无敌模式Fire and Ice

梁晨 3180102160

Task1: 用GB找出生命值地址

- 尝试高精度分析

  按步骤操作：

  1.将GB.exe和FIRE.exe放进d盘的reverse02目录中。

  2.打开DosBoxDeb，输入mount c: d:\reverse02。将d:reverse02目录虚拟成c盘

  3.输入c: 进入c盘

  4.输入gb运行GB.exe

  5.输入fire运行FIRE.exe

  6.进入游戏开始玩耍，每当重新进入一局游戏或死亡时，设置Analysis Value为丢掉的命的数量，如开局时设为0，丢掉一条命时设为1，以此类推。不在gameover前设置新的需要 分析的变量。

  需要特别注意，一开始的Analysis Value不能设为0且第一次和第二次设置的Analysis Value不能重复。在这两个条件的限制下，尽量多设置分析变量，用光0x0C次机会，注意不单单在丢性命或重新开局时设制新的分析变量，也要玩一会儿，如走几步或打一个怪，就设置一个新的分析变量，这样可以充分排除无关变量的影响。

  结果如下：

可以发现，每次分析出来的变量取交集是空集。且如果在不同的几局游戏内进行一次分析，经常会得出符合要求的变量数count==0的情况。由此可知用高精度的方法分析出的充其量只是与生命值有关的缓存值。换用低精度分析。

- 尝试低精度分析

  步骤与高精度分析时相同，除了一开始设置Analysis Value时将模式设为L而不是H。在低精度分析下，更应该注意充分地排除无关变量，在不同时刻多多设置Analysis Value。

  分析结果如下：

进行多次分析取交集，发现此交集中有且仅有1000:9966这一个地址。于是我们基本可以断定与生命值有关的变量存在其中。且规律是1000:[9966]==FD+丢的命的数量。可以看到丢三条命，即gameover时恰好发生overflow。

- 锁住1000：9966进行验证

  如果还不放心，可以选中这个分析出的地址，进入Game Table，alt+1选中第一行，输入life，可以看到对应地址自动变为选中的地址，之后shift+1锁住此地址。继续游戏，发现已经变为无敌模式。

Task2：找出使生命减少的指令

- 设置硬件断点

  先alt+pause进入debug模式，然后输入bpm 1000:9966在此地址处设置一硬件断点，使得此处值被改变时，发生中断。按f5继续游戏。

```
DEBUG: Set memory breakpoint at 1000:9966
```

- 不同时机观察使生命值变化的指令

  以下三图分别为从三条命减到两条命，从两条命减到一条命，和从一条命到gameover时发生中断的状态截图：

  图1:

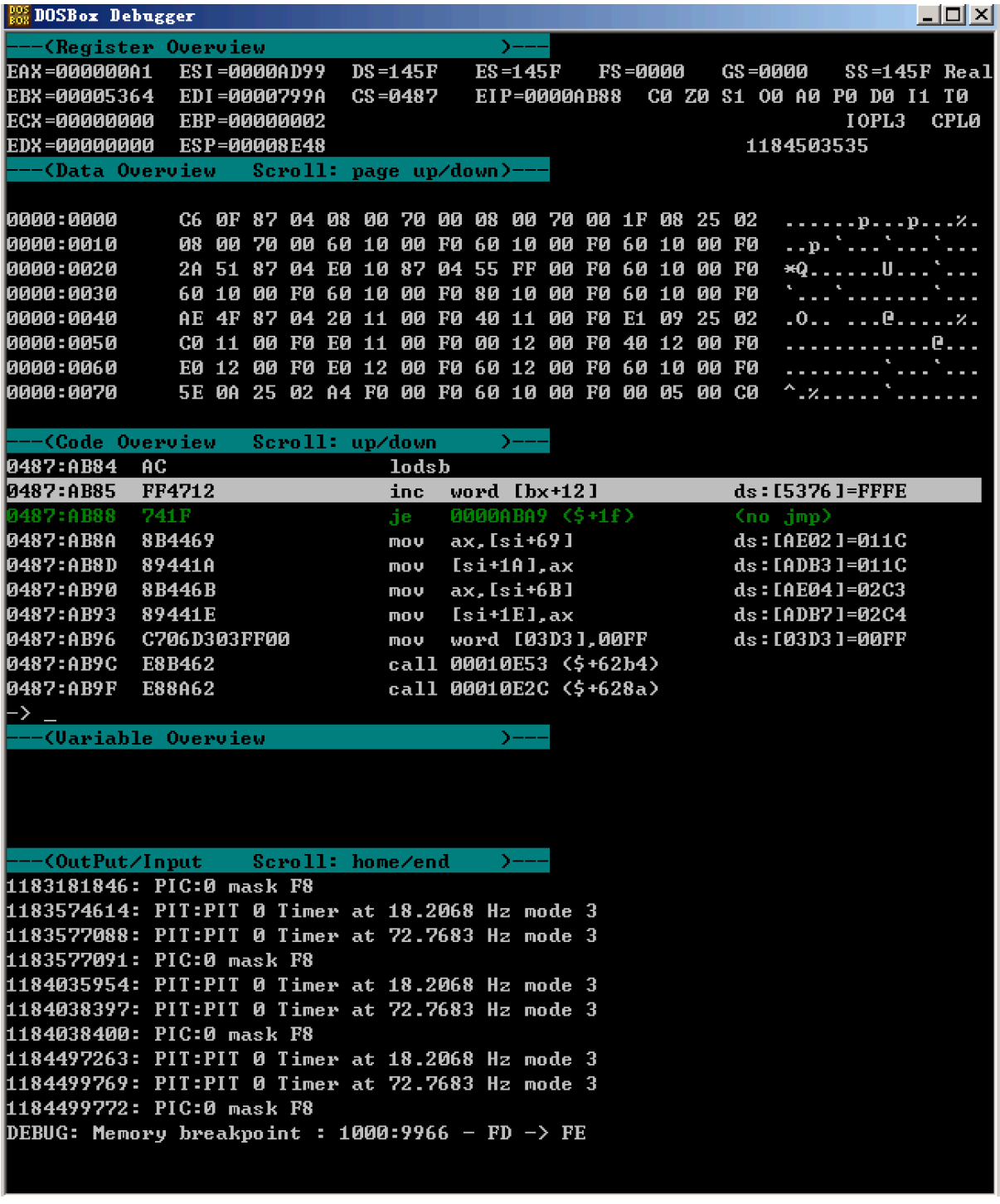


  图2:

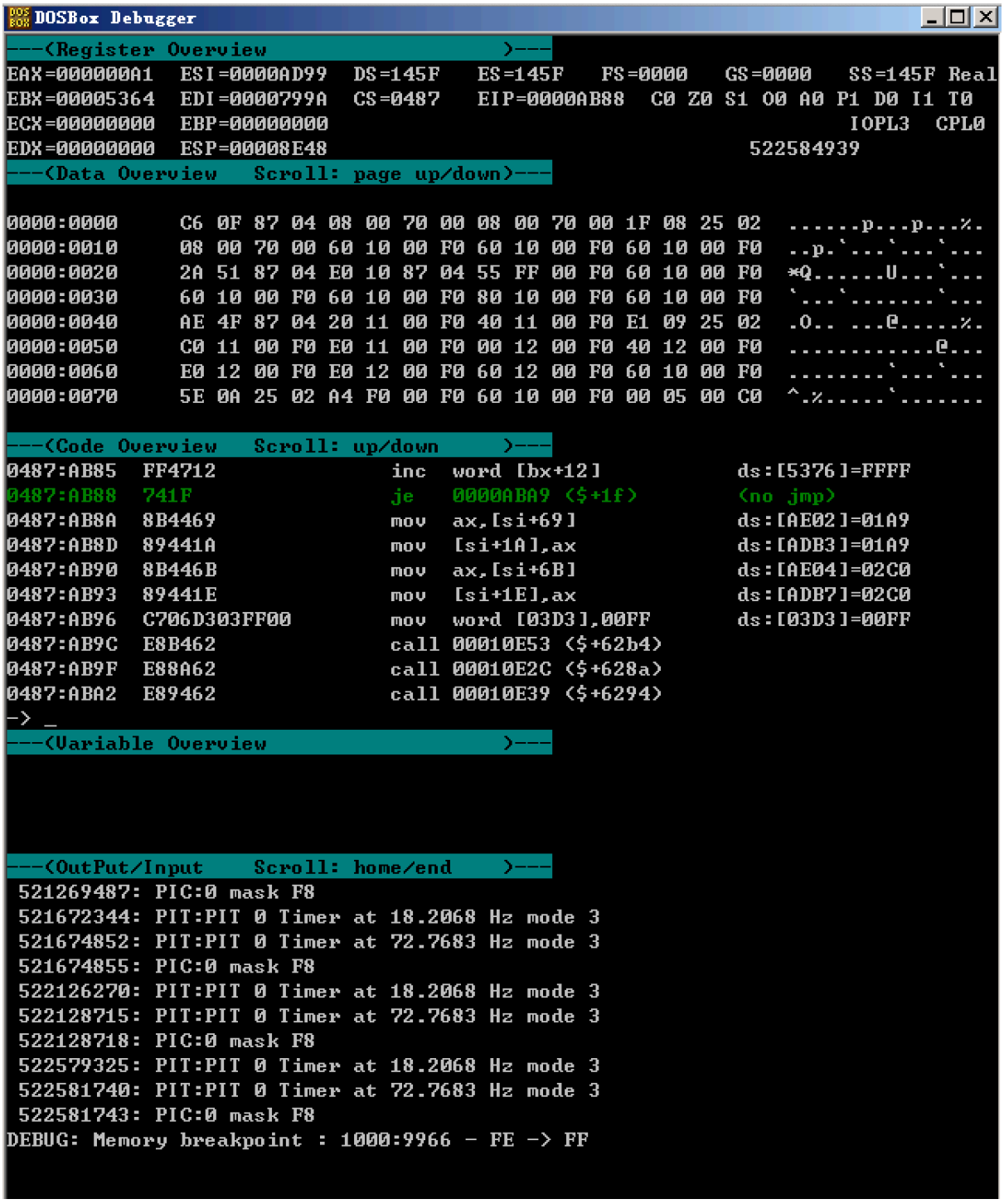
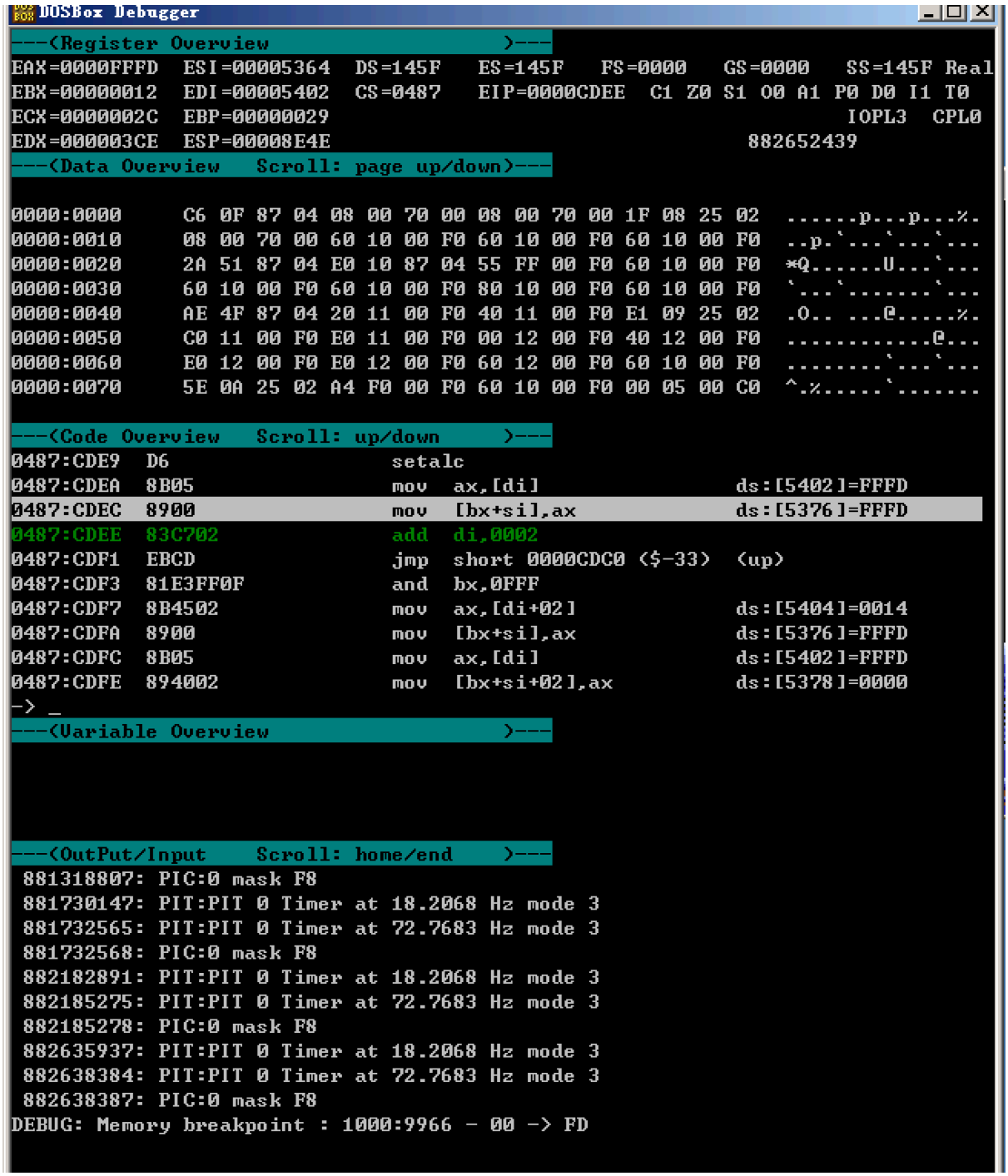

