p^[n].m[i]);  end;
       gotoxy(msx[7],y3); write(E22p^[n].oc);       end;
end;
PROCEDURE DelDat(p:byte);
var  i,j,n,y1,y2,y3:byte;
begin
     For j:=1 to 5 Do begin
       n:=j+5*(p-1);
       y1:=my[1]+4*(j-1); y2:=my[2]+4*(j-1); y3:=my[7]+4*(j-1);
       window(msx[1],y1,mex[1],y1); clrscr;
       for i:=2 to 6 do begin
           window(msx[i],y2,mex[i],y2); clrscr; end;
       window(msx[7],y3,mex[7],y3); clrscr;  end;  window(1,1,80,25);
end;

PROCEDURE USER_MODE;
CONST mfn: string = 'A:mode.dat';
VAR Mdrec :MdrecT;  mf: File of MdrecT;
    c,i,j,l,n,pg,y:byte; ch:char; done:boolean; T : ^arr40ofstr;
BEGIN
New(E22p); for n:=1 to 20 Do begin With E22p^[n] Do Begin
    p:=''; for i:=2 to 6 do m[i]:=''; oc:=''; End; end;
pg:=1;
If Not Exist(mfn) Then begin clrscr; WriteItem(pg); WriteEscr5;  end
Else begin
     clrscr; writeln('Reading...');
     assign(mf,mfn); reset(mf); n:=1;
     While Not Eof(mf) Do begin  Read(mf,Mdrec);
        with E22p^[n] Do Begin
        p:=Mdrec.pat;
        for c:=0 to 4 do m[c+2] := Mdrec.md[c];
        if Mdrec.oa[1] <> '' then begin
            c:=1;
            while Mdrec.oa[c] <> '' do begin
               oc:=oc+Mdrec.oa[c]+','; c:=c+1; end;
```

```
TYPE  InsRecordT    = array [1..20] of str40;
      InsPtrT       = ^InstructionT;
      InstructionT = record
                       prev. : InsPtrT;
                       dat   : InsRecordT;
                       next  : InsPtrT;
                     end;

CONST fn :STRING = 'a:UINS.DAT';
      sx :array[1..20] of byte = (13,37,57,15,41,28,28,28,28,28,28,28,28,28,28
                                         ,28,28,28,28,28);
      sy :array[1..20] of byte =
(2,2,2,3,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18);
      ex :array[1..20] of byte=(24,48,64,15,48,57,57,57,57,57,57,57,57,57,57
                                         ,57,57,57,57,57);

VAR   First,Last,Current,Old : InsPtrT;
      PgNo,LastPg :word;    Quit :boolean;   Choice :char;
      insfi :File of InsRecordT;
I,N:BYTE;    es:string;

PROCEDURE ShowMainMenu;
begin  WINDOW(1,24,80,25); TEXTattr:= lightgray*16+black; CLRSCR;
     gotoxy(5,1);
     writeln('FIRST-[Home]    LAST-[End]    PREV-[PgUp]    NEXT-[PgDn]    GO TO-
[^P]');
     gotoxy(7,2);
     write('EDIT-[Enter]          DELETE-[^Y]          SAVE-[F2]          EXIT-[Esc]');
     textattr:=black*16+lightgray;  WINDOW(1,1,80,25);
end;

PROCEDURE DisplayEditScreen;
begin clrscr;  Textattr:= black*16 + white;
     write('RECORD'); gotoxy(6,2); write('OPCODE');
     gotoxy(28,2);  write('MNEMONIC');
     gotoxy(52,2); write('MODE'); gotoxy(6,3); write('NO.BYTES');
     gotoxy(28,3); write('GENERIC MODE');
     gotoxy(6,4); write('GENERIC INSTRUCTIONS');
Showmainmenu;
end;

PROCEDURE WritePg;
begin
     gotoxy(8,1); write(PgNo);
end;
PROCEDURE DelPg;
begin window(8,1,11,1); clrscr;  end;

PROCEDURE NewDataPtr;
var i:byte;
Begin
     New(First);
     First^.Prev := Nil;  First^.Next := Nil;  Last := First;
     for i:=1 to 20 do First^.dat[i]:='';      LastPg := 1;
End;

PROCEDURE OutfromFile;
Begin
     New(First); First^.Prev:=Nil; First^.Next:=Nil; Last:=First; PgNo:=0;
     Assign(insfi,fn);    Reset(insfi);
     While  Not Eof(insfi)  Do  begin
        Read(insfi,Last^.Dat);
        Old := Last;
        New(Last);
        Old^.Next := Last;   Last^.Prev := Old;
        Last^.Next := Nil;   PgNo := PgNo+1;    end;
     Close(insfi);
     LastPg := PgNo;
End;

PROCEDURE IntoFile;
begin
     current:= First;
     assign(insfi,fn);  rewrite(insfi);
     While  current <> nil  Do
     begin write(insfi,current^.Dat);
           current := current^.Next;
     end;
     close(insfi);
```

```
end;

PROCEDURE DisposeMemory;
begin
     While  Last <> nil  Do
     begin  current := Last^.prev;
            Dispose(Last);
            Last := current;  end;
end;

PROCEDURE WriteCurrentPg;
var  i:byte;
Begin
     WritePg;  i:=1;
     While (current^.Dat[i] <> '') and (i<=20) Do
     begin gotoxy(sx[i],sy[i]);
           write(current^.Dat[i]);
           i:=i+1;
     end;
End;

PROCEDURE Delcurrentdat;
var  i:byte;
begin
     DelPg;
     For i:=1 to 20 Do begin
         window(sx[i],sy[i],ex[i],sy[i]); clrscr;  end;
     window(1,1,80,25);
end;

PROCEDURE FirstPage;
Begin
     If PgNo<>1 Then  begin
        DelcurrentDat;
        Current := First;  PgNo := 1; WriteCurrentPg; end;
End;
PROCEDURE LastPage;
Begin
     If PgNo<>LastPg Then begin
        DelcurrentDat;
        Current := Last;  PgNo := LastPg+1; WriteCurrentPg; end;
End;
PROCEDURE PageUp;
Begin
     If  Current <> First  Then  begin
          DelcurrentDat;
          Current := Current^.Prev;
          PgNo := PgNo-1;           WriteCurrentPg; end;
End;
PROCEDURE PageDown;
var i:byte;
Begin
     If  Current^.Next <> Nil  Then
          Current := Current^.Next
     Else  begin
         Old := Last;
         New(Last);   for i:=1 to 20 do Last^.Dat[i]:='';
         Old^.Next := Last;
         Last^.Prev := Old;   Last^.Next := Nil;
         Current := Last;    LastPg := PgNo+1;
         end;
     DelcurrentDat;
     PgNo := PgNo+1;     WriteCurrentPg;
End;

PROCEDURE SelectPage;
Var  n:word;
Begin
     Clrscr;     Write('Record...');  Readln(n); clrscr;
     DisplayEditScreen;
     While (n < PgNo) and (Current^.prev<>Nil) do begin
current:=current^.prev;  PgNo:=PgNo-1; end;
     While (n > PgNo) and (Current^.next<>Nil) do begin
current:=current^.next;  PgNo:=PgNo+1; end;
     WritecurrentPg;
End;

PROCEDURE ClearData;
var i:byte;
```

```
Begin     old := current;
     if current^.next <> nil then
     begin  current := current^.Next;
            current^.Prev := old^.Prev;
            old^.Prev^.Next := current;
            if old = first then first:=current; end
     else
     begin  current:=current^.prev;
            current^.next:=Nil;  Last:=current;
            PgNo:=PgNo-1;
     end;
Dispose(old);  LastPg := LastPg-1;
DelcurrentDat; WriteCurrentPg;
End;

