《程序设计课程设计》实验报告

实验名称　《冯诺依曼式计算机 CPU 模拟器程序设计》概要设计<003>

班　级　201**********班

组　号　无

姓　名　***

1. 输入、输出设计

1.1 输入

（1） 以文件的方式读取，从文件中读取指令序列，该文件以停机指令为结尾。

（2） 程序读取文件，执行相应指令，手动从键盘输入执行输入指令时的内容。

1.2 输出

（1） 程序每执行一条指令输出 CPU 寄存器状态

（2） 输出运行输出指令时要求输出的运行结果

（3） 停机指令时输出代码段内存

（4） 停机指令时输出数据段内存

2. 高层数据结构定义

2.1 全局常量定义

```
#define SIXTEEN 16
#define EIGHT 8
#define X1 129
#define X2 257
```

2.2 全局数据结构定义

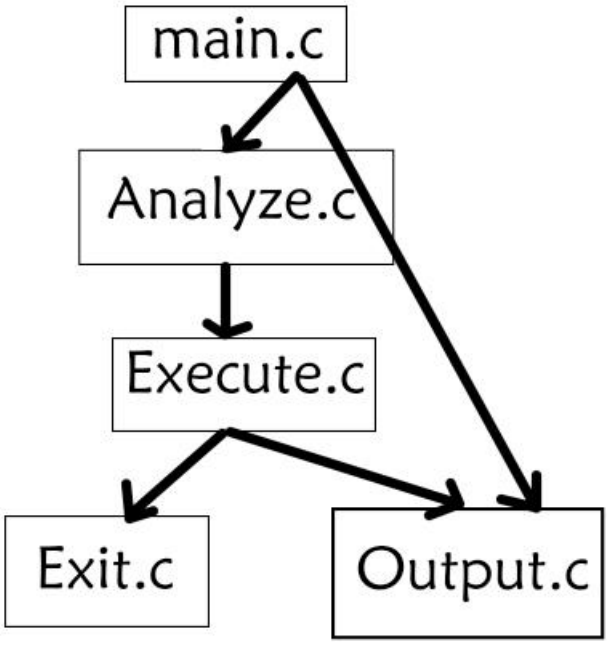
```
typedef struct systemRegister{
 int proCount;
 int instruRegister;
 int signRegister;
}     SystemReg;
typedef SystemReg * SystemReg_PTR;

typedef struct genrealRegister{
 int dataReg[4];
 int ptrReg[4];
}       GPR;
typedef GPR * GPR_PTR;
```

3. 系统模块划分（说明共分成哪些程序模块，各模块功能概述）

3.1 系统模块划分

[模块划分思路说明]

[模块关系图，独立线程的模块要用红色标记出来，如图所示]



1. 模块名称：main.c

模块功能简要描述：

（1）定义 11 个寄存器，数据段内存和代码段内存（一维数组），指令内存（二维数组），并赋予初值，打开文件，逐行读取文件内的指令，并将其存到定义好的指令内存当中，每读取一行代码，都要计算并存储代码段，读取完毕后，关闭文件；

（2）根据程序计数器的指令地址，读取其指向的内存中的指令；

（3）调用相应函数获取操作码、指令寄存器内存、立即数，程序计数器指向下一条指令；

（4）调用 analyzeInstruction()函数，即进入 Analyze.c 模块

（5）调用 print()函数，打印寄存器状态，即进入 Output.c；

2. 模块名称：Analyze.c

模块功能简要描述：

（1）根据操作对象段二进制码得到具体操作的对象；

（2）根据操作码，分析执行哪一种操作；

（3）调用函数执行该条指令，即进入 Execute.c 模块

3. 模块名称：Execute.c

模块功能简要描述：

（1） 根据 Analyze.c 模块的分析，执行相应的操作（包括指令需要的输入、输出操作）

（2） 如果为停机指令，则结束程序，同时打印寄存器状态，即进入 Output.c 模块和 Exit.c 模块；

（3） 若未结束程序，读取下一条指令，即回到 main.c 模块的（2）；

4. 模块名称：Output.c

模块功能简要描述：

（1） 输出系统寄存器状态；

（2） 输出通用寄存器状态；

5. 模块名称：Exit.c

模块功能简要描述：

（1） 输出代码段和数据段；

（2） 退出程序;

