ZJUQS-II SWORD 版模拟器使用说明

目录

一、功能描述

本工具实现了一个小型的 MIPS 架构 CPU，能够运行、调试一部分 MIPS 指令集，并实时显示寄存器和内存的变化。本工具同时模拟了 SWORD 平台上的硬件外设（包括 VGA、七段码、滑动开关、阵列按钮、计数器、LED 灯和 PS2 键盘），通过图形化界面模拟运行结果，并提供了一系列硬件接口供汇编代码调用。

模拟器旨在模拟计算机硬件相关的实验课所实现的 CPU，并提供完整的图形化 SWORD 环境，方便用户调试代码，节省往返实验室的时间。

二、运行方法

模拟器使用了 Java 语言编写，并使用 JavaFX 框架来实现跨平台图形化界面，因此首先需要安装 JRE 环境。此处推荐安装已经内嵌了 JavaFX 的 Java 8。因为 Java 11 已经移除了 JavaFX，如果电脑已经配置了 11+的 Java 版本，则需要手动进行编译。编译方法见第四部分。

在完成 Java 环境配置后，双击.jar 文件即可打开模拟器。如果不能正常打开，可以通过以下命令来运行本模拟器：

```
1.  java -jar [模拟器所在路径]
```

本模拟器已在 Windows 10 平台通过测试。如果在 macOS 或 Linux 平台上遇到运行问题，请在第四部分的 github 地址提供反馈，或尝试自行编译。

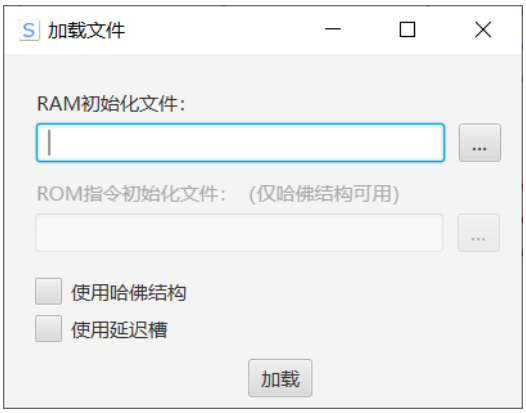
三、使用说明

3.1 主界面介绍

上图为模拟器的各个图形化界面组成部分。左下角为功能按钮。

3.2 加载汇编代码

点击“加载”按钮即可载入初始化文件至内存中。载入界面如下：



模拟器支持模拟哈弗架构（即代码储存在 ROM 中、数据储存在 RAM 中）和冯诺依曼架构（代码和数据均储存在 RAM 中）。用户可以根据需要，选择是否使用哈弗架构。**一般而言，单周期和流水线系统可以采用哈弗架构，而多周期系统采用冯诺依曼架构。**

对于哈弗架构，模拟器将从 ROM 中读取代码，从 RAM 中读写数据（即 lw、sw 指令操作的地址在 RAM 中）。对于冯诺依曼架构，模拟器将从 RAM 中读取代码、读写数据。无论使用何种架构，模拟器均从对应代码内存的第一行开始运行。

对于流水线系统，请勾选“使用延迟槽”。

模拟器支持载入 coe、bin 和 hex 文件类型至内存中。对于后缀名为 coe 和 hex 的文件，模拟器会自动将字符串解析为字节数据。对于 bin 类型的文件，模拟器将直接把字节数据载入内存。

3.3 内存模拟

RAM 和 ROM 的内存空间均为 32K × 32 位 WORD，地址空间为 0x00000000H~0x0000FFFF，可通过 sw 或 lw 指令操作 RAM。

点击“内存查看”按钮，即可打开内存调试窗口（包括 RAM 和 ROM）。界面如下：

用户可以选择指定地址的内存数据。点击“查看 PC 指针”将跳转到 PC 寄存器储存的地址处。此功能适合在单步调试时使用，每次点击“单步执行”按钮，内存调试窗口将会实时高亮 PC 位置。

