

Etapele de bază ale conceperii unui program sunt analiza problemei, analiza și construirea interfeței, scrierea codului, testarea și depanarea. Delphi a fost gândit astfel încât programatorul care dispune de o analiză serioasă și bine pusă la punct să poată trece rapid peste partea de construire a interfeței, concentrându-se asupra scrierii codului și, în final, a depanării. Platforme non-vizuale (Borland Pascal 7 sau Borland C++ 3) înseamnă: interfața construită după o muncă imensă se poate face în doar câteva ore cu un instrument vizual (Delphi, Visual C++, Visual Basic etc.). De asemenea, folosirea obiectelor și a claselor este simplificată la maxim.

Delphi, ca toate instrumentele vizuale, este în mod exclusiv bazat pe tehnicile de programare orientată pe obiecte. Fiind un descendent al limbajului Pascal, beneficiază de lizibilitatea foarte bună a codului.

În Delphi nu mai există, din punctul de vedere al compilatorului, diferențe între alocarea statică și dinamică a memoriei. Interfața de operare a Delphi-ului este compusă din patru elemente: tabela de componente (obiecte), inspectorul de obiecte (Object Inspector), editorul de cod și editorul de interfață.

Tabela de componente. Este o bară de butoane,

fiecare reprezentând o clasă de obiecte. Aceste butoane sunt grupate pe categorii (Standard, Additional, Internet etc.). O categorie foarte importantă este cea a obiectelor standard Windows 9X, cu ajutorul căruia aplicația creată va lucra rapid.

Inspectorul de obiecte (Object Inspector) sau mai explicit, editorul de attribute și asocieri, este instrumentul utilizat pentru a modifica attributele unui obiect (textul scris pe un buton, vizibilitatea unei cutii de selectare) și pentru a face legătura între metoda (operația) descrisă în cod de programator și obiectul căruia îi este dedicată metoda. Fiecare obiect al unei clase are un număr de metode și attribute predefinite la care se pot însă adăuga metodele și attributele proprii programatorului, la nivel de clasă.

Editorul de cod este un instrument de lucru cu care sunt obișnuiți utilizatorii de programare în Pascal sau C. Aspectul diferă

puțin (sau mai mult) față de versiunile în mod text ale platformelor amintite mai sus.

Obișnuitele Breakpoint-uri (puncte în care execuția programului este oprită în cazul depanării) se fac cu un simplu clic pe editor în stânga liniei la care dorim plasarea acestuia.

Editorul de interfață formează, împreună cu tabela de componente și inspectorul de obiecte, 'baza' proiectării vizuale. Este o fereastră (Form) pe care se așează componentele vizuale și nonvizuale cu care va lucra programul. Fiecărei ferestre îi corespunde un 'unit' care conține, pe lângă funcțiile și procedurile proprii programatorului, descrierile metodelor implementate de programator și folosite de fereastră precum și obiectele de pe aceasta. Pentru

plasarea acestora (de exemplu plasarea unui buton) se selectează din tabela de componente clasa din care face parte obiectul pe care dorim să-l folosim și se specifică (printr-un clic) fereastra în care trebuie plasat. În această fereastră va apare un obiect al cărui atribute și asocieri cu metodele folosite de acesta le putem modifica din inspectorul de obiecte. Din paleta de componente nu se selectează un anumit obiect, ci o clasă de obiecte care va fi reprezentată în fereastră de un obiect. Două atribute specifice obiectelor vizuale, dimensiunea și poziția în fereastră, pot fi modificate direct din fereastră, fără a folosi inspectorul de obiecte, în mod similar cu aplicarea aceleiași operații pe o fereastră a unui program Windows. Este foarte important faptul că o fereastră este ea însăși o

componentă (obiectul 'Form1' aparține clasei 'Tform'). O nouă fereastră se creează cu ajutorul unui buton special, în stânga paletei de componente.

Aplicația este văzută ca un ansamblu de forme - ferestre și dialoguri clasice Windows - care ocupă un rol central în arhitectura programelor. Forma este entitatea care cuprinde celelalte componente și al cărei scop global permite o viziune completă a interacțiunii acestora și facilitează obținerea unei funcționalități specifice. Reflectarea formei în cod se face ca o unitate distinctă (Pascal unit) care declară în partea de interfață o clasă de tip TForm având drept câmpuri obiectele componente. Inițial noua formă nu include metode specifice,

dar conține o apreciabilă doză de funcționalitate prin cele câteva zeci de metode private și publice moștenite de la TForm ca și prin comportamentul standard al obiectelor componente care asigură aspectul și controlul interacțiunilor dintre componente. În faza aceasta forma este perfect compilabilă și utilizabilă chiar fără vreo linie suplimentară de cod. Cum din unitatea asociată de cod nu reiese nimic, se poate pune întrebarea cum sunt încorporate definițiile proprietăților, pozițiilor componentelor și formei, într-un cuvânt, desenul? Răspunsul este dat de asocierea fișierului unitate cu un fișier de tip resursă binară Windows (cu extensie DFM) care se link-editează în codul executabil și are același rol ca și resursele standard dar care nu conține dialoguri, imagini bitmap sau iconuri, ci un format

special încărcat de constructor la crearea obiectului formă.

Componentele 'vizibile' pot fi controale clasice Windows gen butoane, editoare, liste de selecție, etc. dar și o varietate de controale sofisticate cu funcții noi oferite în standard de Delphi. Tot vizibile sunt și componentele de tip dialog care automatizează funcțiile comune de tip selecție de fișier, font, culoare, etc. În spatele interfeței cu utilizatorul de pe formă se găsește componentele invizibile care conferă funcționalitatea aplicației din care cele mai semnificative sunt obiectele de acces la bazele de date. Un avantaj remarcabil îl conferă lui Delphi capacitatea de a produce și integra în bibliotecă componente complet noi sau derivate din

tipurile existente.

