

虚拟机设计报告

虚拟机设计报告

一、基本信息

- 实验目的
- 虚拟机基本信息

二、组成部件

三、处理器架构

- 通用寄存器与寻址方式
 - 寄存器列表:
 - 寻址方式:
- 特殊功能寄存器
- 机器状态
- 指令类型及格式
- 取指流程
- 指令列表
 - 特殊指令
 - 传送指令
 - 双操作数算术与逻辑运算指令
 - 单操作数算术与逻辑运算指令
 - 测试指令
 - 跳转指令
 - 堆栈操作
 - 中断操作
- 中断
 - 中断向量
 - 中断操作的实现

四、虚拟存储系统

- 预留空间
- 用户代码区
- 栈区
- 用户数据区
- 输入缓冲区
- 输出缓冲区
- 显示缓冲区

五、汇编语言

- 语法单元
- 语法要求
- 函数调用
 - 寄存器内容的保存
 - 函数参数
 - 返回值
- 库函数

六、虚拟机实现框架

- 程序结构
- 虚拟存储系统程序框架
 - AbstractFile接口
 - ReadableFile接口

- iii. WritableFile接口
- iv. Memory类
- v. Keyboard类
- vi. TextOutput类
- vii. Display类
- viii. IOBridge类
- 3. 处理器程序框架
- 4. 主类框架
- 七、测试
 - 1. 求素数程序
 - 2. 生命游戏
- 八、问题与后续计划
 - 1. 总结
 - 2. 当前问题
 - 3. 后续计划
- 附件
 - 文件结构(Intelij IDEA项目)
 - Main.java
 - Processor.java
 - AbstractFile.java
 - ReadableFile.java
 - WritableFile.java
 - IOBridge.java
 - Memory.java
 - Keyboard.java
 - TextOutput.java
 - Display.java

一、基本信息

1. 实验目的

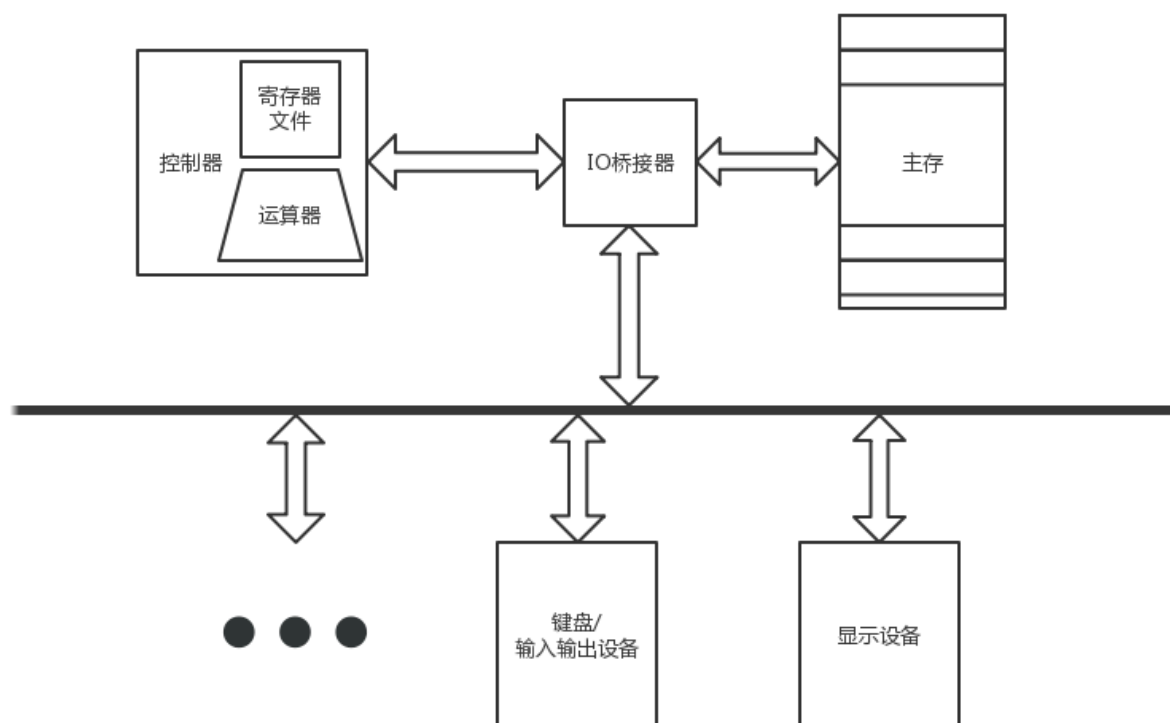
此虚拟机是计算机组成原理课程设计的一项重要内容，目的是通过高级语言实现软件，模拟冯诺依曼结构计算机系统及其工作原理，直观地显示机器运行的过程，综合运用软硬件知识，提升自身对计算机的认知水平。

2. 虚拟机基本信息

本次实验使用高级语言构建虚拟机，模拟冯诺依曼结构计算机的运行。此虚拟机使用Java 8(Intelij IDEA) 开发，测试环境为Windows 10 64bit，拥有文本和图形用户界面。存储系统包含寄存器与主存以及文件设备，不设缓存。指令系统源自Y86-64架构，目前仅支持有符号64位整型的算数与逻辑运算。

二、组成部件

结构示意图:



- **控制器**：通过函数来控制各部件的运行，模拟控制器的工作。
- **运算器**：用高级语言的计算功能模拟运算器的工作。
- **寄存器**：15个通用寄存器，3个专用寄存器。通用寄存器使用长度为15的64位整型向量来模拟；专用寄存器用成员变量保存。
- **IO桥接器**：提供虚拟地址映射。
- **存储器**：用字节数组模拟。
- **键盘输入部件**：用数组模拟输入缓冲区，与存储器共用虚拟地址编码。
- **文本输出部件**：用数组模拟输出缓冲区，与存储器共用虚拟地址编码。
- **显示设备**：用数组模拟显示缓冲区，与存储器共用虚拟地址编码。
- **总线**：体现为控制函数中的参数和局部变量，起到传输数据的作用。

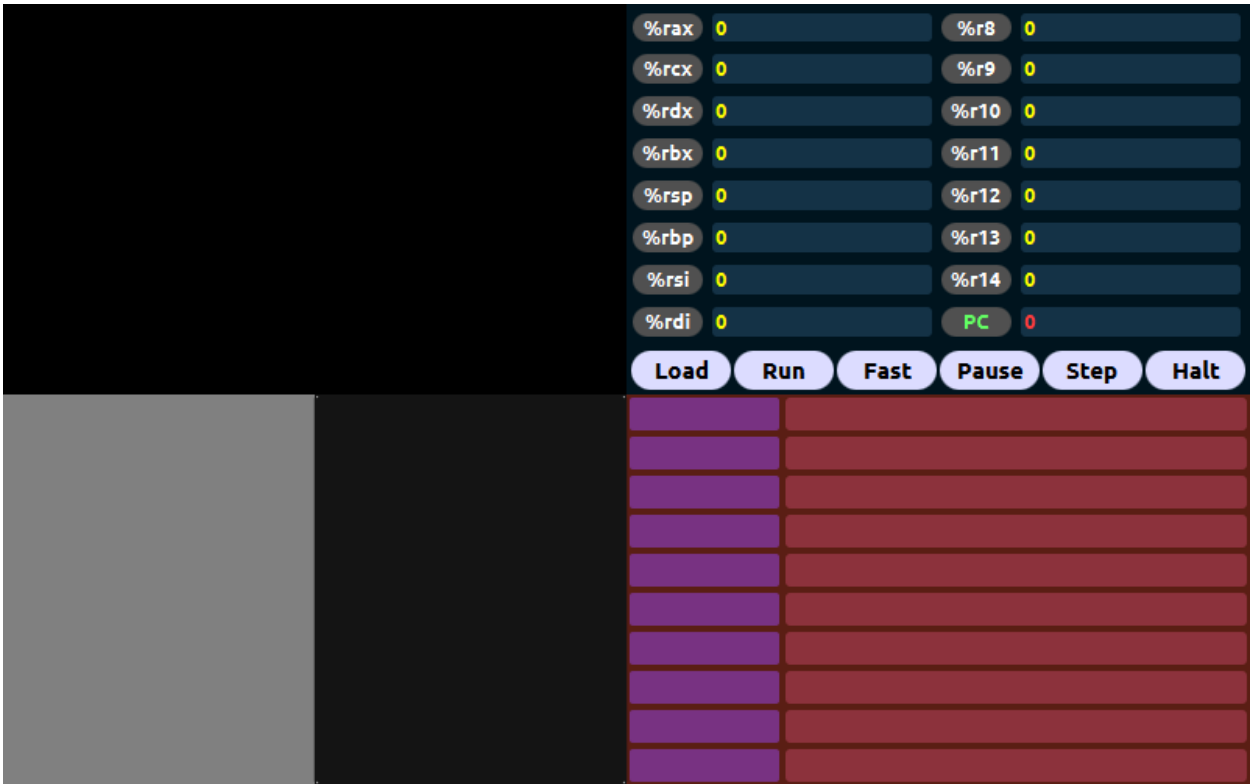
外观设计图:



各部分说明:

- 左上角为**显示屏**，分辨率为200 X 125，默认黑色背景。支持RGBA颜色空间，对Alpha通道尚未提供完全的支持。
- 右上角为**处理器信息**，实时显示15个寄存器保存的数值以及程序计数器中保存的值。其下方6个按钮分别是：
 - 加载（Load）：从文件加载可执行目标文件；
 - 慢速运行（Run）：以约1000Hz的速率运行程序；
 - 快速运行（Fast）：以约15MHz的速率运行程序；
 - 暂停（Pause）：暂停运行；
 - 单步运行（Step）：单步运行程序；
 - 停机（Halt）：终止程序运行，并将程序状态恢复为刚载入时的状态。
- 左下角为**输入和输出窗口**，左侧为输入窗口，目前尚未实现；右侧为输出窗口，显示来自处理器的输出，不能手动修改。
- 右下角为**内存监视器**，最多支持10个条目的检测，每个条目中左侧是地址输入框，可以读取10进制或16进制整数，输入内存地址后右侧文本框会实时显示以输入地址为起始16字节的数据。

效果图:



三、处理器架构

DY64：源自Y86-64 ISA（CS:APP第四章所描述的指令集架构），在其基础上做了一些修改和扩充。

1. 通用寄存器与寻址方式

i. 寄存器列表：

编号	名称	功能
0	%rax	函数返回值
1	%rcx	第四个参数
2	%rdx	第三个参数
3	%rbx	被调用者保存
4	%rsp	栈指针寄存器
5	%rbp	被调用者保存
6	%rsi	第二个参数
7	%rdi	第一个参数
8	%r8	第五个参数
9	%r9	第六个参数
A	%r10	调用者保存
B	%r11	调用者保存
C	%r12	被调用者保存
D	%r13	被调用者保存
E	%r14	被调用者保存

ii. 寻址方式:

R 代表寄存器文件， M 代表存储器。

类型	格式	操作数值	名称
立即数	\$Imm	Imm	立即数寻址
寄存器	r_a	R[r_a]	寄存器寻址
存储器	Imm()	M[Imm]	绝对寻址
存储器	(r_b)	M[R[r_b]]	间接寻址
存储器	Imm(r_b)	M[Imm+R[r_b]]	基址+偏移量寻址

2. 特殊功能寄存器

- 程序计数器 (PC)：记录下一条取指地址
- 指令寄存器 (IR)：记录当前指令
- 条件码寄存器 (CC)：记录当前条件码 (溢出，负数，零，中断等)

- 机器状态 (State)：记录当前机器状态（正常，各种类型异常）

3. 机器状态

机器状态 (State) 有如下类型：

no	标识符	含义
0	VM_OK	正常状态
1	VM_HLT	遇到停机指令
2	VM_INS	非法指令
3	VM_REG	非法寄存器表示
4	VM_ADR	非法内存引用
5	VM_LOG	逻辑错误
6	VM_PC	指令地址错误

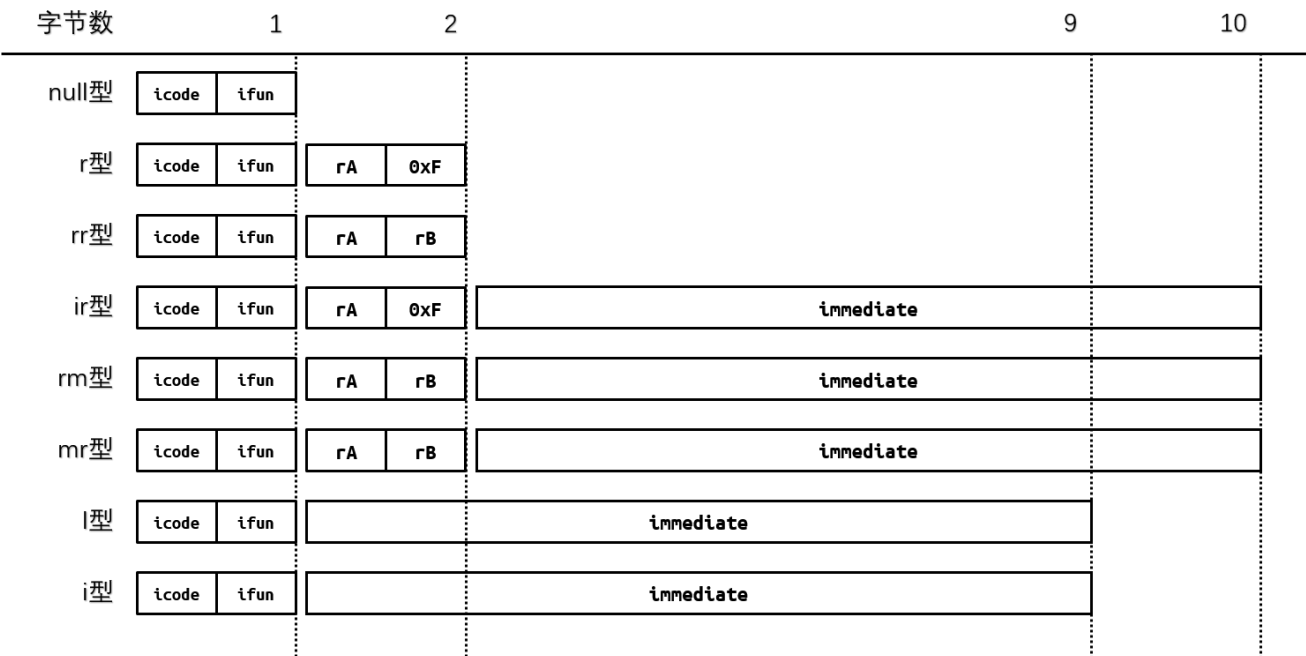
在目前的设计中，当程序运行出现异常后，处理器会直接终止运行，并将异常信息报告给外界。

