

# Lab3-Extra

提交

更新

## 准备工作：创建并切换到 lab3-extra 分支

请在**自动初始化分支后**，在开发机依次执行以下命令：

```
$ cd ~/学号
$ git fetch
$ git checkout lab3-extra
```

初始化的 lab3-extra 分支基于课下完成的 lab3 分支，并且在 tests 目录下添  
加了 lab3\_ri 样例测试目录。

## 问题描述

在 Lab3 的课下实验中，我们主要介绍了 0 号异常（即时钟中断），以及异常的分发和处理过程。在本次 Extra 中，我们希望大家实现 10 号异常 RI 的处理函数。

在 See-MIPS-Run-Linux 中对 RI 的描述如下：

异常号	助记符	描述
10	RI	不认识的（或者非法的）指令码。

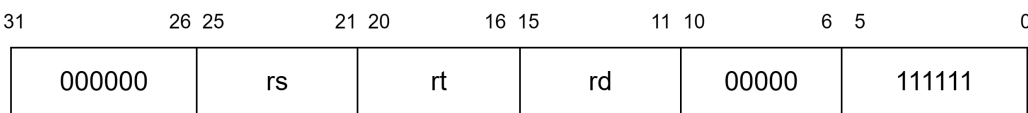
你需要利用 RI 异常的处理函数，实现本题新增的两条 **自定义指令**。

新增 **自定义指令** 的描述如下：

## pmaxub 指令

**指令格式：** pmaxub rd, rs, rt

**机器码** (高位对应 **高地址**，低位对应 **低地址**)：



寄存器值的对应字节。

伪代码：

提交

更新

```
rd = 0;
for (i = 0; i < 32; i += 8) {
    u_int rs_i = rs & (0xff << i);
    u_int rt_i = rt & (0xff << i);
    if (rs_i < rt_i) {
        rd = rd | rt_i;
    } else {
        rd = rd | rs_i;
    }
}
```

## cas 指令

指令格式： cas rd, rs, rt

机器码 (高位对应 高地址，低位对应 低地址)：

31	26 25	21 20	16 15	11 10	6 5	0
000000	rs	rt	rd	00000	111110	

指令描述：

从寄存器 rs 寄存器中存储的 地址 中取值，判断与 rt 寄存器的值是否相等。如果相等，则将 rd 寄存器的值存入 rs 寄存器中的地址。且不论是否相等，都要将 rs 寄存器中地址 原先存储的值 存入 rd 寄存器。

伪代码：

```
tmp = *rs;
if (*rs == rt) {
    *rs = rd;
}
rd = tmp;
```

## 题目要求

1. 建立 RI 异常的处理函数。

🔊

📄

➤

🔍

👤

对于本题中的 **自定义指令**，需按以上描述正确执行，执行完成后让地址寄存器指向下一条指令。

对于其他的非法指令（不符合题意且触发 RI 异常的指令），直接跳过该条指令即可。

**注意：**

无论触发异常的是 **自定义指令** 还是非法指令，你都需要在异常处理结束前让 EPC + 4，以免再次触发相同异常。

提交

更新

## 提示

1. 在完成异常处理函数时：
- 可以通过更改 CP0 中 EPC 寄存器（对应 Trapframe 结构体中的成员 cp0\_epc）的值（+4），使得异常恢复后执行的是下一条指令。
- 本题涉及对寄存器值的读取和修改。你需要访问或修改保存现场的 Trapframe 结构体（定义在 include/trap.h 中）中对应通用寄存器。
- 本题中涉及到对内存的访问，可以直接用虚拟地址进行访存（本质是通过 TLB 来获取物理地址），也可以先查询页表获得物理地址，然后再转换成 kseg0 段的虚拟地址来进行访存。

2. 可以在 kern/genex.S 中，用 BUILD\_HANDLER 宏来构建异常处理函数：

```
BUILD_HANDLER ri do_ri
```

3. 可以在 kern/traps.c 修改异常向量组，并实现异常处理函数：

```
void do_ri(struct Trapframe *tf) {  
    // 你需要在此处实现问题描述的处理要求  
}
```

## 题目约束

- 本题保证发生 RI 异常的指令（包括自定义指令和其他非法指令）的上一条指令不会是任意跳转指令，即保证发生 RI 异常的指令 **不会出现在延迟槽** 中。
- 本题中的两条指令在现实中都有其原型，如感兴趣可以课下自行查阅它们的硬件实现。

## 提交评测

2024-04-24 21:06:58 | 评测冷却: 137s



样例中，包含对 `pmaxub`、`cas` 和未定义指令的三个测试点，其内容分别如下。



对于 `pmaxub` 指令，当 `$t8 = 0x12345678`，`$t9 = 0x87654321` 时，执行如下指令，应得到 `$t7 = 0x87655678`，且 `$t8` 和 `$t9` 的值保持不变。

```
pmaxub $t7, $t8, $t9
```

更新

对于 `cas` 指令，当 `$t8` 保存的地址对应的值为 3，`$t9 = 3`，`$t7 = 5` 时，执行如下指令，应得到 `$t8` 保存的地址对应值为 5，`$t7 = 3`。

```
cas $t7, $t8, $t9
```

对于未定义指令，你只需要跳过即可，保证内存和各个寄存器的值都不发生变化（除了 EPC 寄存器），正确输出如下。

```
env 00000800 reached end PC: 0x00400180, $v0 = 0x00000000
[00000800] free env 00000800
i am killed ...
```

你可以使用：

- `make test lab=3_ri && make run` 在本地测试上述样例（调试模式）
- `MOS_PROFILE=release make test lab=3_ri && make run` 在本地测试上述样例（开启优化）

## 提交评测 & 评测标准


请在开发机中执行下列命令后，在课程网站上提交评测。






```
$ cd ~/学号
$ git add -A
$ git commit -m "message" # 请将 message 改为有意义的信息
$ git push
```

在线评测时，所有的 `.mk` 文件、所有的 `Makefile` 文件、`init/init.c` 以及 `tests/` 和 `tools/` 目录下的所有文件都可能被替换为标准版本，因此请同学们在本机开发时，**不要**在这些文件中编写实际功能所依赖的代码。

具体测试点内容和分数如下。

提交评测

2024-04-24 21:06:58 | 评测冷却: 137s 

    	2	仅包含任意未定义指令	20	提交
	3	仅包含 pmaxub 指令	20	
	4	仅包含 cas 指令	20	更新
	5	包含任意未定义指令以及两个自定义指令	30	