DOSBox Debugger
---(Register Overview)---
EAX=000000A1 ESI=0000AD99 DS=145F ES=145F FS=0000 GS=0000 SS=145F Real
EBX=00005364 EDI=0000799A CS=0487 EIP=0000AB88 C0 Z0 S1 O0 A0 P1 D0 I1 T0
ECX=00000000 EBP=00000000 IOPL3 CPL0
EDX=00000000 ESP=00008E48 522584939
---(Data Overview Scroll: page up/down)---
0000:0000 C6 0F 87 04 08 00 70 00 08 00 70 00 1F 08 25 02 ......p...p...%.
0000:0010 08 00 70 00 60 10 00 F0 60 10 00 F0 60 10 00 F0 ..p.`...`...`...
0000:0020 2A 51 87 04 E0 10 87 04 55 FF 00 F0 60 10 00 F0 *Q......U...`...
0000:0030 60 10 00 F0 60 10 00 F0 80 10 00 F0 60 10 00 F0 `...`.......`...
0000:0040 AE 4F 87 04 20 11 00 F0 40 11 00 F0 E1 09 25 02 .O.. ...@.....%.
0000:0050 C0 11 00 F0 E0 11 00 F0 00 12 00 F0 40 12 00 F0 ....@...
0000:0060 E0 12 00 F0 E0 12 00 F0 60 12 00 F0 60 10 00 F0 ....`...`...
0000:0070 5E 0A 25 02 A4 F0 00 F0 60 10 00 F0 00 05 00 C0 ^.%.....`.......
---(Code Overview Scroll: up/down)---
0487:AB85 FF4712 inc word [bx+12] ds:[5376]=FFFF
0487:AB88 741F je 0000ABA9 ($+1f) (no jmp)
0487:AB8A 8B4469 mov ax,[si+69] ds:[AE02]=01A9
0487:AB8D 89441A mov [si+1A],ax ds:[ADB3]=01A9
0487:AB90 8B446B mov ax,[si+6B] ds:[AE04]=02C0
0487:AB93 89441E mov [si+1E],ax ds:[ADB7]=02C0
0487:AB96 C706D303FF00 mov word [03D3],00FF ds:[03D3]=00FF
0487:AB9C E8B462 call 00010E53 ($+62b4)
0487:AB9F E88A62 call 00010E2C ($+628a)
0487:ABA2 E89462 call 00010E39 ($+6294)
-> _
---(Variable Overview)---
---(OutPut/Input Scroll: home/end)---
521269487: PIC:0 mask F8
521672344: PIT:PIT 0 Timer at 18.2068 Hz mode 3
521674852: PIT:PIT 0 Timer at 72.7683 Hz mode 3
521674855: PIC:0 mask F8
522126270: PIT:PIT 0 Timer at 18.2068 Hz mode 3
522128715: PIT:PIT 0 Timer at 72.7683 Hz mode 3
522128718: PIC:0 mask F8
522579325: PIT:PIT 0 Timer at 18.2068 Hz mode 3
522581740: PIT:PIT 0 Timer at 72.7683 Hz mode 3
522581743: PIC:0 mask F8
DEBUG: Memory breakpoint : 1000:9966 - FE -> FF