PROCEDURE USER_INS;
BEGIN
     If  Exist(fn)  Then  OutfromFile
     Else  NewDataPtr;
     Current := First;    PgNo := 1;  Quit := False;
DisplayEditScreen;
     WriteCurrentPg;   Textattr:= black*16 + lightgray;
     I:=1;   GOTOXY(SX[I],SY[I]);
     Repeat
         ReadMapKey(choice);
         case  choice  of
         ^D : begin if i<20 then i:=i+1; gotoxy(sx[i],sy[i]); end;
         ^U : begin if i>1  then i:=i-1; gotoxy(sx[i],sy[i]); end;
         ^M : begin   es:=current^.dat[i];
CASE I OF 1     : N:=12;
          2     : N:=8;
          3     : N:=12;
          4     : N:=1;
          5     : N:=8;
          6..20: N:=40;  END;
                 MakeRevWin(SX[i],SY[I],EX[I],SY[I]); clrscr;
                 EditLine(es,N); ClrEol; current^.dat[i]:=es;
                 Window(1,1,80,25);
                 if i<20 then i:=i+1;
                 gotoxy(sx[i],sy[i]); end;
         ^A : BEGIN FirstPage; I:=1; GOTOXY(SX[I],SY[I]); END;
         ^Z : BEGIN LastPage;  I:=1; GOTOXY(SX[I],SY[I]); END;
         ^B : BEGIN PageUp;    I:=1; GOTOXY(SX[I],SY[I]); END;
         ^F : BEGIN PageDown;  I:=1; GOTOXY(SX[I],SY[I]); END;
         ^P : BEGIN SelectPage; I:=1; GOTOXY(SX[I],SY[I]); END;
         ^Y : BEGIN ClearData; I:=1; GOTOXY(SX[I],SY[I]); END;
         ^S : begin  InToFile;  quit:=true;  end;
         ^[ : Quit := True;
         end;
     Until   Quit;   DisposeMemory;  textattr:=black*16+lightgray;
END;□
```

```
uses crt,dos;
TYPE  FilenameT  = string[12];   CharSet = Set of char;
      str2  = string[2];         arr40ofstr =array[1..40] of string;
      str4  = string[4];
      str5  = string[5];
      str6  = string[6];
      str8  = string[8];
      str10 = string[10];
      str40 = string[40];
      WrecT = record w,h,l:str4; end;
      MdrecT= record pat:str8; md:array[0..4] of str8;
                     oa :array[1..12] of str5;    end;
         fT = record sn:str2; gn:str10; end;
         arr010offT = array[0..10] of fT;
      frecT = record rt:char; rn:byte; fregn:str4;
                     f:arr010offT;             end;
         arr10ofstr6 = array[1..10] of str6;
         arr40ofstr4 = array[1..40] of str4;
         arr4ofstr4  = array[1..4] of str4;
      arecT = record cms,ops:char;  mimp:arr10ofstr6;
                     rs:arr40ofstr4;
                     idxreg:arr4ofstr4;       end;

CONST cmasep : set of char = [' ',',',^I,^M];
VAR   cursormode : integer Absolute $0040:$0060;
      vport      : integer Absolute $0040:$0063;
      regs       : registers;

PROCEDURE SetCursor(top,bottom:byte);
begin
     regs.Ah := 1;
     regs.Ch := top;
     regs.Cl := bottom;
     intr($10,regs);
end;
PROCEDURE CursorOn;
begin
     port[vport]   := 10;
     port[vport+1] := Hi(cursormode) and $DF;
     port[vport]   := 11;
     port[vport+1] := Lo(cursormode);
end;
PROCEDURE CursorOff;
begin
     port[vport]   := 10;
     port[vport+1] := Hi(cursormode) or $20;
     port[vport]   := 11;
     port[vport+1] := Lo(cursormode);
end;

PROCEDURE MakeRevWin(x1,y1,x2,y2:byte);
begin
     Window(x1,y1,x2+1,y2);  TextAttr:=$70;
end;

PROCEDURE ReturnNormWin(s:string);
begin
     Textattr:= $01;  gotoxy(1,1);  write(s);
end;

FUNCTION Exist(finame:FileNameT):boolean;
var fi:File;
begin
     assign(fi,finame);
     {$I-}  reset(fi);  {$I+}
     if ioresult = 0  then  begin Exist:=true; close(fi);  end
     else  Exist:=false;
end;

{PROCEDURE OpenFile(fn:FilenameT; var f:File);
begin   assign(f,fn);  reset(f);    end;

PROCEDURE RewriteFile(fn:FilenameT; var f:File);
begin   assign(f,fn);  rewrite(f);   end;}

PROCEDURE ReadMapKey (var k:char);
Begin
     k := Readkey;
     if  k = #0  then  begin
```

```
          k := Readkey;
          case  k  of
{HOME}    #71  :  k := ^A;
{END}     #79  :  k := ^Z;
{LEFT}    #75  :  k := ^L;
{RIGHT}   #77  :  k := ^R;
{UP}      #72  :  k := ^U;
{DOWN}    #80  :  k := ^D;
{DEL}     #83  :  k := ^X;
{PgUp}    #73  :  k := ^B;
{PgDn}    #81  :  k := ^F;
{F2}      #60  :  k := ^S;
{F6}      #64  :  k := ^E;
{F10}     #68  :  k := ^[;
          else    k := #00;
          end;        end;
End;
PROCEDURE ProcessChar (k:char;maxl:byte;var s:string;var p:byte);
Var  l,x :byte;
Begin
     l := length(s);  k:=upcase(k);
     If (p > l) and (p<=maxl) Then  begin
              s := s+k;
              p := p+1;
              write(k);
                    end
     Else
         if (p<=l) and (l < maxl) then  begin
              x := wherex;
              Insert(k,s,p);  p := p+1;
              Gotoxy(1,wherey);
              write(s);
              Gotoxy(x+1,wherey);  end;

End;

PROCEDURE Right (s:string; maxl:byte; var p:byte);
Var   l:byte;
Begin
     l := length(s);
     if (p <= l) and (l <> 0) and (l<maxl) then  begin
          p := p+1;
          Gotoxy(wherex+1,wherey);  end;
End;

PROCEDURE Left (var p:byte);
Begin
     if p > 1 then  begin
          p := p-1;
          Gotoxy(wherex-1,wherey);  end;
End;

PROCEDURE BackSpace (var s:string; var p:byte);
Var  Tail :string;  l,x,y :byte;
Begin
     l := length(s);
     If p > 1  Then  begin
         y := wherey;
         Tail := Copy(s,p,l-p+1);
         Delete(s,p-1,1);   p := p-1;
         Write(^H);   x := wherex;
         ClrEOL;
         Gotoxy(x,y);  Write(Tail);  Gotoxy(x,y);

                      end;
End;

PROCEDURE DelChar (var s:string; var p:byte);
Var   Tail :string;    l,x,y :byte;
Begin
     l := length(s);
     If l <> 0 Then Begin   x := Wherex;    y := wherey;
     if (p < l)   then  begin
          Tail := Copy(s,p+1,l-p);
          Delete(s,p,1);
          clrEol;
          gotoxy(x,y);  Write(Tail);   end
     else begin delete(s,p,1); gotoxy(x,y); ClrEol; end;
     Gotoxy(x,y);   End;
```