Particularizarea unui component se face prin modificarea vizuală a proprietăților acestuia. Inspectorul de proprietăți permite specificarea aspectului unei forme sau al unui control vizibil din formă facilitând spre exemplu setarea culorii, a unui font specific, etc. Mult mai spectaculoase sunt proprietățile care descriu interacțiunea componentelor cu alte componente sau cu structuri de date în memorie sau pe disc, diferențierea comportamentului în interacțiune cu utilizatorul. Toată această informație se stochează în întregime în resursă și

permite, spre exemplu, conceperea unei machete sofisticate de introducere a datelor fără generarea vreunei metode specifice în secțiunea de cod.

Când un component întâlnește o condiție deosebită - spre exemplu, un clic de mouse pe un component-buton sau actualizarea unei înregistrări pe disc pentru un component-tabelă - acesta generează un eveniment care se traduce prin apelul unei metode dedicate a formei care conține obiectul, al cărei scop este executarea unei funcții specifice. Evenimentele care pot fi generate de un component sunt

accesibile în editorul vizual, la fel ca și proprietățile: un dublu clic pe un eveniment din listă și Delphi selectează sau generează automat o metodă specifică pentru formă și plasează cursorul în editorul de cod acolo unde trebuie scris sau există deja codul specific evenimentului.

Această tehnică de notificare a formei de către component la apariția unui eveniment reprezintă o modalitate elegantă de delegare a responsabilității reacției către formă, singurul obiect care deține o perspectivă globală și este în măsură să acționeze corespunzător.

Despre limbajul de asamblare

Limbajul de asamblare (ASM) permite înțelegerea la nivel de amănunt a ceea ce întâmplă în realitate într-un calculator.

Există mai multe motive pentru care programarea în ASM este necesară. Codul generat în ASM este în general foarte rapid. Unele module de program trebuie implementate în ASM datorită

acestei viteze de lucru. Uneori, o parte a unui program este scrisă într-un limbaj de nivel înalt, iar modulele critice sunt scrise ca proceduri ASM, apelate la rândul lor de modulele de nivel înalt.

Pe de altă parte, există situații în care e nevoie de acces direct la dispozitive de intrare/ieșire sau la locații fizice de memorie, iar aceste operații nu pot fi executate în unele limbaje de nivel înalt.

Utilizarea în diferite aplicații mixte (limbaj de nivel înalt și limbajul de

asamblare) este de multe ori soluția cea mai eficientă.

În ASM, calculatorul este văzut la nivelul hardware: adrese fizice de memorie, registre, întreruperi etc.

Unitatea de bază a informației memorate este bitul. Un bit reprezintă o cifră binară ce poate avea valoarea '0' sau '1'. Modelul hardware corespunzător este acela de bistabil, un circuit electronic cu două stări stabile, codificate 0 și 1, capabil să memoreze un bit de

informație. Un grup de bistabili
formează un registru. În general un
registru de

n

biți va putea memora 2

n

combinații distincte. Aceste
combinații se numesc octeți sau
bytes.

Programul de citire și scriere

În figura 4-1 este ilustrată interfața programului.

