ANNEXE A : Source des programmes

Cette partie réunit tous les sources nécessaires à la constitution du fichier exécutable qui fait l'objet de ce mémoire. L'environnement est MSDOS en mode ligne de commande et il faut disposer de NASM (disponible librement sous licence GPL) qui est le seul programme externe. Le tout peut tenir sur une simple disquette.

Pour assembler le programe, il faut effectuer 3 étapes qui peuvent être réunies dans un seul fichier batch, souvent appelé `M.BAT` :

```
labels3 sp65.asm
nasm -o v.exe -l v.lst out.asm
resize v.exe
rem Pour lancer le programme si l'assemblage a réussi :
v
```

- `LABELS3` effectue un prétraitement en générant des tables de labels et en éliminant les commentaires (pour accélérer l'assemblage). Il faut cependant faire attention à respecter les numéros de lignes, sinon les messages d'erreur de NASM seront difficiles à comprendre.
- `NASM` est l'assembleur. Il transforme le source (modifié par `LABELS3`) en un binaire brut. Le source est cependant structuré pour correspondre à un format reconnaissable par MS−DOS.
- `RESIZE` "patche" le binaire en changeant une valeur de l'entête du fichier afin que MS−DOS reconnaisse le fichier correctement. D'habitude cela était effectué à la main mais l'automatisation a permis de ne plus penser à ce détail de bas niveau lorsque le développement est devenu plus complexe.

A.1 : fichier RESIZE.PAS :

```
{ patche l'en-tête d'un fichier au format EXE }

program resize;
uses crt,dos;

var f: file;
    n: word;
    dumb: word;

begin
  assign(f,paramstr(1));
  reset(f,1);
  n:=FileSize(f);
  n:=(n+511) shr 9; { division par 512}
  seek(f,4);
  BlockWrite(f,n,2,dumb);
  close(f);
end.
```

A.2 : fichier LABELS3.PAS :

```
{$G+}{$Q+}{$R+}{$S-}{$M 2048,0,0}

{
 functions:
  - remove the comments (->reduces the assembling time)
  - replaces all the ^$xxx__ by ^_xxxnn
  - creates an include file
 beware: this naughty hack is a bag of bugz !
}

program labels;

var file_inc:text;
    file_in,file_asm:file;
    str:string[20];
    c,d,e:char;
    i,j,l,taille:word;
    nb_names:word;
    names: array[0..10] of string[20];
    occur1:array[0..10] of byte;
    occur2:array[0..10] of byte;
    t:array[0..32767]of char;
    u:array[0..16384]of char;

function streq(s1,s2:string):boolean;
 var i:byte;
 begin
  i:=0;
  repeat
   if s1[i]<>s2[i] then
    begin
     streq:=false;
     exit;
    end;
   inc(i);
  until i>byte(s1[0]);
  streq:=true;
 end;

function new_char: char;
 begin
  inc(l);
  if l >= taille then
   begin
    blockread(file_in,t,32768,taille);
    l:=0;
   end;
  new_char:=t[l];
 end;

procedure write_char(c:char);
 label wb;
 begin

  u[i]:=c;
  inc(i);
  if i >= 16384 then
   begin
    blockwrite(file_asm,u,16384);
    i:=0;
```

```
   end;
wb:
  e:=d;
  d:=c;
 end;

label ok;

begin
 nb_names:=0;
 assign(file_in,paramstr(1));
 assign(file_asm,'out.asm');
 reset(file_in,1);
 rewrite(file_asm,1);
 d:=' ';
 e:=' ';
 l:=0;
 i:=0;

 repeat
  c:=new_char;

  {removes the comments}
  if c=';' then
    repeat
      c:=new_char;
    until (c=char($D)) or (taille=0)
  else

  if c='$' then
    if (d=char($A)) or (d=char($D)) then
      begin
{remplace le $ par un _ (plus sympathique pour l'assembleur)}
        str:='';
        c:='_';
{constitue la chaine et cherche '__' }
        repeat
         str:=str+c;
         write_char(c);
         c:=new_char;
        until c='_';
{rechercher le label dans la table}
        j:=0;
        while j<nb_names do
         begin
{comparer la ligne et le label:}
          if streq(names[j],str) then
           begin
            occur1[j]:=occur1[j]+1;
            if occur1[j]>9 then
              begin
                occur1[j]:=0;
                occur2[j]:=occur2[j]+1;
                if occur2[j]>9 then
                  begin
                    writeln('trop de labels');
                    halt(255);
                  end;
              end;
            goto ok;
           end;
         j:=j+1;
         end;

{label non trouvé: créer le label}
        {1) copier le nom}
        names[nb_names]:=str;
        {compteurs:}
        occur1[nb_names]:=0;
        occur2[nb_names]:=0;
        j:=nb_names;
        nb_names:=nb_names+1;
        if nb_names>10 then
        begin
          writeln('trop de labels');
          halt(255);
        end;

ok:
        write_char(char(occur2[j]+ord('0')));
        if new_char<>'_' then
        begin
          writeln('mauvais format: '+str);
          halt(255);
        end;
        c:=char(occur1[j]+ord('0'));

   {     writeln(str); }
      end;
  write_char(c);
```

```
 until taille=0;
 blockwrite(file_asm,u,i-1);
 close(file_in);
 close(file_asm);


{ecrit le fichier de symboles:}
 assign(file_inc,'out.inc');
 rewrite(file_inc);
 if nb_names<>0 then
  begin
  j:=0;
  repeat
    l:=occur1[j]+(occur2[j]*10);
    i:=0;
    writeln (file_inc,'start'+names[j]+':');
    while (i<=l) do
    begin
      if (i and 7)=0 then
        write(file_inc,'  dw ');
      write(file_inc,names[j]);
      write(file_inc,i div 10);
      write(file_inc,i mod 10);
      if ((i and 7)=7) or (i=l) then
        writeln(file_inc,'')
      else
        write(file_inc,', ');
      i:=i+1;
    end;
    j:=j+1;
  until j=nb_names;
 end;
 close(file_inc);
end.
```

A.3 : fichier SP73.ASM :

```
;*****************************************************************
;*                                                               *
;* Build with:                                                   *
;*   labels2 sp73.asm                                            *
;*   nasm -o v.exe out.asm                                       *
;*   resize v.exe                                                *
;* Then type:                                                    *
;*   v                                                           *
;* to run                                                        *
;*                                                               *
;*****************************************************************

; Description : real-time, interactive CFD program on a desktop PC.
;
; This program uses the FHP3 model with optimized algorithms.
; Entirely written in assembly langage, it provides the maximum
; performance available on a MMX-capable PC.
; Not only it is fast but it is also interactive and intuitive
; with a simple mouse-driven GUI and high display speed.
; It runs under raw real mode MSDOS though and is M$-Windows unfriendly.
;
;
; Being limited to 64MB of RAM by HIMEM.SYS, it can simulate
; flows up to 8000 (sqrt(64M)) * 2 (ideal FHP3 Re/cell) * 0.3 (maximum Mach)
; => Re=5000. Good knowledge of the FHP model and conventional
; Fluid Dynamics is necessary to obtain correct results (it's not a toy).
; The Galilean invariance problem is not treated here. You've been warned !
;
;
; This program has been written for a Master of MicroElectronics-
; MicroInformatics at the University Paris 8 - Saint Denis during
; the years 1998-2000 by Yann Guidon (whygee@f-cpu.org).
; The master book will be freely downloadable from the web site.
; This book can be considered as a manual for the making of the
; program, its theory and use.
;
;                       -*-*-*-*-*-*-
;
; Warnings :
;
;   This program is placed under the GNU Public Licence (GPL).
; It is therefore free for any use, provided that the source
; remains in the package, and that the package remains freely
; available (without paying) to anyone anywhere.
; Look at your Linux distribution or www.gnu.org for further
; details on the GPL.
;
;   This program is provided "as is" with absolutely
; no garantee of ANY kind, not even that it will work or
; fit to anyone's needs. You have to fully read the source code
; and its comments as well as well as all the provided files before
; attempting to execute the program. The author can not be held
; responsible for any damage caused by the program or its use.
; You're a grown up and educated boy, right ? So don't come cry
; on my shoulders, you've been warned.
;
; This is the usual message at the beginning of any GNUed file:
;
;*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*
;
; This program is free software; you can redistribute it and/or
; modify it under the terms of the GNU General Public License
; as published by the Free Software Foundation; either version 2
; of the License, or (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
; GNU General Public License for more details.
; You should have received a copy of the GNU General Public License
; in the file GNU_General_Public_License, along with this program;
; if not, write to the Free Software Foundation, Inc.,
; 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
;
;*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*
;                       -*-*-*-*-*-*-
;
;
; Program requirements :
; Hardware :
; - PC with at least a Pentium MMX processor
;     (tested successfully on Intel Pentium MMX,
;     Intel Pentium II-266 and Cyrix/IBM 200MHz (true clock) CPU)
; - dual CPU support soon to be supported (the stub
;     is ready but not the algorithm)
; - At least 4MB of RAM
; - a serial mouse connected to COM1 (IRQ 4 / base=03F8h)
; - a VESA2-compatible video adapter with 1024*768*256
;     color capability (Use UNIVBE 5.1 if the card is
;     not compatible with the VESA 2 standard)
; - at least several MB of hard disk space if you want to save pictures.
;
;
; Software :
; - must run in raw MSDOS mode (not in a Windows' DOS box
;     or under Windows NT). The easiest way is to exit Windows 95/98
;     and click on "exit to MSDOS". Or add "command.com" at the end
;     of your autoexec.bat file.
; - No protected mode manager must be running. usually, there is
;     EMM386 or QEMM386. simply comment them out in your autoexec.bat
;     or config.sys file.
; - HIMEM.SYS must be installed in the config.sys file as to use
;     the high memory.
; - the mouse driver (recent one preferred) must be installed.
;
; You can boot on a diskette if you prefer, or use MSDOS's
; multiple configuration capability. FreeDos has not been tested yet.
;
; The program will halt and display an error message if a problem
; arises. If no message is displayed, or the computer is frozen
; for a long time, then reset your computer. The program needs
; several seconds though to initialize the mouse and the screen,
; and one screenshot can require twenty seconds to complete,
; so don't panic.
;
;                       -*-*-*-*-*-*-
;
;
; The keyboard :
;  (each key corresponds to one "button" on the screen)
;
;
; "Escape" : Quit
;  (i don't use the Q key because its place changes from
;    one keyboard to another [AZERTY/QWERTY...])
;  Press Escape to exit the program. If in running mode,
;    it will complete the current pass and exit. it will display
;    a sandpile during completion.
;  If an immediate exit is needed, for example during a long
;    computation, press Escape three times, and the program will
;    exit "cleanly" to MSDOS. It is provided as a way to always
;    keep the computer under control if the program ever behaves badly.
;
;
; "P" : Play/pause
;  Switch between run and stop states. the display is updated with the
;  symbol corresponding to the current activity:
;    - a "PLAY" symbol appears if the program is paused
;    - a "PAUSE" symbol indicates that the program computes
;    - a "WAIT" (sandpile) symbol indicates that the program completes
;      the current computation pass before pausing.
;
;
; "S" : Step by step mode
;  Each time "S" is pressed, the program computes one pass.
;  To compute slowly, keep the "S" key pressed, and release it to stop.
;
;
; "R" : Reset the computation
;  Stop the computation and reset the settings and the cycle counter.
```

```
;                               -*-*-*-

; The following keys are active only in "pause" state, when the "play"
; symbol is displayed:
;
;
; Direction arrows (left, right, up, down):
;  scrolls the display (if possible)
;
;
; Page up / Page down:
;  Zoom in / Zoom out
;
;
; "+"/"-" on the numeric pad:
;  increase/decrease the strip count (the compute window's size)
;
;
; "C": Calibration
;  Runs the computations for all window sizes, and finds which size
;  gives the best performance. This changes the strip count to the
;  best size, depending on only one parameter: speed. This may be
;  influenced by any BIOS setting or keypress. The mouse is disabled
;  during calibration but the keyboard can still "emergency" exit
;  the program.
;
;
;                               -*-*-*-
;
;
; Two functions can only be accessed through the mouse in "pause" mode.
; The picture backup (snapshot") represented by a tiny camera button,
; ensures that the mouse cursor is NOT on the computation surface, so
; the pictures are "clean". The "screenshots" last about twenty
; seconds, and generate uncompressed BMP files from the display (700KB).
;
;
; The other function is the tunnel resizing. By clicking with the
; left button on the computation surface, one can change the size
; of the tunnel. If the tunnel is too large, the program simply
; exits with an error message.

;*****************************************************************
;
;
;
;
;
;
;                                                       the code.
;
;*****************************************************************


; to debug the AP and to know why it hangs:
;%define short_IDT
; comment this line out to enable the full AP IDT,
; 450 byte are then added to the binary.

;*****************************************************************
;* Some definitions:                                             *
;*****************************************************************

; For the IDT and the GDT descriptors:
ID_CODE     equ 10011010b
ID_DATA     equ 10010010b
CODEmask    equ 00001000b
CPL         equ 01100000b
DEFAULT_32  equ 01000000b
BIG         equ 10000000b

; for the APIC:
APIC_BASE   equ 0FEE00000h
APIC_ID     equ APIC_BASE + 020h
APIC_VER    equ APIC_BASE + 030h
APIC_SVR    equ APIC_BASE + 0F0h
APIC_ICR_LO equ APIC_BASE + 0300h
APIC_TIMER  equ APIC_BASE + 0390h

; For the serial mouse:
COM1INTR    equ 0Ch
COM1PORT    equ 03F8h

; colors:
NUMBER_BG   equ 252 ; blue
NUMBER_FG   equ 253 ; green
MOUSE_COLOR equ 254 ; white
BACKGROUND  equ 255 ; black

; video:  the code depends on XRESOLUTION=1024 !
; (multiplying by 1024 is shifting left by 10, which is much faster !)
XRESOLUTION equ 1024
YRESOLUTION equ 768
VIDEOMODE   equ 105h ; 1024*768*256 LFB

; minimum XMS memory size:
size_block  equ 3000000 ; "3 MB should be enough for any scientist" :-P
size_kb     equ (size_block+1023)>>10

; modifier block size:
Mblocks equ 5000
MBsize equ 16*Mblocks

; structure of a cell:
C equ  0*8;  F     E
B equ  1*8;   ^   ^
D equ  2*8;    \ /
G equ  3*8; A<--G-->D
A equ 13*8;    / \
E equ 14*8;   v   v
F equ 15*8;  B     C
VB0 equ  4*8;
VB1 equ  5*8;
VB2 equ  6*8;
VB3 equ  7*8; "VB"="Video Biffer"
VB4 equ  8*8;
VB5 equ  9*8;
VB6 equ 10*8;
VB7 equ 11*8;
WALL equ 12*8; pointer to the modification list
CEL equ 16*8 ; size of a cell

; structure of the vertical, 2D buffer:
B2 equ 0*8
C2 equ 1*8
CEL2 equ 2*8



;*****************************************************************
;*    Memory layout at run time:
;*
;* -16..-1: MCB (Memory control block)
;*     byte[-16]=4Dh if not last block else =5Ah
;*     word[-13]=size (in paragraphs) of the block
;* 0..255: PSP (Program Segment Prefix), pointed to by DS and ES
;*     word[0]=21CDh="INT 20h" (program termination)
;*     128..255: Program command line parameters,
;*       byte[128]=number of characters
;*       first char at byte[129] (usually a space)
;* 100h: program starting point in real mode, RM setup, PM setup,
;*     IRQ and trap handlers, fonts, initialized data
;* endfile: unitialised data,AP stack, AP jump zone (5KB), MP stack
;*   BMP buffer (1K), modifier buffer (16K)
;*****************************************************************

;*****************************************************************
;*                         EXE HEADER                            *
;* this EXE header creates .COM-like programs: IP=100h, CS=DS=SS    *
;* but the stack and the memory are set by DOS, so we spare system  *
;* calls, tests and the associated error messages :-)               *
;*****************************************************************

header_size equ 32 ; this is the simplest case
header_para equ header_size >> 4
add_mem equ 8192+MBsize ; required additional memory
 ; VBE blocks+AP stack+scratch+MP stack+BMP+modifiers
db 4Dh,5Ah      ; MZ signature
dw 0            ; length mod 512 (can be 0)
dw 0            ; length div 512 + 1, may cause an error
; when the program's size increases
dw 0            ; no item to relocate (too complex and unuseful here)
dw header_para  ; Header size in paragraphs
dw add_mem>>4   ; Minimum extra memory (paragraphs) to allocate
; (must be non zero for CS=DS=ES=SS)
dw add_mem>>4   ; Maximum extra memory (paragraphs) to allocate
dw -10h         ; Initial SS (before fixup)
dw MP_stack_top ; Initial SP
dw 0            ; Checksum (=0->none)
dw 100h         ; Initial IP \   (Executes the code just after the header)
dw -10h         ; Initial CS /(before fixup)
dw 0            ; offset of the relocation table in the file
dw 0            ; (no overlay)
;pad here to align to para. boundary (was: "first relocation item")
TIMES 4 db 0
ORG 0E0h ; to be reduced as the header grows
; and now, we feel the same as in a .COM file :-D
```

```
;*****************************************************************
;*  Make sure the label tables are generated in the correct order   *
;*****************************************************************

$base__  equ 0FCh ; in the end of the PSP, there is no harm
$base1__ equ 0FCh ; $base1__ because there is not enough $base__
$base2__ equ 0FCh ; $base2__ because there is not enough $base1__
$base3__ equ 0FCh ; etc ...
$base4__ equ 0FCh ; etc ...
$base5__ equ 0FCh ; etc ...
$video__ equ 0FCh
$mem__   equ 0FCh
; and now we can use these labels safely.

;*****************************************************************
;*          Some data overwriting the already executed code         *
;*****************************************************************

DOS_IDT    equ 100h ; 6 bytes (PWORD)
;AP_setup   equ 106h ; 1 byte -> moved to a SMC
old_mask   equ 107h ; 1 byte
SEMAPHORE  equ 108h ; 1 bytes -> this one is used very often !
; it should not be on the same cache line as another variable
; to prevent bus overuse by the MESI system.

;*****************************************************************
;*                                                               *
;*         16B CODE SECTION (Hardware setup in real mode)          *
;*                                                               *
;* This part is executed only once, speed is not required but code  *
;* size is a important. And since it is executed only once, some    *
;* parts may be overwritten by data.                             *
;*                                                               *
;*****************************************************************


;*************
;* DETECTION * before we do anything
;*************

;-----------------------------
; Pentium+ specific features:
;-----------------------------
; Never tested on NON-INTEL CPUs !

 xor eax,eax ; will hang the computer if <386
 inc al ; eax=1
 cpuid ; CPUID with EAX=1 -> get processor characteristics

 test dh,10b ; Local APIC -> dual P6+ detection
 setnz [AP_setup] ; overwrites already executed code

; the multiproc code can be disabled from the command line:
 cmp byte [80h],0
 je no_SP_bypass
   cmp byte [82h],'s'
   jne no_SP_bypass
     mov byte [AP_setup],0 ; disables the biprocessor code
no_SP_bypass:

 shr edx,24  ; put bit 23 in the carry flag
 mov dx,MSG_MMX
 jnc the_end ; exit if MMX is not supported

;---------------------------
; checking if in real mode:
;---------------------------
 mov dx,MSG_PROT_MODE
 mov eax,cr0
 shr al,1
 jc the_end

;-------
; MOUSE:
;-------
 xor ax,ax ; detect the mouse driver
 int 33h
 mov dx,MSG_NO_MOUSE
 inc ax ; AX=FFFFh+1=0!
 jnz the_end ; we could branch to a "custom" routine
 ; for a mouse driver but it's not the purpose of this program

;---------------------------
; gets the mouse parameters:
;---------------------------
 mov ax,24h
 int 33h
 mov dx,MSG_NO_MOUSE
 inc ax
 je the_end
 cmp ch,2     ; check the mouse type
 jne the_end
 mov dx,MSG_WRONG_MOUSE_PORT
 cmp cl,2     ; check the port (should be COM1)
 jb the_end

;------------------
; VBE 2 LFB setup :
;------------------

; 1- detection
 mov ax,4F00h
 mov di,VbeInfoBlock
 int 10h
 mov dx,MSG_NO_VESA
 cmp ax,byte 004Fh
 jne the_end
 cmp dword [VbeSignature],'VESA' ; check the signature
 je not_the_end0
the_end:
 jmp __the_end ; at the end of the code
not_the_end0:

; 2- verification
 mov ax,4F01h ; Return VBE mode information
 mov cx,VIDEOMODE  ; Mode number
 mov di,ModeInfoBlock ; ES:DI = Pointer to ModeInfoBlock structure
 int 10h ; this is important because we get the linear framebuffer address
 cmp ax,byte 004Fh
 jne the_end
 mov ax,[ModeAttributes]
 and ax,10100001b
 cmp ax,10000001b ; D0=1(available) + D5=0(VGA) + D7=1(LFB)
 jne the_end

;-----
; XMS:
;-----
 mov ax,4300h
 int 2Fh ; XMS detection
 mov dx,MSG_NO_XMS
 cmp al,80h
 jne the_end
 mov al,10h
 int 2Fh ; get XMSpointer
 mov [XMSpointer+2],es
 mov [XMSpointer],bx   ; Self Modifying Code (SMC)
 mov ah,8
 call far [XMSpointer] ; get the amount of free XMS
 mov [XMS_SIZE],ax ; SMC
 mov dx,MSG_NO_XMS
 cmp ax,size_kb
 jb the_end


;*****************************************************************
;*  now there's no error message, we can prepare                    *
;* the program (video+memory) for the mode switch                   *
;*****************************************************************

;---------------------------------------
; Sets video mode to 1024*768*256*LFB :
;---------------------------------------
 mov ax,4F02h
 mov bx,VIDEOMODE+4000h
 int 10h
; reset the text mode for the next message
 mov byte [jmpshortlabel - 1],0  ; SMC

; modify the color palette: (normal VGA code)
 mov al,256-(size_palette/3)
 mov si,color_palette
 mov cx,size_palette
 mov dx,3C8h
 out dx,al
 inc dl
 rep outsb


;-------------------------
; Allocates the XMS block:
;-------------------------
 mov dx,0 ; SMC
XMS_SIZE equ $-2
 mov ah,9
 db 09Ah ; opcode for "FAR CALL [immediate]"
XMSpointer: dd 0 ;get the XMS block
 mov [XMShandle],dx ; SMC
```

```
 mov ah,0Ch
 call far [XMSpointer] ;lock the block

 push dx ; linear address in dx:ax
 push bx ; popped later


;----------------------
; Sets the descriptors:
;----------------------
 db 66h      ; NASM doesn't like "mov eax,cs"
 mov ax,cs   ; that clears the high part of EAX.
 mov [cs2],ax ; SMC, could also be done by the EXE header relocation
 mov [cs3],ax
;;; mov [reloc_seg],ax ; provided by the EXE header
 shl eax,4 ;base address in bytes

; relocates to the "absolute addresses"
 mov di,start_base
loop_link:
   mov si,[di]
   add di,byte 2
   add [si],eax
   cmp di,start_video
 jne loop_link

; relocation of the video addresses :
 mov eax,[PhysBasePtr]
loop_link3:
   mov si,[di]
   add di,byte 2
   add [si],eax
   cmp di,start_mem
 jne loop_link3

 pop eax ; we assume here that HIMEM aligns the block to a 1K boundary
 add eax,1024
 and eax,-1024
loop_link2:
   mov si,[di]
   add di,byte 2
   add [si],eax
   cmp di,end_base
 jne loop_link2


;Enable the A20 line
 mov ah,03h
 call far [XMSpointer]

;disable the interrupts
 cli
 mov al,80h
 out 70h,al ;set the NMI mask
 in al,21h ; backup of the PIC mask
 mov [old_mask],al
 mov al,11101101b ; mouse+kbd
 out 21h,al ;enable the COM1 + KBD interrupts in the main PIC

 sidt [DOS_IDT]  ; save the real mode IDT
 lidt [NEW_SIZE] ; then load ours

; will be popped later by RETFD at the end of the mode change:
 push word SEL_CODE_32
 push word code_32


switch_PM32: ; entry point for the AP to switch to PMODE
 lgdt [NEW_SIZE] ; our descriptor table
 push dword 11000000000010b ;sets the correct bits in the flag register
 popfd

;SWITCHES INTO PROTECTED MODE
 mov eax,cr0
 or al,1  ; sets the PE bit: "Protection Enabled"
 and eax,~(3<<29) ; reset CD and NW to enable the caches (->bug!)
 mov cr0,eax
 jmp SEL_CODE_16:here
here:

; fixes the segments to the right selectors
 mov ax,SEL_REAL
 mov ss,ax
 mov es,ax
 mov ax,SEL_FLAT
 mov ds,ax

;"calls" the 32b code
  retf  ; FAR return changes CS (the segment register)
```

```
;*****************************************************************


;*****************************************************************

; The 32b code returns here (if something wrong happens) :

FatalErrorProc:
 mov dx,MSG_FATAL_ERROR ; nothing more to say...

