计算机组成原理期末实验报告

（作者 王文茏,软件学院13级教务二班,学号13331256,邮箱 [wwl0816@qq.com](mailto:wwl0816@qq.com)）

摘要：在Win32环境下，使用MVC思想架构，同时应用多文件多模块的软件设计实践，以MASM6.15为主要汇编工具，Sublime Text 3为代码编写工具，综合利用多种汇编命令语句，以能够看懂和较熟练地使用汇编命令为基本目标进行软件设计开发。

关键词：汇编；游戏；1024；

1. **选题背景**

由于大一时自己用C语言实现过这个游戏，对其中的逻辑比较熟悉，认为用汇编实现的话能够使用到较多的汇编语句，而且实现起来也并不复杂。另一个原因是这学期学习过MVC模式设计思想，想在实践中应用一下，而这个游戏可以很清晰地把后台逻辑层和UI显示层区分开来，所以最终就决定了这个选题。

1. **技术路线**

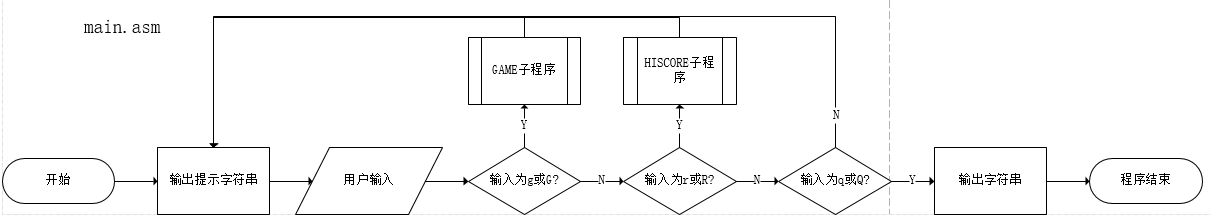
程序分为一个主模块和三个子模块，其中排行榜模块由于时间关系暂未能完全实现，现只能查看最高分。

图一：各模块示意图

程序共分为4个文件：main.asm, game.asm, rank.asm, lib.asm。

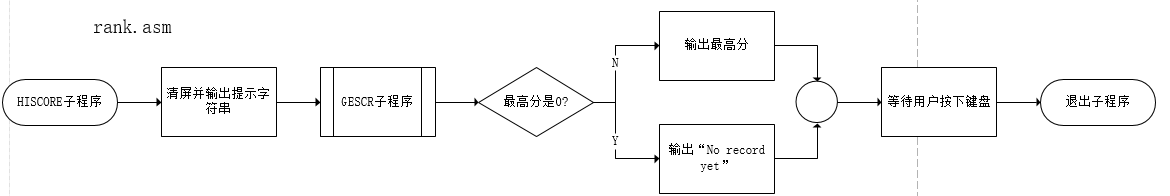
其中lib.asm存储了程序中频繁使用的如输出字符串、清屏、输出回车、得到用户输入等程序段，在其他文件中可以以宏的方式调用这些功能。

main.asm是程序的主模块，程序的主界面、功能选择都在这里实现。



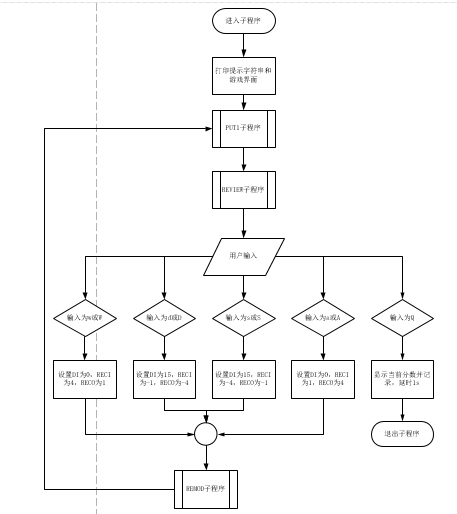
图二：总流程图

rank.asm实现了存储最高分记录的功能，包括了创建、读写、关闭文件等功能。能够更新最高分。



图三：最高分实现流程图

game.asm是程序的核心模块，实现了整个游戏功能。其中又以GAME函数为主函数，其他如REVIE函数则作为GAME调用的子函数。



图四：游戏功能实现流程图

1. **实现过程**

主模块和最高分模块都没有涉及到复杂的算法。主模块主要是输出提示信息和得到用户输入，最高分模块则主要是调用了INT 21H中断中的文件操作功能，在这里就不赘述了，主要说明一下game模块。

game模块分为两大部分，后台数据处理模块和前端界面输出模块。

后台数据部分，根据游戏本身的特点，可以不用存储2、4、8……2048……等大数，而是只存储其相对于2的幂次，比如8存储为3,1024存储为10等。这样就可以用长度为16的DB数据段来存储游戏中各方格对应的数据。程序中用MODE数据段来存储，其与游戏中各方格的对应关系为：

