**计组设计报告 程序源代码**

网络工程161班 陈扬

**目录**

[头文件 memory.h 2](#_Toc526724332)

[cpp文件 memory.cpp 3](#_Toc526724333)

[头文件 register.h 5](#_Toc526724334)

[cpp文件 register.cpp 6](#_Toc526724335)

[头文件 fetch.h 8](#_Toc526724336)

[cpp文件 fetch.cpp 9](#_Toc526724337)

[头文件 decoding.h 11](#_Toc526724338)

[cpp文件 decoding.cpp 12](#_Toc526724339)

[头文件 execute.h 18](#_Toc526724340)

[cpp文件 execute.cpp 20](#_Toc526724341)

[头文件 log.h 36](#_Toc526724342)

[cpp文件 log.cpp 37](#_Toc526724343)

[头文件 panel.h 44](#_Toc526724344)

[cpp文件 panel.cpp 45](#_Toc526724345)

[头文件 overdata.h 54](#_Toc526724346)

[头文件 pipeline.h 57](#_Toc526724347)

[cpp文件 pipeline.cpp 59](#_Toc526724348)

[头文件 control.h 70](#_Toc526724349)

[cpp文件 control.cpp 72](#_Toc526724350)

[cpp文件 main.cpp 90](#_Toc526724351)

**头文件 memory.h**

#pragma once

#include "overalldata.h"

class CMemory

{

public:char mem[MEMORY\_NUM]; //内存

private:

int instruction\_size; //内存指令区域的长度

int data\_size; //内存数据区域的长度

int memory\_state; //当前内存状态

int current\_instruction\_address; //当前指令地址

public:

CMemory(); //默认构造函数

~CMemory(); //析构函数

int SetMemoryBusy(); //设置内存状态为忙碌

int SetMemoryFree(); //设置内存状态为空闲

int ReturnCurrentState(int &); //返回内存当前状态

int InitAllMemoryWith0(); //初始化全部内存 0

int InitAllMemoryWithValue(int); //初始化全部内存 指定值

int WriteToMemory(int, char); //向指定内存写入数据

int ReadFromMemory(int, char &); //从指定内存读取数据

int ResetCurrentInstructionAddress(); //重置当前的添加指令位置

int GetCurrentInstructionAddress(); //获取当前的添加指令位置

int AddInstructionAddress(); //指令位置自增

};

**cpp文件 memory.cpp**

#include "memory.h"

CMemory::CMemory()

{

SetMemoryFree(); //内存默认为空闲状态

InitAllMemoryWithValue(MEMORY\_INIT\_DATA); //默认初始化全部内存为0

instruction\_size = MEMORY\_INSTRUCTION\_SIZE; //存放指令区域大小

data\_size = MEMORY\_DATA\_SIZE; //存放数据区域大小

current\_instruction\_address = 0; //指令区域当前的起始位置

}

CMemory::~CMemory()

{

}

int CMemory::SetMemoryBusy()

{

memory\_state = MEMORY\_BUSY;

return 0;

}

int CMemory::SetMemoryFree()

{

memory\_state = MEMORY\_FREE;

return 0;

}

int CMemory::ReturnCurrentState(int &state)

{

state = memory\_state;

return 0;

}

int CMemory::InitAllMemoryWith0()

{

for (int i = instruction\_size; i < MEMORY\_NUM; i++)

mem[i] = 0;

return 0;

}

int CMemory::InitAllMemoryWithValue(int init\_num)

{

for (int i = instruction\_size; i < MEMORY\_NUM; i++)

mem[i] = init\_num;

return 0;

}

int CMemory::WriteToMemory(int memory\_num, char write\_data)

{

mem[memory\_num] = write\_data;

return 0;

}

int CMemory::ReadFromMemory(int memory\_num, char &read\_data)

{

read\_data = mem[memory\_num];

return 0;

}

int CMemory::ResetCurrentInstructionAddress()

{

current\_instruction\_address = 0;

return 0;

}

int CMemory::GetCurrentInstructionAddress()

{

return current\_instruction\_address;

}

int CMemory::AddInstructionAddress()

{

current\_instruction\_address += 4;

return 0;

}

**头文件 register.h**

#pragma once

#include "overalldata.h"

#include <string>

using namespace std;

class CRegister

{

public:int reg[REGISTER\_NUM]; //所有的32个寄存器

private:

int register\_state; //寄存器当前状态

public:

CRegister(); //默认构造函数

~CRegister(); //析构函数

int SetRegisterBusy(); //设置寄存器为忙碌

int SetRegisterFree(); //设置寄存器为空闲

int ReturnCurrentState(int &); //返回寄存器当前状态

int InitAllRegisterWith0(); //初始化全部寄存器 0

int InitAllRegisterWithValue(int); //初始化全部寄存器 指定值

int WriteToOneRegister(int, int); //向指定的寄存器中写入数据

int ReadFromOneRegister(int, int &); //从指定的寄存器中读取数据

};

**cpp文件 register.cpp**

#include "register.h"

CRegister::CRegister()

{

SetRegisterFree(); //寄存器开始默认不被使用

InitAllRegisterWith0(); //将所有寄存器默认初始为0

reg[0] = 0; //$0恒为0

}

CRegister::~CRegister()

{

}

int CRegister::SetRegisterBusy()

{

register\_state = REGISTER\_BUSY;

return 0;

}

int CRegister::SetRegisterFree()

{

register\_state = REGISTER\_FREE;

return 0;

}

int CRegister::ReturnCurrentState(int &state)

{

state = register\_state;

return 0;

}

int CRegister::InitAllRegisterWithValue(int init\_num) //注意！$0恒为0，不能够初始化

{

for (int i = 1/\*将$0寄存器排除\*/; i < REGISTER\_NUM; i++)

reg[i] = init\_num;

return 0;

}

int CRegister::InitAllRegisterWith0() //注意！$0恒为0，不能够初始化

{

for (int i = 1/\*将$0寄存器排除\*/; i < REGISTER\_NUM; i++)

reg[i] = 0;

return 0;

}

int CRegister::WriteToOneRegister(int register\_num, int write\_num)

{

reg[register\_num] = write\_num;

return 0;

}

int CRegister::ReadFromOneRegister(int register\_num, int &read\_data)

{

read\_data = reg[register\_num];

return 0;

}

**头文件 fetch.h**

#pragma once

#include "overalldata.h"

class CFetch

{

public:int PC; //PC指针

private:

int fetch\_state; //当前取指令状态

public:

CFetch(); //默认构造函数

~CFetch(); //析构函数

int SetFetchBusy(); //设置当前取指状态为忙碌

int SetFetchFree(); //设置当前取指状态为空闲

int ReturnCurrentState(int &); //返回取指类当前状态

int GetPC(); //获取PC指针的值

int PCSelfAdd(); //PC指针自增

int EditPC(int); //修改PC指针的值

int ResetPC(); //重置PC指针的值

int FetchInstructionFromMemory(int, char[], int &); //从内存的指定区域取指令

};

**cpp文件 fetch.cpp**

#include "fetch.h"

CFetch::CFetch()

{

SetFetchFree(); //将取指状态默认设置为空闲

PC = FETCH\_PC\_INIT\_DATA; //初始化PC指针的值

}

CFetch::~CFetch()

{

}

int CFetch::SetFetchBusy()

{

fetch\_state = FETCH\_BUSY;

return 0;

}

int CFetch::SetFetchFree()

{

fetch\_state = FETCH\_FREE;

return 0;

}

int CFetch::ReturnCurrentState(int &state)

{

state = fetch\_state;

return 0;

}

int CFetch::GetPC()

{

return PC;

}

int CFetch::PCSelfAdd()

{

PC += PC\_ADD\_VALUE;

return 0;

}

int CFetch::EditPC(int edit\_num)

{

PC = edit\_num;

return 0;

}

int CFetch::ResetPC()

{

PC = FETCH\_PC\_INIT\_DATA;

return 0;

}

int CFetch::FetchInstructionFromMemory(int memory\_num, char mem[], int &instruction\_data)

{

char data;

unsigned int data\_transfer;

for (int i = 0; i < 4; i++)

{

data = mem[memory\_num + i];

data\_transfer = (unsigned int)data;

data\_transfer &= 0x00ff; //去掉前面的符号位扩展

instruction\_data |= (data\_transfer << (8 \* (3 - i)));

}

return 0;

}

**头文件 decoding.h**

#pragma once

#include "overalldata.h"

class CDecoding

{

private:

int decoding\_state; //当前译码状态

public:

CDecoding(); //默认构造函数

~CDecoding(); //析构函数

int SetDecodingBusy(); //设置当前译码状态为忙碌

int SetDecodingFree(); //设置当前译码状态为空闲

int ReturnCurrentState(int &); //返回取指类当前状态

int BitSelectTransform(int, int, int, int &); //提取从m位到n位的值

int InstructionsDecoding(int, InstructionStruct &, int &); //对指令进行译码

bool InstructionCheckZero(int instruction\_type, InstructionStruct data); //检测该指令是否为写入$zero0寄存器的指令

};

**cpp文件 decoding.cpp**

#include "decoding.h"

CDecoding::CDecoding()

{

SetDecodingFree(); //默认设置译码为空闲

}

CDecoding::~CDecoding()

{

}

int CDecoding::SetDecodingBusy()

{

decoding\_state = DECODING\_BUSY;

return 0;

}

int CDecoding::SetDecodingFree()

{

decoding\_state = DECODING\_FREE;

return 0;

}

int CDecoding::ReturnCurrentState(int &state)

{

state = decoding\_state;

return 0;

}

int CDecoding::BitSelectTransform(int instruction, int m, int n, int &result)

{

int bit, bitnum = 0;

for (int i = m; i <= n; i++)

{

bit = (instruction >> i) & 1;

if (bit == 1)

bitnum |= (1 << (i - m));

else if (bit == 0)

bitnum &= (~(1 << (i - m)));

}

result = bitnum;

return 0;

}

int CDecoding::InstructionsDecoding(int instru, InstructionStruct &data,int &instru\_type)

{

int op, rs, rt, rd, shamt, func, immediate, address;

BitSelectTransform(instru, 26, 31, op);

BitSelectTransform(instru, 21, 25, rs);

BitSelectTransform(instru, 16, 20, rt);

BitSelectTransform(instru, 11, 15, rd);

BitSelectTransform(instru, 6, 10, shamt);

BitSelectTransform(instru, 0, 5, func);

BitSelectTransform(instru, 0, 15, immediate);

BitSelectTransform(instru, 0, 25, address);

switch (op)

{

case R\_TYPE\_OP: //R-Type

switch (func)

{

case R\_TYPE\_ADDU\_FUNC: //R-Type addu指令

data.rs = rs;

data.rt = rt;

data.rd = rd;

data.shamt = SHAMT\_ERROR\_CODE;

data.immediate = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = R\_TYPE\_ADDU\_NO;

break;

case R\_TYPE\_SUBU\_FUNC: //R-Type subu指令

data.rs = rs;

data.rt = rt;

data.rd = rd;

data.shamt = SHAMT\_ERROR\_CODE;

data.immediate = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = R\_TYPE\_SUBU\_NO;

break;

case R\_TYPE\_AND\_FUNC: //R-Type and指令

data.rs = rs;

data.rt = rt;

data.rd = rd;

data.shamt = SHAMT\_ERROR\_CODE;

data.immediate = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = R\_TYPE\_AND\_NO;

break;

case R\_TYPE\_OR\_FUNC: //R-Type or指令

data.rs = rs;

data.rt = rt;

data.rd = rd;

data.shamt = SHAMT\_ERROR\_CODE;

data.immediate = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = R\_TYPE\_OR\_NO;

break;

case R\_TYPE\_XOR\_FUNC: //R-Type xor指令

data.rs = rs;

data.rt = rt;

data.rd = rd;

data.shamt = SHAMT\_ERROR\_CODE;

data.immediate = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = R\_TYPE\_XOR\_NO;

break;

case R\_TYPE\_NOR\_FUNC: //R-Type nor指令

data.rs = rs;

data.rt = rt;

data.rd = rd;

data.shamt = SHAMT\_ERROR\_CODE;

data.immediate = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = R\_TYPE\_NOR\_NO;

break;

case R\_TYPE\_SLL\_FUNC: //R-Type sll指令

data.rs = 0x00;

data.rt = rt;

data.rd = rd;

data.shamt = shamt;

data.immediate = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = R\_TYPE\_SLL\_NO;

break;

case R\_TYPE\_SRL\_FUNC: //R-Type srl指令

data.rs = 0x00;

data.rt = rt;

data.rd = rd;

data.shamt = shamt;

data.immediate = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = R\_TYPE\_SRL\_NO;

break;

case R\_TYPE\_JR\_FUNC: //R-Type jr指令

data.rs = rs;

data.rt = RS\_RT\_RD\_ERROR\_CODE;

data.rd = RS\_RT\_RD\_ERROR\_CODE;

data.shamt = SHAMT\_ERROR\_CODE;

data.immediate = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = R\_TYPE\_JR\_NO;

break;

}

break;

case I\_TYPE\_ADDI: //I-Type addi指令

data.rs = rs;

data.rt = rt;

data.immediate = immediate;

data.rd = RS\_RT\_RD\_ERROR\_CODE;

data.shamt = SHAMT\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = I\_TYPE\_ADDI\_NO;

break;

case I\_TYPE\_ANDI: //I-Type andi指令

data.rs = rs;

data.rt = rt;

data.immediate = immediate;

data.rd = RS\_RT\_RD\_ERROR\_CODE;

data.shamt = SHAMT\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = I\_TYPE\_ANDI\_NO;

break;

case I\_TYPE\_ORI: //I-Type ori指令

data.rs = rs;

data.rt = rt;

data.immediate = immediate;

data.rd = RS\_RT\_RD\_ERROR\_CODE;

data.shamt = SHAMT\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = I\_TYPE\_ORI\_NO;

break;

case I\_TYPE\_XORI: //I-Type xori指令

data.rs = rs;

data.rt = rt;

data.immediate = immediate;

data.rd = RS\_RT\_RD\_ERROR\_CODE;

data.shamt = SHAMT\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = I\_TYPE\_XORI\_NO;

break;

case I\_TYPE\_LW: //I-Type lw指令

data.rs = rs;

data.rt = rt;

data.immediate = immediate;

data.rd = RS\_RT\_RD\_ERROR\_CODE;

data.shamt = SHAMT\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = I\_TYPE\_LW\_NO;

break;

case I\_TYPE\_SW: //I-Type sw指令

data.rs = rs;

data.rt = rt;

data.immediate = immediate;

data.rd = RS\_RT\_RD\_ERROR\_CODE;

data.shamt = SHAMT\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = I\_TYPE\_SW\_NO;

break;

case I\_TYPE\_BEQ: //I-Type beq指令

data.rs = rs;

data.rt = rt;

data.immediate = immediate;

data.rd = RS\_RT\_RD\_ERROR\_CODE;

data.shamt = SHAMT\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = I\_TYPE\_BEQ\_NO;

break;

case J\_TYPE\_J: //J-Type j指令

data.address = address;

data.rs = RS\_RT\_RD\_ERROR\_CODE;

data.rt = RS\_RT\_RD\_ERROR\_CODE;

data.rd = RS\_RT\_RD\_ERROR\_CODE;

data.shamt = SHAMT\_ERROR\_CODE;

data.immediate = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = J\_TYPE\_J\_NO;

break;

default:

data.rs = RS\_RT\_RD\_ERROR\_CODE;

data.rt = RS\_RT\_RD\_ERROR\_CODE;

data.rd = RS\_RT\_RD\_ERROR\_CODE;

data.shamt = SHAMT\_ERROR\_CODE;

data.immediate = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

data.address = IMMEDIATE\_ADDRESS\_ERROR\_CODE;

instru\_type = INSTRUCTION\_TYPE\_ERROR\_CODE;

break;