4. 指令类型及格式

所有指令的第一个字节均为操作码，其中高4位 (icode) 代表指令种类，低4位 (ifun) 代表指令功能。所有包含寄存器ID的字节中，高4位代表第一个寄存器，低4位代表第二个寄存器（0xF 代表不选中寄存器）。立即数均为64位，采用小端法存储。

目前共有7种类型的指令：

1. **null型**：无操作数，长度为一个字节。
2. **r型**：单操作数，操作数源为寄存器，长度为2字节。第2个字节存放寄存器id。
3. **rr型**：双操作数，操作数源均为寄存器，长度为2字节。第2个字节存放寄存器id。
4. **ir型**：双操作数，第一个操作数源为立即数，第二个操作数源为寄存器，长度为10字节。第2个字节存放寄存器id，第3-10字节存放立即数。
5. **rm型**：双操作数，第一个操作数源为寄存器，第二个操作数源为存储器的某个位置，长度为10字节。第2个字节存放寄存器id，第3-10字节存放立即数。
6. **mr型**：双操作数，第一个操作数源为存储器的某个位置，第二个操作数源为立即数，长度为10字节。第2个字节存放寄存器id，第3-10字节存放立即数。
7. **l型**：单操作数，在汇编指令该操作数中是代表指令地址的标签，在机器指令中是指令地址，长度为9字节。第2-9字节存放立即数。编写汇编代码时操作数为标签，在汇编时标签会被翻译成立即数。
8. **i型**：单操作数，操作数为立即数。第2-9字节存放立即数。



5. 取指流程

每次取指时，处理器会先加载mem[PC]位置的第一个字节，根据此字节的内容判断指令类型和长度，再依据判断结果读取相应字节数的指令，将取出的指令存放在ir（指令寄存器）中。处理器执行指令时会根据ir中的数据进行操作。

6. 指令列表

下列符号在指令中分别代表：

- `rA, rB`：寄存器ID
- `R`：寄存器，`R[rA]`表示寄存器rA中的值
- `m`：虚拟存储地址，寻址方式如上所述
- `M`：虚拟存储系统，可看作字节数组，`M[m]`表示地址m处的值
- `I`：立即数
- `Label`：标签，汇编器会将其翻译为立即数

i. 特殊指令

op码	指令名称与格式	功能
00	halt	停机
01	nop	空操作
02	ret	函数返回 <code>%rsp += 8, R[%rsp] -> pc</code>
03	iret	中断返回

ii. 传送指令

op码	指令名称与格式	功能
10	irmov i, rA	将立即数传送到寄存器 I -> R[rA]
20	rmmovq rA, m	将寄存器中64位数据传送到主存 R[rA] -> M[m]
21	rmmovl rA, m	将寄存器中低32位数据传送到主存 R[rA] & 0xffffffff -> M[m]
22	rmmovw rA, m	将寄存器中低16位数据传送到主存 R[rA] & 0xffff -> M[m]
23	rmmovb rA, m	将寄存器中低8位数据传送到主存 R[rA] & 0xff -> M[m]
24	rrmmovq rA, m	将64位数据从主存传送到寄存器 M[m] -> R[rA]
25	rrmmovl rA, m	将32位数据从主存传送到寄存器 M[m]& 0xffffffff -> R[rA]
26	rrmmovw rA, m	将16位数据从主存传送到寄存器 M[m]& 0xffff -> R[rA]
27	rrmmovb rA, m	将8位数据从主存传送到寄存器 M[m]& 0xff -> R[rA]
30	rrmov rA, rB	将数据从寄存器rA传送到寄存器rB R[rA] -> R[rB]
31	cmove	条件传送, 当状态码为相等 (e) 时发生传送 if equal then R[rA] -> R[rB]
32	cmovne	状态码为不相等 (ne) 时发生传送 if not equal then R[rA] -> R[rB]
33	cmovg	状态码为大于 (g) 时发生传送 if greater then R[rA] -> R[rB]
34	cmovge	状态码为大于等于 (g) 时发生传送 if greater or equal then R[rA] -> R[rB]
35	cmovl	状态码为小于 (l) 时发生传送 if less then R[rA] -> R[rB]
36	cmovle	状态码为小于等于 (g) 时发生传送 if less or equal equal then R[rA] -> R[rB]

iii. 双操作数算术与逻辑运算指令

更新条件码

op码	指令名称与格式	功能
40	add rA, rB	$R[rA] + R[rB] \rightarrow R[rA]$
41	sub rA, rB	$R[rA] - R[rB] \rightarrow R[rA]$
42	and rA, rB	$R[rA] \& R[rB] \rightarrow R[rA]$
43	or rA, rB	$R[rA] R[rB] \rightarrow R[rA]$
44	xor rA, rB	$R[rA] \wedge R[rB] \rightarrow R[rA]$
45	sal rA, rB	$R[rA] \ll (R[rB] \& 0x3f) \rightarrow R[rA]$ R[rA] 左移量为R[rB]的低6位
46	sar rA, rB	$R[rA] \gg (R[rB] \& 0x3f) \rightarrow R[rA]$ 算术右移, 填充符号位
47	shr rA, rB	$R[rA] \gg (R[rB] \& 0x3f) \rightarrow R[rA]$ 逻辑右移, 填充0
48	mul rA, rB	$R[rA] * R[rB] \rightarrow R[rA]$
49	idiv rA, rB	$R[rA] / R[rB] \rightarrow R[rA], R[rA] \% R[rB] \rightarrow R[\%rax]$ 有符号数整除, 商存放在R[rA]中, 余数存放在R[%rax]中 (注: 如果 $rA == \%rax$, 余数会被抛弃, 不保存)

iv. 单操作数算术与逻辑运算指令

更新条件码

op码	指令名称与格式	功能
50	not rA	<code>~(R[rA]) -> R[rA]</code>
51	neg rA	<code>-(R[rA]) -> R[rA]</code>
52	inc rA	<code>R[rA]++</code>
53	dec rA	<code>R[rA]--</code>
54	cltq rA	对R[rA]的低32位进行符号拓展 <code>(int64)(int32)R[rA] -> R[rA]</code>
55	cwtq rA	对R[rA]的低16位进行符号拓展 <code>(int64)(int16)R[rA] -> R[rA]</code>
56	cbtq rA	对R[rA]的低8位进行符号拓展 <code>(int64)(int8)R[rA] -> R[rA]</code>
57	cqtl rA	保留R[rA]的低32位，其余位清零 <code>R[rA] &= 0xffffffff</code>
58	cqtw rA	保留R[rA]的低16位，其余位清零 <code>R[rA] &= 0xffff</code>
59	cqtb rA	保留R[rA]的低8位，其余位清零 <code>R[rA] &= 0xff</code>

v. 测试指令

更新条件码

op码	指令名称与格式	功能
60	cmp rA, rB	将R[rA]与R[rB]比较大小 <code>update cc with R[rA] - R[rB]</code>
61	test rA, rB	将(R[rA] & R[rB])和0比较大小 <code>update cc with R[rA] & R[rB]</code>

vi. 跳转指令

op码	指令名称与格式	功能
70	jmp label	无条件跳转到label处 <code>label -> pc</code>
71	je label	比较结果为相等时跳转到label处 <code>if equal then label -> pc</code>
72	jne label	结果为不相等时跳转到label处 <code>if not equal then label -> pc</code>
73	jg label	结果为大于时跳转到label处 <code>if greater then label -> pc</code>
74	jge label	结果为大于等于时跳转到label处 <code>if greater or equal then label -> pc</code>
75	jl label	结果为小于时跳转到label处 <code>if less then label -> pc</code>
76	jle label	结果为小于等于时跳转到label处 <code>if less or equal then label -> pc</code>
77	call label(function name)	函数调用 <code>%rsp -= 8, pc -> R[%rsp], label -> pc</code>

vii. 堆栈操作

op码	指令名称与格式	功能
80	push rA	将R[rA]压进栈中 <code>%rsp -= 8, R[rA] -> R[%rsp]</code>
81	pop rA	弹出栈顶并保存至R[rA] <code>%rsp += 8, R[%rsp-8] -> R[rA]</code>

viii. 中断操作

op码	指令名称与格式	功能
90	int i	触发软中断（i为立即数，中断向量下标）

7. 中断

i. 中断向量

目前的中断向量中只有两个有效地址：

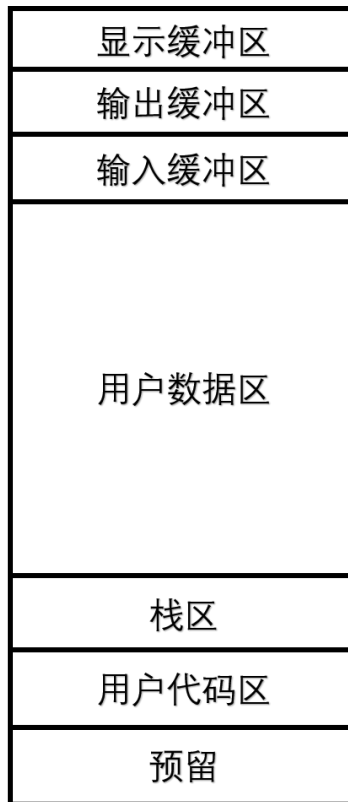
- 地址2：清空输出缓冲区
- 地址3：将显存的内容显示到屏幕上

ii. 中断操作的实现

当前版本的虚拟机没有真实地模拟中断操作，软件触发的陷阱（软中断）和普通指令的运行方式相同，没有设置中断标志位。此外，中断程序使用Java编写，而不是汇编语言，因为当前的指令集无法完备地支持虚拟机的所有操作。

四、虚拟存储系统

地址	分区
0x00000000 ~ 0x00ffffff	预留空间
0x01000000 ~ 0x01ffffff	用户代码区
0x02000000 ~ 0x02ffffff	栈区（读写）
0x03000000 ~ 0x0ffffff	用户数据区（读写）
0x10000000 ~ 0x1007ffff	输入缓冲区（只读）
0x10080000 ~ 0x100ffff	输出缓冲区（只写）
0x10100000 ~ 0x10200000	显示缓冲区（只写）



1. 预留空间

此部分空间为虚拟机预留的存储空间，存放了与IO和系统功能相关的信息，范围是[0x00000000, 0x01000000)。

地址0x0 ~ 0x7存放的64位整型数据存储的是输入缓冲区的大小，用于读入操作的实现（尚未实现）。

地址0x8 ~ 0xf存放的64位整型数据存储的是输出缓冲区的大小，用于输出操作的实现。

地址0x10 ~ 0x13存放的32位整型数据存储的是随机数种子，在程序加载到内存中时会被自动置入，用于伪随机随机函数的实现。

2. 用户代码区

该段区域存放二进制用户代码，范围是[0x01000000, 0x02000000)。

读入的目标文件代码会被放置在以地址0x01000000为起始的连续内存空间中，代码若超出用户代码区的长度则超出部分会被截断。加载完目标文件后，PC会被设置为目标文件中main函数的地址。该区段理论上是只读的，但是目前的实现中还没有对此区域进行保护，若操作不当，可能会使代码被修改。

3. 栈区

该段区域用于存放函数参数、返回地址以及函数的局部变量等数据，范围是[0x02000000, 0x03000000)。

运行时，栈顶向低地址方向扩张，即：栈底为高地址，栈顶为低地址。成功加载目标代码后，栈指针会被初始化为0x03000000（栈区的上边界）。入栈操作会将栈指针的值减去8并存入数据，出栈操作会将栈指针的值加上8并取出数据。在调用函数或函数返回指令过程中，PC值出入栈的操作会自动执行。另外，栈中数据的读写不只局限于栈顶，从栈顶到栈底的任意位置都可随机访问读写。

理论上，应有栈溢出检测机制，但从该虚拟机实现的复杂性上考虑，没有加入该功能。若操作不当致使栈溢出，可能会覆盖用户代码区的内容，导致运行错误。

4. 用户数据区

此区域存放用户数据，范围是[0x03000000, 0x10000000)。

此区域用户可自由支配，用于存放全局变量，局部变量，数组等等。

5. 输入缓冲区

此区域存放来自键盘输入的临时数据，范围是[0x10000000, 0x10080000)。

缓冲区大小的数据（0x0 ~ 0x7）会实时反映缓冲区所含有效数据的字节数。目前输入尚未实现。

6. 输出缓冲区

此区域存放待输出的临时数据，范围是[0x10080000, 0x10100000)。

缓冲区大小的数据（0x8 ~ 0xf）**不会**实时反映缓冲区的有效字节数，但是在触发中断，**调用输出功能之时必须与缓冲区真实大小保持一致**，不然可能会导致输出结果异常。