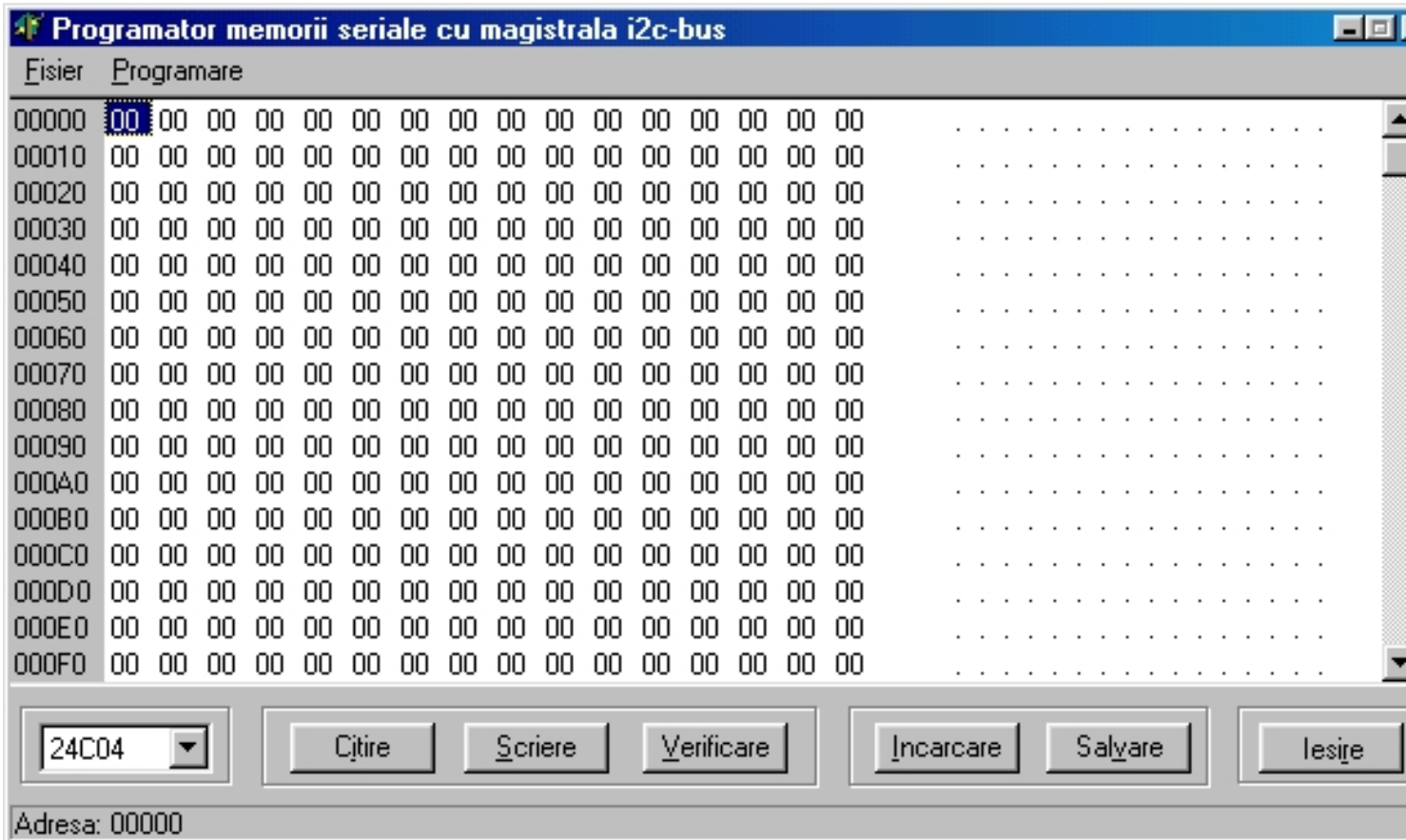


Figura 4-1: Interfața programului de citire/scriere a memoriilor EEPROM

Fișierele program sunt următoarele:

eeprom.dpr - fișierul de proiect delphi

eeprom.dfm - fișierul de resursă binară

eeprom.pas - fișierul principal

IV.3.1. Conținutul fișierului eeprom.dpr

program eeprom;

uses

Forms,

```
eeprom in 'eeprom.pas'  
{Form1};
```

```
{$R *.RES}
```

begin

Application.Initialize;

**Application.CreateForm(T
Form1, Form1);**

Application.Run;

end.

Conținutul fișierului eeprom.dfm

object Form1: TForm1

Left = 192

Top = 106

BorderIcons =

[biSystemMenu,
biMinimize]

BorderStyle = bsSingle

Caption = 'Eeprom
programmer'

ClientHeight = 309

ClientWidth = 591

Color = clBtnFace

Font.Charset =
DEFAULT_CHARSET

Font.Color =
clWindowText

Font.Height = -11

Font.Name = 'MS Sans
Serif'

Font.Style = []

Menu = MainMenu1

OldCreateOrder = False

Position =
poDesktopCenter

OnActivate =
FormActivate

OnCreate =
FormCreate

PixelsPerInch = 96

TextHeight = 13

object Bevel1: TBevel

Left = 4

Top = 253

Width = 88

Height = 33

Shape = bsFrame

end

object Bevel2: TBevel

Left = 347

Top = 253

Width = 150

Height = 34

Shape = bsFrame

end

object Bevel3: TBevel

Left = 508

Top = 253

Width = 77

Height = 33

Shape = bsFrame

end

object Bevel4: TBevel

Left = 104

Top = 253

Width = 231

Height = 34

Shape = bsFrame

end

object statusBar:
TStatusBar

Left = 0

Top = 292

Width = 591

Height = 17

Panels = <>

SimplePanel = True

SizeGrip = False

end

object selEeprom:
TComboBox

Left = 12

Top = 260

Width = 72

Height = 21

Style =
csDropDownList

ItemHeight = 13

TabOrder = 1

```
OnChange =  
selEepromChange
```

```
end
```

```
object sGrid:  
TStringGrid
```

Left = -3

Top = 0

Width = 589

Height = 245

ColCount = 36

DefaultColWidth = 12

**DefaultRowHeight =
15**

RowCount = 16

FixedRows = 0

Font.Charset =
DEFAULT_CHARSET

Font.Color = clBlack

Font.Height = -12

```
Font.Name = 'MS  
Sans Serif'
```

```
Font.Style = []
```

```
Options =  
[goDrawFocusSelected]
```

ParentFont = False

ScrollBars = ssVertical

TabOrder = 2

OnClick = sGridClick

**OnKeyPress =
sGridKeyPress**

**OnMouseMove =
sGridMouseMove**

OnSelectCell =
sGridSelectCell

RowHeights = (15
15 15 15 15
15 15 15 15
15 15 15 15
15

15 15)

end

object ExitButton:
TBitBtn

Left = 517

Top = 260

Width = 60

Height = 21

Caption = 'lesi&re'

TabOrder = 3

OnClick =
ExitButtonClick

end

object readButton:
TButton

Left = 116

Top = 260

Width = 60

Height = 21

Caption = 'C&itire'

TabOrder = 4

```
OnClick =  
readButtonClick
```

```
end
```

```
object PageWrite:  
TButton
```

Left = 188

Top = 260

Width = 60

Height = 21

Caption = '&Scriere'

TabOrder = 5

OnClick =
PageWriteClick

end

object Verificare:
TButton

Left = 262

Top = 260

Width = 60

Height = 21

Caption = '&Verificare'

TabOrder = 6

OnClick =
VerificareClick

end

object Button5: TButton

Left = 358

Top = 260

Width = 60

Height = 21

Caption = '&Incarcare'

TabOrder = 7

OnClick = Load1Click

end

object Button6: TButton

Left = 429

Top = 260

Width = 60

Height = 21

Caption = 'Sal&vare'

TabOrder = 8

OnClick =
SaveAs1Click

end

object MainMenu1:
TMainMenu

Left = 480

Top = 65532

object File1:
TMenuItem

Caption = '&Fisier'

object Load1:
TMenuItem

Caption =
'&Incarcare'

