UEFI Development Exploration 35 – Option ROM Prequel 2

原创 luobing4365 Posted on 2019-10-15 23:26:59 Read 1.4k Collection 6 Likes 3 copyright

Category columns: UEFI Development Article Tags: UEFI Programming Low-level programming Legacy BIOS Option ROM Checksum algorithm Assembly language

UEFI Development This column includes this content 503 Subscribe 104 articles Subscribe to our column

(Please keep it-> Author: Luo Bing https://blog.csdn.net/luobing4365)

In the development process of Option ROM of Legacy BIOS , in addition to paying attention to the code structure, you also need to be careful about the size of the code generated file, because the BIOS file does not have much space reserved. I usually control the generated file to around 12K.

Of course, in the later product development, we began to cooperate with Lenovo and Founder, and the Option ROM files were compiled into the BIOS by BIOS engineers, so the above restrictions did not need to be so strict.

1. Checksum Problem

Option ROM files require **the byte checksum** to be 0. However, during the sales of isolation cards, it was found that some BIOS require **the word checksum** to be 0.

The so-called byte checksum is to add up the contents of the file byte by byte, and the sum is 0; while the word checksum is to add up the contents of the file word by word, and the sum is 0.

There is no such tool available, you have to write it yourself.

The question is, how to ensure that a file satisfies both the byte checksum and the word checksum to be zero?

The algorithm I designed is easy to understand. The steps are as follows:

1) First calculate the word checksum, add a word at the end of the file to ensure that its word checksum is zero;
2) Calculate its byte checksum and get a number, such as 0xFE;
3) Add the word 0x00FE to the file, and then add 0xFE words 0xFFFF;
4) This ends.

Step 1 has ensured that the word check is zero, and the added 0xFE times 0xFFFF plus 0x00FE also make the word check zero (0xFFFF can be regarded as -1, which is very easy to think about); in step 2, the byte checksum is already 0xFE, and the added 0x00FE makes the byte checksum 0xFE+0xFE. The added 0xFE times 0xFFFF is exactly 2*0xFE*(-1), making the byte checksum zero.

By adding redundant data in this way, both the byte checksum and the word checksum are zero.

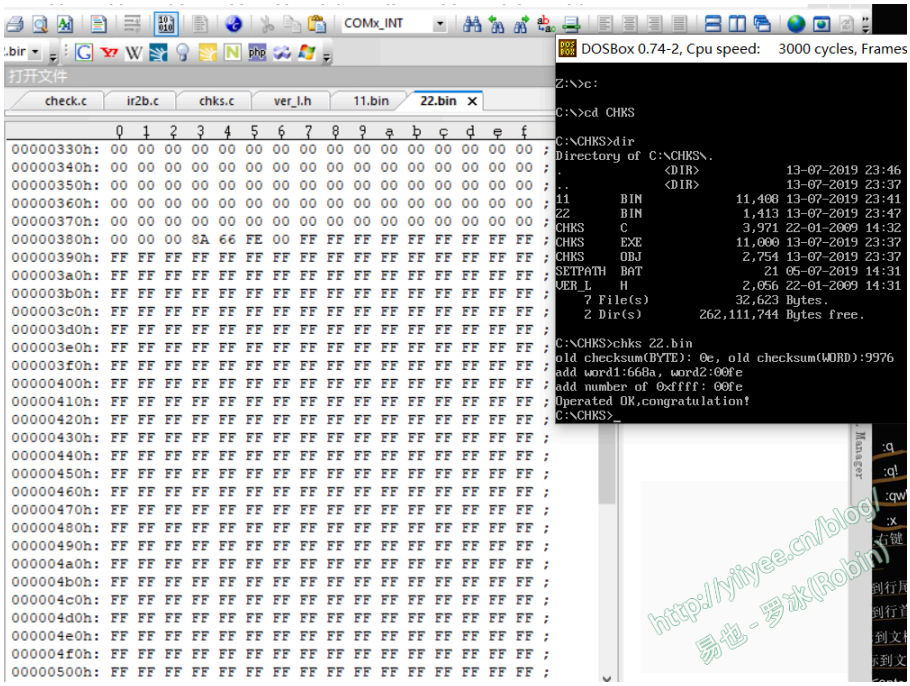

Figure 1 Adjusting the checksum

It is easy to calculate that the maximum number of redundant bytes that need to be added is 257*2=514, which does not have a significant impact on the size of the generated file.

According to the above algorithm, a corresponding tool can be written to convert the compiled executable file (generally compiled into a *.com executable file) into a ROM file that meets the requirements.