}

return 0;

}

bool CDecoding::InstructionCheckZero(int instruction\_type, InstructionStruct data)

{

switch (instruction\_type)

{

case R\_TYPE\_ADDU\_NO:

case R\_TYPE\_SUBU\_NO:

case R\_TYPE\_AND\_NO:

case R\_TYPE\_OR\_NO:

case R\_TYPE\_XOR\_NO:

case R\_TYPE\_NOR\_NO:

case R\_TYPE\_SLL\_NO:

case R\_TYPE\_SRL\_NO:

if (data.rd == 0)

return true;

break;

case R\_TYPE\_JR\_NO:

break;

case I\_TYPE\_ADDI\_NO:

case I\_TYPE\_ANDI\_NO:

case I\_TYPE\_ORI\_NO:

case I\_TYPE\_XORI\_NO:

case I\_TYPE\_LW\_NO:

if (data.rt == 0)

return true;

break;

case I\_TYPE\_SW\_NO:

break;

case I\_TYPE\_BEQ\_NO:

case J\_TYPE\_J\_NO:

break;

default:

break;

}

return false;

}

**头文件 execute.h**

#pragma once

#include <string>

#include "overalldata.h"

using namespace std;

class CExecute

{

private:

int execute\_state; //当前执行的状态

int instruction\_num; //当前指令数量

public:

CExecute(); //默认构造函数

~CExecute(); //析构函数

int SetExecuteBusy(); //设置当前执行的状态为忙碌

int SetExecuteFree(); //设置当前执行的状态为空闲

int ReturnCurrentState(int &); //返回取指类当前状态

int sign\_extended(int &); //对数据进行符号位扩展

int zero\_extended(int &); //对数据进行0扩展

//R-Type

int execute\_addu(InstructionStruct, int[]); //addu指令

int execute\_subu(InstructionStruct, int[]); //subu指令

int execute\_and(InstructionStruct, int[]); //and指令

int execute\_or(InstructionStruct, int[]); //or指令

int execute\_xor(InstructionStruct, int[]); //xor指令

int execute\_nor(InstructionStruct, int[]); //nor指令

int execute\_sll(InstructionStruct, int[]); //sll指令

int execute\_srl(InstructionStruct, int[]); //srl指令

int execute\_jr(InstructionStruct, int[], int &); //jr指令

//I-Type

int execute\_addi(InstructionStruct, int[]); //addi指令

int execute\_andi(InstructionStruct, int[]); //andi指令

int execute\_ori(InstructionStruct, int[]); //ori指令

int execute\_xori(InstructionStruct, int[]); //xori指令

int execute\_lw(InstructionStruct, int[], char[]); //lw指令

int execute\_sw(InstructionStruct, int[], char[]); //sw指令

int execute\_beq(InstructionStruct, int[], int &); //beq指令

//J-Type

int execute\_j(InstructionStruct, int &); //j指令

//指令执行控制

int ExecuteControl(int instruction\_type, InstructionStruct instruction\_data, char memory[], int reg[], int &PC);

int ExecuteControlCheckIfOk(int instruction\_type, InstructionStruct instruction\_data, char memory[], int reg[], int &PC);

int ExecuteControlAction(int instruction\_type, InstructionStruct instruction\_data, char memory[], int reg[], int &PC);

//指令过程预知

string KnowExecuteDetail(int instruction\_type, InstructionStruct data, char mem[], int reg[], int &PC);

int KnowExecuteResult(int instruction\_type, InstructionStruct data, char mem[], int reg[], int &PC);

int KnowMemoryReadContent(int instruction\_type, InstructionStruct data, char mem[], int reg[]);

string KnowWhichToWriteTo(int instruction\_type, InstructionStruct data, char mem[], int reg[]);

};

**cpp文件 execute.cpp**

#include "execute.h"

CExecute::CExecute()

{

SetExecuteFree(); //设置执行类的状态为空闲

}

CExecute::~CExecute()

{

}

int CExecute::SetExecuteBusy()

{

execute\_state = EXECUTE\_BUSY;

return 0;

}

int CExecute::SetExecuteFree()

{

execute\_state = EXECUTE\_FREE;

return 0;

}

int CExecute::ReturnCurrentState(int &state)

{

state = execute\_state;

return 0;

}

int CExecute::sign\_extended(int &num)

{

int sign = (num >> 15) & 1; //取符号

for (int i = 16; i <= 31; i++) {

if (sign == 1) num |= (1 << i);

else num &= ~(1 << i);

}

return 0;

}

int CExecute::zero\_extended(int & num)

{

for (int i = 16; i <= 31; i++) {

num &= ~(1 << i);

}

return 0;

}

int CExecute::execute\_addu(InstructionStruct data, int reg[])

{

reg[data.rd] = (unsigned int)reg[data.rs] + (unsigned int)reg[data.rt];

return 0;

}

int CExecute::execute\_subu(InstructionStruct data, int reg[])

{

reg[data.rd] = (unsigned int)reg[data.rs] - (unsigned int)reg[data.rt];

return 0;

}

int CExecute::execute\_and(InstructionStruct data, int reg[])

{

reg[data.rd] = reg[data.rs] & reg[data.rt];

return 0;

}

int CExecute::execute\_or(InstructionStruct data, int reg[])

{

reg[data.rd] = reg[data.rs] | reg[data.rt];

return 0;

}

int CExecute::execute\_xor(InstructionStruct data, int reg[])

{

reg[data.rd] = reg[data.rs] ^ reg[data.rt];

return 0;

}

int CExecute::execute\_nor(InstructionStruct data, int reg[])

{

reg[data.rd] = !(reg[data.rs] | reg[data.rt]);

return 0;

}

int CExecute::execute\_sll(InstructionStruct data, int reg[])

{

reg[data.rd] = reg[data.rt] << data.shamt;

return 0;

}

int CExecute::execute\_srl(InstructionStruct data, int reg[])

{

reg[data.rd] = reg[data.rt] >> data.shamt;

return 0;

}

int CExecute::execute\_jr(InstructionStruct data, int reg[], int &PC)

{

PC = reg[data.rs];

return 0;

}

int CExecute::execute\_addi(InstructionStruct data, int reg[])

{

int change = data.immediate;

sign\_extended(change);

reg[data.rt] = reg[data.rs] + change;

return 0;

}

int CExecute::execute\_andi(InstructionStruct data, int reg[])

{

int change = data.immediate;

zero\_extended(change);

reg[data.rt] = reg[data.rs] & change;

return 0;

}

int CExecute::execute\_ori(InstructionStruct data, int reg[])

{

int change = data.immediate;

zero\_extended(change);

reg[data.rt] = reg[data.rs] | change;

return 0;

}

int CExecute::execute\_xori(InstructionStruct data, int reg[])

{

int change = data.immediate;

zero\_extended(change);

reg[data.rt] = reg[data.rs] ^ change;

return 0;

}

int CExecute::execute\_lw(InstructionStruct data, int reg[], char mem[])

{

int change = data.immediate;

sign\_extended(change);

reg[data.rt] = mem[reg[data.rs] + change];

return 0;

}

int CExecute::execute\_sw(InstructionStruct data, int reg[], char mem[])

{

int change = data.immediate;

sign\_extended(change);

mem[reg[data.rs] + change] = reg[data.rt];

return 0;

}

int CExecute::execute\_beq(InstructionStruct data, int reg[], int &PC)

{

int change = data.immediate;

sign\_extended(change);

if (reg[data.rs] == reg[data.rt])

PC = PC + (change << 2);

return 0;

}

int CExecute::execute\_j(InstructionStruct data, int &PC)

{

PC = data.address;

return 0;

}

int CExecute::ExecuteControl(int instruction\_type, InstructionStruct instruction\_data, char memory[], int reg[], int & PC)

{

int instruction\_check = ExecuteControlCheckIfOk(instruction\_type, instruction\_data, memory, reg, PC);

int action\_check;

if (instruction\_check != EXECUTE\_CONTROL\_CHECK\_SUCCESS)

return instruction\_check;

else

{

action\_check = ExecuteControlAction(instruction\_type, instruction\_data, memory, reg, PC);

return action\_check;

}

}

int CExecute::ExecuteControlCheckIfOk(int instruction\_type, InstructionStruct instruction\_data, char memory[], int reg[], int & PC)

{

//检测指令是否非法

if (instruction\_type == INSTRUCTION\_TYPE\_ERROR\_CODE)

return EXECUTE\_CONTROL\_TYPE\_ERROR;

switch (instruction\_type)

{

case R\_TYPE\_ADDU\_NO:

//检测数据是否合法

if (instruction\_data.rd == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rd > 31 || instruction\_data.rd < 0)

return EXECUTE\_CONTROL\_RD\_ERROR;

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

break;

case R\_TYPE\_SUBU\_NO:

//检测数据是否合法

if (instruction\_data.rd == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rd > 31 || instruction\_data.rd < 0)

return EXECUTE\_CONTROL\_RD\_ERROR;

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

break;

case R\_TYPE\_AND\_NO:

//检测数据是否合法

if (instruction\_data.rd == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rd > 31 || instruction\_data.rd < 0)

return EXECUTE\_CONTROL\_RD\_ERROR;

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

break;

case R\_TYPE\_OR\_NO:

//检测数据是否合法

if (instruction\_data.rd == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rd > 31 || instruction\_data.rd < 0)

return EXECUTE\_CONTROL\_RD\_ERROR;

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

break;

case R\_TYPE\_XOR\_NO:

//检测数据是否合法

if (instruction\_data.rd == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rd > 31 || instruction\_data.rd < 0)

return EXECUTE\_CONTROL\_RD\_ERROR;

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

break;

case R\_TYPE\_NOR\_NO:

//检测数据是否合法

if (instruction\_data.rd == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rd > 31 || instruction\_data.rd < 0)

return EXECUTE\_CONTROL\_RD\_ERROR;

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

break;

case R\_TYPE\_SLL\_NO:

//检测数据是否合法

if (instruction\_data.rd == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rd > 31 || instruction\_data.rd < 0)

return EXECUTE\_CONTROL\_RD\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

if (instruction\_data.shamt == SHAMT\_ERROR\_CODE || instruction\_data.shamt > 31 || instruction\_data.shamt < 0)

return EXECUTE\_CONTROL\_SHAMT\_ERROR;

break;

case R\_TYPE\_SRL\_NO:

//检测数据是否合法

if (instruction\_data.rd == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rd > 31 || instruction\_data.rd < 0)

return EXECUTE\_CONTROL\_RD\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

if (instruction\_data.shamt == SHAMT\_ERROR\_CODE || instruction\_data.shamt > 31 || instruction\_data.shamt < 0)

return EXECUTE\_CONTROL\_SHAMT\_ERROR;

break;

case R\_TYPE\_JR\_NO:

//检测数据是否合法

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

break;

case I\_TYPE\_ADDI\_NO:

//检测数据是否合法

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

if (instruction\_data.immediate == IMMEDIATE\_ADDRESS\_ERROR\_CODE || instruction\_data.immediate > 65535 || instruction\_data.immediate < 0)

return EXECUTE\_CONTROL\_IMMEDIATE\_ERROR;

break;

case I\_TYPE\_ANDI\_NO:

//检测数据是否合法

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

if (instruction\_data.immediate == IMMEDIATE\_ADDRESS\_ERROR\_CODE || instruction\_data.immediate > 65535 || instruction\_data.immediate < 0)

return EXECUTE\_CONTROL\_IMMEDIATE\_ERROR;

break;

case I\_TYPE\_ORI\_NO:

//检测数据是否合法

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

if (instruction\_data.immediate == IMMEDIATE\_ADDRESS\_ERROR\_CODE || instruction\_data.immediate > 65535 || instruction\_data.immediate < 0)

return EXECUTE\_CONTROL\_IMMEDIATE\_ERROR;

break;

case I\_TYPE\_XORI\_NO:

//检测数据是否合法

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

if (instruction\_data.immediate == IMMEDIATE\_ADDRESS\_ERROR\_CODE || instruction\_data.immediate > 65535 || instruction\_data.immediate < 0)

return EXECUTE\_CONTROL\_IMMEDIATE\_ERROR;

break;

case I\_TYPE\_LW\_NO:

//检测数据是否合法

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

if (instruction\_data.immediate == IMMEDIATE\_ADDRESS\_ERROR\_CODE || instruction\_data.immediate > 65535 || instruction\_data.immediate < 0)

return EXECUTE\_CONTROL\_IMMEDIATE\_ERROR;

break;

case I\_TYPE\_SW\_NO:

//检测数据是否合法

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

if (instruction\_data.immediate == IMMEDIATE\_ADDRESS\_ERROR\_CODE || instruction\_data.immediate > 65535 || instruction\_data.immediate < 0)

return EXECUTE\_CONTROL\_IMMEDIATE\_ERROR;

break;

case I\_TYPE\_BEQ\_NO:

//检测数据是否合法

if (instruction\_data.rs == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rs > 31 || instruction\_data.rs < 0)

return EXECUTE\_CONTROL\_RS\_ERROR;

if (instruction\_data.rt == RS\_RT\_RD\_ERROR\_CODE || instruction\_data.rt > 31 || instruction\_data.rt < 0)

return EXECUTE\_CONTROL\_RT\_ERROR;

if (instruction\_data.immediate == IMMEDIATE\_ADDRESS\_ERROR\_CODE || instruction\_data.immediate > 65535 || instruction\_data.immediate < 0)

return EXECUTE\_CONTROL\_IMMEDIATE\_ERROR;

break;

case J\_TYPE\_J\_NO:

//检测数据是否合法

if (instruction\_data.address == IMMEDIATE\_ADDRESS\_ERROR\_CODE || (instruction\_data.address > (MEMORY\_INSTRUCTION\_SIZE / 4 - 1)) || instruction\_data.address < 0)

return EXECUTE\_CONTROL\_ADDRESS\_ERROR;

break;

default:

return EXECUTE\_CONTROL\_TYPE\_ERROR;

break;

}

return EXECUTE\_CONTROL\_CHECK\_SUCCESS;

}

int CExecute::ExecuteControlAction(int instruction\_type, InstructionStruct instruction\_data, char memory[], int reg[], int & PC)

{

switch (instruction\_type)

{

case R\_TYPE\_ADDU\_NO:

execute\_addu(instruction\_data, reg);

break;

case R\_TYPE\_SUBU\_NO:

execute\_subu(instruction\_data, reg);

break;

case R\_TYPE\_AND\_NO:

execute\_and(instruction\_data, reg);

break;

case R\_TYPE\_OR\_NO:

execute\_or(instruction\_data, reg);

break;

case R\_TYPE\_XOR\_NO:

execute\_xor(instruction\_data, reg);

break;

case R\_TYPE\_NOR\_NO:

execute\_nor(instruction\_data, reg);

break;

case R\_TYPE\_SLL\_NO:

execute\_sll(instruction\_data, reg);

break;

case R\_TYPE\_SRL\_NO:

execute\_srl(instruction\_data, reg);

break;

case R\_TYPE\_JR\_NO:

execute\_jr(instruction\_data, reg, PC);

break;

case I\_TYPE\_ADDI\_NO:

execute\_addi(instruction\_data, reg);

break;

case I\_TYPE\_ANDI\_NO:

execute\_andi(instruction\_data, reg);

break;

case I\_TYPE\_ORI\_NO:

execute\_ori(instruction\_data, reg);

break;

case I\_TYPE\_XORI\_NO:

execute\_xori(instruction\_data, reg);

break;

case I\_TYPE\_LW\_NO:

execute\_lw(instruction\_data, reg, memory);

break;

case I\_TYPE\_SW\_NO:

execute\_sw(instruction\_data, reg, memory);

break;

case I\_TYPE\_BEQ\_NO:

execute\_beq(instruction\_data, reg, PC);

break;

case J\_TYPE\_J\_NO:

execute\_j(instruction\_data, PC);

break;

default:

return EXECUTE\_CONTROL\_TYPE\_ERROR;

break;

