实验四 运行虚拟机

一、实验目的

完成虚拟机的全部程序，并运行：

1. 实现程序的加载；
2. 实现内存映射寄存器；
3. 运行虚拟机。

二、实验内容

1. 加载程序

之前提到了从内存加载和执行指令，但指令是如何进入内存的呢？将汇编程序转换为机器码时，得到的是一个文件，其中包含一个指令流和相应的数据。只需要将这个文件的内容复制到内存就算完成加载了。

程序的前16比特规定了这个程序在内存中的起始地址，这个地址称为`origin`。因此，加载时应该首先读取这16比特，确定起始地址，然后才能依次读取和放置后面的指令及数据。

下面是将LC-3程序读到内存的代码：

```
void read_image_file(FILE* file) {
    uint16_t origin; /* the origin tells us where in memory to place the image
*/
    fread(&origin, sizeof(origin), 1, file);
    origin = swap16(origin);

    /* we know the maximum file size so we only need one fread */
    uint16_t max_read = UINT16_MAX - origin;
    uint16_t* p = memory + origin;
    size_t read = fread(p, sizeof(uint16_t), max_read, file);

    /* swap to little endian */
    while (read-- > 0) {
        *p = swap16(*p);
        ++p;
    }
}
```

注意读取前16比特之后，对这个值执行了`swap16()`。这是因为LC-3程序是大端（big-endian），但现在大部分计算机都是小端的（little-endian），因此需要做大小端转换：

```
uint16_t swap16(uint16_t x) {
    return (x << 8) | (x >> 8);
}
```

> 注：大小端（Endianness）是指对于一个整型数据，它的每个字节应该如何解释。在小端中，第一个字节是最低位，而在大端中刚好相反，第一个字节是最高位。据我所知，这个顺序完全是人为规定的。不同的公司做出的抉择不同，因此我们这些后来人只能针对大小端做一些特殊处理。对于本实验中大小端相关的内容，知道这些就足够了。

再封装一下前面加载程序的函数，接受一个文件路径字符串作为参数，这样更加方便：

```
int read_image(const char* image_path) {
    FILE* file = fopen(image_path, "rb");
    if (!file) { return 0; };
    read_image_file(file);
    fclose(file);
    return 1;
}
```

2. 内存映射寄存器（Memory Mapped Registers）

某些特殊类型的寄存器是无法从常规寄存器表（register table）中访问的。因此，在内存中为这些寄存器预留了特殊的地址。要读写这些寄存器，只需要读写相应的内存地址。这些称为内存映射寄存器（MMR）。内存映射寄存器通常用于处理与特殊硬件的交互。

LC-3有两个内存映射寄存器需要实现，分别是：

- KBSR：键盘状态寄存器（keyboard status register），表示是否有键按下；
- KBDR：键盘数据寄存器（keyboard data register），表示哪个键按下了。虽然可以用`GETC`来请求键盘输入，但这个`trap routine`会阻塞执行，直到从键盘获得输入。`KBSR`和`KBDR`使得我们可以轮询设备的状态然后继续执行，因此程序不会阻塞。

```
enum {
    MR_KBSR = 0xFE00, /* keyboard status */
    MR_KBDR = 0xFE02  /* keyboard data */
};
```

内存映射寄存器使内存访问稍微复杂了一些。这种情况下不能直接读写内存位置，而要使用`setter`和`getter`辅助函数。当获取输入时，`getter`会检查键盘输入并更新两个寄存器（也就是相应的内存位置）。

```
void mem_write(uint16_t address, uint16_t val) {
    memory[address] = val;
}

uint16_t mem_read(uint16_t address)
{
    if (address == MR_KBSR) {
        if (check_key()) {
            memory[MR_KBSR] = (1 << 15);
            memory[MR_KBDR] = getchar();
        } else {
            memory[MR_KBSR] = 0;
        }
    }
    return memory[address];
}
```

至此实现了虚拟机的最后一部分。只要实现了前面提到的trap routine和指令，虚拟机就即将能够运行了。

3. 平台相关的细节

本节包含一些与键盘交互以及显示相关的代码。如果不感兴趣可以直接复制粘贴。

如果不是在Unix类系统上运行本程序，例如Windows，那本节内容需要替换为相应的平台实现。

```
uint16_t check_key() {
    fd_set readfds;
    FD_ZERO(&readfds);
    FD_SET(STDIN_FILENO, &readfds);

    struct timeval timeout;
    timeout.tv_sec = 0;
    timeout.tv_usec = 0;
    return select(1, &readfds, NULL, NULL, &timeout) != 0;
}
```

下面是特定于Unix的设置终端输入的代码：

```
struct termios original_tio;

void disable_input_buffering() {
    tcgetattr(STDIN_FILENO, &original_tio);
    struct termios new_tio = original_tio;
    new_tio.c_lflag &= ~ICANON & ~ECHO;
    tcsetattr(STDIN_FILENO, TCSANOW, &new_tio);
}

void restore_input_buffering() {
    tcsetattr(STDIN_FILENO, TCSANOW, &original_tio);
}
```

当程序被中断时，需要将终端的设置恢复到默认：

```
void handle_interrupt(int signal) {
    restore_input_buffering();
    printf("\n");
    exit(-2);
}
signal(SIGINT, handle_interrupt);
disable_input_buffering();
```

4. 运行虚拟机

现在可以编译和运行这个LC-3虚拟机了。

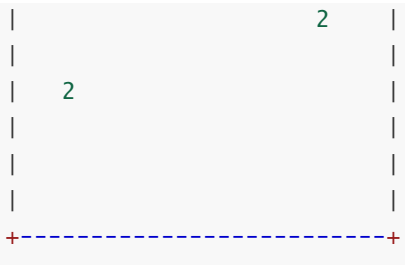
使用GCC等C语言编译器编译这个虚拟机，然后下载汇编后的两个小游戏：

- 2048.obj：https://arthurchiao.art/assets/img/write-your-own-virtual-machine-zh/2048.obj
- rogue.obj：https://justinmeiners.github.io/lc3-vm/supplies/rogue.obj

> 同时在学习通7.3节中提供。

用如下命令执行：`lc3-vm path/to/2048.obj`。

```
Play 2048!
{2048 Example 13}
Control the game using WASD keys.
Are you on an ANSI terminal (y/n)? y
+--------------------------+
|                          |
|                          |
|                          |
```


2
2