浙江大学

计算机组成课程设计报告

The Tank War——Only One Shot

中文题目: 坦克大战——最后一颗子弹

英文题目: The Tank War——Only One Shot

学生姓名: 葛浩

学　　号: 3180103494

指导老师: 施青松

专业类别: 计算机科学与技术

所在学院: 计算机科学与技术学院

2020 年 6 月 15 日

Declare 申明

特在此声明，“*Only One Shot*”项目相关的所有工作均为个人独立工作成果，并没有对任何代码进行借鉴。

模块概览

多周期 *MCPU* 模块：	支持 18+ 条指令
***I/O* 外部设备接口模块：**	支持 VGA、PS2 的 IO 交互
***MIPS* 汇编软件模块：**	对 tank war 游戏的实现

目录

1 绪论

1.1 设计背景

本项目为计算机组成综合性课程设计——简易 SOC 或微控制器应用设计 (软硬协同设计) 的成果。项目的出发点是基于计算机组成 Exp03~Exp12 已经搭建完毕的单周期或多周期 CPU，在支持的 18+ 条扩展指令的基础上，搭建一个小型应用程序，需要有基本的 IO 交互与运行逻辑，具备一定的设计意义。

这里采用支持 18+ 指令的多周期 MCPU，并扩展了 VGA 显示和 PS2 键盘交互的功能，设计了一款简单却有意义的坦克大战小游戏——Only One Shot。该游戏设计的基本思想，是利用简单的键盘交互来实现用户对坦克上下左右移动的控制，以及子弹的发射，并通过刷新显示在屏幕上，这一部分检验了 vga 显示和 ps2 的交互功能。游戏将每隔一定时间在随机位置生成障碍物，这一部分将由对计数器 counter 的读取来实现。每轮游戏都将记录当前游戏得分以及多轮游戏的最高分，这一部分将由七段数码管作为显示设备。除此之外，如何验证子弹是否击中障碍物、坦克是否遇到障碍物、障碍物是否出界... 这些都考验了复杂的逻辑设计能力。

值得一提的是，本项目既然叫做"Only One Shot"，顾名思义，它的特点就是：坦克只有一颗可利用的子弹。只有当该子弹击中障碍物或用户主动放弃当前已经击出的子弹时，才能重新使该子弹出膛来射击另一个方向的障碍物。为了增加难度，子弹并不是立马发射出去的，我单独设计了一个子弹从预备发射到出膛的动画过程，也就是说，用户在发射子弹之前，务必要估计自己在被障碍物撞上之前是否能够成功发射子弹。

通过该项目设计，能够较好地检验多周期 CPU 的功能，并扩展了外部 I/O 设备接口，能够锻炼复杂的汇编程序设计能力，进而加深对计算机运行过程的认知。独有的 only one shot 设计理念，也算是在简陋的框架环境下对创新的一点尝试。希望借助本项目，能够帮助自己进一步地理解计算机组成原理，提高汇编程序设计水平。

1.2 国内外现状分析

为了更好地认知计算机组成及其实验课在计算机专业上的重要性，这里查阅了一些文献和资料，并对 MIT、UC-Berkeley、Stanford 大学、CMU 这四所大学相关课程的实验情况进行了简要地总结。这些信息都源自这些大学相关课程的最新课程网站。

通过对多个美国一流大学在相关课程方面教学内容的调查，基本可以总结如下：除 MIT 的课

程内容较偏底层硬件外，其他三所学校相关课程的教学内容和实验内容基本类似，其教学理念和教学思路也非常相似，基本上都是按照“C 语言程序 → 汇编语言程序 → 机器目标代码 → 处理器结构”为主线组织内容，都是站在计算机系统的高度来阐述计算机硬件系统的结构和设计思想。

	MIT	UC-Berkeley	Stanford University	CMU
教材**	未指定(讲义)	C"K&R"+ COD "P&H" + WSC"B&H"	APP"B&O"+ C"K&R"	APP"B&O"+ C"K&R",
模型机	自定义(RISC)	MIPS(RISC)	IA32	IA32
助教人数	12	7	15	未列出
先行课程或要求	了解程序设计语言，具备电子技术基础知识	了解和掌握C语言程序设计技术	了解和掌握C语言程序设计技术	了解和掌握C语言程序设计技术
实验内容	共8个实验，涉及门电路特性、ALU、图灵机、汇编、处理器设计、鼠标中断方式I/O等	共12个实验和4个大作业，涉及到Debug、EC2、MapReduce，MIPS汇编、数据级并行、线程级并行、Cache、虚拟存储管理等，大作业和实验成绩占60%	共8个实验和7个大作业，涉及数据的表示、堆区分配、过程调用和栈的构成及使用、溢出、编译工具和编译优化等，大作业和实验成绩占60%	共7个实验，涉及数据的表示、Cache、缓冲区溢出、过程调用及栈的构成与使用、堆的分配、代理设置等，实验成绩占50%
实验手段	各类模拟器	编程、云计算平台、模拟器	编程	编程
相关后继课程及其实验教学内容	“数字系统设计”和“计算机体系结构及系统”等，涉及CPU设计实验和体系结构方面的模拟实验	“数字系统设计”和“计算机体系结构及工程”等，涉及CPU设计实验和体系结构方面的模拟实验，前者是同时为CS和EE的学生开设的课程	后继课程为“Digital Systems II”，教材为COD"P&H"，主要是流水线CPU和存储系统设计，实验为用DHL设计CPU(CS学生不需要)	CS学生可选ECE开设的课程“Introduction to Computer Architecture”，教材为COD"P&H"，主要是流水线CPU和存储系统设计，实验为用Verilog语言设计CPU

图 1: 美国部分大学计算机专业相关课程实验教学基本情况

由于浙大的计算机组成教材采用国外出版的 “Computer Organization and Design” 一书，因此在课程设计和思想方面，均与国外一流大学保持同步。此外，国外高校也积极采用在自行设计的 CPU 上运行汇编程序 project 的方式来检验学生学习的能力。大多数项目设计基本思想都比较接近，经过调查，虽然同样有坦克大战类型的游戏出现，但赋以 “Only One Shot” 特性的汇编程序是本项目所特有的。

1.3 主要内容和难点

这里将简要介绍本设计要完成的主要内容、功能、技术要求和目的，以及实现的重点难点。

1.3.1 主要内容

1、SOC 基本框架
2、支持 18+ 指令的多周期 MCPU
3、实现 vga 和 ps2 的 I/O 接口

4、实现 Only One Shot 完整功能汇编程序，‘.asm‘文件

1.3.2 设计功能

mips 汇编文件能够在多周期 MCPU 上成功运行，实现只带有一发子弹的坦克大战游戏。游戏规则如下：

1) 每隔一定时间将会在随机位置产生障碍物，坦克可以选择发射子弹或不发射，并且可以上下左右自由移动。

2) 用户只有一发子弹，在之前射出的子弹尚未击中障碍物，而用户又重新按下发射键的时候，之前发射的子弹将会消失，并重新被装载到坦克车身上，开始出膛射击。

3) 如果子弹击中障碍物，则该子弹会与障碍物同时消失，用户将会获得 1 分。

4) 如果障碍物到达并超出边界，则用户将会被惩罚而扣 2 分。

5) 如果障碍物撞上坦克，则游戏结束。

6) 计分板将会实时更新当前用户得分，以及历史最高得分。

注意，子弹出膛需要一定的时间 (这里专门设计了子弹出膛的动画)。因此在障碍物与坦克很接近的情况下，即使用户按下子弹发射键，如果在子弹成功出膛前，坦克就已经与障碍物相撞了，则游戏同样结束。故用户需要额外判定当前时机是否适合发射子弹，这也是本游戏的特色之一。