7. 显示缓冲区

此区域存放显示数据，范围是[0x10100000, 0x10200000)。

该区存放的即是每个像素的颜色信息，每个像素信息占四字节，其中R(Red)、G(Green)、B(Blue)、A(Alpha)各占一字节。当前的设计中，对Alpha通道还无法提供完整支持。由于显示器的分辨率为200 X 125，实际有效的数据区域范围是[0x10100000, 0x101186a0)。访问此存储区时，必须保证数据对齐，即，地址为4的倍数，且访存指令只能为 `writel`。

五、汇编语言

此虚拟机使用的汇编指令完全基于本机的指令系统。

1. 语法单元

本机汇编语言共有5种语法单元：指令标识符、立即数、寄存器标识符、虚拟存储地址、地址标签。

指令标识符，唯一确定地标识指令，指令列表中所有指令的名称都是指令标识符。

立即数，用于立即数寻址，以字符 `$` 开头，支持十进制或十六进制表示（十六进制需加上 `0x` 前缀，字母大小写均可）。

寄存器标识符，唯一确定地标识寄存器文件，用于寄存器寻址。寄存器列表中的所有寄存器名称都是寄存器标识符。

虚拟存储地址，唯一确定地标识虚拟存储地址，用于基址寻址、偏移量寻址和基址+偏移量寻址。格式为 `I(rB)`，其中 `I` 为立即数，不需要以 `$` 开头，支持十进制或十六进制表示（十六进制需加上 `0x` 前缀，字母大小写均可）；同时，`I` 也可以是地址标签，汇编器会将地址标签替换为立即数；`rB` 为寄存器标识符，表示基址寄存器。所表示的地址为 `I+R[rB]`，`R[rB]` 代表寄存器 `rB` 存储的数值。立即数和寄存器标识符均不省略时，寻址方式为基址+偏移量寻址；省略立即数时，寻址方式变为基址寻址；省略寄存器标识符时，寻址方式变为绝对寻址；两者不允许同时省略。

地址标签，表示下一条指令的起始地址，方便跳转指令和虚拟存储地址的编写。地址标签分为两类，一类是目的标签，另一类是源标签。目的标签表示的是跳转语句将要跳转的目的地址，标签末尾要写上 `:`；源标签会在汇编过程中被替换为目的地址。函数名也是地址标签。汇编器提供了几个地址标签：

```
1 // 数据区段起始位置
2 data_sec_pos 0x03000000
3
4 // 存放输入缓冲区大小的位置
5 in_buf_size 0x00000000
6
7 // 输入缓冲区起始位置
8 in_buf_pos 0x10000000
9
10 // 存放输出缓冲区大小的位置
11 out_buf_size 0x00000008
12
13 // 输出缓冲区起始位置
14 out_buf_pos 0x10080000
15
16 // 显示缓冲区起始位置
17 disp_buf_pos 0x10100000
18
19 // 随机数种子的位置
20 random_seed_pos 0x00000010
```

2. 语法要求

汇编器的语法要求有：

- 每条指令占一行，地址标签占一行；
- 字符 `;` 为单行注释符，从 `;` 字符起始到行末都为注释内容；
- - 无操作数指令格式为 `instruction`；
 - 单操作数指令格式为 `instruction <operand>`，指令标识符与操作数之间必须间隔一个空格；
 - 双操作数指令格式为 `instruction <operand 1>, <operand 2>`，指令标识符于第一个操作数之间必须间隔一个空格，操作数1个操作数2之间必须间隔一个 `,` 和一个空格；
 - 目的标签末尾应加上 `:`，如 `main:`，`.L3:` 等；源标签末尾不加 `:`；源标签必须有唯一的目的标签相对应，目的标签则可以对应任意多个源标签；
- 每个汇编程序中必须有一个 `main` 函数，虚拟机将 `main:` 当作入口执行指令。

3. 函数调用

i. 寄存器内容的保存

函数调用过程中，调用者寄存器的值应当被保存。从设计上说，一部分寄存器应由调用者保存，这些寄存器是 `%rax, %rcx, %rdx, %rsi, %rdi, %r8, %r9, %r10, %r11`，另一部分则由被调用者保存，这些寄存器是 `%rbx, %rbp, %r12, %r13, %r14`。`%rsp` 即栈指针寄存器，应由调用者和被调用者共同维护。

ii. 函数参数

函数的前六个参数顺序是 `%rdi,%rsi,%rdx,%rcx,%r8,%r9`，第七、第八乃至更多的参数则由栈来传递。所有参数应当在调用函数之前就存入相应的寄存器或栈中。超出6个参数的部分，应当以逆序入栈，即，最后一个入栈的应是第七个参数。在调用函数时，返回地址会被压入栈中，因此对于被调用者而言第七个参数地址是 `8(%rsp)`，第八个参数地址是 `16(%rsp)`。

iii. 返回值

函数返回值保存在 `%rax` 中，若有多余一个返回值，可以利用指针传递返回值。

4. 库函数

汇编器提供了一些库函数，实现了打印数据、绘图、随机数等功能。以下列举了几个函数：

```
1 println // 接收一个64位整型参数，将其当作有符号数打印并换行
2 draw    // 接受参数x, y, v, 在显示器(x, y)位置绘制颜色为v的像素点
3 repaint // 刷新显示屏，打印显示缓冲区的内容
4 random  // 随机数函数返回[0, 48271)之间的一个随机数
```

六、虚拟机实现框架

1. 程序结构

本程序使用java实现，主要包含以下代码文件：

- Main.java：完成虚拟机逻辑层面的组装与运行；
- Processor.java：实现处理器功能；
- AbstractFile.java：接口，为文件设备提供抽象接口；
- ReadableFile.java：接口，为可读设备提供抽象接口；
- WritableFile.java：接口，为可写设备提供抽象接口；
- IOBridge.java：IO桥接器，实现ReadableFile和WritableFile接口，连接一切文件设备；
- Memory.java：主存，实现ReadableFile和WritableFile接口；
- Keyboard.java：模拟键盘设备，实现ReadableFile接口；
- TextOutput.java：文本输出设备，实现WritableFile接口；
- Display.java：模拟显示设备，实现WritableFile接口。

注：下列代码只是源代码的抽象示意，许多具体的实现细节没有在其中体现。

2. 虚拟存储系统程序框架

i. AbstractFile接口

定义了getMaxSize(), isReadable(), isWritable()抽象方法。

```
1 interface AbstractFile {
2     methods:
3         getMaxSize()
4         isReadable()
5         isWritable()
6 }
```

ii. ReadableFile接口

继承AbstractFile接口，定义了readq(), readl(), readw(), readb()四个读方法。

```
1 interface ReadableFile extends AbstractFile {
2     methods:
3         readq()
4         readl()
5         readw()
6         readb()
7 }
```

iii. WritableFile接口

继承AbstractFile接口，定义了writeq(), writel(), writew(), writeb()四个写方法。

```
1 interface WritableFile extends AbstractFile {
2     methods:
3         writeq()
4         writel()
5         writew()
6         writeb()
7 }
```

iv. Memory类

模拟主存，实现ReadableFile和WritableFile接口，用一维字节数组模拟主存存储空间。

```
1 class Memory implements ReadableFile, WritableFile {
2     variables:
3         byte ram[MAX_SIZE] //随机访问存储器
4
5     methods:
6         @Override { // 继承自接口的方法
7             getMaxSize()
8             isReadable()
9             isWritable()
```



```

10         readq()
11         readl()
12         readw()
13         readb()
14         writeq()
15         writel()
16         writew()
17         writeb()
18     }
19
20     class MemoryPane extends Pane {
21         monitors[10]    //10个内存监视器条目
22     }
23 }

```

v. Keyboard类

模拟键盘输入设备，实现ReadableFile接口，用一维字节数组模拟输入缓冲区。

```

1  class Keyboard implements ReadableFile {
2  variables:
3      byte ram[MAX_SIZE]  //随机访问存储器
4
5  methods:
6      @Override {        // 继承自接口的方法
7          getMaxSize()
8          isReadable()
9          isWritable()
10         readq()
11         readl()
12         readw()
13         readb()
14     }
15
16     class KeyboardPane extends Pane {
17     }
18 }

```

vi. TextOutput类

模拟文本输出设备，实现WritableFile接口，用一维字节数组模拟输出缓冲区。

```

1  class TextOutput implements WritableFile {
2  variables:
3      byte ram[MAX_SIZE]  //随机访问存储器
4
5  methods:
6      @Override {        // 继承自接口的方法

```

```

7      getMaxSize()
8      isReadable()
9      isWritable()
10     writeq()
11     writel()
12     writew()
13     writeb()
14 }
15
16 class TextOutputPane extends Pane {
17     TextArea textArea    //输出文本框
18
19     methods:
20         print()          //输出缓冲区内容
21     }
22 }

```

vii. Display类

模拟显示输出设备，实现WritableFile接口，用一维字节数组模拟显示缓冲区。

```

1  class Display implements WritableFile {
2      variables:
3          byte ram[MAX_SIZE] //随机访问存储器
4
5      methods:
6          @Override {        // 继承自接口的方法
7              getMaxSize()
8              isReadable()
9              isWritable()
10             writeq()
11             writel()
12             writew()
13             writeb()
14         }
15
16         class DisplayCanvas extends Canvas {
17             methods:
18                 paint()      //绘图
19         }
20     }

```

viii. IOBridge类

模拟IO桥接器，用来链接CPU与文件设备，实现了ReadableFile和WritableFile接口。以下是IOBridge的简易框架：

```

1  class IOBridge implements ReadableFile, WritableFile {

```

```

2  variables:
3      Memory memory
4      Keyboard keyboard
5      TextOutput textOutput
6      Display display
7
8      Processor processor
9
10 methods:
11     loadObject()          // 加载目标文件的方法
12
13     @Override {          // 继承自接口的方法
14         getMaxSize()
15         isReadable()
16         isWritable()
17         readq()
18         readl()
19         readw()
20         readb()
21         writeq()
22         writel()
23         writew()
24         writeb()
25     }
26 }

```

3. 处理器程序框架

处理器包含15个通用寄存器，和专用寄存器：PC、IR、CC、State（如上所述）。考虑到设计复杂度和拟真程度等因素，该处理器使用非流水线化的结构以使运行流程变得更加简单、清晰。

其运行流程是：每一指令周期开始时，先从PC所给地指出取指，保存到IR中，然后运行指令，期间可能会更改CC中的值，或者读写内存和通用寄存器，最后更新PC的值。如运行过程中出现异常，处理器会更新State寄存器的内容并抛出异常。如果运行正常则进入下一指令周期。

以下是Processor的简易框架：

```

1  class Processor {
2      variables:
3          IOBridge ioBridge
4
5          long regs[15] //寄存器用一维64位整型数组模拟
6          int pc //程序计数器
7          byte cc //条件寄存器
8          byte state //机器状态
9
10         byte ir[10] //指令寄存器用字节数组模拟
11
12     methods:
13         void fetch() //取指 + 计算下一PC地址
14         // PC保存的值为目标文件中指令的绝对地址，在取指过程中会将PC映射到真实地址

```

```

15     void exec() //译码 + 执行 + 访存 + 更新PC
16
17     class ProcessorPane extends Pane {
18         Button load      // 加载程序
19         Button run       // 运行程序
20         Button fast      // 快速运行程序
21         Button pause     // 暂停
22         Button step      // 单步运行
23         Button halt      // 停机并复原初始状态
24
25         registers & pc messages //寄存器和PC的值
26     }
27 }

```

4. 主类框架

```

1  class Main extends Application {
2  variables:
3      Processor processor
4      IOBridge ioBridge
5      Memory memory
6      Keyboard keyboard
7      TextOutput textOutput
8      Display display
9
10 methods:
11     void start()
12 }
13
14 Main.start() {
15     //链接文件设备到处理器
16     link memory, keyboard, textOutput, display to ioBridge
17     link ioBridge to processor
18
19     init a scene
20     add memoryPane, KeyboardPane, TextOutputPane,
21     DisplayCanvas and ProcessPane to the scene
22
23     add the scene to the stage
24     show the stage
25 }

