Many people have posted asking for some fool to give an example of protected mode programming, to allow everyone to laugh at their code.

Introduction

I wrote my first protected mode program many years ago to learn all of the idiosyncracies of the Intel architecture. I still recommend this approach to anyone, because there's nothing like learning from your mistakes! But in the spirit of sharing knowledge, I offer my first program to all for edification and amusement. I am not saying it is good code, nor that it couldn't be done better, but since I was more interested in getting it to work than in doing it the best way, I believe this code is relatively straightforward.

Requirements

I first had to work out what features I wanted to try - and being the fool I was I thought I'd try everything! This included:

- Accessing extended memory
- Multiple tasks
- Local descriptor tables
- Multiple privilege levels
- 16 and 32-bit code
- Revectoring hardware interrupts
- Direct hardware access (A20, PIC, timer, keyboard, screen etc.)
- Paging (although I dropped this for this program)

So this program is designed to start up multiple tasks, that each do the same thing - bounce a ball in its own window. If I could get it fast enough, they would seemingly all be able to bounce together.

High Level Design

To make it easier on myself, I decided to effectively hard code as much as possible, letting the assembler do most of the work. This turned out to be a good idea, since I could have a look at the actual assembled constants and make sure I was telling it what I thought I was! My most invaluable tools at this stage were the '386 Intel manual, and Borland's TD - not to debug, but to have a look to the disassembled code. I even found a bug in Borland's disassembler!

So I wanted the program to do the following steps:

- Set up core system tables in preparation of the switch to protected mode
- Switch to protected mode
- Set up rest of system tables (esp. exception handlers)
- Set up each user task
- Twiddle thumbs while tasks ran, until user pressed <Esc>
- Return to real mode
- Restore system to DOS state

This last turned out to be invaluable, since having to reboot the computer every time to get out of my program wasted a lot of time (30 seconds reboot, 5 seconds running!). As it was, the system rebooted itself more than enough times until I got the exception handlers right!

Lessons I Learned

To help any other novice protected-mode programmers out there, I can recommend the following:

1) Spend the time to set up exception handlers. It is so easy to make a mistake that merely dumping all of the register contents onto the screen (in hex) is invaluable for not only finding which instruction caused the problem, but also in finding out what error the '386 is complaining about. If you don't do this step, the most likely scenario is that any exception will result in another, which will result in a double exception, which will result in another exception, which will shut down the CPU, which will reboot the computer.
2) Examine the listing file the assembler can produce. Make sure the assembler

and you both think you're talking about the same thing! Use a debugger (unfortunately DOS's DEBUG only accepts 8086 instructions) and type in the assembled hex bytes, and see if the debugger thinks it's the same instruction as you've typed. (Even this isn't perfect - the debugger may be assuming you're in 16-bit mode for a 32-bit instruction, producing wierd results! It also may have bugs itself!)

Idiosyncracies I Discovered

1) Protected mode far jumps can't be assembled. Borland's TASM (anyway) can't be convinced to assemble JMP OFFSET ProtMode:OFFSET ProtProc. So I had to dummy these up with DBs and DWs.
2) The '386 does not save a task's LDT into its TSS on a task switch. So if LDT changes, you either have to update the TSS manually, or prevent task switching.
3) The Trace bit in the TSS is not automatically switched off when the task is entered. This means that if tasks are still being switched behind the debugger, the T bit needs to be switched off manually to prevent re-entering the debugger!

Tools I Used

I have assembled this program with three different version of Borland's TASM. I don't own MASM, but I do not believe I have done anything Borland specific. If anyone has any trouble, contact me and I will try to fix it.

The Code

I have had trouble attaching text to News articles so that everyone can decode the result. I also do not have the facilities to put it in an ftp site. If anyone wants to mail me I'll deliver the complete source to them so that they can do it for me. For now I will post the code in long-hand, in five parts (~300 lines per part), and I will thread them off this message so that it doesn't clutter the Newsgroup.

Part 1 - Definitions
Part 2 - Real mode entry point
Part 3 - Protected mode entry point
Part 4 - User code (small)
Part 5 - Interrupt and exception handlers (large)

WARNINGS (One legal, one moral)

As a professional programmer, I identify with my code. This may be old, and not how I do things now, but it is still mine. I don't mind showing this code to others so that they can learn, and you can experiment to your heart's content, but DO NOT USE ANY PART of this code in new software, whether commercial or otherwise.

Finally, this code is the ultimate in unfriendliness! It assumes that the whole machine belongs to it - including ALL RAM. This means that it should be run on a vanilla system - no HIMEM.SYS, no memory managers, and DEFINITELY no disk-cache programs! Your memory WILL be written over!

