Lab3 Extra

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-adel_ades` 样例测试目录。

问题描述

课下实验中，我们主要介绍了异常的分发、异常向量组和 0 号异常（时钟中断）的处理过程。在本次 Extra 中，我们希望大家拓展 4 号异常 `AdEL` 和 5 号异常 `AdES` 的异常分发，并在此基础上实现自定义的异常处理。

在 `MIPS` 指令集中，对这两种异常的描述如下：

ExcCode(异常码)	助记符	描述
4	AdEL	在 load/fetch 指令执行过程中发生的地址错误。
5	AdES	在 store 指令执行过程中发生的地址错误。

以上错误可能是由于地址不对齐，也可能是由于违反了访问权限

在本次题目中，我们只考虑地址不对齐导致的上述错误，且只考虑 `lw` 和 `sw` 指令（即必须四字节对齐）的情况。

且我们要实现自定义的异常处理——通过**修改相应的 lw 或 sw 指令**，将未 4 字节对齐的地址的低 2 位（二进制格式下）置 0 ，得到符合对齐要求的地址。例如：对于地址 `0x7f000009` ，需要将其修改为 `0x7f000008` （即修改后地址 `new_addr = old_addr & ~0x3` ），方可得到符合要求的地址。**注意，这里我们要求仅修改指令中的立即数部分，不要修改任何寄存器中的值**

在异常处理完成，返回用户态之前，需要输出相应信息：

- lw处理完成：

```
printk("AdEL handled, new imm is : %04x\n", new_inst & 0xffff); // 这里的 new_ins
```

- sw处理完成：

```
printk("AdES handled, new imm is : %04x\n", new_inst & 0xffff); // 这里的 new_ins
```

注意上述中的异常处理函数需要完成的输出内容不包含双引号而且应该**单独成行**，并注意**避免其他非必要的输出**，否则将影响评测结果。

`lw` 和 `sw` 指令的机器码如下：

指令	31...26	25...21	20...16	15...0	行为
lw	100011	base	rt	imm	GPR[rt] ← memory[GPR[base] + (sign-extend)imm]
sw	101011	base	rt	imm	memory[GPR[base] + (sign-extend)imm] ← GPR[rt]

题目要求

1. 建立 AdEL 和 AdES 异常的处理函数。
2. 完成异常分发。
3. 在处理函数中，你需要通过修改相应指令以确保重新执行指令时的地址是四字节对齐的。

提示

1. 可以在 `kern/genex.S` 中，用 `BUILD_HANDLER` 宏来构建异常处理函数：

```
BUILD_HANDLER adel do_adel
BUILD_HANDLER ades do_ades
```

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

```
// 声明 handle 函数
extern void handle_adel(void);
extern void handle_ades(void);

// exception_handlers 请按以下代码实现
void (*exception_handlers[32])(void) = {
    [0 ... 31] = handle_reserved,
    [0] = handle_int,
    [2 ... 3] = handle_tlb,
    [4] = handle_adel,
    [5] = handle_ades,
#if !defined(LAB) || LAB >= 4
    [1] = handle_mod,
    [8] = handle_sys,
#endif
};

void do_adel(struct Trapframe *tf) {
    // 在此实现相应操作以使修改后指令符合要求
}

void do_ades(struct Trapframe *tf) {
    // 在此实现相应操作以使修改后指令符合要求
}
```

3. 本题涉及对寄存器值的读取。你需要访问保存现场的 `Trapframe` 结构体（定义在 `include/trap.h` 中）中对应通用寄存器。

4. 本题涉及对内存的访问，由于我们要修改的指令部分在 `kuseg` 区间，这一部分的虚拟地址需要通过 TLB 来获取物理地址。我们设置了程序中保存操作指令代码的 `.text` 节权限为**只读**，这部分空间在页表中仅被映射为 `PTE_V`，而不带有 `PTE_D` 权限，因此经由页表项无法对物理页进行写操作。可以考虑查询 `curenv` 的页表获取对应指令的物理地址，再转化为 `kseg0` 的虚拟地址，从而修改相应内容。

5. 本题要求自定义行为是要修改对应的指令的**立即数部分**，而不是通过在内核态用四字节对齐的地址直接获得数据，在这种做法下使用 `epc+4` 跳过异常将**无法通过测试**。

题目约束

1. 保证地址出错**当且仅当** lw 和 sw 访问的地址**未四字节对齐**！
2. 保证不会出现取指发生 AdEL 或 AdES 异常！
3. **不保证** lw 和 sw 指令的立即数为正数，但**保证**异常指令的初始立即数和按要求修改后的立即数都在 `signed short` (16位)的取值范围内！

4. 地址不对齐的指令中立即数对应的地址**可能是四字节对齐**的，即地址不对齐可能是寄存器的值未对齐，但本题要求仅修改指令中的立即数部分！
5. 保证可以**仅通过修改立即数部分的方式**得到正确结果！

样例输出 & 本地测试

对于样例：

```
int __attribute__((optimize("O0"))) main() {
    unsigned int src1[2]; // 数组 src1

    src1[0] = 1;
    src1[1] = 2;
    unsigned int dst = 3;
    unsigned int *ptr = src1;

    // lw test
    asm volatile("lw %0, 5(%1)\n\t"
             : "=r"(dst) /* 输出操作数，也是第0个操作数 %0 */
             : "r"(ptr)  /* 输入操作数，也是第1个操作数 %1 */
    );
    if (dst == 3) {
        return -1;
    } else if (dst != 2) {
        return -2;
    }

    // sw test
    dst = 4;
    ptr = (unsigned int*) ((unsigned int)ptr | 0x1);
    asm volatile("sw %0, 1(%1)\n\t"
             :
             : "r"(dst) , "r"(ptr)
    );
    dst = src1[0];
    if (dst == 1) {
        return -3;
    } else if (dst != 4) {
        return -4;
    }

    return dst;
}
```

运行正确的结果应当如下：

```
AdEL handled, new imm is : 0004
AdES handled, new imm is : ffff
env 00000800 reached end pc: 0x00400180, $v0=0x00000004
```

```
[00000800] free env 00000800
i am killed ...
```

对于错误处理的结果(函数返回值为-1到-4之间的某个整数)，将提示报错信息，仅供参考！

你可以使用：

- `make test lab=3_adel_ades && make run` 在本地测试上述样例（调试模式）
- `MOS_PROFILE=release make test lab=3_adel_ades && make run` 在本地测试上述样例（开启优化）
- **注意**：由于 lw 和 sw 指令在开启编译优化时可能被优化掉或不可达，因此在我们的测试代码中加入 `__attribute__((optimize("O0")))` 表示该函数不可被优化，若同学们开启优化进行测试，请**务必保留该特性**。

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，因此请同学们在本地开发时，**不要**在这些文件中编写实际功能所依赖的代码。

具体要求和分数分布如下：

测试点序号	测试点内容	测试点分值
1	样例	10分
2	弱测，循环执行同一条未四字对齐的 lw 指令	15分
3	弱测，循环执行同一条未四字对齐的 sw 指令	15分
4	弱测，立即数偏移为负数	30分