

## 一：虚拟机设计总述

为了模拟 64 位虚拟机，自定义指令集，用函数的方式来实现 CPU 等功能，并且做出图形界面，不仅便于用户操作，还易于展示虚拟机执行过程。并且发布至 Windows PC, Android, WEB 三大平台，以期适应广大用户群体。

本工程最大特点就是界面实现力求华丽，精心设计 UI、图片背景、音乐音效等元素，在设置中用户可以自主调节指令加入速度、指令运行速度、对比度、饱和度、亮度、声音大小等，并且点击设置里的帮助可直接下载本文档作为参考，注重用户交互体验，使新用户可快速上手。

## 二：虚拟机开发简介

开发者： 翟泽鹏

开发工具：unity 2017.3.1f1 + vs2015

开发环境：Windows 10

打开工程：在 unity 官网 <https://unity3d.com/cn> 下载 unity，新版 unity 也可打开旧版项目。

自己编写的 C#源码在 VM\_SourceCode/VM//Assets/Scripts 目录下。

各平台下载网址：

Windows：

[https://github.com/zzp-seeker/vmPC/raw/master/VM\\_zzpseeker\\_SetUp.exe](https://github.com/zzp-seeker/vmPC/raw/master/VM_zzpseeker_SetUp.exe)

安卓：

[https://github.com/zzp-seeker/VM\\_android/raw/master/VM\\_zzpseeker.apk](https://github.com/zzp-seeker/VM_android/raw/master/VM_zzpseeker.apk)

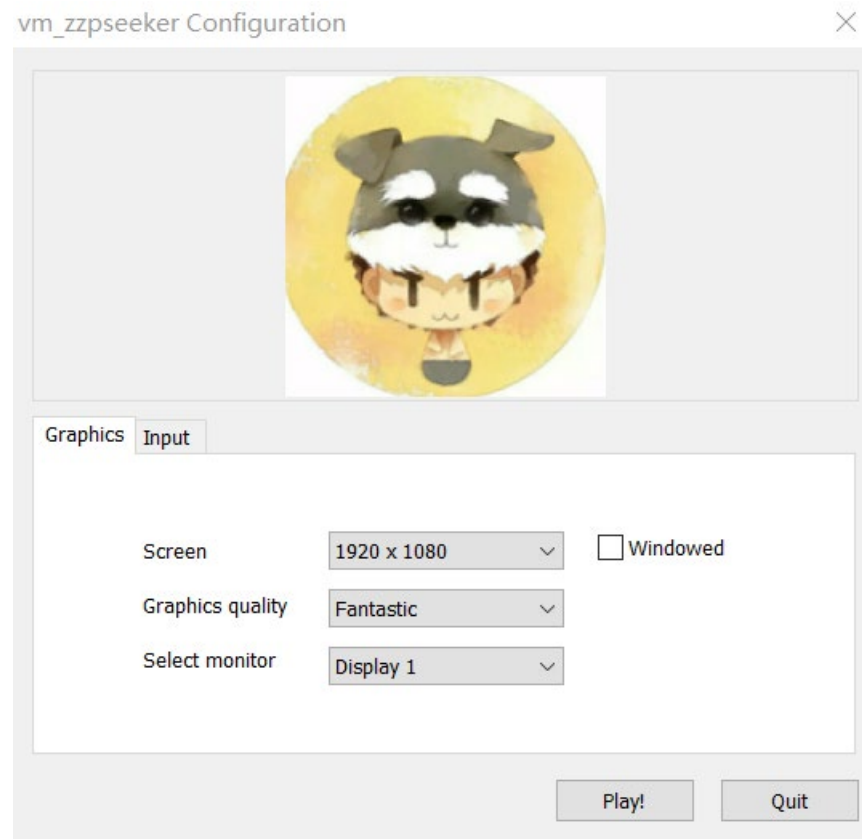
网页 (pc 端)：

<https://zzp-seeker.github.io/ee/>

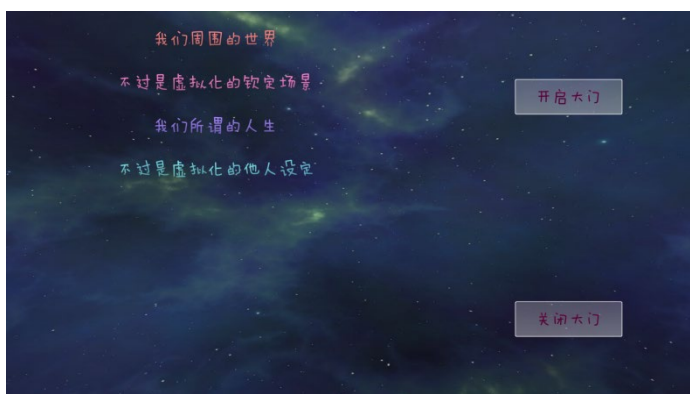
### 三：虚拟机使用帮助

#### 1. PC Windows

1) 安装完成后, 打开 VM\_zzpseeker.exe, 进入如下界面, 将 Screen 分辨率设为 1920\*1080, Graphics quality 建议设为 Fantastic, Select monitor 设为 Display 1。点击 play!



2) 打开后, 出现之下界面, 左边为动画式显示文本, 鼠标左键点击屏幕任意处可下一处文字快速到来, 点击开始大门进入虚拟机, 点击关闭大门退出。



3) 点击开始大门之后, 进入如下界面, 接下来介绍整个界面布局与各个按钮作用。



整个界面设计华丽，背景是星空天空盒，随机旋转视角，带你领略星空的神秘与美丽。

4) 输入指令有两种方式，一是点击右上角的铅笔，出现如下界面。



我们可以看见屏幕中间出现了一个面板，鼠标点击面板可以进行编辑，可手动输入，ctrl+c、ctrl+v、ctrl+a 等常用文本编辑指令也皆可用。面板右边有几个按钮，前五个按钮是一些样例，样例一是基本输入输出，样例二较为复杂的指令，样例三是斐波那契数列，样例四带有中断，样例五是错误样例，会输出异常情况，样例六是中断程序（带有中断字样）。

编辑完成后，再次点击铅笔，即可按照设置中的指令加入速度加入指令。

另外，在点击铅笔按钮后，若指令栏内没有指令，则编辑栏内默认显示样例 1。若指令栏内有指令，则编辑栏内显示的即为指令栏内所对应的指令。如下图所示。





第二种方式是通过屏幕下面的指令输入栏进行逐条输入。

点击指令输入栏，即可手动输入指令。我们可以看见指令输入栏四角分别有个小按钮，以及指令输入栏右边有两个朝向不同的箭头。接下来说明它们功能。左上按钮：复制指令栏内容；左下按钮：在该指令栏最后粘贴内容；右上按钮：清空指令栏内容；右下按钮：将指令栏内容全复制并清空指令栏；朝右箭头：加入指令；朝左箭头：将指令栏最后一条内容拿出并加入指令栏。在指令栏输入时会有**输入字符限制，只能输入英文字母和数字**。


另外，鼠标悬浮于四角的按钮会显示帮助文本信息，而且可以结合以上两种方式进行指令输入。

5)加入指令后，如下图所示，下图加入的是样例四，带有中断指令。





下面我们说说如何浏览各个指令，首先整个指令面板可以通过最右的滑条或者鼠标放置滑条稍左或者小手那一竖条的位置进行鼠标滚轮滚动或者鼠标左键上下拖动可以拖动界面内容。另外鼠标位于某个指令栏中也可进行鼠标滚轮滚动或者鼠标左右拖动可拖动指令内容。


