BuggyOS 手册

计算机系 171860508 张天昀
计算机系 171860592 吕云哲

2018 年 12 月 14 日

目录

1 架构简介

1.1 寄存器

BuggyOS 共有 32 个 32 位寄存器。其中前两个寄存器为保留寄存器。

- 0 号寄存器为零寄存器，只读不写，数据始终为 0。
- 1 号寄存器为地址寄存器 (`at`)，用于保存内存地址。
- 其余寄存器均可任意使用。

为了实现寄存器读写功能，寄存器模块有两个读端口和一个写端口。每当时钟周期到来，寄存器模块将读地址的数据加载，并根据写信号判断是否写入新的值。

1.2 数据存储器

BuggyOS 使用 IP-RAM 实现数据存储，共有 65536 个 32 位的存储单元，存储地址由 `0x1000 0000` 开始。
栈区使用一个额外的 RAM 存放数据，并在控制单元内根据地址判断 `rdata` 应该取哪个存储器的数据。

- `0x1000 0000 - 0x1000 20D0`（共 8400 字节）的地址为 VGA 显示缓存。
  CPU 可通过修改该区域的内容来修改显示内容。
- `0x1000 20D0 - 0x1000 20D4`（共 4 字节）为 PS/2 键盘端口。
  当键盘输入有效时，键盘输入的对应 ASCII 码将会被送到内存该地址处。
- `0x1000 2100 - 0x1000 2300`（共 512 字节）为输入缓存。
  CPU 将键盘输入保存至该区域，来实现字符串处理。
- `0x1000 2400 - 0x1000 2a00`（共 1536 字节）为只读数据段，用于存放常量与字符串。
- `0x7fff f800 - 0x7fff ffff`（共 8192 字节）为栈区。

1.3 指令存储器

BuggyOS 使用 256x32 的寄存器实现指令存储,加载时从 `*.mips` 文件读取指令并写入内存。指令地址由 `0x0040 0000` 开始。

1.4 程序计数器

程序计数器从 `0x0040 0000` 开始计数，每次执行指令后，根据控制信号调整计时器的值。控制信号的种类、含义如下：

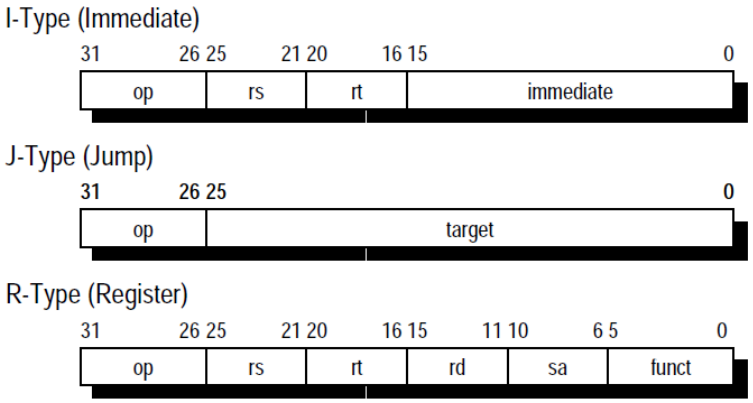
- `000`：非跳转指令，`PC = PC + 4`。
- `001`：无条件跳转至立即数指定位置，`PC = PC[31:28] + offset<<2`。
- `010`：跳转至寄存器，`PC = Rs`。

- `011`：分支跳转，具体见指令集 `BEQ/BNE` 指令。

程序计数器计算出 PC 的值后，PC 信号传入指令存储器，将对应的 32 位指令加载至 `instr` 端口，并传送给译码器。

1.5 译码器

译码器首先根据传入的指令判断 R、J、I 类型，并分解为对应的地址、功能、偏移、寄存器代码。



图三种类型指令的组成

随后，根据不同的指令，设置不同的控制信号。控制信号的种类如下：

- `mem_wren`：只对 `SW` 指令有效，控制是否将数据写入数据 RAM。
- `reg_wren`：只对跳转、`SW` 指令**无效**，控制是否将数据写入寄存器。
- `reg_dmux_sel`：只对 `LW` 指令**无效**，详情见下一节详细说明。
- `reg_rmux_sel`：只对 R 类指令有效。
- `reg_is_upper`：只对 `LUI` 指令有效，设置后数据左移 16 位。
- `alu_imux_sel`：只对 I 类指令有效。
- `alu_op`：ALU 控制信号。
- `pc_control`：程序计数器控制信号。

