北京邮电大学
计算机组成原理课程设计

64 位虚拟机设计报告

学院：计算机学院

姓名：裴　家　亮

班级：2016211310

学号：2016211181

目录

一、虚拟机硬件结构.....2

（一） 数据通路.....2
（二） 中央处理器.....2
（三） 内存.....3
（四） 输入/输出设备.....4
（五） 中断处理器.....4

二、指令集架构.....5

（一）操作数与寻址模式.....5
（二） 指令集.....6
1. 一般传送指令.....6
2. 堆栈操作指令.....6
3. 算术运算指令.....7
4. 逻辑运算指令.....8
5. 移位指令.....9
6. 条件转移指令与无条件转移指令.....10
7. 特殊指令.....12

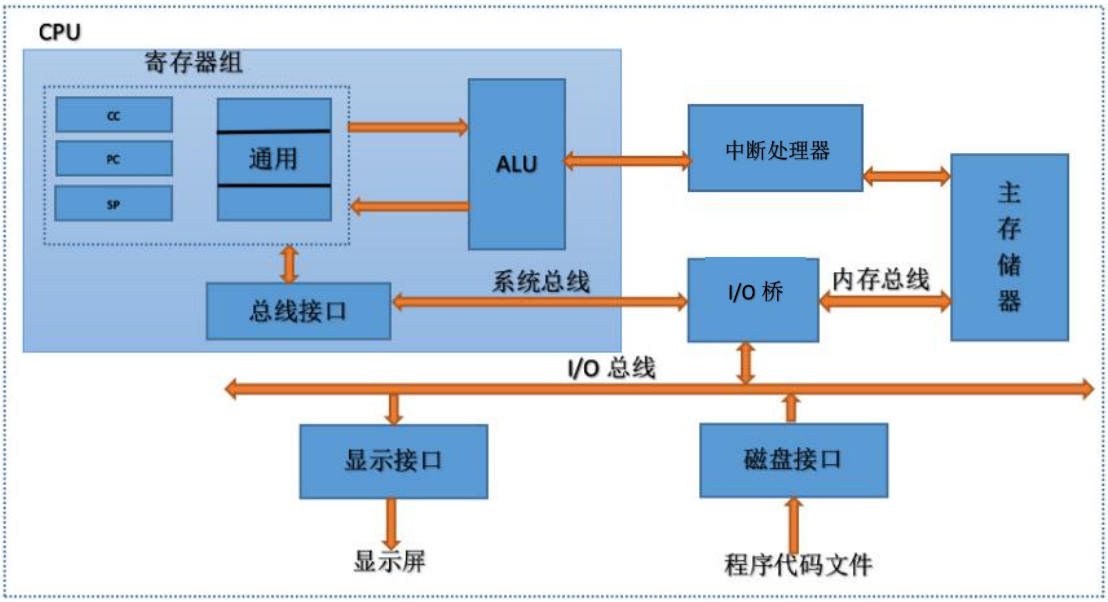
三、虚拟机软件实现框架.....13

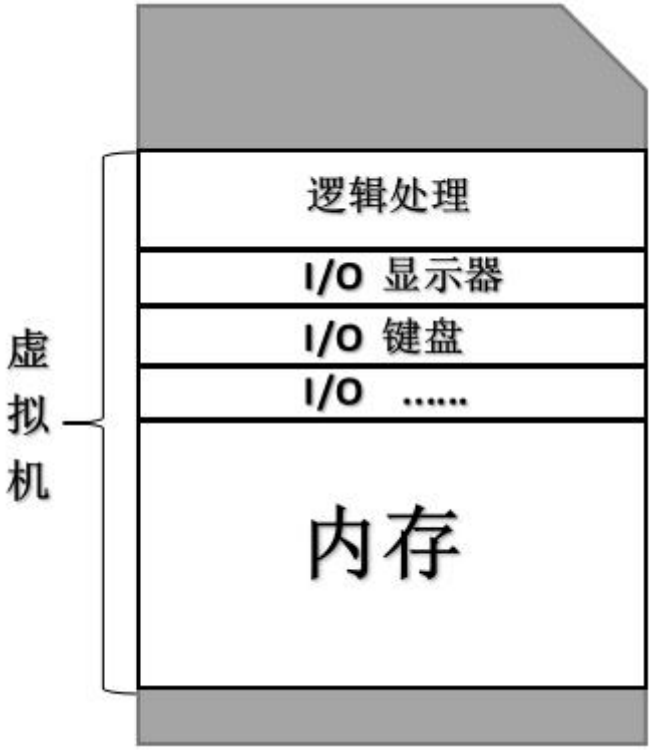
（一） 整体框架.....13
（二） 数据结构.....13
（三）部分子函数设想.....14
（四）环境及语言.....15
（五） 异常处理设计.....15
（六） 界面设计.....15
（七） 展望.....16

一、虚拟机硬件结构

（一） 数据通路

1. 整体框架





2. 简要概述

① 虚拟机整体硬件结构由 **CPU**、**内存**、**I/O**、**中断处理器**构成。
② 其中 CPU 由**寄存器组**、**运算器**组成，寄存器组又分为通用寄存器和特殊寄存器。
③ I/O 设备主要是磁盘和显示器，并由 I/O 桥负责与内存、CPU 统筹处理。
④ 各个部分通过总线连接。（在软件实现中通过函数接口体体现）

（二） 中央处理器

1. 虚拟机的 CPU 部分由寄存器组、运算器组成。

2. 寄存器组分为特殊寄存器组和通用寄存器组，共 16 个寄存器，编号从 **0x0** 到 **0xF**。

编号	名称	描述
0	R1	通用寄存器
1	R2	通用寄存器
2	R3	通用寄存器
3	R4	通用寄存器
4	R5	通用寄存器
5	R6	通用寄存器
6	R7	通用寄存器
7	R8	通用寄存器
8	R9	通用寄存器
9	R10	通用寄存器
A	R11	通用寄存器
B	R12	通用寄存器
C	R13	通用寄存器
D	CC	条件码寄存器
E	PC	程序计数器
F	SP	栈指针寄存器