在每个指令栏左面会有一个白色小框，点击后会让小手指向该指令。在整个指令栏左边的 Introduce 面板中会有小手指向的指令的功能简要介绍。

6) 接下来我们就可以运行指令了，点击屏幕正中间上方的运行按钮 ，这时候指令就开始按照设置中的指令运行速度运行。


此时虚拟机进入运行状态，在运行状态不能进行指令编辑等操作。在运行过程中，Memory 面板会将内存中数据输出，Output 面板会输出指令要求输出的内容，Register 面板显示各个寄存器的内容，Introduce 面板会显示该指令具体运行情况。另外，如不单击运行按钮，而是双击运行按钮，虚拟机直接进入暂停状态，单步执行。

在运行过程中，可以随时点击暂停按钮 ，虚拟机进入暂停状态。

此时虚拟机进入暂停状态，在暂停状态下，指令并不会自动运行。我们可以点击上一步按钮 ，查看上一步指令运行情况，但是已经执行过的指令并不会再执行一遍，仅供调试时

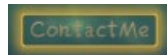
查看指令运行情况使用。我们也可以点击下一步按钮 ，若下一步指令并没有执行，则执行该指令，否则仅供查看该指令。

在暂停过程中，也可以随时再次点击暂停按钮，虚拟机又进入运行状态。

在遇到 STOP 指令或异常情况时，我们可以点击停止按钮 ，虚拟机处于就绪状态，这时候我们又可以编辑指令了。

### 总述：主要按钮功能

#### 1. ContactMe



鼠标悬浮上面可以以动画方式显示一串文字

#### 2. 上一步 暂停 运行 停止 下一步

**运行：**当虚拟机处于就绪状态时，点击运行即可按照设置中指令运行速度开始执行指令。

**暂停：**当虚拟机处于运行状态时，点击暂停即可进入暂停状态，再次点击该按钮即可进入运行状态。

**上一步：**当虚拟机处于暂停状态时，点击上一步可以查看上一步指令执行情况，但不会再次运行该指令。

**下一步：**当虚拟机处于暂停状态时，如果下一步指令未运行，点击下一步可以运行下一步指令，否则只是查看指令执行情况，但不会再次运行该指令。

**停止：**当虚拟机遇到 STOP 指令或者出现异常时，点击指令可停止虚拟机，使虚拟机重新进入就绪状态。

#### 3. 设置

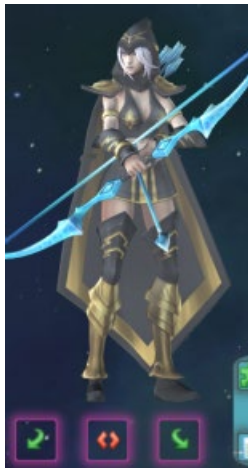


#### 编辑



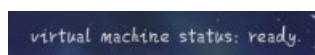
## 其他设计

### 1) .艾希英雄 3d 模型



可以看见艾希下方分别由三个按钮，边上两个可以长按旋转艾希，中间按钮单击可以变换动作。

### 2) .虚拟机状态显示



分别有四个状态：就绪，运行中，暂停中，停止四个状态。

### 3) .Tips 设计



在很多时候这里的 tip 会显示（如点击某个按钮，出现异常操作等），以此进一步增强用户交互体验。

### 4) .设置

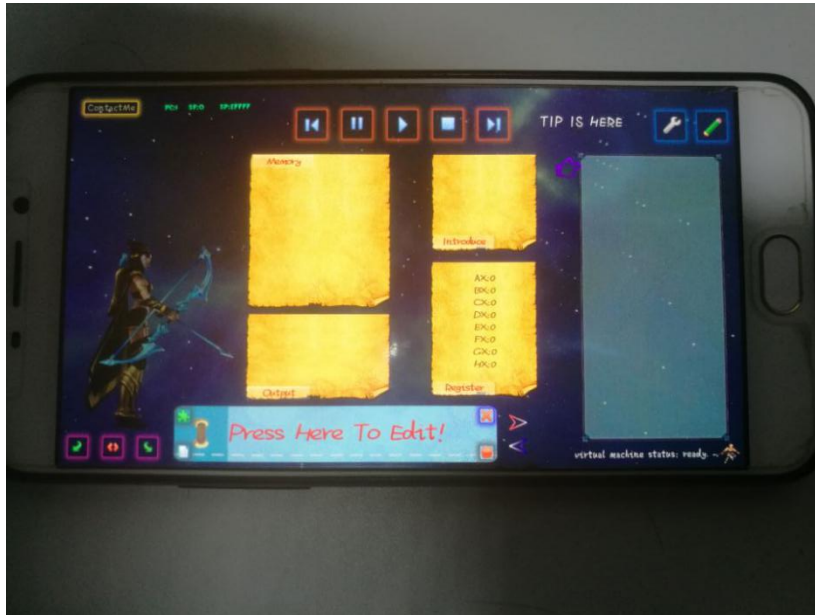


进入设置界面后，我们可以发现背景被模糊化，这是采用高斯模糊形成的，同时还有很多设置，点击 help 还可以获得本文档，十分人性化。



## 2. Android

该程序仅为主流屏幕分辨率 1920\*1080 设计，下载 apk 后，操作与 pc 相同，只要安卓系统较新，就不会有什么问题。以下为示意图。

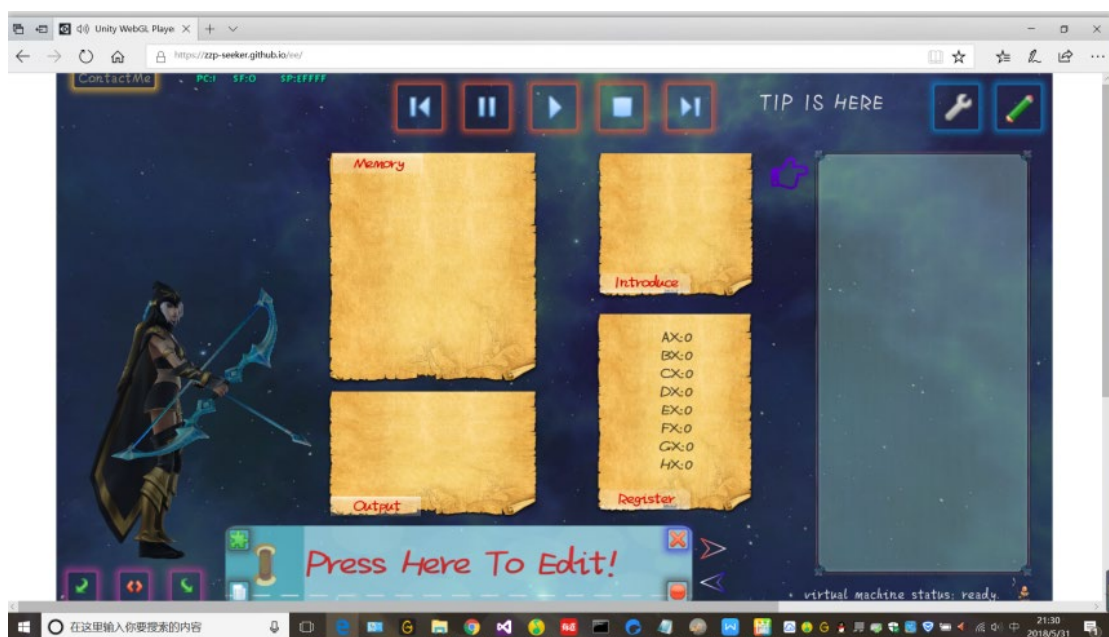


## 3. 网页 (PC)

由于手机网页分辨率以及其他原因，目前不适合手机上打开网页。

若通过网址打开，刚开始会有加载页面，加载完毕后，建议点击主画面右下角的全屏按钮，然后操作与 pc 相同。

若想打开本地 html 文件，则需要浏览器支持 webgl，则可以参考 <http://www.khwebgl.com/?p=628#WebGL-2> 该网页进行操作使浏览器支持 webgl。