John Burger

```
;*****************************************************************************
; Copyright (c) John Burger, 1990-1995. All rights reserved.
; This code has been placed in the public domain by John Burger for example
; purposes only - to demonstrate various techniques for programming the
; Intel 80386 et. al. in protected mode. It is available for other people to
; experiment with, NOT to include in other software, whether commercial or
; otherwise.
;*****************************************************************************

; This program switches into protected mode (on a '386+), starts many tasks
; that all do the same thing in different windows, and continues until the
; <Esc> key is pressed and released.
;
; Minimum system: '386 with 2Mb RAM (to test A20 line)
;                 Standard A20 line
; If A20 line non-standard, or <2Mb RAM available, modify the code to use DOS
; RAM rather than extended RAM.
;
; *** WARNING ***
; System should be 'vanilla': No HIMEM.SYS, memory manager or disk cache.
; This program accesses the hardware directly, and will TRASH any memory-
; manager by writing all over it!

;
; Experiments to try:
;  - Various values for WindowWidth and WindowHeight:
;      0,0 maximises number of tasks;
;  - Various values for SecTick to slow down and speed up operation:
;      Below 19 and above 65535 will cause compiler errors.
;      Higher values may cause GP faults!
;  - Insert an INT 3 anywhere in ProtSeg, BallCode or IntCode.
;  - Insert any INT in the above segments, AFTER they've been set up.
;      Look for *** in ProtSeg
;  - Modify code to generate various exceptions:
;      Crash the stack (MOV SP,1; PUSH AX)
;      Access illegal memory (INC BYTE PTR ES:[0FFFFFFFh]
;      Invalid op-codes (LOCK XOR AX,AX)
;      Enable single-step debugging (PUSHF; POP AX; OR AX,100h; PUSH AX; POPF)
;        The single-step debug handler simply waits for the user to press
;        <Enter> or <Space>, then returns. To indicate waiting, it uses a
;        "dingle"<tm> in the top right corner of the screen. Pressing <Enter>
;        stops tracing.
;

          WARN                          ; Listen to every assembler complaint!

          .386P                         ; Going to be doing 386 Prot Mode inst

;
; Stack segment, reserved by DOS but used in protected mode also.
;
StackSeg  SEGMENT   STACK PARA USE32    ; DOS stack
          DB        100 DUP ('STACK ')  ; (Used to ID memory)
StackTop  EQU       $                   ; For 16 bit code
StackSeg  ENDS

StackSize EQU       1024                ; For 32 bit stacks

;
; Multitasking constants:
;   Reduce SecTick for slower machines.
;      Note certain values may cause errors.
;   Change WindowWidth and WindowHeight to change number of windows.
;      Note certain values may cause errors.
;   Change BounceTime to slow down the bouncing.
;   Change Ball to change character that bounces.
;
SecTick   EQU       1000                ; Task switches per second
```

```
WindowWidth  EQU    4                   ; Width of a window
WindowHeight EQU    3                   ; Height of a window
BounceTime EQU      1                   ; Time delay before moving
Ball       EQU      1                   ; Character to bounce

;===========================================================================
;
; System constants. These are defined by the hardware.
;
ClockFreq EQU       1193200             ; Timer 0 frequency

; Constants in Granular byte of descriptor table entry
PageGran  EQU       80h
ByteGran  EQU       00h
More64K   EQU       40h
Less64K   EQU       00h
LargeAddr EQU       40h
SmallAddr EQU       00h
Available EQU       10h
LimitHi   EQU       0Fh

; Constants in Type byte of descriptor table entry
Present     EQU     80h
NotPresent  EQU     00h
DPL         EQU     60h
Privilege0  EQU     00h
Privilege1  EQU     20h
Privilege2  EQU     40h
Privilege3  EQU     60h
Memory      EQU     10h
System      EQU     00h
Execable    EQU     08h
NotExecable EQU     00h
Gate386     EQU     08h
Gate286     EQU     00h
ExpandDown  EQU     04h
ExpandUp    EQU     00h
Conform     EQU     04h
NonConform  EQU     00h
Writable    EQU     02h
NotWritable EQU     00h
Readable    EQU     02h
NotReadable EQU     00h
Accessed    EQU     01h
NotAccessed EQU     00h

; Bit indicating descriptor is in LDT
LocalDT   EQU       04h

; Descriptor types
AvailTSS  EQU       1
LDT       EQU       2
BusyBit   EQU       2
BusyTSS   EQU       3
CallGate  EQU       4
TaskGate  EQU       5
IntGate   EQU       6
TrapGate  EQU       7

Int386    EQU       Present+Gate386+IntGate   ; Disable ints within routine
Trap386   EQU       Present+Gate386+TrapGate  ; Keep ints
Task386   EQU       Present+TaskGate          ; Not Gate386 - not defined!

;===========================================================================
;
; System structures. These are defined by the hardware.
;
```

```
; Descriptor structure
Descriptor STRUC
LimitLo   DW        ?
BaseLo    DW        ?
BaseMid   DB        ?
DescType  DB        ?
DescGran  DB        ?
BaseHi    DB        ?
Descriptor ENDS

; Gate structure
Gate      STRUC
OffsetLo  DW        ?
Selector  DW        ?
Count     DB        ?
GateType  DB        ?
OffsetHi  DW        ?
Gate      ENDS

; Pseudo-descriptor table pointer (GDT/IDT)
DTPtr     STRUC               ; Descriptor Table Pointer
Limit     DW        ?
Base      DD        ?
DTPtr     ENDS

;=============================================================================
; System segments. The structure of these is defined by the hardware.

;
; Interrupt Descriptor Table.
; Intel has reserved the first 32 interrupts, and given the first half of them
; functions. Note that hardware vectors have been moved!
; Since it is such a small program (ha!), the entire table can be hard-coded
; here. Note that the descriptor fields are simply offsets into the GDT.
; Note also that four faults are represented by their own tasks, as
; recommended by Intel - Debug, Double, BadTSS, and BadStack. These all
; guarantee that a valid state will be available to the system to process the
; fault.
;
IDT       SEGMENT   PARA USE16
Divide    Gate      <OFFSET DivideInt,OFFSET IntCode,0,Trap386,0>
;Debug     Gate      <0,OFFSET DebugTSS,0,Task386,0> ; Debug as separate task?
Debug     Gate      <OFFSET DebugInt,OFFSET IntCode,0,Trap386,0>
NMI       Gate      <OFFSET NMIInt,OFFSET IntCode,0,Trap386,0>
Break     Gate      <OFFSET BreakInt,OFFSET IntCode,0,Trap386,0>
Overflow  Gate      <OFFSET OverInt,OFFSET IntCode,0,Trap386,0>
OutBound  Gate      <OFFSET BoundInt,OFFSET IntCode,0,Trap386,0>
InvalidOp Gate      <OFFSET OpInt,OFFSET IntCode,0,Trap386,0>
No387     Gate      <OFFSET No387Int,OFFSET IntCode,0,Trap386,0>
Double    Gate      <0,OFFSET DoubleTSS,0,Task386,0>
Over387   Gate      <OFFSET Over387Int,OFFSET IntCode,0,Trap386,0>
BadTSS    Gate      <0,OFFSET BadTSSTSS,0,Task386,0>
NoSegment Gate      <OFFSET NoSegInt,OFFSET IntCode,0,Trap386,0>
BadStack  Gate      <0,OFFSET StackTSS,0,Task386,0>
General   Gate      <OFFSET GenInt,OFFSET IntCode,0,Trap386,0>
PageFault Gate      <OFFSET PageInt,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int15,OFFSET IntCode,0,Trap386,0>