3. 寄存器组说明：

① 特殊寄存器组
A.**CC**（Condition Code Register）：条件码寄存器，维护 CF（进位标志）、ZF（零标志）、SF（符号标志）、OF（溢出标志），供条件转移指令使用。（在高级语言实现中，如何更新这些条件码是一个难点。）
B.**PC**（Program Counter）：程序计数器，维护当前执行程序指令的地址。
C.**SP**（Stack Pointer）：栈指针，维护栈区的栈顶地址。

② 通用寄存器组
提供临时存储、保存返回值等功能。

4. ALU 是一个抽象的概念，可以认为是一个子程序，封装了处理各种指令的子函数，供虚拟机调用，实现指令系统的功能。

5. 虚拟机的 CPU 为 64 位，因此将寄存器设置为能够存储 64 位数据的类型，尽管 CC 只需要 4 位就可以满足功能，但是考虑到扩展性和方便统一处理，将 CC 也定义成 64 位，其存储结构如下:

Other	Other	Other	...	...	CF	ZF	SF	OF
63	62	61	...	...	3	2	1	0
0	0	0	...	...	0/1	0/1	0/1	0/1

CC 目前低四位分别维护 CF、ZF、SF、OF 四个条件码，用 0/1 表示状态，在具体实现中，这样的存储结构十分容易扩展到更多的条件码。

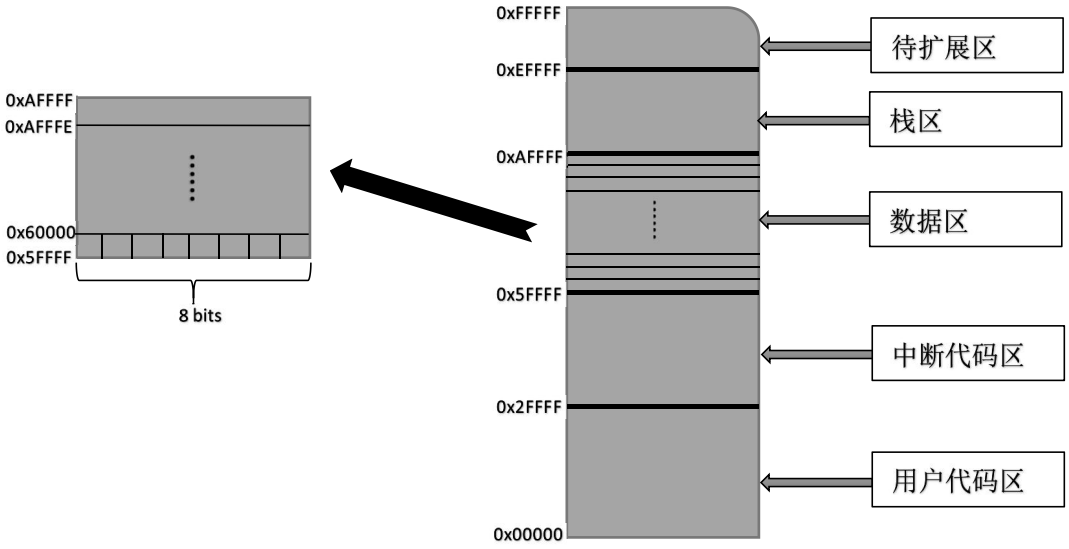
（三）内存

1. 虚拟机的内存大小为 1024*1024 字节，即 1MB。
2. 具体实现中，将每个存储单元大小设置为单个字节（8bits），因此地址范围为 0x00000~0xFFFFF。

3. 内存将分为用户代码区、中断代码区、数据区、栈区四大分区，为提供扩展性，在四大分区之外仍留有一定空间的待扩展区。
4. 内存分区说明：

分区名称	功能	地址范围	备注
用户代码区	存储用户的代码指令序列	0x00000~0x2FFFF	可读/可写
中断代码区	存储中断程序指令序列	0x30000~0x5FFFF	只读
数据区	提供程序需要的数据读写	0x60000~0xAFFFF	可读/可写
栈区	程序栈	0xB0000~0xEFFFF	可读/可写
待扩展区	为虚拟机提供扩展空间	0xF0000~0xFFFFF	由扩展功能决定