运行情况如上图所示，点击主画面右下角的全屏按钮，即可进入畅快体验。

## 四：虚拟机设计描述

### 1. 总述

64 位 CPU，1M 内存，按字节寻址。

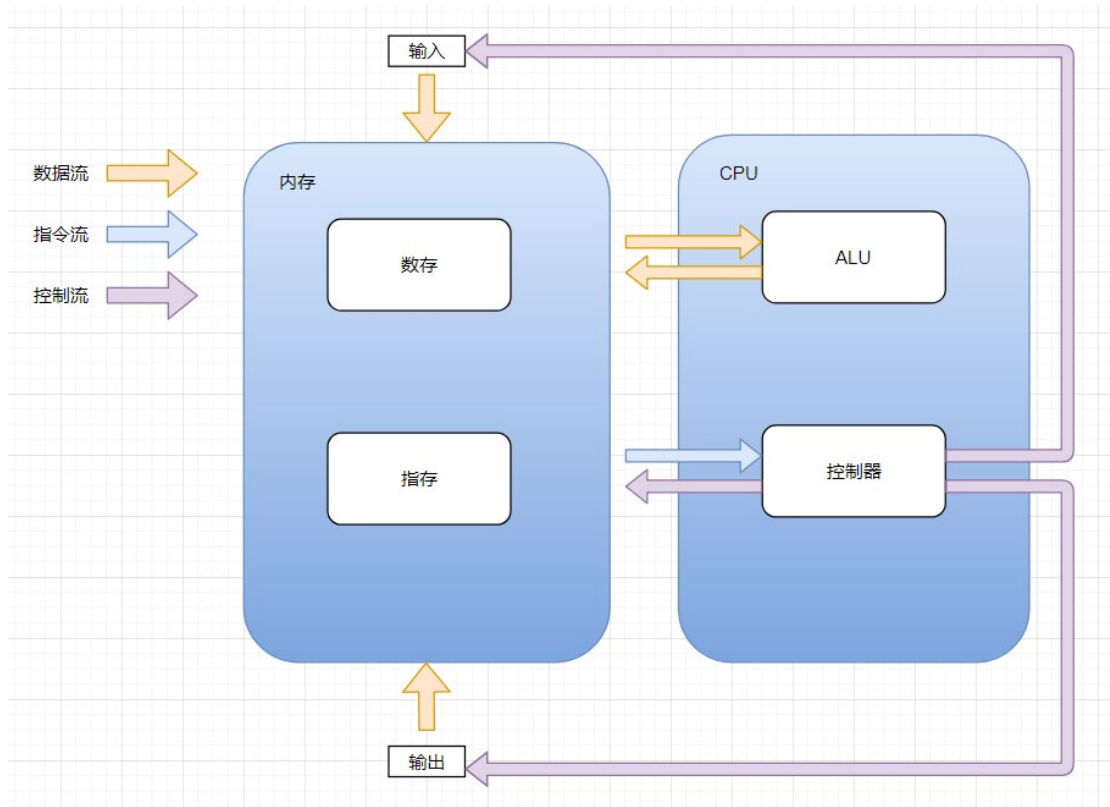
地址：00000 到 FFFFF，共  $2^{20}$  字节，

数据段：30000 到 CFFFF，共  $3 * 2^{18}$  字节。

寄存器：共 8 个：AX, BX, CX, DX, EX, FX, GX, HX，存储 64 位数。

CPU 处理的数据皆为 64 位（二进制）无符号整数。

### 2. 硬件体系结构



### 3. 操作数

指令中操作数共 3 种形式：立即数，寄存器，内存。

#### 1) 立即数：

一个 16 位的 16 进制常数，不会省略前导零，字母采用大写。如：02AC 0582 A598 C485。

#### 2) 寄存器：

通用寄存器：

AX, BX, CX, DX, EX, FX, GX, HX，均为大写。

指令寄存器：IR

程序计数器：PC

堆栈指针寄存器：SP

状态字寄存器：SF（SF 符号标志：当 CMP 指令结果为负时为 1，否则为 0）

#### 3) 内存：

有“立即数直接寻址”与“寄存器间接寻址”两种方式。



立即数直接寻址：T+立即数 如 T02C0 。

寄存器间接间接寻址：T+寄存器 如 TAX 。

存储模式：小端存储

整型数据的高字节保存在高地址中，数据的低字节保存在低地址中。该 CPU 处理的是 64 位数，占 8 个字节，故在内存中占相邻的 8 个地址。如读取某一地址的数据，则会读取到该地址以及其后 7 个相邻地址的数据，并将高地址作为高位，低地址作为低位，组成读取出来的 64 位数。如写入某一地址的数据，则会将 64 位数写入到 8 个相邻地址中。且高位写入高地址，低位写入低地址。

#### 4. 指令设计

共 22 种指令。

RUN：标识着程序的开始。如无特殊说明，内存和寄存器均已初始化为 0。

STOP：标识着程序的正常结束。

ECHO A：将操作数 A 中的值输出。

ADD A B：将操作数 A 中的值与 B 中的值相加，结果存回 A。相加产生溢出时，直接将溢出部分丢弃即可（截断）——无需向更高位进位，存回 A 的同样是一个 16 位（二进制位）无符号整数。A 不能为立即数，A、B 不能为字符串。

INC A：将操作数 A 中的值加 1，结果存回 A。同样忽略溢出，A 不能是立即数。

SUB A：将操作数 A 中的值减 1，结果存回 A。同样忽略溢出，A 不能是立即数。

AND A B：将操作数 A 与 B 进行与运算，结果存回 A，A 不能为立即数。

OR A B：将操作数 A 与 B 进行或运算，结果存回 A，A 不能为立即数。

XOR A B：将操作数 A 与 B 进行异或运算，结果存回 A，A 不能为立即数。

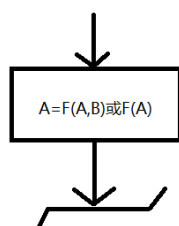
LSH A：将操作数 A 进行左移一位运算，结果存回 A，A 不能为立即数。

RSH A：将操作数 A 进行右移一位运算，结果存回 A，A 不能为立即数。

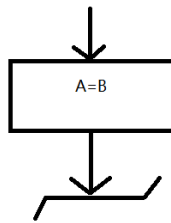
MUL A B：将操作数 A 中的值与 B 中的值相乘，结果存回 A。相乘产生溢出时，直接将溢出部分丢弃即可（截断）——无需向更高位进位，存回 A 的同样是一个 16 位（二进制位）无符号整数。A 不能为立即数。

