

tomasulo动态调度算法模拟

学号 17300240016 姓名 何千羽 学院 计算机科学技术学院 完成时间 2019/5/27

tomasulo动态调度算法模拟

一、实验目的

二、实验环境

三、功能实现的说明

1.显示界面+读取

- (1) 文件内容
- (2) 思路以及源代码

2.START键——单步

- (1) 界面展示
- (2) 思路以及源代码
 - a.对于issue函数
 - b.对于execute函数
 - c.对于writeback函数

3.AUTO键——自动运行

- (1) 自动运行
 - a.界面展示
 - b.思路以及源代码
- (2) PAUSE/CONTINUE键——暂停或重新开始
 - a.界面展示
 - b.思路以及源代码
- (3) STOP键——停止
 - a.界面展示
 - b.思路以及源代码

4.RESET键——重置

- (1) 界面展示
- (2) 思路以及源代码

5.SAVE键——保存

- (1) 界面展示
- (2) 思路以及源代码

6.EDIT键——设置

- (1) 界面展示
- (2) 思路以及源代码

四、思路以及源代码

五、参考文献

一、实验目的

理解指令动态调度的原理，尤其是tomasulo动态调度算法的执行过程。

二、实验环境

使用软件：Visual Studio 2019

程序类型：Winform窗体应用

使用语言：C#

三、功能实现的说明

1. 显示界面+读取

显示当前时钟、指令状态表、保留站信息表和寄存器状态表

The screenshot shows a Windows application window titled "Form1". The interface is divided into several sections:

- Instruction Table:** A grid showing instructions issued from F0 to F6. The columns are: Instruction, Issue, Exec-begin, Exec-end, and WriteBack.
- Control Buttons:** Buttons for START, CLOCK:1, AUTO, CLOCK:0, PAUSE/CONTINU, STOP, RESET, and EDIT.
- Reservation Station Table:** A table titled "Content of the ReservationStation" showing reservation station slots (R1-R8) for various operations like LOAD, ADD, MUL, and MULT. It includes columns for Name, Busy, OP, Vj, Vk, Qj, Qk, and A.
- Register Table:** A table titled "Register" showing the state of registers F0-F12. The columns are Field, F0, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, and F12. An example entry shows F6 as LOAD1.
- SAVE Button:** A large green button labeled "SAVE" located at the bottom right.

指令是从read_from_file()函数中，从instruction.txt文件中读入，若需要运行可执行文件，需要更改文件路径。

(1) 文件内容



文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

1	L.D	F6,34(R2)
2	L.D	F2,45(R3)
3	MUL.D	F0,F2,F4
4	SUB.D	F8,F2,F6
5	DIV.D	F10,F0,F6
6	ADD.D	F6,F8,F2

(2) 思路以及源代码

直接触发，按行读取指令形成string，因为split函数不知为何在本电脑上出现问题，故选择采用最基础的判断分割方法。解析每行string，为每条指令的每个变量赋值。同时，load函数与非load函数分开讨论。

```
private void read_from_file()
{
    int mark = 0;
    System.IO.StreamReader file = new
System.IO.StreamReader(@"\\Mac\\Home\\Desktop\\instruction.txt");
    while ((line = file.ReadLine()) != null)
    {
        mark = 0;
        int flag = 0;
        Instruction I0 = new Instruction();
        for (int i = 0; i < line.Length-1; i++)
        {
            if (flag == 0)
            {
                mark += line[i] - 48;
                if (line[i + 1] == ' ' || line[i + 1] == '\t') flag =
1;
                else mark = mark * 10;
            }
            else if (flag == 1&&line[i]!=' '&&line[i]!='\t')
            {
                I0.name += line[i];
                if (line[i + 1] == ' ' || line[i + 1] == '\t')
```

```

    {
        switch (I0.name)
        {
            case "ADD.D":
                I0.op = AddOp;
                break;
            case "SUB.D":
                I0.op = SubOp;
                break;
            case "MUL.D":
                I0.op = MultOp;
                break;
            case "DIV.D":
                I0.op = DivOp;
                break;
            case "L.D":
                I0.op = LoadOp;
                break;
        }
        flag = 2;
        I0.name += " ";
    }

}

else if (flag == 2 && line[i] != ' ' && line[i] != '\t'){

    if (I0.name == "L.D ")
    {

        if (flag == 2 && char.IsDigit(line[i]))
        {
            I0.rd += line[i] - 48;
            if (line[i + 1] != ',') I0.rd = I0.rd * 10;
            else flag = 3;
        }
    }
    else if (flag == 2 && char.IsDigit(line[i]))
    {
        I0.rd += line[i] - 48;
        if (line[i + 1] != ',') I0.rd = I0.rd * 10;
        else flag = 3;
    }
}
else if (flag == 3 && char.IsDigit(line[i]))
{
    if(I0.name == "L.D ")
    {
        I0.imm += line[i] - 48;
        if (line[i + 1] != '(') I0.imm = I0.imm * 10;
        else flag = 4;
    }
}

