Klausur Assembler am 24. August 2005

(IA 13.0 451 / IA 14.0 451 / IA 15.0 451 / Dauer: 120 Minuten)

Aufgabe

Übersetze aus dem nachfolgenden Pascal-Programm *draw_voxel* die Routine *DrawVoxelPlane* unter Verwendung des integrierten Turbo-Pascal-Assemblers in eine äquivalente (d.h. möglichst bedeutungstreue) 8086-Assembler-Routine.

Zwecks Vereinfachung der Bearbeitung dieser Aufgabenstellung folgt auf das vollständige Pascal-Programm *draw_voxel* ein bereits vorbereitetes „Template" der Routine *DrawVoxelPlane*. Zur besseren Orientierung sind dabei jeweils die Anzahl der Zeilen in der „Musterlösung" mit angegeben. Bitte beachtet, das außerhalb der freigehaltenen Bereiche angegebene Lösungen leider nicht gewertet werden können.

Das PTL-Team wünscht viel Erfolg

```
program draw_voxel;
{$M 16384, 65535, 65535}
Type  TVScreen = Array [0..63999] of Byte;
      PVScreen = ^TVScreen;
Const Scal_Fact : Array[0..6,0..3] of Word =
      ( ( 2, 0,   158, 0  )   ,
        ( 2, 286, 136, 10 )   ,
        ( 2, 667, 106, 20 )   ,
        ( 3, 200, 97,  30 )   ,
        ( 4, 0,   79,  40 )   ,
        ( 5, 333, 59,  50 )   ,
        ( 8, 0,   39,  60 ) ) ;
Const Filename : String =
      'voxel.raw' + #0;
Function MyKeyPressed : boolean; Assembler;
Asm
  mov ax,0100h
  int 16h
  mov ax,0
  jz  @loop1
  mov ax,1
@loop1:
End;
Function GetMyKey : word; Assembler;
Asm
  xor  ax,ax
  int  16h
End;
Procedure ReadLandScape(pntr:PVScreen); Assembler;
Asm
  push ds
  mov  ax,3d02h
  lea  dx,filename +1
  int  21h
  mov  bx,ax
  mov  ax,4200h
  xor  cx,cx
  xor  dx,dx
  int  21h
  mov  ax,3f00h
  mov  cx,64000
  lds  dx,pntr
  int  21h
  mov  ax,3e00h
  int  21h
  pop  ds
End;
Function MyGetMem(size : word) : pointer; Assembler;
Asm
  mov  ax,4800h
  mov  bx,size
  mov  cl,4
  shr  bx,cl
  inc  bx
  int  21h
  xor  dx,dx
  xchg ax,dx
End;
Procedure MyFreeMem(pntr : pointer); Assembler;
Asm
  les  di,pntr
  mov  ax,4900h
  int  21h
End;
Procedure GetVirtualScreen(var myscreen : pvscreen);
Begin
  myscreen := mygetmem (65535);
End;
```

```
Procedure FreeVirtualScreen(myscreen : pvscreen);
Begin
  myfreemem (myscreen);
End;
Procedure ClearVirtualScreen(myscreen : pvscreen; col : byte);
var cnt   :word;
Begin
  for cnt:=0 to 63999 do
      myscreen^[cnt]:= col;
End;
Procedure ShowVirtualScreen(myscreen : pvscreen);
var cnt   :word;
Begin
  for cnt:=0 to 63999 do
      mem[$0a000:cnt]:=myscreen^[cnt];
End;
Procedure SetPaletteEntry(colorno : byte; r,g,b : byte);
Begin
  port[$3c8] := colorno;
  port[$3c9] := r;
  port[$3c9] := g;
  port[$3c9] := b;
End;
Procedure WaitRetrace;
Begin
  repeat until Port[$03da] and 8 = 0;
  repeat until Port[$03da] and 8 = 8;
End;
Procedure DrawRectangle(x,y,width,height:word;color:byte;myscreen:pvscreen);
var cnt1  :word;
    cnt2  :word;
begin
  for cnt1:= 1 to height do
    for cnt2:= 1 to width do
        myscreen^[(y+cnt1-1)*320+(x+cnt2-1)]:=color;
end;
Procedure DrawVoxelPlane(src,dst:pvscreen;index:integer);
var cnt1  :word;
    cnt2  :word;
    cnt3  :word;
    offs  :word;
    inc1  :word;
    inc2  :boolean;
    width :word;
    origin:word;
begin
  for cnt1:=0 to 6 do
  begin
    offs:=index*160;
    offs:=offs+scal_fact[cnt1,3];
    cnt3:=0;
    inc1:=scal_fact[cnt1,1];
    inc2:=false;
    for cnt2:=0 to scal_fact[cnt1,2] do
    begin
      origin:=src^[offs+cnt2];
      width:=scal_fact[cnt1,0];
      if inc2 then inc(width);
      drawrectangle(cnt3,199-origin shr 1 div (cnt1+1),width,