```

七、测试

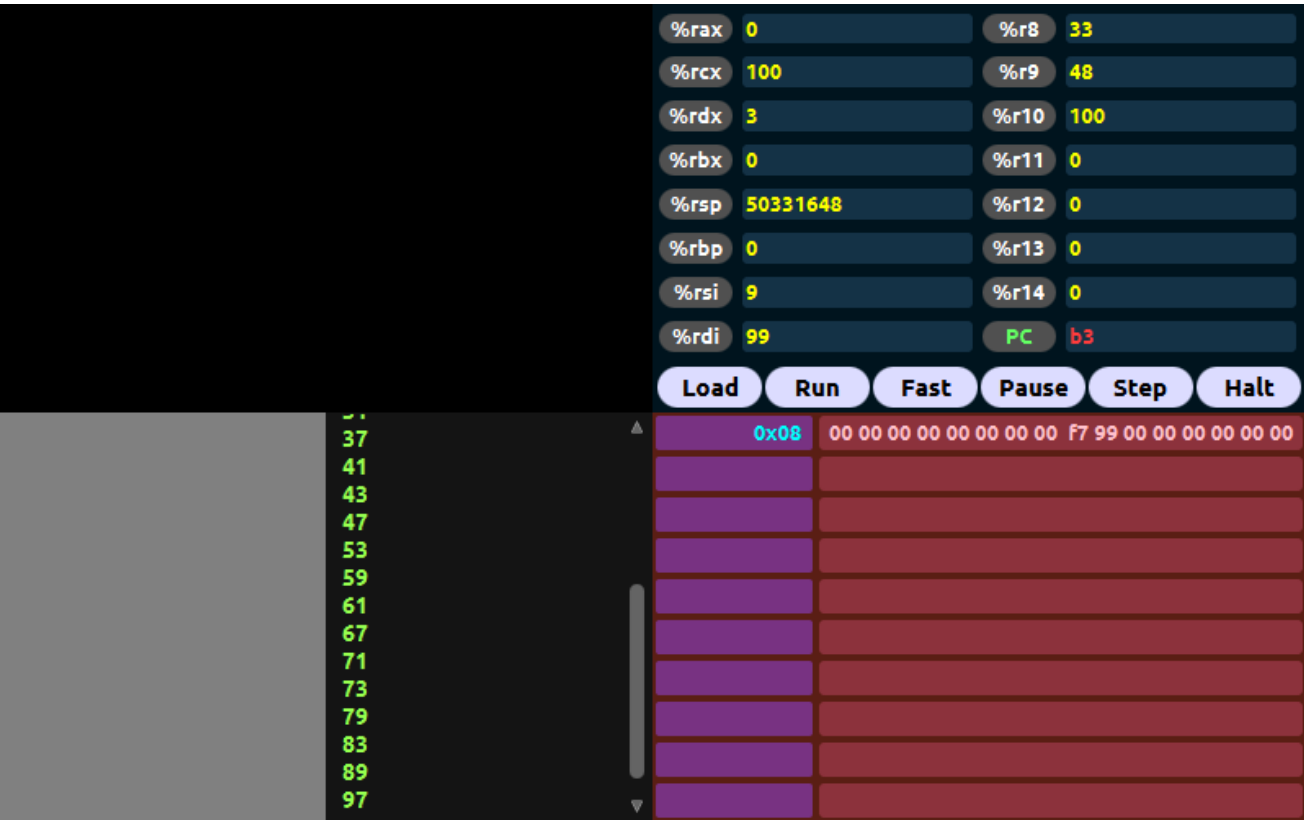
为了检验虚拟机的正确性和性能，我编写了一些汇编程序进行测试

1. 求素数程序

该程序打印100以内的素数

```
1 ; program prime
2 ; print all prime numbers below 100
3
4 ; function prime(n) returns boolean
5 prime:
6     irmov $2, %rdx ; i = 2
7     irmov $1, %rax
8     cmp %rdi, %rax ; n <= 1
9     jg .L3
10    xor %rax, %rax ; return 0
11    ret
12 .L3:
13    cmp %rdx, %rdi
14    jge .L4
15    rrmov %rdx, %rsi
16    mul %rsi, %rdx
17    cmp %rsi, %rdi ; i * i > n ?
18    jg .L4
19    rrmov %rdi, %r8
20    idiv %r8, %rdx ; n / i
21    test %rax, %rax ; test n % i
22    je .L4
23    irmov $1, %rax
24    inc %rdx ; i++
25    jmp .L3
26 .L4:
27    ret
28
29 ; function main
30 main:
31    irmov $2, %rcx ; i = 0
32    irmov $100, %r10 ; limit = 0
33 .L1:
34    rrmov %rcx, %rdi
35    call prime ; prime(i)
36    test %rax, %rax
37    je .L2
38    rrmov %rcx, %rdi
39    push %rcx
40    push %r10
41    call println
42    pop %r10
43    pop %rcx
44 .L2:
45    inc %rcx
46    cmp %rcx, %r10
47    jl .L1
48    halt
```

运行结果：



2. 生命游戏

该程序模拟康威生命游戏的运行。生命游戏是一种二维元胞自动机，具体规则可参考[维基百科-康威生命游戏](#)。其中细胞的颜色会随周期型变化。

```
1 ; function dx(x, d)
2 dx:
3     push %rbx
4     add %rdi, %rsi
5     rrmov %rdi, %rax
6     rrmov %rdi, %rbx
7     irmov $200, %rsi
8     add %rdi, %rsi
9     sub %rbx, %rsi
10    cmp %rax, %rsi
11    cmovge %rbx, %rax
12    test %rax, %rax
13    cmovl %rdi, %rax
14    pop %rbx
15    ret
16
17 ; function dy(y, d)
18 dy:
19     push %rbx
20     add %rdi, %rsi
21     rrmov %rdi, %rax
22     rrmov %rdi, %rbx
```

```

23     irmov $125, %rsi
24     add %rdi, %rsi
25     sub %rbx, %rsi
26     cmp %rax, %rsi
27     cmovge %rbx, %rax
28     test %rax, %rax
29     cmovl %rdi, %rax
30     pop %rbx
31     ret
32
33 ; function init
34 init:
35 ; initialize the cell matrix
36     push %rbx
37     irmov $25000, %rbx
38     irmov $7, %r8
39     irmov $3, %r9
40     irmov $1, %r10
41     rrmov %rbx, %rdx
42     xor %rcx, %rcx
43 .L2:
44     call random
45     and %rax, %r8
46     xor %rdi, %rdi
47     cmp %rax, %r9
48     cmovl %r10, %rdi
49     rmmovb %rdi, data_sec_pos(%rdx)
50     inc %rcx
51     inc %rdx
52     cmp %rcx, %rbx
53     jl .L2
54     pop %rbx
55     ret
56
57 ; function next
58 next:
59     push %rbx
60     push %rbp
61     push %r12
62     push %r13
63     push %r14
64     irmov $25000, %rbx
65     irmov $50000, %rbp
66
67     xor %rcx, %rcx
68 ; .L41:
69 ;     rrmov %rcx, %rdx
70 ;     add %rdx, %rbx
71 ;     mrmovb data_sec_pos(%rdx), %rax
72 ;     add %rdx, %rbx
73 ;     rmmovb %rax, data_sec_pos(%rdx)
74 ;     inc %rcx
75 ;     cmp %rcx, %rbx

```

```

76 ;    jl .L41
77
78 ; calculate
79     irmov $200, %r8
80     irmov $125, %r9
81     irmov $1, %r12
82     irmov $-1, %r11
83     xor %rcx, %rcx
84     xor %r10, %r10
85     rrmov %rbp, %rdx
86 .L3:
87     ; store x + 1
88     rrmov %rcx, %rdi
89     rrmov %r12, %rsi
90     call dx
91     push %rax ; push x + 1
92     ; store x - 1
93     rrmov %rcx, %rdi
94     rrmov %r11, %rsi
95     call dx
96     push %rax ; push x - 1
97     push %rcx ; push x
98     ; store y + 1
99     rrmov %r10, %rdi
100    rrmov %r12, %rsi
101    call dy
102    push %rax ; push y + 1
103    ; store y - 1
104    rrmov %r10, %rdi
105    rrmov %r11, %rsi
106    call dy
107    push %rax ; push y - 1
108    push %r10 ; push y
109    ; count
110    xor %r14, %r14 ; count = 0
111        ; x-1, y-1
112    mrmovq 32(%rsp), %rdi
113    mrmovq 8(%rsp), %rsi
114    mul %rdi, %r9
115    add %rdi, %rsi
116    add %rdi, %rbx
117    mrmovb data_sec_pos(%rdi), %r13
118    add %r14, %r13 ; count++ if alive
119        ; x-1, y
120    mrmovq 32(%rsp), %rdi
121    mrmovq (%rsp), %rsi
122    mul %rdi, %r9
123    add %rdi, %rsi
124    add %rdi, %rbx
125    mrmovb data_sec_pos(%rdi), %r13
126    add %r14, %r13 ; count++ if alive
127        ; x-1, y+1
128    mrmovq 32(%rsp), %rdi

```



```
129     mrmovq 16(%rsp), %rsi
130     mul %rdi, %r9
131     add %rdi, %rsi
132     add %rdi, %rbx
133     mrmovb data_sec_pos(%rdi), %r13
134     add %r14, %r13 ; count++ if alive
135         ; x, y-1
136     mrmovq 24(%rsp), %rdi
137     mrmovq 8(%rsp), %rsi
138     mul %rdi, %r9
139     add %rdi, %rsi
140     add %rdi, %rbx
141     mrmovb data_sec_pos(%rdi), %r13
142     add %r14, %r13 ; count++ if alive
143         ; x, y+1
144     mrmovq 24(%rsp), %rdi
145     mrmovq 16(%rsp), %rsi
146     mul %rdi, %r9
147     add %rdi, %rsi
148     add %rdi, %rbx
149     mrmovb data_sec_pos(%rdi), %r13
150     add %r14, %r13 ; count++ if alive
151         ; x+1, y-1
152     mrmovq 40(%rsp), %rdi
153     mrmovq 8(%rsp), %rsi
154     mul %rdi, %r9
155     add %rdi, %rsi
156     add %rdi, %rbx
157     mrmovb data_sec_pos(%rdi), %r13
158     add %r14, %r13 ; count++ if alive
159         ; x+1, y
160     mrmovq 40(%rsp), %rdi
161     mrmovq (%rsp), %rsi
162     mul %rdi, %r9
163     add %rdi, %rsi
164     add %rdi, %rbx
165     mrmovb data_sec_pos(%rdi), %r13
166     add %r14, %r13 ; count++ if alive
167         ; x+1, y+1
168     mrmovq 40(%rsp), %rdi
169     mrmovq 16(%rsp), %rsi
170     mul %rdi, %r9
171     add %rdi, %rsi
172     add %rdi, %rbx
173     mrmovb data_sec_pos(%rdi), %r13
174     add %r14, %r13 ; count++ if alive
175     ; restore rsp
176     irmov $48, %rax
177     add %rsp, %rax
178     push %r8
179     push %r9
180     irmov $2, %r8
181     irmov $3, %r9
```

```

182     cmp %r14, %r9
183     jg .L6
184     cmp %r14, %r8
185     jl .L6
186     jg .L7
187     sub %rdx, %rbx
188     mrmovb data_sec_pos(%rdx), %rax
189     add %rdx, %rbx
190     test %rax, %rax
191     jne .L7
192 .L6:
193     xor %rax, %rax
194     jmp .L8
195 .L7:
196     irmov $1, %rax
197 .L8:
198     rmmovb %rax, data_sec_pos(%rdx)
199     pop %r9
200     pop %r8
201     ; increment
202     inc %rdx
203     inc %r10
204     cmp %r10, %r9
205     jl .L3
206     xor %r10, %r10
207     inc %rcx
208     cmp %rcx, %r8
209     jl .L3
210     ; copy
211     xor %rcx, %rcx
212 .L4:
213     rrmov %rcx, %rdx
214     add %rdx, %rbp
215     mrmovb data_sec_pos(%rdx), %rax
216     sub %rdx, %rbx
217     rmmovb %rax, data_sec_pos(%rdx)
218     inc %rcx
219     cmp %rcx, %rbx
220     jl .L4
221     pop %r14
222     pop %r13
223     pop %r12
224     pop %rbp
225     pop %rbx
226     ret
227 ; end function next
228
229 ; function paint(v) v:color
230 paint:
231     push %rbx
232     irmov $25000, %rbx
233     irmov $0x000000ff, %r10
234
235     rrmov %rdi, %r9

```

```

235     rrmov %rbx, %rax    ; address
236     xor %rcx, %rcx     ; pos = 0
237 .L1:
238     mrmovb data_sec_pos(%rax), %r8
239     rrmov %r9, %rsi
240     test %r8, %r8
241     cmovl %r10, %rsi
242     rrmov %rcx, %rdi
243     call _draw
244     inc %rcx             ; pos++
245     inc %rax             ; address++
246     cmp %rcx, %rbx
247     jl .L1
248     pop %rbx
249     call repaint
250     ret
251
252 ; function color: a scheme of changing color
253 color:
254     irmov $0xff000000, %r8
255     irmov $0x00ff0000, %r9
256     irmov $0x0000ff00, %r10
257     rrmov %rdi, %rsi
258     and %rsi, %r8
259     cmp %rsi, %r8
260     je .L21              ; (255,x,x)
261     rrmov %rdi, %rsi
262     and %rsi, %r9
263     cmp %rsi, %r9
264     je .L22              ; (x,255,x)
265     rrmov %rdi, %rsi
266     and %rsi, %r10
267     cmp %rsi, %r10
268     je .L23              ; (x,x,255)
269 .L21:
270     rrmov %rdi, %rsi
271     and %rsi, %r9
272     cmp %rsi, %r9
273     je .L32              ; (255,255,0)
274     rrmov %rdi, %rsi
275     test %rsi, %r10
276     je .L24              ; (255,x,0)
277 .L31:                    ; reduce blue
278     irmov $0x00000500, %rsi
279     sub %rdi, %rsi
280     jmp .L27
281 .L22:
282     rrmov %rdi, %rsi
283     and %rsi, %r10
284     cmp %rsi, %r10
285     je .L33              ; (0,255,255)
286     rrmov %rdi, %rsi
287     test %rsi, %r8

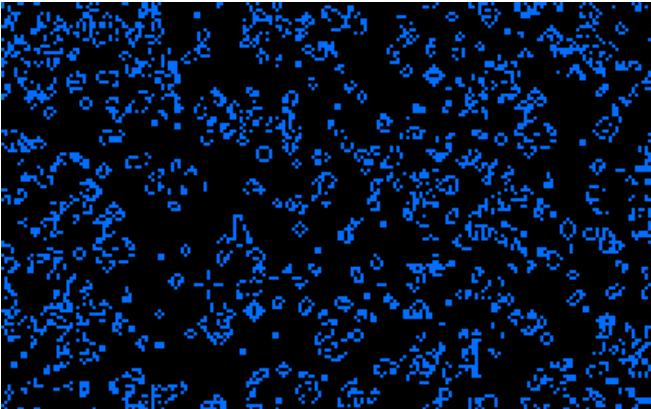
```

```

288     je .L25                                ; (0,255,x)
289 .L32:                                     ; reduce red
290     irmov $0x05000000, %rsi
291     sub %rdi, %rsi
292     jmp .L27
293 .L23:
294     rrmov %rdi, %rsi
295     and %rsi, %r8
296     cmp %rsi, %r8
297     je .L31                                ; (255,0,255)
298     rrmov %rdi, %rsi
299     test %rsi, %r9
300     je .L26                                ; (x,0,255)
301 .L33:                                     ; reduce green
302     irmov $0x00050000, %rsi
303     sub %rdi, %rsi
304     jmp .L27
305 .L24:                                     ; add green
306     irmov $0x00050000, %rsi
307     add %rdi, %rsi
308     jmp .L27
309 .L25:                                     ; add blue
310     irmov $0x00000500, %rsi
311     add %rdi, %rsi
312     jmp .L27
313 .L26:                                     ; add red
314     irmov $0x05000000, %rsi
315     add %rdi, %rsi
316 .L27:
317     rrmov %rdi, %rax
318     ret
319
320 ; function main
321 main:
322     call init
323     irmov $0x00ffffff, %rdi
324     push %rdi
325     call paint
326 .L0:
327     call next
328     pop %rdi
329     call color
330     push %rax
331     rrmov %rax, %rdi
332     call paint
333     jmp .L0
334     halt