1.6 D/R/I 选择器

为了应对不同指令数据来源不同的情况，BuggyOS 使用了 3 个二选一选择器来进行数据的选择。

- D-MUX：当 `reg_dmux_sel` 信号有效时，把 ALU 的运算结果写入寄存器，否则将内存读取的结果写入寄存器。
- R-MUX：当 `reg_rmux_sel` 信号有效时，使用 Rt 寄存器保存数据，否则使用 Rd 寄存器保存数据。
- I-MUX：当 `alu_imux_sel` 信号有效时，使用符号拓展的低 16 位指令，即立即数作为第二个操作数，否则使用 Rt 寄存器作为第二个操作数。

1.7 逻辑运算单元

逻辑运算单元根据控制信号 op，对输入的操作数 rs 和 rt 进行运算，结果存入 rd，并据结果设置标志位 zf 和 of。控制信号的种类如下：

- 0001：按位与。rd = rs & rt。
- 0010：按位或。rd = rs | rt。
- 0011：无符号数加。rd = (uns)rs + (uns)rt。
- 0100：按位异或。rd = rs $\oplus$ rt。
- 0101：按位或非。rd = ~(rs | rt)。
- 0110：无符号数减。rd = (uns)rs - (uns)rt。
- 0111：有符号数比较。若 rs < rt, rd 为 1，否则为 0。
- 1000：逻辑或算术左移。rd = rt << sa。
- 1001：逻辑右移。rd = (uns)rt >> sa。
- 1010：算术右移。rd = (int)rt >>> sa。
- 1011：有符号数加。rd = (int)rs + (int)rt。
- 1100：有符号数减。rd = (int)rs - (int)rt。
- default: 不运算。rd = rt。

标志位的含义如下：

- zf：零标志。运算结果为 0 时置位，否则清零。
- of：溢出标志。有符号数运算且发生溢出时置位，否则清零。

1.8 控制单元

本单元负责各个模块之间的信号、数据交互，以及处理 IO 模块发来的 RAM 读写信号。几乎全为实例化内容。

2 CPU 调试指南

2.1 简介

BuggyOS 使用开关、LED、七段数码管来进行中间内容的输出。当开关为全部低电平时，七段数码管输出当前的指令地址（从 0x0040 0000 开始）。

2.2 查看解码信息

10 个 LED 输出标志着当前指令的解码状态，按从左到右的顺序，分别为：

- 第 9 位：`mem_wren` 内存写入信号（SW 指令判断）
- 第 8 位：`reg_wren` 寄存器写入信号
- 第 7 位：`reg_dmux_sel` 寄存器写入 ALU 运算结果信号
- 第 6 位：`reg_rmux_sel` 使用 Rt 寄存器信号（R 类型指令判断）
- 第 5 位：`alu_imux_sel` 第二操作数为立即数信号（I 类型指令判断）
- 第 4-1 位：`alu_op` ALU 控制信号
- 第 0 位：`pc_control != 0` 发生跳转信号（J、B 类型指令判断）

2.3 查看变量

调整左侧 5 个开关，可以查看 CPU 运算的中间变量。默认情况下 6 个数码管显示当前程序计数器的值；当开关任意一位有效时，七段数码管左侧四位输出查看的变量的值（的低 16 位），右侧两位继续显示程序计数器。按下 KEY[2]，即可查看当前变量的高 16 位。

BuggyOS 根据开关的组合来决定输出的变量内容。注意左侧的开关享有更高的优先级，不同的开关组合输出的内容如下：

- 第 9 位有效：输出 Rs 寄存器（ALU 第一操作数）的值。
- 第 8 位有效：输出 ALU 的第二操作数（Rt 或立即数）的值。
- 第 7 位有效：输出 ALU 的运算结果。
- 第 6 位有效：输出写入寄存器的值（运算结果或内存数据）。
- 第 5 位有效：输出内存读取的数据。
- 高 5 位均无效：根据低 5 位开关决定寄存器地址，输出对应的寄存器的值。

2.4 指令的执行与重置

在调试模式下，除了存储器模块使用系统时钟 `sys_clk`，其他时钟都使用人工输入。

按下 KEY[0]，就会经过一个时钟周期，执行 1 个指令。

如果需要重启系统，需要同时按下 KEY[3] 和 KEY[0]，将程序计数器置为 `0x0040 0000`。注意此版本下重置系统不会清空寄存器的值。

3 软件设计指南

3.1 状态设计

系统共有四个主要状态：

- 闲置：不断向键盘端口询问是否有数据输入；
- 输入：将键盘端口的数据移动至缓冲区，并判断输入是否为回车，是则进入下一阶段；
- 判断：将缓冲区的字符串与指令进行匹配，如果匹配成功进入下一阶段，否则跳转至错误处理；
- 计算：运行指令对应的函数，运行时无视键盘的输入，运行完成后回到闲置状态。