DIV A B：将操作数 A 中的值与 B 中的值相除，结果存回 A。不能整除时，将小数点后丢弃，存回 A 的同样是一个 16 位无符号整数。A 不能为立即数。

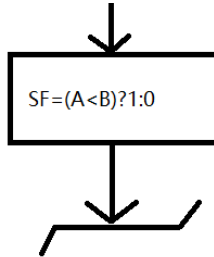
3-13 指令流程图 F 代表某操作



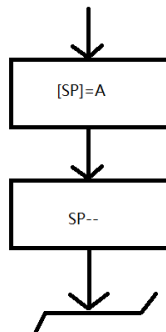
MOV A B : 将操作数  $B$  中的值写入  $A$ ,  $A$  不能是立即数。



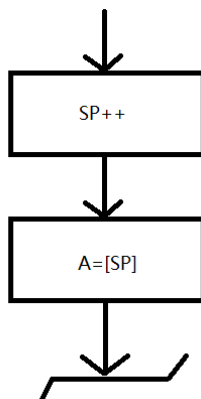
CMP A B: 比较操作数  $A$  和  $B$  中的值的大小, 结果存到状态字寄存器 SF 中。 $A$ ,  $B$  必须同为整型数。



PUSH A : 将操作数  $A$  中的值 push 入堆栈区。



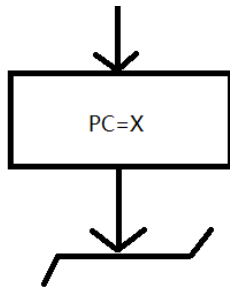
POP A : 将栈顶数值 pop 给  $A$ ,  $A$  不能是立即数。



跳转指令。

无条件跳转指令

JMP X 不妨假设操作数  $X$  中的值是  $i$ ，则该指令执行完后，将去执行第  $i$  条指令。

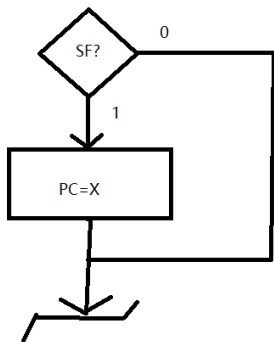


条件跳转指令

将状态字寄存器作为条件。

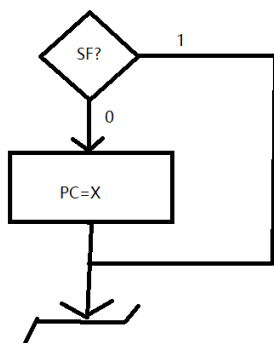
JSA X

当  $SF=1$  时跳转。



JSB X

当  $SF=0$  时跳转。



中断指令

INT: 设置中断

RTI: 从中断返回

特权指令

CRA: 停止 CPU 工作

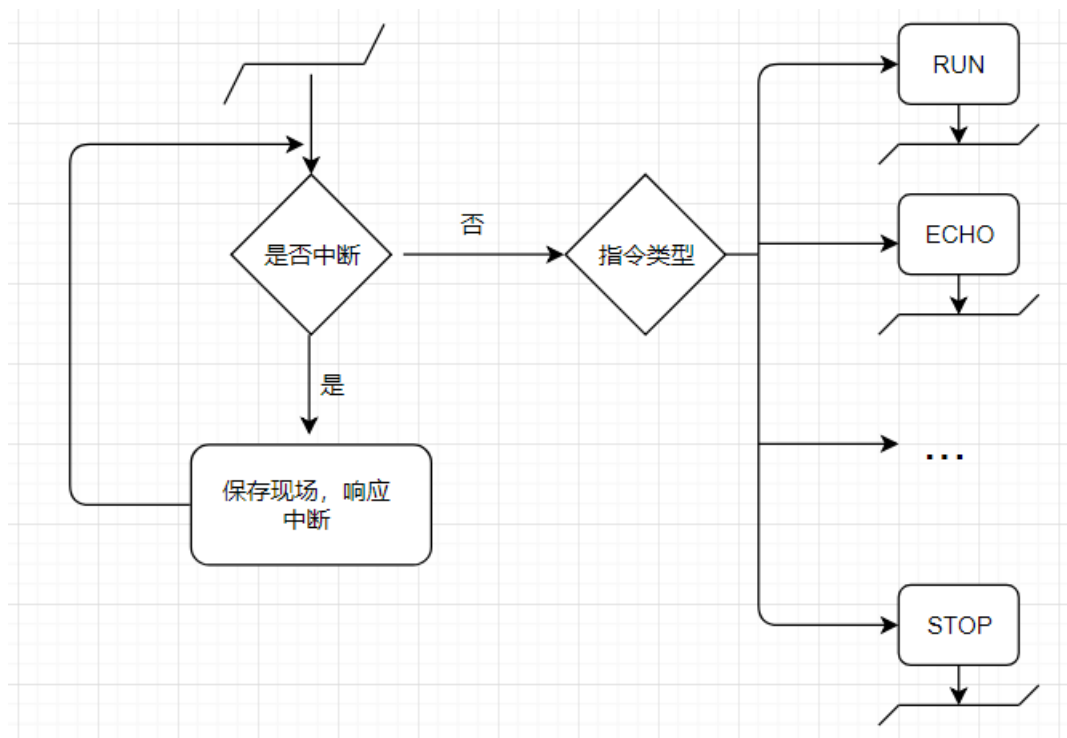
## 指令总结

操作指令中带有颜色的为操作数可以为字符串类型

指令类别	序号	操作码	功能
标识	1	RUN	标识程序开始
	2	STOP	标识程序结束
IO	3	ECHO A	输出 A
操作	4	ADD A B	A=A+B
	5	INC A	A=A+1
	6	SUB A	A=A-1
	7	AND A B	A=A and B
	8	OR A B	A=A or B
	9	XOR A B	A=A xor N
	10	LSH A	A<<
	11	RSH A	A>>
	12	MUL A B	A=A*B
	13	DIV A B	A=A/B
	14	CMP A B	SF=(A<B)?1:0
	15	MOV A B	A=B
跳转	16	JMP X	PC=X
	17	JSA X	if (SF) PC=X
	18	JSB X	if (!SF) PC=X
堆栈	19	PUSH A	[SP]=A SP--
	20	POP A	SP++ A=[SP]
中断	21	INT	设置中断
	22	RTI	中断返回
特权	23	CRA	停止 CPU 运行