Bad387    Gate      <OFFSET Bad387Int,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int17,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int18,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int19,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int20,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int21,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int22,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int23,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int24,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int25,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int26,OFFSET IntCode,0,Trap386,0>
```

```
          Gate      <OFFSET Int27,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int28,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int29,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int30,OFFSET IntCode,0,Trap386,0>
          Gate      <OFFSET Int31,OFFSET IntCode,0,Trap386,0>
Timer     Gate      <OFFSET TimerInt,OFFSET IntCode,0,Int386,0>
Keyboard  Gate      <OFFSET KeyInt,OFFSET IntCode,0,Int386,0>
Slave     Gate      <OFFSET SlaveInt,OFFSET IntCode,0,Int386,0>
COM2      Gate      <OFFSET COM2Int,OFFSET IntCode,0,Int386,0>
COM1      Gate      <OFFSET COM1Int,OFFSET IntCode,0,Int386,0>
          Gate      <OFFSET IRQ5,OFFSET IntCode,0,Int386,0>
          Gate      <OFFSET IRQ6,OFFSET IntCode,0,Int386,0>
Printer   Gate      <OFFSET PrintInt,OFFSET IntCode,0,Int386,0>
          Gate      <OFFSET IRQ8,OFFSET IntCode,0,Int386,0>
          Gate      <OFFSET IRQ9,OFFSET IntCode,0,Int386,0>
          Gate      <OFFSET IRQ10,OFFSET IntCode,0,Int386,0>
          Gate      <OFFSET IRQ11,OFFSET IntCode,0,Int386,0>
          Gate      <OFFSET IRQ12,OFFSET IntCode,0,Int386,0>
          Gate      <OFFSET IRQ13,OFFSET IntCode,0,Int386,0>
          Gate      <OFFSET IRQ14,OFFSET IntCode,0,Int386,0>
          Gate      <OFFSET IRQ15,OFFSET IntCode,0,Int386,0>
IDTLimit  EQU       $-1
IDT       ENDS

;
; Global Descriptor Table.
;
; None of the Base fields can be filled in, as it is not know where the code
; will load into memory.
;
; Only the first entry is reserved by Intel. This is the unused descriptor to
; allow the system to detect null pointer references.
; The others are arbitrary.
; The two MSDOS ones are for the return to DOS - to reset the descriptors back
; to valid values.
; GDTData and IDTData are aliases for the GDT and IDT respectively.
; ProtCode is the protected-mode code segment for intialisation and wrapup.
; IntCode is the protected-mode code segment for interrupt code. (32-bit)
; Screen is the CGA/EGA/VGA/SVGA text screen.
; MainTask is the task state segment of the executive.
; MainStack is the level 0 stack for the main task.
; StackTSS is the task state segment for a stack fault.
; TSSStack is the stack for the stack fault task.
; BadTSSTSS is the task state segment for a TSS fault.
; BTStack is the stack for the BadTSS fault task.
; DoubleTSS is the task state segment for a double fault.
; DblStack is the stack for the double fault task.
; DebugTSS is the task state segment for the debug trap.
; DbgStack is the stack for the debug trap task.
; More is the room for all of the TSSs and LDTs of the user tasks.
;
GDT       SEGMENT   PARA USE16
          Descriptor<>
MSDOSCode Descriptor<0FFFFh,?,?,Present+Memory+Execable+Readable,ByteGran,0>
MSDOSData Descriptor<0FFFFh,?,?,Present+Memory+Writable,ByteGran,0>
GDTData   Descriptor<GDTLimit,?,?,Present+Memory+Writable,ByteGran,0>
IDTData   Descriptor<IDTLimit,?,?,Present+Memory+Writable,ByteGran,0>
ProtCode  Descriptor<ProtLimit,?,?,Present+Memory+Execable+Readable,ByteGran,0>
IntCode   Descriptor<IntLimit,?,?,Present+Memory+Execable+Readable,ByteGran+LargeAddr,0>
Screen    Descriptor<3999,8000h,0Bh,Present+Privilege3+Memory+Writable,ByteGran,0>
StackTSS  Descriptor<TSSLimit,?,?,System+Gate386+AvailTSS,ByteGran,0>
TSSStack  Descriptor<StackTop-1,?,?,Present+Memory+Writable,ByteGran+More64K,0>
BadTSSTSS Descriptor<TSSLimit,?,?,System+Gate386+AvailTSS,ByteGran,0>
BTStack   Descriptor<StackTop-1,?,?,Present+Memory+Writable,ByteGran+More64K,0>
DoubleTSS Descriptor<TSSLimit,?,?,System+Gate386+AvailTSS,ByteGran,0>
DblStack  Descriptor<StackTop-1,?,?,Present+Memory+Writable,ByteGran+More64K,0>
DebugTSS  Descriptor<TSSLimit,?,?,System+Gate386+AvailTSS,ByteGran,0>
DbgStack  Descriptor<StackTop-1,?,?,Present+Memory+Writable,ByteGran+More64K,0>
```

```
MainTask  Descriptor<TSSLimit,?,?,Present+System+Gate386+AvailTSS,ByteGran,0>
MainStack Descriptor<StackTop-1,?,?,Present+Memory+Writable,ByteGran+More64K,0>
UserTasks EQU       $
          ORG       0FFFFh
GDTLimit  EQU       $
GDT       ENDS

;
; Task State Segment.
; The structure is defined by Intel. Note multiple TSSs will be created.
;
TSS       SEGMENT   PARA USE16
BackLink  DW        0
          DW        0
ESP0      DD        0
SS0       DW        0
          DW        0
ESP1      DD        0
SS1       DW        0
          DW        0
ESP2      DD        0
SS2       DW        0
          DW        0
PDBR      DD        0                   ; Page Directory Base Register
EIPReg    DD        ?
EFlags    DD        ?
EAXReg    DD        ?
ECXReg    DD        ?
EDXReg    DD        ?
EBXReg    DD        ?
ESPReg    DD        ?
EBPReg    DD        ?
ESIReg    DD        ?
EDIReg    DD        ?
ESSeg     DW        ?
          DW        0
CSSeg     DW        ?
          DW        0
SSSeg     DW        ?
          DW        0
DSSeg     DW        ?
          DW        0
FSSeg     DW        ?
          DW        0
GSSeg     DW        ?
          DW        0
LDTR      DW        0                   ; Local Descriptor Table Register
          DW        0
Tracing   DW        0
IOMapBase DW        TSSSize
TSSSize   EQU       $-BackLink
TSSLimit  EQU       TSSSize-1
TSS       ENDS

; Debugger TSS
TSSDebug  SEGMENT   PARA USE16
          DB        TSSSize DUP (?)
TSSDebug  ENDS

; Double TSS
TSSDouble SEGMENT   PARA USE16
          DB        TSSSize DUP (?)
TSSDouble ENDS

; Bad TSS TSS
TSSTSS    SEGMENT   PARA USE16
          DB        TSSSize DUP (?)
TSSTSS    ENDS
```

```
; Bad Stack TSS
BadStackTSS SEGMENT PARA USE16
          DB        TSSSize DUP (?)
BadStackTSS ENDS

;
; Local Descriptor Table.
; Each user gets their own LDT, with a descriptor for:
; Stack0:    level 0 stack
; Stack3:    level 3 stack (levels 1 and 2 not used)
; TaskCode:  user code segment - all user code is in the same segment
; TaskData:  user data segment - all user data is in a private segment
;
TaskLDT   SEGMENT   PARA USE16
Stack0    Descriptor<>
Stack3    Descriptor<>
TaskCode  Descriptor<>
TaskData  Descriptor<>
LDTLimit  EQU       $-1
TaskLDT   ENDS

; DOS/Protected mode data
Data      SEGMENT   PARA USE16
OldIDT    DTPtr     <>                  ; Old (DOS) IDT value
GDTPtr    DTPtr     <GDTLimit,?>        ; GDT pointer constructed here
IDTPtr    DTPtr     <IDTLimit,?>        ; IDT pointer constructed here
Corner    DW        0101h               ; The starting window corner
TaskJump  LABEL     PWORD               ; This is the task jump address
          DD        0                   ; The offset part is unused
NextTask  DW        0                   ; The next task to run
LastTask  DW        0                   ; The last task in the GDT
KeyPress  DB        0                   ; A one-key buffer!
TaskLock  DB        0                   ; When 1, do NOT task switch!
Data      ENDS
```