                    origin shr 1 div (cnt1+1),origin,dst);
      cnt3:=cnt3+scal_fact[cnt1,0];
      if inc2 then
      begin
        inc2:=false;
        inc(cnt3);
      end;
      inc1:=inc1+scal_fact[cnt1,1];
      if inc1>=1000 then
      begin
        inc1:=scal_fact[cnt1,1];
        inc2:=true;
      end;
    end;
    dec (index);
    if index<0 then index:=399;
  end;
end;
var landscape : pvscreen;
    vscreen   : pvscreen;
    origin    : integer;
    mykey     : word;
begin
  asm
    mov ax,13h
    int 10h
  end;
  getvirtualscreen(vscreen);
  getvirtualscreen(landscape);
  for origin:=000 to 127 do
      setpaletteentry(origin,origin shr 1,origin shr 2,origin shr 3);
  for origin:=128 to 255 do
      setpaletteentry(origin,(255-origin) shr 1,origin shr 2,origin shr 3);
  clearvirtualscreen(vscreen,0);
  readlandscape(landscape);
  origin:=0;
  repeat
    drawvoxelplane(landscape,vscreen,origin);
    waitretrace;
    showvirtualscreen(vscreen);
    clearvirtualscreen(vscreen,0);
    if mykeypressed then mykey:=getmykey;
    if mykey shr 8 = 72 then
    begin
      inc (origin);
      if origin=400 then origin:=0;
      mykey:=0;
    end;
    if mykey shr 8 = 80 then
    begin
      dec (origin);
      if origin<0 then origin:=399;
      mykey:=0;
    end;
  until (mykey and $7F=27);
  asm
    mov  ax,02h
    int  10h
  end;
  freevirtualscreen(vscreen);
  freevirtualscreen(landscape);
end.
```

```
Procedure DrawVoxelPlane(src,dst:pvscreen;index:integer); Assembler;

{ src   -> [BP+12] "Segment"
  src   -> [BP+10] "Offset"
  dst   -> [BP+8]  "Segment"
  dst   -> [BP+6]  "Offset"
  index -> [BP+4]}

Asm

{ Ausser den lokalen Variablen ist der Stackrahmen bereits eingerichtet ! }

{var cnt1  :word;    -> [BP-2]
     cnt2  :word;    -> [BP-4]
     cnt3  :word;    -> [BP-6]
     offs  :word;    -> [BP-8]
     inc1  :word;    -> [BP-10]
     inc2  :boolean; -> [BP-11]
     width :word;    -> [BP-13]
     origin:word;    -> [BP-15]}

{ ### 1 Zeile ### }



{ begin}

{  for cnt1:=0 to 6 do}

{ ### 4 Zeilen ### }








{  begin}

{    offs:=index*160;}

{ ### 4 Zeilen ### }








{    offs:=offs+scal_fact[cnt1,3];}

{ ### 9 Zeilen ### }
{    cnt3:=0;}
{ ### 1 Zeile ### }

{    inc1:=scal_fact[cnt1,1];}
{ ### 8 Zeilen ### }
```

```
{    inc2:=false;}
{ ### 1 Zeile ### }

{    for cnt2:=0 to scal_fact[cnt1,2] do}
{ ### 11 Zeilen ### }
```

```
{    begin}
{      origin:=src^[offs+cnt2];}

{ ### 7 Zeilen ### }
```

```
{      width:=scal_fact[cnt1,0];}

{ ### 7 Zeilen ### }
```

```
{      if inc2 then inc(width);}

{ ### 4 Zeilen ### }
```

```
{      drawrectangle(cnt3,199-origin shr 1 div (cnt1+1),width,
                     origin shr 1 div (cnt1+1),origin,dst);}

{ ### 16 Zeilen ### }
{      cnt3:=cnt3+scal_fact[cnt1,0];}
{ ### 8 Zeilen ### }
```

```
{      if inc2 then}
{ ### 2 Zeilen ### }
```

```
{      begin}
{        inc2:=false;}
{ ### 1 Zeile ### }
```

```
{        inc(cnt3);}
{ ### 1 Zeile ### }
```

```
{      end;}
{ ### 1 Zeile ### }
{      inc1:=inc1+scal_fact[cnt1,1];}
{ ### 9 Zeilen ### }
```

```
{      if inc1>=1000 then}
{ ### 2 Zeilen ### }
```

```
{      begin}
{        inc1:=scal_fact[cnt1,1];}
{ ### 8 Zeilen ### }
```

```
{        inc2:=true;}
{ ### 1 Zeile ### }
```

```
{      end;}
{ ### 1 Zeile ### }
{    end;}
{ ### 3 Zeilen ### }




{    dec (index);}
{ ### 1 Zeile ### }

{    if index<0 then index:=399;}
{ ### 4 Zeilen ### }





{  end;}
{ ### 4 Zeilen ### }





end;
```