#### 4. 指令执行流程图



#### 5. 指令执行异常情形

存取非法 ACCESS\_VIOLATION

跳转错误 RUNTIME\_ERROR

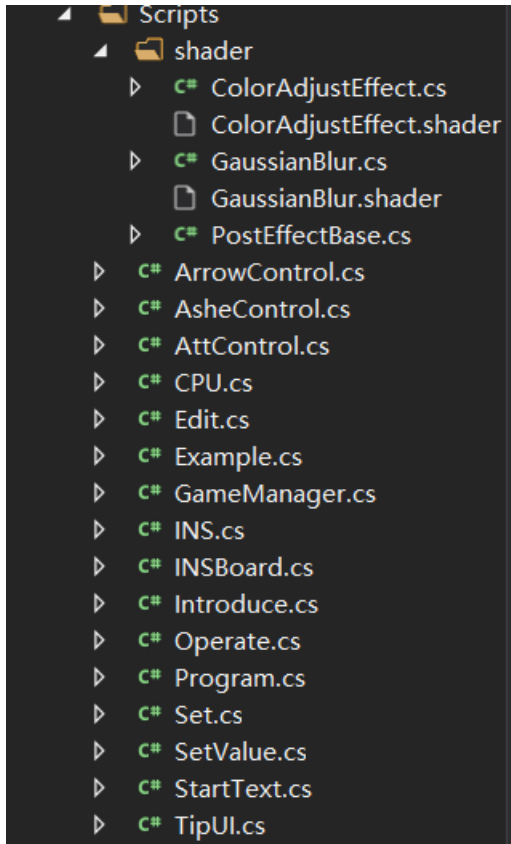
特权指令 ILLEGAL\_INSTRUCTION

不可识别 INSTRUCTION ISN'T RECOGNIZED

## 五：代码设计与说明

### 1. 总述

源码在 VM\_SourceCode/VM//Assets/Scripts 目录下。



其中 CPU.cs、Operate.cs、Program.cs 是内核程序代码，其余都是界面程序代码。在 shader 文件夹中有三个 C#程序源码以及两个 shader 源码。这是处理画面以及更换材质的代码，其中 shader 所用语言是 CG（GPU 高级绘制语言）。

如上图所示，共 21 个程序源码，在图中由上到下代码行数各程序代码行数分别为：35，79，46，91，40，68，118，66，144，76，145，393，48，288，67，414，49，181，36，81，54。总计 2500 余行代码。

为突出要点，之后会详述内核程序源码，而对界面以及 shader 源码只会简述。

### 2. 各程序代码说明

#### 1) CPU.cs

在该脚本中主要定义了 CPU 主要的状态字，常量等，并用 C#中的 Dictionary 容器模拟内存与寄存器，还定义了 CPU 初始化、保存现场、恢复现场等函数。并采用委托事件等 C#高级用法，当 CPU 一些状态改变时即使通知订阅该时间的对象。

```
public class CPU : MonoBehaviour { //CPU 类 继承 MonoBehaviour 类（unity 默认类）
    public bool intFlag = false;    //中断标志
    public bool RunFlag = false;    //运行标志
    public bool StopFlag = false;   //停止标志
```

```

public int WORD_LENGTH = 64;    //字长 64 位
public int PER_STO_LENGTH = 8;  //按字节编址
public delegate void valueChanged(string newValue, int index); //声明委托
public event valueChanged stateChangedEventListener; //声明与委托关联的事件
public List<int> PCList = new List<int>(); //记录 PC 值情况，以便调试时进行 PC 转移

int mPC = 1;
public int PC    //PC 值
{
    get { return mPC; }
    set
    {
        mPC = value;
        PCList.Add(value);
        if (stateChangedEventListener != null)
            stateChangedEventListener(value.ToString(), 1); //PC 改变时通知事件
    }
}

int mSF = 0;
public int SF    //状态字 SF，当负时为 1，其他为 0
{
    get { return mSF; }
    set
    {
        mSF = value;
        if (stateChangedEventListener != null)
            stateChangedEventListener(value.ToString(), 2); //SF 改变时通知事件
    }
}

string mSP = "EFFFF";
public string SP    //堆栈 SP 指针
{
    get { return mSP; }
    set
    {
        mSP = value;
        if (stateChangedEventListener != null)
            stateChangedEventListener(value, 3);    //SP 改变时通知事件
    }
}

```

```

public bool intRunFlag;      //前缀 int 表明是中断程序
public bool intStopFlag;
public List<int> intPCList;
public int intPC;
public int intSF;
public string intSP;
public Dictionary<string, string> intAddress= new Dictionary<string, string>();

public Dictionary<string, string> Address = new Dictionary<string, string>();
//内存与寄存器 AX, BX, CX, DX, EX, FX, GX, HX

public void Init()    //初始化 CPU
{
    Address.Clear();
    PCList.Clear();
    Address.Add("AX", "0"); Address.Add("BX", "0"); Address.Add("CX", "0");
Address.Add("DX", "0");
    Address.Add("EX", "0"); Address.Add("FX", "0"); Address.Add("GX", "0");
Address.Add("HX", "0");
    RunFlag = false;
    StopFlag = false;
    PC = 1;
    SF = 0;
    intFlag = false;
    SP = "FFFFFF";
}

public void intInit() //初始化中断程序
{
    //初始化 8 个寄存器
    intAddress.Clear();
    intPCList.Clear();
    intRunFlag = false;
    intStopFlag = false;
    intPC = 1;
    intSF = 0;
    intFlag = false;
    intSP = "FFFFFF";
}

void Awake()
{
    Init(); //脚本被唤醒时调用 awake
}

```



```

public void Store() //保存现场
{
    intRunFlag = RunFlag;
    intStopFlag = StopFlag;
    intPC = PC;
    intSF = SF;
    intSP = SP;
    intPCList = PCList;

    foreach (KeyValuePair<string, string> kvp in Address)
        intAddress.Add(kvp.Key, kvp.Value);

    Init();
    intFlag = true;

    GameManager.Game.UpdateMemory();
}

public void ReStore() //恢复现场
{
    RunFlag = intRunFlag;
    StopFlag = intStopFlag;
    PC = intPC;
    SF = intSF;
    SP = intSP;
    PCList = intPCList;

    Address.Clear();
    foreach (KeyValuePair<string, string> kvp in intAddress)
        Address.Add(kvp.Key, kvp.Value);

    intFlag = false;

    foreach (KeyValuePair<string, string> kvp in Address)
        Debug.Log("address:" + kvp.Key + "value" + kvp.Value);

    GameManager.Game.UpdateMemory();
}
}

```

## 2) Operate.cs

在该脚本里主要模拟了读取指令、识别指令以及指令所要求的一些操作。注意很多函数返回 string 类型，这是因为返回的是一些相关的输出字符串，在之后的关键的 F 与 G 函数进行调用。F 函数是读取该指令，但并不运行（在加入指令后仅供简要显示这个指令作用）。G 函数是读取该指令，且运行。

```
public class Operate : MonoBehaviour {

    public string FoundAddress(string x)    //寻址 例:T00055:寻址到 00055    TAX:设
    AX=00033 寻址到 00033    AX:寻寄存器到 AX
    {
        if (x != null)
        {
            if (x[0] == 'T') {
                x = x.Substring(1);
                if (x.Length == 2 && x[1] == 'X') {
                    if (!Camera.main.GetComponent<CPU>().Address.ContainsKey(x))
//异常处理
                    {
                        Debug.Log("1");
                        Camera.main.GetComponent<CPU>().StopFlag = true;
                        GameManager.Game.State = 3;
                        GameManager.Game.Tip("ACCESS_VIOLATION:Memory do not contain the
key of " + x + " " + x.Length);
                        return "ERROR";
                    }
                    x = Camera.main.GetComponent<CPU>().Address[x];
                }
            }
        }
        return x;
    }

    public string FoundNum(string x)        //寻数 例:00055:数值 00055    T00055:地址 00055 及
    后面 7 个的值    TAX    AX:寄存器里的值
    {
        if (x != null)
        {
            if (x[0] == 'T')
            {
                x = x.Substring(1);
                if (x.Length == 2 && x[1] == 'X')
                {
                    if (!(x[0] >= 'A' && x[0] <= 'H'))
```

```

        {
            Debug.Log("2");
            Camera.main.GetComponent<CPU>().StopFlag = true;
            GameManager.Game.State = 3;
            GameManager.Game.Tip("ERROR:ACCESS_VIOLATION");
            return "ERROR";
        }
        x = StdData(Camera.main.GetComponent<CPU>().Address[x]);
    }
    else x = StdData(x);
}
else if (x.Length == 2 && x[1] == 'X')
{
    if (!(x[0] >= 'A' && x[0] <= 'H'))
    {
        Debug.Log("3");
        Camera.main.GetComponent<CPU>().StopFlag = true;
        GameManager.Game.State = 3;
        GameManager.Game.Tip("ERROR:ACCESS_VIOLATION");
        return "ERROR";
    }
    x = Camera.main.GetComponent<CPU>().Address[x];
}
}
return x;
}