```
End;
PROCEDURE EDITLine(var st:string; ll:byte);
Var    k :char;    pos:byte;
Begin
     gotoxy(1,1);   write(st);  gotoxy(1,1);   pos := 1;
     Repeat
        ReadMapKey(k);
        If  k < #32  Then   begin
           case  k  of
           ^H : BackSpace(st,pos);
           ^X : DelChar(st,pos);
           ^L : Left(pos);
           ^R : Right(st,ll,pos);
           end;                end
        Else    ProcessChar(k,ll,st,pos);
     Until  k = #13;    ReturnNormWin(st);
     Normvideo;
End;
PROCEDURE Token(s:string;sep:CharSet;n:byte;var tk:arr40ofstr);
Var i,p:byte;
Begin
     if s<>'' then begin  s:=s+^M;
         for i:=1 to n do tk[i] := ''; i:=1; p:=1;
         While s[p] <> ^M Do begin
           while (s[p] in sep) do p:=p+1;
           while Not(s[p] in sep) do begin
              tk[i] := tk[i]+s[p];  p:=p+1; end;
           i:=i+1;             end;   end;
End;
```
□

ภาคผนวก ข

ข้อมูลการโปรแกรมชุดคำสั่งสำหรับการจำลองซีพียู 6502

```
RECORD 1
     OPCODE 69               MNEMONIC ADC             MODE IMM
     NO.BYTES 2              GENERIC MODE IMM
     GENERIC INSTRUCTIONS    ADC A,IMMBYTE

RECORD 2
     OPCODE 65               MNEMONIC ADC             MODE ZPG
     NO.BYTES 2              GENERIC MODE ABS
     GENERIC INSTRUCTIONS    ADC A,MEM

RECORD 3
     OPCODE 61               MNEMONIC ADC             MODE INX
     NO.BYTES 2              GENERIC MODE IDX1IDR
     GENERIC INSTRUCTIONS    ADC A,MEM

RECORD 4
     OPCODE 71               MNEMONIC ADC             MODE INY
     NO.BYTES 2              GENERIC MODE IDRIDX2
     GENERIC INSTRUCTIONS    ADC A,MEM

RECORD 5
     OPCODE 75               MNEMONIC ADC             MODE ZPX
     NO.BYTES 2              GENERIC MODE IDX1
     GENERIC INSTRUCTIONS    ADC A,MEM

RECORD 6
     OPCODE 7D               MNEMONIC ADC             MODE ABX
     NO.BYTES 3              GENERIC MODE IDX1
     GENERIC INSTRUCTIONS    ADC A,MEM

RECORD 7
     OPCODE 79               MNEMONIC ADC             MODE ABY
     NO.BYTES 3              GENERIC MODE IDX2
     GENERIC INSTRUCTIONS    ADC A,MEM

RECORD 8
     OPCODE 90               MNEMONIC BCC             MODE REL
     NO.BYTES 2              GENERIC MODE REL
     GENERIC INSTRUCTIONS    IF C RESET THEN JUMP

RECORD 9
     OPCODE B0               MNEMONIC BCS             MODE REL
     NO.BYTES 2              GENERIC MODE REL
     GENERIC INSTRUCTIONS    IF C SET THEN JUMP
```

```
RECORD 10
     OPCODE F0              MNEMONIC BEQ          MODE REL
     NO.BYTES 2             GENERIC MODE REL
     GENERIC INSTRUCTIONS   IF Z SET THEN JUMP

RECORD 11
     OPCODE 24              MNEMONIC BIT          MODE ZPG
     NO.BYTES 2             GENERIC MODE ABS
     GENERIC INSTRUCTIONS   LOAD T1
                            AND  T1,$40
                            IF Z RESET THEN SET V
                            LOAD T1
                            AND  A,T1
                            TESTSIGN T1

RECORD 12
     OPCODE 30              MNEMONIC BMI          MODE REL
     NO.BYTES 2             GENERIC MODE REL
     GENERIC INSTRUCTIONS   IF N SET THEN JUMP

RECORD 13
     OPCODE D0              MNEMONIC BNE          MODE REL
     NO.BYTES 2             GENERIC MODE REL
     GENERIC INSTRUCTIONS   IF Z RESET THEN JUMP

RECORD 14
     OPCODE 10              MNEMONIC BPL          MODE REL
     NO.BYTES 2             GENERIC MODE REL
     GENERIC INSTRUCTIONS   IF N RESET THEN JUMP

RECORD 15
     OPCODE 00              MNEMONIC BRK          MODE IMP
     NO.BYTES 1             GENERIC MODE IMP
     GENERIC INSTRUCTIONS   SET B
                            STOP

RECORD 16
     OPCODE 50              MNEMONIC BVC          MODE REL
     NO.BYTES 2             GENERIC MODE REL
     GENERIC INSTRUCTIONS   IF V RESET THEN JUMP

RECORD 17
     OPCODE 70              MNEMONIC BVS          MODE REL
     NO.BYTES 2             GENERIC MODE REL
     GENERIC INSTRUCTIONS   IF V SET THEN JUMP
```

```
RECORD 18
     OPCODE 18              MNEMONIC CLC           MODE IMP
     NO.BYTES 1             GENERIC MODE IMP
     GENERIC INSTRUCTIONS   RESET C

RECORD 19
     OPCODE D8              MNEMONIC CLD           MODE IMP
     NO.BYTES 1             GENERIC MODE IMP
     GENERIC INSTRUCTIONS   RESET D

RECORD 20
     OPCODE 58              MNEMONIC CLI           MODE IMP
     NO.BYTES 1             GENERIC MODE IMP
     GENERIC INSTRUCTIONS   RESET I

RECORD 21
     OPCODE 88              MNEMONIC CLV           MODE IMP
     NO.BYTES 1             GENERIC MODE IMP
     GENERIC INSTRUCTIONS   RESET V

RECORD 22
     OPCODE C9              MNEMONIC CMP           MODE IMM
     NO.BYTES 2             GENERIC MODE IMM
     GENERIC INSTRUCTIONS   CP65 A,IMMBYTE

RECORD 23
     OPCODE CD              MNEMONIC CMP           MODE ABS
     NO.BYTES 3             GENERIC MODE ABS
     GENERIC INSTRUCTIONS   CP65 A,MEM

RECORD 24
     OPCODE C5              MNEMONIC CMP           MODE ZPG
     NO.BYTES 2             GENERIC MODE ABS
     GENERIC INSTRUCTIONS   CP65 A,MEM

RECORD 25
     OPCODE C1              MNEMONIC CMP           MODE INX
     NO.BYTES 2             GENERIC MODE IDX1IDR
     GENERIC INSTRUCTIONS   CP65 A,MEM

RECORD 26
     OPCODE D1              MNEMONIC CMP           MODE INY
     NO.BYTES 2             GENERIC MODE IDRIDX2
     GENERIC INSTRUCTIONS   CP65 A,MEM
```

RECORD 27
OPCODE D5 MNEMONIC CMP MODE ZPX
NO.BYTES 2 GENERIC MODE IDX1
GENERIC INSTRUCTIONS CP65 A,MEM

RECORD 28
OPCODE DD MNEMONIC CMP MODE ABX
NO.BYTES 3 GENERIC MODE IDX1
GENERIC INSTRUCTIONS CP65 A,MEM

RECORD 29
OPCODE D9 MNEMONIC CMP MODE ABY
NO.BYTES 3 GENERIC MODE IDX2
GENERIC INSTRUCTIONS CP65 A,MEM

RECORD 30
OPCODE E0 MNEMONIC CPX MODE IMM
NO.BYTES 2 GENERIC MODE IMM
GENERIC INSTRUCTIONS CP65 X,IMMBYTE

RECORD 31
OPCODE EC MNEMONIC CPX MODE ABS
NO.BYTES 3 GENERIC MODE ABS
GENERIC INSTRUCTIONS CP65 X,MEM

RECORD 32
OPCODE E4 MNEMONIC CPX MODE ZPG
NO.BYTES 2 GENERIC MODE ABS
GENERIC INSTRUCTIONS CP65 X,MEM

RECORD 33
OPCODE C0 MNEMONIC CPY MODE IMM
NO.BYTES 2 GENERIC MODE IMM
GENERIC INSTRUCTIONS CP65 Y,IMMBYTE

RECORD 34
OPCODE CC MNEMONIC CPY MODE ABS
NO.BYTES 3 GENERIC MODE ABS
GENERIC INSTRUCTIONS CP65 Y,MEM

RECORD 35
OPCODE C4 MNEMONIC CPY MODE ZPG
NO.BYTES 2 GENERIC MODE ABS
GENERIC INSTRUCTIONS CP65 Y,MEM