3.4 外设模拟（硬件接口文档）

模拟器提供了内存地址总线（Address Bus）的方式作为硬件接口。每个外设都对应着某个地址，用户可以通过 lw、sw 指令读写该地址，从而调用该硬件接口。

模拟器已经预定义了一系列地址接口。可以在**配置 – 自定义总线外设地址**处进行修改。

3.4.1 VGA 显示模拟

VGA 显示支持图形和文本两种模式，可以在左下角位置进行切换。

1. 模式控制寄存器

VGA 的高级配置储存在**模式控制寄存器**中。该寄存器数据格式如下：

M：模式位，置 0 为文本模式，置 1 为图形模式
C：硬件光标使能位，置 1 时开启硬件光标。（该功能暂未实现）
Font：字库模式 1xx 保留。
Font = 000：英文 8*8 (标准/启动)
Font = 001：英文 8*16
Font = 010：中文 16*16
Font = 011：中文 32*32
R：分辨率选择，0000 表示关闭显示(VGA)输出，除下列外其余暂时保留。

模式	分辨率@刷新率	长宽比	标准类型
1	640 * 480 @ 60Hz	4:3	工业标准(启动)
…			

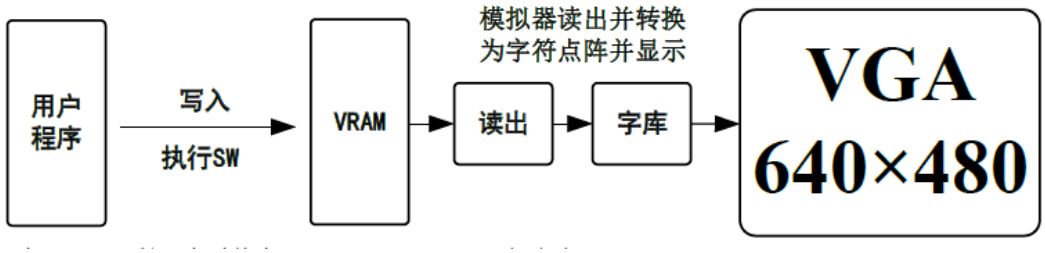
模式寄存器缺省状态为 MODE=1100_0…0_000_0001。可以在**配置 – VGA 模式控制寄存器**处进行修改。

2. 文本模式

文本模式下，模拟器将读写 **VRAM_TEXT** 外设的内存总线。每个字符将占用 2 字节的数据，格式如下：

每个字符按照从左到右、从上到下的顺序储存在内存中。其中 8*8 西文字库一行 80 字符，60 行。16*16 汉字字库一行 40 字符，30 行。

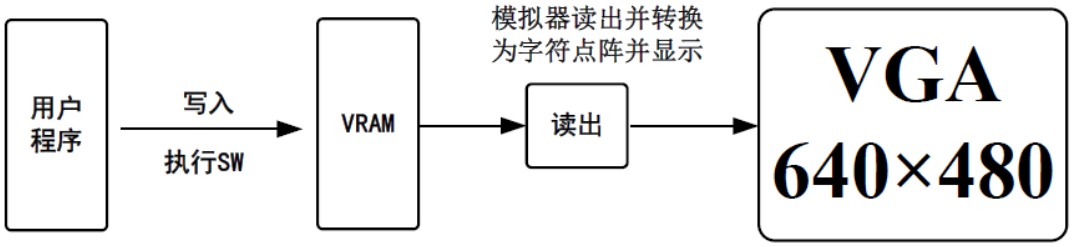
模拟器的执行流程如下：



3. 图形模式

图形模式对应的外设总线为 **VRAM_GRAPH**。每个像素 16 位，**低 12 位为像素点 RRRR_GGGG_BBBB**，高四位为待定属性。VGA 储存 640*480 点像素，按照从左到右、从上到下的顺序储存在 VRAM_GRAPH 中。

模拟器的执行流程如下：



3.4.2 七段显示模拟

七段码对应的外设总线为 **7-Segment**。模拟器只使用总线地址的前 24 位，当 SW 地址的前 24 位与预定义的总线地址前 24 位相等时，进行以下操作：