ProtError:
Back_2_16b:
 cli
; EOI (if an interrupt crashed the code)
 mov al,20h ;code: End Of Interrupt
 out 20h,al ;free the interrupt chanel
 emms ; clears the MMX registers
; restores the cached part(s):
 mov ax,SEL_REAL
 mov ds,ax
 mov sp,MP_stack_top

;SWITCHES BACK TO REAL MODE
 mov eax,cr0
 and al,0FEh ; reset the PE bit
 mov cr0,eax

 db 0EAh     ;opcode for a FAR JMP to the 16b code
 dw back_2_real
cs2: dw 0 ;will have been set by a previous instruction
back_2_real:

; Here we are in Real Mode:
 mov ax,cs
 mov ds,ax
 mov ss,ax
 mov es,ax

; restores the DOS environment

 lidt [DOS_IDT] ;restore the DOS' IDT
 xor al,al
 out 70h,al ;NMI mask removed
 mov al,[old_mask]
 out 21h,al ; restores the PIC state
 sti
 push dx

;************************
;* FREE THE RESSOURCES: *
;************************

;--------------
; free the XMS:
;--------------
 mov ah,4
 call far [XMSpointer] ;disable the A20 line
 mov ah,0Dh
 db 0BAh ;"mov dx,imm16" ; SMC
XMShandle: dw 0
 call far [XMSpointer] ;unlock the XMS block
 mov ah,0Ah
 call far [XMSpointer] ;unallocate the XMS block

; text mode:
 jmp short no_text
jmpshortlabel: ; SMC
   mov ax,3
   int 10h
no_text:

; error message ?
 pop dx
 test dx,dx
 jz no_message
;---------
; THE END:
;---------
__the_end:
   mov ah,9 ;dx=message
   int 21h
no_message:
;exits:
 mov ax,4C00h
 int 21h ; Return to MSDOS



;*****************************************************************
```

```
;*                        MP TRAP HANDLER                          *
;*****************************************************************

ProtectionErrorProc: ; dumps the processor state (useful when debugging)
 pop eax ; get first (which ?) element from top of the stack
 mov di,MSG_END-2 ; end of the ASCII number
 call display_number_routine
 pop eax
 mov di,MSG_END2-2
 call display_number_routine
 pop eax
 mov di,MSG_END3-2
 call display_number_routine
 mov dx,MSG_PROT_ERROR ; error message
 jmp short Back_2_16b ; clean exit

display_number_routine:
number_loop:
   mov bl,al
   and bl,0Fh
   cmp bl,9
   jbe no_add
     add bl,'A'-('9'+1)
no_add:
   add [es:di],bl
   dec di
   shr eax,4
 jnz number_loop
 ret


;*****************************************************************
;*       DATA that should NOT be overwritten: (starting here)    *
;*****************************************************************

; RUN time error messages:
MSG_SIZE:        db "Array too huge$"
MSG_FATAL_ERROR: db "Fatal error$"
MSG_FATAL_AP:    db "AP fatal error$"
MSG_PROT_ERROR:  db "Protection error:",
                 db 0Dh,0Ah,"Code: 00000000h"
MSG_END:         db 0Dh,0Ah,"CS:   00000000h"
MSG_END2:        db 0Dh,0Ah,"EIP:  00000000h"
MSG_END3:        db '$'
MSG_CELLS:       db "too many cells$"

;%ifndef short_IDT
;AP_DIV_0: db "AP div 0$"
;AP_DEBUG: db "AP debug$"
;AP_NMI:   db "AP NMI$"
;AP_BREAK: db "AP breakpoint$"
;AP_OV:    db "AP Overflow$"
;AP_BOUND: db "AP Bound$"
;AP_OPCODE: db "AP Invalid opcode$"
;AP_FPU:   db "no AP FPU$"
;AP_COPRO: db "AP coproc. segment overrun$"
;AP_TSS:   db "invalid AP TSS$"
;AP_SEGM:  db "AP segment not present$"
;AP_STACK: db "AP stack segment fault$"
;AP_GPF:   db "AP GPF$"
;AP_PAGE:  db "AP Page fault$"
;AP_RESER: db "AP reserved$"
;AP_FPUER: db "AP FPU error$"
;AP_ALIGN: db "AP Alignment error$"
;%endif



;*****************************************************************
;*                         MAIN IDT-GDT:                         *
;*****************************************************************


;The Global Descriptor Table is interleaved with the
;Interrupt Descriptor Table to save space.
;Entries 0 to 6 are not used.

align 8
 dw 0
NEW_SIZE:     dw New_limit ; Size of the IDT/GDT
$base__:      dw GDT,0 ;base adress
GDT           equ $ - (7*8)

;INT 7: segment descriptor
DESC_REAL:    dw 0FFFFh
$base__:      db 0,0,0,ID_DATA, 0,0
 SEL_REAL     equu DESC_REAL - GDT ;alias for the real mode return

;INT 8: Fatal error handler
ERR_VECT:     dw FatalErrorProc,SEL_CODE_16,8600h,0

;INT 9: Keyboard interrupt IDT descriptor
KBD_VECT:     dw KbdHandler,SEL_CODE_16,8600h,0

;INT 10: Flat segment descriptor
DESC_FLAT:    dw 0FFFFh,0
              db 0,ID_DATA,DEFAULT_32+BIG+0Fh,0
 SEL_FLAT     equ DESC_FLAT - GDT

;INT 11: 32b code segment descriptor
DESC_CODE_32: dw 0FFFFh
$base__:      db 0,0,0,ID_CODE,DEFAULT_32,0
 SEL_CODE_32  equ DESC_CODE_32 - GDT  ;the aimed 32b. code segment

;INT 12: Serial port, COM1 IDT descriptor
COM1_VECT:    dw MouseHandler,SEL_CODE_16,8600h,0

;INT 13: Protection error IDT descriptor
PROT_VECT:    dw ProtectionErrorProc,SEL_CODE_16,8600h,0

;INT 14:
DESC_CODE_16: dw 0FFFFh
$base__:      db 0,0,0,ID_CODE, 0,0
 SEL_CODE_16  equ DESC_CODE_16 - GDT ;the original 16b. code segment

New_limit     equ $-GDT



;*****************************************************************
;*                              AP IDT                           *
;*****************************************************************


;AP_IDT_BASE: dw AP_IDT_LIMIT ; size of the AP's IDT
;$base__: dw AP_IDT_RELOC,0 ; base address
;
;%ifdef short_IDT
;AP_IDT_RELOC equ $ -(8*8)
;%else
;AP_IDT_RELOC equ $
;%endif
;
;; start of the interrupt table:
;AP_IDT:
;
;%ifndef short_IDT
;;INT 0: AP div 0
; dw APINT0,SEL_CODE_16,8600h,0
;;INT 1: AP debug
; dw APINT1,SEL_CODE_16,8600h,0
;;INT 2: AP NMI
; dw APINT2,SEL_CODE_16,8600h,0
;;INT 3: AP debug
; dw APINT3,SEL_CODE_16,8600h,0
;;INT 4: AP Overflow
; dw APINT4,SEL_CODE_16,8600h,0
;;INT 5: AP Bound
; dw APINT5,SEL_CODE_16,8600h,0
;;INT 6: AP Invalid opcode
; dw APINT6,SEL_CODE_16,8600h,0
;;INT 7: no FPU
; dw APINT7,SEL_CODE_16,8600h,0
;%endif
;
;;INT 8: Fatal error handler
; dw AP_FATAL_ERROR,SEL_CODE_16,8600h,0
;
;%ifndef short_IDT
;;INT 9: coproc. segment overrun
; dw APINT9,SEL_CODE_16,8600h,0
;;INT 10: invalid TSS
; dw APINT10,SEL_CODE_16,8600h,0
;;INT 11: segment not present
; dw APINT11,SEL_CODE_16,8600h,0
;;INT 12: stack segment fault
; dw APINT12,SEL_CODE_16,8600h,0
;;INT 13: GPF
; dw APINT13,SEL_CODE_16,8600h,0
;;INT 14: Page fault
; dw APINT14,SEL_CODE_16,8600h,0
;;INT 15: reserved
; dw APINT15,SEL_CODE_16,8600h,0
;;INT 16: FPU error
; dw APINT16,SEL_CODE_16,8600h,0
;;INT 17: Alignment error
; dw APINT17,SEL_CODE_16,8600h,0
;%endif
```

```
;
;;end of the interrupt table:
;AP_IDT_LIMIT  equ $-AP_IDT_RELOC
;
;%ifndef short_IDT
;; the AP interrupt routines:
;APINT0:
; mov dx,AP_DIV_0
; jmp short AP_shutdown
;APINT1:
; mov dx,AP_DEBUG
; jmp short AP_shutdown
;APINT2:
; mov dx,AP_NMI
; jmp short AP_shutdown
;APINT3:
; mov dx,AP_BREAK
; jmp short AP_shutdown
;APINT4:
; mov dx,AP_OV
; jmp short AP_shutdown
;APINT5:
; mov dx,AP_BOUND
; jmp short AP_shutdown
;APINT6:
; mov dx,AP_OPCODE
; jmp short AP_shutdown
;APINT7:
; mov dx,AP_FPU
; jmp short AP_shutdown
;%endif
;
;AP_FATAL_ERROR:
; mov dx,MSG_FATAL_AP
;AP_shutdown:
; mov word[es:main_error_code],dx ; error code for the main processor
; mov dword[es:back_to_rm-4],0 ; exits the main processor
;endlessloop:
; halt ; shutdown
; jmp short endlessloop
;
;
;%ifndef short_IDT
;APINT9:
; mov dx,AP_COPRO
; jmp short AP_shutdown
;APINT10:
; mov dx,AP_TSS
; jmp short AP_shutdown
;APINT11:
; mov dx,AP_SEGM
; jmp short AP_shutdown
;APINT12:
; mov dx,AP_STACK
; jmp short AP_shutdown
;APINT13:
; mov dx,AP_GPF
; jmp short AP_shutdown
;APINT14:
; mov dx,AP_PAGE
; jmp short AP_shutdown
;APINT15:
; mov dx,AP_RESER
; jmp short AP_shutdown
;APINT16:
; mov dx,AP_FPUER
; jmp short AP_shutdown
;APINT17:
; mov dx,AP_ALIGN
; jmp short AP_shutdown
;%endif
;



;*****************************************************************
;*                    MOUSE INTERRUPT HANDLER                    *
;*****************************************************************

align 16
MouseHandler:
; backup
; (The backup is not complete here because it takes several IRQs
; to complete the packet.)
 push ax
 push bp
 mov bp,0 ; so that we can point to SS without
; using the expensive ss: segment override prefix,
; without even changing DS.
; the point of it all is to access the 16 bit segment
; that holds the mixed data+code part without speed loss
; and furious code bloat.
; option 1: change DS (slow: the processor lookups the GDT)
; option 2: SS override to access the data in 16 bit mode
; (since the stack is in the same place) but it requires a
; segment override prefix (a single byte that requires
; one cycle to decode, that is: slow)
; the [bp+xxx] form is a native 8086 form that uses
; ss as segment register, it is meant to be used as
; stack frame pointer. it is quickly decoded (no
; cycle penalty) and doesn't bloat much the code.

 push dx
; read the byte coming from the mouse/COM port
 mov dx,COM1PORT
 push cx
 mov cx,[bp+mouse_buttons]
 in al,dx

 mov dx,[bp+mouse_buffer] ; get the last byte in the queue
; first byte ?
 test al,64
 jz not_first
   xor ch,ch ; yes: clear the byte counter
not_first:
 inc ch   ; update the counter

; last byte ?
 cmp ch,3
 je other_byte

; update the variables
 mov dl,dh
 mov dh,al
 mov [bp+mouse_buttons],cx
 pop cx
 mov [bp+mouse_buffer],dx
 pop dx

; EOI
 mov al,20h ;code: End Of Interrupt
 out 20h,al ;free the interrupt chanel
 pop bp
 pop ax
 iret

; end of the first part (the packet is not complete)
;*****************************************************************


other_byte:
 pushad  ; save everything here, now.
 push cx ; save the buttons
 push dx ; save the byte queue

; MMX backup (interleaved with the other instructions
; to prevent write buffer overflows and other silliness)
; this is also useful to fill the pipeline stalls :-)
 movq [bp+backup_mm0],mm0

;; the following piece of code extracts the X and Y deltas
;; (mouse move distance in "mickeys"). The code has been
;; optimized for classic Pentium dual-issue pipelines.
;; X and Y extraction have been interleaved to fill the
;; pipeline stalls due to data dependencies. The
;; remaining stalls have been filled by MMX backup instructions.
;; (what an excuse for ugly code !...)
;; Expect disastrous performance on PPro architecture,
;; because bytewise manipulations are expensive.
;; If i was Intel...
 mov bh,dl ; extract X delta
 and dl,12 ; ... Y delta
 and bh,3
 movq [bp+backup_mm1],mm1
 shl bh,6
 movq [bp+backup_mm2],mm2
 or bh,dh
 shl dx,12
 sar bx,8
 or dh,al
 mov al,bl
 sar dx,8

; acceleration:
 imul bl
 movq [bp+backup_mm3],mm3
 mov cx,ax
 mov al,dl
 imul dl
```

```
 movq [bp+backup_mm4],mm4
 add cx,ax  ; CX = dx*dx + dy*dy

 movq [bp+backup_mm4],mm4

 mov si,bx ; temporary copy of Xdelta and Ydelta
 mov ax,dx

 movq [bp+backup_mm5],mm5

 cmp cx,byte 70  ; coef1 -> acc=2
 jb no_acceleration
   add bx,si
   add ax,dx

   cmp cx,160 ; coef2 -> acc=3
   jb no_acceleration
     add bx,si
     add ax,dx
no_acceleration:

 movq [bp+backup_mm6],mm6

; coordinates clipping in Y:
 add ax,[bp+mouse_y]
 jns not_neg1
   xor ax,ax
   jmp short not_sup1
not_neg1:
 cmp ax,YRESOLUTION-1
 jbe not_sup1
   mov ax,YRESOLUTION-1
not_sup1:

 movq [bp+backup_mm7],mm7

; coordinates clipping in X:
 add bx,[bp+mouse_x]
 jns not_neg2
   xor bx,bx
   jmp short not_sup2
not_neg2:
 cmp bx,XRESOLUTION-1
 jb not_sup2
   mov bx,XRESOLUTION-1
not_sup2:

; ax and bx are not written back now because the cursor
; must be erased first.

; Coordinates in the tunnel space:

;;; mov cl,[bp+zoom]
; the tunnel size here is implicitely limited to 64K*64K (4G sites !)
;[12/3/99: is it sure ??? 32b ???]
 mov si,ax
 mov di,bx
; shl si,cl
; shl di,cl
; shr si,4
; shr di,4
 add si,[bp+orig_y]
 add di,[bp+orig_x]
 mov [bp+tunnel_y],si
 mov [bp+tunnel_x],di

; mouse button press ?

 pop dx
 pop cx

; left button click ?
 test dl,32 ; the current status
 jz near no_button_press
 test cl,32 ; the old status
 jnz near no_button_press

; on the bottom bar ?
 cmp ax,YRESOLUTION-32
 ja mouse_bar

 cmp byte [bp+pause_flag],1
 jne near no_button_press

; resize the tunnel:
   mov [bp+YSIZESMC],si ; coordinates in the tunnel space !
   mov [bp+XSIZESMC],di
   mov dword [bp+cycle_count],0
   pushad
   mov ecx,change_size
   call SEL_CODE_32:return_relay
   popad
   jmp no_button_press

mouse_bar:
 push word no_button_press

;Button coordinates:
; Function start  end
;Quit:      0     15
;Pause:     32    48
;Reset:     56    72
;Step:      80    96
;Calibrate: 104   120
;Strip - :  128   144
;Strip #:   144   176 (number)
;Strip + :  176   192
;ArrowUp:   224   240
;ArrowDown: 248   264
;ArrowLeft: 272   288
;ArrowRight:296   312
;ZoomOut:   344   360
;ZoomIn:    408   424
;ScreenShot:440   456


; QUIT ?
 cmp bx,16
 jbe near do_quit

no_quit:

; PAUSE ?
 cmp bx,byte 32
 jb near no_plus ; nothing
 cmp bx,byte 48
 jb near pause ; ugly, ugly.... but we spare a jump ! :-P

; RESET ?
 cmp bx,byte 56
 jb near no_plus
 cmp bx,byte 72
 jb near do_reset


; STEP ?
 cmp bx,byte 80
 jb near no_plus
 cmp bx,byte 96
 jb near step


 cmp byte [bp+pause_flag],1
 jne near no_plus

; CALIBRATE ?

 cmp bx,byte 104
 jb near no_plus
 cmp bx,byte 120
 jb near do_calibrate

; STRIP MINUS ?
 cmp bx,128
 jb near no_plus
 cmp bx,144
 jb near strip_minus ; ugly, ugly...

; STRIP PLUS ?
 cmp bx,176
 jb no_plus
 cmp bx,192
 jb near strip_plus

; Arrow Up ?
 cmp bx,224
 jb no_plus
 cmp bx,240
 jb near scroll_up

; Arrow Down ?
 cmp bx,248
 jb no_plus
 cmp bx,264
 jb near scroll_down

; Arrow Left ?
 cmp bx,272
```

```
 jb no_plus
 cmp bx,288
 jb near scroll_left

; Arrow Right ?
 cmp bx,296
 jb no_plus
 cmp bx,312
 jb near scroll_right

; Zoom Out ?
 cmp bx,344
 jb no_plus
 cmp bx,360
 jb near zoom_out

; Zoom In ?
 cmp bx,408
 jb no_plus
 cmp bx,424
 jb near zoom_in

;ScreenShot ?
 cmp bx,440
 jb no_plus
 cmp bx,456
 jb near save_screen

no_plus:
 pop di

no_button_press:
; update the variables
 mov cl,dl
 and cl,48
 mov [bp+mouse_buttons],cx

; Now that the MMX registers are saved,
; we can erase the old cursor (if needed):

 call SEL_CODE_32:erase_cursor

; restore:
 popad
 movq mm0,[bp+backup_mm0]
 movq mm1,[bp+backup_mm1]
 movq mm2,[bp+backup_mm2]
 pop cx
 movq mm3,[bp+backup_mm3]
 movq mm4,[bp+backup_mm4]
 movq mm5,[bp+backup_mm5]
 pop dx
 movq mm6,[bp+backup_mm6]
 movq mm7,[bp+backup_mm7]
 pop bp

; EOI
 mov al,20h ;code: End Of Interrupt
 out 20h,al ;free the interrupt channel
 pop ax
 iret


;*****************************************************************
;*                      TINY SHARED ROUTINES (in the SHORT range ?) *
;*****************************************************************

do_calibrate:
 mov dword[bp+back_to_rm-4],calibration-back_to_rm
 mov byte [bp+pause_flag],2 ;wait
 ret

;*****************************************************************
; called on STEP order:

step:
 mov dword[bp+limit],0 ; compute only once
 mov dword[bp+back_to_rm-4],run-back_to_rm
 mov byte [bp+pause_flag],2 ; wait
 ret


;*****************************************************************
;* called on RESET order:

do_reset:
 mov dword[bp+limit],0 ; stop the computation
 mov dword[bp+back_to_rm-4],reset-back_to_rm
 mov byte [bp+pause_flag],2 ;wait
 ret


;*****************************************************************
; called on RUN/PAUSE order:

pause:
 push ax
 mov al,[bp+pause_flag]
 push ebx
 xor ebx,ebx
 test al,al
 mov al,2
 jz exit_pause

 mov al,0
 dec ebx
 mov dword[bp+back_to_rm-4],run-back_to_rm

exit_pause:
 mov [bp+pause_flag],al
 mov dword [bp+limit],ebx
 pop ebx
 pop ax
 ret


;*****************************************************************
; called on EXIT order:

do_quit:
 mov dword[bp+back_to_rm-4],0 ; exits the main processor
 mov dword[bp+limit],0; stop the computation
 ret

;*****************************************************************
; called on STRIP+/- order:

strip_plus:
 push ax

 mov al,[bp+strip_count]
 cmp al,32
 jae no_strip_change
 inc al
 jmp strip_change

strip_minus:
 push ax

 mov al,[bp+strip_count]
 cmp al,1
 jbe no_strip_change
 dec al

strip_change:

 mov [bp+strip_count],al


 push ax ; ->stack pointer still aligned to 4 bytes
 pushad
 mov ecx,modify_SMC
 mov esi,0
$base__ equ $-4
 call SEL_CODE_32:return_relay
 popad
 pop ax

no_strip_change:
 pop ax
 ret


;*****************************************************************
;*                  KEYBOARD INTERRUPT HANDLER                   *
;*****************************************************************

KbdHandler:
 push ax
 mov al,20h ;code: End Of Interrupt
 out 20h,al ;free the interrupt chanel
 push bp
 in al,60h  ;read the character from the kbd controler
 mov bp,0
 cmp al,129 ;ESCAPE key break code (that way, DOS won't get the break code)
 jne no_exit

 call do_quit
```

```
; emergency ?
 dec byte [bp+escape_count]
 jnz no_key
 jmp SEL_CODE_32:back_to_rm

no_exit:

 push word nothing

 cmp al,25 ; 'P' make code
 je near pause

 cmp al,31 ; 'S' make code
 je near step

 cmp al,147 ; 'R' break code
 je near do_reset


 cmp byte [bp+pause_flag],1
 jne no_plus_key

 cmp al,74 ; '-' make code
 je strip_minus

 cmp al,78 ; '+' make code
 je strip_plus

 cmp al,174 ; 'c' break code
 je near do_calibrate

 cmp al,73 ; PgUp make code
 je near zoom_out

 cmp al,81 ; PgDown make code
 je near zoom_in

 cmp al,72 ; ArrowUp make code
 je near scroll_up

 cmp al,80 ; ArrowDown make code
 je near scroll_down

 cmp al,75 ; ArrowLeft make code
 je near scroll_left

 cmp al,77 ; ArrowRight make code
 je near scroll_right

no_plus_key:
 pop ax ; pop the tricky return address off the stack
no_key:
 pop bp
 pop ax
 iret

; end of the first part (without a bar update)
;*****************************************************************

nothing: ; second exit point for the KDB routine
 push ax ; -> stack pointer still 4-byte aligned
 pushad
 mov ecx,display_bar
 movq [bp+bmm0],mm0
 movq [bp+bmm1],mm1
 movq [bp+bmm2],mm2
 movq [bp+bmm3],mm3
 movq [bp+bmm4],mm4
 movq [bp+bmm5],mm5
 movq [bp+bmm6],mm6
 movq [bp+bmm7],mm7
 call SEL_CODE_32:return_relay
 movq mm0,[bp+bmm0]
 movq mm1,[bp+bmm1]
 movq mm2,[bp+bmm2]
 movq mm3,[bp+bmm3]
 movq mm4,[bp+bmm4]
 movq mm5,[bp+bmm5]
 movq mm6,[bp+bmm6]
 movq mm7,[bp+bmm7]
 popad
 pop ax
 pop bp
 pop ax
 iret

;*****************************************************************
;*                          ZOOM IN OUT                          *
;*****************************************************************

zoom_in:
 push ax
 mov al,[bp+zoom]
 cmp al,4
 jbe no_zoom ; don't zoom more if already maximum
 dec al
 jmp short zoom_ok

zoom_out:
 push ax
 mov al,[bp+zoom]
 cmp al,6
 jae no_zoom
 inc al
zoom_ok:
 mov [bp+zoom],al
 mov word [bp+orig_x],0 ; should be further
 mov word [bp+orig_y],0 ; analyzed !
 pushad
 mov ecx,reset_display
 mov esi,0
$base__ equ $-4
 call SEL_CODE_32:return_relay
 popad
no_zoom:
 pop ax
 ret

;*****************************************************************
;*                              SCROLL                           *
;*****************************************************************

; vertical scroll:
scroll_up:
 push eax
 mov ax,[bp+orig_y]
 test ax,ax
 jz no_scroll_change
 sub ax,byte 64
 jnc v_scroll_write_back
 mov word [bp+orig_y],0
 jmp short no_v_scroll_write_back

scroll_down:
 push eax
; do not scroll down if YSIZE<YRES.
 mov eax,[bp+YSIZESMC]
; size adjustment here (zoom)
 push cx
 mov cl,[bp+zoom]
 shl eax,4
 shr eax,cl
 pop cx
 sub eax,YRESOLUTION-32
 jb no_scroll_change
 cmp eax,[bp+orig_y] ; ax=(YSIZE*zoom)-YRESOLUTION
 jbe no_scroll_change
 mov ax,[bp+orig_y]
 add ax,byte 64

v_scroll_write_back:
 mov [bp+orig_y],ax
no_v_scroll_write_back:

 pushad
 mov esi,0
$base__ equ $-4
 mov ecx,display_tunnel_plus ; also recompute some stuffs
 call SEL_CODE_32:return_relay
 popad

no_scroll_change:
 pop eax
 ret


; horizontal scroll:
scroll_left:
 push eax
 mov ax,[bp+orig_x]
 test ax,ax
 jz no_scroll_change
 sub ax,byte 64 ; coarse granularity, no scroll < 64, no further checks.
 jmp short h_scroll_write_back

scroll_right:
 push eax
; do not scroll right if XSIZE<XRES
 mov eax,[bp+XSIZESMC]
```

```
; size adjustment here (zoom)
 push cx
 mov cl,[bp+zoom]
 shl eax,4
 shr eax,cl
 pop cx
 sub eax,XRESOLUTION
 jb no_scroll_change
 cmp eax,[bp+orig_x]
 jbe no_scroll_change
 mov ax,[bp+orig_x]
 add ax,byte 64

h_scroll_write_back:
 mov [bp+orig_x],ax
 jmp short no_v_scroll_write_back


;*****************************************************************
; here we are in 32 b mode !!!