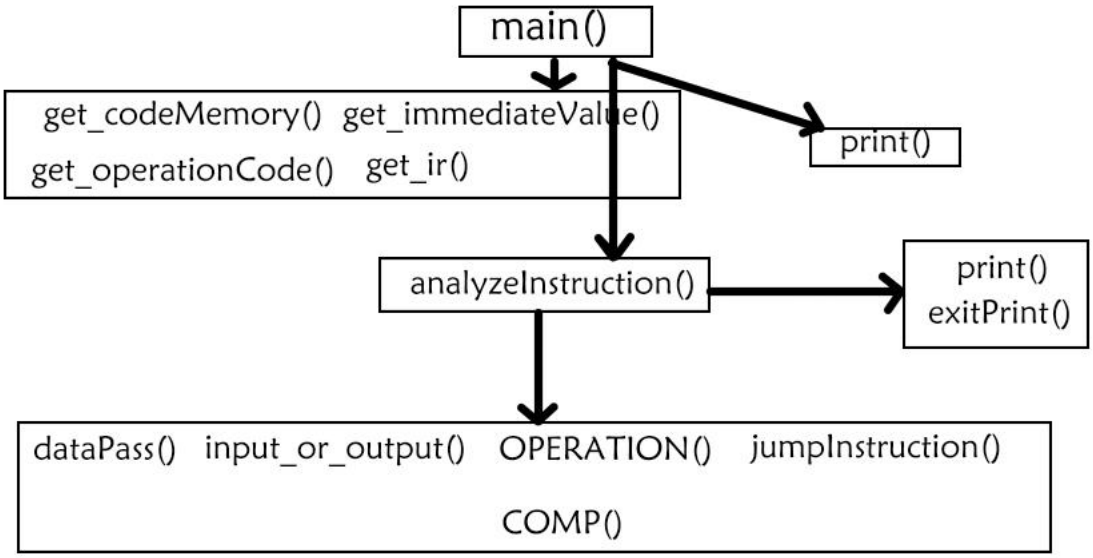
3.2 各模块函数说明

序号	函数原型	功能	参数	返回值
0	main(void)	定义寄存器、内存，打开文件，读取内存，储存代码段，并调动函数完成 CPU 的工作	void	void
1	int get_operationCode(char *ch,int i,int operCode)	获取操作码，并转换成十进制	ch, i， operCode;（ch 是一条指令的字符串，i 是操作码的第一位的数组下标,operCode 初值为 0，用来储存计算的操作码）	operCode（返回操作码）
2	int get_immediateValue(char *ch,int i,int immeValue)	获取立即数，并转换成成十进制	ch, i,immeValue;（ch 是一条指令的字符串，i 是立即数的第一位的数组下标,immeValue 初值为 0，用来储存计算的立即数）	immeValue（返回立即数）
3	int get_ir(char *ch,int i,int ir)	获取 ir 的值（也就是指令寄存器的值）	ch,i,ir（ch 是一条指令的字符串，i 是获取 ir 的第一位的数组下标,ir 初值为 0，用来储存计算的 ir 值）	Ir（返回 ir 的值）
4	int get_codeMemory(char *ch,int i,int codeMemory)	获取每一条指令对应的代码段	ch, i,codeMemory（ch 是一条指令的字符串，i 是获取代码段的第一位的数组下标,codeMemory 初值为 0, 用来储存代码段）	codeMemory（返回代码段）
5	void analyzeInstruction(int operCode,int immeValue,GPR_PTR gprPtr, SystemReg_PTR sysReg_ptr,int codeMemory[X1],int dataMemory[X2],char *ch)	分析指令，并调用相应函数完成后续操作	operCode, immeValue, gprPtr, sysReg_ptr, codeMemory[X1], dataMemory[X2], *ch（operCode 是操作码，immeValue 是立即数，gprPtr 是通用寄存器结构体地址，sysReg_ptr 是系统寄存器结构体地址，	void

