PPCA2017 小作业：带 5 级流水的 MIPS 模拟器

2017 年 6 月 26 日

目录

1 注意

- 注意处理转义符
- 注意处理注释符号 #
- 虽然数据简单, 但还是尽量保证鲁棒性.

2 时间/要求

见课程主页.

3 初始化

- $sp(30 号寄存器) 为数据栈顶的地址
- 其他 31 个寄存器和 lo, hi 寄存器以及数据空间都默认为 0
- 所需要的数据空间大小 4M(4 * 1024 * 1024 byte)

4 正确性

要求实现的是一个能够正确解释 MIPS 汇编语言中与整数运算有关的一个子集的模拟器。关于程序的正确性方面，要求实现的模拟器有和 ~~SPIM~~ 标准结果相同答案.

4.1 要求实现的汇编语言特性

4.2 汇编器指令

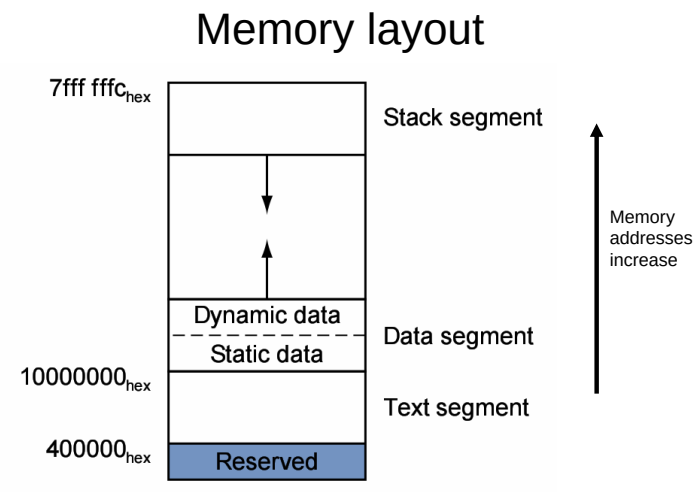

图 1: 内存分配

MIPS 的内存分配有一定顺序，栈空间从高地址开始到低地址，堆空间和静态空间从低地址开始到高地址，如下图，对于之前的那个例子，如果执行 `lw $t1, static_b`，则 `$t1` 的值是 10。在正式执行汇编程序之前，汇编器会读取整个文件，按照文件的顺序理解每一个语句，如果这条语句是汇编器指令，那么就会按照要求执行相应的操作，例如分配一定长度的内存，对于出现在 data 区的 Label，改 Label 指向的是这个 Label 后面一个 data 成员的内存地址。例如

```
        .word 5
static_b:
        .word 10
```

其中 `static_b` 在内存中指向如图2。

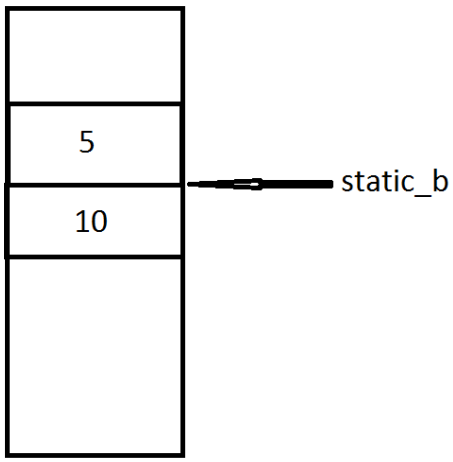

图 2: static_b 指向的位置

需要实现的 MIPS 汇编器指令如下:

.align n	把目前的内存的位置与 2^n byte 对齐.
.ascii str	存一个 string 到内存中, 但没有结束符.
.asciiz str	存一个 string 到内存中, 并且有结束符.
.byte b1, ..., bn	存 n 个 byte 到连续的一段内存中.
.half h1, ..., hn	存 n 个 halfWord 到连续的一段内存中.
.word w1, .. ., wn	存 n 个 Word 到连续的一段内存中.
.space n	分配 n 个 byte 到内存中
.data	数据区开始标志
.text	代码区开始标志

Label 由字母（大小写敏感，数字和下划线以及其他一部分的可见符号构成的字符串，每一个 Label 后面有一个冒号，可以理解这是一个指针.