1.3.3 技术要求与目的

本设计要求掌握基本的 verilog 语法，具备一定的 mips 汇编能力，并能够熟练地构建 ise 工程，进行仿真、调试与物理验证。

1.3.4 重点与难点

本设计虽然思路比较清晰，但在具体实践的过程中，也有不少难点需要克服。

1) 尽管多周期 CPU 支持 18+ 条指令，但实际上未能满足项目开发的实际需求，需要额外扩展指令，或者以软件的形式加以实现。

2) 由于障碍物是随机位置生成的，如何获取随机数，并恰当地转化为符合屏幕大小的坐标是一个难点。

3) 每次生成一个新的障碍物，都需要有指定的地址来保存它的位置信息，否则下一次刷新画面时，该障碍物就会丢失。

4) 每次障碍物由于被击中或超出界面而消失，都需要在指定的地址上抹除该障碍物的位置信息，否则下一次刷新时，该障碍物依旧会出现。

5) 由于障碍物的数量每时每刻都在变化，位置也在时刻更新，因此如何去遍历更新这些数据，在什么地方删去旧的记录，什么地方插入新的记录，也是一个难点。

6) 子弹每次的发射位置都必须是坦克的”弹仓”位置，且需要有一个从坦克尾到坦克头发射的过渡动画。

7) 坦克需要能够随着 ps2 键盘的按下进行上下左右移动和子弹的发射

8) 如何判断子弹是否与障碍物相遇，坦克是否与障碍物相遇，障碍物是否出界，需要利用多少个点的相对位置来判断，也是一个难点。

9) 如何在刷新的时候实时处理坦克、子弹、障碍物在屏幕上的画面。

10) 如何利用七段数码管来统计当前得分，并同时保存并显示历史最高分。

2 设计原理

2.1 设计相关内容

2.1.1 理论要点

1) 多周期 CPU 理论知识，包括控制器和数据通路，取址、译码、执行、回写等多个阶段。

2) SOC 顶层框架搭建的基本知识

3) VGA、PS2 等外部 I/O 接口的扩展知识

3) MIPS 汇编程序设计的理论知识，包括 32 条指令的实现细节，以及如何用现有指令合成伪指令的方式。

4) 软硬件协同设计的基本思想

2.1.2 技术工具

1) 代码编辑器——VSCode

2) soc 框架工程搭建工具——ISE14.7

2) 将 mips 转化为 coe 文件的汇编器 mipsasm

3) 执行 coe 文件的模拟器 ZJUQS-II.SWORD.Emulator.v1.2.3

2.1.3 设计方法

自顶向下设计思想

2.2 设计方案

2.2.1 物体位置信息的存放

这里主要围绕如何存储实时更新变化的障碍物、坦克和子弹的话题给出一个解决方案。

在 mips 编程规范中，.data 可以用于自定义标签和该标签上存放的内容，同样也支持数组的使用。因此这里可以将障碍物、坦克和子弹的位置信息存储在数组里，并在每次刷新屏幕的时候，遍历数组取出这些位置信息并更新。那么问题来了，怎么判别哪个地址段上的数据属于坦克，哪段数据又属于障碍物？为了解决这个问题，还需要额外设计一个 num 变量，用于存放当前障碍物的个数信息。此时如果要删除或插入一条新的记录，则可以根据 num 来遍历

障碍物数组的所有数据，并进行整体的挪动或更新。

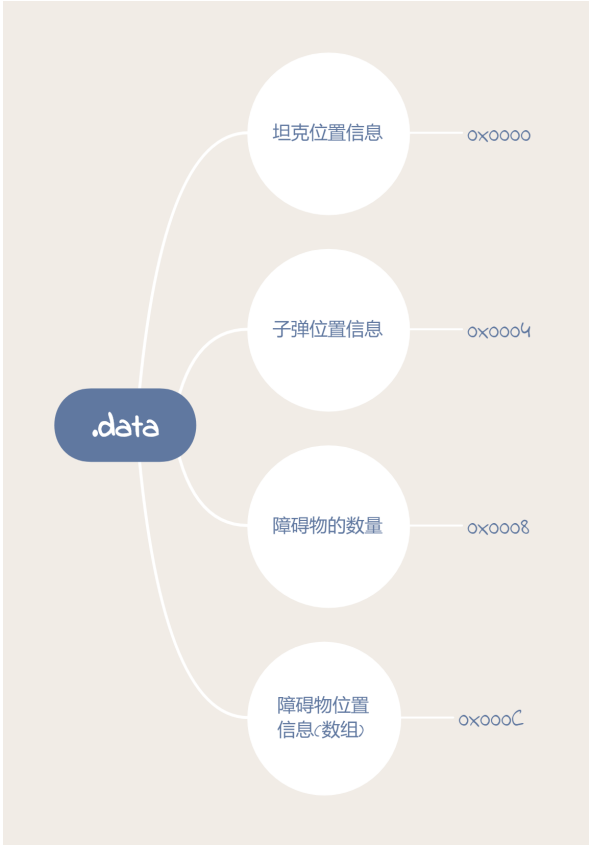

图 2: .data 标签及数据分配示意图

最后还有个问题，当.data 里设计完指定的标签后，我们通常是用 la $rt, label 这条伪指令将 label 所表示的地址值赋值给 $rt 寄存器的，然而我所设计的多周期 CPU 并不支持这样一条伪指令。实际上，深入了解 la 指令之后可以发现，我们完全可以用 addi $rt, $rs, address 指令事先计算出 label 的值，然后再用 lw $rt, $rs 来获取 label 标记地址上的信息。此时，物体位置信息存放的问题就有了一个初步的解决方案。

2.2.2 生成障碍物

程序需要每隔一段时间在随机位置生成障碍物。实际上，我们不可能采取系统的主频来作为生成频率，那会导致障碍物生成过快，从而很快挤满屏幕。

为了解决这个问题,这里用 $t9 寄存器作为障碍物生成的时间间隔变量,作为 flag, 当 $t9=0x20000

时生成新的障碍物，并且将 flag $t9 重新置为 0。这里采用这个数字作为 flag 的计时上限，是因为它是我经过多次参数的调整从而得到的一个比较合适的数字。一开始平均 1s-2s 生成新的障碍物，当障碍物数量多起来之后，需要调用的判断函数随之增加，flag 递增的速度也会随之降低。因此可以获得，障碍物数量少的时候生成的速度快，障碍物数量多的时候生成的速度慢的效果，可以使玩家获得较好的体验感。

2.2.3 随机坐标的获取

随机坐标通过读写 GPIO 端口的 counter 计数器得到，并将其转化为符合 640 的 x 轴范围，480 的 y 轴范围的坐标。再进一步通过绘制函数显示在屏幕上。