BITS 32


;*****************************************************************
;*             ERASE_CURSOR: 32b part of the mouse handler       *
;*****************************************************************

erase_cursor:
 mov ecx,[bp+mouse_y] ; fetches the old coordinates
 mov edx,[bp+mouse_x]
 cmp ecx,0 ; YSIZE SMC
YSIZE32_0 equ $-4 ; SIZE SMC
 jae do_cursor_erase
 cmp edx,0 ;XSIZE
XSIZE32_0 equ $-4 ; SIZE SMC
 jb no_cursor_erase

do_cursor_erase:
 shl ecx,10 ; IF XRESOLUTION is 1024
 and dl,0F8h ; align to a 8 byte boundary
 movq mm0,[es:bg_color] ; load the background color
 lea edi,[edx+ecx-(32+(XRESOLUTION*(CURSOR_LINES-1)))]
$video__ equ $-4 ; i'm rather amazed by this LEA :-)
 mov cl,CURSOR_LINES

loop_erase_cursor:
 movq [edi],mm0
 movq [edi+8],mm0
 movq [edi+16],mm0
 movq [edi+24],mm0
 movq [edi+32],mm0
 add edi,XRESOLUTION
 dec cl
 jnz loop_erase_cursor

no_cursor_erase:

; backup:
 and ebx,0FFFFh ; keeps only bx and ax
 and eax,0FFFFh
 mov [bp+mouse_x],ebx
 mov [bp+mouse_y],eax

; IF the cursor is over the bottom bar,
; refresh its contents
 push dword common_return_address
 cmp eax,YRESOLUTION-32
 jae near display_bar
; else simply display the cursor
 jmp no_bar_update ; branch optimisation to do here ???

return_relay: ; this simple function is merged with erase_cursor (saves some bytes :-D)
 call ecx
common_return_address:
 db 066h ; caller=16b
 retf

;;*****************************************************************
;;*                                                               *
;;*                           AP CODE                             *
;;*                                                               *
;;*****************************************************************
;
;; this is run during the configuration detection
;AP_INIT:
; mov ax,cs
; mov ds,ax
; lock
; inc byte [processor_count]
; hlt
;
;; this is the real start
;AP_RUN: ; in real mode
; mov ax,cs
; mov ds,ax
; mov ss,ax
; mov sp,AP_stack
; lidt [AP_IDT_BASE]
; push word SEL_CODE_32
; push word AP_32
; jmp switch_PM32 ; "calls" the PMODE switch routine

BITS 32

;AP_32: ; AP entry point in 32b protected mode
;
;;******************** AP RUN *******************
;
;AP_mainloop:
;
; jmp AP_mainloop


;*****************************************************************
;*                              WAIT                             *
;*****************************************************************

wait_routine: ; little wait loop
 mov eax,3000000
wait_INIT:
 dec eax
 jnz wait_INIT
 ret


;*****************************************************************
;*           MAIN PROCESSOR'S ENTRY POINT IN THE 32b WORLD       *
;*****************************************************************


code_32:
  call change_size
  sti

; detect_AP

 mov al,0
AP_setup equ $-1
; test al,al
; jz near halte
;
;;********** AP DETECTION **********
;; Here we test the multiprocessing capabilities
; mov byte [es:AP_setup],0 ; reset the flag, in case the APIC is connected to nothing else
;
;; 1) We reset the other processor(s) (in case they were already running)
; mov dword [APIC_ICR_LO],0C0500h ; broadcast INIT signal
;
;; 2) We setup some code to jump to.
;; this code is simply a JMP to a fixed destination,
;; but the JUMP must be aligned on a 4KB boundary.
; mov edi,AP_landing_zone+4095
;$base__ equ $-4
; and edi,0FF000h
; mov byte [edi], 0EAh ; JMP opcode
; mov dword [edi+1],AP_INIT ;
;reloc_seg equ $-2 ; sensitive code !
;
;; give some time for the processors to reset
; call wait_routine

;; 3) We command the other processor(s) to start
;; executing the detection routine
; mov ebx,edi
; shr ebx,12
; or ebx,0C0600h ; SIPI signal
; mov [APIC_ICR_LO],ebx ; send the signal
;
;; another little wait loop (during which
;; the other processor(s) write to the (SMC) variable)
; call wait_routine
;
;; 4) check if exactly one other processor is present
; mov al,0 ; (SMC)
;processor_count equ $-1
; cmp al,1
; jne halte ; if not one other processor, run the monoprocessor program
;
```

```
;; set the semaphore to 0
; mov byte [es:SEMAPHORE],0
;
;; 5) reset the AP (once again)
; mov dword [APIC_ICR_LO],0C0500h ; broadcast INIT
;
;; change the destination of the jump to the initialisation routine
; mov word [edi+1],AP_RUN ; points to another starting routine
;; setup some variables
;; mov byte [es:AP_setup],1 ; now, we are sure the APIC is connected to another processor
;; mov dword [es:MP_label],main_loop_MP-(MP_label+4)   ; SMC !
;; mov dword [es:MP_label2],main_loop_MP-(MP_label2+4) ; SMC !
;
;; Wait for the INIT to complete
; call wait_routine
;
;; 6) restart the AP
; mov [APIC_ICR_LO],ebx ; SIPI
;

;*****************************************************************
;*                      THE WAITING ROUTINE                      *
;*****************************************************************

halte:
  hlt
  jmp halte
back_to_rm:
; mov dword [APIC_ICR_LO],0C0500h ; INIT signal to the AP(s)
 mov dx,0
main_error_code equ $-2
exit_with_message:
 jmp SEL_CODE_16:Back_2_16b


;*****************************************************************
;*                         CHANGE_SIZE                           *
;*****************************************************************

change_size:
; Parameters: none
; registers used: all (no mercy !)

 mov esi,0 ; as to avoid prefixes or similar worries
$base__ equ $-4

; update the random number generator:
 RDTSC
 add eax,edx
;;;; add [esi+RANDOM],eax
;;;; add [esi+RANDOM+4],eax

 mov eax,[esi+XSIZESMC]
 mov ebx,[esi+YSIZESMC]

; eax=XSIZE
; ebx=YSIZE
; other registers: used and trashed

; 1) round the size to nearest
 add eax, byte 31
 add ebx, byte 3
 and al,~(64-1)
 and bl,~(8-1)

; 2) size clip ; probably CMPXCHG could help here

 cmp eax,256 ; minimum: 2*64(MMX)*2(columns)
 jae no_eax_SP
 mov eax,256
no_eax_SP:
 cmp ebx,256 ; minimum: 32(maximum strip mining)*4(synchronized parts)
 jae no_ebx_SP
 mov ebx,256
no_ebx_SP:
 cmp eax,0FF00h ; maximum: 64K-256
 jb no_eax_SP2
 mov eax,0FF00h;
no_eax_SP2:
 cmp ebx,0FF00h ; maximum: 64K-256
 jb no_ebx_SP2
 mov ebx,0FF00h
no_ebx_SP2:


; 3) check if enough memory (does a dirty exit :-()
 mov ecx,eax
 mov edx,ebx
 inc edx  ; garde
 imul ecx,edx
 push ecx

 lea edx,[eax*5] ; compute the buffer's size
 shl edx,3 ; edx=XSIZE*80*32/64=(XSIZE*5)*8

 lea ecx,[edx+1023] ; round up 1KB
 shr ecx,10
 cmp ecx,65535
 ja near back_to_rm
 cmp cx,[esi+XMS_SIZE]
 ja near back_to_rm

; backup:
 mov [esi+XSIZESMC],eax
 mov [esi+YSIZESMC],ebx


; 4) modify the constants in the instructions:

 call modify_SMC

; 5) clean the tunnel

 mov edi,0
$mem__ equ $-4
 pop ecx
 push edi
 add ecx,edi

 pxor mm0,mm0

align 16
loop_fill_mem1:
   movq [edi],mm0
   movq [edi+8],mm0 ; this is enough to saturate the data bus !
   add edi,16
   cmp edi,ecx
 jb loop_fill_mem1

 pop edi
 add edi,[esi+XSIZESMC]
 add edi,[esi+XSIZESMC]
 sub ecx,[esi+XSIZESMC]
 sub ecx,[esi+XSIZESMC]

%ifdef gilbruz

 pcmpeqb mm0,mm0 ; mm0=-1
; psrlq mm0,10
; psllq mm0,5
loop_fill2:
 movq [edi+(CEL*0)+A],mm0
 movq [edi+(CEL*0)+B],mm0
 movq [edi+(CEL*0)+C],mm0
 movq [edi+(CEL*0)+D],mm0
 movq [edi+(CEL*0)+E],mm0
 movq [edi+(CEL*0)+F],mm0
 movq [edi+(CEL*0)+G],mm0

 movq [edi+(CEL*1)+A],mm0
 movq [edi+(CEL*1)+B],mm0
 movq [edi+(CEL*1)+C],mm0
 movq [edi+(CEL*1)+D],mm0
 movq [edi+(CEL*1)+E],mm0
 movq [edi+(CEL*1)+F],mm0
 movq [edi+(CEL*1)+G],mm0

 movq [edi+(CEL*2)+A],mm0
 movq [edi+(CEL*2)+B],mm0
 movq [edi+(CEL*2)+C],mm0
 movq [edi+(CEL*2)+D],mm0
 movq [edi+(CEL*2)+E],mm0
 movq [edi+(CEL*2)+F],mm0
 movq [edi+(CEL*2)+G],mm0

 movq [edi+(CEL*3)+A],mm0
 movq [edi+(CEL*3)+B],mm0
 movq [edi+(CEL*3)+C],mm0
 movq [edi+(CEL*3)+D],mm0
 movq [edi+(CEL*3)+E],mm0
 movq [edi+(CEL*3)+F],mm0
 movq [edi+(CEL*3)+G],mm0

 add edi,[esi+XSIZESMC]
 cmp edi,ecx
 jb near loop_fill2

%else
```

```
%define test_vectors_on

 push eax
 push ebx
 push ecx
 push edx


 mov eax,[esi+XSIZESMC]
 lea ebx,[eax*8]

 lea ecx,[eax*5]

 add edi,ecx
 lea edi,[edi+eax*4]

 push edi

 mov edx,edi

; A+G -> F+B
 mov al,1000001b
 call test_vector
 mov al,0100001b
 call test_vector
 mov al,0010001b
 call test_vector
 mov al,0001001b
 call test_vector
 mov al,0000101b
 call test_vector
 mov al,0000011b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi

; F+B -> A+G
 mov al,1010000b
 call test_vector
 mov al,0101000b
 call test_vector
 mov al,0010100b
 call test_vector
 mov al,0001010b
 call test_vector
 mov al,1000100b
 call test_vector
 mov al,0100010b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi

; A+D -> B+E or C+F
 mov al,1001000b
 call test_vector
 call test_vector2
 mov al,0100100b
 call test_vector
 call test_vector2
 mov al,0010010b
 call test_vector
 call test_vector2

 lea edi,[edx+CEL]
 mov edx,edi


; A+C+G -> A+B+D or B+C+F
 mov al,1010001b
 call test_vector
 mov al,0101001b
 call test_vector
 mov al,0010101b
 call test_vector
 mov al,0001011b
 call test_vector
 mov al,1000101b
 call test_vector
 mov al,0100011b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi


; A+B+E -> (?
 mov al,1100100b
 call test_vector
 mov al,0110010b
 call test_vector
 mov al,1011000b
 call test_vector
 mov al,0101100b
 call test_vector
 mov al,0010110b
 call test_vector
 mov al,1001010b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi


; A+C+F -> (?
 mov al,1010010b
 call test_vector
 mov al,1101000b
 call test_vector
 mov al,0110100b
 call test_vector
 mov al,0011010b
 call test_vector
 mov al,1001100b
 call test_vector
 mov al,0100110b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi

; A+D+G -> B+D+F or A+C+E
 mov al,1001001b
 call test_vector
 call test_vector2
 mov al,0100101b
 call test_vector
 call test_vector2
 mov al,0010011b
 call test_vector
 call test_vector2

 lea edi,[edx+CEL]

 mov edx,edi

; A+C+E -> B+E+G or A+D+G or C+F+G
 mov al,1010100b
 call test_vector
 call test_vector2
 call test_vector3
 mov al,0101010b
 call test_vector
 call test_vector2
 call test_vector3


 pop edi

 lea edi, [edi+8*ebx]
 lea edi, [edi+8*ebx]

 push edi

 mov edx,edi

; A+G -> F+B
 mov al,0111110b
 call test_vector
 mov al,1011110b
 call test_vector
 mov al,1101110b
 call test_vector
 mov al,1110110b
 call test_vector
 mov al,1111010b
 call test_vector
 mov al,1111100b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi

; F+B -> A+G
 mov al,0101111b
 call test_vector
 mov al,1010111b
```

```
 call test_vector
 mov al,1101011b
 call test_vector
 mov al,1110101b
 call test_vector
 mov al,0111011b
 call test_vector
 mov al,1011101b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi

; A+D -> B+E or C+F
 mov al,0110111b
 call test_vector
 call test_vector2
 mov al,1011011b
 call test_vector
 call test_vector2
 mov al,1101101b
 call test_vector
 call test_vector2

 lea edi,[edx+CEL]
 mov edx,edi


; A+C+G -> A+B+D or B+C+F
 mov al,0101110b
 call test_vector
 mov al,1010110b
 call test_vector
 mov al,1101010b
 call test_vector
 mov al,1110100b
 call test_vector
 mov al,0111010b
 call test_vector
 mov al,1011100b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi


; A+B+E -> (?
 mov al,0011011b
 call test_vector
 mov al,1001101b
 call test_vector
 mov al,0100111b
 call test_vector
 mov al,1010011b
 call test_vector
 mov al,1101001b
 call test_vector
 mov al,0110101b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi


; A+C+F -> (?
 mov al,0101101b
 call test_vector
 mov al,0010111b
 call test_vector
 mov al,1001011b
 call test_vector
 mov al,1100101b
 call test_vector
 mov al,0110011b
 call test_vector
 mov al,1011001b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi

; A+D+G -> B+D+F or A+C+E
 mov al,0110110b
 call test_vector
 call test_vector2
 mov al,1011010b
 call test_vector
 call test_vector2
 mov al,1101100b
 call test_vector
 call test_vector2


 lea edi,[edx+CEL]
 mov edx,edi

; A+C+E+G -> B+E+G or A+D+G or C+F+G
 mov al,1010101b
 call test_vector
 call test_vector2
 call test_vector3
 mov al,0101011b
 call test_vector
 call test_vector2
 call test_vector3

 pop edi
 lea edi, [edi+8*ebx]
 lea edi, [edi+8*ebx]
 mov edx,edi

; counter-verifications :

 mov al,1b
 call test_vector
 mov al,1111110b
 call test_vector
 mov al,1111111b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi

; A+B
 mov al,1100000b
 call test_vector
 mov al,0110000b
 call test_vector
 mov al,0011000b
 call test_vector
 mov al,0001100b
 call test_vector
 mov al,0000110b
 call test_vector
 mov al,1000010b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi

; A+B+G
 mov al,1100001b
 call test_vector
 mov al,0110001b
 call test_vector
 mov al,0011001b
 call test_vector
 mov al,0001101b
 call test_vector
 mov al,0000111b
 call test_vector
 mov al,1000011b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi

; A+B+C
 mov al,1110000b
 call test_vector
 mov al,0111000b
 call test_vector
 mov al,0011100b
 call test_vector
 mov al,0001110b
 call test_vector
 mov al,1000110b
 call test_vector
 mov al,1100010b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi

; A+B+C+G
 mov al,1110001b
 call test_vector
 mov al,0111001b
```

```
 call test_vector
 mov al,0011101b
 call test_vector
 mov al,0001111b
 call test_vector
 mov al,1000111b
 call test_vector
 mov al,1100011b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi

; A+B+C+D
 mov al,1111000b
 call test_vector
 mov al,0111100b
 call test_vector
 mov al,0011110b
 call test_vector
 mov al,1001110b
 call test_vector
 mov al,1100110b
 call test_vector
 mov al,1110010b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi

; A+B+C+D+G
 mov al,1111001b
 call test_vector
 mov al,0111101b
 call test_vector
 mov al,0011111b
 call test_vector
 mov al,1001111b
 call test_vector
 mov al,1100111b
 call test_vector
 mov al,1110011b
 call test_vector

 lea edi,[edx+CEL]
 mov edx,edi

; A+B+C+D+E+G
 mov al,1111101b
 call test_vector
 mov al,1111101b
 call test_vector
 mov al,0111111b
 call test_vector
 mov al,1011111b
 call test_vector
 mov al,1101111b
 call test_vector
 mov al,1110111b
 call test_vector



 pop edx
 pop ecx
 pop ebx
 pop eax

%endif

 call create_modify_lists

; 6) and now, reset the display

reset_display:

 mov edi,0
$video__ equ $-4
 movq mm0,[esi+bg_color]

loop_fill_bg_SP:
   movq [edi],mm0
   movq [edi+8],mm0
   add edi, byte 16
   cmp edi,XRESOLUTION*YRESOLUTION
$video__ equ $-4
 jb loop_fill_bg_SP

display_tunnel_plus:
; Coordinates in the tunnel space:
 mov eax,[esi+mouse_y]
 mov ebx,[esi+mouse_x]
 mov cl,[esi+zoom]
 shl eax,cl
 shl ebx,cl
 shr eax,4
 shr ebx,4
 add eax,[esi+orig_y]
 add ebx,[esi+orig_x]
 mov [esi+tunnel_y],eax
 mov [esi+tunnel_x],ebx

 call modify_zoom

 call display_tunnel

; 7) Display the bottom bar
; Nota (bene bene): I removed the call and the ret,
; avoided the nasty jump trick by merging the routines.


;*****************************************************************
;*                          DISPLAY_BAR                          *
;*****************************************************************

display_bar: ; used by the mouse handler, change_size, pause...
; it first displays the numbers (mouse coord etc) then displays the cursor
; parameters:
;  (none)
; used:
;  edi:screen coordinates
;  edx:displayed symbol
;  ebx:char count, etc.

; cycles, count:
 call display_cycles

; QUIT button
 mov edi,XRESOLUTION*(YRESOLUTION-24)
$video__ equ $-4
 mov edx,QUIT
 call display_number

; PLAY button
 add edi,byte 32
 mov dl,PLAY
 add dl,[es:pause_flag]
 call display_number

; RESET button
 add edi,byte 24
 mov dl,RAZ
 call display_number

; STEP button:

 add edi,byte 24
 mov dl,STEP
 call display_number

; STRIP COUNT
 add edi,byte 80
 mov eax,[es:strip_count]
 xor edx,edx
 mov bl,2
 call display_count

; + buttons:
 cmp byte [es:pause_flag],1
 jne no_plus_minus_cal

; MINUS
 mov dl,MINUS
 call display_number
; CALIBRATE button:
 sub edi,byte 24
 mov dl,CALIBRATE
 call display_number

; PLUS
 add edi,byte 72
 mov dl,PLUS
 call display_number

 jmp short no_space

no_plus_minus_cal:
 mov dl,SPACE
 call display_number
```

```
;CALIBRATE
 sub edi,byte 24
 call display_number
; PLUS
 add edi,byte 72
 call display_number

no_space:

 add edi,byte 48
 mov dl,ARROW_UP
 call display_number

 add edi,byte 24
 mov dl,ARROW_DOWN
 call display_number

 add edi,byte 24
 mov dl,ARROW_LEFT
 call display_number

 add edi,byte 24
 mov dl,ARROW_RIGHT
 call display_number

 add edi,byte 48
 mov dl,ZOOM_OUT
 call display_number

 add edi,byte 64
 mov dl,ZOOM_IN
 call display_number

 add edi,byte 32
 mov dl,SCREENSHOT
 call display_number

 sub edi,48
 mov eax,[es:zoom]
 sub eax,4
 pushfd
; absolute value: (should be computed somewhere else !)
 mov edx,eax
 sar edx,31 ; edx=FFFF.. or 0
 xor eax,edx
 sub eax,edx; inc eax (or not).
 mov bl,2
 xor edx,edx ; forget it if you want to trigger a DIV/0 trap.
 call display_count

 popfd
 mov dl,PLUS
 sbb dl,0
 call display_number


no_bar_update: ; jumped here from erase_cursor

; display mouse_Y (tunnel space)
 mov eax,[es:tunnel_y]
 mov edi,(XRESOLUTION*(YRESOLUTION-23))-400
$video__ equ $-4
 mov bl,5
 xor edx,edx ; forget it if you want to trigger a DIV/0 trap.
 call display_count

; display mouse_X (tunnel space)
 sub edi,byte 8
 mov eax,[es:tunnel_x]
 mov bl,5
 call display_count

; display the cursor
 mov eax,[es:mouse_y]
 mov ebx,[es:mouse_x]
 jmp display_cursor
; no ret... display_cursor does it for us.