5. 整体结构图示：

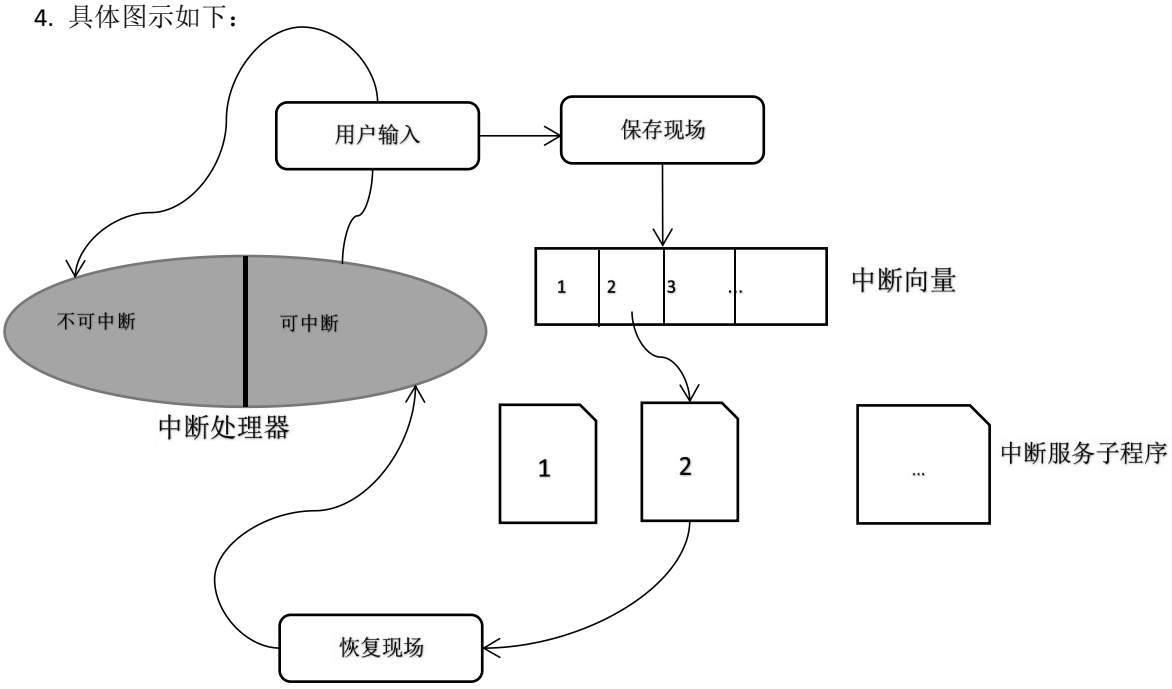


（四） I/O

1. 虚拟机的 I/O 设备主要为显示设备和磁盘。
2. 显示设备提供结果显示、设备监控两大功能，统一由显示控制器协调处理。
①显示控制器：在高级语言实现中体现为一子函数，能够打包程序运行结果与寄存器等设备的运行时数值并交由显示设备输出。
②设备监控：为用户提供监视寄存器、内存数据的功能。
③结果显示：前期虚拟机通过命令行显示，后期改进为图形化显示。
3. 磁盘设备主要负责存储用户源代码，磁盘控制器能够将用户源代码转换成虚拟机可识别的数据结构并由 I/O 桥加载到内存中，最终由 CPU 调取执行。

（五）中断处理器

1. 中断处理器可分为两种状态：可中断状态和不可中断状态。
2. 中断处理器在可中断状态下，等待用户输入，当响应成功后，将其转换成中断指令并送往 CPU，CPU 根据具体中断指令在中断向量中选择具体终端服务子程序，将当前操作暂停并保存现场，转而执行中断程序，同时设置中断处理器状态为不可中断状态，此时若用户再次输入，中断处理器将不应答。
3. 当中断程序执行完毕后，中断控制器将负责恢复现场。
4. 具体图示如下：



二、 指令集架构

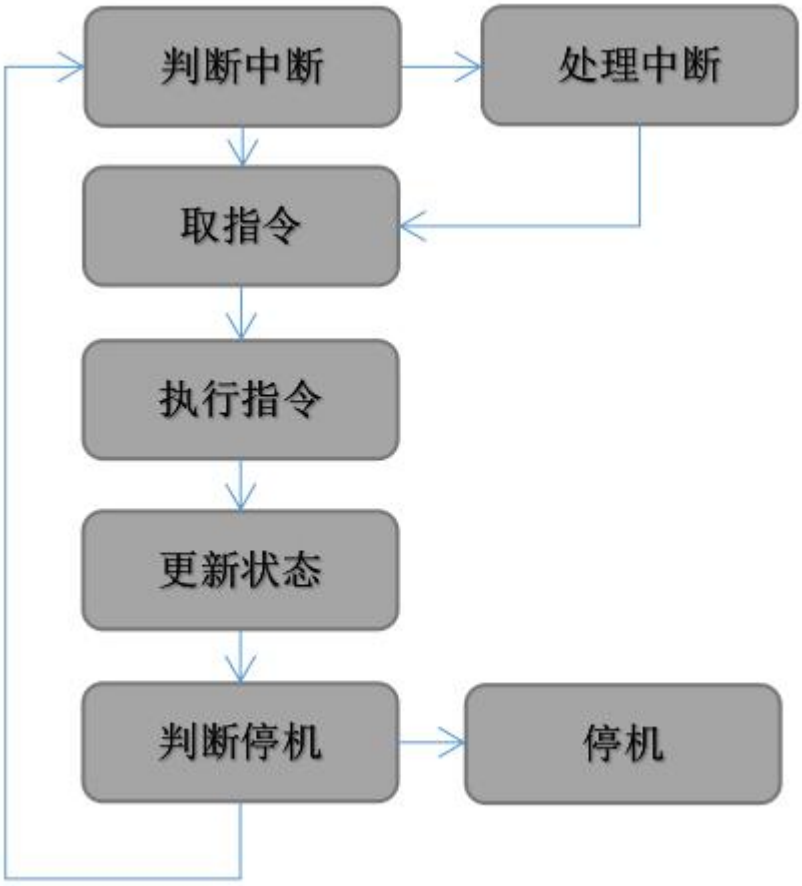
（一）操作数与寻址模式

1. 操作数：
①立即数：64 位数据，用 I 表示。
②寄存器，寄存器编号，用 R 表示。
③内存，内存地址，用 M 表示。
2. 操作数组合类型：
①零操作数，如 HLT。
②单操作数，如 INC、DEC。
③双操作数：（考虑到时序问题，禁止内存到内存直接操作）
A. RR：寄存器到寄存器
B. RM：寄存器到内存