```

```

        else
        {
            I0.rs += line[i] - 48;
            if (line[i + 1] != ',') I0.rs = I0.rs * 10;
            else flag = 4;
        }
    }
    else if (flag == 4)
    {
        if (I0.name == "L.D ")
        {
            if (char.IsDigit(line[i]))
            {
                I0.rs += line[i] - 48;
                if (line[i + 1] != ')') I0.rs = I0.rs * 10;
                else
                {
                    flag = 5;
                    break;
                }
            }
        }
        else
        {

            if (i + 1 == line.Length - 1)
            {
                if (char.IsDigit(line[i]))
                {
                    I0.rt += line[i] - 48;
                    I0.rt = I0.rt * 10;
                    I0.rt += line[i + 1] - 48;
                    break;
                }
                else I0.rt = line[i + 1] - 48;
            }
            else continue;
        }
    }
    Inst[mark - 1] = I0;
}
Inst_size = mark;
}

```

2.START键——单步

从运行开始，用户每按一下"START"键，当前时钟增加1，更新四个表。

(1) 界面展示

Instruction					State					Content of the ReservationStation				
Instruction	Issue	Exc-begin	Exc-end	WriteBack	Name	Busy	Op	Vj	Vk	Qj	Qk	A		
L.D F6,34(R2)	1	2	0	0	LOAD1	True	L.D					34		
L.D F2,45(R3)	2	0	0	0	LOAD2	True	L.D					45		
MUL.D F0,F2,F4	0	0	0	0	ADD1	False								
SUB.D F8,F2,F6	0	0	0	0	ADD2	False								
DIV.D F10,F0,F6	0	0	0	0	ADD3	False								
ADD.D F6,F8,F2	0	0	0	0	MULT1	False								
					MULT2	False								

ResStation													Register			
Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12			
Qj	LOAD2															
	LOAD1															

Instruction					State					Content of the ReservationStation				
Instruction	Issue	Exc-begin	Exc-end	WriteBack	Name	Busy	Op	Vj	Vk	Qj	Qk	A		
L.D F6,34(R2)	1	2	3	0	LOAD1	True	L.D					34+Regs[F2]		
L.D F2,45(R3)	2	3	0	0	LOAD2	True	L.D					45		
MUL.D F0,F2,F4	3	0	0	0	ADD1	False								
SUB.D F8,F2,F6	0	0	0	0	ADD2	False								
DIV.D F10,F0,F6	0	0	0	0	ADD3	False								
ADD.D F6,F8,F2	0	0	0	0	MULT1	True	MULT.D		Regs[F4]			LOAD2		
					MULT2	False								

ResStation													Register			
Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12			
Qj	MULT1															
	LOAD2															
	LOAD1															