+----------+----------+----------+----------+

+ MODE[0] + MODE[1] + MODE[2] + MODE[3] +

+----------+----------+----------+----------+

+ MODE[4] + MODE[5] + MODE[6] + MODE[7] +

+----------+----------+----------+----------+

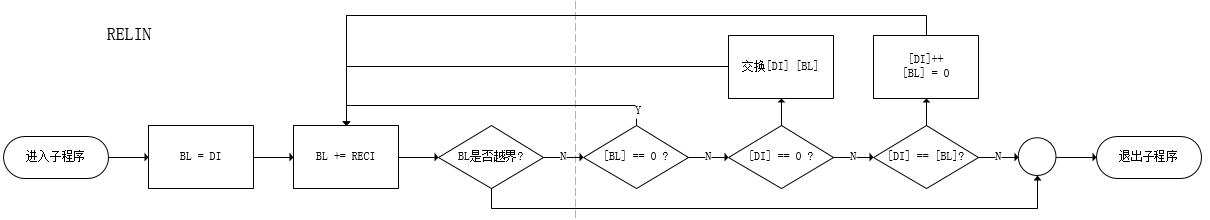
+ MODE[8] + MODE[9] + MODE[10] + MODE[11] +

+----------+----------+----------+----------+

+ MODE[12] + MODE[13] + MODE[14] + MODE[15] +

+----------+----------+----------+----------+

当用户输入一移动指令后，REMOD负责对MODE的更新操作。GAME根据用户输入的方向设置好DI、RECI和RECO（分别代表遍历的起点，内层遍历递增量和外层遍历递增量）后调用REMOD。REMOD会从MODE[DI]，以RECO为每次循环的递增量对REMOD进行遍历，每次对MODE[DI]调用一次RELIN子程序对MODE[DI]进行更新。RELIN则会对MODE[DI]之后的方块进行遍历（以RECO为递增量），将其与MODE[DI]进行比较以确定是否要改变两者的值，直至越界或者不可能更新时（两者的值不相同且都不为0）就退出。其流程图如下：



图五：更新MODE实现流程图

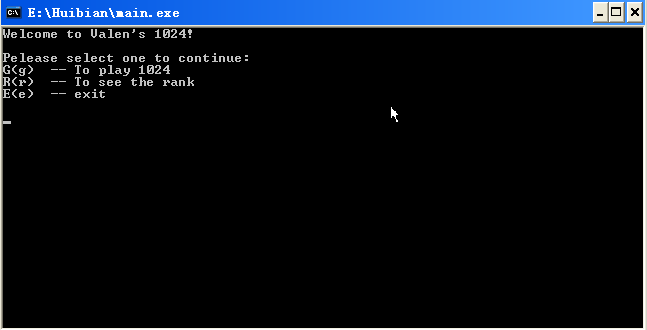
对MODE更新完毕后会调用PUT1将MODE中随机一个方格中的0变为1，实现随机放置一个2的效果。其实现方式是通过INT 1AH读取时钟计数器作为随机数，对16取模得到一个0~15的数值i，之后将MODE[i]之后最近的（超过15则循环至0）一个值为0的方格置为1。

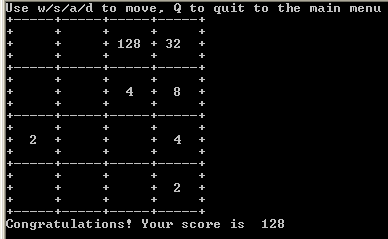
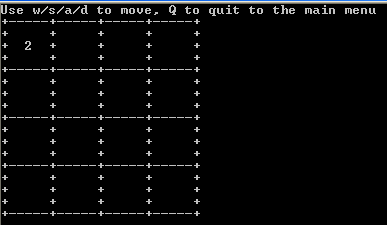
PUT1执行后调用REVEW根据MODE中的值查表得2相对于其的幂次的值，查表得到该格子在屏幕中的位置，将光标定位到相应位置后，将要打印的值输出到屏幕。这样将MODE中的值都打印到屏幕中后，跳转至等待用户输入的程序段，等待用户继续输入下一次指令。

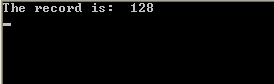
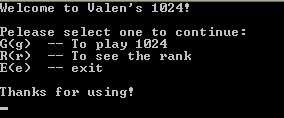
当用户输入的是Q（不能是q）时，程序会输出一段提示字符串和用户游戏的最高分，并调用rank.asm中的SAVSCR函数记录当前分数。之后进入一段空循环，实现延时的功能，最后退出至主页面。

