DOS Tool Hacking Notes

# 2015.7.20

## 实现功能

1. 屏幕初始化，设置蓝色背景，设置header and tail。
2. 任意位置显示任意字符（无法显示字符颜色）。
3. 划分屏幕为7个区块。

|  |  |  |
| --- | --- | --- |
| 0 4f | | |
| 1. 4f   3 4f | | |
| 4 5  Total 17  14 5 | 6 35  6 35 | 36 4f  36 4f |
| 15 4f  17 4f | | |
| 18 4f | | |

## 遗憾

1. 没有实现字符的颜色

## 明日任务

1. 读取键盘输入F6，实现F6功能

# 2015.7.21

## 实现功能

1. 键盘F6键盘编码读取，以及F6显示BDF相关内容。
2. 重构显示代码，分出header及tail。

## 明日任务

1. 打印键盘编码
2. 获取上下左右键盘编码，完善键盘服务程序
3. 设置键盘编码与功能之间的映射，粗略状态机
4. 设置某区域背景，前景颜色
5. 重构显示函数

# 2015.7.22

## 键盘编码

|  |  |  |
| --- | --- | --- |
| 按键 | Scan Code | ASCII Code |
| F6 | 40 | 00 |
| F7 | 41 | 00 |
| F10 | 44 | 00 |
| Pgup | 49 | 00 |
| Pgdown | 51 | 00 |
| ↑ | 48 | 00 |
| ↓ | 50 | 00 |
| ← | 4b | 00 |
| → | 4d | 00 |
| Enter | 1c | 0d |
| Esc | 01 | 1b |
| A | 1e | 61 |
| B | 30 | 62 |
| C | 2e | 63 |
| D | 20 | 64 |
| E | 12 | 65 |
| F | 21 | 66 |
| 1 | 02 | 31 |
| 2 | 03 | 32 |
| 3 | 04 | 33 |
| 4 | 05 | 34 |
| 5 | 06 | 35 |
| 6 | 07 | 36 |
| 7 | 08 | 37 |
| 8 | 09 | 38 |
| 9 | 0a | 39 |
| 0 | 0b | 30 |
| Alt+f | 21 | 00 |
| Alt+r | 13 | 00 |

## 明日任务

1. 实现F6状态机
2. 重构打印某行BDF数据

## BUG记录

1. 上翻，下翻刷屏问题；
2. F10退出黑屏

# 2015.7.23

## 实现任务

1. F6功能

## 本周任务

1. 读出一个PCIe设备的Configure Space Header
2. 让蜂鸣器Beep

## 协作问题

当两人都有不同看法时，一个人先说出“你先听我说”，就按照该人的想法走下去，如果发现错误，再执行另外一人想法。这样可以避免贞操掉光，打乱对方的走位。

## BUG记录

* 1. F6功能中，在第一行，多次按↑，按键无响应。按Esc后按F6会正常。

# 2015.7.26

## 实现任务

1. 蜂鸣器响应
2. 读取所有Device的Configuration Space
3. 从F6界面按enter键，进入主界面显示配置空间
4. 第5区域显示设备信息，例如申请的中断号…
5. 时间显示