(2) 思路以及源代码

通过Start按钮触发，每次执行RunOneCycle()函数，其可以完成这个时钟周期的 指令流出、执行与写入操作、并更新四个表（时钟表、指令状态表、保留站状态表、寄存器状态表），故完成更新。

```

private void Button1_Click(object sender, EventArgs e)
{
    Clock++;
    RunOneCycle();
    Update_Clock_Board();
}

private void RunOneCycle()
{
    //完成这个时钟周期的 指令流出、执行与写入操作
    ISSUE(Inst, ResStation, RegisterStatus);
    EXECUTE(Inst, ResStation, RegisterStatus);
    WRITEBACK(Inst, ResStation, RegisterStatus);
    Done = false;
}

```

```

if (Total_WRITEBACKS == Inst_size)
    Done = true;
//更新四个表
Update_Instruction_Board();
Update_ReservationStation_Board();
Update_Register_Board();
//具体代码随实验报告附上
}

```

a.对于issue函数

- 判断是否有空闲保留站

已有ResStation[]数组，遍历数组判断是否有空闲保留站。通过Num_LOAD_RS、Num_ADD_RS、Num_MULT_RS在遍历时区分不同的保留站。

```

int RsLoadStart = Num_LOAD_RS - Num_LOAD_RS;
int RsLoadEnd = Num_LOAD_RS;
int RSAddStart = Num_LOAD_RS;
int RSAddEnd = Num_LOAD_RS + Num_ADD_RS;
int RSSubStart = Num_LOAD_RS;
int RSSubEnd = Num_LOAD_RS + Num_ADD_RS;
int RSMulStart = Num_LOAD_RS + Num_ADD_RS;
int RSMulEnd = Num_LOAD_RS + Num_ADD_RS + Num_MULT_RS;
switch (r)
{
    case LoadOp:
        for(int i = RsLoadStart; i < RsLoadEnd; i++)
        {
            if (!RESSTATION[i].busy)
            {
                r = i;
                currentInst_ISSUE++;
                RESSTATION[i].op = LoadOp;
                rsFree = true;
                break;
            }
        }
        if (!rsFree)
            return 1;
        break;
    case AddOp:
        for (int i = RSAddStart; i < RSAddEnd; i++)
        {
            if (!RESSTATION[i].busy)
            {
                r = i;
                currentInst_ISSUE++;

```

```

        RESSTATION[i].op = AddOp;
        rsFree = true;
        break;
    }
}

//.....MultOp/DivOp类似

```

- 若保留站空闲，则将信息更新入保留站

- 将load和其他指令分开讨论部分
- 对于非load指令

Vj和Qj不可兼得，故分别讨论指令所需操作数是否准备好，若准备好

```
if (REGSTATUS[INST[currentInst_ISSUE - 1].rs].Qi == RegStatusEmpty)
```

则更新Vj，将Qj设为OperandAvailable

```

RESSTATION[r].Vj += "Regs[F" + INST[currentInst_ISSUE - 1].rs + "]";
RESSTATION[r].Qj = OperandAvailable;

```

否则将即将产生此操作数的保留站记录在 RESSTATION[r].Qj中，该过程由寄存器状态表 (REGSTATUS[]) 为媒介完成

```
RESSTATION[r].Qj = REGSTATUS[INST[currentInst_ISSUE - 1].rs].Qi;
```

Vk、Qk同理

- 对于load指令

将A部分的值更改为指令中的立即数字段

```
RESSTATION[r].A += INST[currentInst_ISSUE - 1].imm.ToString() ;
```

- 相同执行部分

- 将保留站状态更新为true

```
RESSTATION[r].busy = true;
```

- 将保留站此刻的指令记录下来

```
RESSTATION[r].instNum = currentInst_ISSUE - 1;
```

- 记录下此刻的issueClock，之后可以显示在指令状态表上

```
INST[currentInst_ISSUE - 1].issueClock = Clock;
```

- 更新寄存器状态表实现后续换名

```
REGSTATUS[INST[currentInst_ISSUE - 1].rd].Qi = r;
```

- 保证issue执行一个时钟周期

即issue部分执行的代码在这个周期流入，至少得在下一个周期流出。因为代码书写时为

```
issue();
execute();
writeback();
```