OnClick =
Load1Click

end

object SaveAs1:
TMenuItem

Caption = '&Salvare'

OnClick =
SaveAs1Click

end

object N1:
TMenuItem

Caption = '-'

end

object Exit1:
TMenuItem

Caption = 'I&esire'

OnClick = Exit1 Click

end

end

object Edit1 :
TMenuItem

Caption =
'&Programare'

object Read1:
TMenuItem

Caption = '&Citire'

OnClick =
Read1Click

end

object Write1:
TMenuItem

Caption = '&Scriere'

OnClick =
Write1 Click

end

object Verify1:
TMenuItem

Caption =
'&Verificare'

OnClick =
Verify1Click

end

object N2:
TMenuItem

Caption = '-'

end

object ClearBuffer1:
TMenuItem

Caption = 'Stergere
&buffer'

OnClick =
ClearBuffer1 Click

end

end

end

object openDlg:
TOpenDialog

Left = 516

Top = 65532

end

object saveDlg:
TSaveDialog

Left = 552

Top = 65532

end

end

IV.3.3. Conținutul fișierului eeprom.pas

unit eeprom;

interface

//clauza uses spune
compilatorului ce unit-uri
sunt utilizate

uses

Windows, Messages,
SysUtils, Classes,

Graphics, Controls,
Forms, Dialogs,

Grids, ExtCtrls,
StdCtrls, Buttons,
Menus, ComCtrls,
ToolWin;

//declaratii de tipuri,
proceduri, functii

type

TForm1 =
class(TForm)

```
    statusBar:  
TStatusBar;
```

```
    selEeprom:  
TComboBox;
```

```
    sGrid: TStringGrid;
```

ExitButton: TBitBtn;

MainMenu1:
TMainMenu;

File1: TMenuItem;

Edit1 : TMenuItem;

Exit1 : TMenuItem;

Load1 : TMenuItem;

Read1 : TMenuItem;

Write1 : TMenuItem;

Verify1 : TMenuItem;

N1: TMenuItem;

SaveAs1:
TMenuItem;

ClearBuffer1:
TMenuItem;

openDlg:
TOpenDialog;

saveDlg:
TSaveDialog;

readButton:

TButton;

PageWrite: TButton;

Verificare: TButton;

Button5: TButton;

Button6: TButton;

Bevel1: TBevel;

Bevel2: TBevel;

Bevel3: TBevel;

Bevel4: TBevel;

N2: TMenuItem;

```
procedure  
sGridKeyPress(Sender  
: TObject; var Key:  
Char);
```

```
procedure  
FormCreate(Sender:  
TObject);
```

```
procedure  
sGridSelectCell(Sende  
r: TObject; ACol,  
ARow: Integer;
```

```
var CanSelect:  
Boolean);
```

```
procedure  
sGridClick(Sender:  
TObject);
```

```
procedure  
ExitButtonClick(Sender  
: TObject);
```

```
procedure  
selEepromChange(Sen  
der: TObject);
```

```
procedure  
convert(str: String);
```

```
procedure  
Load1Click(Sender:  
TObject);
```

```
procedure  
SaveAs1Click(Sender:  
TObject);
```

```
procedure  
sGridMouseMove(Sen  
der: TObject; Shift:  
TShiftState; X,
```

Y: Integer);

procedure
ClearBuffer1 Click(Sen
der: TObject);

```
procedure  
setGrid(rSize: Integer);
```

```
procedure inport;
```

```
procedure  
i2c_start;
```

```
procedure i2c_stop;
```

```
procedure  
gridFocus();
```

```
procedure  
FormActivate(Sender:
```

TObject);

procedure
readButtonClick(Sender:
TObject);

procedure

PageWriteClick(Sender
: TObject);

procedure
VerificareClick(Sender:
TObject);

```
function ack:  
Integer;  
  
procedure data1;  
  
procedure data0;
```

procedure clock1;

procedure clock0;

procedure delay;

```
procedure  
longdelay;
```

```
procedure byteout;
```

```
end;
```

`var //declararea
variabilelor`

`Form1: TForm1;`

`verify, oneTest, cond,
editing: Boolean;`

hexValue: Char;

oldFileName, con,
rectValue, nextValue:
String;

eepromType: Word;

readFile, saveFile:
file of Byte;

location:array[1..524
288] of byte;

location_verify:array[

1..524288] of byte;

to_mem,i ,readed,
index, cx, cy,
from_mem: Integer;

const //declararea
constantelor

eeeproms: array[0..9]
of Integer =

(8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096);

//

1 2 4 8 16 32 64
128 256 512

SCL: Integer = 1;

SDA: Integer = 2;

implementation

{ \$R *.DFM }

//procedura de
examinare a apasarii
unei taste

procedure

```
TForm1.sGridKeyPress  
s(Sender: TObject; var  
Key: Char);
```

```
var
```

```
a, intT : Integer;
```

```
myRect: TGridRect;
```

```
begin
```

```
    if (sGrid.Col <> 0)  
OR (sGrid.Col < 16)  
OR editing then
```

begin

```
    if (Key < Char(71))  
And (Key > Char(64))  
Or (Key < Char(103))  
And (Key > Char(96))  
Or (Key < Char(58))
```

