ICS扩展实验-LC-3模拟器

功能目标

- 设计LC-3模拟器，对输入的文本进行分析，并在x86的机器架构上使用软件模拟执行，最终输出执行的结果
- 实现了以下指令
    - 所有访存指令
    - 所有算数运算指令
    - 控制流转移指令，包括
        - 条件跳转
        - 无条件跳转
        - 过程返回以及中断返回
- 支持的访存模式
    - 立即数访存
    - PC相对寻址
    - 寄存器相对寻址
    - 间接寻址
- 支持的处理器级别
    - 由于不能模拟中断信号，所以处理器永远出于用户级，这就意味着
        - 内存空间的全体对于程序都是可见的
        - RTI指令和RET指令相同，不会引起异常
        - 程序过程中的异常和中断不会加以处理
- 程序在裸机上运行，不支持操作系统，所以我们的模拟器并不支持陷入Trap
- 不支持伪指令
- 汇编语言出现语法错误的情况，比如用到的常数越界，这种情况我们不负责检查，应当是汇编器的任务

源代码

包含有两个文件，一个用于匹配和识别指令的头文件，另外一个主控函数

头文件

定义了各个宏，用于字符串匹配

```
#define ADD "ADD"
#define AND "AND"
#define BR "BR"
#define JMP "JMP"
#define JSR "JSR"
#define JSRR "JSRR"
#define LD "LD"
#define LDI "LDI"
#define LDR "LDR"
#define LEA "LEA"
#define NOT "NOT"
```

```
#define RET "RET"
#define RTI "RTI"
#define ST "ST"
#define STR "STR"
#define TRAP "TRAP"
#define SYMBOL 1
#define COMMAN 0
#include<string.h>
#include<string>
using namespace std;
char* compareopcode(char *opcode){
    if(strcmp(opcode,ADD)==0){
        return ADD;
    }
    else if(strcmp(opcode,AND)==0){
        return AND;
    }
    else if(strcmp(opcode,BR)==0){
        return BR;
    }
    else if(strcmp(opcode,JMP)==0){
        return JMP;
    }
    else if(strcmp(opcode,JSR)==0){
        return JSR;
    }
    else if(strcmp(opcode,JSRR)==0){
        return JSRR;
    }
    else if(strcmp(opcode,LD)==0){
        return LD;
    }
    else if(strcmp(opcode,LDI)==0){
        return LDI;
    }
    else if(strcmp(opcode,LDR)==0){
        return LDR;
    }
    else if(strcmp(opcode,LEA)==0){
        return LEA;
    }
    else if(strcmp(opcode,NOT)==0){
        return NOT;
    }
    else if(strcmp(opcode,RET)==0){
        return RET;
    }
    else if(strcmp(opcode,RTI)==0){
        return RTI;
    }
    else if(strcmp(opcode,ST)==0){
        return ST;
    }
    else if(strcmp(opcode,STR)==0){
        return STR;
    }
    else if(strcmp(opcode,TRAP)==0){
        return TRAP;
```

```
    }
    else{
        return "\0";
    }
}
int returninstrtype(char *str){
    for(int i=0;str[i]!='\0';i++){
        if(str[i]==':')
            return i;
    }
    return COMMAN;
}
char* returnopcode(char *str){
    char tempstr[10]={'\0'};
    for(int i=0;str[i]!=' ';i++){
        tempstr[i]=str[i];
    }
    return compareopcode(tempstr);
}
typedef struct symbol{
    int PC;
    string name;
    symbol *next;
}symbol;
int find(symbol *list,string target){
    symbol* pointer=list;
    while(pointer){
        if(pointer->name==target){
            return pointer->PC;
        }
        else{
            pointer=pointer->next;
        }
    }
    return -1;
}
char jump_target[10];
void returnjumptarget(char *str){
    int i;
    for(i=0;str[i]!='\0';i++){
        jump_target[i]=str[i];
    }
    jump_target[i]='\0';
}
```