图3:



从每张图input/output一栏最后一行可以看出1000:9966处变量的变化为FD->FE->FF(->00)->FD，与Task1中的变化相同。在前两次，使得变量变化的指令为：

```
inc word [bx+12]
```

最后一次使得变量变化的指令为：

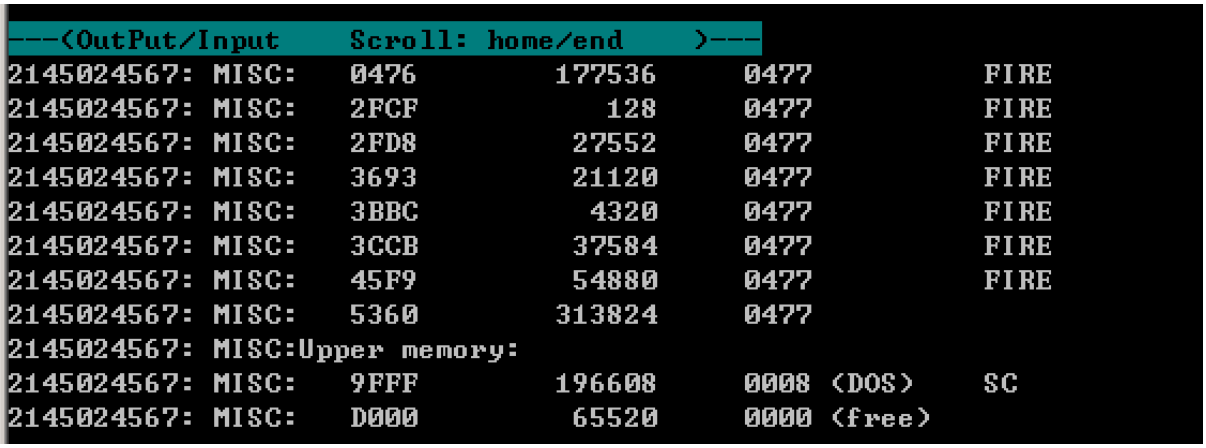
```
mov ax,[di]
mov [bx+si], ax
```

我们可以在图中看到前两次的[bx+12]和最后一次的[bx+si]即为ds:[5376]，又有ds == 145F。可以算出145F:5376==1000:9966。

Task3：计算相对于PSP的地址

- 输入dos mcbs查看PSP地址

  输出如下：



  可知psp段地址为0477。

- 计算

  我们又知道cs==0487。我们只要将0487:AB85处的指令改为nop即可，0487:AB85=0477:AC85。
  我们又知道存放生命值的地址为1000:9966，同样可以计算得到1000:9966=0477:EB96。