}

return EXECUTE\_CONTROL\_ACTION\_SUCCESS;

}

string CExecute::KnowExecuteDetail(int instruction\_type, InstructionStruct data, char mem[], int reg[], int & PC)

{

string str = "";

int change = 0;

switch (instruction\_type)

{

case R\_TYPE\_ADDU\_NO:

str = str + "(unsigned)" + ALLREGISTERNAME[data.rs] + " + (unsigned)" + ALLREGISTERNAME[data.rt];

break;

case R\_TYPE\_SUBU\_NO:

str = str + "(unsigned)" + ALLREGISTERNAME[data.rs] + " - (unsigned)" + ALLREGISTERNAME[data.rt];

break;

case R\_TYPE\_AND\_NO:

str = str + ALLREGISTERNAME[data.rs] + " AND " + ALLREGISTERNAME[data.rt];

break;

case R\_TYPE\_OR\_NO:

str = str + ALLREGISTERNAME[data.rs] + " OR " + ALLREGISTERNAME[data.rt];

break;

case R\_TYPE\_XOR\_NO:

str = str + ALLREGISTERNAME[data.rs] + " XOR " + ALLREGISTERNAME[data.rt];

break;

case R\_TYPE\_NOR\_NO:

str = str + ALLREGISTERNAME[data.rs] + " NOR " + ALLREGISTERNAME[data.rt];

break;

case R\_TYPE\_SLL\_NO:

str = str + ALLREGISTERNAME[data.rt] + " << " + to\_string(data.shamt);

break;

case R\_TYPE\_SRL\_NO:

str = str + ALLREGISTERNAME[data.rt] + " >> " + to\_string(data.shamt);

break;

case R\_TYPE\_JR\_NO:

str = str + "PC <- " + ALLREGISTERNAME[data.rt];

break;

case I\_TYPE\_ADDI\_NO:

change = data.immediate;

sign\_extended(change);

str = str + ALLREGISTERNAME[data.rs] + " + " + to\_string(change);

break;

case I\_TYPE\_ANDI\_NO:

change = data.immediate;

zero\_extended(change);

str = str + ALLREGISTERNAME[data.rs] + " AND " + to\_string(change);

break;

case I\_TYPE\_ORI\_NO:

change = data.immediate;

zero\_extended(change);

str = str + ALLREGISTERNAME[data.rs] + " OR " + to\_string(change);

break;

case I\_TYPE\_XORI\_NO:

change = data.immediate;

zero\_extended(change);

str = str + ALLREGISTERNAME[data.rs] + " XOR " + to\_string(change);

break;

case I\_TYPE\_LW\_NO:

change = data.immediate;

sign\_extended(change);

str = str + "等待访存过程执行";

break;

case I\_TYPE\_SW\_NO:

change = data.immediate;

sign\_extended(change);

str = str + "等待访存过程执行";

break;

case I\_TYPE\_BEQ\_NO:

change = data.immediate;

sign\_extended(change);

if (reg[data.rs] == reg[data.rt])

str = str + "PC + " + to\_string(change) + "<<2";

else str = str + "，寄存器值不相等，不执行";

break;

case J\_TYPE\_J\_NO:

str = str + to\_string(data.address);

break;

default:

return 0;

break;

}

return str;

}

int CExecute::KnowExecuteResult(int instruction\_type, InstructionStruct data, char mem[], int reg[], int & PC)

{

int change = 0;

switch (instruction\_type)

{

case R\_TYPE\_ADDU\_NO:

return ((unsigned int)reg[data.rs] + (unsigned int)reg[data.rt]);

break;

case R\_TYPE\_SUBU\_NO:

return ((unsigned int)reg[data.rs] - (unsigned int)reg[data.rt]);

break;

case R\_TYPE\_AND\_NO:

return (reg[data.rs] & reg[data.rt]);

break;

case R\_TYPE\_OR\_NO:

return (reg[data.rs] | reg[data.rt]);

break;

case R\_TYPE\_XOR\_NO:

return (reg[data.rs] ^ reg[data.rt]);

break;

case R\_TYPE\_NOR\_NO:

return (!(reg[data.rs] | reg[data.rt]));

break;

case R\_TYPE\_SLL\_NO:

return (reg[data.rt] << data.shamt);

break;

case R\_TYPE\_SRL\_NO:

return (reg[data.rt] >> data.shamt);

break;

case R\_TYPE\_JR\_NO:

return (reg[data.rs]);

break;

case I\_TYPE\_ADDI\_NO:

change = data.immediate;

sign\_extended(change);

return (reg[data.rs] + change);

break;

case I\_TYPE\_ANDI\_NO:

change = data.immediate;

zero\_extended(change);

return (reg[data.rs] & change);

break;

case I\_TYPE\_ORI\_NO:

change = data.immediate;

zero\_extended(change);

return (reg[data.rs] | change);

break;

case I\_TYPE\_XORI\_NO:

change = data.immediate;

zero\_extended(change);

return (reg[data.rs] ^ change);

break;

case I\_TYPE\_LW\_NO:

change = data.immediate;

sign\_extended(change);

return (mem[reg[data.rs] + change]);

break;

case I\_TYPE\_SW\_NO:

change = data.immediate;

sign\_extended(change);

return (reg[data.rs] + change);

break;

case I\_TYPE\_BEQ\_NO:

change = data.immediate;

sign\_extended(change);

if (reg[data.rs] == reg[data.rt])

return (PC + (change << 2));

else return 0;

break;

case J\_TYPE\_J\_NO:

return (data.address);

break;

default:

return 0;

break;

}

return 0;

}

int CExecute::KnowMemoryReadContent(int instruction\_type, InstructionStruct data, char mem[], int reg[])

{

int change = 0;

switch (instruction\_type)

{

case I\_TYPE\_LW\_NO:

change = data.immediate;

sign\_extended(change);

return mem[reg[data.rs] + change];

case I\_TYPE\_SW\_NO:

change = data.immediate;

sign\_extended(change);

return mem[reg[data.rs] + change];

default:

return 0;

break;

}

return 0;

}

string CExecute::KnowWhichToWriteTo(int instruction\_type, InstructionStruct data, char mem[], int reg[])

{

string str = "";

int change = 0;

switch (instruction\_type)

{

case R\_TYPE\_ADDU\_NO:

str = str + ALLREGISTERNAME[data.rd];

break;

case R\_TYPE\_SUBU\_NO:

str = str + ALLREGISTERNAME[data.rd];

break;

case R\_TYPE\_AND\_NO:

str = str + ALLREGISTERNAME[data.rd];

break;

case R\_TYPE\_OR\_NO:

str = str + ALLREGISTERNAME[data.rd];

break;

case R\_TYPE\_XOR\_NO:

str = str + ALLREGISTERNAME[data.rd];

break;

case R\_TYPE\_NOR\_NO:

str = str + ALLREGISTERNAME[data.rd];

break;

case R\_TYPE\_SLL\_NO:

str = str + ALLREGISTERNAME[data.rd];

break;

case R\_TYPE\_SRL\_NO:

str = str + ALLREGISTERNAME[data.rd];

break;

case R\_TYPE\_JR\_NO:

str = str + "PC";

break;

case I\_TYPE\_ADDI\_NO:

str = str + ALLREGISTERNAME[data.rt];

break;

case I\_TYPE\_ANDI\_NO:

str = str + ALLREGISTERNAME[data.rt];

break;

case I\_TYPE\_ORI\_NO:

str = str + ALLREGISTERNAME[data.rt];

break;

case I\_TYPE\_XORI\_NO:

str = str + ALLREGISTERNAME[data.rt];

break;

case I\_TYPE\_LW\_NO:

str = str + ALLREGISTERNAME[data.rt];

break;

case I\_TYPE\_SW\_NO:

change = data.immediate;

sign\_extended(change);

str = str + "MEMORY[" + to\_string(reg[data.rs] + change) + "]";

break;

case I\_TYPE\_BEQ\_NO:

change = data.immediate;

sign\_extended(change);

if (reg[data.rs] == reg[data.rt])

str = str + "PC";

else str = str + "，寄存器值不相等，不写回";

break;

case J\_TYPE\_J\_NO:

str = str + "PC";

break;

default:

return 0;

break;

}

return str;

}

**头文件 log.h**

#pragma once

#include "overalldata.h"

#include <fstream>

#include <string>

#include <time.h>

using namespace std;

class CLog

{

private:

char regname[REGISTER\_NUM][REGISTER\_MAX\_NAME\_SIZE]; //寄存器名字

ofstream logfile; //写记录的文件

bool ifnew; //判断是否第一次写文件

int current\_log\_no; //当前记录标号

public:

CLog(); //默认构造函数

~CLog(); //析构函数

int InitRegName(); //初始化寄存器名字

string GetRegisterName(int); //获取寄存器名字

int InitLogNo(); //初始化记录标号 current\_log\_no

string GetInstructionLogDetail(int, InstructionStruct); //获取指令记录内容

string GetInstructionString(int, InstructionStruct); //获取指令内容

int WriteLog(int, InstructionStruct); //根据指令内容，写文件

void LogBegin(); //开始写文件

void LogEnd(); //结束写文件

void ClearLog(); //删除所有记录

};

**cpp文件 log.cpp**

#include "log.h"

CLog::CLog()

{

//初始化寄存器名字

InitRegName();

//初始化ifnew变量

ifnew = true;

//初始化记录数量

InitLogNo();

}

CLog::~CLog()

{

}

int CLog::InitRegName()

{

strcpy(regname[0], "$zero");

strcpy(regname[1], "$at");

strcpy(regname[2], "$v0");

strcpy(regname[3], "$v1");

strcpy(regname[4], "$a0");

strcpy(regname[5], "$a1");

strcpy(regname[6], "$a2");

strcpy(regname[7], "$a3");

strcpy(regname[8], "$t0");

strcpy(regname[9], "$t1");

strcpy(regname[10], "$t2");

strcpy(regname[11], "$t3");

strcpy(regname[12], "$t4");

strcpy(regname[13], "$t5");

strcpy(regname[14], "$t6");

strcpy(regname[15], "$t7");

strcpy(regname[16], "$s0");

strcpy(regname[17], "$s1");

strcpy(regname[18], "$s2");

strcpy(regname[19], "$s3");

strcpy(regname[20], "$s4");

strcpy(regname[21], "$s5");

strcpy(regname[22], "$s6");

strcpy(regname[23], "$s7");

strcpy(regname[24], "$t8");

strcpy(regname[25], "$t9");

strcpy(regname[26], "$k0");

strcpy(regname[27], "$k1");

strcpy(regname[28], "$gp");

strcpy(regname[29], "$sp");

strcpy(regname[30], "$fp");

strcpy(regname[31], "$ra");

return 0;

}

string CLog::GetRegisterName(int reg\_no)

{

string str;

str = regname[reg\_no];

return str;

}

int CLog::InitLogNo()

{

current\_log\_no = 1;

return 0;

}

string CLog::GetInstructionLogDetail(int instruction\_type, InstructionStruct data)

{

string str = "";

switch (instruction\_type)

{

case R\_TYPE\_ADDU\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "addu rd, rs, rt\t" + "rd <- rs + rt" + "\n";

str = str + "\t" + "rd=" + regname[data.rd] + ", rs=" + regname[data.rs] + ", rt=" + regname[data.rt] + "\n";

break;

case R\_TYPE\_SUBU\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "subu rd, rs, rt\t" + "rd <- rs - rt" + "\n";

str = str + "\t" + "rd=" + regname[data.rd] + ", rs=" + regname[data.rs] + ", rt=" + regname[data.rt] + "\n";

break;

case R\_TYPE\_AND\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "and rd, rs, rt\t" + "rd <- rs & rt" + "\n";

str = str + "\t" + "rd=" + regname[data.rd] + ", rs=" + regname[data.rs] + ", rt=" + regname[data.rt] + "\n";

break;

case R\_TYPE\_OR\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "or rd, rs, rt\t" + "rd <- rs | rt" + "\n";

str = str + "\t" + "rd=" + regname[data.rd] + ", rs=" + regname[data.rs] + ", rt=" + regname[data.rt] + "\n";

break;

case R\_TYPE\_XOR\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "xor rd, rs, rt\t" + "rd <- rs ^ rt" + "\n";

str = str + "\t" + "rd=" + regname[data.rd] + ", rs=" + regname[data.rs] + ", rt=" + regname[data.rt] + "\n";

break;

case R\_TYPE\_NOR\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "nor rd, rs, rt\t" + "rd <- ~(rs | rt)" + "\n";

str = str + "\t" + "rd=" + regname[data.rd] + ", rs=" + regname[data.rs] + ", rt=" + regname[data.rt] + "\n";

break;

case R\_TYPE\_SLL\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "sll rd, rt, shamt\t" + "rd <- rt + shamt" + "\n";

str = str + "\t" + "rd=" + regname[data.rd] + ", rt=" + regname[data.rt] + ", shamt=" + to\_string((int)data.shamt) + "\n";

break;

case R\_TYPE\_SRL\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "srl rd, rt, shamt\t" + "rd <- rt >> shamt" + "\n";

str = str + "\t" + "rd=" + regname[data.rd] + ", rt=" + regname[data.rt] + ", shamt=" + to\_string((int)data.shamt) + "\n";

break;

case R\_TYPE\_JR\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "jr rs\t" + "PC <- rs" + "\n";

str = str + "\t" + "rs=" + regname[data.rs] + "\n";

break;

case I\_TYPE\_ADDI\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "addi rs, rt, imm\t" + "rt <- rs + (sign-extend)immediate" + "\n";

str = str + "\t" + "rs=" + regname[data.rs] + ", rt=" + regname[data.rt] + ", imm=" + to\_string(data.immediate) + "\n";

break;

case I\_TYPE\_ANDI\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "andi rs, rt, imm\t" + "rt <- rs & (zero-extend)immediate" + "\n";

str = str + "\t" + "rs=" + regname[data.rs] + ", rt=" + regname[data.rt] + ", imm=" + to\_string(data.immediate) + "\n";

break;

case I\_TYPE\_ORI\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "ori rs, rt, imm\t" + "rt <- rs | (zero-extend)immediate" + "\n";

str = str + "\t" + "rs=" + regname[data.rs] + ", rt=" + regname[data.rt] + ", imm=" + to\_string(data.immediate) + "\n";

break;

case I\_TYPE\_XORI\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "xori rs, rt, imm\t" + "rt <- rs ^ (zero-extend)immediate" + "\n";

str = str + "\t" + "rs=" + regname[data.rs] + ", rt=" + regname[data.rt] + ", imm=" + to\_string(data.immediate) + "\n";

break;

case I\_TYPE\_LW\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "lw rt, imm\t" + "rt <- memory[rs + (sign-extend)immediate]" + "\n";

str = str + "\t" + "rt=" + regname[data.rt] + ", rs=" + regname[data.rs] + ", imm=" + to\_string(data.immediate) + "\n";

break;

case I\_TYPE\_SW\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "sw rt, imm\t" + "memory[rs + (sign-extend)immediate] <- rt" + "\n";

str = str + "\t" + "rs=" + regname[data.rs] + ", rt=" + regname[data.rt] + ", imm=" + to\_string(data.immediate) + "\n";

break;

case I\_TYPE\_BEQ\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "beq rs, rt, imm\t" + "if (rs == rt) PC <- PC + 4 + (sign-extend)immediate + 2" + "\n";

str = str + "\t" + "rs=" + regname[data.rs] + ", rt=" + regname[data.rt] + ", imm=" + to\_string(data.immediate) + "\n";

break;

case J\_TYPE\_J\_NO:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "j address\t" + "PC <- address" + "\n";

str = str + "\t" + "address=" + to\_string(data.address) + "\n";

break;

default:

str = str + to\_string(current\_log\_no) + "-\t";

str = str + "error instruction\t" + "can not decode";

break;