1. 若滑动开关 SW[0]=1，SW 输出内容内容在七段码上以 16 进制方式显示；
2. 若滑动开关 SW[0]=0，则需要利用 SW 地址的后 8 位：若为偶数地址，SW 输出内容在低四位七段码上以图形点阵形式显示；若为奇数地址，SW 输出内容在高四位七段码上以图形点阵形式显示。

3.4.3 GPIO 输出（写）模拟（LED 显示）

当 SW 指令地址与外设 **GPIO** 地址相等时，SW 指令输出内容对应如下：

a. $D_{17\sim2}$ 16 位输出对应 LED 显示($LED_{15\sim0}$)。

b. $D_{1\sim0}$ 二位控制硬件计数器对应计数器 4 个通道 $counter\_ch_{0\sim3}$。参见硬件计数器模拟。**（暂未实现）**

c. $D_{31\sim18}$ 14 位输出对应硬件光标。其中 $Cursor\_x[6:0]=D_{24\sim18}$，$Cursor\_y[6:0]= D_{31\sim25}$。**（暂未实现）**

3.4.4 GPIO 输入（读）模拟（滑动开关读取）

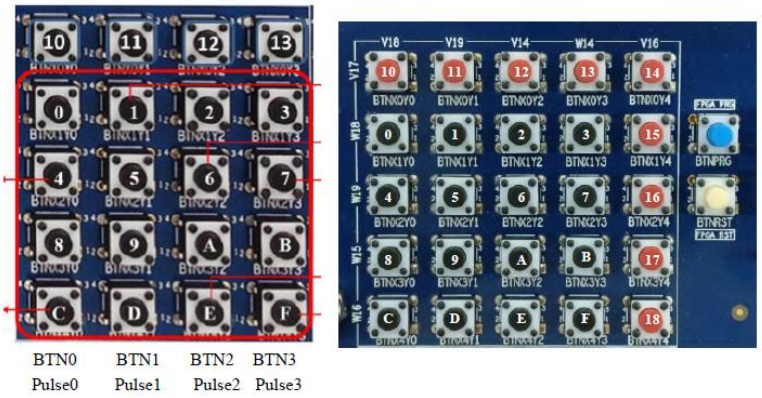
当 LW 地址与外设 **GPIO** 地址相等时，LW 指令模拟执行结果为：将下列信息写入目的寄存器：

{counter0_out, counter1_out, counter2_out, 8'h000, BTN[4:0], SW[15:0]}

其中：SW[15:0]为滑动开关当前状态，BTN[4:0]为阵列键盘列状态值（SWORDV2 是 4 位，SWORDV4 是 5 位，一律按 5 位输入）。counter0_out,counter1_out,counter2_out 为硬件计数器 0，1，2 溢出位**（暂时只实现 counter0_out）**。BTN 定义参见阵列键盘模拟。

注意：SW[0]为特殊按键，具有切换七段码图形/文字显示模式的功能。

3.4.5 阵列键盘模拟



（a）SWORD-V2　　(b) SWORD-V4

每个按键对应一个 KCODE[4:0]。对应关系见上图。

当 LW 地址与外设 **Button** 地址相等时，LW 指令模拟执行结果为：将下列信息写入目的寄存器，同时 KRDY 清零：

{ KRDY,21'h000000, BTN[4:0], KCODE [4:0]}

关于 KRDY 说明：当键按下时 KRDY=1，读取后 KRDY=0，此时只有按键释放后才能下次按键。也就是不支持连续按键。

若没有读取键盘则 KRDY 一直保持 1 不变，无论按键是否释放，但 KCODE [4:0]是最近按下的键编码。

3.4.6 硬件计数器模拟

1. 计数器说明

在实际的 CPU 上，硬件计数器用于储存当前的时钟周期数。模拟器使用当前系统的真实时钟来模拟器计数器。用户可以在**配置 – 自定义计数器频率**调整当前的时钟频率和计数器的分频数（即 clk_div[x]）。

硬件计数器为 32 位计数器，兼容 8253，共三个计数通道，其中溢出信号分别为 counter0_out、 counter1_out,和 counter2_out。暂模拟通道 0，工作方法 0，参考 8253。