2.3 硬件设计

硬件设计部分参照多周期 MCPU 的实验报告，具体内容待返校物理验证后填写。

2.4 系统软件设计

这一部分将详细介绍”Only One Shot” 汇编代码的设计细节，为了方便阅读，部分代码分析与解读直接放在注释里。

2.4.1 模块概览

为了便于理解，我将本项目的函数模块化，并绘制了对应的思维导图，从下图中可以看出各个模块之间的相互调用关系，以及整个程序运行的流程。

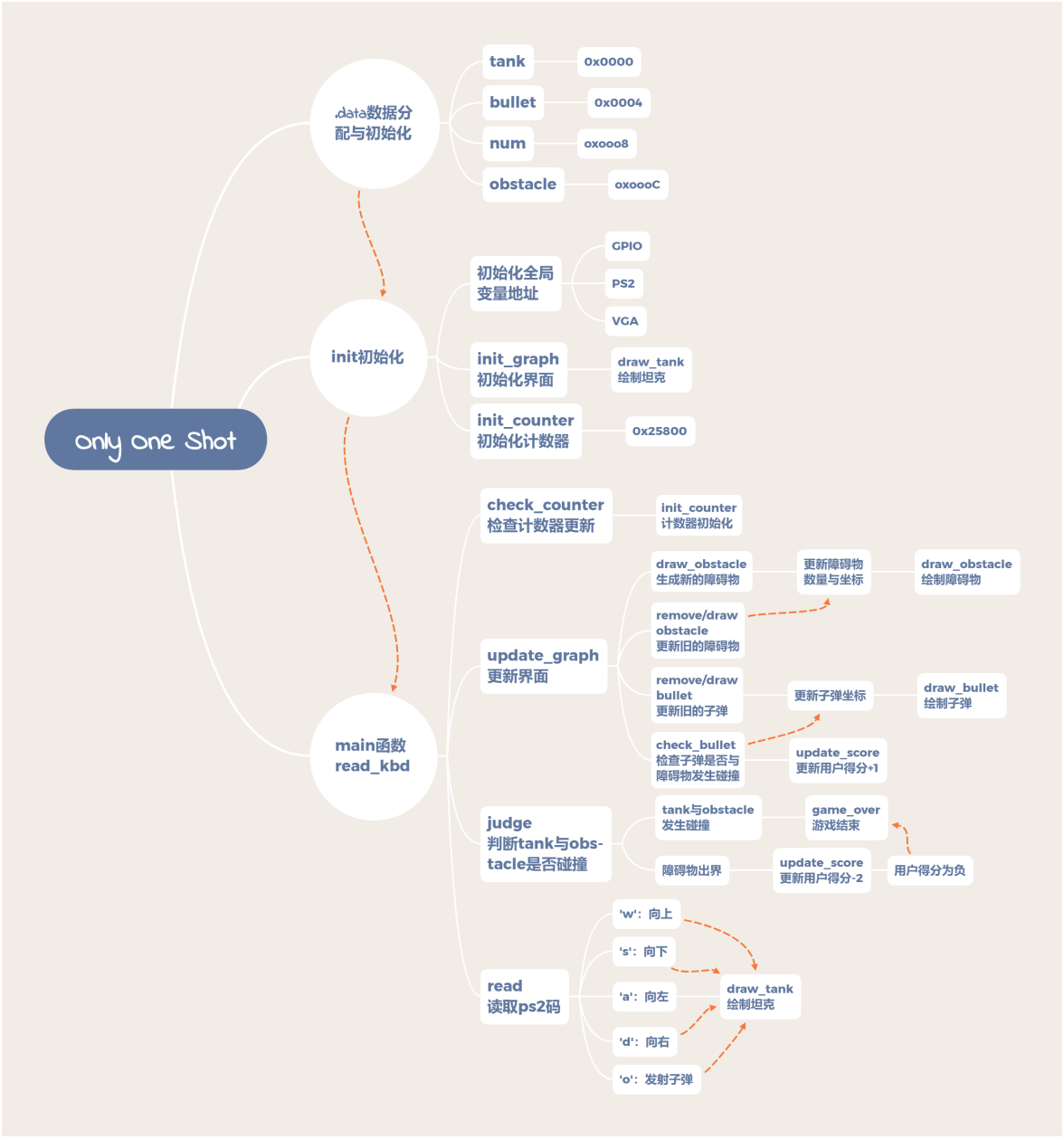

图 3: 模块概览图

2.4.2 .data 空间的数据分配

```
.data 0x00
tank: .word 0x0 # 记录tank的(x,y)坐标，其中前16位是x，后16位是y address=0x0000
bullet: .word 0x0 # 记录bullet的(x,y)坐标，其中前16位是x，后16位是y address=0x0004
num: .word 0x0 # 记录障碍物的个数 address=0x0008
obstacle: .word 0x0 # 记录障碍物的(x,y)坐标，其中前16位是x，后16位是y address=0x000C
```

2.4.3 init 初始化模块

```
.text 0x40
init:
ori $sp, $zero, 0xFFFF # 先给$sp一个地址，避免溢出
lui $k0, 0xFFFF # $k0 = 0xFFFFFF00 GPIO地址
ori $k0, $k0, 0xFF00
addi $k1, $k0, 4 # $k1 = 0xFFFFFF04 Counter地址
# li $s7, 0xFFFFD000
lui $s7, 0xFFFF # $s7 = 0xFFFFD000 ps2地址
ori $s7, $s7, 0xD000
lui $s6, 0x000C
ori $s6, $s6, 0x2000 # $s6 = vram_graph = 0x000C2000 vga地址
add $t8, $zero, $zero # 当前得分初始化为0
add $t9, $zero, $zero # flag初始化为0
jal init_graph # 先初始化界面
jal init_counter # 初始化计数器
```

通过该初始化程序将初始化一些全局变量:

1) $k0=0xFFFFFF00 GPIO 地址

2) $k1=0xFFFFFF04 Counter 地址

3) $s7=0xFFFFD000 ps2 地址

4) $s6=0x000C2000 vga 地址

5) $sp=0x0000FFFF 堆栈指针地址

6) $t0= 当前点地址

7) $s0= 当前点的颜色

8) $t8= 当前得分 (每击中一个障碍物 +1 分，每出界一个障碍物-2 分)

9) $t9= 是否生成新的障碍物的 flag, 当 $t9=0x00020000 时生成新的障碍物

2.4.4 ps2 读取模块

本模块通过一个无限循环来读取 ps2 的键盘输入，并根据输入的有效按键，反映到 vga 的实际显示上。比如，'w','s','a','d' 按键分别对应坦克的上下左右移动，'o' 按键则是对应着子弹的发射功能。

除此之外，基于无限循环 loop 的特性，每次调用该函数时都会先后访问计数器模块、图形显示更新模块、判断出界 or 碰撞模块等。因此该模块也可以被看作是本项目的**主函数**。
该模块主要分为两部分：

1、键盘码读取（包含其他功能函数的调用）

```
read_kbd:
jal check_counter # 检查计数器是否要重新赋值
jal update_graph # 每次读取ps2前先更新一下vga画面，主要是更新障碍物的位置
jal judge # 判断更新完位置的障碍物是否与tank相撞
lui $s5, 0x8000 # $s5 = 0x80000000 $s5最高位为1，用于取出ps2的ready信号
addi $s4, $zero, 0x00F0 # $s4 = 0x000000F0 ，F0是断码的标志，是所有key的倒数第二个断码
lw $t1, 0($s7) # $t1 = {1'ps2_ready, 23'h0, 8'key}
and $t2, $t1, $s5 # 取出$t1最高位的ready信号放到$t2上
beq $t2, $zero, read_kbd # $t2=0表示没有ps2输入，此时跳回去接着读read_kbd
andi $t2, $t1, 0x00FF # 如果ready=1，取出$t1低八位的通码
beq $t2, $s4, read # 如果$t2=0x00F0，则说明读到了倒数第二个断码，此时跳到read来显示键盘码
j read_kbd
```

2、断码识别，跳转按键对应模块

```
read: # 读入最后一个断码，也就是key的标识码
lw $t1, 0($s7) # $t1 = {1'ps2_ready, 23'h0, 8'key}
and $t2, $t1, $s5 # 同理，取出ready信号
beq $t2, $zero, read_kbd # ready=0，则回去重新读
andi $t2, $t1, 0x00FF # 否则，取出低八位的断码(标识码)
addi $s1, $zero, 0x1d # $s1 = "w"
beq $t2, $s1, up # if $t2 == "w", then tank remove up
addi $s1, $zero, 0x1b # $s1 = "s"
beq $t2, $s1, down # if $t2 == "s", then tank remove down
addi $s1, $zero, 0x1c # $s1 = "a"
beq $t2, $s1, left # if $t2 == "a", then tank remove left
addi $s1, $zero, 0x23 # $s1 = "d"
beq $t2, $s1, right # if $t2 == "d", then tank remove right
addi $s1, $zero, 0x44 # $s1 = "o"
beq $t2, $s1, shot # if $t2 == "o", then tank shot ballet
j read_kbd # 跳回重新读取ps2
```

2.4.5 界面初始化模块

该模块由程序初始化运行时调用，构造画面背景、坦克的初始位置。绘制画面的基本思想是，遍历每一行和每一列，并对每一个坐标点进行像素的赋值。

```
# 初始化界面
init_graph:
addi $sp, $sp, -4
sw $ra, 0($sp)
add $t0, $zero, $s6 # $t0为全局变量，是当前点的地址，初始化为第一个点的地址
add $t1, $zero, $zero # $t1表示当前扫描到的y坐标
add $t2, $zero, $zero # $t2表示当前扫描到的x坐标
loop_y_init: # 每一行的遍历
slti $t3, $t1, 480 # 如果$t1=y>=480，则整个屏幕都已经遍历完了，结束扫描
beq $t3, $zero, end_y_init
add $t2, $zero, $zero # $t2=x重新初始化为当前行的第一个点坐标
loop_x_init:
slti $t3, $t2, 640 # 如果$t2=x>=640，则当前行已经遍历完了，切换到下一行
beq $t3, $zero, end_x_init
addi $s0, $zero, 0x00F0 # 给$s0颜色赋值,点为绿色，注意只有前16位才是rgb有效位
sh $s0, 0($t0) # 把$s0的前16位rgb放到当前点的地址上，后16位全0，不作使用
addi $t0, $t0, 2 # offset+=2，每个点占2byte
addi $t2, $t2, 1 # $t2=x++
j loop_x_init # 继续遍历该行的剩余点
end_x_init:
addi $t1, $t1, 1 # $t1=y++
j loop_y_init # 继续遍历下一行
end_y_init:
addi $a0, $zero, 320 # 在初始位置(320,479)绘制tank
addi $a1, $zero, 479 # $a0=x=0x0140，$a1=y=0x01df
sll $s1, $a0, 16 # 将x=$a0放到$s1的高16位，$s1=0x01400000
or $s1, $s1, $a1 # 将y=$a1放到$s1的低16位，$s1=0x014001df
add $s2, $zero, $zero # 获取存放tank坐标的内存地址$s2=0x0
sw $s1, 0($s2) # 将tank坐标存放在0x0的内存地址上
jal draw_tank # 绘制tank
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra # 遍历完成，返回地址
```

2.4.6 计数器更新模块

功能包括：计数器重新初始化 + 检查计数器是否需要重新初始化。

```
# 重新初始化计数器
init_counter:
addi $t1, $zero, 600
sll $t1, $t1, 8 # 600左移8位:0x0258->0x00025800，方便计数，中心点x范围:(20,620)
sw $t1, 0($k1) # 给计数器一个初始值600*4,是障碍物中心点出现的x轴范围*4
```

```
jr $ra

# 检查计数器是否要重新赋值
check_counter:
addi $sp, $sp, -4
sw $ra, 0($sp)
lw $t3, 0($k0) # 读取GPIO
lui $t1, 0x8000
and $t2, $t3, $t1 # 取出GPIO端口最高位counter0_out信号
beq $t2, $zero, continue # 如果计数器还未计满数字，则不用重新给计数器赋值
jal init_counter # 如果计数器已经计完一遍数字，则重新开始计数
continue:
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
```