```

测试结果：



%rax123

%rcx24

%rdx53124

%rbx25000

%rsp50331544

%rbp50000

%rsi123

%rdi3248

%r8200

%r9125

%r10124

%r11-1

%r121

%r130

%r141

PC2a5

Load

Run

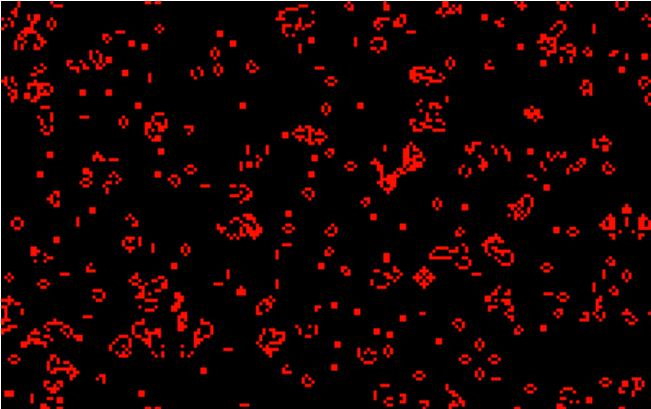
Fast

Pause

Step

Halt

0x08	00 00 00 00 00 00 00 00 a2 09 00 00 00 00 00 00
50331632	99 05 00 00 00 00 00 00 ff ff 6e 00 00 00 00 00
50331544	7c 00 00 00 00 00 00 00 7b 00 00 00 00 00 00 00
0x300b000	01 01 00 01 01 00 01 00 00 00 00 00 00 00 01 01
0x300d000	00 00 00 01 00 01 01 00 00 00 00 00 00 01 00 00



%rax123

%rcx162

%rdx70374

%rbx25000

%rsp50331560

%rbp50000

%rsi125

%rdi248

%r8200

%r9125

%r10124

%r11-1

%r121

%r130

%r140

PC1c9

Load

Run

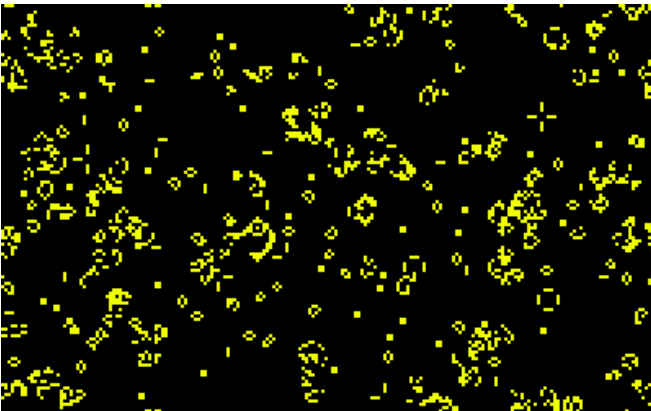
Fast

Pause

Step

Halt

0x08	00 00 00 00 00 00 00 00 a2 09 00 00 00 00 00 00
50331632	99 05 00 00 00 00 00 00 ff 00 0f ff 00 00 00 00
50331544	a8 61 00 00 00 00 00 00 c9 01 00 00 00 00 00 00
0x300b000	00 00 00 01 00 00 00 00 00 01 00 00 00 00 00 00
0x300d000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00



%rax78

%rcx194

%rdx74329

%rbx25000

%rsp50331544

%rbp50000

%rsi79

%rdi49204

%r8200

%r9125

%r1079

%r11-1

%r121

%r130

%r140

PC219

Load

Run

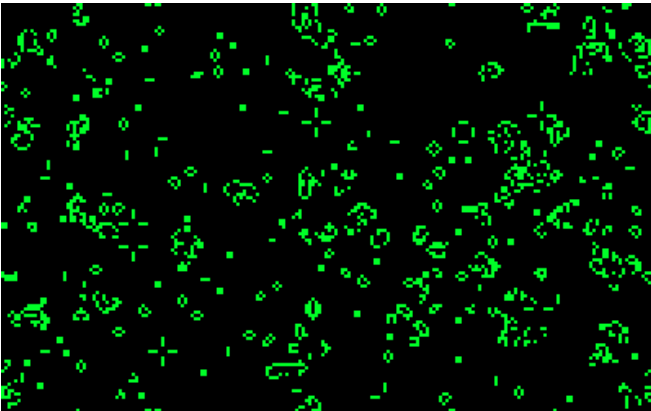
Fast

Pause

Step

Halt

0x08	00 00 00 00 00 00 00 00 a2 09 00 00 00 00 00 00
50331632	99 05 00 00 00 00 00 00 ff 00 ff f5 00 00 00 00
50331544	4f 00 00 00 00 00 00 00 4e 00 00 00 00 00 00 00
0x300b000	00 00 00 01 01 01 00 01 00 01 01 00 00 00 01 01
0x300d000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00



%rax0

%rcx22268

%rdx72268

%rbx25000

%rsp50331592

%rbp50000

%rsi0

%rdi25000

%r8200

%r9125

%r100

%r11-1

%r121

%r130

%r140

PC3ac

Load

Run

Fast

Pause

Step

Halt

0x08	00 00 00 00 00 00 00 00 a2 09 00 00 00 00 00 00
50331632	99 05 00 00 00 00 00 00 ff 23 ff 00 00 00 00 00
50331544	7c 00 00 00 00 00 00 00 7b 00 00 00 00 00 00 00
0x300b000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x300d000	00 00 00 00 00 00 01 01 00 00 00 00 00 00 00 00

八、问题与后续计划

1. 总结

本次实验中，我基本完成了自己预期的设计效果，完成了处理器的正常功能，虚拟存储系统的设计和实现。此外、完成了重要的文本输出和图形显示部分，尤其是图形显示部分，该部分的实现使得本机可以运行较为复杂的图形程序，产生更绚丽的显示效果。

2. 当前问题

当前的设计中仍存在一些缺陷：

1. 当前的虚拟机未完成输入部件的制作；
2. 只支持64位整数操作，虽然提供了一些不同长度数据转换和传送的指令，仍然无法很好的适应32位或更低位程序的运行；
3. 缺乏浮点数操作指令，无法模拟实数运算；
4. 输入与输出缓冲区并不是真正意义上在内存中存在的缓冲区，而是相应设备中的存储区，不利于可能出现的复杂操作的实现；
5. 目前的设计中没有真实地模拟中断操作，在处理软中断时采用的策略是将其当作普通指令去运行，而没有设置中断标志；
6. 异常检测机制和应对措施不够健全。

3. 后续计划

如果未来时间充裕，可能会有一下改进计划：

1. 完成输入部件的开发与测试；
2. 在内存中开辟真正的输入和输出缓冲区，并提供相应的库函数；
3. 真实模拟终端操作，在处理器中设置中断标志位；
4. 完善异常检测机制。

附件

文件结构(IntelliJ IDEA项目)

```
1   DyVm
2   |-- .idea           工程信息
3   |-- asm            汇编代码
4   |-- dyvm-v2        汇编程序
5   |-- hex            汇编程序输出的目标代码
6   |-- info           文本信息
7   |-- out            编译输出
8       |-- production
9           |-- Main.class
10          |-- ...
11 |-- src             源代码
12     |-- Main.java
13     |-- Processor.java
14     |-- AbstractFile.java
15     |-- ReadableFile.java
16     |-- WritableFile.java
17     |-- IOBridge.java
18     |-- Memory.java
```

```

19 | -- Keyboard.java
20 | -- TextOutput.java
21 | -- Display.java
22 | -- text          存储一些文本文件
23 | -- assemble.bat   调用汇编器的批处理程序
24 | -- DyVm.iml
25 | -- README.md
26 | -- stylesheet.css  虚拟机界面的样式表

```

Main.java

```

1  import javafx.application.Application;
2  import javafx.scene.Group;
3  import javafx.scene.Scene;
4  import javafx.stage.Stage;
5
6  import java.io.File;
7
8  public class Main extends Application {
9
10     private final int DisplayWidth = 400;
11     private final int DisplayHeight = 250;
12     private final int Width = 800;
13     private final int Height = 500;
14
15     private Processor processor = new Processor();
16     private IOBridge ioBridge = new IOBridge();
17     private Memory memory = new Memory();
18     private Keyboard keyboard = new Keyboard();
19     private TextOutput textOutput = new TextOutput();
20     private Display display = new Display(DisplayWidth, DisplayHeight);
21
22     public Main() {
23         processor.setIOBridge(ioBridge);
24         memory.setIOBridge(ioBridge);
25         keyboard.setIOBridge(ioBridge);
26         textOutput.setIOBridge(ioBridge);
27         display.setIOBridge(ioBridge);
28     }
29
30     @Override
31     public void start(Stage primaryStage) {
32         // initialize
33         Group root = new Group();
34
35         Processor.ProcessorPane processorPane = processor.getPane();
36         Memory.MemoryPane memoryPane = memory.getPane();
37         Keyboard.KeyboardPane keyboardPane = keyboard.getPane();
38         TextOutput.TextOutputPane textOutputPane = textOutput.getPane();
39
40         Display.DisplayCanvas displayCanvas = display.getCanvas();

```



```

40
41     root.getChildren().addAll(
42         processorPane,
43         memoryPane,
44         keyboardPane,
45         textOutputPane,
46         displayCanvas
47     );
48
49     displayCanvas.setWidth(DisplayWidth);
50     displayCanvas.setHeight(DisplayHeight);
51     displayCanvas.setLayoutX(0);
52     displayCanvas.setLayoutY(0);
53
54     int w = DisplayWidth / 2;
55     keyboardPane.setPrefSize(w, Height - DisplayHeight);
56     keyboardPane.setLayoutX(0);
57     keyboardPane.setLayoutY(DisplayHeight);
58
59     textOutputPane.setPrefSize(w, Height - DisplayHeight);
60     textOutputPane.setLayoutX(w);
61     textOutputPane.setLayoutY(DisplayHeight);
62
63     processorPane.setPrefSize(Width - DisplayWidth, DisplayHeight);
64     processorPane.setLayoutX(DisplayWidth);
65     processorPane.setLayoutY(0);
66
67     memoryPane.setPrefSize(Width - DisplayWidth, Height - DisplayHeight);
68     memoryPane.setLayoutX(DisplayWidth);
69     memoryPane.setLayoutY(DisplayHeight);
70
71     processorPane.configureLoadButton(primaryStage);
72
73     Scene scene = new Scene(root, Width, Height);
74     // add css file
75     File file = new File("stylesheet.css");
76     scene.getStylesheets().add("file:/// " + file.getAbsolutePath().replace('\\', '/'));
77
78     primaryStage.setTitle("DyVM");
79     primaryStage.setScene(scene);
80     primaryStage.show();
81 }
82
83 }

```

Processor.java

```

1  import javafx.animation.KeyFrame;
2  import javafx.animation.Timeline;
3  import javafx.scene.control.Button;