出现了不可避免却不存在的先后顺序。若缺少这一个变量则可能出现，issue部分使该指令流入保留站，下一行的代码execute()便使它在同一周期开始执行的错误情形。

```
// reset ISSUE latency for RS
RESSTATION[r].ISSUE_Lat = 0;
```

b.对于execute函数

遍历保留站列表 (RESSTATION[])

- 非load指令经过三个判断后可执行，load指令经过两个判断后可执行
 - 判断是否有空闲

```
if (RESSTATION[r].busy == true)
```

- 保证issue执行一个时钟周期

```
if (RESSTATION[r].ISSUE_Lat >= ISSUE_Lat)
```

- 非 (load指令) 是否两个操作数均已准备好

```
if ((RESSTATION[r].Qj == OperandAvailable &&
RESSTATION[r].Qk == OperandAvailable) || (RESSTATION[r].op == LoadOp))
```

- 开始执行

- 记录executeBegin时钟

```
if (INST[RESSTATION[r].instNum].executeClockBegin == 0)
{
    INST[RESSTATION[r].instNum].executeClockBegin = Clock;
}
```

- 判断是否达到该指令的延迟时间

```
RESSTATION[r].lat++;
```

- 达到后进行一系列操作

其中，比较奇怪的

```
Register_Memory[INST[RESSTATION[r].instNum].rd] = "000";
```

是专门为寄存器内容来自存储器的寄存器设计，和后续代码有关。为了展现出
Mem[34+Reg[R3]]的效果

```
switch (RESSTATION[r].op)
{
    case (AddOp):
        if (RESSTATION[r].lat == ADD_Lat)
        {
            //RESSTATION[r].result = RESSTATION[r].Vj + RESSTATION[r].Vk;
            // Result is ready to be writtenback
            RESSTATION[r].resultReady = true;
            RESSTATION[r].lat = 0;
            // Set clock cycle when execution ends
            INST[RESSTATION[r].instNum].executeClockEnd = Clock;
            // reset ISSUE latency for RS
            RESSTATION[r].ISSUE_Lat = 0;
            Register_Memory[INST[RESSTATION[r].instNum].rd] = "000";
            //专门为寄存器内容来自存储器的寄存器设计，和后续代码有关。为了展现出
            Mem[34+Reg[R3]]的效果
        }
        .......//SubOp、MultOp、DivOp类似
    case (LoadOp):
        if (RESSTATION[r].lat == LOAD_Lat)
        {
            RESSTATION[r].resultReady = true;
            RESSTATION[r].lat = 0;
            // Set clock cycle when execution ends
            INST[RESSTATION[r].instNum].executeClockEnd = Clock;
            RESSTATION[r].A =
INST[RESSTATION[r].instNum].imm.ToString() + "+Regs[R" +
INST[RESSTATION[r].instNum].rs + "]";
            //专门为寄存器内容来自存储器的寄存器设计，和后续代码有关。为了展现出
            Mem[34+Reg[R3]]的效果
            Register_Memory[INST[RESSTATION[r].instNum].rd] = "Mem[" +
RESSTATION[r].A + "]";
            // reset ISSUE latency for RS
            RESSTATION[r].ISSUE_Lat = 0;
        }
        break;
}
```

.....

c.对于writeback函数

部分判断、操作与Execute函数类似

- 遍历RegisterStatus[], 若某寄存器的Qi等于正在操作的保留站，则赋值

```
if (REGSTATUS[x].Qi == r)
{
    // Write back to Registers
    //REG[x] = RESSTATION[r].result;
    REGSTATUS[x].Qi = RegStatusEmpty;
}
```

