计算机组成原理 大实验报告

计53 王润基 杨跻云 张耀楠

【奋战三星期 做台计算机】2017.11.14 - 2017.12.10

实现的功能

- 流水线CPU，最高频率40MHz
  - 支持25条基本指令和5条扩展指令
  - 支持外部中断
  - 取指Cache，缓解结构冲突
- 扩展功能
  - Boot：开机自动将程序从Flash拷贝到RAM
  - VGA和键盘
  - 实时显示丰富的调试信息，支持断点和单步调试
  - HardTerm：将Term移植到板上，可直接用键盘操作
  - 支持显存，可编程控制显示内容
  - 简单的MIPS16e编译器

各模块简介

基础头文件

- Base：定义了所有的基础类型、常数、函数
- Show：定义了渲染器所需的格式化toString函数

CPU模块

- IF取指

根据上周期PC+1和ID跳转信号，确定本周期PC，读RAM2取指。

如果当前RAM2被MEM占用，则本周期不能取指，向Ctrl给出暂停信号。

下一周期恢复取指，同时向Cache写入PC和指令，之后再遇到冲突时即可直接读Cache取指。

- IFCache取指缓存

  内部有8个寄存器，存储PC+指令对。支持插入、更新、查询表项。

- ID译码

  输入PC+1和指令，识别指令、读寄存器，输出各种控制信号

- EX执行

  只有一个ALU

- MEM访存

  读入EXE中运算器的计算结果，以及是否读/写RAM，是否写某个寄存器的相关信号，进行处理并传递给IO模块和Reg模块。

  后期，访存前的组合逻辑被移入EX阶段，以缓解访存的时间压力。

- Reg（RB）

  主要是实现了寄存器。为了方便，特殊寄存器也统一编号，寄存器数组(15 downto 0)。根据传入的信号，可以读取某两个寄存器的值、修改某个寄存器，以及修改R6和IH寄存器(处理中断时)。

- 触发器（ID_IF IF_ID ID_EX EX_MEM）

  纯边沿触发器，支持放过、清空、暂存、读取，用来单步调试。

- Ctrl

  接收处理内部各模块暂停请求，以及外部控制信号，用来单步调试。

- 中断的实现

  实现了外部中断。当中断信号给到cpu(我们具体展示的中断信号只有某几种键盘事件)，cpu根据IH寄存器的高位判断目前是否可以中断。如果可以中断，修改IH进入核心态并把中断码也写入IH寄存器的某几位，拦截IF到ID旁路的内容实现在下一周期跳转到中断程序，并把当前PC保存到R6。

  在中断程序中，根据中断码进行不同的处理，并最后跳转到R6(回到用户程序)/R7(结束用户程序)。这一部分使用汇编实现。原本的中断程序默认中断时把各种信息都压入栈中，我们认为这一设计极不科学，所以做了较大修改。

IO模块（RamUartCtrl）

协调处理所有的IO请求。

控制的设备接口：

- RAM1
- RAM2
- 串口1
- 串口2
- 1个数据缓冲区

接受的IO请求接口：（优先级依次递减）

- Boot模块：写RAM2（指令区）
- CPU的MEM模块：读写RAM1/RAM2/串口1/串口2/数据缓冲区
- CPU的IF模块：读RAM2（指令区）
- Renderer模块：读RAM1（显存区）

限于CPU要求在1个周期内完成 [提出请求-访存-拿到结果] 的全过程，因此模块主体是纯组合逻辑，写信号的下拉操作利用时钟信号后半周期完成。唯一涉及的时序逻辑是读串口时会强行延迟20周期。

内存地址映射

- BF00/BF01：串口1 数据/标志
- BF02/BF03：串口2 数据/标志
- BF04/BF05：缓冲区 数据/标志
- E000-FFFF：显存区