3.2 寄存器的使用

结合软硬件设计需求和 MIPS 设计风格，寄存器的分配如下：

- `zero/at` （0-1 号）：分别为 0 寄存器和汇编保留寄存器，不使用；
- `v` 寄存器组（1-2 号）：用于保存函数调用返回值；
- `a` 寄存器组（4-7 号）：用于保存函数调用参数；
- `t` 寄存器组（9-15、24、25 号）：用于保存计算中间变量；
- `s` 寄存器组（16-23 号）：用于保存全局变量，如字符串位置等；
- `k` 寄存器组（26-27 号）：用于保存输出参数，即当前光标位置；
- `gp/sp/fp/ra` （28-31 号）：分别为全局指针、堆栈指针、帧指针（不使用）、返回地址。

注意：`gp` 寄存器用作缓冲区指针，初始化时其值为 `0x1000 2100`。

3.3 底层封装

系统设计应优先完成一下函数：

- `push/pop`：将 `s0` 压入栈/弹出栈至 `s0`；
- `_write`：输出一个字符到显存中；
- `print`：打印连续的字符串；
- `println`：打印连续的字符串然后换行；
- `newline`：清除当前一行内容；
- `clear`：清除屏幕所有内容；
- `term`：**清空一行数据**并打印终端符号，然后等待读入；

- `_read`：程序主循环，不断循环直到有字符输入；
- `strcmp`：字符串比较函数；
- `handle`：判断输入指令是否合法并跳转执行；

3.4 程序调用

函数调用时，调用者应负责保存数据。函数的参数应按顺序保存在 `a` 寄存器组中，返回值应使用 `v` 寄存器组保存。

函数调用时，使用 `JAL` 指令（注意不是 `JALR`），返回时使用 `JR $ra`。同时要注意避免调用者、被调用者对同一个寄存器的竞争。

调用者首先将 `$sp` 寄存器减去所需的大小，相当于申请栈区空间；然后再函数返回前将减去的部分补回，相当于释放栈区空间。

3.5 开机程序

注意：硬件并没有实现除 PC 以外的复位操作。

开机程序应保存在 `0x0040 0000` 位置。

开机时，首先将寄存器组复位成固定值，即 `$gp = 0x1000 2100`、`$sp = 0x7fff effc`、`$k0 = $k1 = 0x0000 0000`。然后进行一次循环清除屏幕显存内容。

最后打印欢迎提示，跳转至主循环，进入闲置状态。

4 指令集

以下内容，仅列举实现了的指令。由于没有实现所有的指令，本手册不带有所有指令的速查表。

4.1 指令的 Opcode 表

Special 表示该指令为 R 类型指令，需要根据功能代码在功能码表中寻找对应指令。

十六进制	高三位	低三位	指令	功能
0x00	000	000	Special	R-type instructions
0x02	000	010	J	PC = PC[31:28] || offset<<2
0x03	000	011	JAL	GPR[31] = PC + 4, then J
0x04	000	100	BEQ	if Rs==Rt, PC += (int)offset
0x05	000	101	BNE	if Rs!=Rt, PC += (int)offset
0x08	001	000	ADDI	Rt = Rs + Imm
0x09	001	001	ADDIU	Rt = Rs + UImm
0x0C	001	100	ANDI	Rt = Rs & (0 << 16 || Imm)
0x0D	001	101	ORI	Rt = Rs | Imm
0x0E	001	110	XORI	Rt = Rs ^(uns)Imm

0x0F	001	111	LUI	Rt = Imm << 16
0x23	100	011	LW	Rt = Mem[Rs+offset]
0x2B	101	011	SW	Mem[Rs+offset] = Rt

4.2 R-type 类型的 Function 码表

十六进制	高三位	低三位	指令	功能
0x00	000	000	SLL	Rd = Rt << sa
0x02	000	010	SRL	Rd = (uns)Rt >> sa
0x03	000	011	SRA	Rd = (int)Rt >> sa
0x08	001	000	JR	PC = Rs
0x20	100	000	ADD	Rd = Rs + Rt
0x21	100	001	ADDU	Rd = Rs + URt
0x22	100	010	SUB	Rt = (int)Rs - (int)Rd
0x23	100	011	SUBU	Rt = (uns)Rs - (uns)Rd
0x24	100	100	AND	Rd = Rs & Rt
0x25	100	101	OR	Rd = Rs | Rt
0x26	100	110	XOR	Rd = Rs Rt
0x27	100	111	NOR	Rd = (Rs | Rt)
0x2A	101	010	SLT	if ((int)Rs<(int)Rt) Rd = 1