And (Key > Char(47))
then

begin

if not oneTest then
//prima apasare a

tastei in cadrul unui
camp

begin

oneTest := true;

```
rectValue :=  
sGrid.Cells[cx, cy];
```

```
nextValue :=  
rectValue;
```

```
    rectValue := Key  
+ nextValue[2];
```

```
    nextValue :=  
nextValue[2] +  
rectValue[1];
```

```
sGrid.Cells[cx,  
cy]  
:=UpperCase(rectValue);
```

```
hexValue :=  
sGrid.Cells[cx, cy][1];
```

```
convert(hexValue  
);
```

```
intT :=  
StrToInt(con)*16;
```

```
hexValue :=
```

```
sGrid.Cells[cx, cy][2];
```

```
        convert(hexValue  
);
```

```
        intT := intT +  
StrToInt(con);
```

```
location[(cy)*16 +  
cx] := intT;
```

```
if intT < 20 then
```

```
    sGrid.Cells[cx+1  
9, cy] := '.'
```

else

 sGrid.Cells[cx+1
9, cy] := Chr(intT)

end

else

begin

oneTest := false;

```
rectValue :=  
nextValue[2] + Key;
```

```
sGrid.Cells[cx,  
cy]  
:=UpperCase(rectValue);
```

```
hexValue :=  
sGrid.Cells[cx, cy][1];
```

```
convert(hexValue  
);
```

```
intT :=
```

```
StrToInt(con)*16;
```

```
        hexValue :=  
sGrid.Cells[cx, cy][2];
```

```
        convert(hexValue  
e);
```

```
intT := intT +  
StrToInt(con);
```

```
location[(cy)*16 +  
cx] := intT;
```

```
if intT < 32 then
```

```
        sGrid.Cells[cx+1  
9, cy] := '.'
```

```
else
```

```
sGrid.Cells[cx+1  
9, cy] := Chr(intT);
```

```
if cx = 16 then
```

```
begin
```

CX := 1;

if cy <>
eeproms[index] - 1
then

cy := cy + 1

else

cy := 0

end

else

CX := CX + 1;

**myRect.Left :=
CX;**

```
myRect.Top :=  
cy;
```

```
myRect.Right :=  
cx;
```

```
myRect.Bottom :=
```

cy;

**sGrid.Selection :=
myRect;**

i := cx-1 + cy*16;

```
        statusBar.simplet  
ext := 'Adresa: ' +  
IntToHex(i , 5);
```

```
end
```

```
end
```

end;

end;

//procedura de creare
a interfetei

```
procedure  
TForm1.FormCreate(S  
ender: TObject);
```

```
var
```

```
i, j, counti, countj:
```

Integer;

begin

verify := false;

editing := false;

sGrid.Visible := false;

cond:=false;

```
for countj :=0 to 35  
do //consrtuirea  
grilajului de editor
```

```
if countj = 0 then
```

```
sGrid.ColWidths[0
```

] := 40

else if (countj > 0)
and (countj < 17) then

sGrid.ColWidths[c
ountj] := 20

```
else if (countj > 16)  
and (countj < 41) then
```

```
    sGrid.ColWidths[c  
countj] := 10;
```

```
selEeprom.items.Add
```

('24C01'); //eepromuri
ce pot fi programate

selEeprom.items.Add
('24C02');

selEeprom.items.Add

('24C04');

selEeprom.items.Add
d('24C08');

selEeprom.items.Add
('24C16');

```
selEeprom.items.Add  
('24C32');
```

```
selEeprom.items.Add  
('24C64');
```

```
selEeprom.items.Add
```

('24C128');

selEeprom.items.Add
('24C256');

selEeprom.items.Add
('24C512');

```
selEeprom.ItemIndex  
:= 2; //initializarea cu  
'24c04'
```

```
index := 2;
```

```
editing := true;
```

//editare => da

setGrid(eeproms[ind
ex]); //incarcarea
grilajului cu "0"

sGrid.Visible := true;

```
    cx := 1;  
//linia 1  
  
    cy := 0;  
//coloana 0  
  
end;
```

//procedura de
selectare a celulei

procedure
TForm1.sGridSelectCe
ll(Sender: TObject;
ACol, ARow: Integer;

```
var CanSelect:  
Boolean);
```

```
begin
```

```
cx := ACol;
```

cy := ARow;

end;

**//procedura de
examinare a unui clic
pe grilaj**

```
procedure  
TForm1.sGridClick(Se  
nder: TObject);
```

```
var //variabile
```

```
TheRect: TRect;
```

x, y : Integer;

myRect: TGridRect;

begin

```
oneTest := false;
```

```
rectValue :=  
sGrid.Cells[cx,cy];  
//dreptunghi activ
```

```
sGrid.Refresh;
```

```
if (sGrid.Col Mod 3) =  
0 then
```

```
begin
```

```
myRect.Left := cx;
```

myRect.Top := cy;

myRect.Right := cx;

**myRect.Bottom :=
cy;**

```
sGrid.Selection :=  
myRect;
```

```
end;
```

```
if (cx > 0) And (cx <  
17) And (cy <
```

```
eeproms[index]) then  
//afisarea adresei
```

```
begin
```

```
i := cx-1 + cy*16;
```

```
    statusBar.simpletext  
:= 'Adresa: ' +  
IntToHex(i , 5);
```

```
end
```

```
else if (cx > 19) And
```

(cx < 36) And (cy <
eeproms[index]) then

begin

i := cx-20 + cy*16;

```
        statusBar.simpletext  
t := 'Adresa: ' +  
IntToHex(i , 5);
```

```
end
```

```
else
```

statusBar.simpletext

```
:= ";
```

```
end;
```

//procedura de iesire

```
procedure  
TForm1.ExitButtonClic  
k(Sender: TObject);
```

```
begin
```

```
close();
```

```
end;
```

```
//procedura de  
schimbare a tipului de  
memorie eeprom
```

```
procedure  
TForm1.selEepromCha  
nge(Sender: TObject);
```

```
begin
```

```
    editing := true;
```

```
statusbar.SimpleText  
:= 'Modificarea  
memoriei...';
```

```
sGrid.Visible := true;
```

```
index :=
```

```
selEeprom.ItemIndex;
```

```
    setGrid(eeproms[ind  
ex]);
```

```
sGrid.SetFocus;
```

```
saveAs1.Enabled :=  
true;
```

```
load1.Enabled :=  
true;
```

```
end;
```