```

```
4 import javafx.scene.control.Label;
5 import javafx.scene.layout.BorderPane;
6 import javafx.scene.layout.GridPane;
7 import javafx.scene.layout.HBox;
8 import javafx.stage.FileChooser;
9 import javafx.stage.Stage;
10 import javafx.util.Duration;
11
12 import java.io.File;
13 import java.io.FileNotFoundException;
14 import java.util.Scanner;
15
16 public class Processor {
17
18     // normal speed
19     private final static Duration duration1 = Duration.millis(1);
20     // faster speed
21     private final static Duration duration2 = Duration.millis(1);
22
23     // running mode
24     private int mode = 0;
25     // 0: paused/halted/single-step running
26     // 1: running at normal speed
27     // 2: running at a faster speed
28
29     /* virtual memory (abstract files) */
30     private IOBridge ioBridge = null;
31
32     private ProcessorPane pane = new ProcessorPane();
33
34     // timeline for mode == 1
35     private Timeline t1slow = new Timeline();
36     // timeline for mode == 2
37     private Timeline t1fast = new Timeline();
38     // timeline for refreshing
39     private Timeline t1ref = new Timeline();
40
41     public Processor() {
42         // read instruction types
43         Scanner in = null;
44         try {
45             in = new Scanner(new File("./info/ins_type.txt"));
46         } catch (FileNotFoundException e) {
47             e.printStackTrace();
48         }
49         while (in.hasNext()) {
50             int icode, type;
51             icode = in.nextInt();
52             type = in.nextInt();
53             insTypes[icode] = type;
54         }
55         in.close();
56
```

```

57 // init tslow
58 tslow.setCycleCount(Timeline.INDEFINITE);
59 tslow.getKeyFrames().add(new KeyFrame(duration1,
60     (e) -> {
61         try {
62             next();
63         } catch (Exception e1) {
64             handleException(e1);
65         }
66     })
67 );
68 // init tlfast
69 tlfast.setCycleCount(Timeline.INDEFINITE);
70 tlfast.getKeyFrames().add(new KeyFrame(duration2,
71     (e) -> {
72         try {
73             for (int i = 0; i < 15000; i++) next();
74         } catch (Exception e1) {
75             handleException(e1);
76         }
77     })
78 );
79 // init tlref
80 tlref.setCycleCount(Timeline.INDEFINITE);
81 tlref.getKeyFrames().add(
82     new KeyFrame(Duration.millis(50), e -> {
83         try {
84             refresh();
85         } catch (Exception e1) {
86             handleException(e1);
87         }
88     })
89 );
90 tlref.play();
91 }
92
93 private void handleException(Exception e) {
94     if (mode == 1) tslow.pause();
95     else if (mode == 2) tlfast.pause();
96     mode = 0;
97     loaded = false;
98     tlfast.stop();
99     e.printStackTrace();
100 }
101
102 public void setIOBridge(IOBridge iobrg) {
103     ioBridge = iobrg;
104     ioBridge.setProcessor(this);
105 }
106
107 public ProcessorPane getPane() {
108     return pane;
109 }

```

```

110
111  /* program counter */
112  private int pc;
113
114  /*
115   conditional code
116   cc[2]: ZF (zero)
117   cc[1]: SF (signed)
118   cc[0]: OF (overflow)
119  */
120  private byte cc;
121
122  private boolean cond(int sel) {
123      /* return the conditional state */
124      boolean ret = true;
125      boolean zf = false, sf = false, of = false;
126      if ((cc & 4) > 0) zf = true;
127      if ((cc & 2) > 0) sf = true;
128      if ((cc & 1) > 0) of = true;
129      switch (sel) {
130          case 1: ret = zf; break; // e
131          case 2: ret = !zf; break; // ne
132          case 3: ret = (sf == of) && !zf; break; // g
133          case 4: ret = (sf == of); break; // ge
134          case 5: ret = sf ^ of; break; // l
135          case 6: ret = (sf ^ of) || zf; break; // le
136      }
137      return ret;
138  }
139
140  /* state */
141  private byte state;
142
143  /* registers */
144  private long[] regs = new long[15];
145  final private int rsp = 4;
146
147  /* instructions */
148  private int[] insTypes = new int[15];
149
150  private int inslen;
151  private byte[] ir = new byte[10];
152
153  /*
154   * ***** DISPLAY *****
155   */
156
157  private void refresh() throws Exception {
158      pane.refresh();
159  }
160
161  /*
162   * ***** PROCESS *****

```

```

163  */
164
165  private boolean loaded = false;
166  private int nextPC;
167
168  private void fetch() throws Exception {
169      /* fetch instruction */
170      byte op = ioBridge.readb(pc + Memory.OBJECT_SECTION_POS);
171      ir[0] = op;
172      int type = insTypes[op >> 4 & 0xf];
173      // 0: null
174      // 1: r
175      // 2: rr
176      // 3: ri
177      // 4: rri
178      // 5: i
179
180      switch (type) {
181          case 0: insLen = 1; break;
182          case 1: insLen = 2; break;
183          case 2: insLen = 2; break;
184          case 3: insLen = 10; break;
185          case 4: insLen = 10; break;
186          case 5: insLen = 9; break;
187      }
188
189      nextPC = pc + insLen;
190
191      /* store it in instruction registers */
192      for (int i = 1; i < insLen; i++) {
193          ir[i] = ioBridge.readb(pc + i + Memory.OBJECT_SECTION_POS);
194      }
195  }
196
197  private void exec() throws Exception {
198      /* decode and exec */
199      byte icode = (byte) (ir[0] >> 4 & 0xf);
200      byte ifun = (byte) (ir[0] & 0xf);
201
202      switch (icode) {
203          case 0: runIns0(ifun); break;
204          case 1: runIns1(ifun); break;
205          case 2: runIns2(ifun); break;
206          case 3: runIns3(ifun); break;
207          case 4: runIns4(ifun); break;
208          case 5: runIns5(ifun); break;
209          case 6: runIns6(ifun); break;
210          case 7: runIns7(ifun); break;
211          case 8: runIns8(ifun); break;
212          case 9: runIns9(ifun); break;
213      }
214      /* update pc */
215
216      pc = nextPC;

```

```

216 }
217
218 private void init() throws Exception {
219     /* initialize the processor for the newly loaded program */
220     loaded = true;
221     /* initialize program counter */
222     pc = (int) ioBridge.readq(Memory.OBJECT_SECTION_POS);
223     /* clear output buffer */
224     ioBridge.setOutputBufferSize(0);
225     /* clear the registers */
226     for (int i = 0; i < 15; i++) {
227         regs[i] = 0;
228     }
229     /* reset the stack register */
230     regs[rsp] = Memory.STACK_SECTION_LIMIT;
231     /* put a random seed in memory */
232     ioBridge.setRandomSeed((int) System.currentTimeMillis() % 48271);
233     ioBridge.clearDisplay();
234     ioBridge.clearTextOutput();
235 }
236
237 private void next() throws Exception {
238     /* run the next instruction cycle */
239     if (loaded) {
240         fetch();
241         exec();
242     }
243 }
244
245 private void runIns0(byte ifun) throws Exception {
246     switch (ifun) {
247         case 0:
248             throw new Exception("HALT");
249         case 1:
250             break;
251         case 2:
252             nextPC = (int) ioBridge.readq((int) regs[rsp]);
253             regs[rsp] += 8;
254             break;
255         case 3:
256             // TODO
257             break;
258     }
259 }
260
261 private void runIns1(byte ifun) {
262     if (ifun == 0) {
263         byte rA = (byte) (ir[1] >> 4 & 0xf);
264         long imme = 0;
265         for (int i = 0; i < 8; i++) {
266             imme += ((long)ir[i + 2] & 0xff) << (i << 3);
267         }
268         regs[rA] = imme;

```

```

269     }
270 }
271
272 private void runIns2(byte ifun) throws Exception {
273     byte rA = (byte) (ir[1] >> 4 & 0xf);
274     byte rB = (byte) (ir[1] & 0xf);
275     long imme = 0;
276     for (int i = 0; i < 8; i++) {
277         imme += ((long)ir[i + 2] & 0xff) << (i << 3);
278     }
279     int addr = (int) imme;
280     if (rB != 0xf) addr += (int) regs[rB];
281     switch (ifun) {
282         case 0: ioBridge.writeq(addr, regs[rA]); break;
283         case 1: ioBridge.writel(addr, (int) regs[rA]); break;
284         case 2: ioBridge.writew(addr, (short) regs[rA]); break;
285         case 3: ioBridge.writeb(addr, (byte) regs[rA]); break;
286         case 4: regs[rA] = ioBridge.readq(addr); break;
287         case 5: regs[rA] = 0xffffffffL & (long) ioBridge.readl(addr); break;
288         case 6: regs[rA] = 0xffffL & (long) ioBridge.readw(addr); break;
289         case 7: regs[rA] = 0xffL & (long) ioBridge.readb(addr); break;
290     }
291 }
292
293 private void runIns3(byte ifun) {
294     byte rA = (byte) (ir[1] >> 4 & 0xf);
295     byte rB = (byte) (ir[1] & 0xf);
296     boolean cnd = cond(ifun);
297     if (cnd) regs[rB] = regs[rA];
298 }
299
300 private void runIns4(byte ifun) {
301     byte rA = (byte) (ir[1] >> 4 & 0xf);
302     byte rB = (byte) (ir[1] & 0xf);
303     long vA = regs[rA];
304     long vB = regs[rB];
305     long vC = 0;
306
307     boolean zf, sf, of = false;
308
309     switch (ifun) {
310         case 0: // add
311             vC = vA + vB;
312             of = (vA < 0 == vB < 0) && (vA < 0 != vC < 0);
313             break;
314         case 1: // sub
315             vC = vA - vB;
316             of = (vA < 0 != vB < 0) && (vA < 0 != vC < 0);
317             break;
318         case 2: // and
319             vC = vA & vB; break;
320         case 3: // or
321             vC = vA | vB; break;

```

```

322     case 4: // xor
323         vC = vA ^ vB; break;
324     case 5: // sal
325         vC = vA << (vB & 0x3f); break;
326     case 6: // sar
327         vC = vA >> (vB & 0x3f); break;
328     case 7: // shr
329         vC = vA >>> (vB & 0x3f); break;
330     case 8: // mul
331         vC = vA * vB; break;
332     case 9: // idiv
333         vC = vA / vB;
334         regs[0] = vA % vB;
335         if (rA == 0) vC = regs[0];
336         break;
337 }
338
339 zf = vC == 0;
340 sf = vC < 0;
341 regs[rA] = vC;
342
343 cc = 0;
344 if (zf) cc |= 4;
345 if (sf) cc |= 2;
346 if (of) cc |= 1;
347 }
348
349 private void runIns5(byte ifun) {
350     byte rA = (byte) (ir[1] >> 4 & 0xf);
351     long vA = regs[rA];
352     long vC = 0;
353
354     boolean zf, sf, of = false;
355     switch (ifun) {
356         case 0: vC = ~vA; break; // not
357         case 1: vC = -vA; break; // neg
358         case 2:
359             vC = vA + 1;
360             of = vC < 0 && vA > 0;
361             break; // inc
362         case 3:
363             vC = vA - 1;
364             of = vC > 0 && vA < 0;
365             break; // dec
366         case 4: vC = (long) (int) vA; break; // cltq
367         case 5: vC = (long) (short) vA; break; // cwtq
368         case 6: vC = (long) (byte) vA; break; // cbtq
369         case 7: vC = vA & 0xffffffffL; break; // cqt1
370         case 8: vC = vA & 0xffffL; break; // cqtw
371         case 9: vC = vA & 0xffL; break; // cqtb
372     }
373
374     regs[rA] = vC;

```



```

375
376     zf = (vC == 0);
377     sf = (vC < 0);
378     cc = 0;
379     if (zf) cc |= 4;
380     if (sf) cc |= 2;
381     if (of) cc |= 1;
382 }
383
384 private void runIns6(byte ifun) {
385     byte rA = (byte) (ir[1] >> 4 & 0xf);
386     byte rB = (byte) (ir[1] & 0xf);
387     long vA = regs[rA];
388     long vB = regs[rB];
389     long vC = 0;
390
391     boolean sf, zf, of = false;
392
393     switch (ifun) {
394         case 0: // cmp
395             vC = vA - vB;
396             of = (vA < 0 != vB < 0) && (vA < 0 != vC < 0);
397             break;
398         case 1: // test
399             vC = vA & vB;
400             break;
401     }
402
403     zf = (vC == 0);
404     sf = (vC < 0);
405     cc = 0;
406     if (zf) cc |= 4;
407     if (sf) cc |= 2;
408     if (of) cc |= 1;
409 }
410
411 private void runIns7(byte ifun) throws Exception {
412     /* branch ins */
413     int target = 0;
414     for (int i = 0; i < 4; i++) {
415         target += (((int)ir[i + 1] & 0xff) << (i << 3));
416     }
417     boolean cnd = cond(ifun);
418     if (ifun == 7) {
419         ioBridge.writeq((int) (regs[rsp] - 8), nextPC);
420         regs[rsp] -= 8;
421     }
422     if (cnd) nextPC = target;
423 }
424
425 private void runIns8(byte ifun) throws Exception {
426     byte rA = (byte) (ir[1] >> 4 & 0xf);
427
428     if (ifun == 0) { // push

```

```

428     ioBridge.writeq((int) (regs[rsp] - 8), regs[rA]);
429     regs[rsp] -= 8;
430 } else if (ifun == 1) { // pop
431     long vA = ioBridge.readq((int) regs[rsp]);
432     regs[rsp] += 8;
433     regs[rA] = vA;
434 }
435 }
436
437 private void runIns9(byte ifun) throws Exception {
438     /* interrupt */
439     long imme = 0;
440     for (int i = 0; i < 8; i++) {
441         imme += ((long)ir[1 + i] & 0xff) << (i << 3);
442     }
443     ioBridge.interupt((int) imme);
444 }
445
446 /*
447  * ***** END PROCESS *****
448  */
449
450
451 public class ProcessorPane extends BorderPane {
452     /* ControlPane of the Processor */
453
454     private Button btLoad = new Button("Load");
455     private Button btRun = new Button("Run");
456     private Button btFast = new Button("Fast"); // execute at a faster speed
457     private Button btPause = new Button("Pause");
458     private Button btStep = new Button("Step"); // single step execution
459     private Button btHalt = new Button("Halt");
460
461     private Label[] registers = new Label[15];
462     private Label PCLabel;
463
464     private GridPane centerPane = new GridPane();
465     private HBox bottomPane = new HBox();
466
467     public ProcessorPane() {
468         configureBottomPane();
469         bottomPane.getChildren().addAll(btLoad, btRun, btFast, btPause, btStep, btHalt);
470         bottomPane.setId("bottom-pane");
471         setBottom(bottomPane);
472
473         configureCenterPane();
474         centerPane.setId("center-pane");
475         setCenter(centerPane);
476     }
477
478     private void configureCenterPane() {
479         /* show registers */
480
481         String[] regNames = {