Renderer模块

控制屏幕显示内容。输入VGA正在显示的像素坐标，输出像素颜色。纯组合逻辑。

在调试信息模式下，用大量的if语句判断当前显示区域，计算当前字符。然后读取FontROM，获得像素值。

在显示显存模式下，通过IO模块读取RAM1，解析像素值。

HardTerm模块

为了在硬件上实现Term，我们采用了输入输出流重定向的思想，引入数据缓冲区（DataBuffer）。

原始Kernel对串口的读写相当于数据输入输出流，而用户在电脑上使用Term则是字符输入输出流。Term的作用即是：读取用户的字符输入流，翻译成给Kernel的数据输入流，Kernel执行完毕后写入数据输出流，Term读取后格式化成可读的字符输出流，显示到屏幕上。

HardTerm即为一个用VHDL硬编码的Term模块，它面向四个数据流接口（DataBuffer），用一个大型状态机实现Term的功能。

```
PS2 => CharInBuf  => | Hard | => ByteInBuf =>  | CPU |
       CharOutBuf <= | Term | <= ByteOutBuf <= |     |
```

其它辅助模块

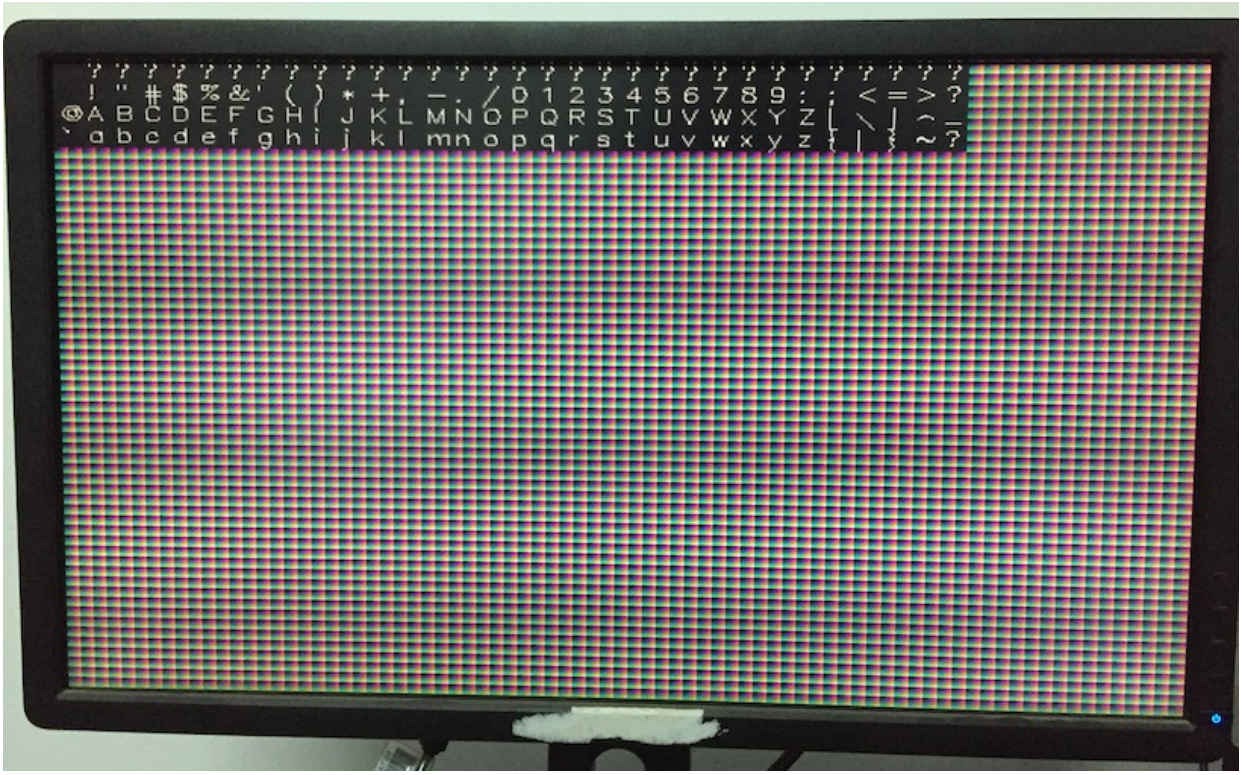
- vga_controller：开源代码。生成VGA控制信号和当前屏幕坐标。
- ps2_keyboard_to_ascii：开源代码。将PS2信号解码为按键信号和ASCII码。
- uart：提供代码。将串口2信号转化为和串口1相同的格式。
- Boot：读取Flash并告知IO模块当前要写RAM2的地址和数据
- DataBuffer：数据缓冲区，提供读写接口（信号格式同RAM）
- Shell：一种特殊的DataBuffer，有两个写接口，额外支持退格和回车
- AsciiToBufferInput：将ASCII信号转化为Buffer写信号
- FontReader：从片内ROM中读取字体点阵数据
- PixelReader：将像素块坐标转化为RAM1地址，委托IO模块读取显存