2.4.7 四边形的绘制和擦除模块

功能：画一个长方形 (只会改变这个长方形范围内的像素值)。
input: ($a0,$a1)= 左上角坐标，($a2,$a3)= 右下角坐标。
input: $s0= 长方形颜色。

```
draw_rectangle:
addi $sp, $sp, -8
sw $t3, 4($sp)
sw $ra, 0($sp)
add $t0, $zero, $s6 # $t0表示当前点的地址，初始化为第一个点的地址
add $t1, $zero, $zero # $t1表示当前扫描到的y坐标
add $t2, $zero, $zero # $t2表示当前扫描到的x坐标
loop_y_rec: # 每一行的遍历
slti $t3, $t1, 480 # 如果$t1=y>=480，则整个屏幕都已经遍历完了，结束扫描
beq $t3, $zero, end_y_rec
add $t2, $zero, $zero # $t2=x重新初始化为当前行的第一个点坐标
loop_x_rec:
slti $t3, $t2, 640 # 如果$t2=x>=640，则当前行已经遍历完了，切换到下一行
beq $t3, $zero, end_x_rec
slt $t3, $t2, $a0 # 如果x的范围不在矩形框内，则直接x++
bne $t3, $zero, jump_x_rec
slt $t3, $a2, $t2
bne $t3, $zero, jump_x_rec
slt $t3, $t1, $a1 # 如果y的范围不在矩形框内，则直接x++(放在这个循环内是为了让offset也得到更新)
bne $t3, $zero, jump_x_rec
```

```
slt $t3, $a3, $t1
bne $t3, $zero, jump_x_rec
sh $s0, 0($t0) # 把$s0的前16位rgb放到当前点的地址上，后16位全0，不作使用
jump_x_rec:
addi $t0, $t0, 2 # offset+=2，每个点占2byte
addi $t2, $t2, 1 # $t2=x++
j loop_x_rec # 继续遍历该行的剩余点
end_x_rec: # 该行遍历结束
addi $t1, $t1, 1 # $t1=y++
j loop_y_rec # 继续遍历下一行
end_y_rec: # 所有行遍历结束
lw $ra, 0($sp)
lw $t3, 4($sp)
addi $sp, $sp, 8
jr $ra
```

注：对擦除函数来说，只会改变这个四边形范围内的像素值，本质上就是把这个四边形重新赋值为背景色，因此函数内容与绘制函数基本一致，这里不再重复列出。

2.4.8 坦克绘制和擦除模块

功能: 绘制 tank，实际上就是调用四边形绘制模块，用几个四边形拼接出坦克的形状。
input: $a0=x, $a1=y, 其中 (x,y) 为 tank 最下方中心点位置。
注意：需要保护变量 $s1, $s2，擦除模块同理。

```
draw_tank:
addi $sp, $sp, -20
sw $s1, 16($sp)
sw $s2, 12($sp)
sw $a0, 8($sp)
sw $a1, 4($sp)
sw $ra, 0($sp)
add $s1, $a1, $zero # 保存input的(x,y)信息
add $s2, $a0, $zero
addi $s0, $zero, 0x0F00 # tank为红色
addi $a0, $s2, -30 # tank body: (x-30,y-20)~(x+30,y)
addi $a1, $s1, -20
addi $a2, $s2, 30
add $a3, $s1, $zero
jal draw_rectangle # 绘制tank body
addi $a0, $s2, -10 # tank head: (x-10,y-40)~(x+10,y-20)
```

```
addi $a1, $s1, -40
addi $a2, $s2, 10
addi $a3, $s1, -20
jal draw_rectangle # 绘制tank head
lw $ra, 0($sp)
lw $a1, 4($sp)
lw $a0, 8($sp)
lw $s2, 12($sp)
lw $s1, 16($sp)
addi $sp, $sp, 20
jr $ra
```

2.4.9 子弹绘制和擦除模块

功能: 绘制 bullet，子弹是一个边长为 10 的正方形。
input: $a0=x, $a1=y, 其中 (x,y) 为 obstacle 最下方中心点位置
注意：需要保护变量 $s1, $s2，擦除模块同理。

```
draw_bullet:
addi $sp, $sp, -20
sw $s1, 16($sp)
sw $s2, 12($sp)
sw $a0, 8($sp)
sw $a1, 4($sp)
sw $ra, 0($sp)
add $s1, $a1, $zero # 保存input的(x,y)信息到($s2,$s1)
add $s2, $a0, $zero
addi $s0, $zero, 0x0F00 # bullet为红色
addi $a0, $s2, -5 # 左上角($s2-5,$s1-10)
addi $a1, $s1, -10
addi $a2, $s2, 5 # 右下角($s2+5,$s1)
addi $a3, $s1, 0
jal draw_rectangle
lw $ra, 0($sp)
lw $a1, 4($sp)
lw $a0, 8($sp)
lw $s2, 12($sp)
lw $s1, 16($sp)
addi $sp, $sp, 20
jr $ra
```

2.4.10 障碍物绘制和擦除模块

功能: 绘制/擦除障碍物，障碍物为边长为 40 的正方形。
input: $a0=x, $a1=y, 其中 (x,y) 为 obstacle 最上方中心点位置。
注意：需要保护变量 $s1, $s2，擦除模块同理。

```
draw_obstacle:
addi $sp, $sp, -20
sw $s1, 16($sp)
sw $s2, 12($sp)
sw $a0, 8($sp)
sw $a1, 4($sp)
sw $ra, 0($sp)
add $s1, $a1, $zero # 保存input的(x,y)信息到($s2,$s1)
add $s2, $a0, $zero
addi $s0, $zero, 0x000F # obstacle为蓝色
addi $a0, $s2, -20 # 左上角($s2-20,$s1)
add $a1, $s1, $zero
addi $a2, $s2, 20 # 右下角($s2+20,$s1+40)
addi $a3, $s1, 40
jal draw_rectangle
lw $ra, 0($sp)
lw $a1, 4($sp)
lw $a0, 8($sp)
lw $s2, 12($sp)
lw $s1, 16($sp)
addi $sp, $sp, 20
jr $ra
```

2.4.11 坐标与地址转换模块

该模块用于空间坐标与实际地址的相互转化，方便根据坐标对对应点的像素值进行读写操作。
功能: 将输入的 x,y 坐标转化为实际地址。
input: (x=$a0,y=$a1)。
output: $v0=address。

```
coordinate_to_address:
addi $sp, $sp, -16
sw $t0, 12($sp)
sw $t3, 8($sp)
sw $t1, 4($sp)
```

```
sw $ra, 0($sp)
add $t0, $zero, $s6 # $t0是当前点的地址，初始化为第一个点的地址
add $t1, $zero, $zero # $t1=y=0
loop_y:
slt $t3, $t1, $a1
beq $t3, $zero, end_y
addi $t0, $t0, 1280 # 640*2=1280 每一行640个点，每个点2byte,即更新$t0为下一行第一个点的位置
addi $t1, $t1, 1 # $t1=y++
j loop_y
end_y:
add $t0, $t0, $a0 # offset_x=2*x
add $t0, $t0, $a0 # $t0已经是(x,y)点的实际地址
add $v0, $t0, $zero
lw $ra, 0($sp)
lw $t1, 4($sp)
lw $t3, 8($sp)
lw $t0, 12($sp)
addi $sp, $sp, 16
jr $ra
```

2.4.12 界面更新模块

该模块为整个程序中**最为重要的模块**，所显示出来的物体移动效果实际上都是通过该模块的坐标修改、重新绘制并刷新界面得到的。该模块主要分为以下几个部分：

1、通过 flag 判断是否生成障碍物

如果 flag!=20，则只更新旧的障碍物，不生成新的障碍物；如果 flag=20，则生成新的障碍物，且 $t9=flag 清零，此时读取 counter 并将之转化为新生成障碍物的坐标。

```
update_graph:
addi $sp, $sp, -4
sw $ra, 0($sp)
addi $t9, $t9, 1 # $t9=flag++
addi $t1, $zero, 20
bne $t9, $t1, update_old # 如果flag!=20，则只更新旧的障碍物，不生成新的障碍物
add $t9, $zero, $zero # 如果flag=20，则生成新的障碍物，且$t9=flag清零
lw $t1, 0($k1) # 范围：0x00000000~0x00025800
srl $t1, $t1, 8 # 右移8bit:0x0000~0x0258<=>(0,600)
addi $t1, $t1, 20 # 获取随机生成的障碍物中心点x坐标(20,620)
...
update_old:
```

```
...
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
```