4.3 算术和逻辑指令

下面提及的 Rdest 为存储结果的寄存器，Rsrc, Rsrc1, Rsrc2 为操作数所在的寄存器，Src2 表示这个操作数既可以是立即数也可以是一个寄存器中的数据（即寄存器标号):

- add Rdest, Rsrc1, Src2 — Rdest = Rsrc1 + Src2
- addu Rdest, Rsrc1, Src2 — (无符号)Rdest = Rsrc1 + Src2
- addiu Rdest, Rsrc1, Imm — (无符号)Rdest = Rsrc1 + Imm
- sub Rdest, Rsrc1, Src2 — Rdest = Rsrc1 - Src2
- subu Rdest, Rsrc1, Src2 — (无符号)Rdest = Rsrc1 - Src2
- mul Rdest, Rsrc1, Src2 — Rdest = Rsrc1 * Src2
- mulu Rdest, Rsrc1, Src2 — (无符号)Rdest = Rsrc1 * Src2
- mul Rdest, Src2 — 相乘, 低 32 位存在 lo, 高 32 位存在 hi
- mulu Rdest, Src2 — (无符号) 相乘, 低 32 位存在 lo, 高 32 位存在 hi
- div Rdest, Rsrc1, Src2 — Rdest = Rsrc1 / Src2
- divu Rdest, Rsrc1, Src2 — (无符号)Rdest = Rsrc1 / Src2
- div Rsrc1, Rsrc2 — lo = Rsrc1 / Src2, hi = Rsrc1 % Src2
- divu Rsrc1, Rsrc2 — (无符号)lo = Rsrc1 / Src2, hi = Rsrc1 % Src2
- xor Rdest, Rsrc1, Src2 — Rdest = Rsrc1 $\hat{}$ Src2
- xoru Rdest, Rsrc1, Src2 — (无符号)Rdest = Rsrc1 $\hat{}$ Src2
- neg Rdest, Rsrc — Rdest = Rsrc 取反
- negu Rdest, Rsrc — (无符号)Rdest = Rsrc 取反
- rem Rdest, Rsrc1, Src2 — Rdest = Rsrc1 % Src2
- remu Rdest, Rsrc1, Src2 — (无符号)Rdest = Rsrc1 % Src2

4.4 常数操作指令

- li Rdest, imm — Rdest = imm

4.5 比较指令

对于所有的比较指令，结果为真返回 1，结果为假则返回 0

- seq Rdest, Rsrc1, Src2 Rdest = Rsrc1 == Src2
- sge Rdest, Rsrc1, Src2 Rdest = Rsrc1 >= Src2
- sgt Rdest, Rsrc1, Src2 Rdest = Rsrc1 > Src2
- sle Rdest, Rsrc1, Src2 Rdest = Rsrc1 <= Src2
- slt Rdest, Rsrc1, Src2 Rdest = Rsrc1 < Src2
- sne Rdest, Rsrc1, Src2 Rdest = Rsrc1 != Src2

4.6 分支与跳转指令

每条 MIPS 的汇编指令都可以有一个或者多个 label，分支与跳转语句中的 label 就是汇编指令的 label，用于跳转到具有特定 label 的语句，注意 label 是全局唯一的

- b label goto label
- beq Rsrc1, Src2, label if (Rsrc1 == Src2) goto label
- bne Rsrc1, Src2, label if (Rsrc1 != Src2) goto label
- bge Rsrc1, Src2, label if (Rsrc1 >= Src2) goto label
- ble Rsrc1, Src2, label if (Rsrc1 <= Src2) goto label
- bgt Rsrc1, Src2, label if (Rsrc1 > Src2) goto label
- blt Rsrc1, Src2, label if (Rsrc1 < Src2) goto label
- beqz Rsrc, label if (Rsrc1 == 0) goto label
- bnez Rsrc, label if (Rsrc1 != 0) goto label
- blez Rsrc, label if (Rsrc1 <= 0) goto label
- bgez Rsrc, label if (Rsrc1 >= 0) goto label
- bgtz Rsrc, label if (Rsrc1 > 0) goto label
- bltz Rsrc, label if (Rsrc1 < 0) goto label
- j label goto label
- jr Rsrc goto 指令地址 in Rsrc
- jal label $31 = 下一条指令的地址, goto label
- jalr Rsrc $31 = 下一条指令的地址, goto 指令地址 in Rsrc

4.7 Load 指令

从指定的内存地址读取数据并且保存到指定的寄存器，内存地址以 `offset(Register)` 或者是 `label` 的形式给出，即，寄存器中的首地址 + 偏移量，偏移量**可正可负**.

- la Rdest, address　　Rdest = address
- lb Rdest, address　　Rdest = data[address : address + 1]
- lh Rdest, address　　Rdest = data[address : address + 2]
- lw Rdest, address　　Rdest = data[address : address + 4]

4.8 Store 指令

从指定的寄存器读取数据并且保存到指定的内存地址，内存地址以 `offset(Register)` 或者是 `label` 的形式给出.