}

return str;

}

int CLog::WriteLog(int instruction\_type, InstructionStruct data)

{

//初始化FILE文件指针，读取、写入、初始位置文件尾

logfile.open("record.txt", ios::out | ios::app);

if (ifnew == false) //进行第二轮输入，则空两行

logfile << endl;

ifnew = false; //已经开始写入

time\_t calendar\_time = time(NULL);

struct tm \* tm\_local = localtime(&calendar\_time);

char time\_str[100]; //字符串，存储时间

strftime(time\_str, sizeof(time\_str), "%G-%m-%d %H:%M:%S", tm\_local);

logfile << "执行时间：" << time\_str << endl;

string str = GetInstructionLogDetail(instruction\_type, data);

logfile << str;

current\_log\_no++;

logfile.close();

return 0;

}

void CLog::LogBegin()

{

logfile.open("record.txt", ios::out | ios::app);

logfile << "-------BEGIN-------" << endl << endl;

InitLogNo();

logfile.close();

}

void CLog::LogEnd()

{

logfile.open("record.txt", ios::out | ios::app);

logfile << endl << "-------END-------" << endl << endl << endl;

InitLogNo();

ifnew = true;

logfile.close();

}

void CLog::ClearLog()

{

logfile.open("record.txt", ios::trunc); //清空文件内容

logfile.close();

}

string CLog::GetInstructionString(int instruction\_type, InstructionStruct data)

{

string str = "";

switch (instruction\_type)

{

case R\_TYPE\_ADDU\_NO:

str = str + "addu " + regname[data.rd] + ", " + regname[data.rs] + ", " + regname[data.rt];

break;

case R\_TYPE\_SUBU\_NO:

str = str + "subu " + regname[data.rd] + ", " + regname[data.rs] + ", " + regname[data.rt];

break;

case R\_TYPE\_AND\_NO:

str = str + "and " + regname[data.rd] + ", " + regname[data.rs] + ", " + regname[data.rt];

break;

case R\_TYPE\_OR\_NO:

str = str + "or " + regname[data.rd] + ", " + regname[data.rs] + ", " + regname[data.rt];

break;

case R\_TYPE\_XOR\_NO:

str = str + "xor " + regname[data.rd] + ", " + regname[data.rs] + ", " + regname[data.rt];

break;

case R\_TYPE\_NOR\_NO:

str = str + "nor " + regname[data.rd] + ", " + regname[data.rs] + ", " + regname[data.rt];

break;

case R\_TYPE\_SLL\_NO:

str = str + "sll " + regname[data.rd] + ", " + regname[data.rt] + ", " + to\_string((int)data.shamt);

break;

case R\_TYPE\_SRL\_NO:

str = str + "srl " + regname[data.rd] + ", " + regname[data.rt] + ", " + to\_string((int)data.shamt);

break;

case R\_TYPE\_JR\_NO:

str = str + "jr " + regname[data.rs];

break;

case I\_TYPE\_ADDI\_NO:

str = str + "addi " + regname[data.rs] + ", " + regname[data.rt] + ", " + to\_string(data.immediate);

break;

case I\_TYPE\_ANDI\_NO:

str = str + "andi " + regname[data.rs] + ", " + regname[data.rt] + ", " + to\_string(data.immediate);

break;

case I\_TYPE\_ORI\_NO:

str = str + "ori " + regname[data.rs] + ", " + regname[data.rt] + ", " + to\_string(data.immediate);

break;

case I\_TYPE\_XORI\_NO:

str = str + "xori " + regname[data.rs] + ", " + regname[data.rt] + ", " + to\_string(data.immediate);

break;

case I\_TYPE\_LW\_NO:

str = str + "lw " + regname[data.rt] + ", " + to\_string(data.immediate);

break;

case I\_TYPE\_SW\_NO:

str = str + "sw " + regname[data.rt] + ", " + to\_string(data.immediate);

break;

case I\_TYPE\_BEQ\_NO:

str = str + "beq " + regname[data.rs] + ", " + regname[data.rt] + ", " + to\_string(data.immediate);

break;

case J\_TYPE\_J\_NO:

str = str + "j " + to\_string(data.address);

break;

default:

str = str + "error instruction, can not decode";

break;

}

return str;

}

**头文件 panel.h**

#pragma once

#include "overalldata.h"

class CPanel

{

private:

public:

int PanelChoose();

//主菜单

int ShowMainPanel(); //显示主菜单

//选项1：指令控制

int ShowChoose1Panel(); //指令控制菜单

//选项2：指令执行

int ShowChoose2Panel(); //指令执行菜单

//选项3：内存控制

int ShowChoose3Panel(); //内存控制菜单

//选项4：寄存器控制

int ShowChoose4Panel(); //寄存器控制菜单

//选项5：流水线

int ShowChoose5Panel(); //流水线菜单

//选项6：使用说明

int ShowChoose6Panel(); //使用说明菜单

//选项7：关于

int ShowChoose7Panel(); //关于菜单

//选项8：退出

int ShowChoose8Panel(); //退出菜单

};

**cpp文件 panel.cpp**

#include <iostream>

#include "panel.h"

#include <stdlib.h>

using namespace std;

int CPanel::PanelChoose()

{

int choose\_num = 0, two\_choose\_num = 0;

choose\_num = ShowMainPanel();

switch (choose\_num)

{

case PANEL\_MAIN\_ADD + 1:

two\_choose\_num = ShowChoose1Panel();

break;

case PANEL\_MAIN\_ADD + 2:

two\_choose\_num = ShowChoose2Panel();

break;

case PANEL\_MAIN\_ADD + 3:

two\_choose\_num = ShowChoose3Panel();

break;

case PANEL\_MAIN\_ADD + 4:

two\_choose\_num = ShowChoose4Panel();

break;

case PANEL\_MAIN\_ADD + 5:

two\_choose\_num = ShowChoose5Panel();

break;

case PANEL\_MAIN\_ADD + 6:

two\_choose\_num = ShowChoose6Panel();

break;

case PANEL\_MAIN\_ADD + 7:

two\_choose\_num = ShowChoose7Panel();

break;

case PANEL\_MAIN\_ADD + 8:

two\_choose\_num = ShowChoose8Panel();

break;

default:

two\_choose\_num = PANEL\_CHOOSE\_ERROR\_CODE;

break;

}

return two\_choose\_num;

}

int CPanel::ShowMainPanel()

{

system("cls");

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MIPS32指令系统\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\* 1-指令控制 2-指令执行 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 3-内存控制 4-寄存器控制 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 5-流水线 6-使用说明 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 7-关于 8-退出 \*" << endl;

cout << "\* \*" << endl;

cout << "\* \*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << endl << "请输入执行序号： ";

int choose\_num = 0;

cin >> choose\_num;

while (choose\_num < 1 || choose\_num > 8)

{

cout << endl << "输入错误！请重新输入： ";

cin >> choose\_num;

}

return choose\_num + PANEL\_MAIN\_ADD;

}

//指令控制

int CPanel::ShowChoose1Panel()

{

system("cls");

cout << "\*\*\*\*\*\*\*\*\*\*\*指令控制\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\* \*" << endl;

cout << "\* 1-输入指令 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 2-查看指令 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 3-修改指令 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 4-清空指令 \*" << endl;

cout << "\* \*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << endl << "请输入执行序号（0返回上级菜单）： ";

int choose\_num = 0;

cin >> choose\_num;

if (choose\_num == 0)

return PANEL\_RETURN\_TO\_MAIN\_PANEL;

while (choose\_num < 1 || choose\_num > 4)

{

cout << endl << "输入错误！请重新输入： ";

cin >> choose\_num;

}

if (choose\_num == 0)

return PANEL\_RETURN\_TO\_MAIN\_PANEL;

return choose\_num + PANEL\_1\_ADD;

}

//指令执行

int CPanel::ShowChoose2Panel()

{

system("cls");

cout << "\*\*\*\*\*\*\*\*\*\*\*指令执行\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\* \*" << endl;

cout << "\* 1-全部执行 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 2-单步执行 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 3-执行记录 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 4-清除记录 \*" << endl;

cout << "\* \*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << endl << "请输入执行序号（0返回上级菜单）： ";

int choose\_num = 0;

cin >> choose\_num;

if (choose\_num == 0)

return PANEL\_RETURN\_TO\_MAIN\_PANEL;

while (choose\_num < 1 || choose\_num > 4)

{

cout << endl << "输入错误！请重新输入： ";

cin >> choose\_num;

}

if (choose\_num == 0)

return PANEL\_RETURN\_TO\_MAIN\_PANEL;

return choose\_num + PANEL\_2\_ADD;

}

//内存控制

int CPanel::ShowChoose3Panel()

{

system("cls");

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*内存控制\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\* \*" << endl;

cout << "\* 1-初始化内存（全0） \*" << endl;

cout << "\* \*" << endl;

cout << "\* 2-初始化内存（指定值） \*" << endl;

cout << "\* \*" << endl;

cout << "\* 3-修改内存 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 4-查看内存 \*" << endl;

cout << "\* \*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << endl << "请输入执行序号（0返回上级菜单）： ";

int choose\_num = 0;

cin >> choose\_num;

if (choose\_num == 0)

return PANEL\_RETURN\_TO\_MAIN\_PANEL;

while (choose\_num < 1 || choose\_num > 4)

{

cout << endl << "输入错误！请重新输入： ";

cin >> choose\_num;

}

if (choose\_num == 0)

return PANEL\_RETURN\_TO\_MAIN\_PANEL;

return choose\_num + PANEL\_3\_ADD;

}

//寄存器控制

int CPanel::ShowChoose4Panel()

{

system("cls");

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*寄存器控制\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\* \*" << endl;

cout << "\* 1-初始化寄存器（全0） \*" << endl;

cout << "\* \*" << endl;

cout << "\* 2-初始化寄存器（指定值） \*" << endl;

cout << "\* \*" << endl;

cout << "\* 3-修改寄存器 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 4-查看寄存器 \*" << endl;

cout << "\* \*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << endl << "请输入执行序号（0返回上级菜单）： ";

int choose\_num = 0;

cin >> choose\_num;

if (choose\_num == 0)

return PANEL\_RETURN\_TO\_MAIN\_PANEL;

while (choose\_num < 1 || choose\_num > 4)

{

cout << endl << "输入错误！请重新输入： ";

cin >> choose\_num;

}

if (choose\_num == 0)

return PANEL\_RETURN\_TO\_MAIN\_PANEL;

return choose\_num + PANEL\_4\_ADD;

}

//流水线

int CPanel::ShowChoose5Panel()

{

system("cls");

cout << "\*\*\*\*\*\*\*\*\*\*\*\*流水线\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\* \*" << endl;

cout << "\* 1-生成流水线 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 2-单步执行 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 3-清空流水线 \*" << endl;

cout << "\* \*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << endl << "请输入执行序号（0返回上级菜单）： ";

int choose\_num = 0;

cin >> choose\_num;

if (choose\_num == 0)

return PANEL\_RETURN\_TO\_MAIN\_PANEL;

while (choose\_num < 1 || choose\_num > 3)

{

cout << endl << "输入错误！请重新输入： ";

cin >> choose\_num;

}

if (choose\_num == 0)

return PANEL\_RETURN\_TO\_MAIN\_PANEL;

return choose\_num + PANEL\_5\_ADD;

}

int CPanel::ShowChoose6Panel()

{

system("cls");

cout << "1.1.1 指令控制" << endl;

cout << " 本功能模块负责指令的输入输出。程序在运行之前，必须有已经输入的指令序列，否则将不能够正常执行而直接退出。如果用户想要正常的使用该程序，则必须将已经准备好的指令序列输入。因为该虚拟计算机基于MIPS32指令集，所以每一条指令的长度固定为32位长度的0/1序列。" << endl;

cout << "在输入时，用户可以选择单行或多行输入，但是在输入完毕后，必须另起一行，并以”##”作为结束标识，否则程序将不能够正确读取并译码。" << endl;

cout << " 在该模块下，有4个子模块，提供更加准确的指令控制，分别为：输入指令、查看指令、修改指令、清空指令，详细介绍如下：" << endl;

cout << " （1）输入指令：提供单行或多行输入指令的功能，输入的指令将顺序存储在内存的00H-FFH区域。每一条32位的指令占用4个内存空间。" << endl;

cout << " （2）查看指令：从内存的00H开始读取，直至FFH，若读取的内存区域不为00H内容，则判断有指令。程序调用“译码”功能，将32位的指令译码为能够阅读的汇编代码。" << endl;

cout << " （3）修改指令：用户指定一个需要修改的指令地址，并给出要修改的指令。程序将根据指令地址与指令内容修改。注意，如果提供的指令地址与指令内容非法，程序将拒绝执行，要求用户重新提供正确的地址与内容。" << endl;

cout << " （4）清空指令：清空程序内存区域中指令区域的所有指令内容，即将内存00H-FFH区域全部强制填为00H。" << endl;

cout << "" << endl;

cout << "1.1.2 指令执行" << endl;

cout << " 本功能模块负责指令的执行控制。当用户输入指令后，便可以使用该功能模块模拟执行指令。注意，此功能模块所提供的指令执行是串行执行方法，与“项目简介”中所描述的“流水线”功能无关。" << endl;

cout << " 该功能模块下可细分为4个子模块，分别为：全部执行、单步执行、执行记录、清除记录，详细介绍如下：" << endl;

cout << " （1）全部执行：一步执行完（内存指令区域的）全部指令，当执行完后，程序会列出所执行过的全部指令内容 。" << endl;

cout << " （2）单步执行：每次执行一条（内存指令区域）指令，当执行完后，程序会列出所执行过的指令的指令内容。" << endl;

cout << " 在“全部执行”与“单步执行”过程中，程序每次执行一条指令，都会向”record.txt”文件中写入一条执行记录，写入方式为追加写入，每次执行都会新起一行，不会修改之前写入过的记录。执行记录详细记录了指令执行的各项细节，包括：执行时间、指令内容、指令解释、涉及到的寄存器。用户可以在程序目录下的”record.txt”文件中查看具体的细节。" << endl;

cout << " （3）执行记录：查看存储在程序同目录下”record.txt”文件中的内容，即所有的指令执行记录。" << endl;

cout << " （4）清除记录：将存储在程序同目录下”record.txt”文件中的内容全部清空。" << endl;

cout << "" << endl;

cout << "1.1.3 内存控制" << endl;

cout << " 本功能模块负责内存内容的添加修改。模拟计算机提供了内存模块，来存储与程序运行相关的数据。内存模块具体分为指令区域与数据区域，指令区域的地址范围为：00H-FFH，数据区域的地址范围为：100H-2FFH。对指令操作时，只影响指令区域，不影响数据区域；对数据操作时，只影响数据区域，不影响指令区域。" << endl;