```
RECORD 36
     OPCODE CE                 MNEMONIC DEC             MODE ABS
     NO.BYTES 3                GENERIC MODE ABS
     GENERIC INSTRUCTIONS      DECREMENT MEM

RECORD 37
     OPCODE C6                 MNEMONIC DEC             MODE ZPG
     NO.BYTES 2                GENERIC MODE ABS
     GENERIC INSTRUCTIONS      DECREMENT MEM

RECORD 38
     OPCODE D6                 MNEMONIC DEC             MODE ZPX
     NO.BYTES 2                GENERIC MODE IDX1
     GENERIC INSTRUCTIONS      DECREMENT MEM

RECORD 39
     OPCODE DE                 MNEMONIC DEC             MODE ABX
     NO.BYTES 3                GENERIC MODE IDX1
     GENERIC INSTRUCTIONS      DECREMENT MEM

RECORD 40
     OPCODE CA                 MNEMONIC DEX             MODE IMP
     NO.BYTES 1                GENERIC MODE IMP
     GENERIC INSTRUCTIONS      DECREMENT X

RECORD 41
     OPCODE 88                 MNEMONIC DEY             MODE IMP
     NO.BYTES 1                GENERIC MODE IMP
     GENERIC INSTRUCTIONS      DECREMENT Y

RECORD 42
     OPCODE 49                 MNEMONIC EOR             MODE IMM
     NO.BYTES 2                GENERIC MODE IMM
     GENERIC INSTRUCTIONS      XOR A,IMMBYTE

RECORD 43
     OPCODE 4D                 MNEMONIC EOR             MODE ABS
     NO.BYTES 3                GENERIC MODE ABS
     GENERIC INSTRUCTIONS      XOR A,MEM

RECORD 44
     OPCODE 45                 MNEMONIC EOR             MODE ZPG
     NO.BYTES 2                GENERIC MODE ABS
     GENERIC INSTRUCTIONS      XOR A,MEM
```

```
RECORD 45
     OPCODE 41               MNEMONIC EOR            MODE INX
     NO.BYTES 2              GENERIC MODE IDX1IDR
     GENERIC INSTRUCTIONS    XOR A,MEM

RECORD 46
     OPCODE 51               MNEMONIC EOR            MODE INY
     NO.BYTES 2              GENERIC MODE IDRIDX2
     GENERIC INSTRUCTIONS    XOR A,MEM

RECORD 47
     OPCODE 55               MNEMONIC EOR            MODE ZPX
     NO.BYTES 2              GENERIC MODE IDX1
     GENERIC INSTRUCTIONS    XOR A,MEM

RECORD 48
     OPCODE 5D               MNEMONIC EOR            MODE ABX
     NO.BYTES 3              GENERIC MODE IDX1
     GENERIC INSTRUCTIONS    XOR A,MEM

RECORD 49
     OPCODE 59               MNEMONIC EOR            MODE ABY
     NO.BYTES 3              GENERIC MODE IDX2
     GENERIC INSTRUCTIONS    XOR A,MEM

RECORD 50
     OPCODE EE               MNEMONIC INC            MODE ABS
     NO.BYTES 3              GENERIC MODE ABS
     GENERIC INSTRUCTIONS    INCREMENT MEM

RECORD 51
     OPCODE E6               MNEMONIC INC            MODE ZPG
     NO.BYTES 2              GENERIC MODE ABS
     GENERIC INSTRUCTIONS    INCREMENT MEM

RECORD 52
     OPCODE F6               MNEMONIC INC            MODE ZPX
     NO.BYTES 2              GENERIC MODE IDX1
     GENERIC INSTRUCTIONS    INCREMENT MEM

RECORD 53
     OPCODE FE               MNEMONIC INC            MODE ABX
     NO.BYTES 3              GENERIC MODE IDX1
     GENERIC INSTRUCTIONS    INCREMENT MEM
```

```
RECORD 54
     OPCODE E8                MNEMONIC INX          MODE IMP
     NO.BYTES 1               GENERIC MODE IMP
     GENERIC INSTRUCTIONS     INCREMENT X

RECORD 55
     OPCODE C8                MNEMONIC INY          MODE IMP
     NO.BYTES 1               GENERIC MODE IMP
     GENERIC INSTRUCTIONS     INCREMENT Y

RECORD 56
     OPCODE 4C                MNEMONIC JMP          MODE ABS
     NO.BYTES 3               GENERIC MODE ABS
     GENERIC INSTRUCTIONS     JUMP

RECORD 57
     OPCODE 6C                MNEMONIC JMP          MODE IDR
     NO.BYTES 3               GENERIC MODE IDR
     GENERIC INSTRUCTIONS     JUMP

RECORD 58
     OPCODE 20                MNEMONIC JSR          MODE ABS
     NO.BYTES 3               GENERIC MODE ABS
     GENERIC INSTRUCTIONS     INCREMENT SP
                              JUMPSUB

RECORD 59
     OPCODE A9                MNEMONIC LDA          MODE IMM
     NO.BYTES 2               GENERIC MODE IMM
     GENERIC INSTRUCTIONS     TRANSFER A,IMMBYTE

RECORD 60
     OPCODE AD                MNEMONIC LDA          MODE ABS
     NO.BYTES 3               GENERIC MODE ABS
     GENERIC INSTRUCTIONS     LOAD A

RECORD 61
     OPCODE A5                MNEMONIC LDA          MODE ZPG
     NO.BYTES 2               GENERIC MODE ABS
     GENERIC INSTRUCTIONS     LOAD A