2、将新生成的 obstacle 插入障碍物数组并绘制

如果生成新的障碍物，则要执行这一过程。注意，新生成的障碍物坐标用一个 word 存储，x 坐标放入寄存器的高 16 位，y 坐标则放入寄存器的低 16 位。插入的时候，新的元素插入数组头，旧的元素往数组尾挪动一个 word。这样做有一个好处是，当最旧的元素出界时，直接从数组尾部移除即可，无需再挪动其他元素在内存的位置。

```
# ------------------将新生成的obstacle插入障碍物数组 begin-----------------------
sll $s1, $t1, 16 # 将($t1,0)压缩到一个寄存器$s1里，x为高16位，y为低16位
addi $s2, $zero, 0x0008 # 获取存放障碍物个数num的内存地址
lw $s3, 0($s2) # 获取当前的障碍物个数$s3=num
addi $s3, $s3, 1 # 障碍物个数num++
sw $s3, 0($s2) # 更新num
addi $s2, $zero, 0x000C # 获取存放障碍物坐标数组头的内存地址$s2
addi $s3, $s3, -1 # $s3=$s3-1
insert_obs: # 寻找存放新增的障碍物(x,y)坐标的内存地址,更新到$s2
beq $s3, $zero, end_insert_obs
lw $s4, 0($s2) # 保存当前扫描到的障碍物坐标到$s4
sw $s1, 0($s2) # 将上一个障碍物的坐标$s1存到当前位置上
add $s1, $zero, $s4 # 更新上一个障碍物的坐标$s1=$s4
addi $s2, $s2, 4
addi $s3, $s3, -1
j insert_obs
end_insert_obs: # 更新最后一个障碍物坐标的位置，插入完毕后，绘制该障碍物
sw $s1, 0($s2) # 将上一个障碍物的坐标$s1存到当前位置上
add $a0, $t1, $zero # ($t1,0)
add $a1, $zero, $zero
addi $s0, $zero, 0x000F # 设置障碍物为蓝色
jal draw_obstacle # 绘制障碍物
# -----------------将新生成的obstacle插入障碍物数组 end---------------------------
```

3、刷新所有已经存在的子弹的位置并绘制

由于障碍物和子弹一旦生成，就会以一定速度往既定方向移动，这个部分就是用于更新子弹的坐标并刷新显示的。需要额外注意的是，在子弹移动的过程中，一旦与某个障碍物发生碰撞，则需要在画面上同时移除子弹和对应的障碍物，以显示出击中的效果。

```
# --------------------------update bullet-----------------------------------
```

```
#update_bullet:
addi $s2, $zero, 0x0004 # 取出bullet坐标地址
lw $s1, 0($s2) # bullet的(x,y)坐标，赋给$s1
beq $s1, $zero, update_obs # 如果bullet坐标为0，则要么是初始状态，要么已经发生碰撞，需要再次按下
    'o'才能触发新的bullet
lui $a0, 0xFFFF # 提取出x坐标赋给$a0
and $a0, $a0, $s1
srl $a0, $a0, 16
ori $a1, $zero, 0xFFFF
and $a1, $a1, $s1 # 提取出y坐标赋给$a1
jal remove_bullet # 移除bullet
addi $a1, $a1, -5 # x坐标不变，更新y坐标$a1=y-5
jal draw_bullet # 重新绘制bullet，以显示移动的效果
sll $s1, $a0, 16
or $s1, $s1, $a1 # 把该障碍物新的坐标重新赋给$s1
addi $a1, $a1, -10 # 获取bullet头部中心点的y坐标
jal check_bullet # 判断是否与障碍物碰撞
beq $v0, $zero, no_bullet_collision
add $s1, $zero, $zero # 如果发生碰撞，则bullet坐标清零
addi $a1, $a1, 10 # 恢复刚刚画的bullet的坐标
jal remove_bullet # 移除刚刚画的bullet
# ----------------------------update score and max_score---------------------
addi $a0, $zero, 1
jal update_score
# ---------------------------------------------------------------------------
no_bullet_collision:
sw $s1, 0($s2) # 将$s1重新存储到指定内存地址
# ---------------------------------------------------------------------------
```

4、刷新所有已经存在的障碍物的位置并绘制

原理与子弹更新部分相同。

```
# ---------------------------update obstacle--------------------------------
update_obs:
addi $s2, $zero, 0x0008 # 获取存放障碍物个数num的内存地址
lw $s3, 0($s2) # 获取当前的障碍物个数$s3=num
addi $s2, $zero, 0x000C # 获取存放障碍物坐标数组头的内存地址$s2
traverse_obs_addr:
beq $s3, $zero, end_traverse
lw $s1, 0($s2) # 遍历每一个obstacle的(x,y)坐标，赋给$s1
lui $a0, 0xFFFF # 提取出x坐标赋给$a0
and $a0, $a0, $s1
srl $a0, $a0, 16
```

```
ori $a1, $zero, 0xFFFF # 注意，addi是有符号位扩展，addi $a1, $zero, 0xFFFF是错误的！应当使用ori
and $a1, $a1, $s1 # 提取出y坐标赋给$a1
jal remove_obstacle # 移除障碍物
addi $a1, $a1, 5 # x坐标不变，更新y坐标$a1=y+5
jal draw_obstacle # 重新绘制障碍物，以显示移动的效果
sll $s1, $a0, 16
or $s1, $s1, $a1 # 把该障碍物新的坐标重新赋给$s1
sw $s1, 0($s2) # 将$s1重新存储到指定内存地址
addi $s2, $s2, 4 # 下一个障碍物的地址
addi $s3, $s3, -1 # num--
j traverse_obs_addr
end_traverse:
# ------------------------------------------------------------------------
```

2.4.13 分数更新模块

该模块用于更新用户已获得的分数，以及历史最高分数。如果当前用户分数为负，则游戏结束。

input: $a0=+1(击中障碍物 +1) or $a0=0(障碍物出界-2)

```
update_score:
addi $sp, $sp, -20
sw $t3, 16($sp)
sw $s1, 12($sp)
sw $s2, 8($sp)
sw $s3, 4($sp)
sw $ra, 0($sp)
beq $a0, $zero, minus_score
addi $t8, $t8, 1
j read_score
minus_score:
slti $t3, $t8, 2 # 如果$t8小于2，则obstacle出界后得分为负，游戏结束
beq $t3, $zero, minus_con
j game_over
minus_con:
addi $t8, $t8, -2
read_score:
lui $s1, 0xFFFF
ori $s1, $s1, 0xFE00 # $t1=七段数码管地址=0xFFFFFE00
lw $s2, 0($s1) # 读取显示的数据，前4个数码管为max_score，后4个数码管为当前score
srl $s2, $s2, 16
slt $s3, $s2, $t8 # 判断是否要更新max score
```

```
beq $s3, $zero, update_r_score
sll $s2, $t8, 16
or $s2, $s2, $t8
j update_all_score
update_r_score:
sll $s2, $s2, 16
or $s2, $s2, $t8
update_all_score:
sw $s2, 0($s1) # 将当前得分更新到七段数码管显示
lw $ra, 0($sp)
lw $s3, 4($sp)
lw $s2, 8($sp)
lw $s1, 12($sp)
lw $t3, 16($sp)
addi $sp, $sp, 20
jr $ra
```

2.4.14 判断坦克是否会与障碍物发生碰撞

这里通过检测坦克上表面点的上方像素点的颜色来判定。

1、判断 tank 是否会与障碍物发生碰撞，如果发生碰撞，则游戏重新开始。

2、判断障碍物是否超出界面，如果超出边界，则清空对应地址上的坐标，且障碍物个数-1。

```
judge:
addi $sp, $sp, -4
sw $ra, 0($sp)
addi $s2, $zero, 0x0000
lw $s1, 0($s2) # 取出tank的坐标
lui $a0, 0xFFFF # 获取tank的x坐标
and $a0, $a0, $s1
srl $a0, $a0, 16
ori $a1, $zero, 0xFFFF
and $a1, $a1, $s1 # 获取tank下边界中心点的y坐标
addi $a1, $a1, -41 # 获取tank上边界上一行中心点的y坐标
jal coordinate_to_address
addi $t1, $zero, 0x000F # 蓝色
lh $s0, 0($v0)
bne $s0, $t1, judge_tank_left
j game_over
judge_tank_left:
addi $a0, $a0, -30
addi $a1, $a1, 20
```

```
jal coordinate_to_address
addi $t1, $zero, 0x000F # 蓝色
lh $s0, 0($v0)
bne $s0, $t1, judge_tank_right
j game_over
judge_tank_right:
addi $a0, $a0, 60
jal coordinate_to_address
addi $t1, $zero, 0x000F # 蓝色
lh $s0, 0($v0)
bne $s0, $t1, judge_obs_traversal
j game_over
judge_obs_traversal:
addi $s2, $zero, 0x0008 # 获取存放障碍物个数num的内存地址
lw $s3, 0($s2) # 获取当前的障碍物个数$s3=num
add $t3, $zero, $s3 # 用$t3保存当前还剩余的障碍物个数，初始化为num
addi $s2, $zero, 0x000C # 获取存放障碍物坐标数组头的内存地址$s2
judge_obs:
beq $s3, $zero, end_judge
lw $s1, 0($s2) # 遍历每一个obstacle的(x,y)坐标，赋给$s1
ori $t1, $zero, 0xFFFF
and $t1, $s1, $t1 # 获取obstacle上边界中心点的y坐标，赋给$t1，
judge_edge:
slti $t2, $t1, 479
bne $t2, $zero, judge_next # 如果还未到达边界，则继续遍历下一个obstacle
sw $zero, 0($s2) # 如果到达边界，则当前扫描到的地址赋值为0
addi $t3, $t3, -1 # 当前剩余的障碍物个数-1
add $a0, $zero, $zero # 障碍物出界，得分-2
jal update_score
judge_next:
addi $s2, $s2, 4 # 下一个障碍物的地址
addi $s3, $s3, -1 # num--
j judge_obs
end_judge:
addi $s2, $zero, 0x0008 # 获取存放障碍物个数num的内存地址
sw $t3, 0($s2) # 更新当前的障碍物个数为$t3
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
```