Burger / John Adriaan (ISE) <u86...@student.canberra.edu> wrote:
>The Code
>========
>I have had trouble attaching text to News articles so that everyone can decode
>the result. I also do not have the facilities to put it in an ftp site. If
>anyone wants to mail me I'll deliver the complete source to them so that they
>can do it for me. For now I will post the code in long-hand, in five parts
>(~300 lines per part), and I will thread them off this message so that it
>doesn't clutter the Newsgroup.
>
>Part 1 - Definitions
>Part 2 - Real mode entry point
>Part 3 - Protected mode entry point
>Part 4 - User code (small)
>Part 5 - Interrupt and exception handlers (large)

I knew I was forgetting something, but I couldn't remember what it was!
It hit. To assemble the above, get it all into one file in the above order.
Then simply assemble it with

TASM PROTMODE   (or whatever assembler you use - no DEFINEs needed)

then

TLINK /3 PROTMODE   (or whatever. There are 32-bit fixups: Borland uses /3)

The result is a 69K-odd .EXE, 4K of which is the code, 64K is the GDT!

```
;============================================================================
;
; This is the DOS code segment, that starts and stops the program.
;

RealSeg   SEGMENT   PARA USE16

          ASSUME    CS:RealSeg,DS:Nothing,ES:Nothing,FS:Nothing,GS:Nothing

Entry     PROC      FAR
;
; Initialise.
;
          MOV       AH,1                ; Set cursor size
          MOV       CX,2607h            ; to invisible!
          INT       10h                 ; using BIOS

          MOV       AX,SEG Data         ; Point FS to data
          MOV       FS,AX
          ASSUME    FS:Data             ; And tell assembler

          MOV       AX,SEG GDT          ; Point DS to GDT
          MOV       DS,AX
          ASSUME    DS:GDT              ; And tell assembler

          SIDT      FS:[OldIDT]         ; Store old IDT pointer (for recall)

;
; Initialise GDT base entries.
;
          XOR       EAX,EAX                 ; Zero EAX
          MOV       AX,SEG RealSeg          ; Get start segment into AX
          MOV       BX,OFFSET MSDOSCode     ; Point to GDT entry
          CALL      StoreDesc               ; And update GDT

          MOV       AX,SEG Data             ; Get data segment into AX
          MOV       BX,OFFSET MSDOSData     ; Point to GDT entry
          CALL      StoreDesc               ; And update GDT

          MOV       AX,SEG ProtSeg          ; Get Prot code segment into AX
          MOV       BX,OFFSET ProtCode      ; Point to GDT entry
          CALL      StoreDesc               ; And update GDT

          MOV       AX,SEG IntSeg           ; Get Int segment into AX
          MOV       BX,OFFSET IntCode       ; Point to GDT entry
          CALL      StoreDesc               ; And update GDT

          MOV       AX,SEG StackSeg         ; Get Stack segment into AX
          MOV       BX,OFFSET MainStack     ; Point to GDT entry
          CALL      StoreDesc               ; And update GDT

          MOV       AX,SEG GDT              ; Get GDT segment into AX
          SHL       EAX,4                   ; Turn into linear address
          MOV       FS:[GDTPtr.Base],EAX    ; Store into GDT pointer
          MOV       BX,OFFSET GDTData       ; Point to GDT entry
          CALL      StoreLin                ; And update GDT

          MOV       AX,SEG IDT              ; Get IDT segment into AX
          SHL       EAX,4                   ; turn into linear address
          MOV       FS:[IDTPtr.Base],EAX    ; Store into IDT pointer
          MOV       BX,OFFSET IDTData       ; Point to GDT entry
          CALL      StoreLin                ; And update GDT

          MOV       AH,0DFh                 ; Enable A20 line
          CALL      GateA20                 ; Call Gate routine
          JNZ       FailA20                 ; Didn't work! Bomb!
          LGDT      FS:[GDTPtr]             ; Point to GDT
```

```
          LIDT      FS:[IDTPtr]             ; Point to IDT

          MOV       EAX,CR0                 ; Get current CR0
          OR        EAX,1                   ; Set Protected Mode bit
          MOV       CR0,EAX                 ; NOW IN PROTECTED MODE!

; Getting the assembler to assemble a protected mode jump with the correct
; segment and offset is next to impossible. So just hard-code it!

          DB        0EAh                    ; JMP FAR ProtCode:ProtMode
          DW        OFFSET ProtMode,OFFSET ProtCode

; Jump back here to return to real mode.

BackReal:
          XOR       AX,AX                   ; No Local Descriptor Table
          LLDT      AX
          MOV       AX,OFFSET MSDOSData     ; Reload ALL segment registers
          MOV       DS,AX                   ; to point them to DOS-type
          MOV       ES,AX                   ; segments
          MOV       FS,AX
          MOV       GS,AX
          MOV       SS,AX
          MOV       SP,0FFFEh               ; Just temporary, I promise!
          MOV       EAX,CR0                 ; Get current CR0
          AND       EAX,NOT 1               ; Reset protected mode bit
          MOV       CR0,EAX                 ; Back in real mode

;
; Now the trick is getting the assembler to assemble a FAR jump to the
; next instruction! It always optimises it out! So hard-code it again.
;
          DB        0EAh
          DW        OFFSET RealMode,SEG RealSeg ; JMP FAR RealSeg:RealMode
RealMode:
          MOV       AX,SEG StackSeg         ; Put stack back
          MOV       SS,AX
          MOV       SP,OFFSET StackTop      ; SP too
          MOV       AX,SEG Data             ; Re-reload all segment registers
          MOV       DS,AX                   ; in real mode
          MOV       ES,AX
          MOV       FS,AX
          MOV       GS,AX
          ASSUME    DS:Data,ES:Nothing,FS:Nothing,GS:Nothing ; Tell assembler
          LIDT      [OldIDT]                ; Restore IDT pointer

;
; Restore timer back to 18.2 times a second
;
          MOV       AL,0                    ; Divisor for sllllllooooooww!
          OUT       40h,AL                  ; Low byte
          JMP       $+2                     ; (tap foot)
          OUT       40h,AL                  ; High byte

;
; Restore Priority Interrupt Controllers (PICs) to DOS expectations
;
          MOV       AL,11h                  ; Initialise PICs
          OUT       20h,AL
          OUT       0A0h,AL

          MOV       AL,08h                  ; PIC1 @ Int 08h
          OUT       21h,AL
          MOV       AL,70h                  ; PIC2 @ Int 70h
          OUT       0A1h,AL

          MOV       AL,00000100b            ; PIC1 has slave on IRQ2
          OUT       21h,AL
```

```
          MOV       AL,02h              ; PIC2 is slaved to IRQ2
          OUT       0A1h,AL

          MOV       AL,01h              ; 80x86 mode
          OUT       21h,AL
          OUT       0A1h,AL

          MOV       AL,00h              ; Enable all interrupts
          OUT       21h,AL
          OUT       0A1h,AL

          MOV       AH,0DDh             ; Restore A20 line
          CALL      GateA20
FailA20:
          STI
          MOV       AH,1                ; Restore cursor
          MOV       CX,0607h
          INT       10h
          MOV       AH,2                ; Cursor to bottom of screen
          MOV       BH,0
          MOV       DX,1700h
          INT       10h
          MOV       AX,4C00h            ; Exit program
          INT       21h
Entry     ENDP

StoreDesc PROC      NEAR
          SHL       EAX,4                ; Turn into linear address
StoreLin:
          MOV       [BX.BaseLo],AX       ; Store into GDT
          SHR       EAX,16               ; Get high part of EAX
          MOV       [BX.BaseMid],AL      ; And store
          RET
StoreDesc ENDP

GateA20   PROC NEAR
          CLI                            ; Critical code!
          CALL      Wait8042             ; Wait for it to be ready
          JNZ SHORT Dead8042             ; Nope, so dead
          MOV       AL,0D1h              ; Program data port
          OUT       64h,AL               ; (to control port)
          CALL      Wait8042             ; Wait to accept command
          JNZ SHORT Dead8042             ; Nope, so dead
          MOV       AL,AH                ; Value to program
          OUT       60h,AL               ; (to data port)
Wait8042:
          XOR       CX,CX                ; Plenty long enough!
Loop8042:
          IN        AL,64h               ; Get current state
          AND       AL,2                 ; Busy?
          LOOPNZ    Loop8042             ; Yep, so keep waiting
Dead8042:
          RET                            ; No longer.
GateA20   ENDP

RealSeg   ENDS
```

```
;============================================================================
;
; Protected mode code segment (still USE16). Used on initial start in
; protected mode, to initialise all tasks, then again to return to DOS.
;
ProtSeg   SEGMENT   PARA USE16

          ASSUME    CS:ProtSeg,DS:Nothing,ES:Nothing,FS:Nothing,GS:Nothing

ProtMode:
          CLI                             ; Not ready yet!
          MOV       AX,OFFSET MainStack   ; Reload SS under protected mode
          MOV       SS,AX
          MOV       ESP,OFFSET StackTop   ; Available!

          PUSH LARGE 2                    ; All special bits zero
          POPFD                           ; In flags register

;
; Reprogram Peripheral Interrupt Controllers (PICs) to move them out of the
; way of the Intel-reserved interrupts.

;
          MOV       AL,11h                ; Initialise PICs
          OUT       20h,AL
          OUT       0A0h,AL

          MOV       AL,20h                ; PIC1 @ 20h
          OUT       21h,AL
          MOV       AL,28h                ; PIC2 @ 28h

          OUT       0A1h,AL

          MOV       AL,00000100b          ; PIC1 has slave on IRQ2
          OUT       21h,AL
          MOV       AL,02h                ; PIC2 is slaved on IRQ2
          OUT       0A1h,AL

          MOV       AL,1                  ; 80x86 mode
          OUT       21h,AL
          OUT       0A1h,AL

          MOV       AL,0                  ; Enable all interrupts
          OUT       21h,AL
          OUT       0A1h,AL
;
; Reprogram timer to speed up task switching.
;
          MOV       AX,ClockFreq/SecTick  ; Divisor

          OUT       40h,AL                ; Low byte
          JMP       $+2                   ; (tap foot)
          MOV       AL,AH                 ; High byte
          OUT       40h,AL

          MOV       AX,OFFSET GDTData     ; Point to desired segments
          MOV       DS,AX
          ASSUME    DS:GDT                ; And tell assembler!

          MOV       AX,OFFSET MSDOSData

          MOV       FS,AX
          ASSUME    FS:Data

          MOV       AX,OFFSET Screen
          MOV       ES,AX
          MOV       GS,AX
```

```
          MOV       FS:[TaskLock],1        ; Disable task switching for now
          MOV       FS:[LastTask],OFFSET MainTask ; Point to current last task

          MOV       AX,0700h+' '           ; Grey space
          XOR       DI,DI                  ; Screen pointer