;*****************************************************************
;*                          DISPLAY_CYCLES                       *
;*****************************************************************

align 16
display_cycles: ; display the Time Stamp Counter:
; used:
;  edx,edi,ebx,eax (non terminal)

 mov eax,[TSC]
$base__ equ $-4
 xor edx,edx
 mov edi,(XRESOLUTION*(YRESOLUTION-23))-16
$video__ equ $-4
 mov bl,10 ; character counter
 call display_count

 mov eax,[cycle_count]
$base__ equ $-4
 sub edi,byte 24
 mov bl,10
 call display_count

 ret

%ifdef test_vectors_on
;*****************************************************************
;*                            TEST_VECTORS                       *
;*****************************************************************



test_vector:
 add edi,ebx

 test al,10000b
 jz not_C
 mov byte [edi+C+4],          100000b
not_C:

 test al,100000b
 jz not_B
 mov word [edi+B+4],     10000000000b
not_B:

 add edi,ecx

 test al,1b
 jz not_G
 mov word [edi+G+4],       100000000b
not_G:

 test al,1000b
 jz not_D
 mov byte [edi+D+4],            1000b
not_D:

 test al,1000000b
 jz not_A
 mov word [edi+A+4],  10000000000000b
not_A:

 add edi,ecx

 test al,100b
 jz not_E
 mov byte [edi+E+4],          100000b
not_E:

 test al,10b
 jz not_F
 mov word [edi+F+4],     10000000000b
not_F:

 ret


test_vector2:
 add edi,ebx

 test al,10000b
 jz not_C2
 mov byte [edi+C+4],          100b
not_C2:

 test al,100000b
 jz not_B2
 mov word [edi+B+4],     10000000b
not_B2:

 add edi,ecx

 test al,1b
 jz not_G2
 mov byte [edi+G+4],       100000b
not_G2:

 test al,1000b
 jz not_D2
```

```
 mov byte [edi+D+4],            1b
not_D2:

 test al,1000000b
 jz not_A2
 mov word [edi+A+4],  10000000000b
not_A2:

 add edi,ecx

 test al,100b
 jz not_E2
 mov byte [edi+E+4],          100b
not_E2:

 test al,10b
 jz not_F2
 mov word [edi+F+4],     10000000b
not_F2:

 ret


test_vector3:
 add edi,ebx

 test al,10000b
 jz not_C3
 mov byte [edi+C+4],          1000b
not_C3:

 test al,100000b
 jz not_B3
 mov word [edi+B+4],     100000000b
not_B3:

 add edi,ecx

 test al,1b
 jz not_G3
 mov byte [edi+G+4],       1000000b
not_G3:

 test al,1000b
 jz not_D3
 mov byte [edi+D+4],            10b
not_D3:

 test al,1000000b
 jz not_A3
 mov word [edi+A+4],  100000000000b
not_A3:

 add edi,ecx

 test al,100b
 jz not_E3
 mov byte [edi+E+4],          1000b
not_E3:

 test al,10b
 jz not_F3
 mov word [edi+F+4],     100000000b
not_F3:

 ret

%endif

;*****************************************************************
;*                           MODIFY_ZOOM                         *
;*****************************************************************

modify_zoom: ; Zoom dependent code
; used:eax,ebx,ecx
; esi=base

; ZOOM SMC: calls the right routine
 cmp byte [esi+zoom],4 ; zoom=1:1
 jne zoom_sel_2

;ZOOM=1:1
 mov dword [esi+DISPLAYLINE_SMC1],display_line-(DISPLAYLINE_SMC1+4)
 mov dword [esi+DISPLAYLINE_SMC2],display_line-(DISPLAYLINE_SMC2+4)
 jmp short end_sel_zoom
zoom_sel_2:
 cmp byte [esi+zoom],5 ; zoom=1:1
 jne zoom_sel_3

;ZOOM=2:1
 mov dword [esi+DISPLAYLINE_SMC1],display_line2-(DISPLAYLINE_SMC1+4)
 mov dword [esi+DISPLAYLINE_SMC2],display_line2-(DISPLAYLINE_SMC2+4)
 jmp short end_sel_zoom
zoom_sel_3:

;ZOOM=4:1
 mov dword [esi+DISPLAYLINE_SMC1],display_line4-(DISPLAYLINE_SMC1+4)
 mov dword [esi+DISPLAYLINE_SMC2],display_line4-(DISPLAYLINE_SMC2+4)
end_sel_zoom:

 mov eax,[esi+XSIZESMC]
 mov ebx,[esi+YSIZESMC]

; clip these ones to the screen resolution:
 mov ecx,XRESOLUTION
 cmp eax,ecx
 jae no_clip_XR
 mov ecx,eax
no_clip_XR:
 mov [esi+XSIZE32_0],ecx ; erase_cursor
 mov [esi+XSIZE32_5],ecx ; display_line
 mov [esi+XSIZE32_6],ecx ; display_line

 mov ecx,YRESOLUTION-32
 cmp ebx,ecx
 jae no_clip_YR
 mov ecx,ebx
no_clip_YR:
 mov [esi+YSIZE32_0],ecx ; erase_cursor

; not a cool problem:
 push eax
 shr eax,1
 mov ecx,XRESOLUTION
 cmp eax,ecx
 jae no_clip_XR3
 mov ecx,eax
no_clip_XR3:
 mov [esi+XSIZE32_7],ecx ; display_line2
 shr eax,1
 mov ecx,XRESOLUTION
 cmp eax,ecx
 jae no_clip_XR4
 mov ecx,eax
no_clip_XR4:
 mov [esi+XSIZE32_9],ecx ; display_line4
 pop eax

 ret


;*****************************************************************
;*                           MODIFY_SMC                          *
;*****************************************************************

modify_SMC:
;used: eax,ebx,ecx,edx,esi
; esi points to the code segment
;output ebx=YSIZE/2 !

 call modify_zoom ; loads eax and ebx

; modification to the first plan: one cell= 64 bytes = 64 sites,
; the cells are all aligned to cache lines and prevents intempestive
; traffic on the bus in multiprocessor configurations (MESI traffic)
; the first intent was to have one descriptor per line but the
; linked list would be too big. We need one descriptor per cell,
; to reduce the size of the linked list, and simplify the multiprocessing
; code. other advantages:
; -we can make "capilarity" or "sticking" walls.
; -we can display the walls with another color that is distinct
; from the "fluid"
; -we can avoid to count (in the density summation) the cells which
; have a wall.

; the first part of the WALL part of the cell is used to hold a pointer
; to the linked list. The second part is a pointer to executable code.
; these pointers are not "executed" when "null".


; SP only:
 mov [esi+YSIZE32_2],ebx ; SP_run
 mov [esi+WIDTH],eax ; SP_run
 mov [esi+WIDTH2],eax
 mov [esi+WIDTH3],eax
 mov [esi+WIDTH4],eax
 mov [esi+WIDTH5],eax
```

```
 mov ecx,F
 sub ecx,eax
 mov [esi+_LINE_F],ecx
 mov [esi+_LINE_F2],ecx
 mov ecx,F-CEL
 sub ecx,eax
 mov [esi+_LINE_F_CELL],ecx
 mov [esi+_LINE_F_CELL2],ecx

 mov ecx,E
 sub ecx,eax
 mov [esi+_LINE_E],ecx
 mov [esi+_LINE_E2],ecx ; (-XSIZE)+E
; mov [esi+_LINE_E3],ecx
; add ecx,CEL
; mov [esi+_LINE_E_CELL],ecx


 mov ecx,eax
 imul ecx,ebx ; size of the tunnel
 add ecx,dword 0
$mem__ equ $-4
;>TRIGGER_1 equ $-4 ; (YSIZE*LINE)+mem
 mov [esi+TRIGGER_1],ecx

 add ecx,eax ; ecx=mem+(XSIZE*(YSIZE+1))
; ecx points to the start of the buffer
 mov [esi+START_BUFFER],ecx

; compute the size of one buffer line
 mov edx,[esi+XSIZESMC]
 lea edx,[edx*5]
 shr edx,2 ; edx=XSIZE*5/4 bytes
 mov [esi+XS32_80],edx

; compute the size of one buffer column:
 mov edx,[esi+strip_count]
 lea edx,[edx*5]
 shl edx,4 ; edx=strip*80 bytes
; mov [esi+STRIP80],edx

 imul edx,eax ; edx=strip*XSIZE
 add ecx,edx ; edx=(strip*80*XSIZE/64)+mem+(XIZE*(YSIZE+1))
 mov [esi+END_BUFFER],ecx

 push eax
;>   por mm0,[edx+0]
;>XSIZE32CELA equ $-4
 add eax,byte CEL+A
 mov [esi+XSIZE32CELA],eax

;>   por mm0,[edx+0]
;>XSIZE32CELD equ $-4
 add eax,byte D-A
 mov [esi+XSIZE32CELD],eax
 pop eax

 push eax
;>STRIP_1 equ $-4 ; (strip*line)+mem
  mul dword [esi+strip_count]
  add eax,0
$mem__ equ $-4
  mov [esi+STRIP_1],eax
 pop eax

;START:
 add eax,0 ; start=LINE+mem
$mem__ equ $-4
 mov [esi+START_1],eax
 mov [esi+START_2],eax

 ret


;*****************************************************************
;*                       DISPLAY_CURSOR                          *
;*****************************************************************

align 16

display_cursor: ; a pure reentering 32b function (bad multiproc souvenirs)
; ebx=x
; eax=y
; mm0..6 are destroyed, the caller must save them !
; all normal registers are saved:
 pushad ; should be replaced by a "progressive" backup/restore

; Y CLIPPING
 mov ecx,CURSOR_LINES-1
 mov edi,eax ; start_display_line=y
 xor esi,esi ; start_cursor_line=0
 sub edi,ecx ; start_display_line=y-CURSOR_LINES
 ja no_clip_cursor_y ; IF clip
 sub ecx,eax ; start_cursor_line=CURSOR_LINES-y
 mov edi,esi ; start_display_line=0
 lea esi,[ecx*4] ; start_cursor_line=(CURSOR_LINES-y)*4
no_clip_cursor_y:

 shl edi,10 ; multiply by 1024,
; instead of imul edi,XRESOLUTION if it's not a power of two
; imul edi,edi,XRESOLUTION

 add edi,0
$video__ equ $-4

; X CLIPPING
 xor eax,eax ; shift=0
 mov edx,ebx
 sub ebx,byte 31  ; x=x-31
 jns no_clip_x
 xchg ebx,eax
 neg al
no_clip_x:

; eax:initial cursor mask shift
; ebx:backup of starting X
; edx:current X
; ecx:free... used for several shifts
; esi:cursor pointer
; edi:current line
; ebp:mask
; mm0:0
; mm1:color mask
; mm2:cursor mask
; mm3:left shift count (<8) then working space for a cursor mask byte
; mm4:shifted cursor mask (8040201008040201h)
; mm5:copy of the byte mask
; mm6: background

 movq mm4,[shifted_mask]
$base__ equ $-4
 pxor mm0,mm0
 movq mm1,[color_mask]
$base__ equ $-4

loop_display_cursor_lines:
 mov ebp,[cursor+esi] ; loads the mask
$base__ equ $-4
 mov ecx,eax ; restore ecx
 mov edx,ebx ; start offset
 shr ebp,cl ; mask adjust if clipping

 bsf ecx,ebp ; search the first interesting bit
 jz end_line_cursor ; if nothing then skip.

;there is at least one bit to display
 add edx,ecx
 shr ebp,cl ; "justifies" the mask to the right

 movd mm2,ebp ; transfer to a 64 bit register
; to avoid the overflow of the right shift
 mov ecx,edx

 and dl,0F8h ; 8-byte alignment
 and ecx,7

 movd mm3,ecx
 psllq mm2,mm3 ; the shifted cursor mask is now in mm2

loop_line_cursor:
 movq mm6,[edi+edx] ;loads the background (in background, hehe !)
 movq mm3,mm2
 punpcklbw mm3,mm3 ;
 punpcklwd mm3,mm3 ; instead of a multiplication
 punpckldq mm3,mm3 ;
 pand mm3,mm4     ; select a byte
 pcmpeqb mm3,mm0 ; if (byte) not zero then 0FFh
 movq mm5,mm3
 pand mm3,mm6 ; background mask
 pandn mm5,mm1 ; color mask
 por mm3,mm5
 movq [edi+edx],mm3 ; write back
 add edx,byte 8
; check if it's worth going on:
 psrlq mm2,8 ; mm2 shifted right by 8
 movd ebp,mm2
 test ebp,ebp
 jnz loop_line_cursor
```

```
end_line_cursor:
 add esi,byte 4
 add edi,XRESOLUTION
 cmp esi,(CURSOR_LINES*4)
 jb loop_display_cursor_lines

 popad
 ret


;*****************************************************************
;*                          DISPLAY_COUNT                        *
;*****************************************************************

align 16
display_count:
; parameters:
;  bl :     precision
;  edi:     video address
;  eax:edx: number
; used:
;  edx: reminder
;  ecx: number base (10)

 mov ecx,10 ; divide count (number base)

loop_display_cycle:
 div ecx  ; reminder in edx

 call display_number

 xor edx,edx ; clear the reminder
 sub edi,byte 16 ; update the pointer
 dec bl ; decrement the counter
 jnz loop_display_cycle

 ret


;*****************************************************************
;*                         DISPLAY_NUMBER                        *
;*****************************************************************

align 16

display_number:
 pushad

;parameters:
; edi=initial place (aligned on a 8-byte boundary)
; edx=number (0..9)

used:
; cl:  FONT_HEIGHT
; mm7: FG color
; mm6: BG color
; mm5: 0
; mm4: background
; mm3: working space
; mm2: byte_mask

 mov cl,FONT_HEIGHT
 movq mm6,[number_background]
$base__ equ $-4
 lea eax,[edx*4+edx] ; eax=5*edx
 movq mm7,[number_foreground]
$base__ equ $-4
 lea eax,[eax*4+edx] ; eax=eax*21
 movq [edi],mm6 ; high margin
 pxor mm5,mm5
 movq [edi+8],mm6 ; high margin (2)
 lea eax, [(eax*2)+font_numbers] ; eax=number*52+font_address
$base__ equ $-4
 movq [edi+XRESOLUTION],mm6 ; high margin (3)
 movq mm2,[shifted_mask]
$base__ equ $-4

 movq [edi+8+XRESOLUTION],mm6 ; high margin (4)
 add edi,XRESOLUTION*2

loop_display_number:
 movzx ebx,word[eax]
 add eax,byte 2

 movq mm4,mm6
 test bl,bl
 jz no_first_half

 movd mm3,ebx
 punpcklbw mm3,mm3
 punpcklwd mm3,mm3
 punpckldq mm3,mm3
 pand mm3,mm2    ; select a byte
 pcmpeqb mm3,mm5 ; if (byte) not zero then 0FFh
 pand mm4,mm3  ; background mask
 pandn mm3,mm7 ; color mask
 por mm4,mm3

no_first_half:
 movq [edi],mm4
 add edi,byte 8

 movq mm4,mm6
 shr ebx,8
 test bl,bl
 jz no_last_half

 movd mm3,ebx
 punpcklbw mm3,mm3
 punpcklwd mm3,mm3
 punpckldq mm3,mm3
 pand mm3,mm2    ; select a byte
 pcmpeqb mm3,mm5 ; if (byte) not zero then 0FFh
 pand mm4,mm3  ; background mask
 pandn mm3,mm7 ; color mask
 por mm4,mm3

no_last_half:
 movq [edi],mm4
 add edi,XRESOLUTION-8
 dec cl
 jnz loop_display_number

; low margin:
 movq [edi],mm6
 movq [edi+8],mm6

 popad
 ret

;*****************************************************************
;*                              RESET                            *
;*****************************************************************

reset:
 mov dword[es:back_to_rm-4],-6 ; loop again
 mov dword[es:cycle_count],0 ; don't go compute
 mov byte [es:pause_flag],1 ; pause
 call change_size
 jmp halte+1 ; don't halt yet. there might be another command in the queue !


;*****************************************************************
;*                        DISPLAY TUNNEL                         *
;*****************************************************************

display_tunnel: ; this is not really optimized, but at least it works.

; compute the increment
 mov ecx,[zoom]
$base__ equ $-4
 sub ecx,byte 4
 mov eax,1      ; may not work if zoomins are implemented
 shl eax,cl
 mov [SMC_ADD],eax
$base__ equ $-4

; compute the upper limit:

; compute the address of the starting line:
 mov eax,[XSIZESMC]
$base__ equ $-4
 mov edx,[orig_y]
$base__ equ $-4
 push edx
 inc edx ; skip the garde
 mul edx
 mov edx,eax ; edx=XSIZE*orig_y
 pop eax ; eax=orig_y
 add edx,dword 0 ; edx=LINE*orig_y+START
$mem__ equ $-4

; eax: starting line
; edx: address of the starting line

loop_display_tunnel:
 call display_line
DISPLAYLINE_SMC1 equ $-4
```

```
 add eax,0         ; u
SMC_ADD equ $-4
 add edx,[XSIZESMC]; v
$base__ equ $-4
 cmp eax,[YSIZESMC]; u
$base__ equ $-4
 jb loop_display_tunnel ; v

 ret



;*****************************************************************
;*                              RUN                              *
;*****************************************************************

run:
 mov dword[es:back_to_rm-4],end_run-back_to_rm

loop_run:
 call main_loop_SP ; SMC when the program is in dual CPU mode
MP_label equ $-4

; *** Misc. display ***


; read the Time Stamp Counter:
 RDTSC ; number in eax:edx
 mov ecx,[strip_count]
$base__ equ $-4
;;;; add [RANDOM],eax
;;;;$base__ equ $-4
;;;; add [RANDOM+4],edx
;;;;$base__ equ $-4
 div ecx ; might trigger "fatal error" if the number is too high
 mov [TSC],eax
$base__ equ $-4
 mov eax,dword [cycle_count]
$base__ equ $-4
 clc ; bug !...
 add eax,ecx
 jnc no_saturation2
 mov eax,-1
no_saturation2:
 push eax
 mov dword [cycle_count],eax
$base__ equ $-4

 call display_cycles

 pop eax
 cmp eax,[limit]
$base__ equ $-4
 jb loop_run

 jmp halte+1 ; skip the HLT, so we don't have
; to wait the next IRQ to exit

end_run:
 mov dword[es:back_to_rm-4],-6
 mov byte [es:pause_flag],1 ; pause
 push dword halte+1 ; skip the HLT, so we don't have
 jmp display_bar
; display_bar returns for us.


;*****************************************************************
;*                         CALIBRATION                           *
;*****************************************************************

calibration:
 mov dword[es:back_to_rm-4],-6 ; reclose the loop

 mov al,11111101b ; kbd only, if the computer hangs
 out 21h,al ; (mouse moves would spoil the results)

 mov eax,[es:mouse_y]
 push eax
 mov dword[es:mouse_y],YRESOLUTION-1 ; change the cursor position,
; because it is updated during the computation and takes +- 20000 cycles

 mov esi,XRESOLUTION-16 ; where to display the count
$video__ equ $-4
 mov ecx,-1 ; 'best_time'
; mov al,1 ; 'strip size' counter
; mov ah,1 ; 'best strip count'
 mov eax,101h
 lea ebx,[esi-16*10]

loop_inc_strip:
   mov [es:strip_count],al
   push eax
   push ebx
   push ecx
   push esi

   mov esi,0
$base__ equ $-4
   call modify_SMC

; unrolled loop:
     call main_loop_SP
MP_label2 equ $-4 ; SMC for dual CPU mode
     call main_loop_SP
MP_label3 equ $-4
     call main_loop_SP
MP_label4 equ $-4

; read the Time Stamp Counter:
   RDTSC ; number in eax:edx
   div dword [strip_count] ; might trigger "fatal error" if the number is too high
$base__ equ $-4

; now eax has the time/strip ratio.

; display the ratio
   pop esi
   mov edi,esi
   mov bl,10
   mov edx,0
   push eax
   call display_count
   pop eax

; is it worth ?
   pop ecx
   pop ebx
   cmp eax,ecx
   jae not_better
   mov ecx,eax ; the new best time

; erase the last mark:
   push edi
   mov edi,ebx
   mov edx,SPACE
   call display_number
   pop edi
   mov ebx,edi

; display the mark
   mov edx,PAUSE
   call display_number
   pop eax
   mov ah,al ; new best strip count
   jmp better

not_better:
   mov edx,SPACE
   call display_number
   pop eax
better:


   sub edi,byte 24
   push eax
   push ebx
   push ecx
   mov edx,0
   and eax,byte 63
   mov bl,2
   call display_count
   pop ecx
   pop ebx
   pop eax

   add esi,XRESOLUTION*23
   inc al
   cmp al,32
 jbe near loop_inc_strip

 mov esi,0
$base__ equ $-4
 mov [esi+strip_count],ah
 call modify_SMC
 mov al,11101101b ; mouse+kbd
 out 21h,al
 pop eax
 mov [esi+mouse_y],eax
```

```
 add dword [esi+cycle_count],(32*33)>>1

 mov byte [esi+pause_flag],1 ; go to pause mode
 push dword halte+1
 call display_bar



;*****************************************************************
;*                  SINGLE PROCESSOR MODE                        *
;*****************************************************************

align 16
main_loop_SP:

; reset the TSC:
 xor eax, eax
 mov ecx,16 ; TSC MSR
 mov edx,eax
 WRMSR

; eax: Y scan
; ecx: head pointer
;  bl: head parity
; edx: tail pointer
; esi: buffer pointer

; mov eax,0 ; Y scan variable (already zeroed for RDTSC)

;head:
 mov ecx,0
START_1 equ $-4 ;
 mov ebx,0 ; bl=parity
; tail:
 mov edx,0
START_2 equ $-4
 mov esi,0
START_BUFFER equ $-4

; flush the buffer:
 mov edi,esi
 pxor mm0,mm0 ; mm0=0
loop_flush_buffer:
 movq [edi+64],mm0
 movq [edi+72],mm0
 add edi,80
 cmp edi,dword 0
END_BUFFER equ $-4
 jbe loop_flush_buffer

; align 8
ext_loop5:



;*****************************************************************
;*                       COMPUTE FUNCTION                        *
;*****************************************************************

; eax: Y scan (not needed, pushed on the stack)-> shift counter (?)
; bl: parity, bh: current line parity
; ecx: head pointer
; edx: tail pointer
; ebp: X counter
; edi: Y counter
; ESI: circular buffer pointer

 pxor mm0,mm0
 mov bh,bl ; line parity
 movq [temp_D],mm0 ; clear the two consecutive dwords (temp_D and temp_E)
$base__ equ $-4
 movq [temp_E],mm0 ; again...  (temp_C and free)
$base__ equ $-4

 push eax
 push ecx
 push esi
 mov edi,ecx ; load head

external_loop:
   push edi
   mov ebp,0 ; horizontal loop counter
   xor bh,1
   push ebx
    jz near odd_loop
loop_compute1:


 call compute_collision
```

```
;*************** déplacement pair: *****************

;1
movq mm0,[XORF]
$base5__ equ $-4
pcmpeqb mm3,mm3
;2
movq mm1,[XORA]
$base5__ equ $-4
; ensuite:
por mm6,mm2     ;       STALL entrelacé....
;3
movq mm2,[XORB]
$base5__ equ $-4
por mm0,mm1
;4
pxor mm3,[WALLMASK]
$base5__ equ $-4
por mm0,mm2
;5
movq mm4,[byte edi+A]
por mm1,mm2
;6
movq [PA],mm7       ;t4 contenait PA
$base5__ equ $-4
por mm0,mm7
;7
movq [PB],mm6       ;t3 contenait PB
$base5__ equ $-4
pand mm0,mm3
;8
pxor mm4,mm0
;                   STALL
;9
movq mm0,[XORC]
$base5__ equ $-4
por mm1,mm6
;10
movq mm5,mm4
por mm1,mm0
;11
por mm2,mm0
movq mm6,mm4
;12
paddd mm5,[RANDOM]
$base5__ equ $-4
pand mm1,mm3
;13
por mm0,mm7
psrlq mm4,1
;14
movq mm7,[temp_B]
$base5__ equ $-4
psllq mm6,63
;15
;;;;movq [RANDOM],mm5
;;;;$base5__ equ $-4
pxor mm7,mm1      ;mm5 free
;16
movq mm1,[XORD]
$base5__ equ $-4
;                   STALL
;16'
por mm6,[byte edi+A-CEL] ; edi+A-CEL
;                   STALL
;17
movq [byte edi+A],mm4
por mm2,mm1
;18
movq [byte edi+A-CEL],mm6; edi+A-CEL
por mm0,mm2
;19
movq [XORD],mm2
$base5__ equ $-4
;                   STALL
;20
movq [byte esi+72],mm7 ; temp_B...