2.4.15 判断子弹是否会与障碍物发生碰撞

功能: 判断 bullet 是否会与障碍物撞上，这里通过检测子弹上边界的上一行的颜色来判定。
input: $a0=x, $a1=y (默认是子弹上边界中心点坐标)
output: $v0=1: 已经碰撞，$v0=0: 尚未碰撞

```
check_bullet:
addi $sp, $sp, -20
sw $s1, 16($sp)
sw $s2, 12($sp)
sw $a0, 8($sp)
sw $a1, 4($sp)
sw $ra, 0($sp)
addi $a1, $a1, -1 # 取出上一行点的y坐标
add $t6, $zero, $a1 # 保存该y坐标到$t6
jal coordinate_to_address
lh $t1, 0($v0) # 取出上一行点的rgb
#add $gp, $zero, $t1
addi $t2, $zero, 0x000F # 与障碍物的蓝色对比
addi $v0, $zero, 0
bne $t1, $t2, check_end
addi $v0, $zero, 1
addi $s2, $zero, 0x0008 # 获取存放障碍物个数num的内存地址
lw $s3, 0($s2) # 获取当前的障碍物个数$s3=num
add $t3, $zero, $s3 # 用$t3保存当前还剩余的障碍物个数，初始化为num
addi $s2, $zero, 0x000C # 获取存放障碍物坐标数组头的内存地址$s2
add $t5, $zero, $zero # flag=0
check_obs:
beq $s3, $zero, check_obs_end
lw $s1, 0($s2) # 遍历每一个obstacle的(x,y)坐标，赋给$s1
beq $t5, $zero, check_obs_con
addi $t2, $s2, -4 # 如果$t5=flag=1,则将当前坐标上移一个byte
sw $s1, 0($t2)
sw $zero, 0($s2) # 当前地址上的坐标清空
j check_bos_next
check_obs_con:
lui $a0, 0xFFFF # 提取出x坐标赋给$a0
and $a0, $a0, $s1
srl $a0, $a0, 16
ori $a1, $zero, 0xFFFF
and $a1, $s1, $a1 # 提取出y坐标赋给$a1
add $t1, $zero, $a1 # 获取obstacle的y坐标上界$t1
```

```
addi $t2, $t1, 40 # 获取obstacle的y坐标下界$t2
slt $t4, $t6, $t2
beq $t4, $zero, check_bos_next
slt $t4, $t1, $t6
beq $t4, $zero, check_bos_next
addi $t5, $zero, 1
addi $t3, $t3, -1
jal remove_obstacle
check_bos_next:
addi $s2, $s2, 4 # 下一个障碍物的地址
addi $s3, $s3, -1 # num--
j check_obs
check_obs_end:
addi $s2, $zero, 0x0008 # 获取存放障碍物个数num的内存地址
sw $t3, 0($s2) # 更新当前的障碍物个数为$t3
check_end:
lw $ra, 0($sp)
lw $a1, 4($sp)
lw $a0, 8($sp)
lw $s2, 12($sp)
lw $s1, 16($sp)
addi $sp, $sp, 20
jr $ra
```

2.4.16 坦克的上下左右移动

由于上下左右移动的函数大致相同，因此这里只展示向上移动的函数内容。

```
up:
addi $sp, $sp, -20
sw $s1, 16($sp)
sw $s2, 12($sp)
sw $a0, 8($sp)
sw $a1, 4($sp)
sw $ra, 0($sp)
addi $s2, $zero, 0x0000 # 存放tank地址的内存空间
lw $s1, 0($s2) # 取出tank地址
lui $a0, 0xFFFF # 提取出x坐标赋给$a0
and $a0, $a0, $s1
srl $a0, $a0, 16
ori $a1, $zero, 0xFFFF # 提取出y坐标赋给$a1
and $a1, $a1, $s1
```

```
jal remove_tank
addi $a1, $a1, -20 # tank坐标上移20
jal draw_tank
sll $s1, $a0, 16
or $s1, $s1, $a1
sw $s1, 0($s2) # 把新的tank坐标存放到指定内存地址
lw $ra, 0($sp)
lw $a1, 4($sp)
lw $a0, 8($sp)
lw $s2, 12($sp)
lw $s1, 16($sp)
addi $sp, $sp, 20
jr $ra
```

2.4.17 子弹发射

用户按下按键'o'，便可触发该函数，子弹发射的初始位置为坦克的尾部，在发射出去之前，会有一个出膛的动画演示过程。

```
shot:
addi $sp, $sp, -20
sw $s1, 16($sp)
sw $s2, 12($sp)
sw $a0, 8($sp)
sw $a1, 4($sp)
sw $ra, 0($sp)
addi $s2, $zero, 0x0004 # 获取存放bullet坐标的内存地址
lw $s1, 0($s2) # 取出bullet坐标
lui $a0, 0xFFFF # 提取出x坐标赋给$a0
and $a0, $a0, $s1
srl $a0, $a0, 16
ori $a1, $zero, 0xFFFF # 提取出y坐标赋给$a1
and $a1, $a1, $s1
jal remove_bullet # 删去原先已经发射的bullet
addi $s2, $zero, 0x0000 # 获取存放tank坐标的内存地址
lw $s1, 0($s2) # 取出tank坐标
lui $a0, 0xFFFF # 提取出x坐标赋给$a0
and $a0, $a0, $s1
srl $a0, $a0, 16
ori $a1, $zero, 0xFFFF # 提取出y坐标赋给$a1
and $a1, $a1, $s1
#addi $a1, $a1, -40
```

```
sll $s1, $a0, 16 # 重新合成bullet的坐标放到$s1里
or $s1, $s1, $a1
addi $s2, $zero, 0x0004 # 获取存放bullet坐标的内存地址
sw $s1, 0($s2) # 将新的bullet坐标存放到指定内存地址
jal draw_bullet
lw $ra, 0($sp)
lw $a1, 4($sp)
lw $a0, 8($sp)
lw $s2, 12($sp)
lw $s1, 16($sp)
addi $sp, $sp, 20
jr $ra
```

2.4.18 游戏结束

清除.data 区域的数据，并清除当前用户得分，重新跳转到初始化 init 模块

```
game_over:
lui $t1, 0xFFFF
ori $s1, $t1, 0xFE00 # $s1=七段数码管地址
lw $s2, 0($s1) # 读取显示的数据，前4个数码管为max_score，后4个数码管为当前score
and $s2, $s2, $t1 # 只保留最高分，最低分清零
sw $s2, 0($s1) # 更新score，开启下一轮
add $s2, $zero, $zero
sw $zero, 0($s2) # tank坐标清零
addi $s2, $s2, 4
sw $zero, 0($s2) # bullet坐标清零
addi $s2, $s2, 4
lw $s1, 0($s2)
game_clear: # obstacle清零
beq $s1, $zero, clear_end
sw $zero, 0($s2)
addi $s2, $s2, 4
lw $s1, 0($s2)
j game_clear
clear_end:
j init
```

