6 团队任务

6.1 选题与分工

本次组原团队任务基于课程设计中支持单级中断的单周期硬布线 RISC-V CPU，实现 Logisim 版复古像素风飞机大战，设计灵感来源于经典游戏。在游戏过程中，玩家操作飞机躲避来自未知势力的子弹群，持续前进，直到生命的最后一刻。未知势力的进攻将随着时间推移越发凶猛，发射更加密集的子弹群阻碍玩家向前探索。

操作界面如图 6.1 所示。

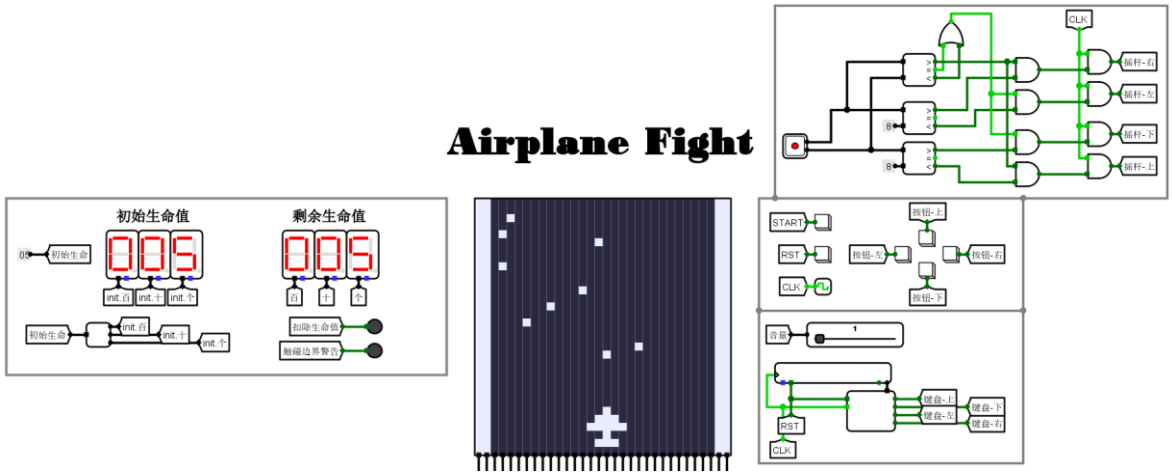

图 6.1 飞机大战操作界面

团队任务分工如表 6.1 所示。

表 6.1 团队任务分工（根据姓名首字母排序）

团队成员	实现内容
曹瑾瑾	硬件实现，包括初版子弹 CPU 与控制台等
董羽栗	软硬件实现，包括终版 CPU 与终版汇编等
龙天云	硬件实现，包括初版飞机 CPU 与控制台等
牛心怡	软件实现，包括初版子弹汇编与音频转换等

6.2 游戏设计与实现

6.2.1 总体设计

飞机大战主要由控制台和 CPU 模块组成。交互模块负责将玩家的输入转化为中断信号传入 CPU，并响应 CPU 传出的各种输出信号（如 LED 点阵、边界警告）。CPU 电

路负责游戏逻辑的控制，输出控制 LED 点阵的信号，并响应交互模块传入的信号。

我个人负责以下工作：

①设计并编写飞机大战的 RISC-V 汇编代码。

②实现若干系统功能调用，用于发出控制游戏交互的信号。

③修改了访存数据通路，实现了对部分内存区域的并发操作。

我负责的任务全部在 CPU 模块之中，因此我重点介绍 CPU 模块的实现。对于玩家来说，飞机大战主要体现为两个模块：飞机模块和子弹模块。在实现过程中，还要考虑到两个模块之间通过碰撞检测逻辑相互关联。

6.2.2 游戏显示驱动设计

作为一款交互式游戏，显示器是必不可少的。我门使用 Logisim 内置的 32×32bit LED 作为图像输出元件，LED 显示图像时需要持续的输入，因为我们设计了一组 LED 缓存寄存器堆作为 LED 的信号输入。考虑到 LED 的大小，缓存寄存器堆的应由 32 个 32 位寄存器组成。需要刷新屏幕时，只需要修改缓存寄存器堆即可。

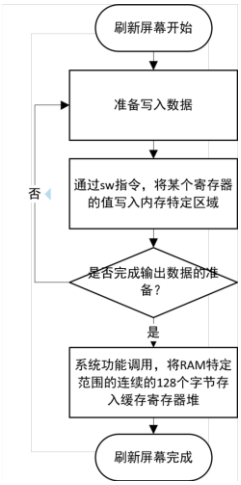

图 6.2 显示模块流程图

下面介绍如何修改缓存寄存器堆。我设计了一个地图寄存器堆，其大小与输出缓存寄存器堆相同，都是 32 个 32 位寄存器。逐个连接地图寄存器的输出到缓存寄存器的输入，这样当缓存寄存器的使能端为 1 时，地图寄存器的值放入对应的缓存寄存器，LED 屏幕完成刷新。