- 通过CDB，在保留站状态表上，将等待该保留站产生结果的所有保留站名字换名为Reg[]形式，从而达成更新状态表的目的

```
for (int y = 0; y < ResStation_size; y++)
{
    // check if any reservation stations are
    // waiting for the given result as an operand
    // Write back to reservation stations
    // Given RS is not longer waiting for this
    // operand value

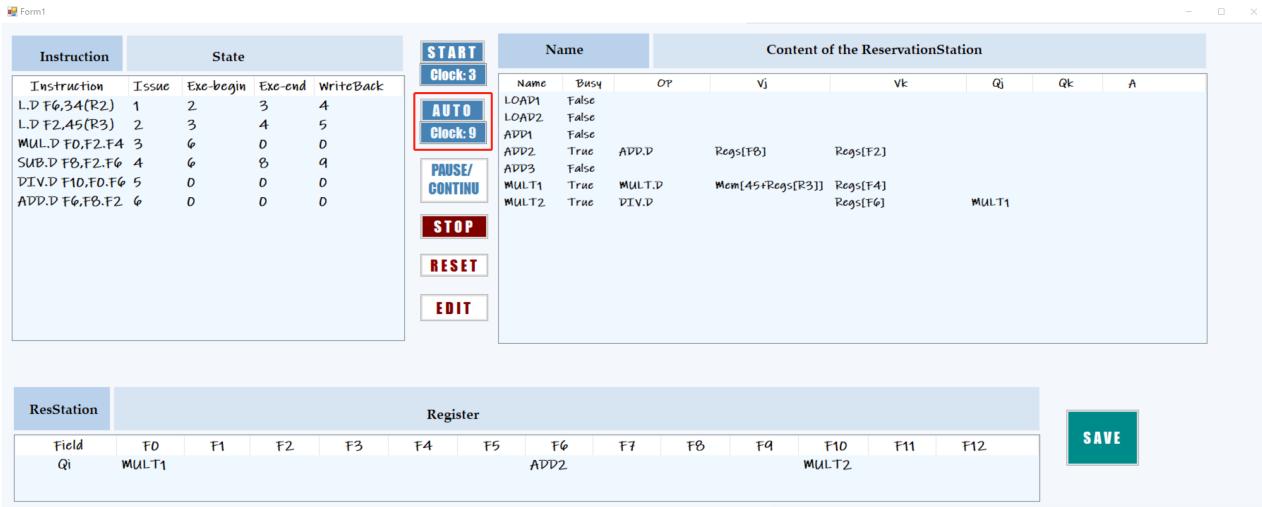
    if (RESSTATION[y].Qj == r)
    {
        if (Register_Memory[INST[RESSTATION[r].instNum].rd] != "000")
            RESSTATION[y].Vj = Register_Memory[INST[RESSTATION[r].instNum].rd];
        else RESSTATION[y].Vj = "Regs[" + Inst[RESSTATION[r].instNum].rd +
    "]";
        RESSTATION[y].Qj = OperandAvailable;
    }
    //....Qk同理
}
```

3.AUTO键——自动运行

(1) 自动运行

模拟程序按照每秒当前时钟增加1，并同步更新系统状态。

a.界面展示



b.思路以及源代码

通过pause_or_continue参数控制是否继续自动执行。

通过

```
System.Threading.Thread.Sleep(1000);
```

保证程序间隔一段时间自动执行一次

```
//auto
private void Button2_Click(object sender, EventArgs e)
{
    run = true;
    while (run)
    {
        if (!pause_or_continue) break;
        //缺少一个执行完毕可以停下来
        //if (instructionUnit.GetCurrentInstructions().Length == 0)
break;
        //如果每个保留站都是空的，则完成
        System.Threading.Thread.Sleep(1000);
        Clock++;
        RunOneCycle();
        Update_Clock_Board1();
        //TODO:每个保留站检查是否清空
        Application.DoEvents();
        if (IsAllBufferFree()) break;
    }
}
```