          MOV       CX,2000                ; 2000 screen locations
          REP       STOSW                  ; Clear screen

          STI                              ; You can start now!

;
; Initialise remaining TSSs, in case a fault happens!
;
          MOV       EAX,SEG TSS
          SHL       EAX,4
          MOV       BX,OFFSET MainTask
          MOV       CX,OFFSET TSSSize
          MOV       DL,Present+System+Gate386+AvailTSS
          MOV       DH,ByteGran
          CALL      AssignMem
          LTR       BX                     ; Point Task Register to it

; Use memory above 1 Meg for TSS stacks
          MOV       ESI,110000h            ; Point to memory above 1Mb+64K

          MOV       EAX,SEG TSSDebug       ; Initialise Debug TSS
          MOV       BX,OFFSET DebugTSS     ; Point to descriptor table entry
          MOV       EBP,OFFSET DebugInt    ; Starting instruction to use
          CALL      AssignTSS

          MOV       EAX,SEG TSSDouble      ; Initialise Double TSS
          MOV       BX,OFFSET DoubleTSS    ; Point to descriptor table entry
          MOV       EBP,OFFSET DoubleInt   ; Starting instruction to use
          CALL      AssignTSS

          MOV       EAX,SEG TSSTSS         ; Initialise BadTSS TSS
          MOV       BX,OFFSET BadTSSTSS    ; Point to descriptor table entry
          MOV       EBP,OFFSET BadTSSInt   ; Starting instruction to use
          CALL      AssignTSS

          MOV       EAX,SEG BadStackTSS    ; Initialise BadStack TSS
          MOV       BX,OFFSET StackTSS     ; Point to descriptor table entry
          MOV       EBP,OFFSET StackInt    ; Starting instruction to use
          CALL      AssignTSS

          MOV       FS:[TaskLock],0        ; Can now enable task switching

; *** Insert any INT after this point ***
; (before this point I don't guarantee anything!)


;
; Now create each task
;
          MOV       BX,OFFSET UserTasks-SIZE Descriptor
          MOV       CX,(80/(WindowWidth+2))*(25/(WindowHeight+2)) ; No. tasks
TaskLoop:
          PUSH      CX

; Allocate LDT
          MOV       CX,OFFSET LDTLimit       ; Local descriptor table size
          CALL      GetMem                   ; Allocate memory
          MOV       DL,Present+Memory+Writable ; As data segment
          MOV       DH,ByteGran
          CALL      AssignMem                ; Modify GDT

; Initialise LDT
          PUSH      DS
```

```
          PUSH      BX                          ; Save GDT pointer
          MOV       DS,BX                       ; Point to LDT
          MOV       BX,-SIZE Descriptor

; Allocate Stack0
          MOV       CX,StackSize-1              ; Size of stack
          CALL      GetMem                      ; Get Stack0
          MOV       DL,Present+Memory+Writable ; As data segment
          MOV       DH,ByteGran+More64k
          CALL      AssignMem                   ; Modify LDT

; Allocate Stack3
          MOV       CX,StackSize-1              ; Size of stack
          CALL      GetMem                      ; Get Stack3
          MOV       DL,Present+Privilege3+Memory+Writable ; As data segment
          MOV       DH,ByteGran+More64k
          CALL      AssignMem                   ; Modify LDT

; Point to user code
          MOV       EAX,SEG BallCode            ; Code segment
          SHL       EAX,4                       ; As linear address
          ADD       BX,SIZE Descriptor          ; Next descriptor
          MOV       CX,OFFSET CodeLimit         ; Bytes of code
          MOV       DL,Present+Privilege3+Memory+Execable+Readable ; As code
          MOV       DH,ByteGran+LargeAddr
          CALL      AssignMem                   ; Modify LDT

; Allocate user data
          MOV       CX,OFFSET DataLimit         ; Size of data
          CALL      GetMem                      ; Get TaskData
          MOV       DL,Present+Privilege3+Memory+Writable ; As data segment
          MOV       DH,ByteGran
          CALL      AssignMem                   ; Modify LDT

          POP       BX                          ; Restore GDT pointer
          POP       DS
          ASSUME    DS:GDT                      ; So tell assembler

          MOV       [BX.DescType],Present+Privilege3+System+LDT ; Is now LDT!

;
; Initialise TaskData to maintain uniqueness
; To pre-initialise data, access is needed to the two variables that define
; the window. To get this access requires loading a segment register. The
; descriptor for this segment is in the LDT for the task. So, temporarily
; point the LDT for the Main task to this LDT, load the segment register,
; and access the variables. BUT. The LDTR is NOT automatically saved on a
; task switch, and the LDT field in MainTask's TSS is zero, not this temporary
; LDT. So if any task switches happen in this critical region, on return a
; GP fault for ES will occur (points to a non-existant LDT!).
;
          CLI                                   ; INTs can stuff up LDTR
          LLDT      BX                          ; Use as LDT for now
          MOV       AX,OFFSET TaskData+LocalDT ; Point to allocated TaskData
          MOV       ES,AX                       ; With ES
          ASSUME    ES:BallData                 ; And tell assembler

          MOV       AX,FS:[Corner]              ; Get global corner
          MOV       ES:[Top],AH                 ; Store as Top
          MOV       ES:[Left],AL                ; And Left

          XOR       AX,AX                       ; Don't point to LDT
          MOV       ES,AX                       ; with ES
          LLDT      AX                          ; anymore!
          STI                                   ; LDTR safe again

          ADD BYTE PTR FS:[Corner],WindowWidth+2    ; Next across
          CMP BYTE PTR FS:[Corner],80-WindowWidth   ; Too far?
```

```
          JB  SHORT AllocTSS                        ; Not yet
          ADD BYTE PTR FS:[Corner+1],WindowHeight+2 ; Yes, so next down
          MOV BYTE PTR FS:[Corner],1                ; And start again
AllocTSS:
          MOV       CX,TSSLimit                ; Size of a TSS
          CALL      GetMem                     ; Get memory for TSS
          MOV       DL,Present+Memory+Writable ; As a data segment
          MOV       DH,ByteGran
          CALL      AssignMem                  ; Modify GDT

          PUSH      DS
          PUSH      ES
          MOV       DS,BX                      ; Point to new segment
          MOV       ES,BX
          ASSUME    DS:TSS                     ; And tell (fool) assembler

          XOR       EAX,EAX                      ; Zero TSS
          XOR       DI,DI
          MOV       CX,TSSSize/4
          REP       STOSD

          MOV       [ESP0],StackSize           ; Initialise Stack0
          MOV       [SS0],OFFSET Stack0+LocalDT
          MOV       [EIPReg],OFFSET BallEntry  ; Entry point
          MOV       [EFlags],3202h             ; IOPL = 3!
          MOV       [ESPReg],StackSize
          MOV       [CSSeg],OFFSET TaskCode+LocalDT+3 ; DPL=3 sets CPL
          MOV       [SSSeg],OFFSET Stack3+LocalDT+3   ; DPL=3
          MOV       [DSSeg],OFFSET TaskData+LocalDT+3 ; DPL=3
          MOV       [LDTR],BX                  ; LDT is current descriptor..
          SUB       [LDTR],SIZE Descriptor     ; ...minus 1
          MOV       [IOMapBase],TSSSize        ; No need to worry about I/O

          POP       ES                         ; Don't point to TSS any more
          POP       DS                         ; Restore GDT pointer
          ASSUME    DS:GDT                     ; So tell assembler

          MOV       [BX.DescType],Present+System+Gate386+AvailTSS ; Now a TSS!
          MOV       FS:[LastTask],BX           ; And can be switched to

          POP       CX                         ; Get task counter
          DEC       CX                         ; Any left?
          JNZ       TaskLoop                   ; Yes, so continue

          MOV       FS:[TaskLock],0            ; Can now start task switching
          AND       [MainTask.DescType],NOT Present ; But not here!
TaskEnd:
          JMP       TaskEnd                    ; Wait for keypress to leave

BackToDOS:
          CLI                                  ; Disable interrupts
; Same assembler problem: protected mode JMP
          DB        0EAh                       ; JMP FAR MSDOSCode:BackReal
          DW        OFFSET BackReal,OFFSET MSDOSCode

GetMem    PROC NEAR
          MOV       EAX,ESI                ; New address
          MOVZX     ECX,CX                 ; Size to allocate
          ADD       ESI,ECX                ; Allocated!
          AND       ESI,NOT 0FFh           ; (Ensure on 256-byte boundary)
          ADD       ESI,100h               ; And don't overlap!
          ADD       BX,SIZE Descriptor     ; New descriptor too
          RET
GetMem    ENDP

AssignMem PROC NEAR
          MOV       [BX.LimitLo],CX        ; Low limit
          MOV       [BX.BaseLo],AX         ; Low base
```

```
          SHR       EAX,16                  ; Get high base
          MOV       [BX.BaseMid],AL         ; Middle base
          MOV       [BX.DescType],DL        ; Type
          MOV       [BX.DescGran],DH        ; Granularity
          MOV       [BX.BaseHi],AH          ; High base (usually zero!)
          RET
AssignMem ENDP

AssignTSS PROC      NEAR
          SHL       EAX,4                   ; Make address linear
          MOV       CX,OFFSET TSSLimit      ; Size of segment
          MOV       DL,Present+Memory+Writable ; Type to allocate
          MOV       DH,ByteGran
          CALL      AssignMem               ; Assign to descriptor

          PUSH      DS                      ; Point to TSS temporarily
          PUSH      ES
          MOV       DS,BX                   ; (Note still a data descriptor)
          MOV       ES,BX
          ASSUME    DS:TSS                  ; Tell (fool) assembler

          XOR       AX,AX                   ; Zero TSS
          XOR       DI,DI
          MOV       CX,TSSSize/2
          REP       STOSW
          MOV       [EIPReg],EBP            ; Starting instruction
          MOV       [EFlags],0202h          ; Flags to use
          MOV       [ESPReg],OFFSET StackTop ; Stack pointer to use
          MOV       [CSSeg],OFFSET IntCode  ; Code segment to use
          MOV       [SSSeg],BX              ; Stack to use
          ADD       [SSSeg],SIZE Descriptor  ; (1 past TSS)
          MOV       [IOMapBase],TSSSize     ; Don't worry about I/O
          POP       ES                      ; Don't point to TSS any more!
          POP       DS
          ASSUME    DS:GDT
          MOV       [BX.DescType],Present+System+Gate386+AvailTSS ; Now a TSS!

          MOV       CX,OFFSET StackTop        ; Allocated stack size
          CALL      GetMem                    ; Next descriptor
          MOV       DL,Present+Memory+Writable ; Writable memory
          MOV       DH,ByteGran+More64k       ; 32-bit access
          CALL      AssignMem                 ; Modify LDT
          RET
AssignTSS ENDP

ProtLimit EQU       $-1                     ; Limit of ProtSeg

ProtSeg   ENDS
```