3 设计实现

注：硬件部分，返校后验证。

3.1 实现方法

3.2 实现过程

3.3 仿真与调试

4 系统测试验证与结果分析

本节将给出程序在模拟器上的运行结果与分析。

4.1 初始化界面

为了简约化，这里将背景色统一设置为绿色，障碍物颜色设置为蓝色，坦克和子弹的颜色则为红色。下图展示了初始化界面，包括一辆红色的坦克和一个刚产生的障碍物。

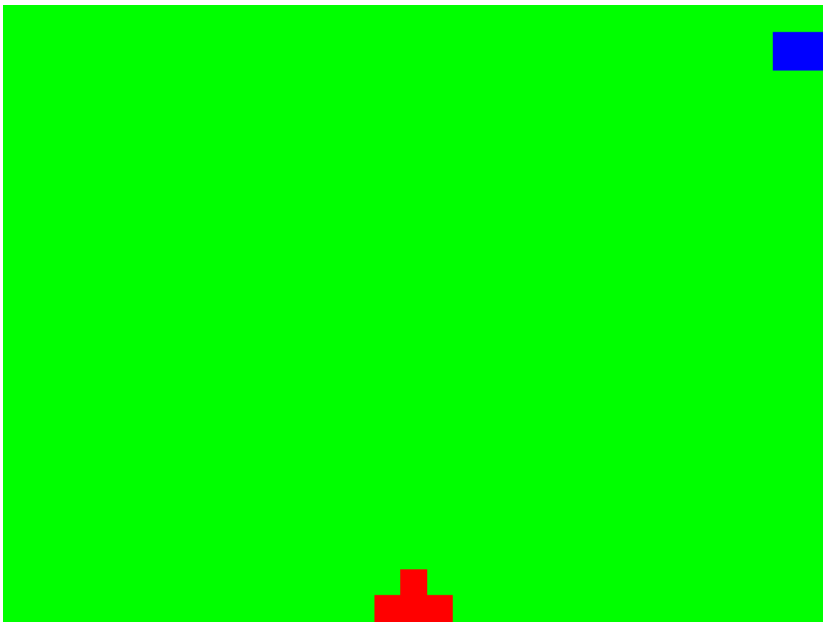

图 4: 初始化界面

4.2 障碍物随机增加和坦克移动

这里随着程序继续运行，障碍物将会随机产生，并不断往下坠落。坦克可以上下左右自由移动，并选择时机发生子弹。下图展示了运行一段时间后的界面，此时障碍物的数量增加，坦克进行了一定的位移。