;                   STALL
;21
por mm2,[PC]
$base5__ equ $-4
;                   STALL
;22
movq mm4,[temp_C2]
$base5__ equ $-4
pand mm2,mm3
;23
por mm1,[PB]
$base5__ equ $-4
```

```
pxor mm4,mm2
;24
movq mm2,[XORE]
$base5__ equ $-4
movq mm6,mm4
;25
movd mm7,[temp_C]
$base5__ equ $-4
por mm0,mm2
;26
movq mm5,[byte edi+D]
psrlq mm4,63
;27
pand mm0,mm3
;                   STALL
;27'
por mm1,mm2
psllq mm6,1
;28
movd [temp_C],mm4
$base5__ equ $-4
pxor mm5,mm0
;29
movq mm0,[XORF]
$base5__ equ $-4
por mm6,mm7
;30
movq mm4,mm5
;                   STALL
;31
movq [byte esi+64],mm6 ;(C)
por mm1,mm0
;32
movq mm6,[byte edi+E]
por mm2,mm0
;33
movd mm7,[temp_D]
$base5__ equ $-4
psrlq mm4,63
;34
;;test al,[byte edi+VB0]       ; prefetch
psllq mm5,1
;35
movd [temp_D],mm4
$base5__ equ $-4
pand mm1,mm3
;36
movq mm0,mm2
por mm5,mm7
;37
por mm0,[XORD]
$base5__ equ $-4
pxor mm6,mm1
;38
por mm2,[PC]
$base5__ equ $-4
movq mm4,mm6
;39
movq mm1,[byte edi+F]
pand mm2,mm3
;40
pand mm3,mm0
;;test al,[edi+VB4]       ; prefetch
;41
por mm0,[PnOr]  ;
$base5__ equ $-4
pxor mm1,mm2
;42
movq [edi+D],mm5
psllq mm4,1
;43
pxor mm3,[byte edi+G]
psrlq mm6,63
;44
movq [edi+10000000],mm1
_LINE_F equ $-4
;                   STALL
;45
movd mm1,[temp_E]
$base5__ equ $-4
;                   STALL
;46
movd [temp_E],mm6
$base5__ equ $-4
por mm4,mm1
;                   STALL
;47
movq [byte edi+G],mm3
;                   STALL
;48
movq [edi+1000000],mm4; edi-LINE+E
_LINE_E equ $-4
;                   STALL

; sortie: COLL dans mm0



    add ebp,byte CEL
    add edi,byte CEL
    add esi,byte 80
    cmp ebp,0 ; line width
WIDTH equ $-4
    jb near loop_compute1
    jmp end_parity

;*************************************************************
;*************************************************************

odd_loop:
loop_compute2:

; register use:
; edi: data pointer (points to the current cell)
; esi: buffer pointer
; ebp: loop counter (line)
; edx: end pointer (last line to scan)
; ecx: free
;  bx: parity counter
; eax: free

call compute_collision

;************** déplacement impair: ****************

;1
movq mm0,[XORF]
$base5__ equ $-4
pcmpeqb mm3,mm3
;2
movq mm1,[XORA]
$base5__ equ $-4
; ensuite:
por mm6,mm2   ;  STALL entrelacé...
;3
movq mm2,[XORB]
$base5__ equ $-4
por mm0,mm1
;4
pxor mm3,[WALLMASK]
$base5__ equ $-4
por mm0,mm2
;5
movq mm4,[byte edi+A]
por mm1,mm2
;6
movq [PA],mm7
$base5__ equ $-4
por mm0,mm7
;7
movq [PB],mm6
$base5__ equ $-4
pand mm0,mm3
;8
por mm1,mm6
pxor mm4,mm0
;9
movq mm0,[XORC]
$base5__ equ $-4
movq mm5,mm4
;10
por mm1,mm0
movq mm6,mm4
;11
paddd mm5,[RANDOM]
$base5__ equ $-4
por mm2,mm0
;12
por mm0,mm7
;;;;;;;;;;;;test al,[byte edi+VB0]       ; prefetch
;13
movq mm7,[temp_B]
$base5__ equ $-4
pand mm1,mm3
;14
;;;; movq [RANDOM],mm5
;;;; $base5__ equ $-4
psllq mm6,63
```

```
;15
psrlq mm4,1
pxor mm7,mm1
;15'
por mm6,[byte edi+A-CEL]
movq mm5,mm7
;16
movq mm1,[XORD]
$base5__ equ $-4
psrlq mm7,1
;17
movq [byte edi+A],mm4
por mm2,mm1
;18
movq mm4,[temp_C2]
$base5__ equ $-4
por mm0,mm1
;19
movq [XORD],mm2
$base5__ equ $-4
;                   STALL
;20
por mm2,[PC]
$base5__ equ $-4
psllq mm5,63
;21
movq [byte edi+A-CEL],mm6
pand mm2,mm3
;22
movq [byte esi+72],mm7  ;B
pxor mm4,mm2
;22'
por mm5,[byte esi+72-80]
;                   STALL
;23
movq mm2,[XORE]
$base5__ equ $-4
;                   STALL
;24
movq [byte esi+72-80],mm5 ;B
por mm0,mm2
;25
movq [byte esi+64],mm4 ;(C)  esi+64
por mm1,mm2
;26
por mm2,[XORA]
$base5__ equ $-4
pand mm0,mm3
;26'
movq mm5,[byte edi+D]
;                   STALL
;27
por mm1,[PB]
$base5__ equ $-4
pxor mm5,mm0
;28
movq mm0,[XORF]
$base5__ equ $-4
movq mm4,mm5
;29
movq mm6,[byte edi+E]
por mm1,mm0
;30
movd mm7,[temp_D]
$base5__ equ $-4
por mm2,mm0
;31
;;;;;;;test al,[byte edi+VB4]       ; prefetch
movq mm0,mm2
;32
por mm2,[PC]
$base5__ equ $-4
pand mm1,mm3
;33
por mm0,[XORD]
$base5__ equ $-4
psrlq mm4,63
;34
pand mm2,mm3
psllq mm5,1
;35
pxor mm2,[byte edi+F]
pand mm3,mm0
;35'
por mm5,mm7
;            STALL
;36
por mm0,[PnOr]
$base5__ equ $-4
pxor mm6,mm1
;37
pxor mm3,[byte edi+G]
movq mm1,mm2
;38
movd [temp_D],mm4
$base5__ equ $-4
psllq mm2,63
;38'
movq [byte edi+D],mm5
psrlq mm1,1
;39
movq [dword edi+10000000],mm6
_LINE_E2 equ $-4
;                   STALL
;40
por mm2,[edi+1000000]
_LINE_F_CELL equ $-4
;                   STALL
;41
movq [edi+10000000],mm1
_LINE_F2 equ $-4
;                   STALL
;42
movq [byte edi+G],mm3
;                   STALL
;43
movq [edi+1000000],mm2
_LINE_F_CELL2 equ $-4


    add ebp,byte CEL
    add edi,byte CEL
    add esi,byte 80
    cmp ebp,0 ; line width
WIDTH2 equ $-4
    jb near loop_compute2

end_parity:
   pop ebx
   pop edi
   sub edi,0
WIDTH3 equ $-4
 cmp edi,edx ; compare tail
 jns near external_loop


 pop esi ; restore the pointer
 pop ecx
 pop eax



;*****************************************************************
;*  end of compute, pointer update                               *
;*****************************************************************

; if head=YSIZE
;   inc ESI,80
; else
;   inc head
;
; if head<strip_count
;   goto calcul
; display_line+cursor
; inc tail
; if tail<YSIZE
;   goto calcul

 cmp ecx,0
TRIGGER_1 equ $-4 ; (YSIZE*LINE)+mem
 jne inc_head ; good chances for successful prediction (big STRIP)
 add esi,1000000
XS32_80 equ $-4
 jmp short no_inc_head
inc_head:
 add ecx,0
WIDTH4 equ $-4
 xor bl,1
no_inc_head:

 cmp ecx,0
STRIP_1 equ $-4 ; (strip*line)+mem
 jbe near ext_loop5

 call display_line
DISPLAYLINE_SMC2 equ $-4

 inc eax
```

```
 add edx,0
WIDTH5 equ $-4
 cmp eax,0 ; YSIZE
YSIZE32_2 equ $-4
 jb near ext_loop5

; END OF THE SCAN

;*****************************

; RANDOM NUMBER GENERATOR UPDATE:
;1) load RANDOM
 movq mm0,[RANDOM]
$base1__ equ $-4

;2) copy before PANDN:
 movq mm1,mm0
;3) load the initial values:
 movq mm4,[RAND_A]
$base1__ equ $-4
 movq mm2,mm0
;4) mask out with PANDN (x1 !RANDOM)
 pandn mm1,mm4
 movq mm5,[RAND_B]
$base1__ equ $-4
;5) (x2 RANDOM)
 pand mm4,mm0
 movq mm3,mm0
 movq mm6,[RAND_C]
$base1__ equ $-4

 pandn mm2,mm5
 pand mm5,mm0
;6) result
 por mm4,mm2
 pandn mm3,mm6
 pand mm6,mm0
 por mm5,mm3
;7) write the result
;;;; movq [RAND_A],mm4
;;;; $base1__ equ $-4

 por mm6,mm1

;;;; movq [RAND_B],mm5
;;;; $base1__ equ $-4
;;;; movq [RAND_C],mm6
;;;; $base1__ equ $-4

 ret

align 16

;***********************************************



compute_collision:

; 1) modification lists:
    mov eax,[edi+WALL]
    pxor mm5,mm5 ; WALLMASK
    test eax,eax
    jz no_modifications1
      push ebx
      push edx
      mov ecx,[eax]
modification_loop1:

; 1) load the parameters
      movzx edx,word [eax+4]
      movzx ebx,word [eax+6]
      movq mm0,[eax+8] ; mask
      add eax,16

; 2) load the data to transform
      movq mm1,[edi+edx]
      movq mm3,mm0
      movq mm2,[edi+ebx]
      movq mm4,mm0

; 3) transform
      pandn mm3,mm1
      pandn mm4,mm2 ; negative mask

      pand mm1,mm0 ; positive mask
      pand mm2,mm0

      por mm1,mm4
      por mm2,mm3

    cmp eax,ecx
      por mm5,mm0

; 4) write back
      movq [edi+ebx],mm1
      movq [edi+edx],mm2

    jb modification_loop1
    pop edx
    pop ebx
no_modifications1:
    movq [WALLMASK],mm5
$base__ equ $-4


; pré-déplacement, popcount, détection.


; B:
     movq mm7,[byte esi+72]; temp B -> B (début)
;stall
     movq mm1,[byte edi+B] ; load B
;stall
     movq [byte edi+B],mm7; temp B -> B (fin)
;stall
     movq [temp_B],mm1  ;fin du fin
$base__ equ $-4
;stall

;C:
     movq mm0,[byte esi+64]; temp C -> C (début)
;stall
     movq mm4,[byte edi+C] ; load C
;stall
     movq [byte edi+C],mm0; temp C -> C (fin)
;stall
     movq [temp_C2],mm4 ;fin du fin
$base__ equ $-4
;stall


;popcount:

;en sortie:
;T3=G!=mm4 *
;T7=ST=mm1 *
;T0=D!=mm3 *
;T1=C!=mm7 *
;T2=B!=mm6 *

movq mm0,mm1  ; V
movq mm6,[byte edi+D]  ; U
pand mm1,mm4  ; HA1
movq mm5,[byte edi+E]  ; U
movq mm7,mm6
movq mm3,[byte edi+F]  ; U
pxor mm4,mm0  ; HA1
movq mm2,[byte edi+A]  ; U
pand mm6,mm5  ; HA3
pxor mm5,mm7  ; HA3
movq mm0,mm4
movq mm7,mm2  ; U
pand mm4,mm3  ; HA2
pand mm2,mm5  ; HA4
pxor mm3,mm0  ; HA2
pxor mm5,mm7  ; HA4
por mm1,mm4   ;
movq mm4,[byte edi+G]  ; =G original (copie pour le XOR ensuite)
por mm6,mm2
movq mm7,mm3
movq mm0,mm1
pand mm3,mm5  ; HA5
pand mm1,mm6  ; HA6
pxor mm5,mm7  ; HA5
movq mm2,mm4   ; copie de G pour le calcul
pxor mm6,mm0  ; HA6
movq mm7,mm5
movq mm0,mm6  ; U
pand mm5,mm4 ; HA8
pand mm6,mm3  ; HA7
pxor mm3,mm0  ; HA8
por mm1,mm6   ;     =ST
pxor mm2,mm7  ; HA8 (G xor T1->mm6, T3=G)
 movq mm6,[temp_B]  ; T2 = B original
$base5__ equ $-4
movq mm7,mm3 ;!!!
pand mm3,mm5  ; HA9
```

```
pxor mm5,mm7  ; HA9 T1 free
 movq mm7,[temp_C2]  ; T1 = C original
$base5__ equ $-4
por mm3,mm1   ; last computation for S2
movq [S0],mm2 ; T6 free
$base5__ equ $-4
 pxor mm4,mm1  ; T3=G!    ****
movq [S1],mm5 ; T5 free
$base5__ equ $-4
 pxor mm6,mm1  ; B!
movq [S2],mm3 ; T0 free
$base5__ equ $-4
 pxor mm7,mm1  ; C!
 movq mm3,[byte edi+D]  ; D
movq mm5,mm7
movq [ST],mm1
$base5__ equ $-4
 pxor mm3,mm1 ; D!

;T3=G!=mm4
;T7=ST=mm1
;T0=D!=mm3
;T1=C!=mm7
;T2=B!=mm6
;T4=mm0
;T5=mm5
;T6=mm2


; détection, acte 1:

movq mm2,mm7       ; U
movq mm5,mm7       ; V

pxor mm6,mm3       ;U1
pxor mm2,mm3       ;V2
movq mm1,mm4       ;U3
pandn mm5,mm6      ;V4
pandn mm6,mm2      ;U5
pxor mm1,mm3       ;V6
movq mm0,mm6       ;U7
pand mm6,mm1       ;V8
movq mm2,[RAND_C]  ;U9
$base5__ equ $-4
pandn mm1,mm0      ;V10
movq mm0,[RANDOM]  ;U11
$base5__ equ $-4
pand mm5,mm4       ;V12
pxor mm0,mm3       ;U13
pand mm2,mm1       ;V14
pand mm1,mm7       ;U15
pand mm0,mm5       ;V16
movq [XORC],mm6    ;U17
$base5__ equ $-4
por mm2,mm0        ;V18
pand mm6,[RANDOM]  ;U19
$base5__ equ $-4
pcmpeqb mm0,mm0    ;V20
movq [PC],mm1      ;U21
$base5__ equ $-4
por mm2,mm6        ;V22
   movq mm6,[byte edi+E]   ;U23
pxor mm0,mm2       ;V24
   pxor mm6,[ST]   ;U25
$base5__ equ $-4
   movq mm2,mm3    ;V26
movq [XORF],mm0    ;U27
$base5__ equ $-4
   movq mm5,mm3    ;V28

;T1=mm0
;T2=mm1
;T3=mm2
;T4=mm5
;_G=mm4
;_RA=mm6
;_RB=mm3
;AXC=mm7

pxor mm7,mm6       ;U1
pxor mm2,mm6       ;V2
movq mm1,mm4       ;U3
pandn mm5,mm7      ;V4
pandn mm7,mm2      ;U5
pxor mm1,mm6       ;V6
movq mm0,mm7       ;U7
pand mm7,mm1       ;V8
movq mm2,[RAND_A]  ;U9
$base5__ equ $-4
pandn mm1,mm0      ;V10
movq mm0,[RANDOM]  ;U11
$base5__ equ $-4
pand mm5,mm4       ;V12
pxor mm0,mm6       ;U13
pand mm2,mm1       ;V14
pand mm1,mm3       ;U15
pand mm0,mm5       ;V16
movq [XORD],mm7    ;U17
$base5__ equ $-4
por mm2,mm0        ;V18
pand mm7,[RANDOM]  ;U19
$base5__ equ $-4
pcmpeqb mm0,mm0    ;V20
movq [PA],mm1      ;U21
$base5__ equ $-4
por mm2,mm7        ;V22
   movq mm7,[byte edi+F]   ;U23
pxor mm0,mm2       ;V24
   pxor mm7,[ST]   ;U25
$base5__ equ $-4
   movq mm2,mm6    ;V26
movq [XORA],mm0    ;U27
$base5__ equ $-4
   movq mm5,mm6    ;V28

;T1=mm0
;T2=mm1
;T3=mm2
;T4=mm5
;_G=mm4
;_RA=mm7
;_RB=mm6
;AXC=mm3

pxor mm3,mm7          ;U1
pxor mm2,mm7          ;V2
movq mm1,mm4          ;U3
pandn mm5,mm3         ;V4
pandn mm3,mm2         ;U5
pxor mm1,mm7          ;V6
movq mm0,mm3          ;U7
pand mm3,mm1          ;V8
movq mm2,[RAND_B]     ;U9
$base5__ equ $-4
pandn mm1,mm0         ;V10
movq mm0,[RANDOM]     ;U11
$base5__ equ $-4
pand mm5,mm4          ;V12
pxor mm0,mm7          ;U13
pand mm2,mm1          ;V14
pand mm1,mm6          ;U15
pand mm0,mm5          ;V16
movq [XORE],mm3       ;U17
$base5__ equ $-4
por mm2,mm0           ;V18
pand mm3,[RANDOM]     ;U19
$base5__ equ $-4
pcmpeqb mm0,mm0       ;V20
movq [PB],mm1         ;U21
$base5__ equ $-4
por mm2,mm3           ;V22
pxor mm0,mm2          ;U23
;              STALL

; détection, acte 2:

;T1=mm0
;T2=mm1
;T3=mm2
;T4=mm5
;_G=mm4
;_RA=mm3
;_RB=mm7
;AXC=mm6

movq mm3,[byte edi+A] ;U1
movq mm2,mm7          ;V2
movq [XORB],mm0       ;U3
$base5__ equ $-4
movq mm1,mm4          ;V4
pxor mm3,[ST]         ;U5
$base5__ equ $-4
movq mm5,mm7          ;V6
pxor mm6,mm3          ;U7
pxor mm2,mm3          ;V8
pxor mm1,mm3          ;U9
pandn mm5,mm6         ;V10
pandn mm6,mm2         ;U11
```

```
movq mm0,mm1          ;V12
movq mm2,[RANDOM]     ;U13
$base5__ equ $-4
pandn mm1,mm6         ;V14
pand mm6,mm0          ;U15
pand mm5,mm4          ;V16
pand mm6,[XORF]       ;U17
$base5__ equ $-4
pxor mm2,mm3          ;V18
movq mm0,[RAND_C]     ;U19
$base5__ equ $-4
pand mm2,mm5          ;V20
movq [XORF],mm6       ;U21
$base5__ equ $-4
pand mm0,mm3          ;V22
movq mm5,[RANDOM]     ;U23
$base5__ equ $-4
pand mm1,mm7          ;V24
pand mm1,[PC]         ;U25
$base5__ equ $-4
por mm0,mm2           ;V26
pandn mm5,mm6         ;U27
;                    STALL V28
movq [PC],mm1         ;U29
$base5__ equ $-4
por mm5,mm0           ;V30
pandn mm5,[XORC]      ;U31
$base5__ equ $-4
;                    STALL V32

;T1=mm0
;T2=mm1
;T3=mm2
;T4=mm5
;_G=mm4
;_RA=mm6
;_RB=mm3
;AXC=mm7

movq mm6,[temp_B]     ;U1
$base5__ equ $-4
movq mm2,mm3          ;V2
movq [XORC],mm5       ;U3
$base5__ equ $-4
movq mm1,mm4          ;V4
pxor mm6,[ST]         ;U5
$base5__ equ $-4
movq mm5,mm3          ;V6
pxor mm7,mm6          ;U7
pxor mm2,mm6          ;V8
pxor mm1,mm6          ;U9
pandn mm5,mm7         ;V10
pandn mm7,mm2         ;U11
movq mm0,mm1          ;V12
movq mm2,[RANDOM]     ;U13
$base5__ equ $-4
pandn mm1,mm7         ;V14
pand mm7,mm0          ;U15
pand mm5,mm4          ;V16
pand mm7,[XORA]       ;U17
$base5__ equ $-4
pxor mm2,mm6          ;V18
movq mm0,[RAND_A]     ;U19
$base5__ equ $-4
pand mm2,mm5          ;V20
movq [XORA],mm7       ;U21
$base5__ equ $-4
pand mm0,mm6          ;V22
movq mm5,[RANDOM]     ;U23
$base5__ equ $-4
pand mm1,mm3          ;V24
pand mm1,[PA]         ;U25
$base5__ equ $-4
por mm0,mm2           ;V26
pandn mm5,mm7         ;U27
;                    STALL V28
movq [PA],mm1         ;U29
$base5__ equ $-4
por mm5,mm0           ;V30
pandn mm5,[XORD]      ;U31 ;XORA
$base5__ equ $-4
;                    STALL V32

;T1=mm0
;T2=mm1
;T3=mm2
;T4=mm5
;_G=mm4
;_RA=mm7
;_RB=mm6
;AXC=mm3

movq mm7,[temp_C2]    ;U1
$base4__ equ $-4
movq mm2,mm6          ;V2
movq [XORD],mm5       ;U3
$base5__ equ $-4
movq mm1,mm4          ;V4
pxor mm7,[ST]         ;U5
$base4__ equ $-4
movq mm5,mm6          ;V6
pxor mm3,mm7          ;U7
pxor mm2,mm7          ;V8
pxor mm1,mm7          ;U9
pandn mm5,mm3         ;V10
pandn mm3,mm2         ;U11
movq mm0,mm1          ;V12
movq mm2,[RANDOM]     ;U13
$base4__ equ $-4
pandn mm1,mm3         ;V14
pand mm3,mm0          ;U15
pand mm5,mm4          ;V16
pand mm3,[XORB]       ;U17
$base4__ equ $-4
pxor mm2,mm7          ;V18
movq mm0,[RAND_B]     ;U19
$base4__ equ $-4
pand mm2,mm5          ;V20
movq [XORB],mm3       ;U21
$base4__ equ $-4
pand mm0,mm7          ;V22
movq mm5,[RANDOM]     ;U23
$base4__ equ $-4
pand mm1,mm6          ;V24
pand mm1,[PB]         ;U25
$base4__ equ $-4
por mm0,mm2           ;V26
movq mm2,mm5          ;U27
pandn mm5,mm3         ;V28
movq mm4,[PC]     ; (entrelacé)
$base4__ equ $-4
por mm5,mm0           ;V30
pandn mm5,[XORE]      ;U31
$base4__ equ $-4


;*************** sélection progressive: ***************