C. MR：内存到寄存器
D. IR：立即数到寄存器
E. IM：立即数到内存
3. 寻址模式
虚拟机采用间接寻址，例如 RM 类型双操作数指令，M（内存地址）是通过索引寄存器数据来获得的。

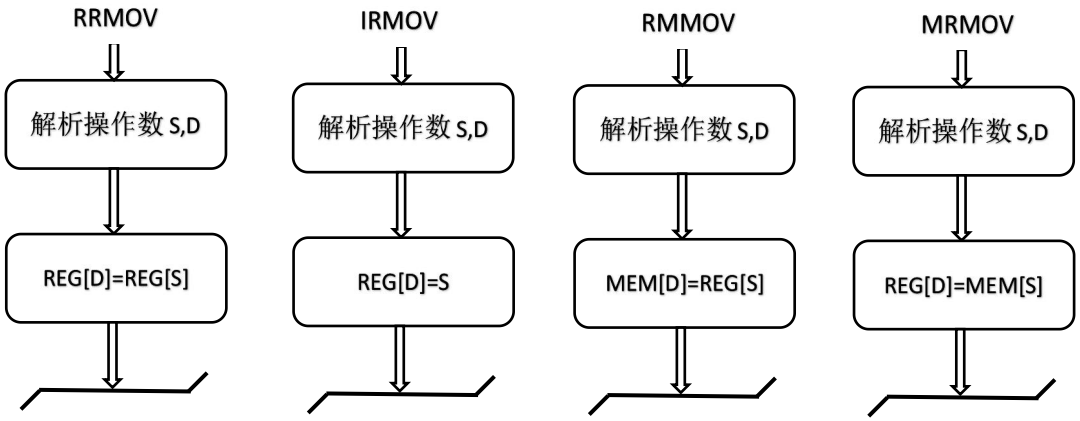
（二）指令集（S:源操作数　D:目的操作数）

1. 指令运行框架（具体执行指令流程在下方）



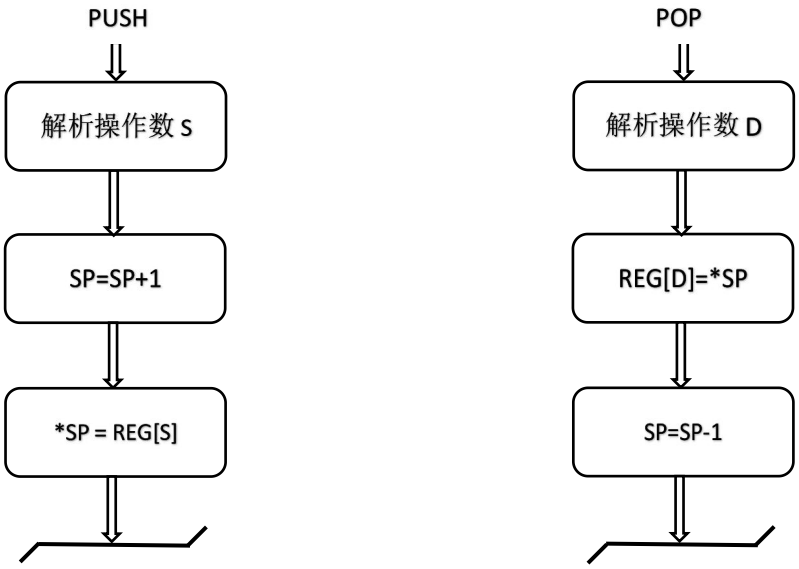
2. 一般传送指令

指令名称	操作数类别	功能说明
RRMOV S,D	RR	寄存器到寄存器一般传送，D<---S
IRMOV S,D	IR	立即数到寄存器一般传送，D<---S
RMMOV S,D	RM	寄存器到内存一般传送，D<---S
MRMOV S,D	MR	内存到寄存器一般传送，D<---S



3. 堆栈操作指令

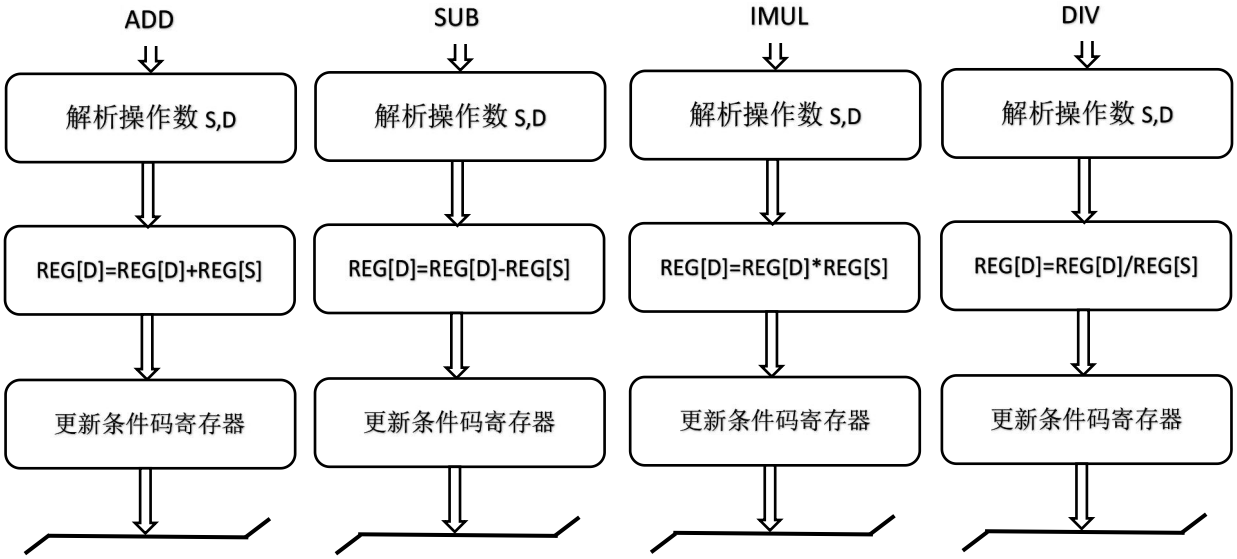
指令名称	操作数类别	功能说明
PUSH S	R	入栈，SP++,*SP<---S
POP D	R	出栈，D<---*SP,SP--