```

public string[] SplitData(string x) //为了小端模式存储，需将 16 位 16 进制拆成 8 个 2 位 16 进制(8 个 8 位，8 个字节)数，存回数组

```

{
    string[] z = new string[Camera.main.GetComponent<CPU>().WORD_LENGTH /
Camera.main.GetComponent<CPU>().PER_STO_LENGTH];
    for (int i = 0; i < z.Length; i++)
    {
        z[i] = ""; z[i] += x[i * 2].ToString(); z[i] += x[i * 2 + 1].ToString();
    }
    return z;
}

```

public string CombineData(string[] x) //将 8 个地址的数据合到一起

```

{
    string z = "";
    foreach (string i in x) z += i;
    return z;
}

```

```

        public string StdData(string x)           //给出地址取数一下取 8 个数据，给出寄存器则取
寄存器里的数据，返回该取出的数据
        {
            x = FoundAddress(x);
            if (x == "ERROR") return "ERROR";
            if (x.Length == 2 && x[1] == 'X')
            {
                if (!Camera.main.GetComponent<CPU>().Address.ContainsKey(x))
//异常处理
                {
                    Debug.Log("4");
                    Camera.main.GetComponent<CPU>().StopFlag = true;
                    GameManager.Game.State = 3;
                    GameManager.Game.Tip("ERROR:ACCESS_VIOLATION.Memory do not contain the
key of " + x + " " + x.Length);
                    return "ERROR";
                }
                return Camera.main.GetComponent<CPU>().Address[x];
            }
            else
            {
                int size = Camera.main.GetComponent<CPU>().WORD_LENGTH /
Camera.main.GetComponent<CPU>().PER_STO_LENGTH;
                string[] z = new string[size];
                for (int i = size - 1; i >= 0; i--)
                {
                    if (!Camera.main.GetComponent<CPU>().Address.ContainsKey(x))
                    {
                        Camera.main.GetComponent<CPU>().StopFlag = true;
                        GameManager.Game.State = 3;
                        GameManager.Game.Tip("ERROR:ACCESS_VIOLATION.Memory do not contain
the key of " + x + " " + x.Length);
                        return "ERROR";
                    }

                    z[i] = Camera.main.GetComponent<CPU>().Address[x];
                    x = Inc(x);
                }
                return CombinedData(z);
            }
        }

        public string StoData(string x, string y) //数据 y 存到地址 x 中,小端模式存储 返回描
述

```



```

{
    string result = "";
    x = FoundAddress(x); y = FoundNum(y);
    if (x == "ERROR" || y == "ERROR") return "ERROR";
    if (x.Length == 2 && x[1] == 'X')
    {
        if (!(x[0] >= 'A' && x[0] <= 'H'))
        {
            Debug.Log("6");
            Camera.main.GetComponent<CPU>().StopFlag = true;
            GameManager.Game.State = 3;
            GameManager.Game.Tip("ERROR:ACCESS_VIOLATION");
            return "ERROR";
        }
        Camera.main.GetComponent<CPU>().Address[x] = y;
        result += y + "->" + x;
    }
    else
    {
        y = Add0To16(y);
        int size = Camera.main.GetComponent<CPU>().WORD_LENGTH /
Camera.main.GetComponent<CPU>().PER_STO_LENGTH;
        string[] z = SplitData(y);
        for (int i = size - 1; i >= 0; i--)
        {
            if(!TestStr(x))
            {
                Debug.Log("7");
                Camera.main.GetComponent<CPU>().StopFlag = true;
                GameManager.Game.State = 3;
                GameManager.Game.Tip("ERROR:ACCESS_VIOLATION");
                return "ERROR";
            }

            Camera.main.GetComponent<CPU>().Address[x] = z[i];
            result += z[i] + "->" + x + " ";
            x = Inc(x);
        }
    }
    GameManager.Game.UpdateMemory();
    return result;
}

public string Add0To16(string x) //加前导0 进行格式化

```

```

{
    if (x.Length < 16) //数据 y 必须是 16 位 16 进制数
    {
        string s = "";
        for (int i = 0; i < 16 - x.Length; i++) s += "0";
        x = s + x;
    }
    return x;
}

public string And(string x, string y) //与操作
{
    x = Add0To16(FoundNum(x)); y = Add0To16(FoundNum(y));
    string result1 = "", result2 = "", result = "";
    foreach (char c in x)
        result1 += String.Format("{0:0000}",
int.Parse(Convert.ToString(Convert.ToInt32(c.ToString(), 16), 2))); //16 进制转 2 进制
    foreach (char c in y)
        result2 += String.Format("{0:0000}",
int.Parse(Convert.ToString(Convert.ToInt32(c.ToString(), 16), 2)));
    for (int i = 0; i < result1.Length; i++)
    {
        if (result1[i] == '1' && result2[i] == '1') result += "1";
        else result += "0";
    }
    return String.Format("{0:X16}", Convert.ToUInt64(result, 2)); //2 进制转
16 进制
}

public string Or(string x, string y) //或操作
{
    x = Add0To16(FoundNum(x)); y = Add0To16(FoundNum(y));
    string result1 = "", result2 = "", result = "";
    foreach (char c in x)
        result1 += String.Format("{0:0000}",
int.Parse(Convert.ToString(Convert.ToInt32(c.ToString(), 16), 2))); //16 进制转 2 进制
    foreach (char c in y)
        result2 += String.Format("{0:0000}",
int.Parse(Convert.ToString(Convert.ToInt32(c.ToString(), 16), 2)));
    for (int i = 0; i < result1.Length; i++)
    {
        if (result1[i] == '0' && result2[i] == '0') result += "0";
        else result += "1";
    }
    return String.Format("{0:X16}", Convert.ToUInt64(result, 2)); //2 进制转 16 进制
}