RECORD 62
     OPCODE A1                MNEMONIC LDA          MODE INX
     NO.BYTES 2               GENERIC MODE IDX1IDR
     GENERIC INSTRUCTIONS     LOAD A
```

RECORD 63

OPCODE B1	MNEMONIC LDA	MODE INY
NO.BYTES 2	GENERIC MODE IDRIDX2	
GENERIC INSTRUCTIONS	LOAD A	

RECORD 64

OPCODE B5	MNEMONIC LDA	MODE ZPX
NO.BYTES 2	GENERIC MODE IDX1	
GENERIC INSTRUCTIONS	LOAD A	

RECORD 65

OPCODE BD	MNEMONIC LDA	MODE ABX
NO.BYTES 3	GENERIC MODE IDX1	
GENERIC INSTRUCTIONS	LOAD A	

RECORD 66

OPCODE B9	MNEMONIC LDA	MODE ABY
NO.BYTES 3	GENERIC MODE IDX2	

RECORD 67

OPCODE A2	MNEMONIC LDX	MODE IMM
NO.BYTES 2	GENERIC MODE IMM	
GENERIC INSTRUCTIONS	TRANSFER X,IMMBYTE	

RECORD 68

OPCODE AE	MNEMONIC LDX	MODE ABS
NO.BYTES 3	GENERIC MODE ABS	
GENERIC INSTRUCTIONS	LOAD X	

RECORD 69

OPCODE A6	MNEMONIC LDX	MODE ZPG
NO.BYTES 2	GENERIC MODE ABS	
GENERIC INSTRUCTIONS	LOAD X	

RECORD 70

OPCODE BE	MNEMONIC LDX	MODE ABY
NO.BYTES 3	GENERIC MODE IDX2	
GENERIC INSTRUCTIONS	LOAD X	

RECORD 71

OPCODE B6	MNEMONIC LDX	MODE ZPY
NO.BYTES 2	GENERIC MODE IDX2	
GENERIC INSTRUCTIONS	LOAD X	

RECORD 72
OPCODE A0 MNEMONIC LDY MODE IMM
NO.BYTES 2 GENERIC MODE IMM
GENERIC INSTRUCTIONS TRANSFER Y,IMMBYTE

RECORD 73
OPCODE AC MNEMONIC LDY MODE ABS
NO.BYTES 3 GENERIC MODE ABS
GENERIC INSTRUCTIONS LOAD Y

RECORD 74
OPCODE A4 MNEMONIC LDY MODE ZPG
NO.BYTES 2 GENERIC MODE ABS
GENERIC INSTRUCTIONS LOAD Y

RECORD 75
OPCODE B4 MNEMONIC LDY MODE ZPX
NO.BYTES 2 GENERIC MODE IDX1
GENERIC INSTRUCTIONS LOAD Y

RECORD 76
OPCODE BC MNEMONIC LDY MODE ABX
NO.BYTES 3 GENERIC MODE IDX1
GENERIC INSTRUCTIONS LOAD Y

RECORD 77
OPCODE 4E MNEMONIC LSR MODE ABS
NO.BYTES 3 GENERIC MODE ABS
GENERIC INSTRUCTIONS LSHR MEM

RECORD 78
OPCODE 46 MNEMONIC LSR MODE ZPG
NO.BYTES 2 GENERIC MODE ABS
GENERIC INSTRUCTIONS LSHR MEM

RECORD 79
OPCODE 4A MNEMONIC LSR MODE IMP
NO.BYTES 1 GENERIC MODE IMP
GENERIC INSTRUCTIONS LSHR A

RECORD 80
OPCODE 56 MNEMONIC LSR MODE ZPX
NO.BYTES 2 GENERIC MODE IDX1
GENERIC INSTRUCTIONS LSHR MEM

RECORD 81
OPCODE 5E MNEMONIC LSR MODE ABX
NO.BYTES 1 GENERIC MODE IDX1
GENERIC INSTRUCTIONS LSHR MEM

RECORD 82
OPCODE EA MNEMONIC NOP MODE IMP
NO.BYTES 1 GENERIC MODE IMP
GENERIC INSTRUCTIONS NOP

RECORD 83
OPCODE 09 MNEMONIC ORA MODE IMM
NO.BYTES 2 GENERIC MODE IMM
GENERIC INSTRUCTIONS OR A,IMMBYTE

RECORD 84
OPCODE 0D MNEMONIC ORA MODE ABS
NO.BYTES 3 GENERIC MODE ABS
GENERIC INSTRUCTIONS OR A,MEM

RECORD 85
OPCODE 05 MNEMONIC ORA MODE ZPG
NO.BYTES 2 GENERIC MODE ABS
GENERIC INSTRUCTIONS OR A,MEM

RECORD 86
OPCODE 01 MNEMONIC ORA MODE INX
NO.BYTES 2 GENERIC MODE IDX1IDR
GENERIC INSTRUCTIONS OR A,MEM

RECORD 87
OPCODE 11 MNEMONIC ORA MODE INY
NO.BYTES 2 GENERIC MODE IDRIDX2
GENERIC INSTRUCTIONS OR A,MEM

RECORD 88
OPCODE 15 MNEMONIC ORA MODE ZPX
NO.BYTES 2 GENERIC MODE IDX1
GENERIC INSTRUCTIONS OR A,MEM

RECORD 89
OPCODE 1D MNEMONIC ORA MODE ABX
NO.BYTES 3 GENERIC MODE IDX1
GENERIC INSTRUCTIONS OR A,MEM

RECORD 90
OPCODE 19 MNEMONIC ORA MODE ABY
NO.BYTES 3 GENERIC MODE IDX2
GENERIC INSTRUCTIONS OR A,MEM

RECORD 91
OPCODE 48 MNEMONIC PHA MODE IMP
NO.BYTES 1 GENERIC MODE IMP
GENERIC INSTRUCTIONS PUSH65 A

RECORD 92
OPCODE 08 MNEMONIC PHP MODE IMP
NO.BYTES 1 GENERIC MODE IMP
GENERIC INSTRUCTIONS PUSH65 P

RECORD 93
OPCODE 68 MNEMONIC PLA MODE IMP
NO.BYTES 1 GENERIC MODE IMP
GENERIC INSTRUCTIONS PULL65 A

RECORD 94
OPCODE 28 MNEMONIC PLP MODE IMP
NO.BYTES 1 GENERIC MODE IMP
GENERIC INSTRUCTIONS PULL65 P

RECORD 95
OPCODE 2E MNEMONIC ROL MODE ABS
NO.BYTES 3 GENERIC MODE ABS
GENERIC INSTRUCTIONS ROLCARRY MEM

RECORD 96
OPCODE 26 MNEMONIC ROL MODE ZPG
NO.BYTES 2 GENERIC MODE ABS
GENERIC INSTRUCTIONS ROLCARRY MEM

RECORD 97
OPCODE 2A MNEMONIC ROL MODE IMP
NO.BYTES 1 GENERIC MODE IMP
GENERIC INSTRUCTIONS ROLCARRY A

RECORD 98
OPCODE 36 MNEMONIC ROL MODE ZPX
NO.BYTES 2 GENERIC MODE IDX1
GENERIC INSTRUCTIONS ROLCARRY MEM

```
RECORD 99
     OPCODE 3E                MNEMONIC ROL             MODE ABX
     NO.BYTES 3               GENERIC MODE IDX1
     GENERIC INSTRUCTIONS     ROLCARRY MEM

RECORD 100
     OPCODE 6E                MNEMONIC ROR             MODE ABS
     NO.BYTES 3               GENERIC MODE ABS
     GENERIC INSTRUCTIONS     RORCARRY MEM

RECORD 101
     OPCODE 66                MNEMONIC ROR             MODE ZPG
     NO.BYTES 2               GENERIC MODE ABS
     GENERIC INSTRUCTIONS     RORCARRY MEM

RECORD 102
     OPCODE 6A                MNEMONIC ROR             MODE IMP
     NO.BYTES 1               GENERIC MODE IMP
     GENERIC INSTRUCTIONS     RORCARRY A

RECORD 103
     OPCODE 76                MNEMONIC ROR             MODE ZPX
     NO.BYTES 2               GENERIC MODE IDX1
     GENERIC INSTRUCTIONS     RORCARRY MEM

RECORD 104
     OPCODE 7E                MNEMONIC ROR             MODE ABX
     NO.BYTES 3               GENERIC MODE IDX1
     GENERIC INSTRUCTIONS     RORCARRY MEM

RECORD 105
     OPCODE 40                MNEMONIC RTI             MODE IMP
     NO.BYTES 1               GENERIC MODE IMP
     GENERIC INSTRUCTIONS     PULL65 P
                              PULL65 PC
                              JUMPTO PC

RECORD 106
     OPCODE 60                MNEMONIC RTS             MODE IMP
     NO.BYTES 1               GENERIC MODE IMP
     GENERIC INSTRUCTIONS     PULL65 PC
                              JUMPTO PC