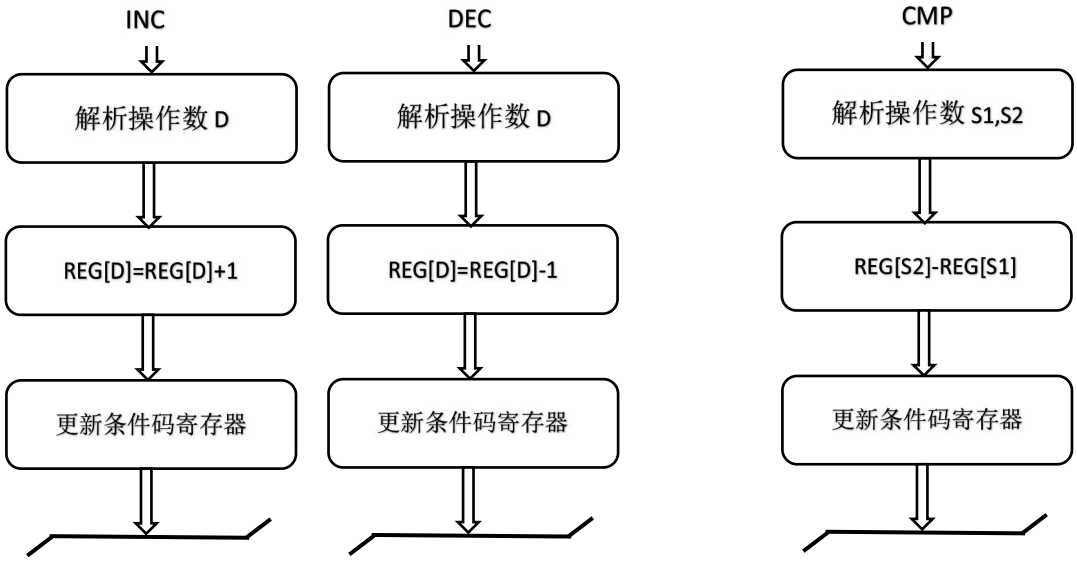


4. 算术运算指令

指令名称	操作数类别	功能说明
INC D	R	自增，D<---D+1
DEC D	R	自减，D<---D-1
ADD S,D	RR	加，D<---D+S
SUB S,D	RR	减，D<---D-S

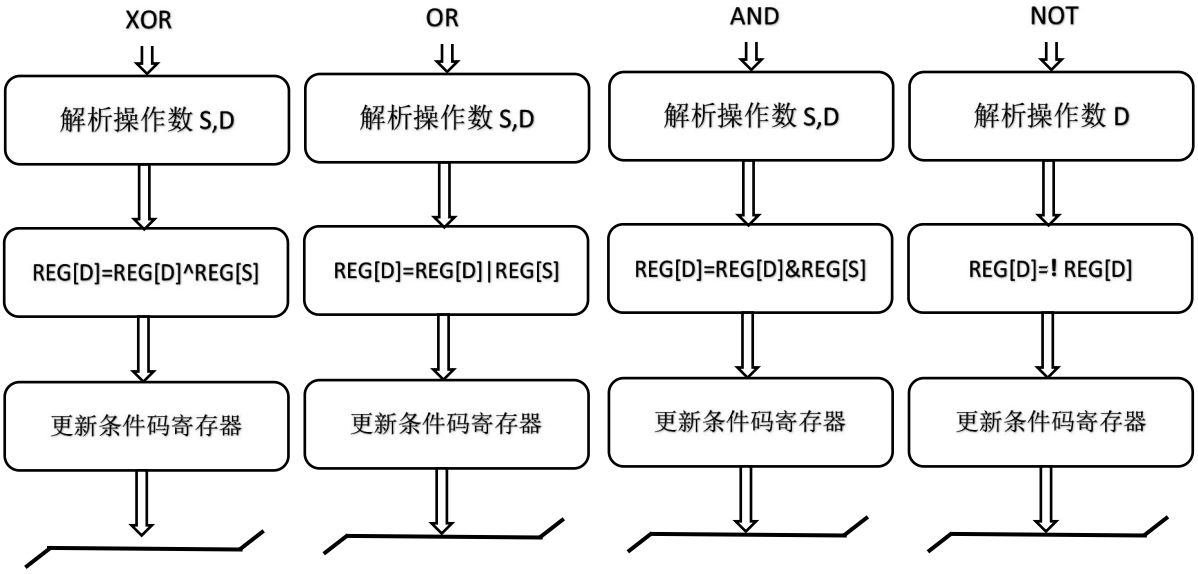
IMUL S,D	RR	乘，D<---D*S
DIV S,D	RR	除，D<---D/S
CMP S1,S2	RR	比较，S2-S1





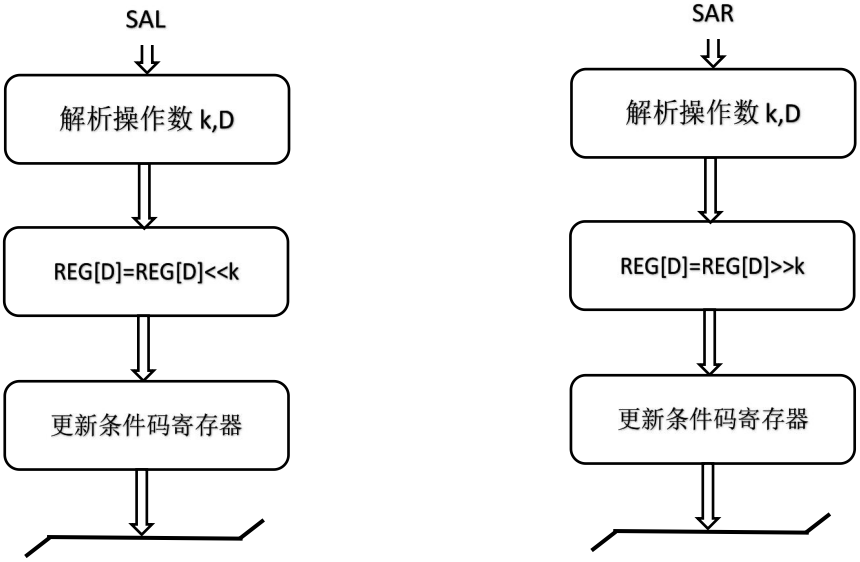
5. 逻辑运算指令

指令名称	操作数类型	功能说明
NOT D	R	逻辑非，D<---!D
XOR S,D	RR	逻辑异或，D<---D^S
OR S,D	RR	逻辑或，D<---D|S
AND S,D	RR	逻辑与，D<---D&S