ISE

İletiyi şu dile çevir: Türkçe

```
;=============================================================================
;
; User data and code. Note that code is set up for entry by initial values in
; TSS.
```

```
;
BallData  SEGMENT   PARA USE16
Top       DB        ?
Left      DB        ?
Bottom    DB        ?
Right     DB        ?
XLoc      DB        ?
YLoc      DB        ?
DeltaX    DB        ?
DeltaY    DB        ?
Counts    DD        ?
DataLimit EQU       $-1
BallData  ENDS

BallCode  SEGMENT   PARA USE32

          ASSUME    CS:BallCode,DS:BallData,ES:Nothing,FS:Nothing,GS:Nothing

BallEntry PROC
          MOV       AX,OFFSET Screen         ; Point to screen
          MOV       ES,EAX                   ; With ES
          MOV       AL,[Top]                 ; Turn current Top
          MOV       [YLoc],AL                ; Into YLoc
          MOV       [Bottom],AL              ; And bottom
          ADD       [Bottom],WindowHeight-1  ; Shift bottom down
          DEC       AL                       ; Turn into memory address
          MOV       AH,80*2                  ; Row factor
          MUL       AH
          MOVZX     EBX,AX                   ; Into pointer

          MOVZX     EAX,[Left]               ; Get column
          MOV       [XLoc],AL                ; Into XLoc
          MOV       [Right],AL               ; And right
          ADD       [Right],WindowWidth-1    ; Shift right across
          DEC       EAX                      ; Turn into memory address
          SHL       EAX,1
          ADD       EBX,EAX                  ; Add into pointer

          MOV       EDI,EBX                  ; Get into STO pointer
          CALL      DrawFrame                ; And draw the frame
          ADD       EBX,2*80+2               ; First position in frame
          MOV BYTE PTR ES:[EBX],Ball         ; Store ball into position
          MOV       [DeltaX],1               ; Head to the right
          MOV       [DeltaY],1               ; And head down
          MOV       [Counts],0               ; Count to time
          STD
Bounce:
          MOV       ECX,BounceTime           ; Wait to bounce
BounceDelay:
          LOOP      BounceDelay              ; (tap foot)

          MOV       EBP,EBX                  ; Old position

; Calculate new position
          MOV       AL,[YLoc]                ; Get current YLoc
          ADD       AL,[DeltaY]              ; Add in Delta
          CMP       AL,[Top]                 ; Too high?
          JB  SHORT YWrap                    ; Yes, so wrap
          CMP       AL,[Bottom]              ; Too low?
          JNA SHORT NoYWrap                  ; No, so don't wrap
YWrap:
          NEG       [DeltaY]                 ; Reverse direction
          ADD       AL,[DeltaY]              ; Undo mistake
          ADD       AL,[DeltaY]              ; Head properly!
NoYWrap:
          MOV       [YLoc],AL                ; Save away
          MOV       AH,80*2                  ; Row factor
          MUL       AH                       ; Turn into memory address
```

```
          MOVZX     EBX,AX                  ; Into index register

          MOVZX     EAX,[XLoc]              ; Get current XLoc
          ADD       AL,[DeltaX]             ; Add in Delta
          CMP       AL,[Left]               ; Too left?
          JB  SHORT XWrap                   ; Yes, so wrap
          CMP       AL,[Right]              ; Too right?
          JNA SHORT NoXWrap                 ; No, so don't wrap
XWrap:
          NEG       [DeltaX]                ; Reverse direction
          ADD       AL,[DeltaX]             ; Undo mistake
          ADD       AL,[DeltaX]             ; Head properly!
NoXWrap:
          MOV       [XLoc],AL               ; Save away
          MOV       AH,0                    ; Turn into memory address
          SHL       EAX,1                   ; Column factor
          ADD       EBX,EAX                 ; Add into memory address
          MOV BYTE PTR ES:[EBP],' '         ; Delete old ball
          MOV WORD PTR ES:[EBX],0F00h+Ball  ; Paint new ball
          JMP       Bounce                  ; And bounce forever!
BallEntry ENDP

DrawFrame PROC      NEAR
          XOR       ECX,ECX                 ; Zero high part of ECX
          MOV       AH,0Fh                  ; White background
          MOV       AL,'Ú'                  ; TopLeft corner
          STOSW                             ; Store on screen
          MOV       AL,'Ä'                  ; Top
          MOV       CL,WindowWidth          ; This many
          REP       STOSW                   ; Store on screen
          MOV       AL,'¿'                  ; TopRight corner
          STOSW                             ; Store on screen
          ADD       EDI,2*(80-WindowWidth-2) ; Next row
          MOV       CL,WindowHeight         ; Height
          JCXZ      NoSide                  ; None!
Side:
          PUSH      ECX                     ; Need this later
          MOV       AL,'³'                  ; A side
          STOSW                             ; Store on screen
          MOV       AL,' '                  ; Blank
          MOV       CL,WindowWidth          ; This many
          REP       STOSW                   ; Store on screen
          MOV       AL,'³'                  ; Other side
          STOSW                             ; Store on screen
          ADD       EDI,2*(80-WindowWidth-2) ; Next row
          POP       ECX                     ; Restore row count
          LOOP      Side
NoSide:
          MOV       AL,'À'                  ; Bottom left
          STOSW                             ; Store on screen
          MOV       AL,'Ä'                  ; Bottom
          MOV       CL,WindowWidth          ; This many
          REP       STOSW                   ; Store on screen
          MOV       AL,'Ù'                  ; Bottom right
          STOSW                             ; Store on screen
          RET
DrawFrame ENDP

CodeLimit EQU   $-1

BallCode  ENDS


;===========================================================================
;
; This segment is for interrupt code. As well as the hardware interrupt
; handlers, there is also code to handle the various faults. Most of the
; code simply displays all of the registers' contents, then bombs back to
```

```
; DOS. The timer and keyboard interrupt, however, is actually used.
; Note this segment is marked as being USE32. This was both as an experiment,
; and also because most instructions used will be 32-bit instructions, since
; I want to be able to display all of each register.
;
IntSeg    SEGMENT   PARA USE32

          ASSUME    CS:IntSeg,DS:Nothing,ES:Nothing,FS:Nothing,GS:Nothing

DivideInt:
          PUSH      0                     ; Pseudo-error code
          PUSH      0                     ; INT 0
          JMP       Interrupt             ; Show registers

;
; Simple code. Entered on single step interrupt, waits for Enter or Space,
; then returns. Note dingle<tm> in top right corner indicates 'waiting'.
; Pressing Enter stops the single-step action
;
DebugInt  PROC
          PUSH      EBX                   ; Need these registers
          PUSH      DS
          PUSH      ES
          MOV       BX,OFFSET MSDOSData   ; Point to Data
          MOV       DS,EBX

          ASSUME    DS:Data               ; And tell assembler

          MOV       BX,OFFSET Screen      ; Point to screen
          MOV       ES,EBX
          STI
KeyLoop:
          INC BYTE PTR ES:[009Eh]         ; Dingle<tm> screen location
          CMP       [KeyPress],57         ; Space pressed?
          JE  SHORT EndDebug              ; Yes, so leave
          CMP       [KeyPress],28         ; Enter pressed?
          JNE       KeyLoop               ; No, so keep waiting
          AND WORD PTR [ESP+14h],NOT 100h ; Yes, so stop tracing
EndDebug:
          MOV       [KeyPress],0          ; Key? What key?