为了方便控制地图寄存器堆，我们将地图寄存器堆设计成了一个 32×4B 的 RAM，并修改访存指令数据通路，通过访存指令访问这个 RAM。于是得到地图显示逻辑如图 6.2 所示。

飞机模块和子弹模块在显示上时两个相对独立的模块。两个模块可以单独设置各自的地图寄存器堆和缓存寄存器堆。两个缓存寄存器堆的输出进行或运算就是最终的 CPU 输出。为了方便起见，我们将飞机模块对应的寄存器堆称为飞机寄存器堆，子弹模块对应的寄存器堆称为子弹寄存器堆。

6.2.3 游戏软件设计

1) 开始游戏模块设计

首先确设置一个标志寄存器标识游戏是否开始，寄存器初始值为 0。完成内存初始化后，开始轮询标志寄存器的值是否不为零。同时编写开始游戏中断服务程序，响应中断后将标志寄存器置 1，开始游戏。

2) 飞机模块设计

首先确定飞机的像素画如图 6.3 所示。需要 2 个寄存器保存飞机的 X、Y 坐标，5 个寄存器保存飞机的像素画。

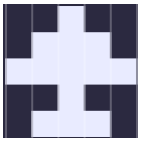

图 6.3 飞机像素画

飞机可以按上下左右四个方向移动，所以要编写四个中断服务程序。每个中断服务程序首先判断飞机是否会飞出边界，如果会飞出边界，则使用系统功能调用发出警报。如果不会飞出边界，则修改 X 或 Y 坐标。并根据 X、Y 坐标清除原来的飞机图案，绘制新的飞机图案，并输出到屏幕。

3) 子弹模块设计

游戏过程中，子弹会从屏幕上方生成，并按一定速率向下移动直到溢出屏幕。子弹生成的位置可以在游戏开始前写入内存，游戏进行过程中从内存中读出。保存子弹位置的数据结构如下：

```
struct Map{
    int col_number;   // col_number=子弹生成的列号* 4 ,取值范围是[8,116]
    int is_golden;    //是否有子弹,为 0 表示没有子弹,为 1 表示有子弹
}map[256];            //map 的初始地址为 512, 结束地址 2560
```

游戏进行过程中从 map 数组中读出子弹的位置并写入子弹寄存器堆。

```
for(int i = 0;i<256;i++){
    if(map[i].is_golden){       //如果有子弹,
        在(col_number/4)那列加一个子弹;
    }
    所有子弹向下移动一格;
}
```

所有子弹的位置都是按行保存在子弹寄存器堆之中的。所以如果需要把整个屏幕向下移动，32 次读内存、位移、写内存操作，时间开销太大。于是设计了一个子弹移动系统功能调用，整个系统功能调用的功能是：将子弹寄存器堆所有寄存器左移一位。这样，所有子弹向下移动一格的开销大大减少。

需要指出的是，虽然 RISC-V 基本指令集中没有随机数生成指令，但是可以用其他编程语言生成 map 数组的初始化代码，并随机化部分参数。可以通过控制参数以实现游戏难度的控制和改变。

4) 碰撞检测逻辑设计

当飞机和子弹碰撞时，子弹会消失，并设置扣除生命系统功能调用去通知交互电路。可以在子弹每次向前移动后，检测飞机和子弹是否碰撞。为了避免碰撞检测的过程被其他中断打断，需要实现把碰撞检测逻辑也设计成一个中断服务程序。但是单级中断 CPU 并没有实现软件开关中断，所以我设置了一个碰撞检测系统功能调用，并将这个碰撞检测系统调用作为中断源，去模拟内部中断。

6.2.4 硬件实现

1) 访存数据通路修改

飞机模块和子弹模块在显示上是两个相对独立的模块。方便起见，将内存地址在 0~127 的区域修改为飞机寄存器堆，128~255 的区域修改为为子弹寄存器堆。子弹寄存器堆和飞机寄存器堆的实现如图 6.4 所示。子弹和飞机的缓存寄存器相或即得到 LED 输出点阵。画面叠加逻辑如图 6.5 所示。