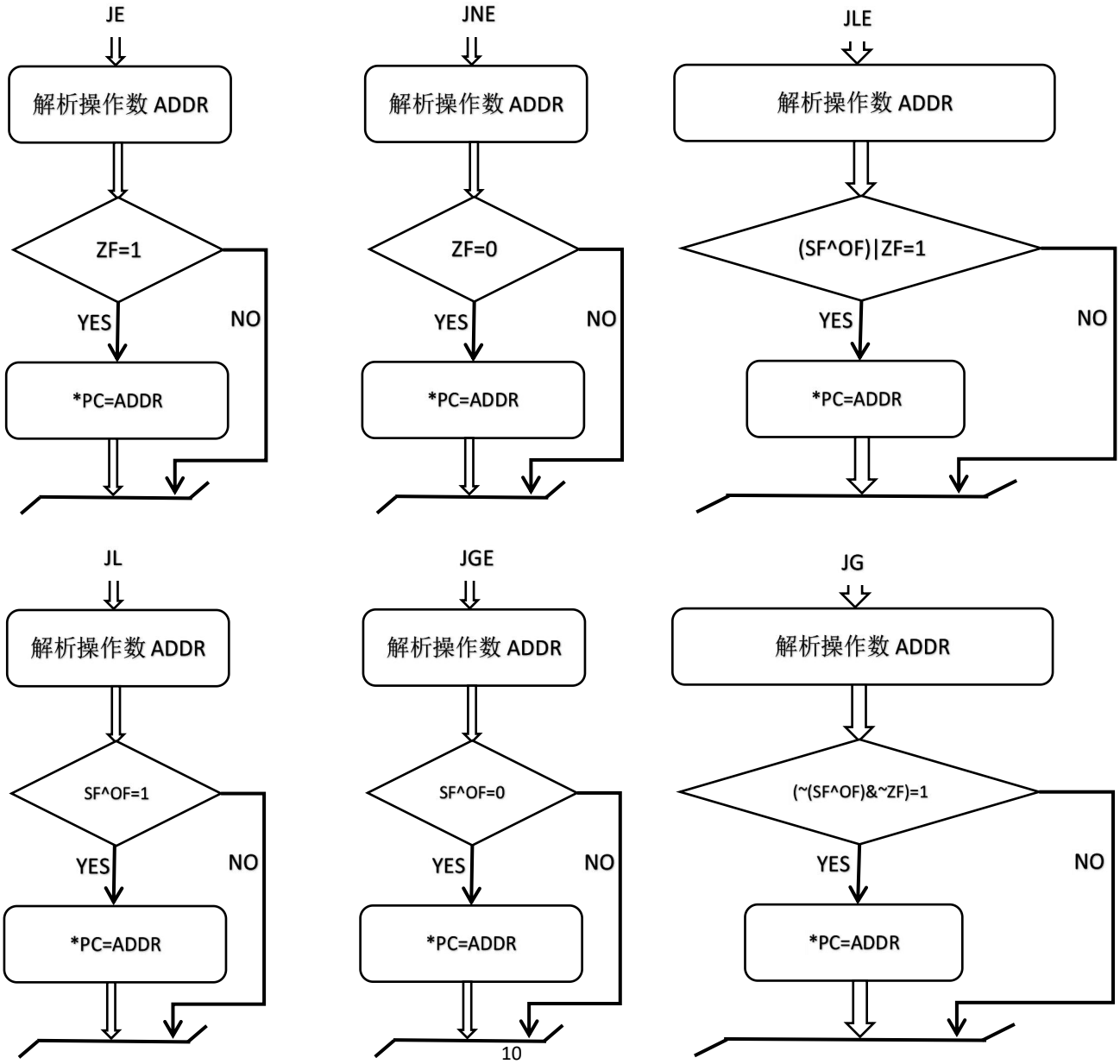
6. 移位指令

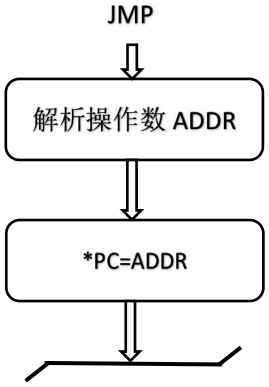
指令名称	操作数类型	功能说明
SAL k,R	IR	左移，D<---D << k
SAR k,R	IR	右移，D<---D >> k