cout << " 本功能模块下可细分为4个子模块，分别为：初始化内存（全0）、初始化内存（指定值）、修改内存、查看内存，详细介绍如下：" << endl;

cout << " （1）初始化内存（全0）：将内存数据区域100H-2FFH全部修改为00H。" << endl;

cout << " （2）初始化内存（指定值）：用户给出需要修改的指定值，程序将数据区域100H-2FFH全部修改为用户给出的指定值。" << endl;

cout << " （3）修改内存：该功能可以指定修改某一特定地址的内存值为某指定值。程序将会要求用户给出需要修改的内存地址与要修改的内存的值并做出修改。注意，若用户给出的内存地址与数据非法，程序将会拒绝执行，并要求用户重新给出可用的数据。" << endl;

cout << " （4）查看内存：程序按照内存地址顺序给出每一个地址的内容。在显示数据时，将先按照指令区域、数据区域进行分类，然后，以每行10个展示。" << endl;

cout << "" << endl;

cout << "1.1.4 寄存器控制" << endl;

cout << " 本功能模块负责寄存器内容的添加修改。模拟计算机提供了寄存器模块，来存储程序运行所需要的各个寄存器数据与寄存器本身。根据MIPS32指令集的设计，程序中共有32个不同的寄存器 ，对应不同的功能。" << endl;

cout << " 本功能模块下可细分为4个子模块，分别为：初始化寄存器（全0）、初始化寄存器（指定值）、修改寄存器、查看寄存器，详细介绍如下：" << endl;

cout << " （1）初始化寄存器（全0）：将32个寄存器全部修改为00H。" << endl;

cout << " （2）初始化寄存器（指定值）：用户给出需要修改的指定值，程序将32个寄存器统一修改为用户给出的指定值。" << endl;

cout << " （3）修改寄存器：该功能可以指定修改某一特定的寄存器值。程序将会要求用户给出需要修改的寄存器序号与要修改的寄存器值并做出修改。注意，若用户给出的寄存器序号与数据非法，程序将会拒绝执行，并要求用户重新给出可用的数据。" << endl;

cout << " （4）查看寄存器：程序读取32个寄存器的值，并将他们的数据以16进制的方式显示出来，供使用者查阅。" << endl;

cout << "" << endl;

cout << "1.1.5 流水线" << endl;

cout << " 本功能模块负责将程序中已有的指令序列使用流水线的方式展示并运行。该功能模块是整个程序的核心，也是整个程序设计的精华所在，笔者设计的程序架构、模块划分及整合以及独特的算法思想，将在这一模块中集中体现。" << endl;

cout << " 为了能够区别模拟计算机中“流水线”功能的独特与其优越性，故单独设计这一功能模块并将它和其他模块区别是有必要的。" << endl;

cout << " 该功能模块为用户详细了解程序内部流水线的运行机制提供了方法，利用图形化的方法 展示了流水线五个步骤 的详细划分。" << endl;

cout << " 本功能模块下可细分为3个功能模块，分别为：生成流水线、单步执行、清空流水线，详细介绍如下：" << endl;

cout << " （1）生成流水线：程序根据当前内存指令区域的指令，依照“流水线算法”（该算法在后文将会详细介绍），生成程序的流水线过程。由于该模拟计算机对于流水线阻滞的处理方法为暂停执行，故采用了图示的方式进行展示。注意，使用该功能，只会生成流水线图，但程序并不会真正执行指令，所以使用完该功能后，程序的PC指针位置、内存数据、寄存器数据都不会改变。同时，生成的流水线图将存储在程序目录下的”pipeline.txt”和”graphics.txt”文件中。其中，”pipeline.txt”文件为记录文件，供读者查阅使用，”graphics.txt”文件为图像生成缓存，供“流水线绘图.exe”调用绘图。" << endl;

cout << " （2）单步执行：程序根据当前内存指令区域的指令，依照“流水线算法”（该算法在后文将会详细介绍），单步生成程序的流水线过程。根据流水线时间划分的原则，使用单步执行也将逐个时间点展示具体的操作过程。每执行一步，程序将会暂停，展示流水线当前的推进情况、所有寄存器中的数据、每一条指令当前所处的状态，同时，PC指针、内存、寄存器中的值都会根据实际的情况改变。用户可以选择是否继续单步执行，若选择返回菜单，则流水线的推进过程直接结束。单步执行的记录存储于程序目录下的”pipeline\_step.txt”文件中，同时，也会生成一份图像生成缓存，存储在”graphics.txt”文件中，供“流水线绘图.exe”调用绘图。" << endl;

cout << " （3）清空流水线：程序将位于程序目录下的”pipeline.txt”、”pipeline\_step.txt”与”graphics.txt”文件全部清空。" << endl;

cout << "" << endl;

cout << "1.1.6 使用说明" << endl;

cout << " 本功能模块具体介绍了程序的使用说明与注意事项。该功能模块中所提供的使用说明与注意事项与本文档的第二部分“功能描述”完全一致，若读者已经熟读并记住了本节内容，则不必再重复查看程序的这一部分。" << endl;

cout << "" << endl;

cout << "1.1.7 关于" << endl;

cout << " 本功能模块介绍了程序的作者信息。具体的信息有：项目名称、作者、学号、班级。" << endl;

cout << "" << endl;

cout << "1.1.8 退出" << endl;

cout << " 本功能模块负责对程序所使用的内存空间进行释放，并安全的结束程序运行，退出程序。" << endl;

cout << "" << endl;

cout << "1.1.9 流水线绘图" << endl;

cout << " “流水线绘图”程序是一个单独的、使用C语言编写的、基于graphics图形库的程序，目的是将图像生成缓存生成为具体的流水线图形图像。对于“流水线绘图”程序的设计方法将在后文作具体的介绍。该程序没有具体的使用说明，只需要执行即可。注意，要使得改程序能够正常执行，请保证在该程序目录下有”graphics.txt”文档以供读取，同时，也要保证”graphics.txt”的格式正确。" << endl;

cout << endl << "输入任意字符返回上级菜单： ";

int choose\_num = 0;

cin >> choose\_num;

return PANEL\_6\_ADD + 1;

}

int CPanel::ShowChoose7Panel()

{

system("cls");

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*关于\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\* \*" << endl;

cout << "\* 项目：MIPS32指令系统 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 作者：陈扬 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 学号：19316117 \*" << endl;

cout << "\* \*" << endl;

cout << "\* 班级：网络工程161班 \*" << endl;

cout << "\* \*" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << endl << "输入任意字符返回上级菜单： ";

int choose\_num = 0;

cin >> choose\_num;

return PANEL\_7\_ADD + 1;

}

int CPanel::ShowChoose8Panel()

{

return PANEL\_8\_ADD + 1; //退出代码

}

**头文件 overdata.h**

#pragma once

//寄存器的参数

constexpr int REGISTER\_BUSY = 1;

constexpr int REGISTER\_FREE = 0;

constexpr int REGISTER\_NUM = 32;

constexpr int REGISTER\_INIT\_DATA = 0;

constexpr int REGISTER\_MAX\_NAME\_SIZE = 6;

//内存的参数

constexpr int MEMORY\_BUSY = 1;

constexpr int MEMORY\_FREE = 0;

constexpr int MEMORY\_NUM = 768; //内存区域长度

constexpr int MEMORY\_INIT\_DATA = 0;

constexpr int MEMORY\_INSTRUCTION\_SIZE = 256; //指令区长度

constexpr int MEMORY\_DATA\_SIZE = MEMORY\_NUM - MEMORY\_INSTRUCTION\_SIZE; //数据区域长度

//取指类的参数

constexpr int FETCH\_BUSY = 1;

constexpr int FETCH\_FREE = 0;

constexpr int FETCH\_PC\_INIT\_DATA = 0;

constexpr int PC\_ADD\_VALUE = 4;

//译码类的参数

constexpr int DECODING\_BUSY = 1;

constexpr int DECODING\_FREE = 0;

struct InstructionStruct

{

char rs;

char rt;

char rd;

char shamt;

short immediate;

short address;

}; //指令中译码得到的所有参数 结构体

constexpr char RS\_RT\_RD\_ERROR\_CODE = 127;

constexpr short IMMEDIATE\_ADDRESS\_ERROR\_CODE = -1;

constexpr int INSTRUCTION\_TYPE\_ERROR\_CODE = -1;

constexpr int SHAMT\_ERROR\_CODE = 127;

//指令中的FUNC标识状况

constexpr int R\_TYPE\_OP = 0x00;

constexpr int R\_TYPE\_ADDU\_FUNC = 0x21;

constexpr int R\_TYPE\_SUBU\_FUNC = 0x23;

constexpr int R\_TYPE\_AND\_FUNC = 0x24;

constexpr int R\_TYPE\_OR\_FUNC = 0x25;

constexpr int R\_TYPE\_XOR\_FUNC = 0x26;

constexpr int R\_TYPE\_NOR\_FUNC = 0x27;

constexpr int R\_TYPE\_SLL\_FUNC = 0x00;

constexpr int R\_TYPE\_SRL\_FUNC = 0x02;

constexpr int R\_TYPE\_JR\_FUNC = 0x08;

constexpr int I\_TYPE\_ADDI = 0x08;

constexpr int I\_TYPE\_ANDI = 0x0c;

constexpr int I\_TYPE\_ORI = 0x0d;

constexpr int I\_TYPE\_XORI = 0x0e;

constexpr int I\_TYPE\_LW = 0x23;

constexpr int I\_TYPE\_SW = 0x2b;

constexpr int I\_TYPE\_BEQ = 0x04;

constexpr int J\_TYPE\_J = 0x02;

//17条命令的对应序号

constexpr int R\_TYPE\_ADDU\_NO = 1;

constexpr int R\_TYPE\_SUBU\_NO = 2;

constexpr int R\_TYPE\_AND\_NO = 3;

constexpr int R\_TYPE\_OR\_NO = 4;

constexpr int R\_TYPE\_XOR\_NO = 5;

constexpr int R\_TYPE\_NOR\_NO = 6;

constexpr int R\_TYPE\_SLL\_NO = 7;

constexpr int R\_TYPE\_SRL\_NO = 8;

constexpr int R\_TYPE\_JR\_NO = 9;

constexpr int I\_TYPE\_ADDI\_NO = 10;

constexpr int I\_TYPE\_ANDI\_NO = 11;

constexpr int I\_TYPE\_ORI\_NO = 12;

constexpr int I\_TYPE\_XORI\_NO = 13;

constexpr int I\_TYPE\_LW\_NO = 14;

constexpr int I\_TYPE\_SW\_NO = 15;

constexpr int I\_TYPE\_BEQ\_NO = 16;

constexpr int J\_TYPE\_J\_NO = 17;

//执行类的参数

constexpr int EXECUTE\_BUSY = 1; //处理类忙

constexpr int EXECUTE\_FREE = 0; //处理类空闲

constexpr int EXECUTE\_CONTROL\_TYPE\_ERROR = 1000; //指令种类错误

constexpr int EXECUTE\_CONTROL\_RD\_ERROR = 2000; //RD数据错误

constexpr int EXECUTE\_CONTROL\_RT\_ERROR = 3000; //RT数据错误

constexpr int EXECUTE\_CONTROL\_RS\_ERROR = 4000; //RS数据错误

constexpr int EXECUTE\_CONTROL\_SHAMT\_ERROR = 5000; //SHAMT数据错误

constexpr int EXECUTE\_CONTROL\_IMMEDIATE\_ERROR = 6000; //IMMEDIATE数据错误

constexpr int EXECUTE\_CONTROL\_ADDRESS\_ERROR = 7000; //ADDRESS数据错误

constexpr int EXECUTE\_CONTROL\_CHECK\_SUCCESS = 7777; //指令执行成功

constexpr int EXECUTE\_CONTROL\_ACTION\_SUCCESS = 666; //指令执行成功

constexpr int REG\_ZERO\_CHANGED = 1; //$zero寄存器值改变

constexpr int REG\_ZERO\_NOCHANGED = 0; //$zero寄存器值未改变

constexpr char ALLREGISTERNAME[32][6] = { "$zero","$at","$v0","$v1","$a0","$a1","$a2","$a3","$t0","$t1","$t2","$t3","$t4","$t5",

"$t6","$t7","$s0","$s1","$s2","$s3","$s4","$s5","$s6","$s7","$t8","$t9","$k0","$k1",

"$gp","$sp","$fp","$ra" };

//控制类的参数

constexpr char HEXNAME[17] = "0123456789ABCDEF";

constexpr int INSTRUCTION\_TRANSFORM\_ERROR\_CODE = -1;

constexpr int INSTRUCTION\_CODE\_OK = 1; //指令语法正确

constexpr int INSTRUCTION\_CODE\_ERROR = -1; //指令语法错误

//constexpr int INSTRUCTION\_CODE\_USELESS = 0; //指令语法正确，但不适合本程序

constexpr int INSTRUCTION\_ACTION\_OK = 1; //执行正确

constexpr int INSTRUCTION\_ACTION\_ERROR = 0; //执行错误

//面板选择类的参数

constexpr int PANEL\_MAIN\_ADD = 100; //主面板选择的唯一标识

constexpr int PANEL\_1\_ADD = 1000; //主面板 1-指令控制 选择的唯一标识

constexpr int PANEL\_2\_ADD = 2000; //主面板 2-指令执行 选择的唯一标识

constexpr int PANEL\_3\_ADD = 3000; //主面板 3-内存控制 选择的唯一标识

constexpr int PANEL\_4\_ADD = 4000; //主面板 4-寄存器控制 选择的唯一标识

constexpr int PANEL\_5\_ADD = 5000; //主面板 5-流水线 选择的唯一标识

constexpr int PANEL\_6\_ADD = 6000; //主面板 6-使用说明 选择的唯一标识

constexpr int PANEL\_7\_ADD = 7000; //主面板 7-关于 选择的唯一标识

constexpr int PANEL\_8\_ADD = 8000; //主面板 8-退出 选择的唯一标识

constexpr int PANEL\_CHOOSE\_ERROR\_CODE = 9999; //错误代码

constexpr int PANEL\_RETURN\_TO\_MAIN\_PANEL = 999; //返回到主菜单

//流水线类的参数

constexpr int PIPELINE\_X\_NUM = MEMORY\_INSTRUCTION\_SIZE / 4;

constexpr int PIPELINE\_Y\_NUM = 1000;

constexpr char PIPELINE\_PERIOD\_NAME[5][11] = { "Fetch","Decode","Execute","ReadMemory","WriteBack" };

constexpr char PIPELINE\_PERIOD\_NAME\_SHORTEN[9] = "UFDEMWXB"; //U-等待；F-取值；D-译码；E-执行；M-访存；W-写回；X-无内容

constexpr int REGISTER\_TIME\_READ = 1; //读

constexpr int REGISTER\_TIME\_WRITE = 1; //写

constexpr int NOTEXISTTHISINSTRUCTIONONONETIME = -1; //该条指令在这个时间点不存在

**头文件 pipeline.h**

#pragma once

#include "overalldata.h"