- 前面定义的宏代表指令特定的操作码
- `returnInstructionType`用于返回指令的类型，将指令分成两个类型，一种是前面带标号的，一种是无标号的裸语句，我们最终要将一段代码，无标号的语句在宏中定义为common
- `returnopcode`的作用是返回一段以文本形式存在的操作码，相当于一个字符串处理函数
- `find`作用是，假设我们遇到了一个标号，现在我们要将这个标号转化为一个有符号数，此时就需要查询符号表找到对应符号代表的PC或者地址
- `returnjumptarget`的作用是返回一段数字文本形式代表的数字，其实可以直接调库实现
- `compareopcode`当我们已经获得一条指令的操作码，我们需要通过字符串比较的方式判断这是什么类型的指令

主控程序

```
//#include<stdio.h>
#include<string>
#include<iostream>
#include<string.h>
#include "instruction.h"
using namespace std;
int main(){
    short memory[65536]={0};
    short Register[8]={0};
    char Instruction[65536][20]={'\0'};//Max instruction number is 1000 and each
instruction's length can be 20
    int PC=0;//program counter
    bool ZERO=false,POSITIVE=false,NAGATIVE=false;
    symbol *list_symbol=NULL;
    string inst;
    char *opcode;
    int start_instr;
    char tag_symbol[10];
    bool Imminstr=false;
    int DP,SR1,SR2;
    int Immnumber;
    int i=0;
    int ORIPC=0;
    while(1){
        gets((char*)(Instruction+i));
        if(strcmp((char*)(Instruction+i),"HALT")==0){
            break;
        }
        if(Instruction[i][0]=='.'&&Instruction[i][1]=='O'&&Instruction[i]
[2]=='R'){
            for(int i=7;Instruction[PC][i];i++){
                ORIPC=ORIPC*10;
                ORIPC+=Instruction[PC][i]-'0';
            }
            i=ORIPC;
            i++;
            continue;
        }
        cout<<"The Instruction is "<<(char *)(Instruction+i)<<endl;
        i++;
        if(returninstrtype((char*)(Instruction+i-1))!=0){
            int i=0;
            for(i=0;Instruction[PC][i]!=':';i++){
                tag_symbol[i]=Instruction[PC][i];
            }
            tag_symbol[i]='\0';

            for(i=i++;Instruction[PC][i]!=' ';i++){

            }
            start_instr=i;
            opcode=returnopcode(&Instruction[PC][i]);
            symbol *temp;
            for(temp=list_symbol;temp!=NULL;temp=temp->next){
```

```
            }
            temp=(symbol*)malloc(sizeof(symbol));
            temp->name=tag_symbol;
            temp->next=NULL;
            temp->PC=PC;
        }
    }
    int MAXinstr=i;
    PC=ORIPC+1;
    int breakpoint;
    printf("Input the breakpoint");
    scanf("%d",&breakpoint);
    while(1){
        if(strcmp((char*)(Instruction+PC),"HALT")==0){
            printf("This is the end of Analyze and We can answer your
request\n");
            break;
        }
        else if(PC==breakpoint)
            break;
        else{
            if(returninstrtype((char*)(Instruction+PC))==0){
                opcode=returnopcode((char*)(Instruction+PC));
                start_instr=0;
            }//if not a command instruction with a symbol
            else{
                int i=0;
                for(i=0;Instruction[PC][i]!=':';i++){
                    tag_symbol[i]=Instruction[PC][i];
                }
                tag_symbol[i]='\0';
                for(i=i++;Instruction[PC][i]!=' ';i++){

                }
                start_instr=i;
                opcode=returnopcode(&Instruction[PC][i]);
            }
        }
        if(strcmp(opcode,ADD)==0){
            Imminstr=false;
            if(Instruction[PC][start_instr+10]=='#'){
                Imminstr=true;
            }
            else{
                Imminstr=false;
            }
            DP=Instruction[PC][start_instr+5]-'0';
            SR1=Instruction[PC][start_instr+8]-'0';
            SR2=Instruction[PC][start_instr+11]-'0';
            if(Imminstr){
                Immnumber=0;
                for(int i=11;Instruction[PC][i]!='\0';i++){
                    Immnumber=Immnumber*10;
                    Immnumber=Immnumber+Instruction[PC][i]-'0';
                }
                Register[DP]=Register[SR1]+Immnumber;

            }
```

```
            else{
                Register[DP]=Register[SR1]+Register[SR2];
            }
            ZERO=false;
            NAGATIVE=false;
            POSITIVE=false;
            if(Register[DP]>0){
                POSITIVE=true;
            }
            else if(Register[DP]==0){
                ZERO=true;
            }
            else{
                NAGATIVE=true;
            }
            PC++;
        }
        else if(strcmp(opcode,AND)==0){
            Imminstr=false;
            if(Instruction[PC][start_instr+10]=='#'){
                Imminstr=true;
            }
            else{
                Imminstr=false;
            }
            DP=Instruction[PC][start_instr+5]-'0';
            SR1=Instruction[PC][start_instr+8]-'0';
            SR2=Instruction[PC][start_instr+11]-'0';
            if(Imminstr){
                Immnumber=0;
                for(int i=11;Instruction[PC][i]!='\0';i++){
                    Immnumber=Immnumber*10;
                    Immnumber=Immnumber+Instruction[PC][i]-'0';
                }
                Register[DP]=Register[SR1]&Immnumber;

            }
            else{
                Register[DP]=Register[SR1]&Register[SR2];