          POP       ES
          POP       DS
          POP       EBX
          IRETD                           ; Return from task
          JMP       DebugInt              ; Re-entry will appear here
DebugInt  ENDP

;
; If above not required, use this instead (modify IDT).
;
BadDebugInt:
          PUSH      0                     ; Pseudo-error code
          PUSH      1                     ; INT 1
          JMP       InterTask             ; Get values from TSS
NMIInt:
          PUSH      0                     ; Pseudo-error code
          PUSH      2                     ; INT 2
          JMP       Interrupt             ; Display registers
BreakInt:
          PUSH      0                     ; Pseudo-error code
          PUSH      3                     ; INT 3
          JMP       Interrupt             ; Display registers

OverInt:
          PUSH      0                     ; Etcetera
          PUSH      4                     ; Etcetera
          JMP       Interrupt             ; Etcetera
BoundInt:
```

```
          PUSH      0
          PUSH      5
          JMP       Interrupt
OpInt:
          PUSH      0
          PUSH      6
          JMP       Interrupt
No387Int:
          PUSH      0
          PUSH      7
          JMP       Interrupt
DoubleInt:
          PUSH      31                    ; Note no pseudo-error code, as
          JMP       InterTask             ; Double Faults have a real one
Over387Int:
          PUSH      0
          PUSH      9
          JMP       Interrupt
BadTSSInt:
          PUSH      10                    ; Note no pseudo-error code
          JMP       InterTask             ; Display registers from TSS

NoSegInt  PROC
          PUSH      EBX                   ; Oops, forgot to mark as present!
          PUSH      DS                    ; (Didn't really, but good test)
          MOV       BX,OFFSET Screen
          MOV       DS,EBX
          INC BYTE PTR DS:[0014h]         ; Indicate on screen

          MOV       BX,OFFSET GDTData     ; Point to GDT
          MOV       DS,EBX
          ASSUME    DS:GDT
          MOV       BX,[ESP+8]            ; Get error code
          AND       BX,NOT 07h            ; Ignore extra bits
          OR        [BX.DescType],Present ; Mark segment as present
          POP       DS
          POP       EBX
          ADD       ESP,4                 ; Ignore error code
          IRETD                           ; And restart instruction
NoSegInt  ENDP

StackInt:
          PUSH      12                    ; Note no pseudo-error code
          JMP       InterTask             ; Display registers from TSS
GenInt:
          PUSH      13                    ; Note no pseudo-error code
          JMP       Interrupt
PageInt:
          PUSH      14                    ; Note no pseudo-error code
          JMP       Interrupt
Int15:
          PUSH      0                     ; Push pseudo-error code
          PUSH      15
          JMP       Interrupt
Bad387Int:
          PUSH      0
          PUSH      16
          JMP       Interrupt
Int17:
          PUSH      0
          PUSH      17
          JMP       Interrupt

Int18:
          PUSH      0
          PUSH      18
          JMP       Interrupt
Int19:
```

```
          PUSH      0
          PUSH      19
          JMP       Interrupt
Int20:
          PUSH      0
          PUSH      20
          JMP       Interrupt
Int21:
          PUSH      0
          PUSH      21
          JMP       Interrupt
Int22:
          PUSH      0
          PUSH      22
          JMP       Interrupt
Int23:
          PUSH      0
          PUSH      23
          JMP       Interrupt
Int24:
          PUSH      0
          PUSH      24
          JMP       Interrupt
Int25:
          PUSH      0
          PUSH      25
          JMP       Interrupt
Int26:
          PUSH      0
          PUSH      26
          JMP       Interrupt
Int27:
          PUSH      0
          PUSH      27
          JMP       Interrupt
Int28:
          PUSH      0
          PUSH      28
          JMP       Interrupt
Int29:
          PUSH      0
          PUSH      29
          JMP       Interrupt
Int30:
          PUSH      0
          PUSH      30
          JMP       Interrupt
Int31:
          PUSH      0
          PUSH      31
          JMP       Interrupt




TimerInt  PROC
          PUSH      EAX                 ; Need these registers
          PUSH      EBX
          PUSH      DS
          PUSH      ES
          MOV       AL,20h              ; Acknowledge interrupt in PIC
```

```
          OUT       20h,AL                ; (Note interrupts still off)

          MOV       AX,OFFSET MSDOSData   ; Point to Data
          MOV       DS,EAX
          ASSUME    DS:Data               ; And tell assembler

          CMP       [TaskLock],0          ; Task switching locked out?
          JNE SHORT NoSwitch              ; Yes, so do nothing

          MOV       AX,OFFSET GDTData     ; Point to GDT alias
          MOV       ES,EAX
          ASSUME    ES:GDT                ; And tell assembler

          STR       AX                    ; Get current task number
          MOVZX     EBX,AX                ; Into index pointer
TestTask:
          ADD       EBX,SIZE Descriptor   ; Look at next descriptor
          CMP       BX,[LastTask]         ; Too far?
          JBE SHORT NotEndGDT             ; Not yet
          MOV       EBX,OFFSET MainTask   ; Yes, so start again
NotEndGDT:
          CMP       BX,AX                 ; Back here again?
          JE  SHORT NoSwitch              ; Yes, so none to switch to
          CMP BYTE PTR ES:[EBX.DescType],Present+Gate386+AvailTSS ; Is TSS?
          JNE       TestTask              ; No, so keep looking

          MOV       [NextTask],BX         ; New task!
          JMP       [TaskJump]            ; So jump to it (task switching)
NoSwitch:
          POP       ES
          POP       DS                    ; When jumps back, continues here
          POP       EBX
          POP       EAX
          IRETD                           ; So return where you left off
TimerInt  ENDP

KeyInt    PROC
          PUSH      EAX                   ; Need these registers
          PUSH      DS
          MOV       AX,OFFSET MSDOSData   ; Point to data
          MOV       DS,EAX

          ASSUME    DS:Data               ; And tell assembler

          MOV       AL,20h                ; Acknowledge PIC
          OUT       20h,AL                ; (Note interrupts still off)
          IN        AL,60h                ; Get character from keyboard
          MOV       [KeyPress],AL         ; Store in global data
          CMP       [KeyPress],129        ; Is it Esc released?
          JE        IntDOS                ; Yes, so back to DOS!
          POP       DS                    ; No, so continue
          POP       EAX
          IRETD
KeyInt    ENDP




SlaveInt:
          PUSH      0                     ; Pseudo-error code
          PUSH      34                    ; Note this interrupt is not
          JMP SHORT IRQ                   ; possible - it is the cascade.
COM2Int:
          PUSH      0
          PUSH      35
```

```
          JMP SHORT IRQ                    ; Acknowledge PIC1
COM1Int:
          PUSH      0
          PUSH      36
          JMP SHORT IRQ
IRQ5:
          PUSH      0
          PUSH      37
          JMP SHORT IRQ
IRQ6:
          PUSH      0
          PUSH      38
          JMP SHORT IRQ
PrintInt:
          PUSH      0
          PUSH      39
          JMP SHORT IRQ
IRQ8:
          PUSH      0
          PUSH      40
          JMP SHORT IRQB                   ; Acknowledge PIC2
IRQ9:
          PUSH      0
          PUSH      41
          JMP SHORT IRQB
IRQ10:
          PUSH      0
          PUSH      42
          JMP SHORT IRQB
IRQ11:
          PUSH      0
          PUSH      43
          JMP SHORT IRQB
IRQ12:
          PUSH      0
          PUSH      44
          JMP SHORT IRQB
IRQ13:
          PUSH      0