(2) PAUSE/CONTINUE键——暂停或重新开始

为保证自动运行可操控性，增加暂停键以及重新开始键（可能有些不灵敏）。按下后程序可暂停或重新开始。

a.界面展示

Instruction		State				Name		Content of the ReservationStation					
Instruction	Issue	Exc-begin	Exc-end	WriteBack	Name	Busy	OP	V _j	V _k	Q _j	Q _k	A	
L.D F6,34(R2)	1	2	3	4	LOAD1	False							
L.D F2,45(R3)	2	3	4	5	LOAD2	False							
MUL.D F0,F2,F4	3	6	0	0	ADD1	False							
SUB.D F8,F2,F6	4	6	8	9	ADD2	True	ADD.D	Regs[F8]	Regs[F2]				
DIV.D F10,F0,F6	5	0	0	0	ADD3	False							
ADD.D F6,F8,F2	6	0	0	0	MULT1	True	MULT.D	Mem[45+Regs[R3]]	Regs[F4]				
					MULT2	True	DIV.D	Regs[F6]	Regs[F6]			MULT1	

ResStation		Register												
Field	Q _i	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
MULT1														
ADD2														

b.思路以及源代码

如上文所述，通过pause_or_continue参数控制是否继续自动执行。

```
//pause
private void Button4_Click(object sender, EventArgs e)
{
    pause_or_continue = !pause_or_continue;
    Button2_Click(sender, e); //auto的代码
}
```

(3) STOP键——停止

为保证自动运行可操控性，增加停止键（可能有些不灵敏）。按下后程序停止，可退出运行。

a.界面展示

Instruction		State				Name		Content of the ReservationStation					
Instruction	Issue	Exc-begin	Exc-end	WriteBack	Name	Busy	OP	V _j	V _k	Q _j	Q _k	A	
L.D F6,34(R2)	1	2	3	4	LOAD1	False							
L.D F2,45(R3)	2	3	4	5	LOAD2	False							
MUL.D F0,F2,F4	3	6	0	0	ADD1	False							
SUB.D F8,F2,F6	4	6	8	9	ADD2	False							
DIV.D F10,F0,F6	5	0	0	0	ADD3	False							
ADD.D F6,F8,F2	6	10	12	13	MULT1	True	MULT.D	Mem[45+Regs[R3]]	Regs[F4]				
					MULT2	True	DIV.D	Regs[F6]	Regs[F6]			MULT1	

ResStation		Register												
Field	Q _i	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
MULT1														
ADD2														

b.思路以及源代码

```

private void Button3_Click(object sender, EventArgs e)
{
    run = false; //同样与auto部分代码有关
}

```

4.RESET键——重置

恢复到系统初始状态，当前时钟设置为0，同时指令状态表、保留站信息表和寄存器状态表设置为初始状态；

(1) 界面展示

Instruction		State				Name		Content of the ReservationStation					
Instruction	Issue	Exe-begin	Exe-end	WriteBack	Name	Busy	Op	Vj	Vk	Qj	Qk	A	
L.D F6,34(R2)	0	0	0	0	LOAD1	False							
L.D F2,45(R3)	0	0	0	0	LOAD2	False							
MUL.D F0,F2,F4	0	0	0	0	ADD1	False							
SUB.D F0,F2,F6	0	0	0	0	ADD2	False							
DIV.D F10,F0,F6	0	0	0	0	ADD3	False							
ADD.D F6,F8,F2	0	0	0	0	MULT1	False							
					MULT2	False							

ResStation		Register												
Field	Qj	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12

SAVE

重新点击START键一切都会从头再来。

Instruction		State				Name		Content of the ReservationStation					
Instruction	Issue	Exe-begin	Exe-end	WriteBack	Name	Busy	Op	Vj	Vk	Qj	Qk	A	
L.D F6,34(R2)	1	2	3	0	LOAD1	True	L.D						
L.D F2,45(R3)	2	3	0	0	LOAD2	True	L.D						
MUL.D F0,F2,F4	3	0	0	0	ADD1	False							
SUB.D F0,F2,F6	0	0	0	0	ADD2	False							
DIV.D F10,F0,F6	0	0	0	0	ADD3	False							
ADD.D F6,F8,F2	0	0	0	0	MULT1	True	MULT.D						
					MULT2	False							