2. Further discussion on program structure

When writing a program, you must always remember that it is the BIOS that calls the Option ROM. Therefore, the first thing Oprom needs to do is to save the values of each register and return them after execution to prevent the BIOS from going on strike.

```
.Model Tiny,C
.486
.code
org 0h
start:
        DW 0AA55h                           ;扩展BIOS标志
        DB 40h                              ;本程序大于32K值为80H,否则为40H
       ;保存各寄存器
        cli
        mov bx,ds
        mov ax,9000h
        mov ds,ax
        xor si,si
        mov ax,cs
        mov [si],ax          ;store cs [9000:0]
        add si,2
        mov [si],bx          ;store ds [9000:2]
        add si,2
        mov ax,es            ;store es [9000:4]
        mov [si],ax
        add si,2
        mov ax,ss            ;store ss [9000:6]
        mov [si],ax
        add si,2
        mov ax,sp            ;store sp [9000:8]
        mov [si],ax
        mov ax,8000h         ;ss:sp   8000:mystack
        mov ss,ax
        mov ax,offset mystack
        mov sp,ax
        mov ds,bx
        sti
        call main                       ;入口标志
        cli
        mov ax,9000h
        mov ds,ax
        xor si,si
        ;mov  cs,[si]          ;cs 不回送 错误1：不能用寄存器传送cs
        add si,4
        mov es,[si]
        add si,2
        mov ss,[si]
        add si,2
        mov sp,[si]
        mov si,2
        mov ax,[si]
        mov ds,ax
        sti
        retf                                ;远程返回
;##############################################  数据段开始  ##################
```


Figure 2 Option ROM program structure

I usually save each segment register and general register to 0x900:0 (learning from Linux), and point the stack to 0x8000:mystack. When the main program starts, all data will also be copied to 0x8000:0 (the data segment and the stack segment are in the same segment).

After completing these tasks, the main function will call **the MAIN_MOUDLE** function to start working.

This arrangement has been tested on many motherboards. In theory, these memory spaces will not be used by BIOS. Fortunately, each generation of motherboards and BIOS needs to be modified, and there is no need to be compatible with too many motherboards. If you want to write an Option ROM yourself, it is not recommended to occupy so many segments. It is better to put it in one segment.

The above arrangements are for real mode. If you need to use protected mode code, remember to return to real mode. In later products, I once needed to access more than 1M of memory (mainly to achieve the security functions required by the Security Bureau). As for how to enter protected mode, I will not discuss it in detail.

3 How to debug your code

Option ROM must be embedded in BIOS to run, which makes debugging difficult.

Fortunately, all Option ROM codes call BIOS interrupts and can be run under DOS. Therefore, the code structure can be modified to run under DOS.

```
.DATA
;############################################  数据段开始  ##############
;数据段：显示部分所用数据，硬盘控制所用数据(结构)，提示信息所用字符串
DATABEGIN LABEL BYTE
include SY_DATA.ASM
INCLUDE SY_HZ.ASM
ENDDATA         DW  0AA55H      ;结束标志
;############################################  数据段结束  ##############
;
;############################################  代码段开始  ##############
;+++++++++++++++++++++++++++++++ 主程序开始 +++++++++++++++++++++++++++++++
.CODE
START:
 MOV AX,@DATA
 MOV DS,AX
 MOV ES,AX
 CALL MAIN_MOUDLE
 ;
 MOV  AX,4C00H
 INT 21H
;+++++++++++++++++++++++++++++++ 主程序结束 +++++++++++++++++++++++++++++++
;
;;+++++++++++++++++++++++++++++++ 子程序开始 +++++++++++++++++++++++++++++++
;子程序有六个部分：1 显示部分子程序  2 硬盘控制子程序  3 控制卡的子程序
;          4 读取CMOS信息(为了以后扩展，后加的子程序也放在这个部分)
;          5 为主程序直接调用的界面子程序
;          6 主干模块
;-----------------主干模块 开始-----------------------------------
MAIN_MOUDLE PROC NEAR
  ;检测卡的类型：SY卡、隔离卡、卡不存在
  ;填写card_type
  PORT80H 09900H
;  jmp MAINM_ISSY   ;2011-6-15 13:47:35调试使用的代码，不去检测ORG  luobing
  CALL  GET_CARD_TYPE       ;CF=0 神郁卡存在 CF=1 神郁卡不存在
  JNC MAINM_ISSY            ;SY卡
  ;
  ;lbdebug 2009-6-29 14:11
  JMP MAINM_SWF_NOTEXIST
;  jmp MAINM_ISSY
;IF SUPPORT_ESC
```


Figure 3 Program structure under DOS

Note the MAIN_MOUDLE in the figure, which calls the same function of Option ROM.

In DOS environment, you can debug the code using debug.exe. The instructions for using Debug have been recorded in previous blogs.

Of course, if it does not involve communication with the hardware card, such as debugging image display, it can be debugged in the DOS virtual machine . I used to use Virtual PC, an old virtual machine, to debug DOS programs, and recently started using DOS BOX.