			codeMemory 和 dataMemory 分别是代码段和数据段内存）	
6	void exitPrint(int codeMemory[X1],int dataMemory[X2])	结束程序并打印代码段和数据段；	codeMemory[X1], dataMemory[X2], （ codeMemory 和 dataMemory 分别是代码段和数据段内存）	void
7	void print(SystemReg_PTR sysReg_ptr,GPR_PTR gprPtr)	打印各个变量状态（即每读取执行一行指令打印寄存器状态）	sysReg_ptr,gprPtr （sysReg_ptr 是系统寄存器的结构体的地址，gprPtr 是通用寄存器结构体的地址）	void
8	void dataPass(GPR_PTR gprPtr,int immeValue,int dataMemory[X2],int data,int ptr)	数据传送（即：立即数传递或者数据寄存器和地址寄存器相互传递）	gprPtr,immeValue, dataMemory[X2] ， data,ptr （gprPtr 是通用寄存器结构体的地址，immeValue 是立即数，dataMemory 是数据段内存，data 和 ptr 是两个操作对象）	void
9	void OPERATION(int operCode,GPR_PTR gprPtr,int immeValue,int dataMemory[X2],int data,int ptr)	实现加减乘除和逻辑运算	operCode,gprPtr,immeValue, dataMemory[X2],data, ptr（operCode 是操作码，gprPtr 是通用寄存器结构体的地址，immeValue 是立即数，dataMemory 是数据段内存，data 和 ptr 是两个操作对象）	void
10	void COMP(GPR_PTR gprPtr,int immeValue, int dataMemory[X2], SystemReg_PTR sysReg_ptr, int data,int ptr)	比较指令：（1）数据寄存器与立即数比较（2）数据寄存器与地址指向的数据比较；进而修改标志寄存器的值	gprPtr,immeValue, dataMemory[X2], sysReg_ptr,data,ptr （gprPtr 是通用寄存器结构体的地址，immeValue 是立即数，dataMemory 是数据段内存，sysReg_ptr 是系统寄存器结构体地址，data 和 ptr 是两个操作对象）	void
11	void input_or_output (int operCode,GPR_PTR	实现输入输出指令：从输入端口	operCode,gprPtr,data （operCode 是操作	void

	gprPtr,int data)	输入一个整数存到数据寄存器，或输出数据寄存器的值到输出端口	码，gprPtr 是通用寄存器结构体的地址，data 为操作对象）	
12	void jumpInstruction(int immeValue,char *ch,SystemReg_PTR sysReg_ptr)	实现跳转指令：（1）无条件跳转指令（2）根据标志寄存器内容，判断是否跳转，并执行	immeValue,ch, sysReg_ptr;（immeValue 为立即数，ch 为一条指令的字符串，sysReg_ptr 为系统寄存器结构体地址）	void

3.3 函数调用图示及说明



解释说明：

1. main() 函数打开文件，首先逐行读取二进制字符串，并存入定义好的指令内存中。同时调用 get_codeMemory(),计算并储存代码段，；
2. 根据程序计数器内存地址，取出相应的指令，并让程序计数器指向下一条指令，调用 get_operationCode(),get_ir(),get_immediateValue(),分别获取操作码，指令寄存器内存和立即数；
3. 然后调用 analyzeInstruction()函数，在这个函数中，根据操作码，调用 dataPass()（数据传送函数),jumpInstruction()(跳转指令函数),COMP()(比较指令函数),OPERATION()（加减乘除和逻辑运算函数）以及 input_or_output（输入输出函数）；
4. 调用 print(),输出寄存器内存,如果为停机指令，则先调用 print(),再调用 exitPrint（）输出数据段代码段，并结束程序，否则读取程序计数器指向的下一条指令，重复操作；

4. 高层算法设计

核心算法：

a. 定义 11 个寄存器并初始化，打开文件 dict.dic;
b. 按行读取指令文件的内容，并存入内存当中,获取代码内存数据 codeMemory[]，存入代码段 codeMemory[]；关闭文件；

c. 根据程序计数器（指令地址），读取程序计数器指向的指令；
d. 让程序计数器+4，即对应下一条指令的地址；
e. 调用函数，计算出该指令代表的一个操作码 operCode（8 位字符），指令寄存器内存 ir 和一个立即数 immeValue（16 位字符）；
f. 分析操作码，获取操作对象 data 和 ptr，根据不同的指令操作，传入相应的操作对象到不同的函数实现指令功能；
g. 调用 print()函数，输出系统寄存器和通用寄存器的状态；
h. 完成操作后，根据程序计数器指向的内存地址，读取下一条指令并重复 d、e、f、g、h 操作;
i. 直到读取到结束指令时，输出代码段内存、数据段内存，结束程序;