RECORD 107
     OPCODE E9                MNEMONIC SBC             MODE IMM
     NO.BYTES 2               GENERIC MODE IMM
     GENERIC INSTRUCTIONS     SBC A,IMMBYTE
```

RECORD 108

OPCODE ED	MNEMONIC SBC	MODE ABS
NO.BYTES 3	GENERIC MODE ABS	
GENERIC INSTRUCTIONS	SBC A,MEM	

RECORD 109

OPCODE E5	MNEMONIC SBC	MODE ZPG
NO.BYTES 2	GENERIC MODE ABS	
GENERIC INSTRUCTIONS	SBC A,MEM	

RECORD 110

OPCODE E1	MNEMONIC SBC	MODE INX
NO.BYTES 2	GENERIC MODE IDX1IDR	
GENERIC INSTRUCTIONS	SBC A,MEM	

RECORD 111

OPCODE F1	MNEMONIC SBC	MODE INY
NO.BYTES 2	GENERIC MODE IDRIDX2	
GENERIC INSTRUCTIONS	SBC A,MEM	

RECORD 112

OPCODE F5	MNEMONIC SBC	MODE ZPX
NO.BYTES 2	GENERIC MODE IDX1	
GENERIC INSTRUCTIONS	SBC A,MEM	

RECORD 113

OPCODE FD	MNEMONIC SBC	MODE ABX
NO.BYTES 3	GENERIC MODE IDX1	
GENERIC INSTRUCTIONS	SBC A,MEM	

RECORD 114

OPCODE F9	MNEMONIC SBC	MODE ABY
NO.BYTES 3	GENERIC MODE IDX2	
GENERIC INSTRUCTIONS	SBC A,MEM	

RECORD 115

OPCODE 38	MNEMONIC SEC	MODE IMP
NO.BYTES 1	GENERIC MODE IMP	
GENERIC INSTRUCTIONS	SET C	

RECORD 116

OPCODE F8	MNEMONIC SED	MODE IMP
NO.BYTES 1	GENERIC MODE IMP	
GENERIC INSTRUCTIONS	SET D	

```
RECORD 117
     OPCODE 78              MNEMONIC SEI          MODE IMP
     NO.BYTES 1             GENERIC MODE IMP
     GENERIC INSTRUCTIONS   SET I


RECORD 118
     OPCODE 8D              MNEMONIC STA          MODE ABS
     NO.BYTES 3             GENERIC MODE ABS
     GENERIC INSTRUCTIONS   STORE A


RECORD 119
     OPCODE 85              MNEMONIC STA          MODE ZPG
     NO.BYTES 2             GENERIC MODE ABS
     GENERIC INSTRUCTIONS   STORE A


RECORD 120
     OPCODE 81              MNEMONIC STA          MODE INX
     NO.BYTES 2             GENERIC MODE IDX1IDR
     GENERIC INSTRUCTIONS   STORE A


RECORD 121
     OPCODE 91              MNEMONIC STA          MODE INY
     NO.BYTES 2             GENERIC MODE IDRIDX2


RECORD 122
     OPCODE 95              MNEMONIC STA          MODE ZPX
     NO.BYTES 2             GENERIC MODE IDX1
     GENERIC INSTRUCTIONS   STORE A


RECORD 123
     OPCODE 9D              MNEMONIC STA          MODE ABX
     NO.BYTES 3             GENERIC MODE IDX1
     GENERIC INSTRUCTIONS   STORE A


RECORD 124
     OPCODE 99              MNEMONIC STA          MODE ABY
     NO.BYTES 3             GENERIC MODE IDX2
     GENERIC INSTRUCTIONS   STORE A


RECORD 125
     OPCODE 8E              MNEMONIC STX          MODE ABS
     NO.BYTES 3             GENERIC MODE STORE X
     GENERIC INSTRUCTIONS


RECORD 126
     OPCODE 86              MNEMONIC STX          MODE ZPG
     NO.BYTES 2             GENERIC MODE ABS
     GENERIC INSTRUCTIONS   STORE X
```

RECORD 127

OPCODE 96	MNEMONIC STX	MODE ZPY
NO.BYTES 2	GENERIC MODE IDX2	
GENERIC INSTRUCTIONS	STORE X	

RECORD 128

OPCODE 8C	MNEMONIC STY	MODE ABS
NO.BYTES 3	GENERIC MODE ABS	
GENERIC INSTRUCTIONS	STORE Y	

RECORD 129

OPCODE 84	MNEMONIC STY	MODE ZPG
NO.BYTES 2	GENERIC MODE ABS	
GENERIC INSTRUCTIONS	STORE Y	

RECORD 130

OPCODE 94	MNEMONIC STY	MODE ZPX
NO.BYTES 2	GENERIC MODE IDX1	
GENERIC INSTRUCTIONS	STORE Y	

RECORD 131

OPCODE AA	MNEMONIC TAX	MODE IMP
NO.BYTES 1	GENERIC MODE IMP	
GENERIC INSTRUCTIONS	TRANSFER X,A	

RECORD 132

OPCODE A8	MNEMONIC TAY	MODE IMP
NO.BYTES 1	GENERIC MODE IMP	
GENERIC INSTRUCTIONS	TRANSFER Y,A	

RECORD 133

OPCODE BA	MNEMONIC TSX	MODE IMP
NO.BYTES 1	GENERIC MODE IMP	
GENERIC INSTRUCTIONS	TRANSFER X,SPL	

RECORD 134

OPCODE 8A	MNEMONIC TXA	MODE IMP
NO.BYTES 1	GENERIC MODE IMP	
GENERIC INSTRUCTIONS	TRANSFER A,X	

RECORD 135

OPCODE 9A	MNEMONIC TXS	MODE IMP
NO.BYTES 1	GENERIC MODE IMP	
GENERIC INSTRUCTIONS	TRANSFER SPL,X	

```
RECORD 136
     OPCODE 98                MNEMONIC TYA            MODE IMP
     NO.BYTES 1               GENERIC MODE IMP
     GENERIC INSTRUCTIONS     TRANSFER A,Y


RECORD 137
     OPCODE 2C                MNEMONIC BIT            MODE ABS
     NO.BYTES 3               GENERIC MODE ABS
     GENERIC INSTRUCTIONS     LOAD T1
                              AND T1,$40
                              IF Z RESET THEN SET V
                              LOAD T1
                              AND A,T1
                              TESTSIGN T1


RECORD 138
     OPCODE 6D                MNEMONIC ADC            MODE ABS
     NO.BYTES 3               GENERIC MODE ABS
     GENERIC INSTRUCTIONS     ADC A,MEM


RECORD 139
     OPCODE 29                MNEMONIC AND            MODE IMM
     NO.BYTES 2               GENERIC MODE IMM
     GENERIC INSTRUCTIONS     AND A,IMMBYTE


RECORD 140
     OPCODE 2D                MNEMONIC AND            MODE ABS
     NO.BYTES 3               GENERIC MODE ABS
     GENERIC INSTRUCTIONS     AND A,MEM


RECORD 141
     OPCODE 25                MNEMONIC AND            MODE ZPG
     NO.BYTES 2               GENERIC MODE ABS
     GENERIC INSTRUCTIONS     AND A,MEM


RECORD 142
     OPCODE 21                MNEMONIC AND            MODE INX
     NO.BYTES 2               GENERIC MODE IDX1IDR
     GENERIC INSTRUCTIONS     AND A,MEM