- sb Rsrc, address　　data[address : address + 1] = Rsrc
- sh Rsrc, address　　data[address : address + 2] = Rsrc
- sw Rsrc, address　　data[address : address + 4] = Rsrc

4.9 数据移动指令

- move Rdest, Rsrc　　Rdest = Rsrc
- mfhi Rdest　　Rdest = hi
- mflo Rdest　　Rdest = lo

4.10 特殊指令

- nop　　啥也不做, 但得占五个周期
- syscall　　只需要模拟下面提到的几个系统调用即可

系统调用:

系统调用编号	说明	参数	结果
1	输出一个整数	$a0: 需要输出的整数	N/A
4	输出一个字符串，这个字符串要求以'\0' 结尾	$a0: 字符串的第一个字符的地址	N/A
5	读入一个整数	N/A	读入的整数保存在 $v0
8	读入一个字符串	$a0: 存放读入字符串的缓冲区 $a1: 读入最长的长度 +1 [1]	N/A
9	分配堆内存	$a0: 需要申请的连续的内存长度[2]	$v0: 申请结果，即内存片段的首地址[3]
10	结束运行	N/A	N/A
17	结束运行（有返回值）[4]	$a0: 程序运行结束的返回值	N/A

5 5 级流水划分

Instruction Fetch 取指令阶段, 获得从 PC 寄存器 (存储下一条指令的位置) 下一条要执行的指令.

Instruction Decode & Data Preparation 对 Instruction Fetch(IF)（取指令）阶段得到的指令进行解码，并且准备计算所需要的数据

Execution 执行完成这个条指令必要的运算，对于算术和逻辑运算，那么计算出运算结果，对于内存操作，那么计算出实际需要 load/store 的内存地址.

Memory Access 访问内存，完成 load/store 指令中要访问内存的部分。

Write Back 回写，把算术，或者逻辑运算的结果写到指定寄存器，把从内存中 load 到的数据写入指定的寄存器。对于分支语句，在这个阶段会改写 PC 寄存器。

5.1 寄存器读写顺序

由于流水线的存在，所以可能在同一个 CPU 时钟周期里面既要读寄存器又要写寄存器，先读还是先写会影响结果的正确性，所以我们规定，同一个 CPU 时钟周期里面先写寄存器，再读寄存器[5]

[19]如果要读入'abc'，那么 $a1 应该设置成 4 或者更大

[20]单位：字节

[21]注意返回的地址要求首地址对齐到 2^2

[22]C 语言中的 exit() 函数

[5]如果把一个周期分成两个半周期，那么可以简单的认为，前半周期写，后半周期读

5.2 对于 Hazard 的处理

流水线在处理的过程中由于各种原因会导致流水线不能像期望的那样正常工作，下面有三大类 Hazard[6]：

Structure Hazard 由于内存是总线上的设备，而总线的特性是同一时间[7]只能接受并执行一个任务请求，即不能再一个周期里面同时执行读取和写入（或者两个读取内存的操作）内存的操作，因此会造成流水线不能正常工作。举个例子，在每条指令的 IF 阶段都需要读取内存获取相应的内存，而第四个阶段 Memory Access(MEM) 中 load/store 指令也会读取或者写入内存，因此会造成总线的操作冲突，流水线因此停滞。

Data Hazard 由于寄存器的内容只有当指令执行到第五个 Write Back 阶段才会更新，所以，对于数据有依赖的情况，流水线就不能正常的运转，需要等在前一个指令完全完成操作才行, 注意：指令 *syscall* 是一个特殊的指令，这个指令会使用 $v0, $a0, $a1 这三个寄存器的值，所以如果出现了相关的数据依赖关系，那么会产生一个 Data Hazard。同时 syscall 本身是一个跳转指令（虽然它在执行中并不会程序员感觉到它是一个跳转指令），所以会造成 Control Hazard。至于 syscall 指令的结果应该在第 5 个阶段的时候和普通的指令一样写回寄存器

Control Hazard 对于分支语句和跳转语句之后的语句不能轻易的上流水线，所以就会造成流水线停滞。

当 Hazard 发生的时候，模拟器应暂停 (或者 lsd 指定) 的做法，等待 Hazard 消失，然后继续运行流水线。

6 Q&A

[6]在 CAAQA 的中文版中，Hazard 被翻译成冒险，但是几乎所有的人都认为这个翻译并不是那么合理，所以我在这里就直接使用英文而不是用中文翻译

[7]1 个时钟周期