进程和问题

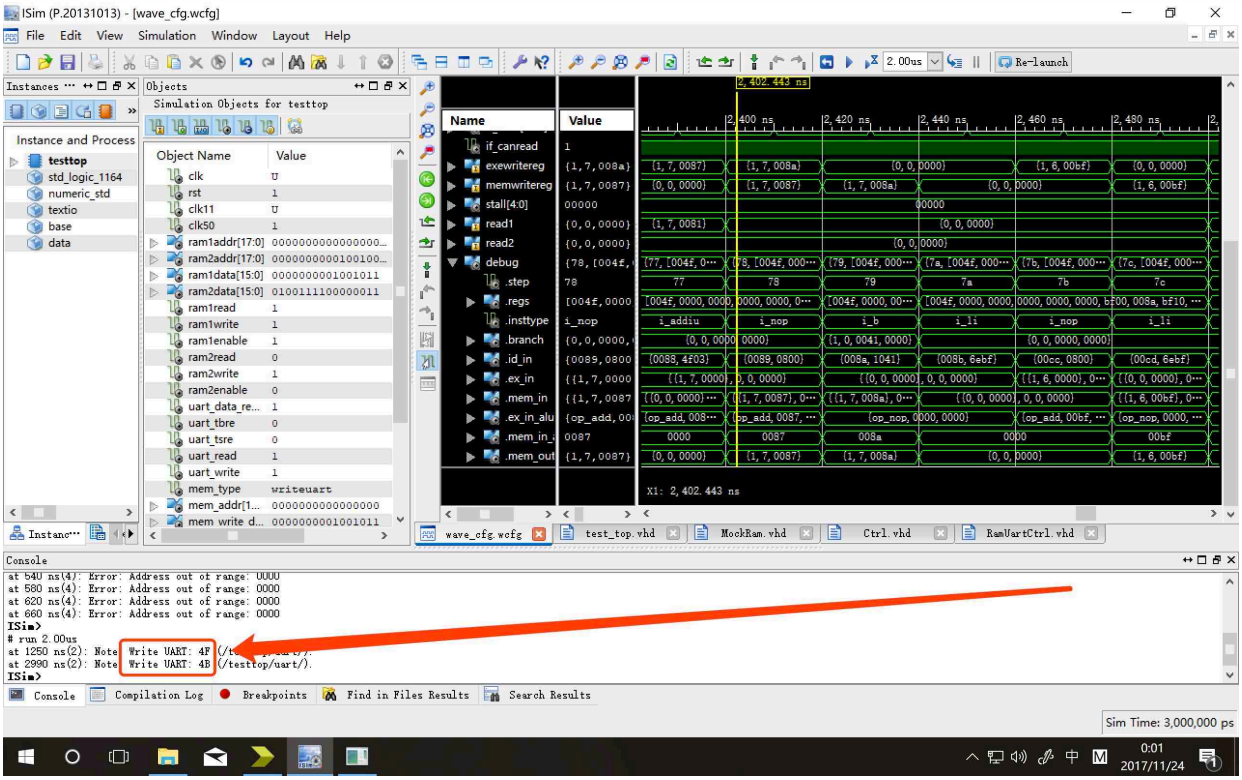
奋战三星期时间表

		周一	周二	周三	周四	周五	周六	周日
9周			布置实验			数据通路检查		VGA/键盘上板
	wrj				设计数据通路	完成各模块接口 PC/IF实现	ALU测试 VGA/键盘模块	寄存器、ID基础测试
	zyn					->	->	->
	yjy					->	ALU实现	ID基础指令实现
10周			控制器设计检查			说明扩展，初步代码检查	内存串口模块上板，流水线上板	
	wrj	->	->	MEM测试 Ctrl实现	整体测试	->		单步+连续模式
	zyn	->	寄存器实现	->	MEM模块实现	->		
	yjy	->	->	ID必要指令	ID完整实现	->	->	->
11周		上板调监控程序	上板调监控程序	监控程序跑通 性能测试		Boot		
	wrj	IOLogger	->	串口2	IFCache		COM3	显存
	zyn	Boot	->	Boot++			扩展指令测试程序	
	yjy	ID完整测试	调监控程序					
12周				COM3 显存		收尾测试		实验检查
	wrj			Shell	->	->	->	生命游戏
	zyn				->	->	->	中断
	yjy			->	->	->	->	HardTerm