RECORD 143
     OPCODE 31                MNEMONIC AND            MODE INY
     NO.BYTES 2               GENERIC MODE IDRIDX2
     GENERIC INSTRUCTIONS     AND A,MEM
```

RECORD 144

OPCODE 35	MNEMONIC AND	MODE ZPX
NO.BYTES 2	GENERIC MODE IDX1	
GENERIC INSTRUCTIONS	AND A,MEM	

RECORD 145

OPCODE 3D	MNEMONIC AND	MODE ABX
NO.BYTES 3	GENERIC MODE IDX1	
GENERIC INSTRUCTIONS	AND A,MEM	

RECORD 146

OPCODE 39	MNEMONIC AND	MODE ABY
NO.BYTES 3	GENERIC MODE IDX2	
GENERIC INSTRUCTIONS	AND A,MEM	

RECORD 147

OPCODE 0E	MNEMONIC ASL	MODE ABS
NO.BYTES 3	GENERIC MODE ABS	
GENERIC INSTRUCTIONS	SHL MEM	

RECORD 148

OPCODE 06	MNEMONIC ASL	MODE ZPG
NO.BYTES 2	GENERIC MODE ABS	
GENERIC INSTRUCTIONS	SHL MEM	

RECORD 149

OPCODE 0A	MNEMONIC ASL	MODE IMP
NO.BYTES 1	GENERIC MODE IMP	
GENERIC INSTRUCTIONS	SHL A	

RECORD 150

OPCODE 16	MNEMONIC ASL	MODE ZPX
NO.BYTES 2	GENERIC MODE IDX1	
GENERIC INSTRUCTIONS	SHL MEM	

RECORD 151

OPCODE 1E	MNEMONIC ASL	MODE ABX
NO.BYTES 3	GENERIC MODE IDX1	
GENERIC INSTRUCTIONS	SHL MEM	

ภาคผนวก ค

ข้อมูลสำหรับการจำลองซีพียู Z80

8-BIT REGISTERS

1. R
2. I
3. IM
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.

16-BIT REGISTERS

	Word	Hbyte	Lbyte
1.	AF	A	F
2.	BC	B	C
3.	DE	D	E
4.	HL	H	L
5.	IX	IXH	IXL
6.	IY	IYH	IYL
7.	AF'	A'	F'
8.	BC'	B'	C'
9.	DE'	D'	E'
10.	HL'	H'	L'
11.			
12.			
13.			
14.			
15.			

FLAGS

FLAG REGISTER NAME F

		Flag name	Generic flag
Flag register:	bit 0	C	CARRY
	bit 1	N	
	bit 2	V	OVERFLOW
	bit 3		
	bit 4	H	HALFCARRY
	bit 5		
	bit 6	Z	ZERO
	bit 7	S	SIGN
Other flags:	1	P	PARITY
	2	IF	
	3		

ASSEMBLER AND ADDRESSING

1) Comment separator ;
2) Operand separator ,
3) Mnemonics determine implied addressing mode (Type "," between mnemonics.)
   IM,PUSH,PULL,EX,RST,RET
4) Reserved names in the operand fields (Type "," between names.)
   A,B,C,D,E,H,L,I,R,0,1,2,3,4,5,6,7,SP,BC,DE,HL,IX,IY,C,Z,M,P,NC,NZ,PE,PO,(C),(BC),(DE),(HL),(IX),(IY)

5) Generic addressing modes:- 0 IMP

1 ABS	5 IDXA	9 IDX1
2 IMM	6 IDXAIMM	10 IDX1IDR
3 IO	7 IDXB	11 IDX2
4 REL	8 IDXBIMM	12 IDRIDX2
		13 IDR

If modes 5-16 are supposed to use, enter the register name for each of the selected modes.

	register name
mode 5,6	IX
mode 7,8	IY
mode 9,10	
mode 11,12	

USER-DEFINED ADDRESSING MODES

1 Pattern ()
Md MNE   Mo IO   Mn ABS   Mb   Mw
Opcode
IN,OUT

2 Pattern
Md MNE   Mo REL   Mn IMM   Mb   Mw
Opcode
JR,DJNZ

3 Pattern ()IX
Md IDXA   Mo   Mn   Mb   Mw
Opcode

4 Pattern ()IY
Md IDXB   Mo   Mn   Mb   Mw
Opcode

5 Pattern (),IX
Md IDXAIMM   Mo   Mn   Mb   Mw
Opcode

USER-DEFINED ADDRESSING MODES

6 Pattern (),IY
Md IDXBIMM   Mo   Mn   Mb   Mw
Opcode

7 Pattern
Md   Mo   Mn   Mb   Mw
Opcode

8 Pattern
Md   Mo   Mn   Mb   Mw
Opcode

9 Pattern
Md   Mo   Mn   Mb   Mw
Opcode

10Pattern
Md   Mo   Mn   Mb   Mw
Opcode

Mnemonic	Z80 Mode	Generic Mode	Generic Instruction
SET0A	IMP	IMP	OR A,$01
SET0(HL)	IMP	IMP	LOADBYPTR T1,HL OR T1,$01 STOREBYPTR T1,HL
SET0	INDEXX	IDXA	LOAD T1 OR T1,$01 STORE T1
RES0A	IMP	IMP	AND A,$FE
RES0(HL)	IMP	IMP	LOADBYPTR T1,HL AND T1,$FE STOREBYPTR T1,HL
RES0	INDEXY	IDXB	LOAD T1 AND T1,$FE STORE T1
LDAB	IMP	IMP	TRANSFER A,B
LDA(HL)	IMP	IMP	LOADBYPTR A,HL
LDA	IMM	IMM	TRANSFER A,IMMBYTE
LDA	ABS	ABS	LOAD A
LD(HL)A	IMP	IMP	STOREBYPTR A,HL
LD_A	ABS	ABS	STORE A
LDI	IMP	IMP	LOADBYPTR T1,HL STOREBYPTR T1,DE INCREMENT DE INCREMENT HL DECREMENT BC
LDIR	IMP	IMP	WHILE BC NONZERO DO LOADBYPTR T1,HL STOREBYPTR T1,DE INCREMENT DE INCREMENT HL DECREMENT BC

Mnemonic	Z80 Mode	Generic Mode	Generic Instruction
LDD	IMP	IMP	LOADBYPTR T1,HL STOREBYPTR T1,DE DECREMENT DE DECREMENT HL DECREMENT BC
LDDR	IMP	IMP	WHILE BC NONZERO DO LOADBYPTR T1,HL STOREBYPTR T1,DE DECREMENT DE DECREMENT HL DECREMENT BC
INA	IO	IO	INPUT A
INA(C)	IMP	IMP	INBYPTR A,C
OUT_A	IO	IO	OUTPUT A
OUT(C) A	IMP	IMP	OUTBYPTR A,C
INI	IMP	IMP	INBYPTR T1,C STOREBYPTR T1,HL INCREMENT HL DECREMENT B
INIR	IMP	IMP	WHILE B NONZERO DO INBYPTR T1,C STOREBYPTR T1,HL INCREMENT HL DECREMENT B
IND	IMP	IMP	INBYPTR T1,C STOREBYPTR T1,HL DECREMENT HL DECREMENT B