          PUSH      45
          JMP SHORT IRQB
IRQ14:
          PUSH      0
          PUSH      46
          JMP SHORT IRQB
IRQ15:
          PUSH      0
          PUSH      47
IRQB:
          PUSH      EAX                    ; Acknowledge PIC2
          MOV       AL,20h
          OUT       0A0h,AL
          POP       EAX
IRQ:
          PUSH      EAX                    ; Acknowledge PIC1
          MOV       AL,20h
          OUT       20h,AL
          POP       EAX


Interrupt PROC
          PUSHAD                           ; Save all registers
          PUSH      DS
          PUSH      ES
          PUSH      FS
          PUSH      GS
          PUSH      SS
          STR       AX                     ; Including faulting task
```

```
          PUSH      EAX
          PUSH      CS
          POP       DS                      ; Point to code, for strings

          MOV       AX,OFFSET Screen        ; Point to screen
          MOV       ES,EAX
          MOV       ESI,OFFSET RegNames     ; Point to strings
          MOV       EDI,[ESP+56]            ; Interrupt number on stack here
;          SHL       EDI,1                   ; Turn into screen address
;          INC BYTE PTR ES:[EDI]             ; One of these!
          MOV       AX,OFFSET MSDOSData     ; Point to data
          MOV       FS,EAX
          MOV       FS:[TaskLock],1         ; Stop task switching!
          STI
          MOV       AH,4Fh                  ; White-on-red!
          AND       EDI,NOT 0Fh             ; Position to tab-stop
          ADD       EDI,160                 ; On next row
DumpRegs:
          CLD                               ; Work forwards
          MOV       ECX,5                   ; Five characters of text
NameLoop:
          LODSB                             ; Get character
          STOSW                             ; Store attrib+char
          LOOP      NameLoop                ; For each character
          LODSB                             ; Get size of register
          MOV       CL,AL                   ; Into counter
          LODSB                             ; Get position on stack
          MOVZX     EBX,AL                  ; Into offset
          LODSB                             ; Get rid of position in TSS
          MOV       EDX,[ESP+EBX]           ; Get value from stack
          CALL      Hex                     ; Display as hex
NoData:
          ADD       EDI,160                 ; Start new row
          AND       EDI,NOT 0Fh             ; At tab-stop
          CMP       ESI,OFFSET EndRegs      ; End of registers?
          JB        DumpRegs                ; No, so continue
          CMP BYTE PTR [ESP+56],32          ; Was it a hardware int?
          JAE SHORT EndInt                  ; Yes, so continue
IntDOS:
          CLI                               ; No, so back to DOS!
          DB        0EAh                    ; JMP FAR ProtCode:BackToDOS
          DD        OFFSET BackToDOS,OFFSET ProtCode
EndInt:
          ADD       ESP,8                   ; Ignore saved TR and SS
          POP       GS                      ; Pop everything else
          POP       FS
          POP       ES
          POP       DS
          POPAD
          ADD       ESP,8                   ; Ignore int number and error code
          IRETD                             ; And return
Interrupt ENDP




InterTask PROC
          MOV       AX,OFFSET MSDOSData     ; Point to data
          MOV       DS,EAX

          ASSUME    DS:Data                 ; And tell assembler

          MOV       [TaskLock],1            ; Stop task switching!
```

```
          MOV       AX,OFFSET Screen       ; Point to screen
          MOV       ES,EAX
          ASSUME    ES:Nothing

          MOV       AX,OFFSET GDTData      ; Point to GDT
          MOV       FS,EAX
          ASSUME    FS:GDT

          STR       BX                     ; Get current task
          MOV       FS:[BX.DescType],Present+Memory ; Turn it into memory
          MOV       GS,EBX                 ; Load into segment reg

          ASSUME    GS:TSS                 ; And tell (fool) assembler

          MOVZX     EBX,GS:[BackLink]      ; Get BackLink TSS
          CMP       EBX,0
          JE  SHORT NotLinked
          MOV       FS:[EBX.DescType],Present+Memory+Writable ; Make writable
          MOV       GS,EBX                 ; Load into segment reg
          MOV       GS:[BackLink],BX       ; Store this TR somewhere
NotLinked:
          PUSH      CS                     ; Point to strings
          POP       DS
          MOV       ESI,OFFSET RegNames    ; Point to strings
          MOV       EDI,[ESP]              ; Position screen pointer
          SHL       EDI,1
          AND       EDI,NOT 0Fh
          INC BYTE PTR ES:[EDI]
          ADD       EDI,80*2
Name2Loop:
          CLD                              ; Work forwards
          MOV       AH,4Fh                 ; White-on-red!
          MOV       ECX,5                  ; Five characters per string
CharLoop:
          LODSB                            ; Get character
          STOSW                            ; Store character+attribute
          LOOP      CharLoop               ; Once for each char
          LODSB                            ; Get length of data
          MOV       CL,AL                  ; Into counter
          LODSB                            ; Ignore position on stack
          LODSB                            ; Get position in TSS
          MOVZX     EBX,AL                 ; Into Index register
          MOV       EDX,GS:[EBX]           ; Get value
          CMP       AL,8                   ; Is it int number?
          JA  SHORT HexIt                  ; No, so hex it
          CMP       AL,0                   ; Is it task number?
          JE  SHORT HexIt                  ; Yes, so hex it
          MOV       EDX,[ESP+EBX-4]        ; No, so fish off stack
HexIt:
          CALL      Hex                    ; Display it
          ADD       EDI,160                ; Go to next row
          AND       EDI,NOT 0Fh            ; To previous tab-stop
          CMP       ESI,OFFSET EndTSSRegs  ; End of registers?
          JB        Name2Loop              ; Not yet

          JMP       IntDOS                 ; Yes, so leave
InterTask ENDP


Hex       PROC      NEAR
          STD                              ; Work backwards
          LEA       EDI,[EDI+ECX*2-2]      ; Point to end of number
HexLoop:
          MOV       AL,DL                  ; Get lowest byte
          AND       AL,0Fh                 ; Isolate low nybble
          ADD       AL,'0'                 ; Turn into ASCII
          CMP       AL,'9'                 ; Too far?
```

```
          JBE SHORT LoopHex               ; No
          ADD       AL,'A'-'9'-1          ; Yes, so turn into hex
LoopHex:
          STOSW                           ; Store
          SHR       EDX,4                 ; Shift in next nybble
          LOOP      HexLoop               ; Loop
          RET                             ; And return
Hex       ENDP

RegNames  DB        'Int: ',2,56,4        ; Name,width,stack pos,TSS pos
          DB        'EAX: ',8,52,40
          DB        'EBX: ',8,40,52
          DB        'ECX: ',8,48,44
          DB        'EDX: ',8,44,48
          DB        'ESI: ',8,28,64
          DB        'EDI: ',8,24,68
          DB        'ESP: ',8,36,56
          DB        'EBP: ',8,32,60
          DB        'EIP: ',8,64,32
          DB        'Flag:',8,72,36
          DB        'CS:  ',4,68,76
          DB        'DS:  ',4,20,84
          DB        'ES:  ',4,16,72
          DB        'FS:  ',4,12,88
          DB        'GS:  ',4,8,92
          DB        'SS:  ',4,4,80
          DB        'Task:',4,0,0
          DB        'Err: ',4,60,8
EndRegs   EQU       $
          DB        'LDT: ',4,-1,96
EndTSSRegs EQU      $

IntLimit  EQU       $-1

IntSeg    ENDS

          END       Entry
```