```

```

    }

    public string Xor(string x, string y) //异或操作
    {
        x = Add0To16(FoundNum(x)); y = Add0To16(FoundNum(y));
        string result1 = "", result2 = "", result = "";
        foreach (char c in x)
            result1 += String.Format("{0:0000}",
int.Parse(Convert.ToString(Convert.ToInt32(c.ToString(), 16), 2))); //16 进制转 2 进制
        foreach (char c in y)
            result2 += String.Format("{0:0000}",
int.Parse(Convert.ToString(Convert.ToInt32(c.ToString(), 16), 2)));
        for (int i = 0; i < result1.Length; i++)
        {
            if (result1[i] == result2[i]) result += "0";
            else result += "1";
        }
        return String.Format("{0:X16}", Convert.ToUInt64(result, 2)); //2 进制转
16 进制
    }

    public string Add(string x, string y) //加运算
    {
        x = Add0To16(FoundNum(x)); y = Add0To16(FoundNum(y));
        ulong z = Convert.ToUInt64(x, 16) + Convert.ToUInt64(y, 16);
        return String.Format("{0:X16}", z);
    }

    public string Mul(string x, string y) //乘运算
    {
        x = Add0To16(FoundNum(x)); y = Add0To16(FoundNum(y));
        ulong z = Convert.ToUInt64(x, 16) * Convert.ToUInt64(y, 16);
        return String.Format("{0:X16}", z);
    }

    public string Div(string x, string y) //除运算
    {
        x = Add0To16(FoundNum(x)); y = Add0To16(FoundNum(y));
        ulong z = (Convert.ToUInt64(x, 16) / Convert.ToUInt64(y, 16)) % 0x10000;
        return String.Format("{0:X16}", z);
    }

    public string Inc(string x) //自增运算
    {
        x = FoundNum(x);
        if (x.Length == 5)
        {

```

```

        if (x == "FFFF") return "00000";
        else return String.Format("{0:X5}", Convert.ToInt32(x, 16) + 0x1);
    }
    else
    {
        if (x == "FFFFFFFFFFFFFF") return "0000000000000000";
        else return String.Format("{0:X16}", Convert.ToUInt64(x, 16) + 0x1);
    }
}

public string Sub(string x)    //自减运算
{
    x = FoundNum(x);
    if (x.Length == 5)
    {
        if (x == "00000") return "FFFFF";
        else return String.Format("{0:X5}", Convert.ToInt32(x, 16) - 0x1);
    }
    else
    {
        if (x == "0000000000000000") return "FFFFFFFFFFFFFF";
        else return String.Format("{0:X16}", Convert.ToUInt64(x, 16) - 0x1);
    }
}

public string Lsh(string x)    //左移操作
{
    x = Add0To16(FoundNum(x));
    return String.Format("{0:X16}", Convert.ToUInt64(x, 16) << 1);
}

public string Rsh(string x)    //右移操作
{
    x = Add0To16(FoundNum(x));
    return String.Format("{0:X16}", Convert.ToUInt64(x, 16) >> 1);
}

public int Cmp(string x, string y)    //比较得到 SF 值
{
    x = Add0To16(FoundNum(x)); y = Add0To16(FoundNum(y));
    if (x.CompareTo(y) < 0) return 1;
    else return 0;
}

public bool p = false; //是否在进行堆栈操作，便于之后处理异常使用
public string Push(string x)    //返回描述
{

```



```

        p = true;
        for (int i = 0; i < 7; i++) Camera.main.GetComponent<CPU>().SP =
Sub(Camera.main.GetComponent<CPU>().SP);
        string result=StoData("T" + Camera.main.GetComponent<CPU>().SP, x);
        Camera.main.GetComponent<CPU>().SP = Sub(Camera.main.GetComponent<CPU>().SP);
        p = false;
        return "压栈操作" + result;
    }

    public string Pop(string x)
    {
        p = true;
        Camera.main.GetComponent<CPU>().SP = Inc(Camera.main.GetComponent<CPU>().SP);
        string result = StoData(x, "T" + Camera.main.GetComponent<CPU>().SP);
        for (int i = 0; i < 7; i++)
        {

Camera.main.GetComponent<CPU>().Address.Remove(Camera.main.GetComponent<CPU>().SP);
            Debug.Log("pop:" + Camera.main.GetComponent<CPU>().SP);
            Camera.main.GetComponent<CPU>().SP =
Inc(Camera.main.GetComponent<CPU>().SP);
        }

Camera.main.GetComponent<CPU>().Address.Remove(Camera.main.GetComponent<CPU>().SP);
        GameManager.Game.UpdateMemory();
        p = false;
        return "出栈操作" + result;
    }

    public string[] AnalysisI(string x)    //解析指令
    {
        if (x != "")
            return x.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
        return new string[] { "" };
    }

    public string F(string[] x)    //显示指令
    {
        for(int i=0;i<x.Length;i++)
            x[i] = x[i].Trim();
        if (x[0] == "RUN") { Camera.main.GetComponent<CPU>().RunFlag = true;
Camera.main.GetComponent<CPU>().PC++; return "开机..."; } //开机操作
        else if (x[0] == "STOP") { Camera.main.GetComponent<CPU>().StopFlag = true;
GameManager.Game.State = 3; return "关机..."; } //关机操作
    }

```

```

        else if (x[0] == "ECHO") { string s = StdData(x[1]);
GameManager.Game.UpdateOutput(s); Camera.main.GetComponent<CPU>().PC++; return "输出" +
s; } //输出操作
        else if (x[0] == "MOV") { Camera.main.GetComponent<CPU>().PC++; return StoData(x[1],
x[2]); } //赋值操作
        else if (x[0] == "ADD") { Camera.main.GetComponent<CPU>().PC++; return StoData(x[1],
Add(x[1], x[2])); } //加运算
        else if (x[0] == "INC") { Camera.main.GetComponent<CPU>().PC++; return StoData(x[1],
Inc(x[1])); } //自增运算
        else if (x[0] == "SUB") { Camera.main.GetComponent<CPU>().PC++; return StoData(x[1],
Sub(x[1])); } //自减运算
        else if (x[0] == "AND") { Camera.main.GetComponent<CPU>().PC++; return StoData(x[1],
And(x[1], x[2])); } //与运算
        else if (x[0] == "OR") { Camera.main.GetComponent<CPU>().PC++; return StoData(x[1],
Or(x[1], x[2])); } //或运算
        else if (x[0] == "XOR") { Camera.main.GetComponent<CPU>().PC++; return StoData(x[1],
Xor(x[1], x[2])); } //异或运算
        else if (x[0] == "LSH") { Camera.main.GetComponent<CPU>().PC++; return StoData(x[1],
Lsh(x[1])); } //左移运算
        else if (x[0] == "RSH") { Camera.main.GetComponent<CPU>().PC++; return StoData(x[1],
Rsh(x[1])); } //右移运算
        else if (x[0] == "MUL") { Camera.main.GetComponent<CPU>().PC++; return StoData(x[1],
Mul(x[1], x[2])); } //乘运算
        else if (x[0] == "DIV") { Camera.main.GetComponent<CPU>().PC++; return StoData(x[1],
Div(x[1], x[2])); } //除运算
        else if (x[0] == "CMP") { Camera.main.GetComponent<CPU>().PC++;
Camera.main.GetComponent<CPU>().SF = Cmp(x[1], x[2]); return "将" +
Add0To16(FoundNum(x[1])) + "与" + Add0To16(FoundNum(x[2])) + "比较"; } //比较操作
        else if (x[0] == "PUSH") { Camera.main.GetComponent<CPU>().PC++; return
Push(x[1]); } //压栈操作
        else if (x[0] == "POP") { Camera.main.GetComponent<CPU>().PC++; return
Pop(x[1]); } //出栈操作
        else if (x[0] == "JMP") { Camera.main.GetComponent<CPU>().PC =
Convert.ToInt32(FoundNum(x[1]), 10); return "跳转第" + Camera.main.GetComponent<CPU>().PC
+ "条的指令"; } //无条件跳转
        else if (x[0] == "JSA")
        {
            if (Camera.main.GetComponent<CPU>().SF == 1)
            { Camera.main.GetComponent<CPU>().PC = Convert.ToInt32(FoundNum(x[1]), 10); return "SF==1
            故跳转至第" + Convert.ToInt32(FoundNum(x[1]), 10) + "条的指令"; }
            else { Camera.main.GetComponent<CPU>().PC++; return "SF==0 故不跳转"; }
        } //SF=1 时跳转
        else if (x[0] == "JSB")
        {