## 明日任务

1. F7函数
2. Page up 和Page down功能

# 2015.7.27

## 实现功能

1. F10
2. Page up and Page down

## 明日任务

1. Function菜单
2. Read菜单
3. Device Name

# 2015.7.28

## 实现任务

1. Function菜单
2. Read菜单
3. Device Name String

## 明日任务

1. 商量出来中要实现的函数列表
2. 具体实现1个函数

# 2015.7.29

## 实现任务

1. 主界面高亮某一单位数据，并随键盘上下左右移动而做相应动作。

## 明日任务

1. 解决按F7后高亮数据切换问题。
2. 制作基础下拉菜单。

# 2015.7.30

## 实现任务

1. 实现F7

## 明日任务

1. 申请时钟中断实时刷新时间

# 2015.8.10

## BUG记录

1. BORLANDC 3.1编译器只支持10个字符的文件名字（不包括后缀名）。
2. 局部地址可能会变，建议用全局
3. 循环的值

## 实现功能

CPUID

## 下次任务

MMIO

# 2015.8.11

## E280

int 15h （杂项系统服务，Miscellaneous System Service）

参考下边的链接，其中有解释，有汇编代码，可以移植过来：

<http://wiki.osdev.org/Detecting_Memory_%28x86%29>

# 2015.8.24

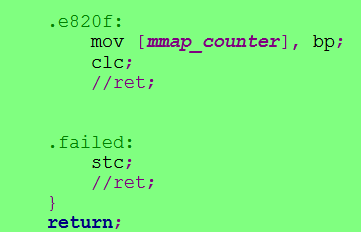
## 实现功能



Detect memory map（E820）；

将文件同步到git服务器上。

## BUG记录



函数中使用汇编语句ret返回无法退出函数；将ret删除，使用return返回功能正常；

## 下次任务

实现Memory test

# 2015.8.25

## 实现功能

Memory Map界面翻页显示

## 问题记录

如PIC中Mmap.jpg所示，第14段显示出Memory remap后的空间，为1FFFFFFF，合理解释为256\*32\*8\*4K。

## 下次任务

向Usable Memory Area读取/写入信息。

# 2015.8.26

## 实现功能

1. 优化显示代码，使界面切换、刷新过程不会出现明显的屏幕闪烁，即先黑屏后刷新。
2. 改变数据数据索引结构，将Memory map分离出cfg\_spa\_arr，而使用dev\_current进行指引。
3. 实现memory read and write

## 明日任务

实现memory test

# 2015.8.27

## 实现功能

Memory test

## 明日功能

1. Memory test 界面搭建
2. 查找BIOS有关SPD的服务接口

# 2015.8.28

## 实现功能

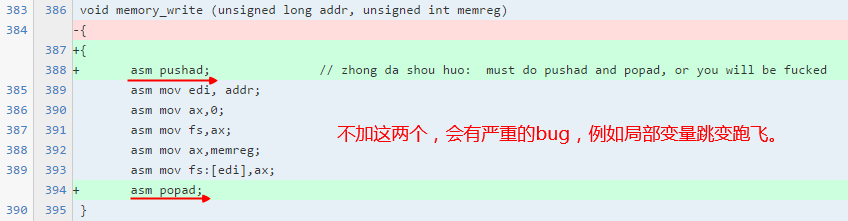
1. Memory test 界面搭建
2. Jason 写了一个大BUG

## Bug记录

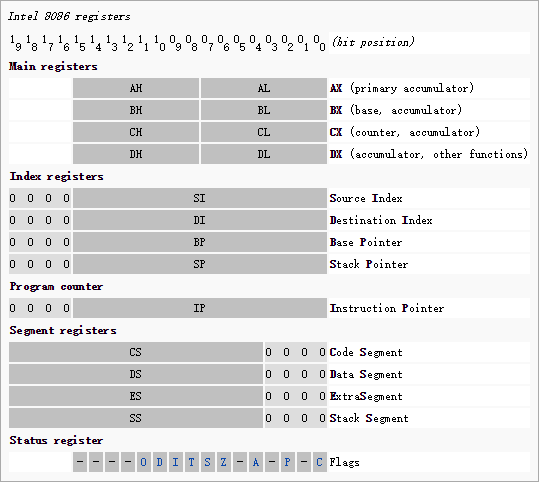
在C与assembly接口处，如未保护现场而调用EDI、EBP等寄存器，将导致附近变量混乱，该BUG严重违反ABI规定，在Function Calling Sequence部分有详细记录函数接口应如何操作。

## C与汇编混编陷阱

发现memory test隐藏的bug，修改工程中所有有关汇编与C语言之间调用结合的部分，在汇编部分中，进入之前要保存变量，退出之前要恢复所有变量。



进入memory\_write之后，我们的for循环中的局部变量jeff跳变跑飞。



首先介绍一下易失性，一些寄存器在函数中常常是变化的，而另外一些却是不变的。这是编译器所决定的。因为寄存器是不会自动保存的（虽然有些汇编语言会自动保存，但是x86 是不会的），所以编码时要自己保存。这句话的意思是：当一个函数被调用，是不保证在函数返回时，易失寄存器上的值不变的；但是函数必须负责保存非易失寄存器中的值。