//procedura de
refacere a grilajului in
functie de capacitatea
memoriei

procedure
TForm1.setGrid(rSize:
Integer);

var //variabile

**i, peer, counti, countj,
intT: Integer;**

strT: String;

begin

sGrid.RowCount :=
rSize;

for counti := 0 to
rSize - 1 do

begin

for countj := 0 to
16 do

begin

if countj = 0 then

 sGrid.Cells[0,
counti] :=
 inttohex(counti,4) +
 IntToStr(0)

else

sGrid.Cells[countj
, counti] := '00';

end;

end;

```
    for counti := 0 to  
rSize - 1 do
```

```
begin
```

```
    for countj:= 20 to 35  
do
```

begin

sGrid.Cells[coun
tj, counti] := '.'

end;

end;

```
    statusBar.simpletext  
:= 'Adresa: ' +  
IntToHex(0 , 5);
```

end;

//procedura de
conversie zecimal =>
hexazecimal

procedure
TForm1.convert(str:
String);

begin;

case hexValue of

'0'..'9': con :=
hexValue;

'A', 'a': con :=
IntToStr(10);

'B', 'b': con :=
IntToStr(11);

'C', 'c': con :=

IntToStr(12);

'D', 'd': con :=

IntToStr(13);

'E', 'e': con :=

IntToStr(14);

```
'F', 'f': con :=  
IntToStr(15);
```

```
end;
```

```
end;
```

//procedura de
incarcare a unui fisier
in editorul hexa

procedure
TForm1.Load1Click(Se
nder: TObject);

var //variabile

**counti, countj, intT, i,
j: Integer;**

myRect: TGridRect;

label there;

begin

 openDlg.FileName :=
";
;

```
openDlg.Execute;
```

```
AssignFile(readFile,  
openDlg.FileName);
```

```
if (openDlg.FileName  
= "") then goto there;
```

//deschiderea fisierului

Reset(readFile);

j := eeproms[index]
div 8 * 128;

```
for j := 1 to  
eeproms[index]*16 do
```

```
location[j] := 0;
```

```
i := 1;
```

```
statusbar.SimpleText  
:= 'Citire ... ';
```

```
while not  
Eof(readFile) do
```

```
begin
```

```
read(readFile,  
location[i]);
```

```
i := i + 1;
```

```
end;
```

```
CloseFile(readFile);  
//inchiderea fisierului
```

```
statusbar.SimpleText  
:= 'Incarcare ...';
```

```
i:=1;
```

```
for cy := 0 to  
eeproms[index] do  
//incarcarea grilajului  
cu cifre hexa
```

```
begin
```

```
for cx := 1 to 16 do
```

```
begin
```

```
    sGrid.Cells[cx,cy]:  
=IntToHex(location[i],2)  
    ;
```

i := i + 1;

end;

end;

```
for counti := 0 to  
eeproms[index] - 1 do  
  //incarcarea cu  
  caractere ascii
```

```
begin
```

```
for countj:= 20 to 35  
do
```

```
begin
```

```
hexValue :=  
sGrid.Cells[countj-19,
```

```
counti][1];
```

```
convert(hexValue);
```

```
intT :=  
StrToInt(con)*16;
```

```
hexValue :=  
sGrid.Cells[countj-19,  
counti][2];
```

```
convert(hexValue);
```

```
intT := intT +
```

StrToInt(con);

if intT < 32 then

sGrid.Cells[coun
tj, counti] := '.'

else

sGrid.Cells[coun
tj, counti] := Chr(intT)

end;

end;

gridFocus();

there:

end;

//procedura de
focusare a grilajului

procedure
TForm1.gridFocus();

var

**myRect: TGridRect;
//variabile**

begin

cx := 1;

cy := 0;

```
editing := true;
```

```
myRect.Left := cx;
```

```
myRect.Top := cy;
```

```
myRect.Right := cx;
```

```
myRect.Bottom :=  
cy;
```

```
sGrid.Selection :=  
myRect;
```

```
//pozicionarea  
dreptunghiului
```

```
sGrid.SetFocus;
```

```
end;
```

//procedura de salvare
a continutului editorului
hexa

procedure
TForm1.SaveAs1Click(
Sender: TObject);

```
var i : Integer;  
//variabile  
  
begin  
  
if savedlg.Execute  
then //fereastră de
```

salvare

begin

```
    AssignFile(saveFile, saveDlg.FileName);  
    //asignarea fisierului
```

```
rewrite(saveFile);
```

```
    for i:=1 to  
eeproms[index]*16 do  
    //salvarea fisierului
```

```
write(saveFile,
```

location[i]);

 CloseFile(saveFile
e);
//inchiderea fisierului

end;

```
gridFocus(); //focus
```

```
end;
```

```
//procedura de  
examinare a mutarii  
mouse-ului deasupra
```

grilajului

```
procedure  
TForm1.sGridMouseMoveM  
ove(Sender: TObject;  
Shift: TShiftState; X,
```