```

```

        if (Camera.main.GetComponent<CPU>().SF == 0)
        { Camera.main.GetComponent<CPU>().PC = Convert.ToInt32(FoundNum(x[1]), 10); return "SF==0
故跳转至第" + Convert.ToInt32(FoundNum(x[1]), 10) + "条的指令"; }
        else { Camera.main.GetComponent<CPU>().PC++; return "SF==1 故不跳转"; }
    } //SF=0 时跳转
    else if (x[0] == "INT") { Camera.main.GetComponent<CPU>().PC++; Program INT = new
Program(); INT.F(); return "中断"; }
    else if (x[0] == "RTI") { Camera.main.GetComponent<CPU>().StopFlag = true;
Debug.Log("返回中断"); return "返回中断"; }
    else if (x[0] == "CRA") { Camera.main.GetComponent<CPU>().StopFlag = true;
GameManager.Game.State = 3; return "ERROR:ILLEGAL_INSTRUCTION. 特权指令：关闭 CPU"; }
    else
    {
        Camera.main.GetComponent<CPU>().StopFlag = true; GameManager.Game.State = 3;
        return "Instruction is not recognized";
    }
}

public string G(string[] x) //仅仅显示指令（指令运行前显示）
{
    for (int i = 0; i < x.Length; i++)
        x[i] = x[i].Trim();
    if (x[0] == "RUN") { return "开机..."; } //开机操作
    else if (x[0] == "STOP") { return "关机..."; } //关机操作
    else if (x[0] == "ECHO") { return "输出" + x[1] + "的内容"; } //输出操作
    else if (x[0] == "MOV") { return x[2] + "->" + x[1] + "(内存小端模式存储)"; } //
赋值操作
    else if (x[0] == "ADD") { return x[1] + "+" + x[2] + "->" + x[1] + "(内存小端模
式存储)"; } //加运算
    else if (x[0] == "INC") { return x[1] + "自增" + "->" + x[1] + "(内存小端模式存
储)"; } //自增运算
    else if (x[0] == "SUB") { return x[1] + "自减" + "->" + x[1] + "(内存小端模式存
储)"; } //自减运算
    else if (x[0] == "AND") { return x[1] + "与" + x[2] + "->" + x[1] + "(内存小端模
式存储)"; } //与运算
    else if (x[0] == "OR") { return x[1] + "或" + x[2] + "->" + x[1] + "(内存小端模
式存储)"; } //或运算
    else if (x[0] == "XOR") { return x[1] + "异或" + x[2] + "->" + x[1] + "(内存小端
模式存储)"; } //异或运算
    else if (x[0] == "LSH") { return x[1] + "左移" + "->" + x[1] + "(内存小端模式存
储)"; } //左移运算
    else if (x[0] == "RSH") { return x[1] + "右移" + "->" + x[1] + "(内存小端模式存
储)"; } //右移运算
    else if (x[0] == "MUL") { return x[1] + "乘" + x[2] + "->" + x[1] + "(内存小端模
式存储)"; } //乘运算

```

```

        else if (x[0] == "DIV") { return x[1] + "除以" + x[2] + "->" + x[1] + "(内存小端
模式存储)"; } //除运算
        else if (x[0] == "CMP") { return "将" + x[1] + "与" + x[2] + "比较"; } //比较操作
        else if (x[0] == "PUSH") { return "将" + x[1] + "压栈"; } //压栈操作
        else if (x[0] == "POP") { return "出栈至" + x[1] + "中"; } //出栈操作
        else if (x[0] == "JMP") { return "跳转至第" + x[1] + "条的指令"; } //无条件跳转
        else if (x[0] == "JSA") { return "SF=1 时跳转至第" + x[1] + "条的指令"; } //SF=1
时跳转
        else if (x[0] == "JSB") { return "SF=0 时跳转至第" + x[1] + "条的指令"; } //SF=0
时跳转

        else if (x[0] == "INT") { return "中断"; }
        else if (x[0] == "RTI") { return "返回中断"; }
        else if (x[0] == "CRA") { return "特权指令：关闭 CPU"; }
        else { return "Instruction is not recognized"; }
    }

    bool TestStr(string x)
    {
        if (p) return true;
        if (x.Length == 5
            && ((x[0] >= '0' && x[1] <= '9') || (x[0] >= 'A' && x[1] <= 'C'))
            && ((x[1] >= '0' && x[1] <= '9') || (x[1] >= 'A' && x[1] <= 'F'))
            && ((x[2] >= '0' && x[2] <= '9') || (x[2] >= 'A' && x[2] <= 'F'))
            && ((x[3] >= '0' && x[3] <= '9') || (x[3] >= 'A' && x[3] <= 'F'))
            && ((x[4] >= '0' && x[4] <= '9') || (x[4] >= 'A' && x[4] <= 'F')))
            return true;
        return false;
    }
}

```

### 3) 其余脚本概述

Program.cs:中断程序执行

ArrowControl.cs:控制指令板旁小手移动

AsheControl.cs: 控制艾希模型

AttControl.cs:控制天空盒背景

Edit.cs:控制编辑按钮作用

GameManager.cs:

控制指令运行以及 memory、output、register、tip 面板等内容

INS.cs:单个指令脚本

INSBoard.cs:控制指令面板

Introduce.cs:控制 Introduce 面板内容

Set.cs:控制设置面板

SetValue.cs:控制设置面板值与滑条对应

StartText.cs:控制初始界面文本内容

TipUI.cs:控制某些按钮对应显示 Tip 板块的内容  
PostEffectBase.cs:屏幕后处理特效基类  
ColorAdjustEffect.shader:屏幕后处理渲染 shader  
ColorAdjustEffect.cs:基于相应 shader 实现控制屏幕对比度、饱和度等  
GaussianBlur.shader:高斯模糊渲染 shader  
GaussianBlur.cs:基于相应 shader 实现高斯模糊

## 六：样例测试

```
RUN
MOV T40000 1122334455667788
MOV T40004 0000002233445566
MOV AX T40002
AND T40001 AX
ECHO T40001
OR AX T40003
ECHO AX
MOV BX 40004
XOR AX TBX
ECHO AX
MOV CX AX
MOV T4000A 0022336677661124
INC CX
CMP T4000A CX
SUB T4000A
JSB 14
ECHO T4000A
PUSH T40008
JMP 22
POP DX
PUSH BX
INT
POP EX
ECHO EX
STOP
```