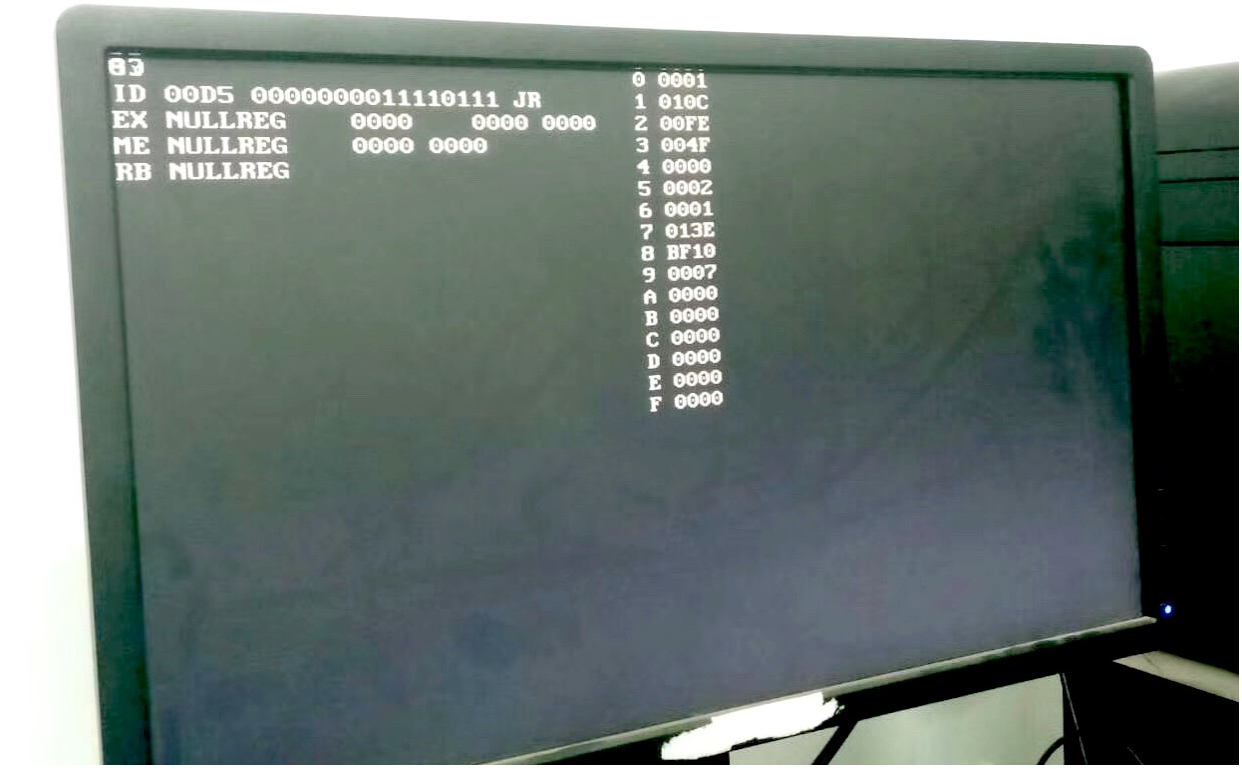
第9周周日，第一版VGA。VGA引脚反了，调了一小时。



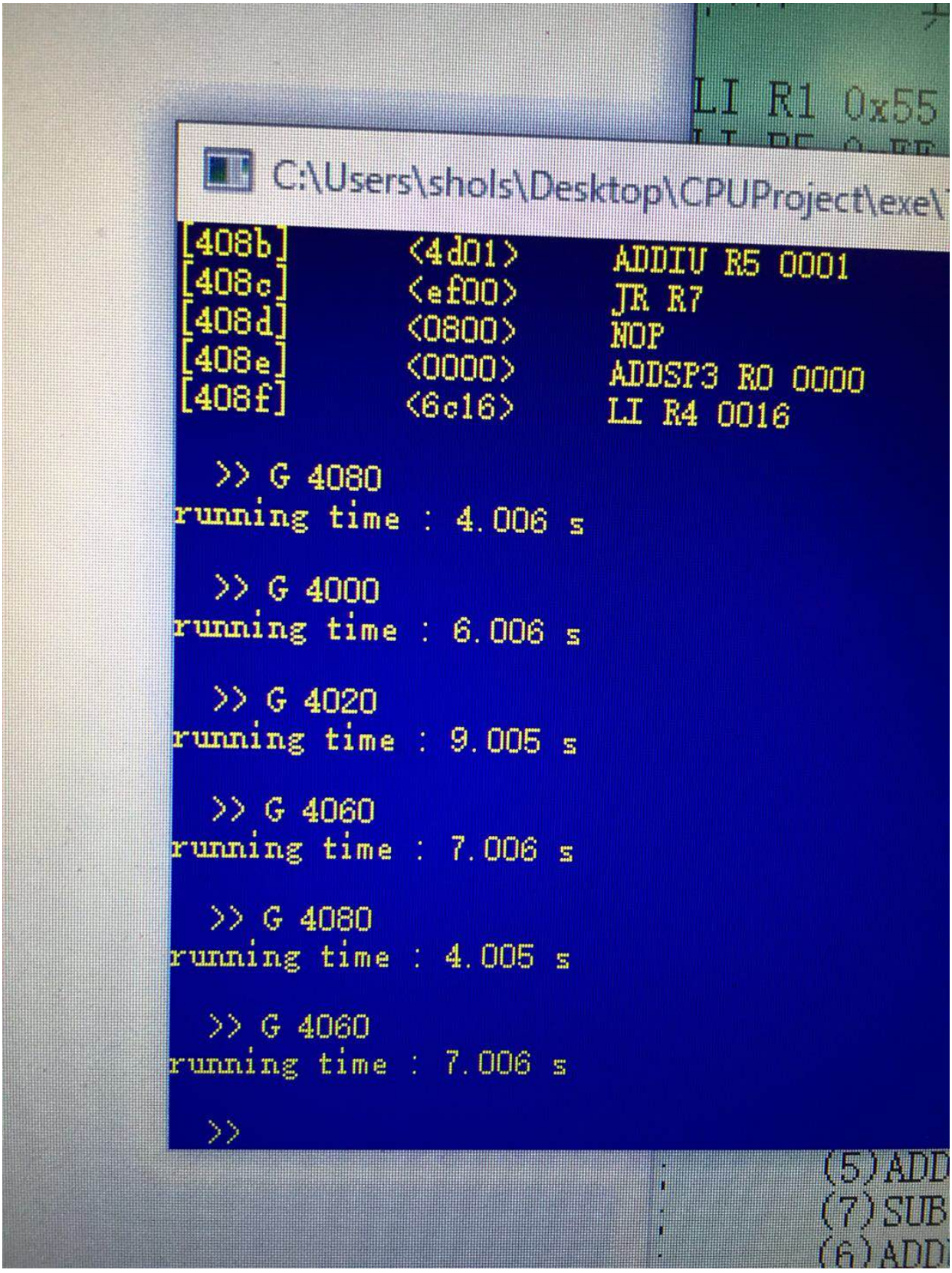
第10周周四，在仿真中跑出OK。