#include <fstream>

using namespace std;

class CPipeline

{

private:

char pipeline\_coordinates[PIPELINE\_X\_NUM][PIPELINE\_Y\_NUM]; //流水线的二维XY坐标系

char pipeline\_coordinates\_type\_2[PIPELINE\_X\_NUM][PIPELINE\_Y\_NUM]; //流水线的二维XY坐标系，用于单步执行

int register\_time\_state[PIPELINE\_Y\_NUM]; //每个时刻的寄存器状态

int function\_state[PIPELINE\_Y\_NUM]; //每个时刻的五个部件的状态

int instruction\_start[PIPELINE\_X\_NUM]; //每一条指令的开始位置

bool if\_write[PIPELINE\_X\_NUM]; //二维XY坐标系中每一行是否被用

public:

CPipeline(); //默认构造函数

~CPipeline(); //析构函数

void ResetAll(); //初始化全部数据

void ResetPipelineCoordinates(); //初始化XY坐标系

void ResetRegsiterTimeState(); //初始化每时刻的寄存器状态数组

void ResetFunctionState(); //初始化每时刻五个部件的状态数组

void ResetInstructionStartAddress(); //初始化每条指令的开始位置

void ResetIfWrite(); //初始化每条指令是否已被解析

void WriteBan(int,int,InstructionStruct); //写入Ban指令

int WhichNeedRead(int, InstructionStruct); //判断指令需要读的寄存器，返回一个int类型的32位数据，对应表示32个寄存器

int WhichNeedWrite(int, InstructionStruct); //判断指令需要写的寄存器，返回一个int类型的32位数据，对应表示32个寄存器

void WriteBitToInt(int &, int, int); //对传入的数据，在特定位置写入特定的 1 / 0

int CheckBit(int content, int address); //返回某一位的值

int CheckInstructionNum(); //返回当前二维XY坐标系中已经用的流水线条数

bool CheckPreviousInstructionIfDown(int, int); //检查前面的指令是否已经执行完毕

bool CheckPreviousOneInstructionIfDown(int, int); //检查某一条指令是否已经执行完毕

int CreatePipeline(int, int, InstructionStruct); //对单条指令进行解析，生成流水线

int CreatePipelineStep(int step\_num); //单步模拟执行流水线

bool CheckAnyInstruction(int step\_num); //判断当前时间点是否还有指令需要执行

void WriteCoordinatesToFile(); //将二维XY坐标系中的数据写入到文件中

void WriteCoordinatesToFileType2(int); //将二维XY坐标系中的数据写入到单步执行记录文件中

int IfExistInstructionOnOneTime(int, char); //判断是否在某一时间点存在某一动作，如果存在则返回该指令的编号，如果不存在，则返回特定值

};

**cpp文件 pipeline.cpp**

#include "pipeline.h"

CPipeline::CPipeline()

{

ResetAll();

}

CPipeline::~CPipeline()

{

}

void CPipeline::ResetAll()

{

ResetPipelineCoordinates();

ResetInstructionStartAddress();

ResetRegsiterTimeState();

ResetFunctionState();

ResetIfWrite();

}

void CPipeline::ResetPipelineCoordinates()

{

for (int i = 0; i < PIPELINE\_X\_NUM; i++)

{

for (int j = 0; j < PIPELINE\_Y\_NUM; j++)

{

pipeline\_coordinates[i][j] = PIPELINE\_PERIOD\_NAME\_SHORTEN[6];

pipeline\_coordinates\_type\_2[i][j] = PIPELINE\_PERIOD\_NAME\_SHORTEN[6];

}

}

}

void CPipeline::ResetRegsiterTimeState()

{

for (int i = 0; i < PIPELINE\_Y\_NUM; i++)

register\_time\_state[i] = 0;

}

void CPipeline::ResetFunctionState()

{

for (int i = 0; i < PIPELINE\_Y\_NUM; i++)

function\_state[i] = 0;

}

void CPipeline::ResetInstructionStartAddress()

{

for (int i = 0; i < PIPELINE\_X\_NUM; i++)

instruction\_start[i] = i;

}

void CPipeline::ResetIfWrite()

{

for (int i = 0; i < PIPELINE\_X\_NUM; i++)

if\_write[i] = false;

}

void CPipeline::WriteBan(int instruction\_no,int instruction\_type,InstructionStruct data)

{

int start = instruction\_start[instruction\_no]; //指令在坐标系中的起始位置

int k = 0; //位置计数器

//判断F是否被占用

while (CheckBit(function\_state[start + k], 1) != 0) //F部件被占用

{

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[6]; //因为是F，所以直接写入'X'

k++;

}

//F部件可以使用

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[1]; //写入'F'

WriteBitToInt(function\_state[start + k], 1, 1); //F部件占用

k++;

//判断D部件是否被占用

while (CheckBit(function\_state[start + k], 2) != 0) //D部件被占用

{

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[0]; //等待

WriteBitToInt(function\_state[start + k], 1, 1); //F部件被占用

k++;

}

//D部件可以使用

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[2]; //写入'D'

WriteBitToInt(function\_state[start + k], 2, 1); //D部件被占用

k++;

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[7];

//本条指令所对应的行已被占用

if\_write[instruction\_no] = true;

}

int CPipeline::WhichNeedRead(int instruction\_type,InstructionStruct data)

{

int content = 0;

switch (instruction\_type)

{

case R\_TYPE\_ADDU\_NO:

case R\_TYPE\_SUBU\_NO:

case R\_TYPE\_AND\_NO:

case R\_TYPE\_OR\_NO:

case R\_TYPE\_XOR\_NO:

case R\_TYPE\_NOR\_NO:

WriteBitToInt(content, data.rs, REGISTER\_TIME\_READ);

WriteBitToInt(content, data.rt, REGISTER\_TIME\_READ);

break;

case R\_TYPE\_SLL\_NO:

case R\_TYPE\_SRL\_NO:

WriteBitToInt(content, data.rt, REGISTER\_TIME\_READ);

break;

case R\_TYPE\_JR\_NO:

WriteBitToInt(content, data.rs, REGISTER\_TIME\_READ);

break;

case I\_TYPE\_ADDI\_NO:

case I\_TYPE\_ANDI\_NO:

case I\_TYPE\_ORI\_NO:

case I\_TYPE\_XORI\_NO:

case I\_TYPE\_LW\_NO:

WriteBitToInt(content, data.rs, REGISTER\_TIME\_READ);

break;

case I\_TYPE\_SW\_NO:

case I\_TYPE\_BEQ\_NO:

WriteBitToInt(content, data.rs, REGISTER\_TIME\_READ);

WriteBitToInt(content, data.rt, REGISTER\_TIME\_READ);

break;

case J\_TYPE\_J\_NO:

break;

default:

break;

}

return content;

}

int CPipeline::WhichNeedWrite(int instruction\_type, InstructionStruct data)

{

int content = 0;

switch (instruction\_type)

{

case R\_TYPE\_ADDU\_NO:

case R\_TYPE\_SUBU\_NO:

case R\_TYPE\_AND\_NO:

case R\_TYPE\_OR\_NO:

case R\_TYPE\_XOR\_NO:

case R\_TYPE\_NOR\_NO:

WriteBitToInt(content, data.rd, REGISTER\_TIME\_WRITE);

break;

case R\_TYPE\_SLL\_NO:

case R\_TYPE\_SRL\_NO:

WriteBitToInt(content, data.rd, REGISTER\_TIME\_WRITE);

break;

case R\_TYPE\_JR\_NO:

break;

case I\_TYPE\_ADDI\_NO:

case I\_TYPE\_ANDI\_NO:

case I\_TYPE\_ORI\_NO:

case I\_TYPE\_XORI\_NO:

case I\_TYPE\_LW\_NO:

WriteBitToInt(content, data.rt, REGISTER\_TIME\_WRITE);

break;

case I\_TYPE\_SW\_NO:

case I\_TYPE\_BEQ\_NO:

break;

case J\_TYPE\_J\_NO:

break;

default:

break;

}

return content;

}

void CPipeline::WriteBitToInt(int &content, int address, int mode)

{

if (mode == 1) //写入1

{

content |= (1 << address);

}

else if (mode == 0) //写入0

{

content &= ~(1 << address);

}

}

int CPipeline::CheckBit(int content, int address)

{

if ((content & (1 << address)) != 0) return 1;

else return 0;

}

int CPipeline::CheckInstructionNum()

{

int num = 0;

for (int i = 0; i < PIPELINE\_X\_NUM; i++)

{

int sum = 0;

for (int j = 0; j < PIPELINE\_Y\_NUM; j++)

sum += pipeline\_coordinates[i][j];

if (sum != 'X'\*PIPELINE\_Y\_NUM)

num++;

}

return num;

}

bool CPipeline::CheckPreviousInstructionIfDown(int instruction\_no, int time\_address)

{

for(int i = instruction\_no - 1; i >= 1; i--)

{

if (CheckPreviousOneInstructionIfDown(i, time\_address) == false)

return false;

}

return true;

}

bool CPipeline::CheckPreviousOneInstructionIfDown(int instruction\_no,int time\_address)

{

for (int i = instruction\_start[instruction\_no]; i < time\_address; i++)

{

if (pipeline\_coordinates[instruction\_no][i] == PIPELINE\_PERIOD\_NAME\_SHORTEN[7])

return true;

if (pipeline\_coordinates[instruction\_no][i] == PIPELINE\_PERIOD\_NAME\_SHORTEN[5])

return true;

}

return false;

}

int CPipeline::CreatePipeline(int instruction\_no, int instruction\_type, InstructionStruct data)

{

int start = instruction\_start[instruction\_no]; //指令在坐标系中的起始位置

int k = 0; //位置计数器

//判断F是否被占用

while (CheckBit(function\_state[start + k], 1) != 0) //F部件被占用

{

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[6]; //因为是F，所以直接写入'X'

k++;

}

//F部件可以使用

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[1]; //写入'F'

WriteBitToInt(function\_state[start + k], 1, 1); //F部件占用

k++;

//判断D部件是否被占用

while (CheckBit(function\_state[start + k], 2) != 0) //D部件被占用

{

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[0]; //等待

WriteBitToInt(function\_state[start + k], 1, 1); //F部件被占用

k++;

}

//D部件可以使用

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[2]; //写入'D'

WriteBitToInt(function\_state[start + k], 2, 1); //D部件被占用

//对读后写的延时处理

int which\_need\_read = WhichNeedRead(instruction\_type, data); //需要读的寄存器

int which\_need\_write = WhichNeedWrite(instruction\_type, data); //需要写的寄存器

register\_time\_state[start + k] |= which\_need\_write;

k++;

while (1)

{

if ((which\_need\_read & register\_time\_state[start + k]) == 0)

break;

//延时等待

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[0];

WriteBitToInt(function\_state[start + k], 2, 1); //D部件被占用

k++;

}

//判断E部件是否被占用

while (CheckBit(function\_state[start + k], 3) != 0) //E部件被占用

{

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[0]; //等待

WriteBitToInt(function\_state[start + k], 2, 1); //D部件被占用

register\_time\_state[start + k] |= which\_need\_write;

k++;

}

//E部件可以使用

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[3]; //写入'E'

WriteBitToInt(function\_state[start + k], 3, 1); //E部件被占用

register\_time\_state[start + k] |= which\_need\_write;

k++;

//判断M部件是否被占用

while (CheckBit(function\_state[start + k], 4) != 0) //M部件被占用

{

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[0]; //等待

WriteBitToInt(function\_state[start + k], 3, 1); //E部件被占用

register\_time\_state[start + k] |= which\_need\_write;

k++;

}

//M部件可以使用

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[4]; //写入'M'

WriteBitToInt(function\_state[start + k], 4, 1); //M部件被占用

register\_time\_state[start + k] |= which\_need\_write;

k++;

//判断指令是否有超前结束的可能

if (instruction\_no >= 1)

{

while (CheckPreviousInstructionIfDown(instruction\_no, start + k) != true)

{

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[0];

register\_time\_state[start + k] |= which\_need\_write;

k++;

}

}

//判断W部件是否被占用

while (CheckBit(function\_state[start + k], 5) != 0) //W部件被占用

{

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[0]; //等待

WriteBitToInt(function\_state[start + k], 4, 1); //M部件被占用

register\_time\_state[start + k] |= which\_need\_write;

k++;

}

//W部件可以使用

pipeline\_coordinates[instruction\_no][start + k] = PIPELINE\_PERIOD\_NAME\_SHORTEN[5]; //写入'W'

WriteBitToInt(function\_state[start + k], 5, 1); //W部件被占用

register\_time\_state[start + k] |= which\_need\_write;

k++;

//对本条指令所占用的寄存器进行释放

register\_time\_state[start + k + 1] &= ~which\_need\_write;

//本条指令所对应的行已被占用

if\_write[instruction\_no] = true;

return 0;

}

int CPipeline::CreatePipelineStep(int step\_num)

{

for (int i = 0; i <= CheckInstructionNum() - 1; i++)

{

for (int j = 0; j <=step\_num; j++)

{

pipeline\_coordinates\_type\_2[i][j] = pipeline\_coordinates[i][j];

}

}

return 0;

}

bool CPipeline::CheckAnyInstruction(int step\_num)

{

int sum = 0;

for (int i = 0; i < PIPELINE\_X\_NUM; i++)

{

sum += pipeline\_coordinates[i][step\_num];

}

if (sum == PIPELINE\_PERIOD\_NAME\_SHORTEN[6] \* PIPELINE\_X\_NUM)

return false;

return true;

}

void CPipeline::WriteCoordinatesToFile()

{

ofstream outfile, graphics;

outfile.open("pipeline.txt", ios::out);

graphics.open("graphics.txt", ios::out);

for (int i = CheckInstructionNum() - 1; i >= 0; i--)

{

outfile << i << "\t";

for (int j = 0;; j++)

{

if (pipeline\_coordinates[i][j - 1] == PIPELINE\_PERIOD\_NAME\_SHORTEN[7])

break;

if (pipeline\_coordinates[i][j - 1] == PIPELINE\_PERIOD\_NAME\_SHORTEN[5])

break;

if (pipeline\_coordinates[i][j] == PIPELINE\_PERIOD\_NAME\_SHORTEN[0]) //U等待

{

outfile << "U ";

graphics << "U";

}

else if (pipeline\_coordinates[i][j] == PIPELINE\_PERIOD\_NAME\_SHORTEN[6]) //X空

{

outfile << " ";

graphics << "X";

}

else if (pipeline\_coordinates[i][j] == PIPELINE\_PERIOD\_NAME\_SHORTEN[7]) //B禁止

{

outfile << "B ";

graphics << "B";

}

else

{

outfile << pipeline\_coordinates[i][j] << " ";

graphics << pipeline\_coordinates[i][j];

}

}

outfile << "\n";

graphics << "\n";

}

outfile.close();

graphics.close();

}

void CPipeline::WriteCoordinatesToFileType2(int step\_num)

{

ofstream outfile, graphics;

outfile.open("pipeline\_step.txt", ios::out);

graphics.open("graphics.txt", ios::out);

for (int i = CheckInstructionNum() - 1; i >= 0; i--)

{

outfile << i << "\t";

for (int j = 0; j <= step\_num; j++)

{

if (pipeline\_coordinates\_type\_2[i][j - 1] == PIPELINE\_PERIOD\_NAME\_SHORTEN[7])

break;

if (pipeline\_coordinates\_type\_2[i][j - 1] == PIPELINE\_PERIOD\_NAME\_SHORTEN[5])

break;

if (pipeline\_coordinates\_type\_2[i][j] == PIPELINE\_PERIOD\_NAME\_SHORTEN[0]) //U等待

{

outfile << "U ";

graphics << "U";

}

else if (pipeline\_coordinates\_type\_2[i][j] == PIPELINE\_PERIOD\_NAME\_SHORTEN[6]) //X空

{

outfile << " ";

graphics << "X";