微软编译器的寄存器使用习惯如下：

易失性寄存器：cx，dx

非易失性寄存器：bx，bp，si，di

其他特殊寄存器：ax，sp

**eax**的作用有2个，1. 作为函数返回值；2. 在做乘法和除法的时候作为专用寄存器。

下面是C 语言中函数返回的例子：

return 3; // 返回3

对应的汇编语言：

mov eax, 3 ; 置EAX=3

ret ; 返回

**ebx** 是一个非易失通用寄存器。它没有特定的用途，但是常被置为一个函数中常用的值（如0 ），以此来加快计算速度。

**ecx**是一个易失通用寄存器。常被用作函数的参数或者是循环的计数器。

\_\_fastcall 的函数会将第一和第二个参数放置在ecx和edx寄存器中。另外，当调用一个类中的成员函数时，不管调用习惯是什么，指向类的指针常常是放在ecx中。另外，ecx常被用作循环计算器。for循环一般（尽管不是总是）会将循环计数放在ecx中。rep指令也会将ecx作为计数器，自动减少直到0.

**edx** 是一个易失通用寄存器，偶尔会被用作函数的参数。就像ecx ，edx 常用在\_\_fastcall 调用的函数中。除了fastcall 调用中存放参数，编译器通常会将局部（短期）变量储存在edx 中。

esi 是一个非易失通用寄存器，常被用作指针。特别的，在rep 一类的指令中，esi 通常指向“源”。因为esi 中数据是不会改变的，所以esi 通常会存储不会变的数据。

edi 是一个非易失通用寄存器，常被用作指针。它和esi 差不多，只不过一般是作为“目标”指针。

ebp 是一个非易失通用寄存器，根据编译器的设置，它有两个截然不同的用途：要么作为框架指针，要么作为一般寄存器。若没有优化编译或者代码是手工写的，ebp 会在函数开头就保存着堆栈的位置。因为堆栈在整个函数过程中是不停变换的，将ebp 指向堆栈的原始位置可以使得方便使用存储在堆栈中的变量。如果编译被优化，当堆栈指针的计算通过指针的移动值计算时（这可能有点乱——IDA 就是自动侦测并校准一个移动的堆栈指针），ebp 就会作为一个保存任何数据的通用寄存器了。

esp 是一个保存着堆栈底端指针的特殊寄存器（堆栈是向低地址生长）。很少直接对ESP 进行数学计算（加减），而且函数的开始和结束时的esp值必须一致。

**参考文献**：

1. 《汇编-32位寄存器的功能及其使用之整理篇》
2. <http://www.cnblogs.com/technology/archive/2010/05/16/1736782.html> 汇编指令
3. <http://www.blogjava.net/youxia/archive/2015/01/29/linux008.html> GCC写16位代码
4. <http://blog.csdn.net/junmuzi/article/details/8605651> x86,ARM,MIPS架构函数调用分析
5. <http://nieyong.github.io/wiki_cpu/> CPU体系架构-ARM/MIPS/x86
6. <http://blog.csdn.net/phunxm/article/details/8985321> x86下C函数调用惯例

# 2015.9.7

## 问题列表

1. 如何在DOS下做32bit内存操作？
2. SMBUS检查Bus Controller是否Busy？
3. 汇编如何Delay？

# 2015.9.8

## 实现功能

Memory Test 以32位pattern进行测试。

# 2015.9.9

## 实现功能

1. 修复scan\_BDF有关class code数据处理bug；
2. 完善class code 解析代码；
3. 由于可预见数据存储数组将无法满足存储需求，更改数据结构，扩大cfg\_spa\_err 至100，memory map E820 功能代码调整至101，memory test 功能代码调整至102。

# 2015.9.14

## 完成功能

1. 添加User Guide界面module，可修改内容；
2. 完成SPD核心功能代码，即完成SMBus以byte形式读取函数。

## 明日任务

1、解析存储于cfg\_spa\_arr[43]内的SPD信息，以人类可理解方式打印于zone 5。