            }
            ZERO=false;
            NAGATIVE=false;
            POSITIVE=false;
            if(Register[DP]>0){
                POSITIVE=true;
            }
            else if(Register[DP]==0){
                ZERO=true;
            }
            else{
                NAGATIVE=true;
            }
            PC++;
        }
        else if(opcode[0]=='B'){
            if(strcmp(opcode,"BRN")==0){
                returnjumptarget(&Instruction[PC][start_instr+4]);
                string target(jump_target);
```

```
                if(find(list_symbol,target)==0){
                    printf("Error!,No such symbol\n");
                    return 0;
                }
                else if(NAGATIVE){
                    PC=find(list_symbol,target);
                }
                else{
                    PC=PC+1;
                }
            }
            else if(strcmp(opcode,"BRZ")==0){
                returnjumptarget(&Instruction[PC][start_instr+4]);
                string target(jump_target);
                if(find(list_symbol,target)==0){
                    printf("Error!,No such symbol\n");
                    return 0;
                }
                else if(ZERO){
                    PC=find(list_symbol,target);
                }
                else{
                    PC++;
                }
            }
            else if(strcmp(opcode,"BRP")==0){
                returnjumptarget(&Instruction[PC][start_instr+4]);
                string target(jump_target);
                if(find(list_symbol,target)==0){
                    printf("Error!,No such symbol\n");
                    return 0;
                }
                else if(POSITIVE){
                    PC=find(list_symbol,target);
                }
                else{
                    PC++;
                }
            }
            else if(strcmp(opcode,"BR")==0){
                returnjumptarget(&Instruction[PC][start_instr+3]);
                string target(jump_target);
                if(find(list_symbol,target)==0){
                    printf("Error!,No such symbol\n");
                    return 0;
                }
                else{
                    PC=find(list_symbol,target);
                }
            }
            else if(strcmp(opcode,"BRZP")==0){
                returnjumptarget(&Instruction[PC][start_instr+5]);
                string target(jump_target);
                if(find(list_symbol,target)==0){
                    printf("Error!,No such symbol\n");
                    return 0;
                }
                else if(ZERO|POSITIVE){
```

```
                PC=find(list_symbol,target);
            }
            else{
                PC++;
            }
        }
        else if(strcmp(opcode,"BRNP")==0){
            returnjumptarget(&Instruction[PC][start_instr+5]);
            string target(jump_target);
            if(find(list_symbol,target)==0){
                printf("Error!,No such symbol\n");
                return 0;
            }
            else if(NAGATIVE|POSITIVE){
                PC=find(list_symbol,target);
            }
            else{
                PC++;
            }
        }
        else if(strcmp(opcode,"BRNZ")==0){
            returnjumptarget(&Instruction[PC][start_instr+5]);
            string target(jump_target);
            if(find(list_symbol,target)==0){
                printf("Error!,No such symbol\n");
                return 0;
            }
            else if(NAGATIVE|ZERO){
                PC=find(list_symbol,target);
            }
            else{
                PC++;
            }
        }
        else if(strcmp(opcode,"BRNZP")==0){
            returnjumptarget(&Instruction[PC][start_instr+6]);
            string target(jump_target);
            if(find(list_symbol,target)==0){
                printf("Error!,No such symbol\n");
                return 0;
            }
            else{
                PC=find(list_symbol,target);
            }
        }
        else{
            printf("Wrong branch instruction\n");
            return 0;
        }

    }
    else if(strcmp(opcode,JMP)==0){
        DP=Instruction[PC][start_instr+5]-'0';
        PC=Register[DP];
    }
    else if(strcmp(opcode,JSR)==0){
        returnjumptarget(&Instruction[PC][start_instr+4]);
        string target(jump_target);
```

```
            Register[7]=PC+1;
            if(find(list_symbol,target)==0){
                printf("Error!,No such symbol\n");
                return 0;
            }
            else{
                PC=find(list_symbol,target);
            }
        }
        else if(strcmp(opcode,JSRR)==0){
            DP=Instruction[PC][6+start_instr]-'0';
            Register[7]=PC+1;
            PC=Register[DP];
        }
        else if(strcmp(opcode,LD)==0){
            int offset=0;
            for(int i=6;Instruction[PC][i+start_instr]!='\0';i++){
                offset=offset*10;