}

else if (pipeline\_coordinates\_type\_2[i][j] == PIPELINE\_PERIOD\_NAME\_SHORTEN[7]) //B禁止

{

outfile << "B ";

graphics << "B";

}

else

{

outfile << pipeline\_coordinates\_type\_2[i][j] << " ";

graphics << pipeline\_coordinates\_type\_2[i][j];

}

}

outfile << "\n";

graphics << "\n";

}

outfile.close();

graphics.close();

}

int CPipeline::IfExistInstructionOnOneTime(int time, char mode)

{

for (int i = 0; i < PIPELINE\_X\_NUM; i++)

{

if (pipeline\_coordinates\_type\_2[i][time] == mode)

return i;

}

return NOTEXISTTHISINSTRUCTIONONONETIME;

}

**头文件 control.h**

#pragma once

#include "overalldata.h"

#include "decoding.h"

#include "execute.h"

#include "fetch.h"

#include "log.h"

#include "memory.h"

#include "panel.h"

#include "pipeline.h"

#include "register.h"

#include <iostream>

#include <iomanip>

#include <string>

#include <fstream>

#include <direct.h>

using namespace std;

class CControl

{

private:

CDecoding decoding;

CExecute execute;

CFetch fetch;

CLog log;

CMemory memory;

CPanel panel;

CPipeline pipeline;

CRegister reg;

int current\_instruction\_num; //当前共有的指令条数

public:

CControl(); //默认构造函数

~CControl(); //析构函数

/\*程序入口相关的函数\*/

void AllBegin(); //程序入口处，直接调用

int PanelControl(); //面板控制

void PressAnyKeyBackToMainMenu(); //按任意键返回主菜单

/\*详细功能\*/

/\*菜单1-指令控制\*/

int TransformFromCharToInstruction(char[], int&); //将输入的字符串转化为指令

int IfInstructionOK(char []); //检查输入的指令是否违法

void Menu11WriteInstruction(); //输入指令

void Menu12CheckInstructon(); //查看指令

void Menu13EditInstruction(); //修改指令

void Menu14ClearInstruction(); //清空指令

/\*菜单2-指令执行\*/

void ResetCurrentInstructionNum(); //重置当前的所有的指令条数

void RefershCurrentInstructionNum(); //刷新当前的所有的指令条数

int RegZeroCheck(); //检测$zero寄存器的状态，并重置为0

int ExecuteOneInstruction(int); //执行一条指令

void Menu21ExecuteAllInstruction(); //执行全部指令

void Menu22ExecuteInstructionByStep(); //单步执行指令

void Menu23CheckExecuteRecord(); //检查指令执行记录

void Menu24ClearExecuteRecord(); //清除指令执行记录

/\*菜单3-内存控制\*/

void Menu31InitMemoryWith0(); //初始化内存 全0

void Menu32InitMemoryWithValue(); //初始化内存 指定值

void Menu33EditMemory(); //修改内存

void Menu34CheckMemory(); //查看内存

/\*菜单4-寄存器控制\*/

void Menu41InitRegisterWith0(); //初始化寄存器 全0

void Menu42InitRegisterWithValue(); //初始化寄存器 指定值

void Menu43EditRegister(); //修改寄存器

void Menu44CheckRegister(); //查看寄存器

/\*菜单5-流水线\*/

void Menu51CheckPipeline(); //查看流水线图

void Menu52StepRunPipeline(); //单步模拟执行流水线

void ShowPC(); //查看当前PC指针的值

void Menu53ClearPipeline(); //清空流水线

/\*菜单6-使用说明\*/

//--此菜单无需函数实现

/\*菜单7-关于\*/

//--此菜单无需函数实现

/\*菜单8-退出\*/

void PanelMenu8Exit();

};

**cpp文件 control.cpp**

#include "control.h"

CControl::CControl()

{

current\_instruction\_num = 0; //当前共有的指令条数

}

CControl::~CControl()

{

}

void CControl::AllBegin()

{

//初始化菜单

PanelControl();

}

int CControl::PanelControl()

{

int choose\_num;

choose\_num = panel.PanelChoose();

while (1)

{

switch (choose\_num)

{

case PANEL\_CHOOSE\_ERROR\_CODE:

cout << endl << "程序运行错误！" << endl;

break;

case PANEL\_1\_ADD+1:

Menu11WriteInstruction();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_1\_ADD+2:

Menu12CheckInstructon();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_1\_ADD+3:

Menu13EditInstruction();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_1\_ADD+4:

Menu14ClearInstruction();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_2\_ADD+1:

Menu21ExecuteAllInstruction();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_2\_ADD+2:

Menu22ExecuteInstructionByStep();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_2\_ADD+3:

Menu23CheckExecuteRecord();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_2\_ADD+4:

Menu24ClearExecuteRecord();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_3\_ADD+1:

Menu31InitMemoryWith0();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_3\_ADD+2:

Menu32InitMemoryWithValue();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_3\_ADD+3:

Menu33EditMemory();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_3\_ADD+4:

Menu34CheckMemory();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_4\_ADD+1:

Menu41InitRegisterWith0();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_4\_ADD+2:

Menu42InitRegisterWithValue();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_4\_ADD+3:

Menu43EditRegister();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_4\_ADD+4:

Menu44CheckRegister();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_5\_ADD+1:

Menu51CheckPipeline();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_5\_ADD+2:

Menu52StepRunPipeline();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_5\_ADD+3:

Menu53ClearPipeline();

PressAnyKeyBackToMainMenu();

break;

case PANEL\_6\_ADD+1: //使用说明

case PANEL\_7\_ADD+1: //关于

case PANEL\_RETURN\_TO\_MAIN\_PANEL:

choose\_num = panel.PanelChoose();

break;

case PANEL\_8\_ADD+1: //退出

PanelMenu8Exit();

break;

default:

cout << endl << "程序运行错误！请检查是否输入了非法字符！" << endl << endl;

return 0;

break;

}

}

return 0;

}

void CControl::PressAnyKeyBackToMainMenu()

{

cout << endl << "输入任意值以返回主菜单！";

int x;

cin >> x;

PanelControl();

}

void CControl::Menu11WriteInstruction()

{

cout << "请输入要添加的指令，以\"##\"结束：" << endl;

char content[40] = { '\0' };

while (1)

{

cin >> content;

if (strcmp(content, "##") == 0) break;

if (IfInstructionOK(content) != INSTRUCTION\_CODE\_OK)

{

cout << "输入的指令错误，请检查后重新输入！" << endl;

return;

}

int instruction\_address = memory.GetCurrentInstructionAddress();

int instruction;

TransformFromCharToInstruction(content, instruction);

for (int j = 0; j < 4; j++)

{

memory.WriteToMemory(instruction\_address + j, (char)(instruction >> (8 \* (3 - j))));

}

memory.AddInstructionAddress();

}

cout << "添加完毕！";

}

void CControl::Menu12CheckInstructon()

{

int s = 0; //start

unsigned int ins = 0;

char data = 0;

//指令相关的数据

InstructionStruct ins\_data;

int type;

string str;

cout << "内存中的所有指令如下：" << endl;

for (int i = 0; i < MEMORY\_INSTRUCTION\_SIZE; i += 4, ins = 0)

{

s = (i / 4) \* 4; //计算得到每条指令的起始位置

for (int j = 0; j < 4; j++)

{

memory.ReadFromMemory(s + j, data);

unsigned int data\_transfer = (unsigned int)data;

data\_transfer &= 0x00ff; //去除前面的符号位扩展

ins |= (data\_transfer << (8 \* (3 - j)));

}

if (ins != 0)

{

if (s < 0x10)

cout << "0" << setiosflags(ios::uppercase) << hex << s << "H：";

else

cout << setiosflags(ios::uppercase) << hex << s << "H：";

decoding.InstructionsDecoding(ins, ins\_data, type);

str = log.GetInstructionString(type, ins\_data);

cout << str << endl;

}

}

}

void CControl::Menu13EditInstruction()

{

cout << "请输入要修改的指令所在地址（任意一个包含的地址即可）： ";

int add;

cin >> add;

if (add >= MEMORY\_INSTRUCTION\_SIZE || add < 0)

{

cout << "输入的地址非法！" << endl;

return;

}

int add\_start = (add / 4) \* 4;

char str\_content[40];

int ins;

if (add\_start < 0x10)

cout << "请输入要在地址" << setiosflags(ios::uppercase) << hex << "0" << add\_start << "H修改的指令内容： ";

else

cout << "请输入要在地址" << setiosflags(ios::uppercase) << hex << add\_start << "H修改的指令内容： ";

cin >> str\_content;

int ifok = IfInstructionOK(str\_content);

if (ifok==INSTRUCTION\_CODE\_OK)

{

TransformFromCharToInstruction(str\_content, ins);

for (int j = 0; j < 4; j++)

memory.WriteToMemory(add\_start + j, (char)(ins >> (8 \* (3 - j))));

cout << "修改完成！" << endl;

}

else

{

cout << "输入的指令错误，请检查后重新输入！" << endl;

}

}

void CControl::Menu14ClearInstruction()

{

for (int i = 0; i < MEMORY\_INSTRUCTION\_SIZE; i++)

{

memory.WriteToMemory(i, 0);

}

fetch.ResetPC(); //清空指令，PC指针复位

ResetCurrentInstructionNum(); //现有的指令条数清零

memory.ResetCurrentInstructionAddress(); //重置指令的添加位置

cout << "从 0" << setiosflags(ios::uppercase) << hex << 0 << "H 到 " << setiosflags(ios::uppercase) << hex << MEMORY\_INSTRUCTION\_SIZE << "H 的内存指令区域已经清空！" << endl;

}

void CControl::ResetCurrentInstructionNum()

{

current\_instruction\_num = 0;

}

void CControl::RefershCurrentInstructionNum()

{

int ins\_num = 0;

for (int i = 0; i < MEMORY\_INSTRUCTION\_SIZE; i += 4)

{

char n1, n2, n3, n4;

memory.ReadFromMemory(i, n1);

memory.ReadFromMemory(i + 1, n2);

memory.ReadFromMemory(i + 2, n3);

memory.ReadFromMemory(i + 3, n4);

if (n1 != 0 || n2 != 0 || n3 != 0 || n4 != 0)

ins\_num++;

}

current\_instruction\_num = ins\_num;

}

int CControl::RegZeroCheck()

{

int reg0\_value;

reg.ReadFromOneRegister(0, reg0\_value);

if (reg0\_value != 0)

{

cout << "检测到$zero寄存器值改变！已经阻止该指令的执行！" << endl;

reg.WriteToOneRegister(0, 0); //将$zero寄存器的值改写为0

return REG\_ZERO\_CHANGED;

}

return REG\_ZERO\_NOCHANGED;

}

int CControl::ExecuteOneInstruction(int instruction\_no)

{

//取指

int instruction = 0; //指令的编码

fetch.FetchInstructionFromMemory(fetch.GetPC(), memory.mem, instruction);

//译码

int instruction\_type; //指令类型

InstructionStruct instruction\_data; //指令的所有提取的数据

decoding.InstructionsDecoding(instruction, instruction\_data, instruction\_type);

if (instruction == 0) return INSTRUCTION\_ACTION\_ERROR; //检测指令是否有效

//执行

int execute\_code = execute.ExecuteControl(instruction\_type, instruction\_data, memory.mem, reg.reg, fetch.PC);

if (execute\_code != EXECUTE\_CONTROL\_ACTION\_SUCCESS)

{

if (execute\_code == EXECUTE\_CONTROL\_TYPE\_ERROR)

{

cout << "指令格式错误！该条指令拒绝执行！" << endl;

}

if (execute\_code == EXECUTE\_CONTROL\_RD\_ERROR)

{

cout << "RD范围错误！该条指令拒绝执行！" << endl;

}

if (execute\_code == EXECUTE\_CONTROL\_RT\_ERROR)

{

cout << "RT范围错误！该条指令拒绝执行！" << endl;

}

if (execute\_code == EXECUTE\_CONTROL\_RS\_ERROR)

{

cout << "RS范围错误！该条指令拒绝执行！" << endl;

}

if (execute\_code == EXECUTE\_CONTROL\_SHAMT\_ERROR)

{

cout << "SHAMT范围错误！该条指令拒绝执行！" << endl;

}

if (execute\_code == EXECUTE\_CONTROL\_IMMEDIATE\_ERROR)

{

cout << "IMMEDIATE范围错误！该条指令拒绝执行！" << endl;

}

if (execute\_code == EXECUTE\_CONTROL\_ADDRESS\_ERROR)

{

cout << "ADDRESS范围错误！该条指令拒绝执行！" << endl;

}

fetch.PCSelfAdd(); //PC指针自增

return INSTRUCTION\_ACTION\_ERROR;

}

else

{

//显示指令执行状况

string action\_instruction\_message = log.GetInstructionLogDetail(instruction\_type, instruction\_data);

cout << action\_instruction\_message << "指令执行成功！" << endl << endl;

//写入文件

log.WriteLog(instruction\_type, instruction\_data);

}

//访存 //写回 //包含在执行过程中，无需再写代码

fetch.PCSelfAdd(); //PC指针自增

//检测$zero寄存器

RegZeroCheck();

return INSTRUCTION\_ACTION\_OK;

}

void CControl::Menu21ExecuteAllInstruction()

{

RefershCurrentInstructionNum(); //刷新当前内存中共有的指令条数

int instruction\_num = current\_instruction\_num; //当前共有的指令条数

log.LogBegin();

for (int i = 0; i < instruction\_num; i++)

ExecuteOneInstruction(i);

log.LogEnd();

if (instruction\_num > 1)

{

char buffer[200];

\_getcwd(buffer, 200);

cout << endl << "指令执行记录已经保存在： " << endl << buffer << "\\record.txt" << endl << "你可以手动打开或者通过本程序查看！" << endl;

}

}

void CControl::Menu22ExecuteInstructionByStep()

{

RefershCurrentInstructionNum(); //刷新当前内存中共有的指令条数

int instruction\_num = current\_instruction\_num; //当前共有的指令条数

log.LogBegin();

int choice = 2;

int k = 0; //k为当前指令条数计数

cout << "开始单步执行：" << endl;

while (1)

{

if (ExecuteOneInstruction(k++) == INSTRUCTION\_ACTION\_ERROR) //执行一条指令

{

log.LogEnd();

break;

}

cout << endl << "单步执行完成，如果继续请按1，如果退出请按0： ";

cin >> choice;

while (choice!=0&&choice!=1)

{

cout << "输入错误，请重新输入： ";

cin >> choice;

}

if (choice == 0)

{

log.LogEnd();

return;

}

else if (choice == 1) continue;

}

}

void CControl::Menu23CheckExecuteRecord()

{

cout << endl << "指令的执行记录如下：" << endl;

ifstream execute\_record;

execute\_record.open("record.txt", ios::in);

if (!execute\_record.is\_open())

{

cout << "文件打开失败！请确保文件存在！" << endl;

return;

}

char c;

execute\_record >> noskipws;

while (!execute\_record.eof())

{

execute\_record >> c;

cout << c;

}

execute\_record.close();

}

void CControl::Menu24ClearExecuteRecord()

{

log.ClearLog();

cout << endl << "record.txt文件已经清空！" << endl;

}

void CControl::Menu31InitMemoryWith0()

{

memory.InitAllMemoryWith0();

cout << "已经将从地址 ";

cout << setiosflags(ios::uppercase) << hex << MEMORY\_INSTRUCTION\_SIZE;

cout << "H 到地址 ";

cout << setiosflags(ios::uppercase) << hex << MEMORY\_NUM - 1;

cout << "H 全部初始化为 0 ！" << endl;