7. 条件转移指令与无条件转移指令

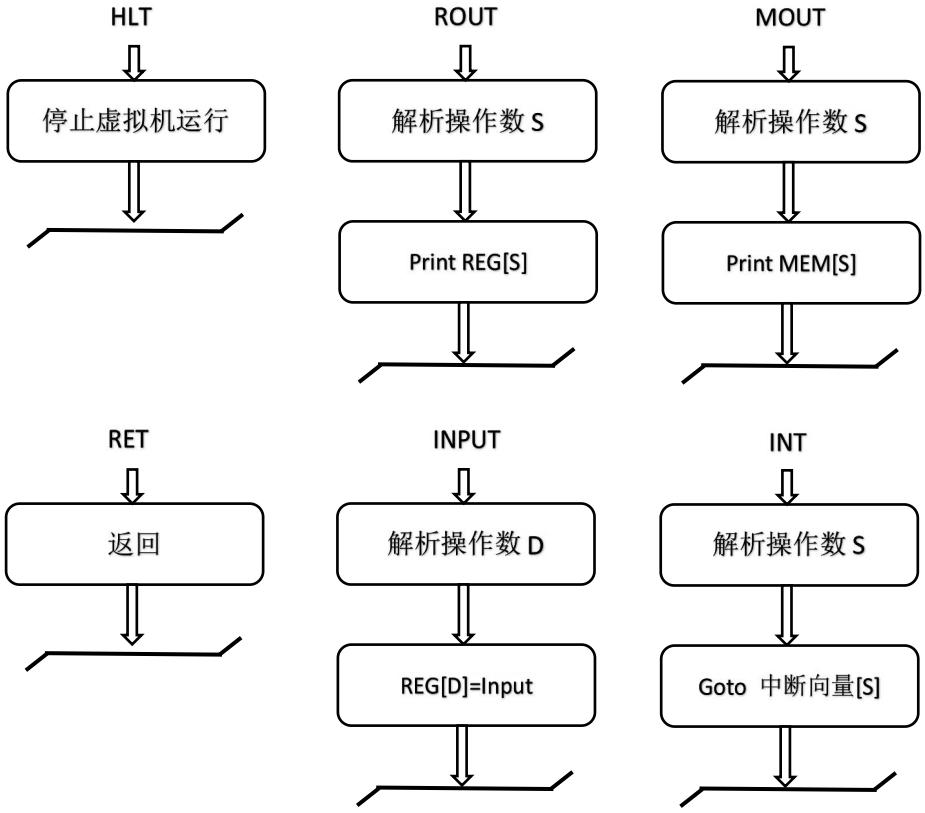
指令名称	操作数类型	功能说明
JMP ADDR	M	无条件转移，PC<---ADDR
JE ADDR	M	条件转移，当 ZF=1 时，PC<---ADDR
JNE ADDR	M	条件转移，当 ZF=0 时，PC<---ADDR
JLE ADDR	M	条件转移，当(SF^OF)|ZF=1 时，PC<---ADDR
JL ADDR	M	条件转移，当 SF^OF=1 时，PC<---ADDR
JGE ADDR	M	条件转移，当 SF^OF=0 时，PC<---ADDR
JG ADDR	M	条件转移，当(~(SF^OF)&~ZF)=1 时，PC<---ADDR





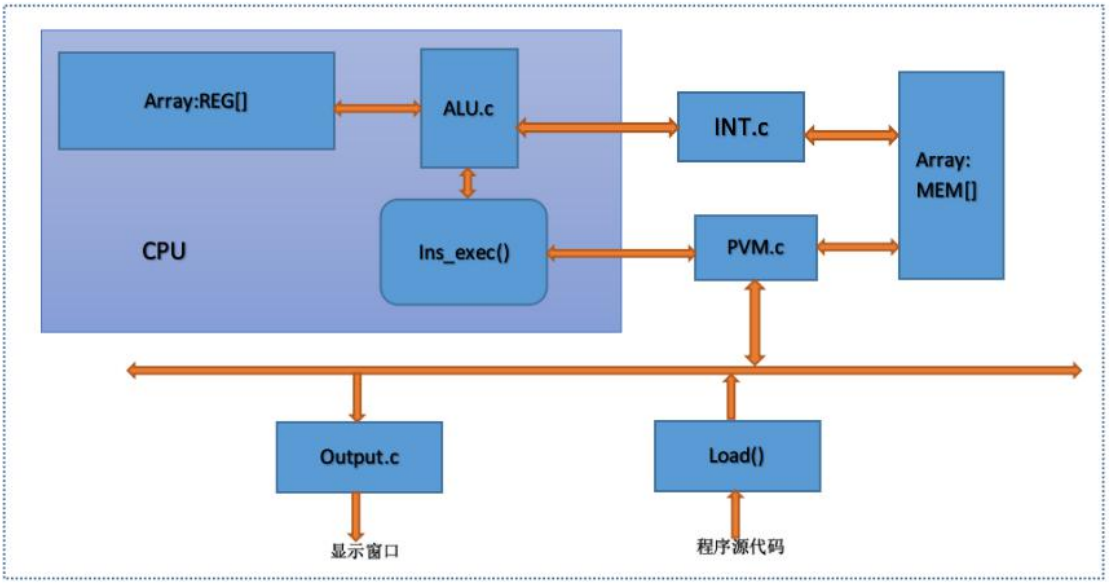
8. 特殊指令

指令名称	操作数类型	功能说明
HLT	NULL	停机指令
ROUT S	R	寄存器输出指令，显示设备<---S
MOUT S	M	内存输出指令，显示设备<---S
INPUT D	R	寄存器输入指令，D<---用户输入
RET	NULL	返回指令
INT S	I	调用中断服务子程序



三、虚拟机软件实现框架

（一）整体框架



（二）数据结构

1. 寄存器：由于每一个寄存器存储 64 位数据，虚拟机采用长度为 16 的 64 位整型数组来模拟寄存器组。其中 REG[0]~REG[12]是通用寄存器，而 REG[13]~REG[15]分别是栈指针、程序计数器、条件码急寄存器。

```
int REG_SP = 13, REG_PC = 14, REG_CC = 15;
int64_t REG[REG_SIZE];
```

2. 内存：虚拟机模拟能力有限，采用了两种不同的数据结构分别模拟指令代码和程序数据。
①代码区（包括用户代码区和中断代码区）：虚拟机定义了一种结构体来存储每一条指令，并且将所有指令存储于该类型的结构体数组中，结构体数组的下标就是代码在内存中对应的地址，PC 指向该结构体数组中的某一个元素。结构体定义如下：

```
typedef struct instruction {
        char insName[10];
        char S[10];
        char D[10];
}instruction;
```

其中 insName、S、D 分别表示指令名称、源操作数、目的操作数，而代码区则有该类型的结构体数组构成，如下:

```
        instruction *USER_CODE[USERCODE_SIZE];
        instruction *INT_CODE[INTCODE_SIZE];
```

② 数据区和栈区：数据区和栈区直接用字节数组模拟，具体定义如下（其中 DATA_SIZE 表示数据区大小，STACK_SIZE 表示栈区的大小）：

```
        uint32_t DATA_SIZE = 0x6FFFF, STACK_SIZE = 0x4FFFF;
        int8_t MEM[DATA_SIZE + STACK_SIZE];
```