;entrée:
; PB dans mmB   (mm1)->T2
; RAND dans mmC (mm2)*
;1 registre inutilisé (mm3)

;1
;->U31
movq mm6,mm1         ;V32 (entrelacé)
;2
movq mm7,[PA]
$base4__ equ $-4
por mm1,mm4
;3
movq [XORE],mm5     ;U3 (entrelacé avec la détection)
$base4__ equ $-4
por mm1,mm7
;4
movq mm5,mm4
movq mm0,mm2
;5
movq [PnOr],mm1
$base4__ equ $-4
pandn mm0,mm1
;6
pand mm1,mm2
movq mm2,mm7
;7
por mm4,mm1
pand mm5,mm1
;8
por mm7,mm0
pand mm2,mm0
;9
movq [PC],mm4
$base4__ equ $-4
por mm6,mm5


ret
```

```
;;*****************************************************************
;;*                      MASTER PROCESSOR MODE                    *
;;*****************************************************************
;
;align 16
;
;main_loop_MP:


;*****************************************************************
;*                      LINE DISPLAY FUNCTIONS                   *
;*****************************************************************


;*****************************************************************
;*                           NORMAL SIZE                         *
;*****************************************************************

align 16

display_line: ;_4: ; normal size (1:1)

; eax: line # (input)
; edi: video line address
; ebp: col. counter
; edx: pointer to the line to display (input)
; MMX: trashed

; esi: pushed, but used: temp buffer

 push esi
 mov edi,eax
 push edx
 add edx,[orig_x] ; horizontal scroll
$base1__ equ $-4
 sub edi,[orig_y] ; vertical scroll
$base1__ equ $-4
 jb near end_display_line ; no display yet
 cmp edi,YRESOLUTION-32
 jae near end_display_line ; too late to display

 movq mm1,[color_mask] ; white color mask
$base1__ equ $-4
 pxor mm2,mm2 ; 0
 movq mm3,[shifted_mask] ; 0102040810204080h
$base1__ equ $-4

 shl edi,10
 xor ebp,ebp
 add edi,0
$video__ equ $-4

 push eax
 push ecx
loop_inc_4:
   pxor mm0,mm0
   por mm0,[edx+A]
   por mm0,[edx+B]
   por mm0,[edx+C]
   por mm0,[edx+D]
   por mm0,[edx+E]
   por mm0,[edx+F]
   por mm0,[edx+G]

;   movq mm0,[esi]
;   mov dword [esi], 0
;   mov dword [esi+4],0

%ifdef display_ptrs
   pxor mm0,[edx+WALL]
%endif

%ifdef display_walls
   mov eax,[edx+WALL]
   test eax,eax
   jz no_modifications3
    mov ecx,[eax] ; pointer to the end of the list
modification_loop3:
    por mm0,[eax+8]
    add eax,16
    cmp eax,ecx
   jb modification_loop3
no_modifications3:
%endif

%ifdef display_dots
   mov eax,[edx+WALL]
   test eax,eax
   jz no_modifications3
    mov ecx,[eax] ; pointer to the end of the list
    push ebx
    mov ebx,0
    pcmpeqb mm4,mm4
modification_loop3:
    cmp dword [eax+4],A+(D<<16)
    jne no_mask_A
    pand mm4,[eax+8] ; mask
    or ebx, byte 1
no_mask_A:
    cmp dword [eax+4],B+(E<<16)
    jne no_mask_B
    pand mm4,[eax+8] ; mask
    or ebx, byte 2
no_mask_B:
    cmp dword [eax+4],C+(F<<16)
    jne no_mask_C
    pand mm4,[eax+8] ; mask
    or ebx, byte 4
no_mask_C:
    add eax,16
    cmp eax,ecx
   jb modification_loop3
   cmp ebx,7
   jne no_por_mm4
   por mm0,mm4
no_por_mm4:
   pop ebx
no_modifications3:
%endif

   movq mm4,mm0
   add edx,byte CEL

   add esi,80

   punpcklbw mm4,mm4
   punpcklwd mm4,mm4
   punpckldq mm4,mm4
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp],mm4

   psrlq mm0,8
   movq mm4,mm0
   punpcklbw mm4,mm4
   punpcklwd mm4,mm4
   punpckldq mm4,mm4
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp+8],mm4

   psrlq mm0,8
   movq mm4,mm0
   punpcklbw mm4,mm4
   punpcklwd mm4,mm4
   punpckldq mm4,mm4
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp+16],mm4

   psrlq mm0,8
   movq mm4,mm0
   punpcklbw mm4,mm4
   punpcklwd mm4,mm4
   punpckldq mm4,mm4
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp+24],mm4

   psrlq mm0,8
   movq mm4,mm0
   punpcklbw mm4,mm4
   punpcklwd mm4,mm4
   punpckldq mm4,mm4
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp+32],mm4

   psrlq mm0,8
   movq mm4,mm0
   punpcklbw mm4,mm4
   punpcklwd mm4,mm4
   punpckldq mm4,mm4
```

```
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp+40],mm4

   psrlq mm0,8
   movq mm4,mm0
   punpcklbw mm4,mm4
   punpcklwd mm4,mm4
   punpckldq mm4,mm4
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp+48],mm4

   psrlq mm0,8
   movq mm4,mm0
   punpcklbw mm4,mm4
   punpcklwd mm4,mm4
   punpckldq mm4,mm4
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp+56],mm4

   add ebp,byte 64 ; V
   cmp ebp,0 ;XSIZE  ; U
XSIZE32_5 equ $-4
 jb near loop_inc_4    ; V
 pop ecx
 pop eax

end_display_line:
 pop edx
 pop esi

; refresh the display elements

do_display_stuff:
; mouse cursor
 cmp eax,[tunnel_y] ; u
$base1__ equ $-4
 je update_cursor_4 ; V, often correctly predicted
 ret
update_cursor_4:
   push ebx
   mov ebx,[mouse_x]
$base1__ equ $-4
   cmp ebx,0 ; XSIZE ; SIZE SMC
XSIZE32_6 equ $-4
   ja no_update_cursor2_4
     push eax
       mov eax,[mouse_y]
$base1__ equ $-4
       mov ebx,[mouse_x]
$base1__ equ $-4
       call display_cursor
     pop eax
no_update_cursor2_4:
   pop ebx
 ret

;*****************************************************************
;*                            HALF SIZE                          *
;*****************************************************************

display_line2: ; half size (2:1)
; eax: line #
; edi: video line address
; ebp: col. counter
; edx: pointer to the line to display

 test eax,1
 jz do_display_even
 ret
do_display_even:
 mov edi,eax
 push edx
 add edx,[orig_x] ; horizontal scroll
$base1__ equ $-4
 sub edi,[orig_y] ; vertical scroll
$base1__ equ $-4
 jb near end_display_line_2 ; no display yet
 cmp edi,(YRESOLUTION-32)*2
 jae near end_display_line_2 ; too late to display

 shl edi,9 ; => (*1024)/2

 movq mm1,[color_mask] ; white color mask
$base1__ equ $-4
 pxor mm2,mm2 ; 0
 movq mm3,[shifted_mask2] ; 0104104001041040h
$base1__ equ $-4

 xor ebp,ebp
 add edi,0
$video__ equ $-4
; align 8

loop_inc_5:
   movq mm0,[edx+A]
   por mm0,[edx+B]
   por mm0,[edx+C]
   por mm0,[edx+D]
   por mm0,[edx+E]
   por mm0,[edx+F]
   por mm0,[edx+G]
   por mm0,[edx+1000]
XSIZE32CELA equ $-4
   por mm0,[edx+1000]
XSIZE32CELD equ $-4

   movq mm4,mm0
   psrlq mm0,1
   por mm0,mm4
   pand mm0,[even_mask]
$base1__ equ $-4

   movq mm4,mm0
   add edx,byte CEL
   punpcklwd mm4,mm4
   punpckldq mm4,mm4
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp],mm4

   psrlq mm0,16
   movq mm4,mm0
   punpcklwd mm4,mm4
   punpckldq mm4,mm4
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp+8],mm4

   psrlq mm0,16
   movq mm4,mm0
   punpcklwd mm4,mm4
   punpckldq mm4,mm4
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp+16],mm4

   psrlq mm0,16
   movq mm4,mm0
   punpcklwd mm4,mm4
   punpckldq mm4,mm4
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp+24],mm4

   add ebp,byte 32 ; V
   cmp ebp,0 ;XSIZE  ; U
XSIZE32_7 equ $-4
 jb near loop_inc_5    ; V

end_display_line_2:
 pop edx

; refresh the display elements
 jmp do_display_stuff

;*****************************************************************
;*                              1/4 SIZE                         *
;*****************************************************************

align 16

display_line4: ; zoom 4:1
; eax: line #
; edi: video line address
; ebp: col. counter
; edx: pointer to the line to display

 test eax,3
```

```
 jz do_display_even_even
 ret

do_display_even_even:
 mov edi,eax
 push edx
 add edx,[orig_x] ; horizontal scroll
$base1__ equ $-4
 sub edi,[orig_y] ; vertical scroll
$base1__ equ $-4
 jb near end_display_line_3 ; no display yet
 cmp edi,(YRESOLUTION-32)*4
 jae near end_display_line_3 ; too late to display

 shl edi,8 ; => (*1024)/4

 movq mm1,[color_mask] ; white color mask
$base1__ equ $-4
 pxor mm2,mm2 ; 0
 movq mm3,[shifted_mask3] ;
$base1__ equ $-4

 xor ebp,ebp
 add edi,0
$video__ equ $-4
; align 8

loop_inc_6:
   movq mm0,[edx+A]
   por mm0,[edx+B]
   por mm0,[edx+C]
   por mm0,[edx+D]
   por mm0,[edx+E]
   por mm0,[edx+F]
   por mm0,[edx+G]
;   por mm0,[edx+1000]
;XSIZE32CELA equ $-4
;   por mm0,[edx+1000]
;XSIZE32CELD equ $-4

   movq mm4,mm0
   psrlq mm0,1
   por mm0,mm4
   pand mm0,[even_mask]
$base1__ equ $-4
   movq mm4,mm0
   psrlq mm0,2
   por mm0,mm4
   pand mm0,[even_even_mask]
$base1__ equ $-4

   movq mm4,mm0
   add edx,byte CEL
   punpckldq mm4,mm4
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp],mm4

   psrlq mm0,32
   movq mm4,mm0
   punpckldq mm4,mm4
   pand mm4 ,mm3    ; select a byte
   pcmpeqb mm4,mm2 ; if (byte) not zero then 0FFh
   pandn mm4,mm1  ; background mask
   movq [edi+ebp+8],mm4

   add ebp,byte 16 ; V
   cmp ebp,0 ;XSIZE  ; U
XSIZE32_9 equ $-4
 jb near loop_inc_6    ; V

end_display_line_3:
 pop edx

; refresh the display elements
 jmp do_display_stuff

;*********************************************************************
;*                    SAVE THE SCREEN TO A BMP FILE                  *
;*********************************************************************

bits 16
align 16

save_screen:
; IRQ off assumed here (it is within the mouse interrupt !)
 pushad

; display a WAIT sign... (it lasts about 20 seconds)
 mov ecx,display_number
 mov edx,SANDPILE
 mov edi,(XRESOLUTION*(YRESOLUTION-24))+32
$video__ equ $-4
 call SEL_CODE_32:return_relay

; switch to real mode: (copy-paste-modify of the stub code)

; the "cached part(s)":
 mov ax,SEL_FLAT
 mov fs,ax ; as to still access the video mem in "flat mode"
 mov ax,SEL_REAL
 mov ds,ax
 mov es,ax
; ss ??? I hope MSDOS doesn't change ss....
; mov sp,MP_stack_top !!! we keep the stack !

;SWITCHES TO REAL MODE
 mov eax,cr0
 and al,0FEh ; reset the PE bit
 mov cr0,eax
; flush the pipeline:
 db 0EAh     ;opcode for a FAR JMP to the 16b code
 dw back_2_real2
cs3: dw 0 ;will have been set by a previous instruction
back_2_real2:

;******** Here we are in Real Mode: **********
 mov ax,cs
 mov ds,ax
 mov ss,ax
 mov es,ax

; restores the DOS environment

 lidt [DOS_IDT] ;restore the DOS' IDT
 xor al,al
 out 70h,al ;NMI mask removed
 mov al,[old_mask]
 out 21h,al ; restores the PIC state
; sti

;*************** here we are ready ! ***********************

; say "hello" !
; mov edi,dword 0
;$video__ equ $-4
; movq mm0,[shifted_mask]
; movq [fs:edi],mm0

; increment the file number:

increment_name:
 mov di,endfilename
loop_inc_name:
 mov al,[di]
 inc al
 mov [di],al
 cmp al,'9'
  jbe no_inc_name
   mov byte [di],'0'
   dec di ; buug ! inc instead of dec
   cmp di,filename+1
   je near exit_bmp ; already one million files...
   jmp short loop_inc_name

no_inc_name:

; find the corresponding file: (thanks to the RBIL !)
;INT 21 - DOS 2+ - GET FILE ATTRIBUTES
;       AX = 4300h
;       DS:DX -> ASCIZ filename
;Return: CF clear if successful
;           CX = file attributes (see #1107)
;           AX = CX (DR DOS 5.0)
;       CF set on error
;           AX = error code (01h,02h,03h,05h) (see #1366 at AH=59h)
 mov ax,4300h
 mov dx,filename
 int 21h
; continue if already exists
 jnc increment_name

;create the file:
;INT 21 - DOS 2+ - "CREAT" - CREATE OR TRUNCATE FILE
;       AH = 3Ch
;       CX = file attributes (see #1089)->nothing interesting
;       DS:DX -> ASCIZ filename
```

```
;Return: CF clear if successful
;           AX = file handle
;       CF set on error
;           AX = error code (03h,04h,05h) (see #1366 at AH=59h/BX=0000h)
 mov ah,3Ch
 xor cx,cx
; dx not modified after the last DOS call...
 int 21h
 jc near exit_bmp ; just in case...

;INT 21 - DOS 2+ - "WRITE" - WRITE TO FILE OR DEVICE
;       AH = 40h
;       BX = file handle
;       CX = number of bytes to write
;       DS:DX -> data to write
;Return: CF clear if successful
;           AX = number of bytes actually written;
;       CF set on error
;           AX = error code (05h,06h) (see #1366 at AH=59h/BX=0000h)
 mov bx,ax
 mov ah,40h
 mov cx,(end_header-bmp_header)
 mov dx,bmp_header
 int 21h
 jc near exit_bmp

; converts the palette to BMP format
; uses di,ax,es,dx,cx
 xor al,al
 mov dx,03c7h ; points the VGA register to the palette
 out dx,al
 mov dl,0C9h ; and now, read the bytes.
 mov di,BMP_buffer
loop_read_palette:
   in al,dx
   shl al,2
   mov [di+2],al
   in al,dx
   shl al,2
   mov [di+1],al ; buug !!! reversed RVB-BVR
   in al,dx
   shl al,2
   mov [di],al
   add di,4
   cmp di,BMP_buffer+1024
 jnz loop_read_palette

; write the palette
 mov ah,40h
 mov cx,1024
 mov dx,BMP_buffer
 int 21h
 jc exit_bmp

; Write the image data:
 mov bp,YRESOLUTION
 mov esi,XRESOLUTION*(YRESOLUTION-1)
$video__ equ $-4
 movq mm1,[all_ones]

loop_write_BMP:

; copy the line to Rmem:
  mov di,BMP_buffer
loop_copy_rmem:
   movq mm0,[fs:esi]
   movq [di],mm0
   add esi,byte 8
   add di,byte 8
   cmp di,BMP_buffer+1024
  jb loop_copy_rmem

; write the line:
  mov dx,BMP_buffer
  mov ah,40h
  mov cx,1024
  int 21h
  jc exit_bmp

  sub esi,2*XRESOLUTION
  dec bp
 jnz loop_write_BMP

;close the file:
;INT 21 - DOS 2+ - "CLOSE" - CLOSE FILE
;       AH = 3Eh
;       BX = file handle
;Return: CF clear if successful
;           AX destroyed
;       CF set on error
;           AX = error code (06h) (see #1366 at AH=59h/BX=0000h)
;Notes: if the file was written to, any pending disk writes are performed, the
;         time and date stamps are set to the current time, and the directory
;         entry is updated
 mov ah,3Eh ; buuuugg ! ax instead of ah
; bx unmodified...
 int 21h

exit_bmp:

;****************** switch back to pmode: **********************
;disable the interrupts
 cli
 mov al,80h
 out 70h,al ;set the NMI mask
 mov al,11101101b ; mouse+kbd
 out 21h,al ;enable the COM1 + KBD interrupts in the main PIC
 lidt [NEW_SIZE] ; then load ours

; will be popped later by RETFD at the end of the mode change:
 push word SEL_CODE_16
 push word back_from_screenshot
 jmp switch_PM32 ; entry point for the AP to switch to PMODE

back_from_screenshot:
 popad
 ret

; the needed data:
filename: db 'IM00000'
endfilename: db '0'-1,'.BMP',0

bmp_header:
bmp_ident: db 'BM'
bmp_file_size: dd (XRESOLUTION*YRESOLUTION)+(end_header-bmp_header)+1024 ;
bmp_reserved:  dd 0
bmp_bit_offset:dd (end_header-bmp_header)+1024;
bmp_biSize:    dd 40; "??? but as long as it works...."
bmp_biWidth:   dd XRESOLUTION;
bmp_biHeight:  dd YRESOLUTION;
bmp_biPlanes:  dw 1;
bmp_biBitCount:dw 8; 256 colors
bmp_biCompr:   dd 0; uncompressed ...
bmp_biSizeImg: dd XRESOLUTION*YRESOLUTION;
bmp_biXPixelPerM:dd 0;
bmp_biYPixelPerM:dd 0; not useful.
bmp_biClrUsed: dd 0;
bmp_biClrImpr: dd 0;
end_header:



;*****************************************************************
;*                          MMX CONSTANTS                        *
;*****************************************************************

align 32 ; Even though MSDOS manages its memory with 16-byte paragraphs...
; solution: recopier le bloc !
XORA: dd 0,0
XORB: dd 0,0
XORC: dd 0,0
XORD: dd 0,0
XORE: dd 0,0
XORF: dd 0,0
PnOr: dd 0,0
PA: dd 0,0
PB: dd 0,0
PC: dd 0,0
temp_D: dd 0
temp_C: dd 0
temp_E: dd 0
free:   dd 0 ; only for the alignment
temp_C2: dd 0,0
temp_B: dd 0,0
RANDOM: dd -1,-1 ; 123456789,987654321 ; or whatever you want...