4.2.1 坦克移动

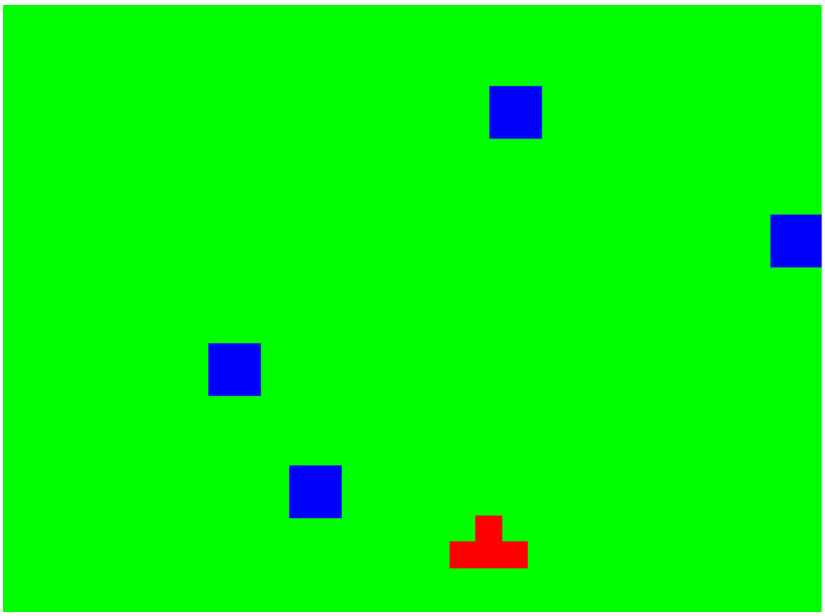

图 5: 一段时间后的界面

4.2.2 障碍物出界，游戏结束

由于出界时，尚未击中任何障碍物，因此得分为负，游戏结束。

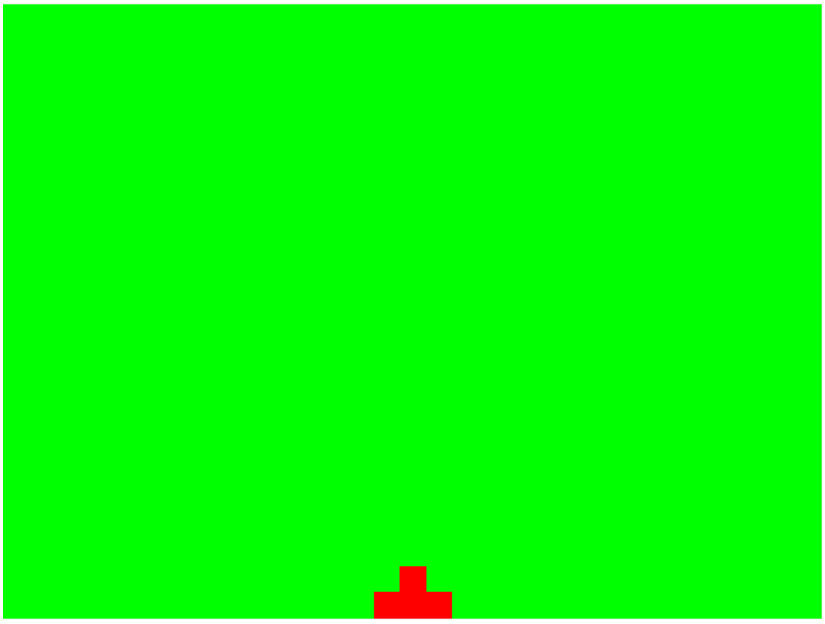

图 6: 重新初始化界面

4.3 坦克发射子弹的过程

为了避免障碍物坠落到边界以下，坦克需要发射子弹来消灭障碍物。下面的几张图演示了子弹发射前后的过程。

4.3.1 子弹在坦克内上膛的过程

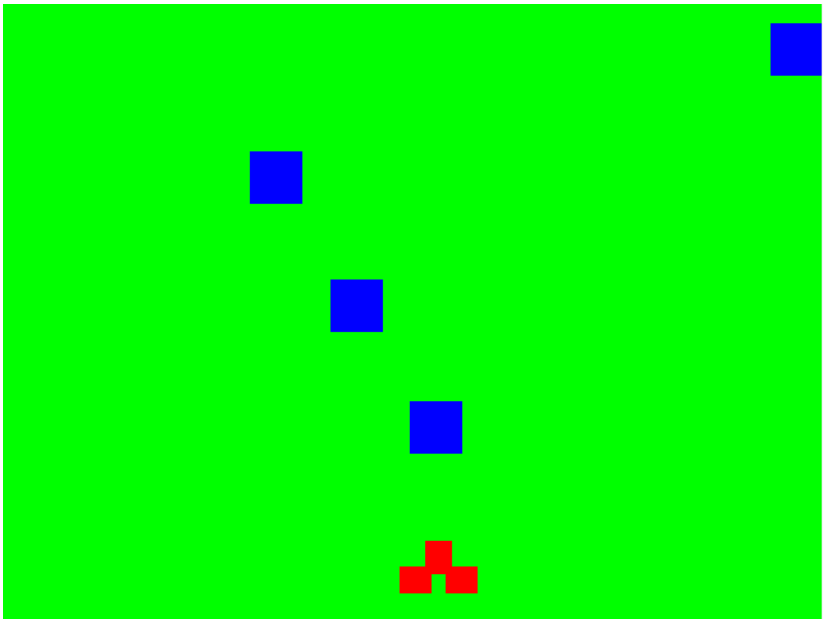

图 7: 子弹上膛

4.3.2 子弹发射之后的状态

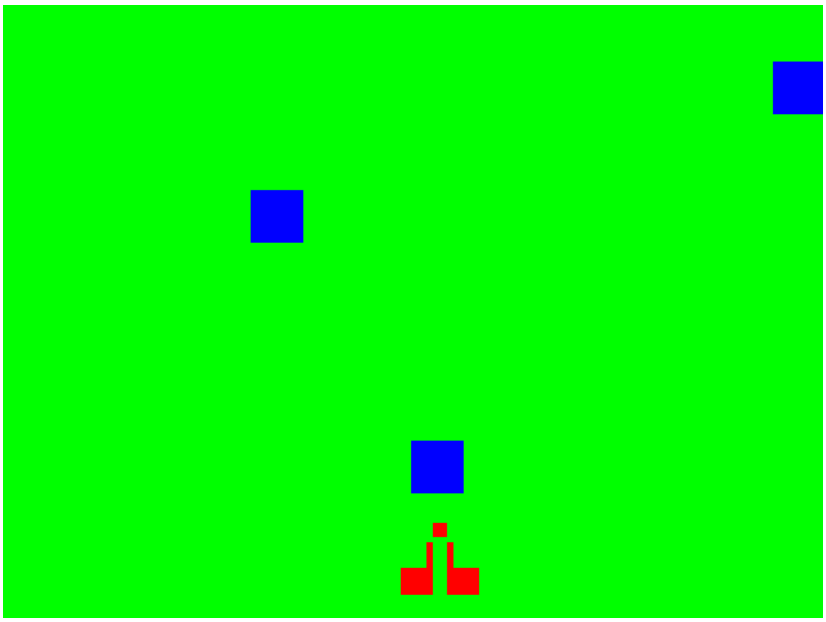

图 8: 子弹发射后

4.3.3 子弹即将击中障碍物

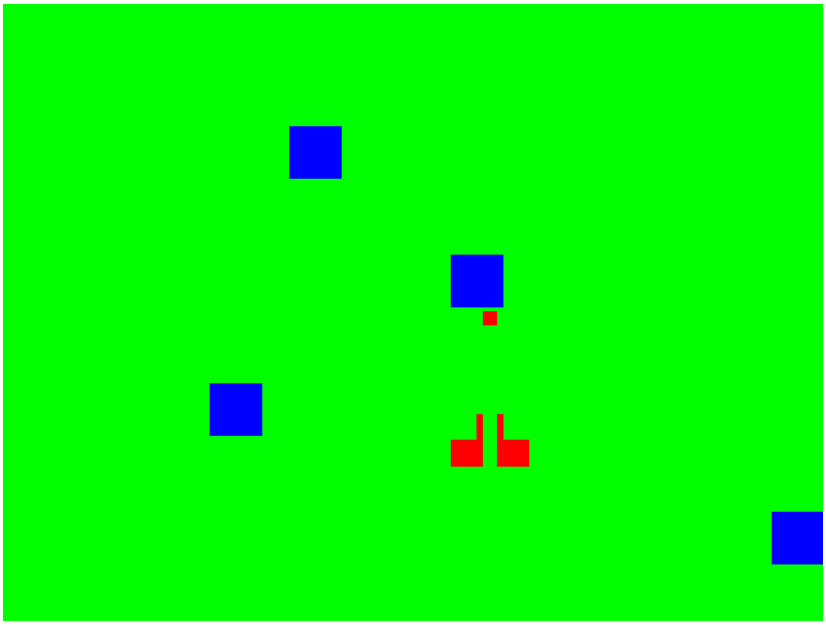

图 9: 即将击中障碍物

4.3.4 击中障碍物后

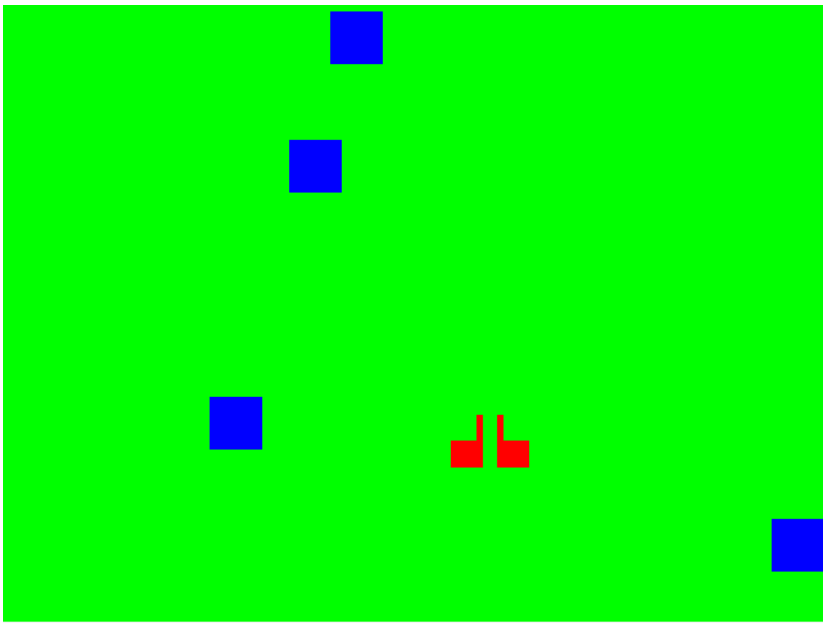

图 10: 击中障碍物后, 子弹与障碍物同时消失

4.3.5 坦克继续移动

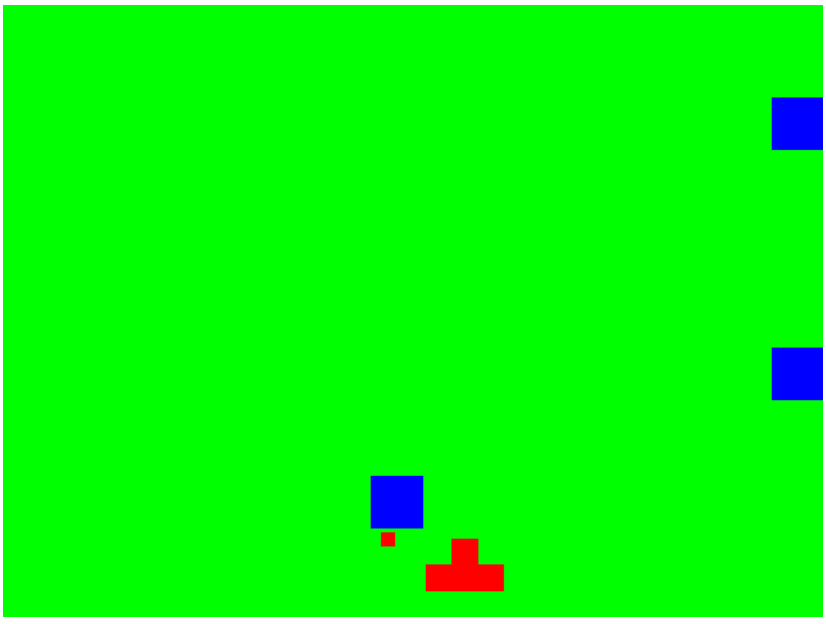

图 11: 发射子弹后, 坦克继续移动

4.4 七段数码管显示得分情况

七段数码管的前四个显示历史最高分，后四个显示当前得分。当坦克击中障碍物时加 1 分，当障碍物超出下边界时扣 2 分。当用户分数为负，或坦克与障碍物相撞时，游戏结束。下图中，当前得分为 2 分，历史最高分为 7 分。

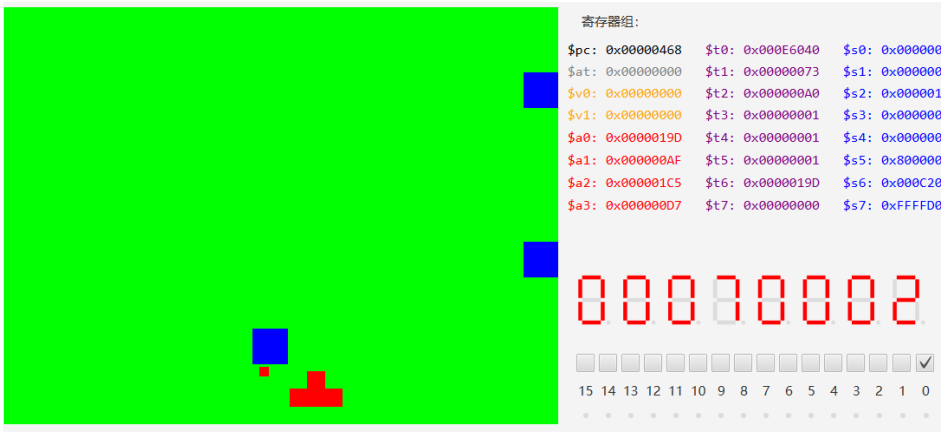

图 12: 七段数码管显示得分情况

5 结论与展望

本次实验由于疫情影响，只能利用模拟器进行软件方面的验证，而硬件方面的验证只能等到 8 月份返校之后进行。在软件部分，主要涉及到 mips 汇编程序设计的编写，由于受到可用指令数量的限制，以及汇编代码的不易读性，因此具体的实现过程中也遇到过不少困难。其中遇到过最多的问题，就是变量和堆栈所造成的。当不同函数之间相互跳转的次数多了以后，很多情况下这些函数又会用到相同的寄存器作为临时变量，因此特别需要注意将变量的值保存到堆栈里，并当函数返回时恢复变量的值。除此之外，如何设计程序的整体框架；如何存放坦克、障碍物、子弹的数据并实时更新与绘制；如何判断子弹击中的是哪个障碍物；如何判断坦克是否与障碍物相撞... 一系列的问题都需要花费大量的时间去分析和处理。

这次汇编大程锻炼了我的细心和耐心，有了第一次汇编 project 作为基础，这次的大程设计起来也没有非常的困难，写了一个多星期就把软件部分给写完了，在这一个多星期里，绝大多数时间也还是花在 debug 上。除此之外，尽管程序在模拟器上成功运行，但是否能够在硬件上调试成功目前尚未知晓。还是期待能早日返校做物理验证吧。