}

void CControl::Menu32InitMemoryWithValue()

{

int init\_num;

cout << "请输入指定值以初始化全部内存： ";

cin >> init\_num;

if (init\_num < 0 || init\_num>127)

{

cout << "不能够用此值进行初始化！" << endl;

cout << "能够使用的值范围为：" << setiosflags(ios::uppercase) << hex << 0 << " - " << setiosflags(ios::uppercase) << hex << 127 << endl;

return;

}

memory.InitAllMemoryWithValue(init\_num);

cout << "已经将从地址 ";

cout << setiosflags(ios::uppercase) << hex << MEMORY\_INSTRUCTION\_SIZE;

cout << "H 到地址 ";

cout << setiosflags(ios::uppercase) << hex << MEMORY\_NUM - 1;

if(init\_num<0x10)

cout << "H 全部初始化为 0" << init\_num << "H ！" << endl;

else

cout << "H 全部初始化为 " << init\_num << "H ！" << endl;

}

void CControl::Menu33EditMemory()

{

int edit\_address, edit\_data;

cout << "请输入要修改的内存地址及要修改的值： ";

cin >> edit\_address >> edit\_data;

if (edit\_address < MEMORY\_INSTRUCTION\_SIZE - 1)

{

cout << "不能修改指令区域中的值！" << endl;

cout << "能够修改的地址范围为：" << setiosflags(ios::uppercase) << hex << MEMORY\_INSTRUCTION\_SIZE << "H - " << setiosflags(ios::uppercase) << hex << MEMORY\_NUM << "H" << endl;

return;

}

if (edit\_address > MEMORY\_NUM - 1)

{

cout << "不存在这样的区域！" << endl;

cout << "能够修改的地址范围为：" << setiosflags(ios::uppercase) << hex << MEMORY\_INSTRUCTION\_SIZE << "H - " << setiosflags(ios::uppercase) << hex << MEMORY\_NUM << "H" << endl;

return;

}

if (edit\_data < 0 || edit\_data>127)

{

cout << "不能够用此值进行修改！" << endl;

cout << "能够使用的值范围为：" << setiosflags(ios::uppercase) << hex << 0 << "H - " << setiosflags(ios::uppercase) << hex << 127 << "H" << endl;

return;

}

memory.WriteToMemory(edit\_address, edit\_data);

cout << "已经将地址为 " << setiosflags(ios::uppercase) << hex << edit\_address << "H 地址的数据修改为 " << setiosflags(ios::uppercase) << hex << edit\_data << "H ！" << endl;

}

void CControl::Menu34CheckMemory()

{

int add = 0 - 16, cal = 0;

char data;

cout << endl << "指令区域：";

for (int i = 0; i < MEMORY\_INSTRUCTION\_SIZE; i++)

{

if (cal % 16 == 0)

{

add += 16;

if (add < 0x10)

cout << endl << "0";

else

cout << endl;

cout << setiosflags(ios::uppercase) << hex << add << "H\t";

}

memory.ReadFromMemory(i, data);

cout << HEXNAME[(data >> 4) & 0x0f] << HEXNAME[(data >> 0) & 0x0f] << "H ";

cal++;

}

add = MEMORY\_INSTRUCTION\_SIZE - 16, cal = 0;

cout << endl << endl << "数据区域：";

for (int i = MEMORY\_INSTRUCTION\_SIZE; i < MEMORY\_NUM; i++)

{

if (cal % 16 == 0)

{

add += 16;

if (add < 0x10)

cout << "0";

cout << endl << setiosflags(ios::uppercase) << hex << add << "H\t";

}

memory.ReadFromMemory(i, data);

cout << HEXNAME[(data >> 4) & 0x0f] << HEXNAME[(data >> 0) & 0x0f] << "H ";

cal++;

}

cout << endl;

}

void CControl::Menu41InitRegisterWith0()

{

reg.InitAllRegisterWith0();

cout << "已经初始化全部寄存器为0（$zero除外）！" << endl;

}

void CControl::Menu42InitRegisterWithValue()

{

cout << resetiosflags(ios::uppercase) << dec;

int init\_num;

cout << "请输入要初始化的值： ";

cin >> init\_num;

reg.InitAllRegisterWithValue(init\_num);

cout << "已经初始化全部寄存器为" << init\_num << "（$zero除外）！" << endl;

}

void CControl::Menu43EditRegister()

{

cout << resetiosflags(ios::uppercase) << dec;

int edit\_no, edit\_data;

cout << "请输入要修改的寄存器序号、值： ";

cin >> edit\_no >> edit\_data;

if (edit\_no == 0)

{

cout << "$zero 恒为0，无法修改值！" << endl;

return;

}

else if (edit\_no < 0 || (edit\_no > REGISTER\_NUM - 1))

{

cout << "无法找到序号为 " << edit\_no << " 的寄存器！" << endl;

return;

}

reg.WriteToOneRegister(edit\_no, edit\_data);

cout << "寄存器 " << log.GetRegisterName(edit\_no) << " 的值已经修改为 " << edit\_data << " ！" << endl;

}

void CControl::Menu44CheckRegister()

{

cout << resetiosflags(ios::uppercase) << dec;

cout << "所有寄存器的值为：" << endl;

int reg\_data, cal = 0;

for (int i = 0; i < REGISTER\_NUM; i++)

{

reg.ReadFromOneRegister(i, reg\_data);

cout << log.GetRegisterName(i) << "：" << reg\_data << "\t\t";

cal++;

if (cal % 4 == 0)

cout << endl;

}

}

void CControl::Menu51CheckPipeline()

{

fetch.ResetPC();//使PC指针归位

pipeline.ResetAll();

RefershCurrentInstructionNum(); //刷新当前内存中共有的指令条数

int instruction\_num = current\_instruction\_num; //当前共有的指令条数

for (int i = 0; i < instruction\_num; i++)

{

int instruction = 0; //指令的编码

fetch.FetchInstructionFromMemory(fetch.GetPC(), memory.mem, instruction);

int instruction\_type; //指令类型

InstructionStruct instruction\_data; //指令的所有提取的数据

decoding.InstructionsDecoding(instruction, instruction\_data, instruction\_type);

if (decoding.InstructionCheckZero(instruction\_type, instruction\_data) == true) //查看指令是否写$zero0

{

pipeline.WriteBan(i, instruction\_type, instruction\_data);

}

else

{

pipeline.CreatePipeline(i, instruction\_type, instruction\_data);

}

fetch.PCSelfAdd(); //PC指针自增

}

pipeline.WriteCoordinatesToFile();

cout << endl << "流水线图如下：" << endl << endl;

ifstream pipeline\_record;

pipeline\_record.open("pipeline.txt", ios::in);

if (!pipeline\_record.is\_open())

{

cout << "文件打开失败！请确保文件存在！" << endl;

return;

}

char c;

pipeline\_record >> noskipws;

while (!pipeline\_record.eof())

{

pipeline\_record >> c;

cout << c;

}

pipeline\_record.close();

}

void CControl::Menu52StepRunPipeline()

{

//准备工作 开始

fetch.ResetPC();//使PC指针归位

pipeline.ResetAll();

RefershCurrentInstructionNum(); //刷新当前内存中共有的指令条数

int instruction\_num = current\_instruction\_num; //当前共有的指令条数

for (int i = 0; i < instruction\_num; i++)

{

int instruction = 0; //指令的编码

fetch.FetchInstructionFromMemory(fetch.GetPC(), memory.mem, instruction);

int instruction\_type; //指令类型

InstructionStruct instruction\_data; //指令的所有提取的数据

decoding.InstructionsDecoding(instruction, instruction\_data, instruction\_type);

if (decoding.InstructionCheckZero(instruction\_type, instruction\_data) == true) //查看指令是否写$zero0

{

pipeline.WriteBan(i, instruction\_type, instruction\_data);

}

else

{

pipeline.CreatePipeline(i, instruction\_type, instruction\_data);

}

fetch.PCSelfAdd(); //PC指针自增

}

//准备工作 结束

cout << "开始单步模拟执行流水线" << endl << endl;

int step = 0; //当前步数

int choice;

fetch.ResetPC();//使PC指针归位

while (1)

{

pipeline.CreatePipelineStep(step);

pipeline.WriteCoordinatesToFileType2(step);

//读取文件

cout << endl;

ifstream pipeline\_record;

pipeline\_record.open("pipeline\_step.txt", ios::in);

if (!pipeline\_record.is\_open())

{

cout << "文件打开失败！请确保文件存在！" << endl;

return;

}

char c;

pipeline\_record >> noskipws;

while (!pipeline\_record.eof())

{

pipeline\_record >> c;

cout << c;

}

pipeline\_record.close();

//检查5个状态是否存在

//Fetch取指状态

if (pipeline.IfExistInstructionOnOneTime(step, 'F') != NOTEXISTTHISINSTRUCTIONONONETIME)

{

int q = pipeline.IfExistInstructionOnOneTime(step, 'F');

cout << "序号" << q << "指令：Fetch取指\t取指内容：";

int instruction = 0; //指令的编码

fetch.FetchInstructionFromMemory(q \* 4, memory.mem, instruction);

for (int p = 31; p >= 0; p--) //输出二进制形式

cout << ((instruction >> p) & 1);

cout << endl;

}

//Decode译码状态

if (pipeline.IfExistInstructionOnOneTime(step, 'D') != NOTEXISTTHISINSTRUCTIONONONETIME)

{

int q = pipeline.IfExistInstructionOnOneTime(step, 'D');

cout << "序号" << q << "指令：Decode译码\t译码内容：";

int instruction = 0; //指令的编码

fetch.FetchInstructionFromMemory(q \* 4, memory.mem, instruction);

int instruction\_type; //指令类型

InstructionStruct instruction\_data; //指令的所有提取的数据

decoding.InstructionsDecoding(instruction, instruction\_data, instruction\_type);

string str = log.GetInstructionString(instruction\_type, instruction\_data);

cout << str << endl;

}

//Wait等待状态

if (pipeline.IfExistInstructionOnOneTime(step, 'U') != NOTEXISTTHISINSTRUCTIONONONETIME)

{

int q = pipeline.IfExistInstructionOnOneTime(step, 'U');

cout << "序号" << q << "指令：Wait等待\t指令暂停执行" << endl;

}

//Ban禁止状态

if (pipeline.IfExistInstructionOnOneTime(step, 'B') != NOTEXISTTHISINSTRUCTIONONONETIME)

{

int q = pipeline.IfExistInstructionOnOneTime(step, 'B');

cout << "序号" << q << "指令：Ban禁止\t要写入$zero，指令直接禁止执行！" << endl;

}

//Execute执行状态

if (pipeline.IfExistInstructionOnOneTime(step, 'E') != NOTEXISTTHISINSTRUCTIONONONETIME)

{

int q = pipeline.IfExistInstructionOnOneTime(step, 'E');

cout << "序号" << q << "指令：Execute执行\t执行内容：";

int instruction = 0; //指令的编码

fetch.FetchInstructionFromMemory(q \* 4, memory.mem, instruction);

int instruction\_type; //指令类型

InstructionStruct instruction\_data; //指令的所有提取的数据

decoding.InstructionsDecoding(instruction, instruction\_data, instruction\_type);

int p = execute.KnowExecuteResult(instruction\_type, instruction\_data, memory.mem, reg.reg, fetch.PC);

string str = execute.KnowExecuteDetail(instruction\_type, instruction\_data, memory.mem, reg.reg, fetch.PC);

cout << str << "\t执行结果：" << p << endl;

}

//Memory执行状态

if (pipeline.IfExistInstructionOnOneTime(step, 'M') != NOTEXISTTHISINSTRUCTIONONONETIME)

{

int q = pipeline.IfExistInstructionOnOneTime(step, 'M');

cout << "序号" << q << "指令：Memory访存\t访存内容：";

int instruction = 0; //指令的编码

fetch.FetchInstructionFromMemory(q \* 4, memory.mem, instruction);

int instruction\_type; //指令类型

InstructionStruct instruction\_data; //指令的所有提取的数据

decoding.InstructionsDecoding(instruction, instruction\_data, instruction\_type);

if (instruction\_type == I\_TYPE\_SW\_NO || instruction\_type == I\_TYPE\_LW\_NO) //只有这两条指令需要访存

{

int p = execute.KnowMemoryReadContent(instruction\_type, instruction\_data, memory.mem, reg.reg);

}

else

{

cout << "该指令无需访存！此过程为空！" << endl;

}

}

//WriteBack执行状态

if (pipeline.IfExistInstructionOnOneTime(step, 'W') != NOTEXISTTHISINSTRUCTIONONONETIME)

{

int q = pipeline.IfExistInstructionOnOneTime(step, 'W');

cout << "序号" << q << "指令：WriteBack写回\t写回内容：";

int instruction = 0; //指令的编码

fetch.FetchInstructionFromMemory(q \* 4, memory.mem, instruction);

int instruction\_type; //指令类型

InstructionStruct instruction\_data; //指令的所有提取的数据

decoding.InstructionsDecoding(instruction, instruction\_data, instruction\_type);

string des = execute.KnowWhichToWriteTo(instruction\_type, instruction\_data, memory.mem, reg.reg);

int p = execute.KnowExecuteResult(instruction\_type, instruction\_data, memory.mem, reg.reg, fetch.PC);

cout << des << " <- " << p << endl;

execute.ExecuteControlAction(instruction\_type, instruction\_data, memory.mem, reg.reg, fetch.PC);

}

//列出所有寄存器

cout << endl;

Menu44CheckRegister();

ShowPC(); //列出PC的值

//键盘控制

cout << endl << "如果要继续，请输入1，否则输入0返回主菜单： ";

cin >> choice;

if (choice == 0) return;

step++;

fetch.PCSelfAdd();

if (pipeline.CheckAnyInstruction(step) == false)

return;

}

}

void CControl::ShowPC()

{

int PC\_value = fetch.GetPC();

cout << "----当前PC指针的值：" << HEXNAME[(PC\_value >> 4) & 0x0f] << HEXNAME[(PC\_value >> 0) & 0x0f] << "H----" << endl;

}

void CControl::Menu53ClearPipeline()

{

pipeline.ResetPipelineCoordinates();

pipeline.ResetFunctionState();

pipeline.ResetInstructionStartAddress();

pipeline.ResetRegsiterTimeState();

pipeline.ResetIfWrite();

}

void CControl::PanelMenu8Exit()

{

exit(0);

}

int CControl::TransformFromCharToInstruction(char instru\_str[],int &instru\_result)

{

for (int i = 0; i < 32; i++)

{

if (instru\_str[i] != '0'&&instru\_str[i] != '1')

{

instru\_result = INSTRUCTION\_TRANSFORM\_ERROR\_CODE;

}

}

for (int i = 0; i < 32; i++)

{

if (instru\_str[i] == '1') instru\_result |= (1 << (31 - i));

else if (instru\_str[i] == '0') instru\_result &= ~(1 << (31 - i));

}

return 0;

}

int CControl::IfInstructionOK(char str[])

{

//第一次循环，检查是否有其他字符

for (int i = 0; i < strlen(str); i++)

if (str[i] != '0' && str[i] != '1')

return INSTRUCTION\_CODE\_ERROR;

//第二次循环，检查是否长度符合规范

if (strlen(str) != 32) return INSTRUCTION\_CODE\_ERROR;

return INSTRUCTION\_CODE\_OK;

}

**cpp文件 main.cpp**

#include "control.h"

#include <stdlib.h>

int main()

{

CControl control;

control.AllBegin();

system("PAUSE");

return 0;

}