三个计数器和控制豁口由 GPIO 输出最底 2 位识别，其对应如下：

counter_ch=00，对应计数器 0。读写对应计数器 0

counter_ch=01，对应计数器 1。读写对应计数器 1

counter_ch=10，对应计数器 2。读写对应计数器 2

counter_ch=11，对应控制寄存器。读写对应计数器控制器。

控制寄存器格式 24 位，每个通道 8 位(可参考 8253)：

Counter0_Ctrl [7:0]= $D_{7\sim0}$ = XX M2 M1 M0 X

Counter1_Ctrl [7:0]= $D_{15\sim8}$ = XX M2 M1 M0 X

Counter2_Ctrl [7:0]= $D_{23\sim16}$ = XX M2 M1 M0 X

2. 计数器模拟

暂时模拟通道 0 工作方式 0，**以上参数可以暂时忽略。**用户通过对外设 **Counter** 地址的 SW 操作，可以将 SW 的数据写入计数器（无论当前计数是否结束）。之后计数开始，按照特定的计数器频率将计数器值-1，直到减到 0，输出 counter0_out=1（见 GPIO 输出）并一起

保持，直到下一次 SW 指令重写计数器值。

Counter 的 LW 操作，将会把计数器值写入目的寄存器。

3.4.7 PS2 键盘模拟

当用户点击 VGA 窗口或右下角 Log 信息输出窗口时，模拟器开始捕捉键盘输入信号作为 PS2 键盘模拟输入。模拟器维护了一个键盘消息队列。当键盘按下时，模拟器将发送键盘通码；当键盘松开时，模拟器将发送键盘断码(**参考附录 B：PS2 键盘协议**)，并将扫描码按顺序存放在键盘消息队列的尾部。

当 LW 指令与外设 **PS2 Keyboard** 地址相等时，将会从键盘消息队列的头部提取出 8 位，写入 LW 指令目的寄存器，并将这 8 位从队列中移除。写入格式如下：

{ps2_ready, 23'h0, key};

其中 ps2_ready 是扫描码有效，当队列为空时即清零。key 为队列头部的 8 位。

3.5 调试说明

对于汇编代码，模拟器有两种调试方式：全速运行和单步调试。

3.5.1 全速运行

全速运行模式下，模拟器将用最快的速度运行汇编代码。运行过程中，用户可以随时选择暂停（F4）。

为了性能考虑，对于外设显示的更新并不是实时的，而是以一定的频率（暂定为 25Hz）进行刷新。

3.5.2 单步调试

单步调试有两种模式：单步执行（F7）和跳过 Jump 的单步执行（F8）。对于后者，模拟器将跳过一切 Jump 指令，包括 branch 指令、j、jr、jal、jalr 等。用户可以在内存查看界面查看当前指令执行的位置。

四、模拟器编译方法

模拟器已在 github 上开源，地址：https://github.com/Keytoyze/Sword_emulator。如果遇到问题或 bug，可提交 issue 进行反馈。

4.1 编译环境

模拟器基于 mvn 来进行编译，因此需要先安装 mvn。如果使用的 Java 版本不低于 11，还需要安装 JavaFX SDK（https://openjfx.io/）。

4.2 编译 jar 包

在项目根目录使用 `mvn jfx:jar` 来编译 jar 可执行文件，编译后的文件位于/target/jfx/app 下。

4.3 编译 native 包

在项目根目录使用 `mvn jfx:native` 来编译本地平台上的可执行文件，编译后的文件位于/target/jfx/native 下。

五、关于本项目

本项目为作者在学习浙江大学“计算机组成”课程时开发的，用于进一步学习计算机硬件知识，以及方便调试最后的大作业。本项目遵循 GNU General Public License v3.0 开源协议，如有问题，请提交 issue 或 PR。也欢迎参与完善本项目。