Y: Integer);

var i: Integer;

begin

```
if (cx < 17) then  
//afisare adresa
```

```
begin
```

```
i := cx-1 + cy*16;
```

```
        statusBar.simpleText  
:= 'Adresa: ' +  
IntToHex(i , 5);
```

```
end
```

```
else if cx > 19 then
```

begin

$i := cx - 20 + cy * 16;$

statusBar.simplete

```
xt := 'Adresa: ' +  
IntToHex(i , 5);
```

```
end
```

```
else
```


hexa

```
procedure  
TForm1.ClearBuffer1Cl  
ick(Sender: TObject);
```

```
var i: Integer;
```

begin

 setGrid(eeproms[ind
ex]);

 for i := 1 to
eeproms[index]* 16 do

//locatii = 0

location[i] := 0;

end;

//procedura de start a
protocolului i2c-bus

procedure
TForm1.i2c_start;

begin

data1;

clock1;

data0;

clock0;

end;

//procedura de stop a
protocolului i2c-bus

```
procedure  
TForm1.i2c_stop;  
  
begin
```

clock0;

data0;

clock1;

data1;

clock0;

end;

//procedura de
examinare a confirmarii
receptiilor corecte de
memorie

function TForm1.ack:
Integer;

begin

data1;

clock1;

```
form1.inport();
```

```
    from_mem :=  
from_mem AND 64;
```

```
    if from_mem = 64  
then
```

ack := 1

else

ack := 0;

clock0;

end;

//procedura de citire a
portului paralel

```
procedure  
TForm1.inport;
```

```
begin
```

```
asm
```

```
mov dx, 0379h;
```

```
in al, dx;
```

```
mov from_mem, al;
```

end;

end;

//procedura de
intrerupere a
alimentarii montajului

procedure no_alim;

begin

asm

```
mov dx, 0378h;
```

```
mov al, 0;
```

```
out dx, al;
```

end;

end;

//procedura de
activare a interfetei

```
procedure  
TForm1.FormActivate(  
Sender: TObject);
```

```
begin
```

sGrid.SetFocus;

end;

//procedura de citire
din memorie

```
procedure  
TForm1.readButtonClick  
k(Sender: TObject);
```

```
var n, m, k :Integer;  
//variabile
```

i, peer, counti, countj,
intT: Integer;

strT: String;

label no_ack, read_ok;

begin

if not verify then
//mesaj de asteptare

statusBar.SimpleText
:= 'Citire ...';

```
i2c_start();  
//startul citirii
```

```
to_mem := 160;
```

```
byteout();
```

```
if ack=1 then
```

```
    goto no_ack;
```

```
    to_mem := 0;
```

```
//citirea incepand de la  
adresa 0
```

```
byteout();
```

```
if ack=1 then
```

```
goto no_ack;
```

```
if eeproms[index] >  
128 then
```

```
begin
```

```
to_mem := 0;
```

```
byteout();
```

```
if ack=1 then
```

```
goto no_ack;
```

clock0;

end;

data1;

```
i2c_start();
```

```
to_mem := 161;
```

```
byteout();
```

```
if ack=1 then
```

```
goto no_ack;
```

```
clock0;
```

```
for n :=1 to  
eeproms[index]*16 do  
//incarcarea datelor  
citite in editor
```

```
begin
```

if not verify then

location[n] := 0

else

```
location_verify[n]  
:=0;
```

```
k := 128;
```

```
for m := 7 downto  
0 do
```

begin

data1;

clock1;

if ack=1 then

from_mem :=

1;

if not verify then

begin

location[n] :=
location[n] +
k*from_mem

end

else

```
        location_verify[  
n] := location_verify[n]  
+ k*from_mem;  
//verificarea datelor
```

clock0;

k := k shr 1;

end;

```
    if n <
      (eeproms[index]*16)
    then //ACK spre
      memorie
```

```
begin
```

data0;

clock1;

clock0;

end;

end;

data0;

```
    i2c_stop();  
//incheierea ciclului de  
citire
```

```
n := 1;
```

```
sGrid.RowCount :=
```

```
eeproms[index];
```

```
    for counti := 0 to  
eeproms[index] - 1 do  
        //reconstructia  
grilajului
```

begin

for countj := 0 to
16 do

begin

if countj = 0 then

begin

sGrid.Cells[0,
counti] :=
inttohex(counti,4) +

IntToStr(0)

end

else

begin

sGrid.Cells[countj
, counti] :=
inttohex(location[n],2);

hexvalue :=

```
sGrid.Cells[countj,  
counti][1];
```

```
convert(hexvalue)
```

```
;  
;
```

```
intT :=
```

```
StrToInt(con)*16;
```

```
        hexvalue :=  
sGrid.Cells[countj,  
counti][2];
```

```
        convert(hexvalue
```

);

intT := intT +
StrToInt(con);

if intT < 20 then

```
sGrid.Cells[coun  
tj+190, counti] := '.'
```

```
else
```

```
sGrid.Cells[coun  
tj+19, counti] :=
```

Chr(intT);

n:=n+1

end;

end;

end;

goto read_ok; //citire
corecta

no_ack: //eroare de
citire

```
MessageDlg('Eroare  
de citire!',  
mtError,[mbOk], 0);
```

read_ok:

statusBar.SimpleText
:= 'Citire completa!';

gridFocus();

no_alim;

end;

//procedura de
comparare a
continutului memoriei

cu continutul editorului

procedure

TForm1.VerificareClick
(Sender: TObject);

var

i:Integer; //varibile

begin

verify := true;

```
statusbar.SimpleText  
:= 'Verificare ...';  
//mesaj de verificare
```

```
readButtonClick(Send  
er); //apelul  
procedurii de citire a  
memoriei
```

```
for i := 1 to  
eeproms[index]*16 do  
//compararea
```

```
begin
```

```
if location_verify[i]
```

```
<> location[i] then
```

```
begin
```

```
    statusbar.simple  
text := 'Datele nu sunt  
identice!';
```