RAND_A: dd -1,-1
RAND_B: dd -1,-1
RAND_C: dd 0,0
WALLMASK: dd 0,0
ST: dd 0,0
S0: dd 0,0
S1: dd 0,0
S2: dd 0,0

mouse_x: dd XRESOLUTION-1  ; middle of the screen
mouse_y: dd YRESOLUTION-1
orig_x: dd 0
orig_y: dd 0
```

```
color_mask:
 db MOUSE_COLOR,MOUSE_COLOR,MOUSE_COLOR,MOUSE_COLOR
 db MOUSE_COLOR,MOUSE_COLOR,MOUSE_COLOR,MOUSE_COLOR
even_mask: dd 55555555h,55555555h
even_even_mask: dd 11111111h,11111111h

shifted_mask:
 db 00000001b
 db 00000010b
 db 00000100b
 db 00001000b
 db 00010000b
 db 00100000b
 db 01000000b
 db 10000000b

shifted_mask2:
 db 00000001b
 db 00000100b
 db 00010000b
 db 01000000b
 db 00000001b
 db 00000100b
 db 00010000b
 db 01000000b

shifted_mask3:
 db 00000001b
 db 00010000b
 db 00000001b
 db 00010000b
 db 00000001b
 db 00010000b
 db 00000001b
 db 00010000b

bg_color:
 db BACKGROUND,BACKGROUND,BACKGROUND,BACKGROUND
 db BACKGROUND,BACKGROUND,BACKGROUND,BACKGROUND
number_foreground:
 db NUMBER_FG,NUMBER_FG,NUMBER_FG,NUMBER_FG
 db NUMBER_FG,NUMBER_FG,NUMBER_FG,NUMBER_FG
number_background:
 db NUMBER_BG,NUMBER_BG,NUMBER_BG,NUMBER_BG
 db NUMBER_BG,NUMBER_BG,NUMBER_BG,NUMBER_BG
all_ones: dd -1,-1

;*********************************************************************
;*                          MOUSE CURSOR                             *
;*********************************************************************

CURSOR_LINES equ 29
cursor:
dd 00000000000000000011000000000000b ; 29
dd 00000000000000000111100000000000b ; 28
dd 00000000000000001111100000000000b ; 27
dd 00000000000000011111100000000000b ; 26
dd 00000000000011111111000000000000b ; 25
dd 00000000000111111111000000000000b ; 24
dd 00000000011111111110000000000000b ; 23
dd 00000000111111111100000000000000b ; 22
dd 00000001111111111000000000000000b ; 21
dd 00000001111111111000000000000000b ; 20
dd 00000011111111110000000000000000b ; 19
dd 00000111111111100000000000000000b ; 18
dd 00000111111111100000000000000000b ; 17
dd 00001111111111000000000000000000b ; 16
dd 00001111111110000000000000000000b ; 15
dd 00011111111000000000000000000000b ; 14
dd 00011111111000011111110000000000b ; 13
dd 00111111110011111111111110000000b ; 12
dd 00111111111111111111111111110000b ; 11
dd 01111111111111111111111111111000b ; 10
dd 01111111111111111111111111111100b ;  9
dd 01111111111111111111111111111110b ;  8
dd 01111111111111111111111000011111b ;  7
dd 11111111111111000000000000000001b ;  6
dd 11111111110000000000000000000000b ;  5
dd 11111110000000000000000000000000b ;  4
dd 11110000000000000000000000000000b ;  3
dd 11000000000000000000000000000000b ;  2
dd 10000000000000000000000000000000b ;  1

;*********************************************************************
;*                            21*16 FONT                             *
;*********************************************************************

FONT_HEIGHT equ 21
font_numbers:
                                                                        ;0
                                                                        dw 0000000000000000b ;1
                                                                        dw 0000001111000000b ;2
                                                                        dw 0000111001110000b ;3
                                                                        dw 0001100000011000b ;4
                                                                        dw 0011000000001100b ;5
                                                                        dw 0110000000000100b ;6
                                                                        dw 0110000000000110b ;7
                                                                        dw 0110000000000110b ;8
                                                                        dw 1110000000000111b ;9
                                                                        dw 1110000000000111b ;10
                                                                        dw 1110000000000111b ;11
                                                                        dw 1100000000000111b ;12
                                                                        dw 1100000000000111b ;13
                                                                        dw 1110000000000110b ;14
                                                                        dw 0110000000000110b ;15
                                                                        dw 0110000000001110b ;16
                                                                        dw 0011000000001100b ;17
                                                                        dw 0011100000011100b ;18
                                                                        dw 0001110000111000b ;19
                                                                        dw 0000111111110000b ;20
                                                                        dw 0000001111000000b ;21

                                                                        ;1
                                                                        dw 0000000110000000b ;1
                                                                        dw 0000000111000000b ;2
                                                                        dw 0000000111100000b ;3
                                                                        dw 0000000111110000b ;4
                                                                        dw 0000000110111100b ;5
                                                                        dw 0000000110000000b ;6
                                                                        dw 0000000110000000b ;7
                                                                        dw 0000000110000000b ;8
                                                                        dw 0000000110000000b ;9
                                                                        dw 0000000110000000b ;10
                                                                        dw 0000000110000000b ;11
                                                                        dw 0000000110000000b ;12
                                                                        dw 0000000110000000b ;13
                                                                        dw 0000000110000000b ;14
                                                                        dw 0000000110000000b ;15
                                                                        dw 0000000110000000b ;16
                                                                        dw 0000000110000000b ;17
                                                                        dw 0000000110000000b ;18
                                                                        dw 0000001111000000b ;19
                                                                        dw 0000001111000000b ;20
                                                                        dw 0000111111110000b ;21

                                                                        ;2
                                                                        dw 0000001111110000b ;1
                                                                        dw 0000011111111000b ;2
                                                                        dw 0000111100001100b ;3
                                                                        dw 0001111000000110b ;4
                                                                        dw 0001111000001110b ;5
                                                                        dw 0001110000000010b ;6
                                                                        dw 0001110000000010b ;7
                                                                        dw 0001111000000000b ;8
                                                                        dw 0000111000000000b ;9
                                                                        dw 0000111100000000b ;10
                                                                        dw 0000011100000000b ;11
                                                                        dw 0000011110000000b ;12
                                                                        dw 0000001111000000b ;13
                                                                        dw 0000000111100000b ;14
                                                                        dw 0000000011110000b ;15
                                                                        dw 0000000001111000b ;16
                                                                        dw 0000000000111100b ;17
                                                                        dw 0000000000011110b ;18
                                                                        dw 0001111111101110b ;19
                                                                        dw 0111111111111111b ;20
                                                                        dw 1110000000111111b ;21

                                                                        ;3
                                                                        dw 0000001111110000b ;1
                                                                        dw 0000011100011100b ;2
                                                                        dw 0000011100000100b ;3
                                                                        dw 0000111000000000b ;4
                                                                        dw 0000111000000000b ;5
                                                                        dw 0000011000000000b ;6
                                                                        dw 0000011000000000b ;7
                                                                        dw 0000001100000000b ;8
                                                                        dw 0000000110000000b ;9
                                                                        dw 0001111111110000b ;10
                                                                        dw 0111111000000000b ;11
                                                                        dw 0111100000000000b ;12
                                                                        dw 1111100000000000b ;13
                                                                        dw 1111000000000000b ;14
                                                                        dw 1110000000000000b ;15
                                                                        dw 1110000000000000b ;16
                                                                        dw 1111000000000000b ;17
                                                                        dw 0111100000000001b ;18
                                                                        dw 0111110000000011b ;19
```

```
dw 0011111100000110b ;20
dw 0000111111111100b ;21

;4
dw 0000001100000000b ;1
dw 0000001110000000b ;2
dw 0000001111000000b ;3
dw 0000001101100000b ;4
dw 0000001100110000b ;5
dw 0000001100111000b ;6
dw 0000001100011000b ;7
dw 0000001100001100b ;8
dw 0000001100000110b ;9
dw 1000001100000010b ;10
dw 1000001100000011b ;11
dw 1111111111111111b ;12
dw 1000001100000000b ;13
dw 0000001100000000b ;14
dw 0000001100000000b ;15
dw 0000001100000000b ;16
dw 0000001100000000b ;17
dw 0000001100000000b ;18
dw 0000001100000000b ;19
dw 0000011110000000b ;20
dw 0001111111100000b ;21

;5
dw 0111111111111100b ;1
dw 0111111111111100b ;2
dw 0110000000001100b ;3
dw 0100000000001100b ;4
dw 0000000000001100b ;5
dw 0000000000001100b ;6
dw 0000111100001100b ;7
dw 0001111111101100b ;8
dw 0011111011111100b ;9
dw 0111000000011100b ;10
dw 1110000000000100b ;11
dw 1110000000000000b ;12
dw 1100000000000000b ;13
dw 1100000000000000b ;14
dw 1110000000000000b ;15
dw 1110000000000001b ;16
dw 0110000000000001b ;17
dw 0111000000000010b ;18
dw 0011110000001110b ;19
dw 0001111111111100b ;20
dw 0000011111110000b ;21

;6
dw 0000011110000000b ;1
dw 0001111111100000b ;2
dw 0011000001110000b ;3
dw 0100000000111000b ;4
dw 0100000000011100b ;5
dw 0000000000001110b ;6
dw 0000000000000110b ;7
dw 0000000000000111b ;8
dw 0011111110000111b ;9
dw 0111111111000011b ;10
dw 0111000111100011b ;11
dw 1100000001110011b ;12
dw 1000000000011011b ;13
dw 1000000000011011b ;14
dw 1000000000001111b ;15
dw 1100000000001110b ;16
dw 0100000000000110b ;17
dw 0110000000000100b ;18
dw 0011000000001100b ;19
dw 0001100000011000b ;20
dw 0000111111110000b ;21

;7
dw 1111111111111100b ;1
dw 1111111111111100b ;2
dw 1100000000001110b ;3
dw 1110000000000110b ;4
dw 0111000000000010b ;5
dw 0011100000000001b ;6
dw 0001110000000000b ;7
dw 0000111000000000b ;8
dw 0000011100000000b ;9
dw 0000011110000000b ;10
dw 0000001111000000b ;11
dw 0000000111100000b ;12
dw 0000000011110000b ;13
dw 0000000001110000b ;14
dw 0000000000111000b ;15
dw 0000000000011100b ;16
dw 0000000000001100b ;17
dw 0000000000001110b ;18
dw 0000000000000110b ;19
dw 0000000000000010b ;20
dw 0000000000000010b ;21

;8
dw 0000111111000000b ;1
dw 0011100000110000b ;2
dw 0011000000011000b ;3
dw 0111000000001100b ;4
dw 0111000000001100b ;5
dw 0011000000001100b ;6
dw 0011100000001100b ;7
dw 0000110000111000b ;8
dw 0000001011000000b ;9
dw 0000011111100000b ;10
dw 0000110000111000b ;11
dw 0011000000011100b ;12
dw 0010000000001110b ;13
dw 0110000000000110b ;14
dw 1110000000000111b ;15
dw 1100000000000111b ;16
dw 1110000000000111b ;17
dw 1110000000000111b ;18
dw 0111000000001111b ;19
dw 0011100000011110b ;20
dw 0000111111111000b ;21

;9
dw 0000011111100000b ;1
dw 0001110000111000b ;2
dw 0011100000001110b ;3
dw 0011100000000110b ;4
dw 0111000000000011b ;5
dw 0110100000000011b ;6
dw 0110110000000011b ;7
dw 1110110000000011b ;8
dw 1110011000000011b ;9
dw 1110001100000110b ;10
dw 0110000111111100b ;11
dw 0111000001110000b ;12
dw 0111000000000000b ;13
dw 0011000000000000b ;14
dw 0011100000000000b ;15
dw 0001110000000000b ;16
dw 0000111000000000b ;17
dw 0000011100000000b ;18
dw 0000001110000000b ;19
dw 0000000111100000b ;20
dw 0000000000111111b ;21

QUIT equ 10
dw 0000001111000000b ;1
dw 0000111001110000b ;2
dw 0001100000011000b ;3
dw 0011000000001100b ;4
dw 0111000000000100b ;5
dw 0110000000000110b ;6
dw 0110000000000110b ;7
dw 1110000000000111b ;8
dw 1110000000000111b ;9
dw 1110000010000111b ;10
dw 1110000111000111b ;11
dw 1100001110000111b ;12
dw 1100001100000111b ;13
dw 1110011000000110b ;14
dw 0110110000000110b ;15
dw 0111110000001110b ;16
dw 0011110000001100b ;17
dw 0011100000011100b ;18
dw 0111110000111000b ;19
dw 1111111111110000b ;20
dw 0110001111000000b ;21

PLAY equ 11
dw 0001111000111100b ;1
dw 0001111000111100b ;2
dw 0001111000111100b ;3
dw 0001111000111100b ;4
dw 0001111000111100b ;5
dw 0001111000111100b ;6
dw 0001111000111100b ;7
dw 0001111000111100b ;8
dw 0001111000111100b ;9
dw 0001111000111100b ;10
dw 0001111000111100b ;11
dw 0001111000111100b ;12
dw 0001111000111100b ;13
```

```
dw 0001111000111100b ;14
dw 0001111000111100b ;15
dw 0001111000111100b ;16
dw 0001111000111100b ;17
dw 0001111000111100b ;18
dw 0001111000111100b ;19
dw 0001111000111100b ;20
dw 0001111000111100b ;21

PAUSE equ 12
dw 0000000000000001b ;1
dw 0000000000000111b ;2
dw 0000000000011111b ;3
dw 0000000000111111b ;4
dw 0000000011111111b ;5
dw 0000000111111111b ;6
dw 0000011111111111b ;7
dw 0000111111111111b ;8
dw 0001111111111111b ;9
dw 0011111111111111b ;10
dw 0111111111111111b ;11
dw 0111111111111111b ;12
dw 0011111111111111b ;13
dw 0001111111111111b ;14
dw 0000111111111111b ;15
dw 0000000111111111b ;16
dw 0000000011111111b ;17
dw 0000000000111111b ;18
dw 0000000000011111b ;19
dw 0000000000000111b ;20
dw 0000000000000001b ;21

SANDPILE equ 13
dw 1111111111111111b ;1
dw 1111111111111111b ;2
dw 1000000000000001b ;3
dw 1000000000000001b ;4
dw 0100000000000010b ;5
dw 0010000000000100b ;6
dw 0001110000111000b ;7
dw 0000111111110000b ;8
dw 0000011111100000b ;9
dw 0000001111000000b ;10
dw 0000000110000000b ;11
dw 0000001001000000b ;12
dw 0000010000100000b ;13
dw 0000100010010000b ;14
dw 0001000000001000b ;15
dw 0010000010000100b ;16
dw 0100000110000010b ;17
dw 1000001111000001b ;18
dw 1000011111100001b ;19
dw 1111111111111111b ;20
dw 1111111111111111b ;21

SPACE equ 14
dw 0,0,0,0,0,0,0,0,0,0,0
dw 0,0,0,0,0,0,0,0,0,0

CALIBRATE equ 15
dw 0000000011100000b ;1
dw 0011111111100000b ;2
dw 1111111111100000b ;3
dw 0000000011111111b ;4
dw 0000000011111100b ;5
dw 0000000011100000b ;6
dw 0000000011100000b ;7
dw 0000000011111100b ;8
dw 0000000011111111b ;9
dw 1111111111100000b ;10
dw 0011111111100000b ;11
dw 0000000011100000b ;12
dw 0000000011100000b ;13
dw 0000000011100000b ;14
dw 0000000011100000b ;15
dw 0000000011100000b ;16
dw 0000000011100000b ;17
dw 0000000011100000b ;18
dw 0000000011100000b ;19
dw 0000000011100000b ;20
dw 0000000011100000b ;21

MINUS equ 16
dw 0000000000000000b ;1
dw 0000000000000000b ;2
dw 0000000000000000b ;3
dw 0000000000000000b ;4
dw 0000000000000000b ;5
dw 0000000000000000b ;6
dw 0000000000000000b ;7
dw 0000000000000000b ;8
dw 0000000000000000b ;9
dw 0000000000000000b ;10
dw 0011111111111110b ;11
dw 0011111111111110b ;12
dw 0011111111111110b ;13
dw 0000000000000000b ;14
dw 0000000000000000b ;15
dw 0000000000000000b ;16
dw 0000000000000000b ;17
dw 0000000000000000b ;18
dw 0000000000000000b ;19
dw 0000000000000000b ;20
dw 0000000000000000b ;21

PLUS equ 17
dw 0000000000000000b ;1
dw 0000000000000000b ;2
dw 0000000000000000b ;3
dw 0000000000000000b ;4
dw 0000000111000000b ;5
dw 0000000111000000b ;6
dw 0000000111000000b ;7
dw 0000000111000000b ;8
dw 0000000111000000b ;9
dw 0000000111000000b ;10
dw 0011111111111110b ;11
dw 0011111111111110b ;12
dw 0011111111111110b ;13
dw 0000000111000000b ;14
dw 0000000111000000b ;15
dw 0000000111000000b ;16
dw 0000000111000000b ;17
dw 0000000111000000b ;18
dw 0000000111000000b ;19
dw 0000000000000000b ;20
dw 0000000000000000b ;21

RAZ equ 18
dw 0000000000000000b ;1
dw 0000000000000000b ;2
dw 0000000000000000b ;3
dw 0000000000000000b ;4
dw 1000000100000011b ;5
dw 1100000110000011b ;6
dw 1110000111000011b ;7
dw 1111000111100011b ;8
dw 1111110111110011b ;9
dw 1111111011111011b ;10
dw 1111111111111111b ;11
dw 1111111011111011b ;12
dw 1111110011110011b ;13
dw 1111000111100011b ;14
dw 1110000111000011b ;15
dw 1100000110000011b ;16
dw 1000000100000011b ;17
dw 0000000000000000b ;18
dw 0000000000000000b ;19
dw 0000000000000000b ;20
dw 0000000000000000b ;21

STEP equ 19
dw 0000000000000000b ;1
dw 0000000000000000b ;2
dw 0000000000000000b ;3
dw 1110011100000001b ;4
dw 1110011100000011b ;5
dw 1110011100000111b ;6
dw 1110011100001111b ;7
dw 1110011100011111b ;8
dw 1110011100111111b ;9
dw 1110011101111111b ;10
dw 1110011111111111b ;11
dw 1110011101111111b ;12
dw 1110011100111111b ;13
dw 1110011100011111b ;14
dw 1110011100001111b ;15
dw 1110011100000111b ;16
dw 1110011100000011b ;17
dw 1110011100000001b ;18
dw 0000000000000000b ;19
dw 0000000000000000b ;20
dw 0000000000000000b ;21

ZOOM_IN equ 20
dw 0000111100000000b ;1
dw 0011100111000000b ;2
dw 0110011001100000b ;3
```

```
dw 0100011000100000b ;4
dw 1100011000110000b ;5
dw 1011111111010000b ;6
dw 1011111111010000b ;7
dw 1100011000110000b ;8
dw 0100011000100000b ;9
dw 0110011001100000b ;10
dw 0011100111000000b ;11
dw 0000111111000000b ;12
dw 0000000011100000b ;13
dw 0000000001110000b ;14
dw 0000000001111000b ;15
dw 0000000000111000b ;16
dw 0000000000011100b ;17
dw 0000000000001110b ;18
dw 0000000000001110b ;19
dw 0000000000000111b ;20
dw 0000000000000010b ;21

ZOOM_OUT equ 21
dw 0000111100000000b ;1
dw 0011100111000000b ;2
dw 0110000001100000b ;3
dw 0100000000100000b ;4
dw 1100000000110000b ;5
dw 1011111111010000b ;6
dw 1011111111010000b ;7
dw 1100000000110000b ;8
dw 0100000000100000b ;9
dw 0110000001100000b ;10
dw 0011100111000000b ;11
dw 0000111111000000b ;12
dw 0000000011100000b ;13
dw 0000000001110000b ;14
dw 0000000001111000b ;15
dw 0000000000111000b ;16
dw 0000000000011100b ;17
dw 0000000000001110b ;18
dw 0000000000001110b ;19
dw 0000000000000111b ;20
dw 0000000000000010b ;21

ARROW_DOWN equ 22
dw 0000000111000000b ;1
dw 0000000111000000b ;2
dw 0000000111000000b ;3
dw 0000000111000000b ;4
dw 0000000111000000b ;5
dw 0000000111000000b ;6
dw 0000000111000000b ;7
dw 0000000111000000b ;8
dw 0000000111000000b ;9
dw 0000000111000000b ;10
dw 0000000111000000b ;11
dw 0000000111000000b ;12
dw 0000000111000000b ;13
dw 0000000111000000b ;14
dw 0011000111000110b ;15
dw 0001100111001100b ;16
dw 0000110111011000b ;17
dw 0000011111110000b ;18
dw 0000001111100000b ;19
dw 0000000111000000b ;20
dw 0000000010000000b ;21

ARROW_UP equ 23
dw 0000000010000000b ;1
dw 0000000111000000b ;2
dw 0000001111100000b ;3
dw 0000011111110000b ;4
dw 0000110111011000b ;5
dw 0001100111001100b ;6
dw 0011000111000110b ;7
dw 0000000111000000b ;8
dw 0000000111000000b ;9
dw 0000000111000000b ;10
dw 0000000111000000b ;11
dw 0000000111000000b ;12
dw 0000000111000000b ;13
dw 0000000111000000b ;14
dw 0000000111000000b ;15
dw 0000000111000000b ;16
dw 0000000111000000b ;17
dw 0000000111000000b ;18
dw 0000000111000000b ;19
dw 0000000111000000b ;20
dw 0000000111000000b ;21

ARROW_LEFT equ 24
dw 0000000000000000b ;1
dw 0000000000000000b ;2
dw 0000000000000000b ;3
dw 0000000000000000b ;4
dw 0000000001000000b ;5
dw 0000000001100000b ;6
dw 0000000000110000b ;7
dw 0000000000011000b ;8
dw 0000000000001100b ;9
dw 1111111111111110b ;10
dw 1111111111111111b ;11
dw 1111111111111110b ;12
dw 0000000000001100b ;13
dw 0000000000011000b ;14
dw 0000000000110000b ;15
dw 0000000001100000b ;16
dw 0000000001000000b ;17
dw 0000000000000000b ;18
dw 0000000000000000b ;19
dw 0000000000000000b ;20
dw 0000000000000000b ;21

ARROW_RIGHT equ 25
dw 0000000000000000b ;1
dw 0000000000000000b ;2
dw 0000000000000000b ;3
dw 0000000000000000b ;4
dw 0000001000000000b ;5
dw 0000011000000000b ;6
dw 0000110000000000b ;7
dw 0001100000000000b ;8
dw 0011000000000000b ;9
dw 0111111111111111b ;10
dw 1111111111111111b ;11
dw 0111111111111111b ;12
dw 0011000000000000b ;13
dw 0001100000000000b ;14
dw 0000110000000000b ;15
dw 0000011000000000b ;16
dw 0000001000000000b ;17
dw 0000000000000000b ;18
dw 0000000000000000b ;19
dw 0000000000000000b ;20
dw 0000000000000000b ;21

SCREENSHOT equ 26
dw 1000001000100000b ;1
dw 0000000000000010b ;2
dw 0010000101000000b ;3
dw 0000100000001000b ;4
dw 0000000000000000b ;5
dw 0000001111000000b ;6
dw 0011101001000000b ;7
dw 0011101001001100b ;8
dw 0111111111111110b ;9
dw 1000001111000001b ;10
dw 1000010000100001b ;11
dw 1000110000110001b ;12
dw 1000110000110001b ;13
dw 1000010000100001b ;14
dw 1000001111000001b ;15
dw 0111111111111110b ;16
dw 0000000000000000b ;17
dw 0000000000000000b ;18
dw 0000000000000000b ;19
dw 0000000000000000b ;20
dw 0000000000000000b ;21



;*****************************************************************
;*                          TUNNEL GEOMETRY                      *
;*****************************************************************

bits 32

create_modify_lists:
 pushad

; 1) flush the buffer:
 pxor mm0,mm0
 mov edi,modifier_buffer
$base2__ equ $-4
loop_flush_modify:
 movq [edi],mm0
 add edi,byte 8
 cmp edi,dword modifier_buffer+MBsize
$base2__ equ $-4
 jnz loop_flush_modify
```

```
; 1') reset the variables: (clears the chained list)
 mov edi,modifier_buffer+15
$base2__ equ $-4
 and edi,-16 ; round up
 mov [item_base],edi
$base2__ equ $-4
 add edi,MBsize-16
 mov [first_item],edi
$base2__ equ $-4
 mov [max_item],edi
$base2__ equ $-4

;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

; 2) scan the list:
 mov esi,segments
$base2__ equ $-4
loop_scan_segments:
  push esi

  mov ecx,[XSIZESMC]
$base2__ equ $-4

  movzx eax,word [esi]
  mul ecx
  shr eax,16
  mov [x1],eax
$base2__ equ $-4

  movzx eax,word [esi+4]
  mul ecx
  shr eax,16
  mov [x2],eax
$base2__ equ $-4


  mov ecx,[YSIZESMC]
$base2__ equ $-4
  dec ecx

  movzx eax,word [esi+2]
  mul ecx
  shr eax,16
  inc eax
  mov [y1],eax
$base2__ equ $-4

  movzx eax,word [esi+6]
  mul ecx
  shr eax,16
  inc eax
  mov [y2],eax
$base2__ equ $-4


  mov eax,[x1]
$base2__ equ $-4
  cmp eax,[x2]
$base2__ equ $-4
  je vertical

  mov eax,[y1]
$base2__ equ $-4
  cmp eax,[y2]
$base2__ equ $-4
  je near horizontal

retour:
  pop esi
  add esi,8
  cmp esi,end_segments
$base2__ equ $-4 ; bug... forgotten !
 jb near loop_scan_segments
 popad
 ret


;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

vertical:
;     if y1>y2 then
;       j:=y1;
;       y1:=y2;
;       y2:=j;
 mov eax,[y1]
$base2__ equ $-4
 mov ecx,[y2]
$base2__ equ $-4
 cmp eax,ecx
 jbe no_exchange1
 xchg eax,ecx
no_exchange1:    ; eax=y1, ecx=y2


;     mask:=1 shl (x1 and 15);
;     x1:=(x1 and $FFF0) shr 1;
 mov edx,1
 mov ebx,[x1]
$base2__ equ $-4
 movd mm0,edx
 mov edx,ebx
 and ebx,63
 and edx,-64 ; x1=edx
 movd mm1,ebx
 psllq mm0,mm1 ; mask=mm0


;terminator 1:
; dir1:=A
; dir2:=D
; make_cell;
; dir1:=B
; dir2:=E
; make_cell;
; dir1:=C
; dir2:=F
; make_cell;
 mov ebp,A + (D<<16)
 call make_cell
 mov ebp,B + (E<<16)
 call make_cell
 mov ebp,C + (F<<16)
 call make_cell

; inc(y1);
; while y1<y2 do
;  begin
;  dir1:=2; {B,C}
;  dir2:=3;
;  make_cell;
;  dir1:=1; {A,D}
;  dir2:=4;
;  make_cell;
;  dir1:=5; {E,F}
;  dir2:=6;
;  make_cell;
;  inc(y1);
; end ;
 jmp short middle_of_the_loop1 ; "while" structure
while_loop1:
  mov ebp,B + (C<<16)
  call make_cell
  mov ebp,A + (D<<16)
  call make_cell
  mov ebp,E + (F<<16)
  call make_cell
middle_of_the_loop1:
  inc eax
  cmp eax,ecx
  jb while_loop1

;{terminator:}
; if y1=y2 then
;  dir1:=1; {A,D}
;  dir2:=4;
;  make_cell;
;  dir1:=2; {B,E}
;  dir2:=5;
;  make_cell;
;  dir1:=3; {C,F}
;  dir2:=6;
;  make_cell;
 cmp eax,ecx
 jne near retour
 mov ebp,A + (D<<16)
 call make_cell
 mov ebp,B + (E<<16)
 call make_cell
 mov ebp,C + (F<<16)
 call make_cell

 jmp retour

;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

horizontal:

; if x1>x2 then
```

```
;   j:=x1;
;   x1:=x2;
;   x2:=j;

 mov ebx,[x1]
$base2__ equ $-4
 mov ecx,[x2]
$base2__ equ $-4
 cmp ebx,ecx
 jbe no_exchange2
 xchg ebx,ecx
no_exchange2:    ; eax=y1, ecx=x2, ebx=x1


;{terminator:}
; j:=x1; {push}
; mask:=1 shl (x1 and 15);
; x1:=(x1 and $FFF0) shr 1;

 mov edx,1
 mov eax,[y1]
$base2__ equ $-4
 movd mm0,edx
push ebx
 mov edx,ebx
 and ebx,63
 and edx,-64 ; x1=edx
 movd mm1,ebx
 psllq mm0,mm1 ; mask=mm0


;terminator 1:
; dir1:=A
; dir2:=D
; make_cell;
; dir1:=B
; dir2:=E
; make_cell;
; dir1:=C
; dir2:=F
; make_cell;
 mov ebp,A + (D<<16)
 call make_cell
 mov ebp,B + (E<<16)
 call make_cell
 mov ebp,C + (F<<16)
 call make_cell

; x1:=j+1; {pop}
; dec (x2);
 pop ebx
 dec ecx
 inc ebx


; while x1<x2 do
;  m1:=x1 and $000F;
;  x1:=x1 and $FFF0;
;  m2:=x2-x1;
;  if m2>15
;   then m2:=15;
;  mask:=65535 shr (15-m2);
;  mask2:=(65535 shl m1);
;  mask:=mask2 and mask;
;
;  dir1:=2; {B,F}
;  dir2:=6;
;  make_cell;
;  dir1:=3; {C,E}
;  dir2:=5;
;  make_cell;
;
;  inc(x1,16);

 jmp short middle_of_the_loop2 ; "while" structure
while_loop2:
  mov edx,ebx
  and ebx,-64
  and edx,63  ; m1=edx
  mov ebp,ecx
  sub ebp,ebx ; m2=ebp
  cmp ebp,63
  jbe no_saturation
   mov ebp,63
no_saturation:
  neg ebp
  add ebp,63
  pcmpeqb mm0,mm0
  movd mm1,ebp
  psrlq mm0,mm1
  pcmpeqb mm2,mm2
  movd mm1,edx
  psllq mm2,mm1
  pand mm0,mm2
  mov edx,ebx

  mov ebp,B + (F<<16)
  call make_cell
  mov ebp,C + (E<<16)
  call make_cell

  add ebx,byte 64
middle_of_the_loop2:
  cmp ebx,ecx
  jb while_loop2


; inc(x2);
; mask:=1 shl (x2 and 15);
; x1:=(x2 and $FFF0) shr 1;
 pcmpeqb mm0,mm0
 lea ebx,[ecx+1]
 mov edx,ebx
 psrlq mm0,63
 and ebx,byte 63
 and edx,-64
 movd mm1,ebx
 psllq mm0,mm1


; dir1:=1; {A,D}
; dir2:=4;
; make_cell;
; dir1:=2; {B,E}
; dir2:=5;
; make_cell;
; dir1:=3; {C,F}
; dir2:=6;
; make_cell;
 mov ebp,A + (D<<16)
 call make_cell
 mov ebp,B + (E<<16)
 call make_cell
 mov ebp,C + (F<<16)
 call make_cell

 jmp retour


;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
;-*-*-*-*-*-*-*-*MAJOR HEADACHE HERE*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

make_cell:
; calling convention:
; edx=X
; eax=Y
; ebp=directions
; mm0=mask

  pushad
  mov edi,eax
  imul edi,[XSIZESMC] ; eax=y1*XSIZE
$base2__ equ $-4
  mov byte [depassement],0
$base2__ equ $-5
  lea edi,[dword edx+edi+WALL]
$mem__ equ $-4

; ebp=direction pointers
; edi=address
; esi=k
; ebx=l
; mm0=mask

; "kill" some particles:
 pcmpeqb mm3,mm3
 pxor mm3,mm0
 movq mm1,[edi+A-WALL]
 movq mm2,[edi+B-WALL]
 pand mm1,mm3
 pand mm2,mm3
 movq [edi+A-WALL],mm1
 movq [edi+B-WALL],mm2
 movq mm1,[edi+C-WALL]
 movq mm2,[edi+D-WALL]
 pand mm1,mm3
 pand mm2,mm3
```

```
 movq [edi+C-WALL],mm1
 movq [edi+D-WALL],mm2
 movq mm1,[edi+E-WALL]
 movq mm2,[edi+F-WALL]
 pand mm1,mm3
 pand mm2,mm3
 movq [edi+E-WALL],mm1
 movq [edi+F-WALL],mm2

%ifndef debugging
  movq mm4,[edi+G-WALL]
  pand mm4,mm3
  movq [edi+G-WALL],mm4
%endif


; if the cell has already a list, test it
 mov esi,[edi]
 test esi,esi
 jnz near add_element

; create a new list entry:

;l=max_item
 mov ebx,[max_item]
$base2__ equ $-4
; is it the very first cell of the buffer ?
 cmp ebx,[first_item]
$base2__ equ $-4
 je B_loop0 ; then inaugurate it !