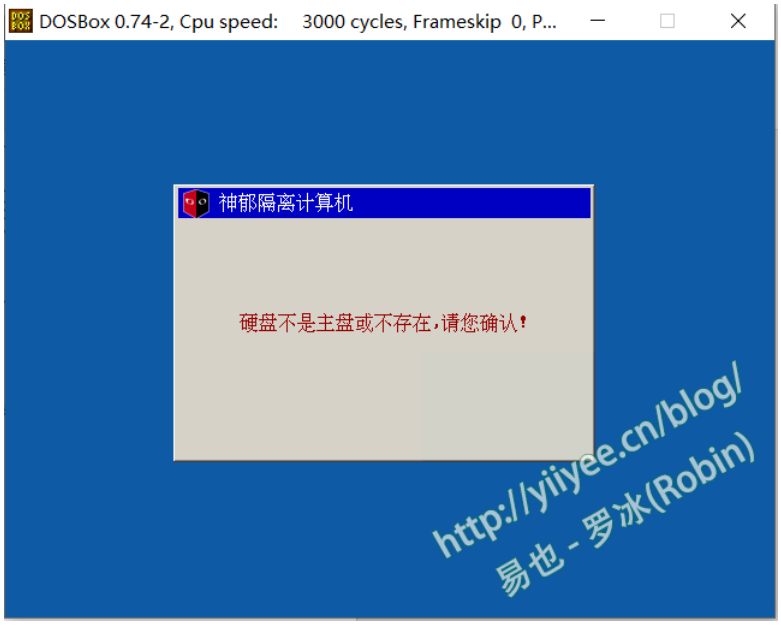

Figure 4 DOS interface

Compile the previous code and run it in DOS box, as shown in Figure 4. It seems that I saw myself sweating and debugging in those days, with mixed feelings.

4Embedded in BIOS

I wrote a tool called c2b.exe to convert .com files to option rom files. Observe the batch file INBIOS.bat provided in this blog, its main commands are as follows:

MASM SYBIOS.ASM
LINK /T SYBIOS.OBJ
C2B SYBIOS.COM SYBIOS.BIN
Cbrom215.exe QDI.BIN /ISA SYBIOS.BIN

Masm and link are assembly compilers provided by Microsoft , c2b.exe is a self-made conversion tool, and cbrom215.exe is a BIOS file processing tool provided by Award. The ISA ROM code can be embedded into the specified BIOS in the command line mode of "/ISA".

The low-level interface software I developed is named SYBIOS. It is compiled and generated as SYBIOS.bin according to the above method. It is embedded into Award's BIOS using Cbrom.exe V2.15. Figure 5 shows the BIOS module embedded with the ISA ROM module. The 15th module is the ISA ROM module.

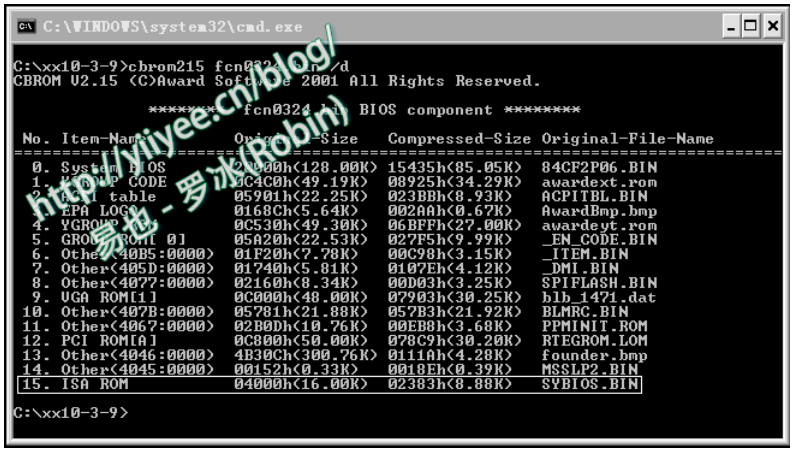

Figure 5 ISA ROM embedded in BIOS

To be fair, it is difficult to write Option ROM under Legacy BIOS and build the framework. All the code is assembly code, which is difficult to maintain. However, the code is not prone to problems, and it is easy to locate the problem by guesswork.

As Legacy BIOS is phased out, these Option ROMs are gradually losing their application scenarios. I share some of the codes used in this article for research purposes only.

Let's regard these two blogs as a farewell to Option ROM under Legacy BIOS. Let's turn our attention back to UEFI. The next blog will describe in detail how to develop Option ROM under UEFI.

Gitee address: https://gitee.com/luobing4365/uefi-explorer
Project code is located in: / X6 Legacy BIOS Oprom
/ Checksum adjustment: Tool code to satisfy both word check and byte check to zero (16-bit compiler compiled, bcc3.1)
/SYBIOS1224: Option ROM example, compiled with Masm6.11