附录 A：模拟器支持的 MIPS 子集

ADD

ADDI

ADDIU

ADDU

AND

ANDI

BEQ

BEQL

BGEZ

BGEZAL

BGEZALL

BGEZL

BGTZ

BGTZL

BLEZ

BLEZL

BLTZ

BLTZAL

BLTZALL

BLTZL

BNE

BNEL

DIV

DIVU

J

JAL

JALR

JR

LB

LBU

LH

LHU

LL

LUI

LW

MFHI

MFLO

MTHI

MTLO

MULT

MULTU

NOR

OR

ORI

SB

SC

SH

SLL

SLLV

SLT

SLTI

SLTIU

SLTU

SRA

SRAV

SRL

SRLV

SUB

SUBU

SW

XOR

XORI

暂不支持：浮点运算与中断相关的指令

附录 B：PS2 键盘协议

KEY	通码	断码	KEY	通码	断码	KEY	通码	断码
A	1C	F0 1C	9	46	F0 46	[	54	F0 54
B	32	F0 32	`	0E	F0 0E	INSERT	E0 70	E0 F0 70
C	21	F0 21	-	4E	F0 4E	HOME	E0 6C	E0 F0 6C
D	23	F0 23	=	55	F0 55	PG UP	E0 7D	E0 F0 7D
E	24	F0 24	\	5D	F0 5D	DELETE	E0 71	E0 F0 71
F	2B	F0 2B	BKSP	66	F0 66	END	E0 69	E0 F0 69
G	34	F0 34	SPACE	29	F0 29	PG DN	E0 7A	E0 F0 7A
H	33	F0 33	TAB	0D	F0 0D	U ARROW	E0 75	E0 F0 75
I	43	F0 43	CAPS	58	F0 58	L ARROW	E0 6B	E0 F0 6B
J	3B	F0 3B	L SHFT	12	F0 12	D ARROW	E0 72	E0 F0 72
K	42	F0 42	L CTRL	14	F0 14	R ARROW	E0 74	E0 F0 74
L	4B	F0 4B	L GUI	E0 1F	E0 F0 1F	NUM	77	F0 77
M	3A	F0 3A	L ALT	11	F0 11	KP /	E0 4A	E0 F0 4A
N	31	F0 31	R SHFT	59	F0 59	KP *	7C	F0 7C
O	44	F0 44	R CTRL	E0 14	E0 F0 14	KP -	7B	F0 7B
P	4D	F0 4D	R GUI	E0 27	E0 F0 27	KP +	79	F0 79
Q	15	F0 15	R ALT	E0 11	E0 F0 11	KP EN	E0 5A	E0 F0 5A
R	2D	F0 2D	APPS	E0 2F	E0 F0 2F	KP .	71	F0 71
S	1B	F0 1B	ENTER	5A	F0 5A	KP 0	70	F0 70
T	2C	F0 2C	ESC	76	F0 76	KP 1	69	F0 69
U	3C	F0 3C	F1	5	F0 05	KP 2	72	F0 72
V	2A	F0 2A	F2	6	F0 06	KP 3	7A	F0 7A
W	1D	F0 1D	F3	4	F0 04	KP 4	6B	F0 6B
X	22	F0 22	F4	0C	F0 0C	KP 5	73	F0 73
Y	35	F0 35	F5	3	F0 03	KP 6	74	F0 74
Z	1A	F0 1A	F6	0B	F0 0B	KP 7	6C	F0 6C
0	45	F0 45	F7	83	F0 83	KP 8	75	F0 75
1	16	F0 16	F8	0A	F0 0A	KP 9	7D	F0 7D
2	1E	F0 1E	F9	1	F0 01	]	58	F0 58
3	26	F0 26	F10	9	F0 09	;	4C	F0 4C
4	25	F0 25	F11	78	F0 78	’	52	F0 52
5	2E	F0 2E	F12	7	F0 07	,	41	F0 41
6	36	F0 36	PRNTSCRN	E0 12 E0 7C	E0 F0 7C E0 F0 12	.	49	F0 49
7	3D	F0 3D	SCROLL	7E	F0,7E	/	4A	F0 4A
8	3E	F0 3E	PAUSE	E1 14 77 E1 F0 14 F0 77	-NONE-			

注：由于系统原因，模拟器无法区分左右的 ctrl、alt、shift 键，因此统一输出左边按键的扫描码。