Task4: 写一时钟驻留程序，产生无敌模式

参照老师写的mypc.asm，http://10.71.45.100/bhh/mypc.asm，写出myfire.asm

```
dgroup group data, code
data segment
old_addr dw 0, 0
addr dw 0, 0
old_1h dw 0, 0
old_21h dw 0, 0
psp dw 0
data ends

code segment
assume cs:code, ds:data
int_8h:
  cmp cs:[fixed], 1
  je goto_old_8h ;if the code in fire is modified, go to old int_8h
  push ax
  push bx
  push ds
  lds bx, dword ptr cs:[psp_addr]
  mov ax, ds:[bx]
  add ax,10h ;get code segment addr
```

```
  mov ds,ax
  cmp dword ptr ds:[0AB85h], 0741247FFh
  jne skip ;if code at AB85 is not inc word [bx+12]
  cmp byte ptr ds:[0AB89h], 01Fh
  jne skip ;if code at AB88 is not je ABA9
  mov word ptr ds:[0AB85h], 9090h
  mov byte ptr ds:[0AB87h], 90h ; change inc word [bx+12] to nop
  mov cs:[fixed], 1
skip:
  pop ds
  pop bx
  pop ax
goto_old_8h:
  jmp dword ptr cs:[old_8h]
old_8h dw 0,0
psp_addr dw 0,0
fixed db 0

main:
  mov ax,data
  mov ds,ax
  call getpsp
  xor ax,ax
  mov es,ax
  mov bx, 8*4
  push es:[bx]
  pop cs:old_8h[0]
  push es:[bx+2]
  pop cs:old_8h[2]
  cli
  mov ax,offset int_8h
  mov es:[bx], ax
  mov es:[bx+2], cs
  sti
  mov dx, offset dgroup:main
  add dx,100h
  add dx, 0Fh
  mov cl, 4
  shr dx, cl ;len=(len+15)/16
  mov ah,31h
  int 21h

getpsp proc near
  push ax
  push bx
  push dx
  push es
  mov ah, 62h
  int 21h
```

```
  mov [psp], bx ;
  xor ax, ax
  mov es, ax
  mov bx, 4
  push es:[bx]
  pop old_1h[0]
  push es:[bx+2]
  pop old_1h[2] ;save old int_1h
  mov word ptr es:[bx], offset int_1h
  mov es:[bx+2], cs ;set new int_1h

  mov bx, 21h*4
  push es:[bx]
  pop old_21h[0]
  push es:[bx+2]
  pop old_21h[2] ;save old int_21h

  pushf
  push cs
  mov ax, offset return_here
  push ax
  mov ah, 62h ;get psp function
  pushf
  pop dx ;DX=FL
  or dx, 0100h
  push dx
  popf ;enable TF

  jmp dword ptr [old_21h]; after this instruction, int_1h will be executed
  ;int 21h/ah=62h to get psp

return_here:
  mov bx, 4
  push old_1h[0]
  pop es:[bx]
  push old_1h[2]
  pop es:[bx+2]
  pop es
  pop dx
  pop bx
  pop ax
  ret
getpsp endp

int_1h:
  push bp
  mov bp,sp
  push ax
  push di
```

```
  push ds
  push es
  mov ax,data
  mov ds,ax
  push addr[0]
  pop old_addr[0]
  push addr[2]
  pop old_addr[2]
  push [bp+2]
  pop addr[0]; addr[0]=ip
  push [bp+4]
  pop addr[2]; addr[2]=cs
  cmp bx, [psp]
  jne int_1h_iret
found_psp:
  les di, dword ptr [old_addr]
  mov ax, es:[di+2] ;0330=old cs:[old ip+2]
  mov cs:psp_addr[0], ax
  mov ax, [bp-6] ;old DS
  mov cs:psp_addr[2], ax ; get psp_addr old ds:[0330]
  mov ax, [bp+6] ; TF
  and ax, not 100h ; to disable TF
  mov [bp+6], ax
int_1h_iret:
  pop es
  pop ds
  pop di
  pop ax
  pop bp
  iret
code ends
end main
```

注意此程序还用到了单步调试获取存放psp段地址的地址psp_addr，此程序在真实DOS系统中可以运行，而在DosBox中不方便运行。在DosBox中运行的程序可直接用int 21h/ah=62获取psp，程序为：

```
dgroup group data, code
data segment
psp dw 0
data ends

code segment
assume cs:code, ds:data
int_8h:
  cmp cs:[fixed], 1
  je goto_old_8h ;if the code in fire is modified, go to old int_8h
  push ax
```

```
  push bx
  push ds
  lds bx, dword ptr cs:[psp_addr]
  mov ax, ds:[bx]
  add ax,10h ;get code segment addr
  mov ds,ax
  cmp dword ptr ds:[0AB85h], 0741247FFh
  jne skip ;if code at AB85 is not inc word [bx+12]
  cmp byte ptr ds:[0AB89h], 01Fh
  jne skip ;if code at AB88 is not je ABA9
  mov word ptr ds:[0AB85h], 9090h
  mov byte ptr ds:[0AB87h], 90h ; change inc word [bx+12] to nop
  mov cs:[fixed], 1
skip:
  pop ds
  pop bx
  pop ax
goto_old_8h:
  jmp dword ptr cs:[old_8h]
old_8h dw 0,0
psp_addr dw 0,0
fixed db 0

main:
  mov ax,data
  mov ds,ax
  call getpsp
  xor ax,ax
  mov es,ax
  mov bx, 8*4
  push es:[bx]
  pop cs:old_8h[0]
  push es:[bx+2]
  pop cs:old_8h[2]
  cli
  mov ax,offset int_8h
  mov es:[bx], ax
  mov es:[bx+2], cs
  sti
  mov dx, offset dgroup:main
  add dx,100h
  add dx, 0Fh
  mov cl, 4
  shr dx, cl ;len=(len+15)/16
  mov ah,31h
  int 21h

code ends
end main
```

Nov. 22, 2020