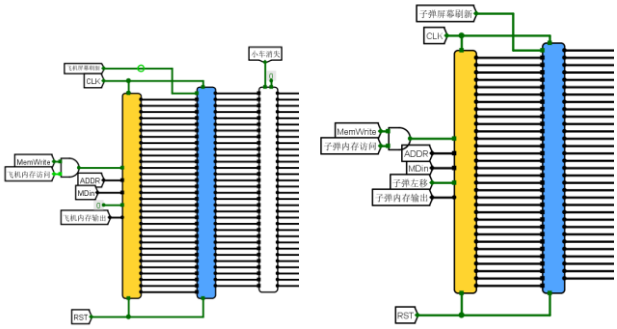

图 6.4 子弹寄存器堆和飞机寄存器堆

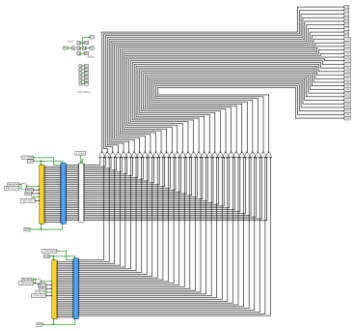

图 6.5 画面叠加逻辑

实现过程中，这个两个地图寄存器堆代替了地址 0~255 范围的 RAM。也就是说，访存指令访问地址 addr<128 时，访问飞机寄存器堆；访问地址 128<=addr<256 时，访问飞机寄存器堆；访问地址 addr>=256 时，使用原来访存的数据通路。

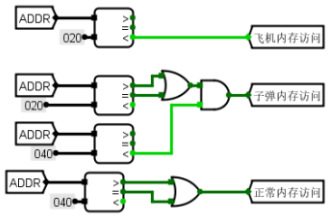

图 6.6 访存判断逻辑

2) 系统功能调用实现

CPU 使用到的系统功能调用汇总如表所示。系统功能调用的判断逻辑如图图 6.7 所示。

表 6.2 系统功能调用

#	系统调用名称	a7	功能
1	子弹显示	34	调用时，刷新子弹的缓存寄存器堆
2	飞机显示	35	调用时，刷新飞机的缓存寄存器堆

#	系统调用名称	a7	功能
3	扣除生命	36	当飞机撞上子弹时调用，用于提示玩家扣除生命
4	碰撞检查	37	用于模拟内部中断，实现碰撞检查过程关中断
5	子弹位移	90	用来将子弹全体向下位移一格
6	边界警报	100	当飞机触碰游戏边界时调用，作为信号提示玩家

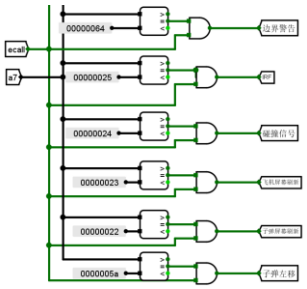

图 6.7 系统功能调用

为了实现子弹位移系统功能调用，需要修改子弹寄存器堆，为子弹寄存器堆增加同步位移的功能。子弹寄存器堆的实现如图 6.8 所示。

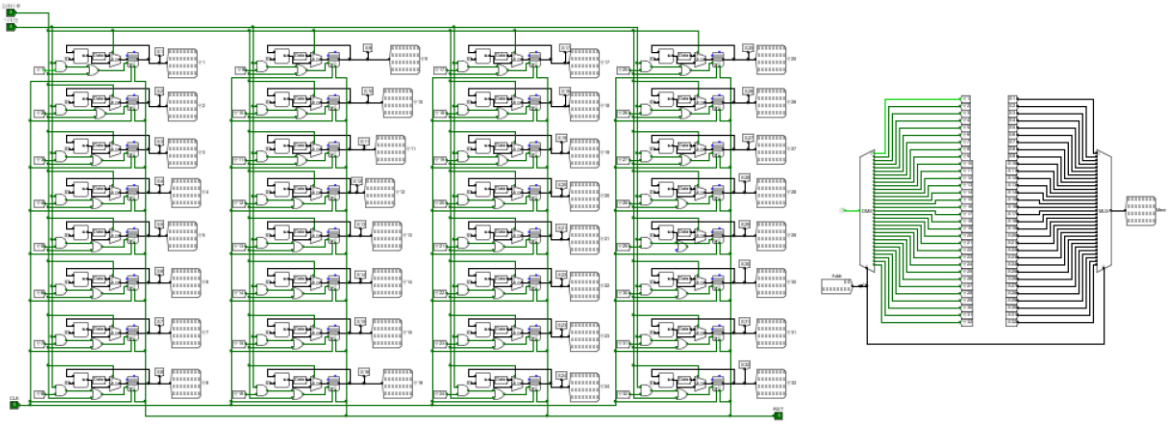

图 6.8 子弹寄存器堆