                offset+=Instruction[PC][i+start_instr]-'0';
            }
            DP=Instruction[PC][4+start_instr]-'0';
            Register[DP]=memory[PC+1+offset];
            ZERO=false;
            NAGATIVE=false;
            POSITIVE=false;
            if(Register[DP]>0){
                POSITIVE=true;
            }
            else if(Register[DP]==0){
                ZERO=true;
            }
            else{
                NAGATIVE=true;
            }
            PC++;
        }
        else if(strcmp(opcode,LDI)==0){
            int offset=0;
            for(int i=7;Instruction[PC][i+start_instr]!='\0';i++){
                offset=offset*10;
                offset+=Instruction[PC][i+start_instr]-'0';
            }
            DP=Instruction[PC][5+start_instr]-'0';
            Register[DP]=memory[memory[PC+1+offset]];
            ZERO=false;
            NAGATIVE=false;
            POSITIVE=false;
            if(Register[DP]>0){
                POSITIVE=true;
            }
            else if(Register[DP]==0){
                ZERO=true;
            }
            else{
                NAGATIVE=true;
            }
            PC++;
        }
```

```
        else if(strcmp(opcode,LDR)==0){
            int offset=0;
            for(int i=10;Instruction[PC][i+start_instr]!='\0';i++){
                offset=offset*10;
                offset+=Instruction[PC][i+start_instr]-'0';
            }
            DP=Instruction[PC][5+start_instr]-'0';
            SR1=Instruction[PC][8+start_instr]-'0';
            Register[DP]=memory[Register[SR1]+offset];
            ZERO=false;
            NAGATIVE=false;
            POSITIVE=false;
            if(Register[DP]>0){
                POSITIVE=true;
            }
            else if(Register[DP]==0){
                ZERO=true;
            }
            else{
                NAGATIVE=true;
            }
            PC++;
        }
        else if(strcmp(opcode,LEA)==0){
            int offset=0;
            for(int i=7;Instruction[PC][i+start_instr]!='\0';i++){
                offset=offset*10;
                offset+=Instruction[PC][i+start_instr]-'0';
            }
            DP=Instruction[PC][start_instr+5]-'0';
            Register[DP]=offset+PC+1;
            ZERO=false;
            NAGATIVE=false;
            POSITIVE=false;
            if(Register[DP]>0){
                POSITIVE=true;
            }
            else if(Register[DP]==0){
                ZERO=true;
            }
            else{
                NAGATIVE=true;
            }
            PC++;
        }
        else if(strcmp(opcode,NOT)==0){
            DP=Instruction[PC][start_instr+5]-'0';
            SR1=Instruction[PC][start_instr+8]-'0';
            Register[DP]=!Register[SR1];
        }
        else if(strcmp(opcode,RET)==0){
            PC=Register[7];
        }
        else if(strcmp(opcode,RTI)==0){
            printf("Error!\n");
            return 0;
        }
        else if(strcmp(opcode,ST)==0){
```

```
            int offset=0;
            for(int i=7;Instruction[PC][i+start_instr]!='\0';i++){
                offset=offset*10;
                offset+=Instruction[PC][i+start_instr]-'0';
            }
            DP=Instruction[PC][start_instr+4]-'0';
            memory[PC+1+offset]=Register[DP];
            PC++;
        }
        else if(strcmp(opcode,STR)==0){
            int offset=0;
            for(int i=10;Instruction[PC][i+start_instr]!='\0';i++){
                offset=offset*10;
                offset+=Instruction[PC][i+start_instr]-'0';
            }
            DP=Instruction[PC][start_instr+5]-'0';
            SR1=Instruction[PC][start_instr+8]-'0';
            memory[Register[SR1]+offset]=Register[DP];
            PC++;
        }
        else if(strcmp(opcode,TRAP)==0){
            printf("Not support for System call");
        }
        else{
            printf("Invalid Instruction and will return\n");
            return 0;
        }
    }
    //printf("%d",sizeof(short));
    while(1){
        printf("If you want to see the memory,put M;\nif you want to see the
register_file,put RF;\n if you want to see the PC,put PC\n No request,put
No\n");
        char command[5];
        scanf("%s",command);
        if(strcmp(command,"PC")==0){
            printf("The PC is %d\n",PC);

        }
        else if(strcmp(command,"M")==0){
            int memadd;
            printf("Input the Memory addr\n");
            scanf("%d",&memadd);
            printf("The Memory location is %d\n",memory[memadd]);
        }
        else if(strcmp(command,"RF")==0){
            int memadd;
            printf("Input the Register addr\n");
            scanf("%d",&memadd);
            printf("The Register location is %d\n",Register[memadd]);
        }
        else if(strcmp(command,"No")==0){
            printf("GoodBye\n");
            break ;
        }
        else{
            printf("Not valid\n");
        }
```

```
    }
    return 0;
}
```