INDR	IMP	IMP	WHILE B NONZERO DO INBYPTR T1,C STOREBYPTR T1,HL DECREMENT HL DECREMENT B

Mnemonic	Z80 Mode	Generic Mode	Generic Instruction
OUTI	IMP	IMP	LOADBYPTR T1,HL OUTBYPTR T1,C INCREMENT HL DECREMENT B
OTIR	IMP	IMP	WHILE B NONZERO DO LOADBYPTR T1,HL OUTBYPTR T1,C INCREMENT HL DECREMENT B
OUTD	IMP	IMP	LOADBYPTR T1,HL OUTBYPTR T1,C DECREMENT HL DECREMENT B
OTDR	IMP	IMP	WHILE B NONZERO DO LOADBYPTR T1,HL OUTBYPTR T1,C DECREMENT HL DECREMENT B
INCA	IMP	IMP	INCREMENT A
INC(HL)	IMP	IMP	LOADBYPTR T1,HL INCREMENT T1 STOREBYPTR T1,HL
INC	INDEXX	IDXA	INCREMENT MEM
DECA	IMP	IMP	DECREMENT A
DEC(HL)	IMP	IMP	LOADBYPTR T1,HL DECREMENT T1 STOREBYPTR T1,HL
LD	IDXXIMM	IDXAIMM	TRANSFER T1,IMMBYTE STORE T1
CPLA	IMP	IMP	COMPLEMENT A
NEGA	IMP	IMP	TWOCOMP A
ADDB	IMP	IMP	ADD A,B

Mnemonic	Z80 Mode	Generic Mode	Generic Instruction
ADD	IMM	IMM	ADD A,IMMBYTE
ADD(HL)	IMP	IMP	LOADBYPTR T1,HL ADD A,T1
ADD	INDEXX	IDXA	ADD A,MEM
ADCB	IMP	IMP	ADC A,B
ADC	IMM	IMM	ADC A,IMMBYTE
ADC(HL)	IMP	IMP	LOADBYPTR T1,HL ADC A,T1
ADC	INDEXX	IDXA	ADC A,MEM
SUBB	IMP	IMP	SUB A,B
SUB	IMM	IMM	SUB A,IMMBYTE
SUB(HL)	IMP	IMP	LOADBYPTR T1,HL SUB A,T1
SUB	INDEXX	IDXA	SUB A,MEM
SBCB	IMP	IMP	SBC A,B
SBC	IMM	IMM	SBC A,IMMBYTE
SBC(HL)	IMP	IMP	LOADBYPTR T1,HL SBC A,T1
SBC	INDEXX	IDXA	SBC A,MEM
DAAA	IMP	IMP	DECADJ A
ANDB	IMP	IMP	AND A,B
AND(HL)	IMP	IMP	LOADBYPTR T1,HL AND A,T1
AND	IMM	IMM	AND A,IMMBYTE
AND	INDEXX	IDXA	AND A,MEM
BIT0A	IMP	IMP	TRANSFER T1,A AND T1,$01
BIT0(HL)	IMP	IMP	LOADBYPTR T1,HL AND T1,$01
BIT0	INDEXX	IDXA	LOAD T1 AND T1,$01
CPB	IMP	IMP	COMPARE A,B

Mnemonic	Z80 Mode	Generic Mode	Generic Instruction
CP	IMM	IMM	COMPARE A,IMMBYTE
CP(HL)	IMP	IMP	LOADBYPTR T1,HL COMPARE A,T1
CP	INDEXX	IDXA	COMPARE A,MEM
CPI	IMP	IMP	LOADBYPTR T1,HL COMPARE A,T1 INCREMENT HL DECREMENT BC
CPIR	IMP	IMP	WHILE Z RESET AND BC NONZERO DO LOADBYPTR T1,HL INCREMENT HL DECREMENT BC COMPARE A,T1
CPD	IMP	IMP	LOADBYPTR T1,HL COMPARE A,T1 DECREMENT HL DECREMENT BC
CPDR	IMP	IMP	WHILE Z RESET AND BC NONZERO DO LOADBYPTR T1,HL DECREMENT HL DECREMENT BC COMPARE A,T1
RLCA	IMP	IMP	ROL A
RRCA	IMP	IMP	ROR A
RLA	IMP	IMP	ROLCARRY A
RRA	IMP	IMP	RORCARRY A
SLAA	IMP	IMP	SHL A
SRAA	IMP	IMP	ASHR A
SRLA	IMP	IMP	LSHR A

Mnemonic	Z80 Mode	Generic Mode	Generic Instruction
LDHL	IMM	IMM	TRANSFER HL,IMMWORD
LDHL	ABS	ABS	LOAD HL
LD_HL	ABS	ABS	STORE HL
LDSPHL	IMP	IMP	TRANSFER SP,HL
PUSHHL	IMP	IMP	PUSH HL
POPHL	IMP	IMP	PULL HL
EXDEHL	IMP	IMP	EXCHANGE DE,HL
EXX	IMP	IMP	EXCHANGE BC,BC' EXCHANGE DE,DE' EXCHANGE HL,HL'
EX(SP)HL	IMP	IMP	PULL T3 PUSH HL TRANSFER HL,T3
INCHL	IMP	IMP	INCREMENT HL
DECHL	IMP	IMP	DECREMENT HL
ADDHLBC	IMP	IMP	ADD HL,BC
ADCHLBC	IMP	IMP	ADC HL,BC
SBCHLBC	IMP	IMP	SBC HL,BC
SCF	IMP	IMP	SET C
CCF	IMP	IMP	RESET C
EI	IMP	IMP	SET IF
DI	IMP	IMP	RESET IF
IM0	IMP	IMP	TRANSFER IM,0
IM1	IMP	IMP	TRANSFER IM,1
IM2	IMP	IMP	TRANSFER IM,2
HALT	IMP	IMP	STOP
NOP	IMP	IMP	NOP
JP	IMM	IMM	JUMP
JR	REL	REL	JUMP

Mnemonic	Z80 Mode	Generic Mode	Generic Instruction
JPC	IMM	IMM	IF C SET THEN JUMP
JPNC	IMM	IMM	IF C RESET THEN JUMP
JPZ	IMM	IMM	IF Z SET THEN JUMP
JPNZ	IMM	IMM	IF Z RESET THEN JUMP
JPM	IMM	IMM	IF S SET THEN JUMP
JPP	IMM	IMM	IF S RESET THEN JUMP
JPPE	IMM	IMM	IF P SET THEN JUMP
JPPO	IMM	IMM	IF P RESET THEN JUMP
JRC	REL	REL	IF C SET THEN JUMP
JRNC	REL	REL	IF C RESET THEN JUMP
JRZ	REL	REL	IF Z SET THEN JUMP
JRNZ	REL	REL	IF Z RESET THEN JUMP
JP(HL)	IMP	IMP	JUMPTO HL
JP(IX)	IMP	IMP	JUMPTO IX
DJNZ	REL	REL	DECREMENT B IF B NONZERO THEN JUMP
CALL	IMM	IMM	JUMPSUB
RST0	IMP	IMP	PUSH PC JUMPTO $00
RST1	IMP	IMP	PUSH PC JUMPTO $08
RST7	IMP	IMP	PUSH PC JUMPTO $38
CALLC	IMM	IMM	IF C SET THEN JUMPSUB
CALLNC	IMM	IMM	IF C RESET THEN JUMPSUB
RET	IMP	IMP	RETSUB
RETC	IMP	IMP	IF C SET THEN RETSUB
RETNC	IMP	IMP	IF C RESET THEN RETSUB
RETI	IMP	IMP	RETSUB

ประวัติผู้เขียน

นางสาวมธุรสชื่นยง เกิดวันที่ 5 กรกฎาคม 2504 ที่กรุงเทพฯ สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัยในปีการศึกษา 2525 และได้เข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต ที่จุฬาลงกรณ์มหาวิทยาลัย เมื่อ พ.ศ.2532 ปัจจุบันเป็นหัวหน้าแผนกสิมูเลเตอร์ ฝ่ายฝึกอบรม ศูนย์ฝึกอบรมบางปะกง การไฟฟ้าฝ่ายผลิตแห่งประเทศไทย