1. **效果评价及体会**

这是程序在32位的Windows XP 系统上运行的效果图：





图六：游戏效果图

总的来说，实现了基本的游戏功能，其中排行榜功能由于时间关系未能实现，打算在暑假的时候如果有时间就把这个项目继续实现下去。

在实验过程中我感触最深的就是原本以为用汇编语言写的代码会很多行，但最终的结果却出乎意料，核心的game.asm文件只有250多行，比大一时用c写的还少。之后分析总结，认为虽然单个汇编语句只能实现一些最基础的功能，而且可用寄存器也有限，不像高级语言可以有各种函数库，变量可以随便定义，这也是一开始我认为会写很多行的原因。但真正一个小软件做下来，发现汇编虽然有以上的特点，但由于其提供的指令已经覆盖了条件分支、条件循环、连续存放数据（类似数组）、堆栈结构、宏程序、子程序调用等基本但频繁使用的程序指令和数据结构，而且其代码由于没有各种括号，单条语句实现功能简单，显得更为紧凑易读。并且汇编可以调用各种系统中断来直接操控显示屏等硬件，相对于高级语言来显得更为方便快捷，比如用INT 10H中断可以方便地直接定位光标到显示屏上任意位置，这如果用高级语言来实现就会显得捉襟见肘。

另一个体会就是按照正确的软件设计实现流程可以大幅提高工作的效率。这次的实验我真正花在编辑代码上的时间不到一天，确定题目后我并没有立即开始写代码，而是先把软件的大的框架，比如要有什么功能，要有多少个文件，每个功能的大概实现方案，各个功能的联系关系考虑清楚，这个过程经历的时间最长，差不多一个星期，但都是利用一些片段时间来想的，总的花费的时间也并不多。大的框架有了之后就开始画每个框架里的流程图，也就是具体的算法，这部分是用了一个下午和晚上在图书馆写写画画做完的。第三步就是具体的代码编辑工作了，由于有了之前两步的工作，这一步进行得十分迅速，第一个可运行的版本只花了一晚上，大概3、4个小时就写出来了，这是我万万没有想到的，要知道大一时我可是花了将近一个礼拜写完的同一个软件啊。

**结语**

这次汇编项目实验虽然只是写了一个简单的小游戏，但在实践的过程中也获益颇深。首先就是发现原先自己对汇编语言的了解还只是停留在书本教材和自己的想当然中，如果不是这次自己亲身经历汇编程序的开发，是不会发现原先自己的见地是如此地不符实际，同时自己的汇编程序开发技能也有所提高，开始的能看懂和能开发的目标基本达成。另一个收获就是终于在真实项目中应用了MVC模式的思想和遵循了基本软件开发流程，之前在课程上学习到的知识终于有了用武之地，不再是纸上谈兵，这也巩固了我对知识的记忆和理解。

总之，通过这次项目实践，不仅提高了我的编程技能，亲身实践了老师课程上讲授的知识，也发现了原先自己见地的狭隘，加深了自己对汇编语言的理解。

**参考文献：**

[1] Muhammad Ali Mazidi. The x86 PC: Assembly Language, Design, and Interfacing, Fifth Edition [M]. 第五版.电子工业出版社, 2008年4月.

[2] 王成耀. 80x86汇编语言程序设计（第二版）[M]. 第2版.人民邮电出版社, 2011年1月.

[3] 王晓虹 毕于深 李 飒.汇编语言 [M]. 第1版.清华大学出版社, 2011年4月.

**附录：程序中用到的汇编指令表**

|  |  |  |  |
| --- | --- | --- | --- |
| **模块化程序设计实践** | **宏汇编** | MACRO | *lib.asm中存储了程序中用到的所有宏片段，其中用到了伪指令 MACRO， ENDM， LOCAL* |
| ENDM |
| LOCAL |
| INCLUDE |
| **子程序调用** | CALL | *game.asm和rank.asm中的函数都是以子程序的方式调用，其中用到了这些伪指令* |
| PROC |
| FAR |
| NEAR |
| RET |
| ENDP |
| INCLUDE |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **一般指令** | **赋值** | MOV | 多处使用 | *最为基本的操作，在整个程序中大量使用* |
| LEA |
| **循环** | LOOP |
| **堆栈** | PUSH | lib.asm:2 | *主要是在子程序和宏中为了不影响寄存器的值而使用* |
| POP |
| **比较** | CMP | game.asm:194 main.asm:25 | *一般是配合跳转指令一起使用* |
| **跳转指令** | JMP | *多用于根据用户输入命令跳转至不同程序段* |
| JA |
| JNA |
| JB |
| JNB |
| JE |
| JNE |
| JNC |
| **计算指令** | ADD | rank.asm:89 | *在rank.asm和game.asm中用于计算和比较，在循环体中也有使用* |
| ADC | rank.asm:90 |
| INC | game.asm:59 |
| DIV | game.asm:80 |
| SHL | game.asm:132 |
| AND | game.asm:78 |
| **位扩展** | CBW | game.asm:130 | *将AL扩展后以便于与SI进行比较和计算* |
| **标志位处理** | CLD | game.asm:182 | *用于设置REP指令中DI的增长方向为正* |
| **字符串处理** | REP | game.asm:184 | *进入game后对MODE进行初始化* |
| STOSB |
| **查表** | XCHG | game.asm:55 | *RELIN模块中交换MODE[DI]和MODE[SI]时使用* |