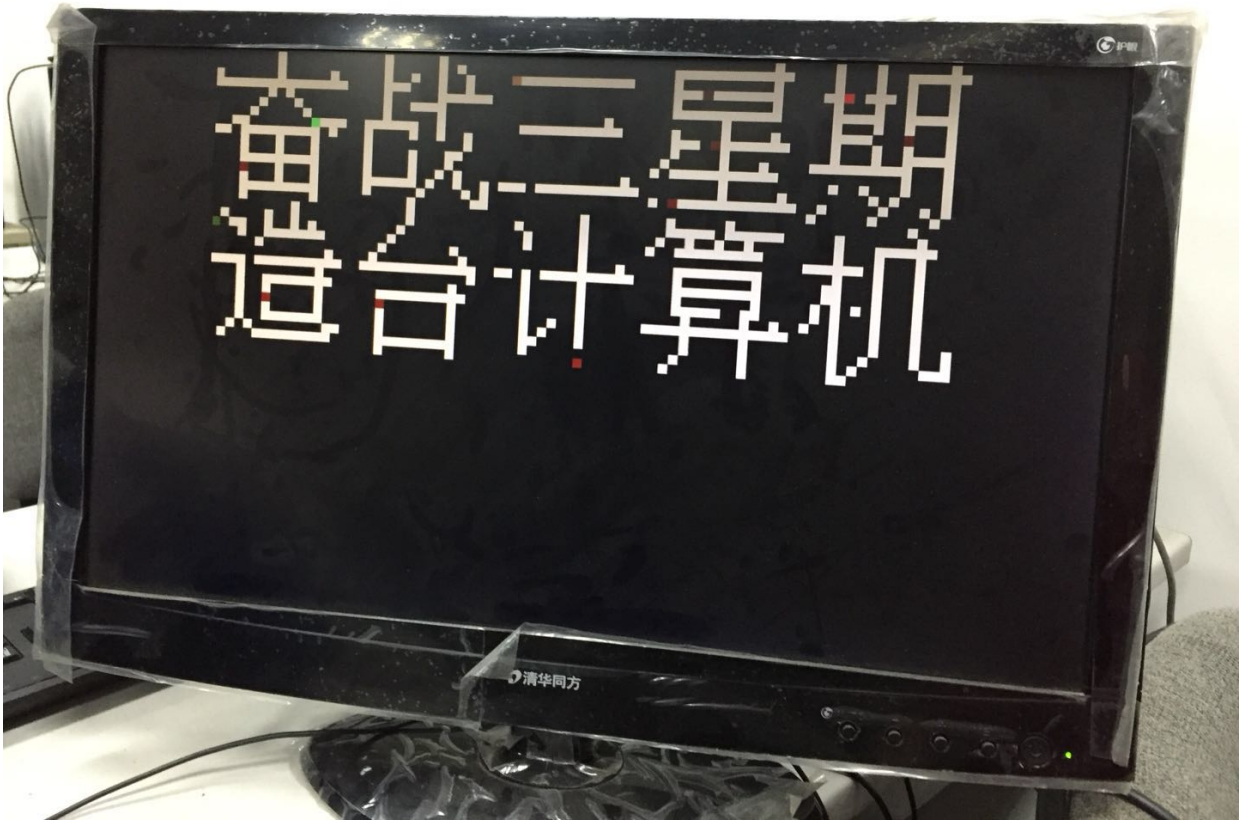
第10周周日，第一版调试信息界面，可以板上输出OK。



第11周周三，监控程序跑通，25MHz性能测试。



第12周周日，生命游戏，显示汉字，实现中断。



生命游戏用汇编编写。由于人肉汇编需要大量样板代码，于是同时用python编写了一个简单的编译器，简化了赋值、访存、读写显存等常用操作。

显示“奋战三星期造台计算机”则是最后的一个小品。首先使用一份开源程序将汉字转成bmp图片，然后用python生成对显存赋值的汇编代码，然后直接复制到Term中下载到RAM里运行。由于串口通信存在固有Bug，因此个别像素没有正常显示。