关于我们定义的用于模拟计算机部件C语言元素

名称	作用
Instruction	存储读入的字符串形式的指令
memory	模拟内存
Register	模拟通用寄存器
NAGATIVE	N标志位
ZERO	Z标志位
POSITIVE	P标志位
PC	Program Counter

模拟器可以实现的功能

- 断点设置，设置PC断点，使得程序运行到目的PC退出
- 在程序退出后(正常退出或是运行到断点)之后可以查看内存单元的值以及寄存器的值

模拟器运行的基本步骤

- 读入指令并加以处理，这里主要是替换标号，换成立即数
- 设置断点
- 执行指令
    - PC指向程序开始点
    - 取指令，判断指令的种类
    - 执行指令，执行算数逻辑运算或者访存操作
    - 修改程序状态寄存器N,Z,P的值
    - 修改Program Counter，顺序递增或者控制流跳转

执行结果截图

```
E:\code\ICS\Demo\cmake-build-debug\Demo.exe
.ORIG 1000
ADD R1,R2,#1
The Instruction is ADD R1,R2,#1
ADD R1,R1,R1
The Instruction is ADD R1,R1,R1
ST R1,#3
The Instruction is ST R1,#3
HALT
Input the breakpoint1002
This is the end of Analyze and We can answer your request
If you want to see the memory,put M;
if you want to see the register_file,put RF;
 if you want to see the PC,put PC
 No request,put No
M
Input the Memory addr
1006
The Memory location is 0
If you want to see the memory,put M;
if you want to see the register_file,put RF;
 if you want to see the PC,put PC
 No request,put No
1005
Not valid
If you want to see the memory,put M;
if you want to see the register_file,put RF;
 if you want to see the PC,put PC
 No request,put No
M
Input the Memory addr
1005
The Memory location is 0
```

我利用的CMake构建工程，所以发送工程压缩包，请用Clion打开