;*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
scan_list:
 movq mm1,mm0
 movd ecx,mm0 ; ecx=mask[1/2]
 psrlq mm1,32
 movd edx,mm1 ; edx=mask[2/2]

; k is not needed anymore (since =0) -> esi=bswap(mask)
 mov esi,ebp
 ror esi,16 ; swap the "pointers"

B_loop4:
; if c[l].a=l+1
 lea eax,[ebx+16]
 cmp eax,[ebx]
 jne suite_loop4

; if mask=c[l].d
 cmp ecx,[ebx+8]
 jne suite_loop4
 cmp edx,[ebx+12]
 jne suite_loop4

; if same directions
 cmp ebp,[ebx+4]
 je same_direction
 cmp esi,[ebx+4]
 jne suite_loop4

; 'link' the list to the tunnel:
same_direction:
 mov [edi],ebx
 popad
 ret

suite_loop4:
 sub ebx, byte 16
 cmp ebx,[first_item]
$base2__ equ $-4 ; forgotten the '$' !!!!! :-(
 jne B_loop4
; if the cell is not found then create a new one:

;*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
; create the first element of all the buffer:
B_loop0:
 lea eax,[ebx+16]
 mov [ebx],eax
 mov [edi],ebx
 mov [ebx+4],ebp
 movq [ebx+8],mm0

; naughty cut
 mov eax,[first_item]
$base2__ equ $-4
 mov dx,MSG_CELLS
 sub eax,byte 16
 cmp eax,[item_base]
$base2__ equ $-4
 jb near exit_with_message
 mov [first_item],eax
$base2__ equ $-4

 popad
 ret


;*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
add_element:
;k=esi
;l=ebx
;@=edi
;ebp=dirs 1/2
;eax
;ecx=dirs 2/2
;edx
;mm2=mask2


; m=c[k].a
; m1=m-1
; l=m1
 mov ebx,[esi]
 mov ecx,ebp ; dir2
 sub ebx,byte 16 ; !!! instead of dec ebx !
 mov [m1],ebx
$base2__ equ $-4
 ror ecx,16  ; dir2/2


B_loop1:  ; search loop: find a similar element in the current sublist
; if ((c[l].b=dir1) and (c[l].c=dir2))
;  or((c[l].b=dir2) and (c[l].c=dir1))
 cmp ebp,[ebx+4]
 je B_found1
 cmp ecx,[ebx+4]
 jne near B_not_found1


B_found1:

; mask2:=c[l].d or mask;
; if mask2<>c[l].d then
;  {complex search}
; else {same mask}
;  if l<k then {don't change the pointer if the list has more cells than expected}
;   im[y1,x1]:=l {link}

; mask2:=c[l].d or mask;
 movq mm2,[ebx+8]
 por mm2,mm0

; if mask2<>c[l].d
 movd eax,mm2
 movq mm1,mm2
 cmp eax,[ebx+8]
 jne modify_then_search
 psrlq mm1,32
 movd eax,mm1
 cmp eax,[ebx+12]
 jne modify_then_search

;               here, it's the same mask (or at least a subset)
 cmp ebx,esi
 jae dont_link
 mov [edi],ebx
dont_link:
 popad
 ret

;*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
;*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
;*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/

; if the element was not found, then copy our sublist at the end
; and modify the mask on the fly

modify_then_search:                 ; fresh wet paint !

; n:=m-1; {end of our original list}
; o:=first_item+1;
; esi=beginning of our sublist
 mov ebp,ebx
 mov ebx,[first_item]
$base2__ equ $-4
 lea ecx,[ebx+16]  ; o = first_item = end of the sublist
 mov edx,[m1]  ; m-1 (end of our original sublist)
```

```
$base2__ equ $-4

; repeat
;  c[first_item].a:=o;
;  c[first_item].b:=c[n].b;
;  c[first_item].c:=c[n].c;
;  if n=l
;   then c[first_item].d:=mask2;
;   else c[first_item].d:=c[n].d;
loop_repeat_copy_list2:
 mov eax,[edx+4]
 movq mm1,[edx+8]
 mov [ebx],ecx
 mov [ebx+4],eax
 cmp edx,ebp
 jne no_change_mm1
  movq mm1,mm2
no_change_mm1:
 movq [ebx+8],mm1

;  dec(n);
;  dec_first_item;
; until (n<k);
 sub ebx,byte 16
 sub edx,byte 16
 cmp ebx,[item_base]
$base2__ equ $-4
 jbe too_many_cells
 cmp edx,esi
 jae loop_repeat_copy_list2

 add ebx, byte 16

 jmp compress_list ; ensures the list is unique !


;*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/

too_many_cells:
 mov dx,MSG_CELLS
 jmp SEL_CODE_16:Back_2_16b

;*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/

B_not_found1:
; check for a mask collision !
; if (c[l].d mask) != 0
 movq mm5,[ebx+8] ;c[l].d
 pand mm5,mm0 ;mask
 movq mm1,mm5
 movq mm2,mm1
 psrlq mm2,32
 por mm1,mm2
 movd eax,mm1
 test eax,eax ; !=0
 jz near no_mask_collision

; AND ((dir1=c[l].b)or(dir1=c[l].c)or(dir2=c[l].b)or(dir2=c[l].c))
 cmp bp,[ebx+4]
 je mask_collision
 cmp cx,[ebx+4]
 je mask_collision
 cmp bp,[ebx+6]
 je mask_collision
 cmp cx,[ebx+6]
 jne near no_mask_collision

mask_collision:

; if depassement!=0 then goto B_loop8
 test byte [depassement],-1
$base2__ equ $-5
 jnz near copy_the_list


;*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/

; else copy the list to the end, and modify elements on the fly:
; create "solid dots" at the crossing of lines.

; n:=m-1; {end of our original list}
; o:=first_item+1;
 mov edx,esi ; limit=k ; bug ????
 mov esi,[m1]  ; m-1
$base2__ equ $-4
 mov ebx,[first_item]
$base2__ equ $-4
 lea ecx,[ebx+16]  ; o
 mov byte [depassement],0
$base2__ equ $-5

; amorce:
 mov eax,ebp
 movq mm1,mm0
 add esi,byte 16 ;      il y a un bug ici ???? (off-by-one)
 jmp short middle_of_the_repeat

; repeat
;  c[first_item].a:=o;
;  c[first_item].b:=c[n].b;
;  c[first_item].c:=c[n].c;
;  c[first_item].d:=c[n].d;
loop_repeat_copy_list:
 mov eax,[esi+4]
 movq mm1,[esi+8]

middle_of_the_repeat:
 mov [ebx],ecx
 mov [ebx+4],eax

 cmp eax,A + (D<<16)
 je modify_A
 cmp eax,D + (A<<16)
 jne modify_mask_B
modify_A:
 or byte [depassement],1
$base2__ equ $-5

 por mm1,mm5
 jmp short end_modify_mask

modify_mask_B:
 cmp eax,B + (E<<16)
 je modify_B
 cmp eax,E + (B<<16)
 jne modify_mask_C
modify_B:
 or byte [depassement],2
$base2__ equ $-5
 por mm1,mm5
 jmp short end_modify_mask

modify_mask_C:
 cmp eax,C + (F<<16)
 je modify_C
 cmp eax,F + (C<<16)
 jne mask_out
modify_C:
 or byte [depassement],4
$base2__ equ $-5
 por mm1,mm5
 jmp short end_modify_mask

mask_out:
 movq mm2,mm5
 pandn mm2,mm1
 movq mm1,mm2

end_modify_mask:
 movq [ebx+8],mm1

;  dec(n);
;  dec_first_item;
; until (n<k);
 sub ebx,byte 16
 sub esi,byte 16
 cmp ebx,[item_base]
$base2__ equ $-4
 jb near too_many_cells
 cmp esi,edx
 jae near loop_repeat_copy_list

; add elements to the list if they were not already present

 test byte [depassement],1
$base2__ equ $-5
 jnz no_A
 mov [ebx],ecx
 mov dword [ebx+4],A + (D << 16)
 movq [ebx+8],mm5
 sub ebx,byte 16
 cmp ebx,[item_base]
$base2__ equ $-4
 jb near too_many_cells

no_A:
 test byte [depassement],2
$base2__ equ $-5
 jnz no_B
```

```
 mov [ebx],ecx
 mov dword [ebx+4],B + (E << 16)
 movq [ebx+8],mm5
 sub ebx,byte 16
 cmp ebx,[item_base]
$base2__ equ $-4
 jb near too_many_cells

no_B:
 test byte [depassement],4
$base2__ equ $-5
 jnz no_C
 mov [ebx],ecx
 mov dword [ebx+4],C + (F << 16)
 movq [ebx+8],mm5
 sub ebx,byte 16
 cmp ebx,[item_base]
$base2__ equ $-4
 jb near too_many_cells

no_C:
 add ebx,byte 16

 jmp compress_list


;*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
;*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/


no_mask_collision:
; dec(l);
; if l>=k then goto loop1;
 sub ebx,byte 16
 cmp ebx,esi
 jae near B_loop1


; it is the end of the list. is it the end of the block ?
; if l=first_item then
;{end of the list reached: we can add an element}
;   c[l].a:=m;
;   goto loop2
 mov eax,[esi]
 cmp ebx,[first_item]
$base2__ equ $-4
 jne not_the_end1
 mov [ebx],eax
 mov [ebx+4],ebp
 movq [ebx+8],mm0
 jmp compress_list


not_the_end1:
; else
;   if c[l].a=m then
;    depassement:=1;
;    goto loop1; {continue as long as we are on the same list}
;  else
; copy the list

 cmp eax,[ebx]
 jne copy_the_list
 mov byte [depassement],-1
$base2__ equ $-5
 jmp B_loop1

copy_the_list:
;loop8:
; n:=m-1; {end of our original list}
; o:=first_item+1;
 mov edx,esi ; limit=k
 mov esi,[m1]  ; m-1
$base2__ equ $-4
 mov ebx,[first_item]
$base2__ equ $-4
 lea ecx,[ebx+16]  ; o

; repeat
;  c[first_item].a:=o;
;  c[first_item].b:=c[n].b;
;  c[first_item].c:=c[n].c;
;  c[first_item].d:=c[n].d;
loop_repeat_copy_list0:
 mov eax,[esi+4]
 movq mm1,[esi+8]
 mov [ebx],ecx
 mov [ebx+4],eax
 movq [ebx+8],mm1
;  dec(n);
;  dec_first_item;
; until (n<k);
 sub ebx,byte 16
 sub esi,byte 16
 cmp ebx,[item_base]
$base2__ equ $-4
 jb near too_many_cells
 cmp esi,edx
 jae loop_repeat_copy_list0

; c[first_item].a:=o;
; c[first_item].b:=dir1;
; c[first_item].c:=dir2;
; c[first_item].d:=mask;
; im[y1,x1]:=first_item;
; dec_first_item;

 mov [ebx],ecx
 mov [ebx+4],ebp
 movq [ebx+8],mm0
; jmp compress_list

;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

compress_list:
; searches the current sublist with everything in the
; whole cell pool. if there is the same sublist somewhere else,
; then link it, else validate the new sublist.

; 1) search a similar subelement
; p1="head" element of our sublist
; p2=maxitem
;l0:
; if *p1<>*p2 then
;l1:
;  dec p2
;  if p2<first_item then
;    first_item==p1, link [edi]=p1, popad ret.
;  else goto l0;
; else
;   (compare the sublist with the list that we found:)
;  if sublist lengths are different then goto l1
;  p3=p2
;  p4=p1
;l2:
;  if *p4<>*p3 then goto l1
;  inc p3, inc p4
;  if p4<c[l].d then goto l2
; (the lists are the same, so we link it)
;  [edi]=p2

; edi=current address in the tunnel
; ebx=pointer to the beginning of our sublist
; other registers: not trustable.

;search a similar subelement:

; p1=ebx (end of the sublist)
; p2=ecx (end of the whole list)
 mov ecx,[max_item]
$base2__ equ $-4

; WARNING: slow and naughty code follows. don't let children watch.

B_search1:
 mov eax,[ebx+4] ; compare the directions
 cmp eax,[ecx+4]
 je B_found2
 ror eax,16
 cmp eax,[ecx+4]
 je B_found2

B_not_found2:
 sub ecx,byte 16
 cmp ecx,[first_item]
$base2__ equ $-4
 ja B_search1

; no element matched the sublist, we "register" it
 mov [edi],ebx
 sub ebx, byte 16
 cmp ebx,[item_base]
$base2__ equ $-4
 jb near too_many_cells
 mov [first_item],ebx
$base2__ equ $-4
```

```
 popad
 ret

B_found2:    ; compare the masks
 mov eax,[ebx+8]
 cmp eax,[ecx+8]
 jne B_not_found2
 mov eax,[ebx+12]
 cmp eax,[ecx+12]
 jne B_not_found2

 mov eax,[ebx]
 sub eax,ebx
 add eax,ecx
 cmp eax,[ecx]
 jne B_not_found2

; scan the sublist:
;   (compare the sublist with the list that we found:)
;  if sublist lengths are different then goto l1
;  p3=c[l].a
;  p4=c[p1].a
;l2:
;  if *p4<>*p3 then goto l1
;  dec p3, dec p4
;  if p4>p1 then goto l2 (don't scan the same elements twice :-D)
; (the lists are the same, so we link it)
;  [edi]=p2

; p4=esi
; p3=edx
 mov esi,[ecx]
 mov edx,[ebx]
 sub esi,byte 16
 sub edx,byte 16

B_search_sublist:
; compare the directions:
 mov eax,[edx+4]
 cmp eax,[esi+4]
 je B_found3
 ror eax,16
 cmp eax,[esi+4]
 jne B_not_found2

B_found3:    ; compare the masks
 mov eax,[edx+8]
 cmp eax,[esi+8]
 jne B_not_found2
 mov eax,[edx+12]
 cmp eax,[esi+12]
 jne B_not_found2

; no need to compare the list size

 sub edx,byte 16
 sub esi,byte 16
 cmp edx,ebx
 ja B_search_sublist

 mov [edi],ecx
; sub ebx, byte 16
; mov [first_item],ebx ; bug de fatigue: recopie
    ; de mémoire sans se rappeler du contexte
;$base2__ equ $-4

 popad
 ret

;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
;-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*


;************ DATA ************

align 4
depassement: dd 0
first_item: dd 0
max_item: dd 0
item_base: dd 0
m1: dd 0
x1: dd 0
y1: dd 0
x2: dd 0
y2: dd 0

segments:
;  x1,y1,x2,y2
dw  0, 0,-1, 0
dw -1, 0,-1,-1
dw -1,-1, 0,-1
dw  0,-1, 0, 0
;dw 15000,15000,20000,15000
;dw 20000,15000,20000,20000
;dw 20000,20000,15000,20000
;dw 15000,20000,15000,15000
;dw  5000,15000,25000,15000

;dw  2000,56000,45000,56000
;dw  2000,55000,45000,55000
;dw 40000,47000,40000,60000
;dw 45000,50000,45000,64000

;dw 20000,0,20000,32000
;dw 20000,33000,20000,-1

end_segments equ $


;*****************************************************************
;*                          MISC VARIABLES:                      *
;*****************************************************************

align 8

TSC: dd 0
cycle_count : dd 0
limit:    dd -1
XSIZESMC: dd 574;1024 ;960
YSIZESMC: dd 400; 640  ;736
zoom: dd 4 ; between 0 and 6
; normal resolution: 4=1:1
; zoom out: half res: 5=2:1, fourth res: 6=4:1
; zoom in: 3=1:2, 2=1:4, 1=1:8, 0=1:16 (not working now)

strip_count: dd 1
tunnel_x: dd 0
tunnel_y: dd 0
mouse_buttons: db 0 ; gets loaded in CL
mouse_count:  db 0   ; byte counter in CH
mouse_buffer: db 0,0 ; in DX
pause_flag: db 1
escape_count: db 3

color_palette:
; 252: number background
 db 0,0,30 ; blue
; 253: number foreground
 db 0,63,0 ; light green
; 254: the mouse cursor:
 db 63,63,63 ; white
; 255: the background:
 db 0,0,0 ; black
end_palette:
size_palette equ $-color_palette

;*****************************************************************
;*                  DATA that CAN be overwritten:                *
;*****************************************************************

align 8

; some register backup room overwriting the setup time messages
; (they won't be used anymore)
backup_mm0 equ $    ;
backup_mm1 equ $+8  ;
backup_mm2 equ $+16 ;
backup_mm3 equ $+24 ; these ones are used by the mouse handler
backup_mm4 equ $+32 ;
backup_mm5 equ $+40 ;
backup_mm6 equ $+48 ;
backup_mm7 equ $+56 ;

bmm0 equ $+64  ;
bmm1 equ $+72  ;
bmm2 equ $+80  ;
bmm3 equ $+88  ; used by the keyboard
bmm4 equ $+96  ;
bmm5 equ $+104 ;
bmm6 equ $+112 ;
bmm7 equ $+120 ;

;backup_2_0 equ $+64
;backup_2_1 equ $+72; used by the MP

; relocation label table here:
```

```
%include "out.inc"

end_base:

; SETUP time error messages (130 overwritable bytes)
MSG_MMX:              db "No MMX$"
MSG_PROT_MODE:        db "Run without EMM386 or Windows$"
MSG_NO_MOUSE:         db "Serial mouse not found on COM1$"
MSG_NO_XMS:           db "Not enough XMS$"
MSG_WRONG_MOUSE_PORT: db "Can't get the mouse IRQ on COM1$"
MSG_NO_VESA:          db "VESA not found$"

align 8

;
; structure of the block returned by the VESA BIOS
;

VbeInfoBlock:
VbeSignature:      db 'VBE2'      ; VBE Signature, for extended informations
VbeVersion         dw 0200h       ; VBE Version
OemStringPtr       equ VbeInfoBlock+6  ;dd Pointer to OEM String
Capabilities       equ VbeInfoBlock+10 ;dd Capabilities of graphics cont.
VideoModePtr       equ VbeInfoBlock+14 ;dd Pointer to Video Mode List
TotalMemory        equ VbeInfoBlock+18 ;dw Number of 64kb memory blocks
OemSoftwareRev     equ VbeInfoBlock+20 ;dw VBE implementation Software revision
OemVendorNamePtr   equ VbeInfoBlock+22 ;dd Pointer to Vendor Name String
OemProductNamePtr  equ VbeInfoBlock+26 ;dd Pointer to Product Name String
OemProductRevPtr   equ VbeInfoBlock+30 ;dd Pointer to Product Revision String
Reserved           equ VbeInfoBlock+34 ;times 222 db 0 ; Reserved for scratch
OemData            equ VbeInfoBlock+256;times 256 db 0 ; Data Area for OEM Strings

;
; Info block for a specified video mode: (overwrites VbeInfoBlock)
;

ModeInfoBlock       equ VbeInfoBlock
ModeAttributes      equ ModeInfoBlock     ; dw mode attributes
WinAAttributes      equ ModeInfoBlock+2   ; db window A attributes
WinBAttributes      equ ModeInfoBlock+3   ; db window B attributes
WinGranularity      equ ModeInfoBlock+4   ; dw window granularity
WinSize             equ ModeInfoBlock+6   ; dw window size
WinASegment         equ ModeInfoBlock+8   ; dw window A start segment
WinBSegment         equ ModeInfoBlock+10  ; dw window B start segment
WinFuncPtr          equ ModeInfoBlock+12  ; dd pointer to window function
BytesPerScanLine    equ ModeInfoBlock+16  ; dw bytes per scan line
XResolution         equ ModeInfoBlock+18  ; dw horizontal resolution
YResolution         equ ModeInfoBlock+20  ; dw vertical resolution
XCharSize           equ ModeInfoBlock+22  ; db character cell width
YCharSize           equ ModeInfoBlock+23  ; db character cell height
NumberOfPlanes      equ ModeInfoBlock+24  ; db number of memory planes
BitsPerPixel        equ ModeInfoBlock+25  ; db bits per pixel
NumberOfBanks       equ ModeInfoBlock+26  ; db number of banks
MemoryModel         equ ModeInfoBlock+27  ; db memory model type
BankSize            equ ModeInfoBlock+28  ; db bank size in KB
NumberOfImagePages  equ ModeInfoBlock+29  ; db number of images
Reserved_2          equ ModeInfoBlock+30  ; db reserved for page function
RedMaskSize         equ ModeInfoBlock+31  ; db size of direct color red mask
RedFieldPosition    equ ModeInfoBlock+32  ; db bit position of lsb of red mask
GreenMaskSize       equ ModeInfoBlock+33  ; db size of direct color green mask
GreenFieldPosition  equ ModeInfoBlock+34  ; db bit position of lsb of green mask
BlueMaskSize        equ ModeInfoBlock+35  ; db size of direct color blue mask
BlueFieldPosition   equ ModeInfoBlock+36  ; db bit position of lsb of blue mask
RsvdMaskSize        equ ModeInfoBlock+37  ; db size of direct color reserved mask
RsvdFieldPosition   equ ModeInfoBlock+38  ; db bit position of lsb of reserved mask
DirectColorModeInfo equ ModeInfoBlock+39  ; db direct color mode attributes
PhysBasePtr         equ ModeInfoBlock+40  ; dd physical address for flat frame buffer
OffScreenMemOffset  equ ModeInfoBlock+44  ; dd pointer to start of off screen memory
OffScreenMemSize    equ ModeInfoBlock+48  ; dw amount of off screen memory in 1k units
Reserved_3          equ ModeInfoBlock+50  ; db 206 dup (?) remainder

AP_stack equ OemData+256-4
; the AP's stack expands down from here

AP_landing_zone equ AP_stack+4

MP_stack_top equ AP_landing_zone+5116
; the main processor's stack expands down from here

BMP_buffer equ MP_stack_top+4

modifier_buffer equ BMP_buffer+1024

the_end_of_it_all equ modifier_buffer + MBsize


;*****************************************************************
;*                            THE END                            *
;*****************************************************************
```