需要注意的是，由于内存存储单元是 8 位数据，而 64 位虚拟机传送的数据通常需要 8 个这样的存储单元，针对这样的情况，本虚拟机采用**大端模式**解码方式，数据的高字节存放在内存的低地址，例如$(1111111111110000000011111010101001010100000000000000000000000)_2$在内存中表示为（从低地址到高地址读）

0x00000	0x00001	0x00002	0x00003	0x00004	0x00005	0x00006	0x00007
11111111	11110000	00001111	10101010	01010101	00000000	00000000	00000000

（三）部分子函数设想

1.VM_load(FILE *file): 加载用户程序，将汇编语言转换成虚拟机可执行的语言。

```
void VM_load(FILE *file) {
    int i = 0;
    char a[10],b[10],c[10];
    while (fscanf(file, "%s %s %s", &INS[i].opName, &INS[i].S, &INS[i].D) >= 1) {
        printf("%s %s %s\n",INS[i].opName, INS[i].S, INS[i].D);
        i++;
    }
    fclose(file);
}
```

2.VM_initialize(): 初始化虚拟机硬件设备（内存、寄存器）等

```
void VM_initialize() {
    MEM = (uint64_t *)malloc(sizeof(uint64_t) * MEM_SIZE);

    for (int i = 0; i < MEM_SIZE; i++) MEM[i] = 0;
    REG = (uint64_t *)malloc(sizeof(uint64_t) * REG_NUM);
    for (int i = 0; i < REG_NUM; i++) REG[i] = 0;

    REG[PC_REG]= 0;
    REG[SP_REG] = STACK_S - 1;
}
```

3.VM_run(): 虚拟机运行程序，简单设想，还有虚拟机状态需要处理。

```
void VM_run() {
    ins *currentIns = NULL;
    currentIns = &INS[REG[PC_REG]];
    while ( strcmp(currentIns->opName, "HLT") ) {
        op_exec(currentIns);
        if (!FLAG_JMP)
            REG[PC_REG]++;
        currentIns = &INS[REG[PC_REG]];
    }
}
```

4.**op_exec(ins *I):** 执行各种指令（亦可能会将指令名用枚举类型表示，从而能够用 **switch-case** 语句处理该函数）

```
void op_exec(ins *I) {
    if ( !strcmp(I->opName,"RRMOV")) {
        int s = toNum(I->S);
        int d = toNum(I->D);
        REG[d] = REG[s];
    }
    if ( !strcmp(I->opName,"IRMOV")) {
        int s = toNum(I->S);
        int d = toNum(I->D);
        REG[d] = s;
    }
    ......
}
```

5. Vm_preprocess(FILE * I):用户代码预处理，目前预处理的部分是将 0/单操作数指令扩展成双操作数指令，方便结构体存储，另外增加了注释功能，“#”后的语句在预处理程序中会去除。

```
void Process(FILE *file) {
    FILE *new_file = fopen("new_pro.txt","w");
    while(!feof(file)) {
        char line[1010];
        fgets(line,1000,file);
        if (line[strlen(line) - 1] == '\n')
            line[strlen(line) - 1] = '\0';
        int s = 0;
        for (int i = 0;i < strlen(line);i++) {
            if (line[i] == ' ') {
                s++;
            }
            if (s == 2) {
                if (line[i] == '#') {
                    line[i] = '\0';
```

```
                    break;
                }
            }
        }
        if (s == 0) {
            strcat(line, " NULL NULL");
        }
        else if (s == 1) {
            strcat(line, " NULL");
        }
        fprintf(new_file, "%s\n",line);
    }
    fclose(new_file);
    printf("Preprocess is finished.\n");
}
```

（四）语言及环境

虚拟机主体采用 C/C++语言编写，运行在 Windows10 64 位系统上，后期图形化设计采用 Qt5。

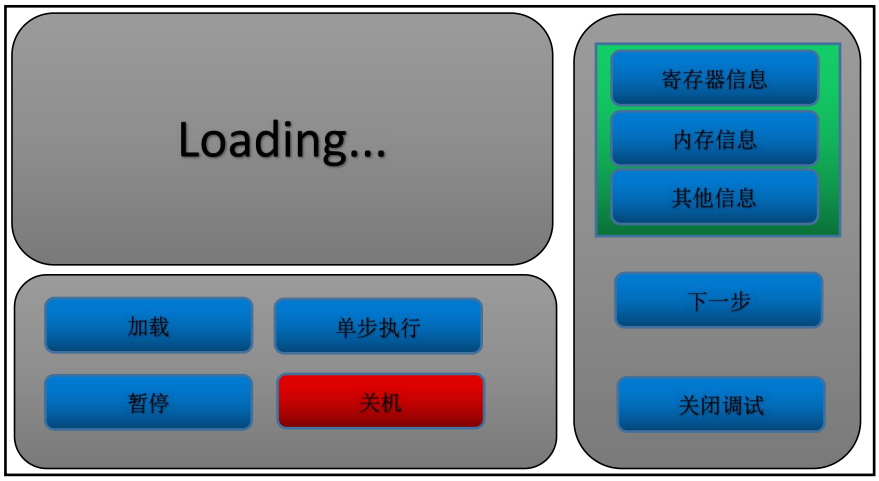
（五）异常处理设计

虚拟机增加了异常检测功能，在虚拟机 VM_run()例程中，每次执行完指令之后都会返回一个状态，表示是否出现异常，异常分成下列几类：

指令异常	指令名、操作数模式不匹配
运算异常	溢出、除 0 等
寻址异常	寄存器 id 超出限制、内存越界等

（六）图形界面设计

1. 虚拟机图形界面主体由三部分构成，分别是控制区、显示区以及调试区。
2. 控制区负责单步执行、暂停、重启等按钮，显示区显示程序运行结果，调试区显示程序运行时硬件状态。其图示如下（按钮仅作演示，只列出少部分）：



（七）展望

1. 实现图形界面，能够将程序运行状态、硬件状态实时显示。
2. 用自己定义的指令集实现一个图形计算器程序，能够在虚拟机上运行。
3. 在具体实现中不断修正指令集架构。