```

```

481         "%rax", "%rcx", "%rdx", "%rbx", "%rsp", "%rbp", "%rsi", "%rdi",
482         "%r8", "%r9", "%r10", "%r11", "%r12", "%r13", "%r14",
483     };
484     final int nameWidth = 50;
485     for (int i = 0; i < 15; i++) {
486         /* show register name */
487         Label name = new Label(regNames[i]);
488         name.setId("register-name");
489         /* show register value */
490         Label label = new Label();
491         label.setId("register-value");
492         centerPane.widthProperty().addListener(ov -> {
493             name.setPrefWidth(nameWidth);
494             label.setPrefWidth(centerPane.getWidth() / 2 - nameWidth);
495         });
496         centerPane.heightProperty().addListener(ov -> {
497             name.setPrefHeight(centerPane.getHeight() / 8);
498             label.setPrefHeight(centerPane.getHeight() / 8);
499         });
500         registers[i] = label;
501         centerPane.add(name, i / 8 * 2, i % 8);
502         centerPane.add(label, i / 8 * 2 + 1, i % 8);
503     }
504     /* show pc */
505     Label PCName = new Label("PC");
506     PCName.setId("pc-name");
507     PCLabel = new Label();
508     PCLabel.setId("pc-value");
509     centerPane.widthProperty().addListener(ov -> {
510         PCName.setPrefWidth(nameWidth);
511         PCLabel.setPrefWidth(centerPane.getWidth() / 2 - nameWidth);
512     });
513     centerPane.heightProperty().addListener(ov -> {
514         PCName.setPrefHeight(centerPane.getHeight() / 8);
515         PCLabel.setPrefHeight(centerPane.getHeight() / 8);
516     });
517
518     centerPane.add(PCName, 2, 7);
519     centerPane.add(PCLabel, 3, 7);
520 }
521
522 private void configureBottomPane() {
523     bottomPane.widthProperty().addListener(ov -> {
524         btLoad.setPrefWidth(getWidth() / 6);
525         btRun.setPrefWidth(getWidth() / 6);
526         btFast.setPrefWidth(getWidth() / 6);
527         btPause.setPrefWidth(getWidth() / 6);
528         btStep.setPrefWidth(getWidth() / 6);
529         btHalt.setPrefWidth(getWidth() / 6);
530     });
531
532     btRun.setOnAction(e -> {
533
534         if (mode == 2) tlfast.pause();

```

```

534     mode = 1;
535     tslow.play();
536 });
537 btFast.setOnAction(e -> {
538     if (mode == 1) tslow.pause();
539     mode = 2;
540     tlfast.play();
541 });
542
543 btPause.setOnAction(e -> {
544     if (mode == 1) {
545         tslow.pause();
546     } else if (mode == 2) {
547         tlfast.pause();
548     }
549     mode = 0;
550 });
551
552 btHalt.setOnAction(e -> {
553     if (mode == 1) {
554         tslow.stop();
555     } else if (mode == 2) {
556         tlfast.stop();
557     }
558     mode = 0;
559     try {
560         if (loaded) init();
561     } catch (Exception e1) {
562         e1.printStackTrace();
563     }
564 });
565
566 btStep.setOnAction(e -> {
567     if (mode == 1) tslow.pause();
568     else if (mode == 2) tlfast.pause();
569     mode = 0;
570     try {
571         next();
572     } catch (Exception e1) {
573         handleException(e1);
574     }
575 });
576 }
577
578 public void configureLoadButton(Stage stage) {
579     FileChooser fileChooser = new FileChooser();
580     fileChooser.setTitle("Choose object file");
581     fileChooser.setInitialDirectory(new File("./hex/"));
582     fileChooser.getExtensionFilters().add(
583         new FileChooser.ExtensionFilter("HEX", "*.hex")
584     );
585     /* load object file */
586     btLoad.setOnAction((e) -> {

```

```

587         if (mode == 1) t1slow.stop();
588         else if (mode == 2) t1fast.stop();
589         mode = 0;
590         File file = fileChooser.showOpenDialog(stage);
591         if (file != null) {
592             try {
593                 ioBridge.loadObject(file);
594                 init();
595             } catch (Exception e1) {
596                 e1.printStackTrace();
597             }
598         }
599     });
600 }
601
602 public void refresh() throws Exception {
603     /* update the values of registers and memory units */
604     for (int i = 0; i < 15; i++) {
605         registers[i].setText(Long.toString(regs[i]));
606     }
607     PCLabel.setText(Integer.toHexString(pc));
608     ioBridge.refresh();
609 }
610 }
611 }

```

AbstractFile.java

```

1 public interface AbstractFile {
2     int getMaxSize();
3     boolean isReadable();
4     boolean isWritable();
5 }

```

ReadableFile.java

```

1 public interface ReadableFile extends AbstractFile {
2     byte readb(int index) throws Exception;
3     short readw(int index) throws Exception;
4     int readl(int index) throws Exception;
5     long readq(int index) throws Exception;
6 }

```

WritableFile.java

```

1 public interface WritableFile extends AbstractFile {
2     void writeb(int index, byte val) throws Exception;
3     void writew(int index, short val) throws Exception;
4     void writel(int index, int val) throws Exception;
5     void writeq(int index, long val) throws Exception;
6 }

```

IOBridge.java

```

1 import java.io.*;
2
3 public class IOBridge {
4
5     private AbstractFile[] files = new AbstractFile[4];
6
7     private Processor processor = null;
8
9     private Memory memory;
10    private Keyboard keyboard;
11    private TextOutput textOutput;
12    private Display display;
13
14    public void setMemory(Memory memory) {
15        this.memory = memory;
16        files[0] = memory;
17    }
18
19    public void setKeyboard(Keyboard keyboard) {
20        this.keyboard = keyboard;
21        files[1] = keyboard;
22    }
23
24    public void setTextOutput(TextOutput textOutput) {
25        this.textOutput = textOutput;
26        files[2] = textOutput;
27    }
28
29    public void setDisplay(Display display) {
30        this.display = display;
31        files[3] = display;
32    }
33
34    public void setProcessor(Processor pro) {
35        processor = pro;
36    }
37
38    public synchronized void setKeyboardInterrupt() {
39        // TODO
40    }

```

```

41
42 public void loadObject(File file) throws IOException {
43     memory.load(file);
44 }
45
46 public void clearTextOutput() {
47     textOutput.clear();
48 }
49
50 public void clearDisplay() {
51     display.clear();
52 }
53
54 public void refresh() throws Exception {
55     memory.refresh();
56 }
57
58 public synchronized void setInputBufferSize() throws Exception {
59     // TODO
60 }
61
62 public synchronized void setOutputBufferSize(long value) throws Exception {
63     memory.writeq(8, value);
64 }
65
66 public long getOutputBufferSize() throws Exception {
67     return memory.readq(8);
68 }
69
70 public synchronized void setRandomSeed(int value) throws Exception {
71     memory.writel(16, value);
72 }
73
74 public void interrupt(int value) throws Exception {
75     switch (value) {
76         case 1: /* TODO */ break;
77         case 2: textOutput.print(); break;
78         case 3: display.paint(); break;
79     }
80 }
81
82 private int absoluteIndex;
83
84 private AbstractFile getAbstractFile(int index) throws Exception {
85     absoluteIndex = index;
86     if (absoluteIndex < 0) throw new Exception("^V^V^V^V^V^V");
87     for (AbstractFile file : files) {
88         if (absoluteIndex < file.getMaxSize()) return file;
89         absoluteIndex -= file.getMaxSize();
90     }
91     throw new Exception("Illegal Memory Address !");
92 }
93

```

```
94     private ReadableFile getReadableFile(int index) throws Exception {
95         AbstractFile file = getAbstractFile(index);
96         if (!file.isReadable()) throw new Exception("NotAReadableFile");
97         return (ReadableFile)file;
98     }
99
100    private WritableFile getWritableFile(int index) throws Exception {
101        AbstractFile file = getAbstractFile(index);
102        if (!file.isWritable()) throw new Exception("NotAWritableFile");
103        return (WritableFile)file;
104    }
105
106    public byte readb(int index) throws Exception {
107        ReadableFile file = getReadableFile(index);
108        return file.readb(absoluteIndex);
109    }
110
111    public short readw(int index) throws Exception {
112        ReadableFile file = getReadableFile(index);
113        return file.readw(absoluteIndex);
114    }
115
116    public int readl(int index) throws Exception {
117        ReadableFile file = getReadableFile(index);
118        return file.readl(absoluteIndex);
119    }
120
121    public long readq(int index) throws Exception {
122        ReadableFile file = getReadableFile(index);
123        return file.readq(absoluteIndex);
124    }
125
126    public void writeb(int index, byte val) throws Exception {
127        WritableFile file = getWritableFile(index);
128        file.writeb(absoluteIndex, val);
129    }
130
131    public void writew(int index, short val) throws Exception {
132        WritableFile file = getWritableFile(index);
133        file.writew(absoluteIndex, val);
134    }
135
136    public void writel(int index, int val) throws Exception {
137        WritableFile file = getWritableFile(index);
138        file.writel(absoluteIndex, val);
139    }
140
141    public void writeq(int index, long val) throws Exception {
142        WritableFile file = getWritableFile(index);
143        file.writeq(absoluteIndex, val);
144    }
145
146 }
```


Memory.java

```
1  import javafx.scene.control.Label;
2  import javafx.scene.control.TextField;
3  import javafx.scene.layout.GridPane;
4
5  import java.io.*;
6
7  public class Memory implements ReadableFile, WritableFile {
8
9      private final int MAX_SIZE;
10     private byte[] ram;
11
12     private IOBridge ioBridge = null;
13
14     private MemoryPane pane = new MemoryPane();
15
16     public MemoryPane getPane() {
17         return pane;
18     }
19
20     final public static int RESERVE_SECTION_POS = 0x00000000;
21     final public static int OBJECT_SECTION_POS = 0x10000000;
22     final public static int STACK_SECTION_POS = 0x20000000;
23     final public static int STACK_SECTION_LIMIT = 0x30000000;
24     final public static int DATA_SECTION_POS = 0x30000000;
25
26     public Memory() {
27         this(0x10000000);
28     }
29
30     public Memory(int maxSize) {
31         ram = new byte[maxSize];
32         MAX_SIZE = maxSize;
33     }
34
35     public void setIOBridge(IOBridge iob) {
36         ioBridge = iob;
37         ioBridge.setMemory(this);
38     }
39
40     @Override
41     public int getMaxSize() {
42         return MAX_SIZE;
43     }
44
45     @Override
46     public boolean isReadable() {
47         return true;
48     }
49 }
```

```

48     }
49
50     @Override
51     public boolean isWritable() {
52         return true;
53     }
54
55     public void load(File file) throws IOException {
56         RandomAccessFile raf = new RandomAccessFile(file, "r");
57         raf.read(ram, OBJECT_SECTION_POS, MAX_SIZE - OBJECT_SECTION_POS);
58     }
59
60     @Override
61     public byte readb(int index) throws Exception {
62         if (index < 0 || index >= MAX_SIZE) {
63             throw new Exception();
64         }
65         return ram[index];
66     }
67
68     @Override
69     public short readw(int index) throws Exception {
70         if (index < 0 || index + 2 > MAX_SIZE) {
71             throw new Exception();
72         }
73         short val = (short) (((short)ram[index] << 8) +
74             ((short) ram[index + 1] & 0xff));
75         return val;
76     }
77
78     @Override
79     public int readl(int index) throws Exception {
80         if (index < 0 || index + 4 > MAX_SIZE) {
81             throw new Exception();
82         }
83         int val = 0;
84         for (int i = 3; i >= 0; i--) {
85             val <<= 8;
86             val += ((int) ram[index + i] & 0xff);
87         }
88         return val;
89     }
90
91     @Override
92     public long readq(int index) throws Exception {
93         if (index < 0 || index + 8 > MAX_SIZE) {
94             throw new Exception();
95         }
96         long val = 0;
97         for (int i = 7; i >= 0; i--) {
98             val <<= 8;
99             val += ((long) ram[index + i] & 0xff);
100     }