`verify := false;`

`exit;`

`end;`

end;

statusbar.SimpleText
:= 'Datele sunt
identice!!'; //mesaj

verify := false;

```
gridFocus();
```

```
end;
```

```
//procedura de  
trimitere a unui semnal  
de date pe nivelul '1'
```

logic

procedure
TForm1.data1;

begin

asm

```
mov dx,0378h;
```

```
mov al,6;
```

out dx,al;

end;

delay;

end;

//procedura de
trimitere a unui semnal
de date pe nivelul '0'
logic

```
procedure  
TForm1.data0;
```

```
begin
```

```
asm
```

```
mov dx,0378h;
```

```
mov al,4;
```

```
out dx,al;
```

end;

delay;

end;

//procedura de
trimitere a unui semnal
de ceas pe nivelul '1'
logic

procedure
TForm1.clock1;

begin

asm

mov dx,037Ah;

```
mov al,255;
```

```
out dx,al;
```

```
end;
```

delay;

end;

//procedura de
trimitere a unui semnal
de ceas pe nivelul '0'

logic

procedure
TForm1.clock0;

begin

asm

```
mov dx,037Ah;
```

```
mov al,0;
```

out dx,al;

end;

delay;

end;

//procedura de
intarziere

procedure
TForm1.delay;

```
var del:integer;
```

```
begin
```

```
    for del:=0 to 1000
```

```
do
```

begin

end;

end;

//procedura de
intarziere pe o durata
mai mare

procedure
TForm1.longdelay;

```
var del:integer;
```

```
begin
```

```
    for del:=0 to  
100000 do
```

begin

end;

end;

//procedura de scriere
pe pagina

procedure
TForm1.PageWriteClic
k(Sender: TObject);

```
label no_ack, read_ok;  
    //variabile si  
etichete
```

```
var cont1, cont2, i, j, k,  
trimite, rot,  
page:Integer;
```

begin

for cont1 := 0 to
eeproms[index]-1 do

begin

```
i2c_start(); //start
```

```
to_mem := 160 +  
(cont1 DIV 16)*2;
```

```
byteout();  
//octetul de control
```

```
if ack = 1 then
```

```
goto no_ack;
```

```
to_mem := cont1*16;
```

```
byteout();  
//adresa locatiei
```

```
if ack = 1 then
```

```
goto no_ack;
```

for cont2 :=1 to 16 do

begin

 to_mem :=
location[(cont1*16) +
cont2]; //start de la

location[1]

byteout(); //date

if ack = 1 then

```
goto no_ack;
```

```
end;
```

```
    i2c_stop();  
//terminarea sesiunii de  
scriere
```

```
    longdelay();longdelay  
();longdelay();longdela  
y();longdelay();
```

```
no_alim;
```

```
end;
```

```
goto read_ok;
```

```
no_ack:
```

```
    MessageDlg('Eroare  
de scriere!',  
mtError,[mbOk], 0);
```

```
//mesaj
```

```
read_ok:
```

```
    statusBar.SimpleText  
:= 'Inscriere completa!';  
    //mesaj
```

```
gridFocus();
```

```
no_alim;
```

```
end;
```

//procedura de
trimitere a unui octet
spre memorie

procedure
TForm1.byteout;

```
var trimite,  
rot,k,i,j,page :integer;  
    //variabile
```

```
begin
```

```
rot := 128;
```

```
//10000000
```

```
for k:=7 downto 0 do
```

```
begin
```

```
    trimite := to_mem  
AND rot;
```

```
if trimite = 0 then
```

```
begin
```

```
data0;  
//data = 0 logic
```

```
clock1;  
//clock = 1 logic
```

```
clock0;
```

```
//clock = 0 logic
```

```
end
```

```
else
```

begin

data1;

//data = 1 logic

clock1;

//clock = 1 logic

```
    clock0;  
//clock = 0 logic
```

```
end;
```

```
rot := rot shr 1;  
//rotire spre dreapta
```

```
end;
```

```
end;
```

end.

**Utilizarea
programului**

Programul se lansează în execuție folosind fișierul eeprom.exe și oferă posibilitatea de a citi, a scrie memorii de tip eeprom cu magistrală serială i2c-bus.

Programul suportă formatul binar pentru

importul și exportul de fișiere. Un editor încorporat care permite modificarea datelor direct de la tastatură, duplicarea rapidă a memoriilor sau ștergerea lor.

Meniul program permite încărcarea fișierelor ce urmează să fie înscrise în memorie, salvarea conținutului memoriei ceea ce se poate atât vizualiza cât și modifica în fereastra program.

Această fereastră este un editor hexa și este format din două părți, una în care apar datele în format hexa și una în care apar corespondentele datelor în format ascii. De exemplu

numărul hexa '5A'
reprezintă litera 'Z' în
format ascii.

Programul este
scris în limbajul de
programare Delphi,
iar rutinele de acces

al portului paralel
sunt scrise în
limbajul de
asamblare.

Datele care urmează
să fie introduse în
memorie pot fi

încărcate în editorul hexa, și se pot efectua modificări în conținutul acestuia. Adresele locațiilor de memorie sunt afișate în bara de stare a programului. Orice operație este

urmată de un mesaj, care apare în bara de stare.

După inițierea unei citiri din memorie, pe durata citirii mesajul din bara de stare este 'Citire ...'. În cazul în care

intervine o eroare,
aceasta va fi
semnalat
utilizatorului. Dacă
citirea sa terminat
cu succes apare
mesajul
'Citire completa!'.

La fel se întâmplă
și în cazul scrierii
memoriei.

Interfata între
calculator și memoria
eeprom cu

magistrala i2c-bus