```

```

101     return val;
102 }
103
104 @Override
105 public void writeb(int index, byte val) throws Exception {
106     if (index < 0 || index >= MAX_SIZE) {
107         throw new Exception();
108     }
109     ram[index] = val;
110 }
111
112 @Override
113 public void writew(int index, short val) throws Exception {
114     if (index < 0 || index + 2 > MAX_SIZE) {
115         throw new Exception();
116     }
117     ram[index] = (byte) (val & 0xff);
118     ram[index + 1] = (byte) (val >> 8 & 0xff);
119 }
120
121 @Override
122 public void writel(int index, int val) throws Exception {
123     if (index < 0 || index + 4 > MAX_SIZE) {
124         throw new Exception();
125     }
126     ram[index] = (byte) (val & 0xff);
127     ram[index + 1] = (byte) (val >> 8 & 0xff);
128     ram[index + 2] = (byte) (val >> 16 & 0xff);
129     ram[index + 3] = (byte) (val >> 24 & 0xff);
130 }
131
132 @Override
133 public void writeq(int index, long val) throws Exception {
134     if (index < 0 || index + 8 > MAX_SIZE) {
135         throw new Exception();
136     }
137     ram[index] = (byte) (val & 0xff);
138     ram[index + 1] = (byte) (val >> 8 & 0xff);
139     ram[index + 2] = (byte) (val >> 16 & 0xff);
140     ram[index + 3] = (byte) (val >> 24 & 0xff);
141     ram[index + 4] = (byte) (val >> 32 & 0xff);
142     ram[index + 5] = (byte) (val >> 40 & 0xff);
143     ram[index + 6] = (byte) (val >> 48 & 0xff);
144     ram[index + 7] = (byte) (val >> 56 & 0xff);
145 }
146
147 public void refresh() throws Exception {
148     pane.refresh();
149 }
150
151 public class MemoryPane extends GridPane {
152     private TextField[] address = new TextField[10];
153     private Label[] value = new Label[10];

```

```

154
155 public MemoryPane() {
156     this.setId("memory-pane");
157     // add components
158     for (int i = 0; i < 10; i++) {
159         TextField tf = new TextField();
160         Label lb = new Label();
161         this.widthProperty().addListener(ov -> {
162             tf.setPrefWidth(getWidth() / 2);
163             lb.setPrefWidth(getWidth() - tf.getWidth());
164         });
165         this.heightProperty().addListener(ov -> {
166             tf.setPrefHeight(getHeight() / 10);
167             lb.setPrefHeight(getHeight() / 10);
168         });
169         tf.setId("memory-text-field");
170         lb.setId("memory-label");
171         address[i] = tf;
172         value[i] = lb;
173         this.add(tf, 0, i);
174         this.add(lb, 1, i);
175     }
176 }
177
178 private int parseInt(String str) {
179     if (str.isEmpty()) return -1;
180
181     StringBuilder builder = new StringBuilder(str);
182     boolean hex = false;
183     if (builder.length() > 2 && builder.substring(0, 2).equals("0x")) {
184         hex = true;
185         builder.delete(0, 2);
186     }
187     int val = 0;
188     if (hex) {
189         int len = builder.length();
190         for (int i = 0; i < len; i++) {
191             char c = builder.charAt(i);
192             if (Character.isDigit(c) || Character.isAlphabetic(c)) {
193                 c = Character.toLowerCase(c);
194                 if (c > 'f') return -1;
195                 if (Character.isDigit(c)) val = (val << 4) + c - '0';
196                 else val = (val << 4) + c - 'a' + 10;
197             } else return -1;
198         }
199     } else {
200         int len = builder.length();
201         for (int i = 0; i < len; i++) {
202             char c = builder.charAt(i);
203             if (Character.isDigit(c)) {
204                 val = val * 10 + c - '0';
205             } else return -1;
206         }

```

```

207     }
208     if (val + 16 > MAX_SIZE || val < 0) return -1;
209     return val;
210 }
211
212 private void refresh() throws Exception {
213     for (int i = 0; i < 10; i++) {
214         int addr = parseInt(address[i].getText());
215         if (addr == -1) {
216             value[i].setText("");
217         } else {
218             StringBuffer strVal = new StringBuffer();
219             for (int j = 0; j < 16; j++) {
220                 if (j > 0) strVal.append(' ');
221                 if (j == 8) strVal.append(' ');
222                 int val = 0xff & (int) readb(addr + j);
223                 StringBuffer byteVal = new StringBuffer(Integer.toHexString(val));
224                 if (byteVal.length() < 2) byteVal.insert(0, '0');
225                 strVal.append(byteVal);
226             }
227             value[i].setText(strVal.toString());
228         }
229     }
230 }
231 }
232
233 }

```

Keyboard.java

```

1  import javafx.scene.layout.Pane;
2
3  public class Keyboard implements ReadableFile {
4
5      /*
6       * TODO
7       */
8
9      public final int MAX_SIZE;
10     private byte[] buffer;
11
12     private IOBridge ioBridge = null;
13
14     private KeyboardPane pane = new KeyboardPane();
15
16     public Keyboard() {
17         this(0x80000);
18     }
19
20     public Keyboard(int maxSize) {

```

```
21     buffer = new byte[maxSize];
22     MAX_SIZE = maxSize;
23 }
24
25 public void setIOBridge(IOBridge iob) {
26     ioBridge = iob;
27     ioBridge.setKeyboard(this);
28 }
29
30 public KeyboardPane getPane() {
31     return pane;
32 }
33
34 @Override
35 public int getMaxSize() {
36     return MAX_SIZE;
37 }
38
39 @Override
40 public boolean isReadable() {
41     return true;
42 }
43
44 @Override
45 public boolean isWritable() {
46     return false;
47 }
48
49 @Override
50 public byte readb(int index) throws Exception {
51     if (index < 0 || index >= MAX_SIZE) {
52         throw new Exception();
53     }
54     return buffer[index];
55 }
56
57 @Override
58 public short readw(int index) throws Exception {
59     if (index < 0 || index + 2 > MAX_SIZE) {
60         throw new Exception();
61     }
62     short val = buffer[index + 1];
63     val <= 8;
64     val += ((short) buffer[index] & 0xff);
65     return val;
66 }
67
68 @Override
69 public int readl(int index) throws Exception {
70     if (index < 0 || index + 4 > MAX_SIZE) {
71         throw new Exception();
72     }
73     int val = 0;
```

```

74     for (int i = 3; i >= 0; i--) {
75         val <<= 8;
76         val += ((int) buffer[index + i] & 0xff);
77     }
78     return val;
79 }
80
81 @Override
82 public long readq(int index) throws Exception {
83     if (index < 0 || index + 8 > MAX_SIZE) {
84         throw new Exception();
85     }
86     long val = 0;
87     for (int i = 7; i >= 0; i--) {
88         val <<= 8;
89         val += ((long) buffer[index + i] & 0xff);
90     }
91     return val;
92 }
93
94 public class KeyboardPane extends Pane {
95     public KeyboardPane() {
96         this.setId("keyboard-pane");
97     }
98 }
99 }

```

TextOutput.java

```

1  import javafx.scene.control.TextArea;
2  import javafx.scene.layout.Pane;
3
4  public class TextOutput implements WritableFile {
5
6      final private int MAX_SIZE = 0x80000;
7
8      private byte[] ram = new byte[MAX_SIZE];
9
10     private IOBridge ioBridge = null;
11
12     private TextOutputPane pane = new TextOutputPane();
13
14     public void setIOBridge(IOBridge iob) {
15         ioBridge = iob;
16         ioBridge.setTextOutput(this);
17     }
18
19     public TextOutputPane getPane() {
20         return pane;
21     }

```

```

22
23 @Override
24 public int getMaxSize() {
25     return MAX_SIZE;
26 }
27
28 @Override
29 public boolean isReadable() {
30     return false;
31 }
32
33 @Override
34 public boolean isWritable() {
35     return true;
36 }
37
38 @Override
39 public void writeb(int index, byte val) throws Exception {
40     if (index < 0 || index >= MAX_SIZE) {
41         throw new Exception();
42     }
43     ram[index] = val;
44 }
45
46 @Override
47 public void writew(int index, short val) throws Exception {
48     if (index < 0 || index + 2 > MAX_SIZE) {
49         throw new Exception();
50     }
51     ram[index] = (byte) (val & 0xff);
52     ram[index + 1] = (byte) (val >> 8 & 0xff);
53 }
54
55 @Override
56 public void writel(int index, int val) throws Exception {
57     if (index < 0 || index + 4 > MAX_SIZE) {
58         throw new Exception();
59     }
60     ram[index] = (byte) (val & 0xff);
61     ram[index + 1] = (byte) (val >> 8 & 0xff);
62     ram[index + 2] = (byte) (val >> 16 & 0xff);
63     ram[index + 3] = (byte) (val >> 24 & 0xff);
64 }
65
66 @Override
67 public void writeq(int index, long val) throws Exception {
68     if (index < 0 || index + 8 > MAX_SIZE) {
69         throw new Exception();
70     }
71     ram[index] = (byte) (val & 0xff);
72     ram[index + 1] = (byte) (val >> 8 & 0xff);
73     ram[index + 2] = (byte) (val >> 16 & 0xff);
74     ram[index + 3] = (byte) (val >> 24 & 0xff);

```



```

75     ram[index + 4] = (byte) (val >> 32 & 0xff);
76     ram[index + 5] = (byte) (val >> 40 & 0xff);
77     ram[index + 6] = (byte) (val >> 48 & 0xff);
78     ram[index + 7] = (byte) (val >> 56 & 0xff);
79 }
80
81 public void clear() {
82     /* clear the buffer and the output pane */
83     pane.clear();
84 }
85
86 public synchronized void print() throws Exception {
87     /* move the content from the buffer to the output pane
88      * then clear the buffer */
89     pane.print();
90 }
91
92 public class TextOutputPane extends Pane {
93     private TextArea textArea = new TextArea();
94     private String textStr = "";
95
96     public TextOutputPane() {
97         textArea.setText(textStr);
98         this.widthProperty().addListener(ov ->
99             textArea.setPrefWidth(this.getWidth()));
100        this.heightProperty().addListener(ov ->
101            textArea.setPrefHeight(this.getHeight()));
102        textArea.setEditable(false);
103        textArea.setId("text-output-text-area");
104        this.setId("text-output-pane");
105        this.getChildren().add(textArea);
106    }
107
108     public void clear() {
109         textStr = "";
110         textArea.setText(textStr);
111     }
112
113     public synchronized void print() throws Exception {
114         long size = ioBridge.getOutputBufferSize();
115         StringBuilder str = new StringBuilder();
116         for (int i = 0; i < MAX_SIZE && i < size; i++) {
117             str.append((char) ram[i]);
118         }
119
120         textStr += str;
121         textArea.setText(textStr);
122         textArea.setScrollTop(Double.MAX_VALUE);
123         ioBridge.setOutputBufferSize(0);
124     }
125 }
126
127 }

```

Display.java

```
1  import javafx.scene.canvas.Canvas;
2  import javafx.scene.canvas.GraphicsContext;
3  import javafx.scene.paint.Color;
4
5  public class Display implements WritableFile {
6
7      final private int MAX_SIZE = 0x100000;
8      final private int W; // display's width
9      final private int H; // display's height
10     final private int MAX_POS;
11
12     private int[] ram = new int [MAX_SIZE]; // display's memory
13
14     private IOBridge ioBridge = null;
15
16     private DisplayCanvas canvas;
17
18     public Display(int w, int h) {
19         W = w / 2; H = h / 2;
20         MAX_POS = W * H;
21         if (MAX_POS > MAX_SIZE) {
22             System.exit(0);
23         }
24
25         canvas = new DisplayCanvas();
26         /* clear the memory */
27         for (int i = 0; i < MAX_SIZE; i++) ram[i] = 0;
28     }
29
30     public void setIOBridge(IOBridge iob) {
31         ioBridge = iob;
32         ioBridge.setDisplay(this);
33     }
34
35     public DisplayCanvas getCanvas() {
36         return canvas;
37     }
38
39     @Override
40     public int getMaxSize() {
41         return MAX_SIZE;
42     }
43
44     @Override
45     public boolean isReadable() {
46         return false;
47     }
```

```

48
49 @Override
50 public boolean isWritable() {
51     return true;
52 }
53
54 @Override
55 public void writeb(int index, byte val) throws Exception {
56     throw new Exception("Data size banned !");
57 }
58
59 @Override
60 public void writew(int index, short val) throws Exception {
61     throw new Exception("Data size banned !");
62 }
63
64 @Override
65 public synchronized void writel(int index, int val) throws Exception {
66     if (index > MAX_SIZE) {
67         throw new Exception("Invalid index !");
68     }
69     if ((index & 3) > 0) {
70         throw new Exception("Data aligning is required !");
71     }
72     index >>= 2;
73     ram[index] = val;
74 }
75
76 @Override
77 public void writeq(int index, long val) throws Exception {
78     throw new Exception("Data size banned !");
79 }
80
81 public void clear() {
82     /* clear the display */
83     for (int i = 0; i < MAX_POS; i++) {
84         ram[i] = 0;
85     }
86     canvas.paint();
87 }
88
89 public synchronized void paint() {
90     /* paint current frame and clear the memory */
91     canvas.paint();
92     for (int i = 0; i < MAX_POS; i++) {
93         ram[i] = 0;
94     }
95 }
96
97 public class DisplayCanvas extends Canvas {
98
99     private GraphicsContext gc;

```

100

```
101 public DisplayCanvas() {
102     super(W * 2, H * 2);
103     gc = this.getGraphicsContext2D();
104     paint();
105 }
106
107 public void paint() {
108     /* draw pixels to the display */
109     gc.setFill(Color.BLACK);
110     gc.fillRect(0, 0, this.getWidth(), this.getHeight());
111
112     int pos = 0;
113     for (int x = 0; x < W; x++) {
114         for (int y = 0; y < H; y++, pos++) {
115             int value = ram[pos];
116             int red = (value >> 24) & 0xff;
117             int green = (value >> 16) & 0xff;
118             int blue = (value >> 8) & 0xff;
119             int alpha = value & 0xff;
120             double r = red / 255.0;
121             double g = green / 255.0;
122             double b = blue / 255.0;
123             double a = alpha / 255.0;
124             gc.setFill(Color.color(r, g, b, a));
125             gc.fillRect(x << 1, y << 1, 2, 2);
126         }
127     }
128 }
129
130 }
131
132 }
```