ResStation		Register												
Field	Qj	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
	MULT1							LOAD1						

SAVE

(2) 思路以及源代码

重新加载所有变量，对已改变的全局变量进行恢复，更新所有表

```

//Reset
private void Button5_Click_1(object sender, EventArgs e)
{
    //重新initialize
}

```

```

Init_Instruction();
Init_ReservationStation();
Init_Regiser();
Init_Clock();
//重新更新表
Update_Clock_Board1();
Update_Clock_Board();
Update_Instruction_Board();
Update_ReservationStation_Board();
Update_Register_Board();

//***** Define Architecture
run = true;
pause_or_continue = true;
Clock = 0;
// used to check if INST == WRITEBACKS to end program
Done = true;
Total_WRITEBACKS = 0;
// Counter for current instruction to issue
currentInst_ISSUE = 0;
Register_Memory = new string[12] { "000", "000", "000", "000",
"000", "000", "000", "000", "000", "000", "000" };
}

```

5.SAVE键——保存

把当前状态保存在文件中，包括指令状态表、保留站信息表和寄存器状态表。若需要运行可执行文件，需要更改文件路径。

(1) 界面展示

Instruction		State				Name		Content of the ReservationStation					
Instruction	Issue	Exe-begin	Exe-end	WriteBack		Name	Busy	Op	Vj	Vk	Qj	Qk	A
L.D F6,34(R2)	1	2	3	0		LOAD1	True	L.D					34 fRegs[R2]
L.D F2,45(R3)	2	3	0	0		LOAD2	True	L.D					45
MUL.D F0,F2,F4	3	0	0	0		ADD1	False						
SUB.D F8,F2,F6	0	0	0	0		ADD2	False						
DIV.D F10,F0,F6	0	0	0	0		ADD3	False						
ADD.D F6,F8,F2	0	0	0	0		MULT1	True	MUL.D					
						MULT2	False						

ResStation		Register									
Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
Qj	MULT1		LOAD2				LOAD1				

instruction1 - 记事本						
Instruction	IS	Ex-Begin	Ex-end	WB		
L.D F6,34(R2)	1	2	3	0		
L.D F2,45(R3)	2	3	0	0		
MUL.D F0,F2,F4	3	0	0	0		
SUB.D F8,F2,F6	0	0	0	0		
DIV.D F10,F0,F6	0	0	0	0		
ADD.D F6,F8,F2	0	0	0	0		

(2) 思路以及源代码

使用System.IO.StreamWriter函数。

```

//Save
private void Button8_Click(object sender, EventArgs e)
{
    using (System.IO.StreamWriter sw = new
System.IO.StreamWriter(@"\\Mac\\Home\\Desktop\\instruction1.txt")) //若执行文件需要
更改路径
    {
        //保存指令状态表
        sw.WriteLine("      Instruction      IS      Ex-Begin      Ex-end
WB");
        for(int i=0;i<Inst_size;i++)
        {
            string Instruction_output1, Instruction_output2,
Instruction_output3, Instruction_output4, Instruction_output5;
            Instruction_output2 = Inst[i].issueClock.ToString();
            Instruction_output3 =
Inst[i].executeClockBegin.ToString();
            Instruction_output4 = Inst[i].executeClockEnd.ToString();
            Instruction_output5 = Inst[i].writebackClock.ToString();
            int j = 0;
            int k = 13 - Inst[i].issueClock.ToString().Length;
            if (Inst[i].op == LoadOp)
            {
                Instruction_output1 = Inst[i].name + " F" + Inst[i].rd
+ "," + Inst[i].imm + "(R" + Inst[i].rs + ") ";
                j = 25 - Instruction_output1.Length;
            }
            else
            {
                Instruction_output1 = Inst[i].name + " F" + Inst[i].rd
+ ",F" + Inst[i].rs + ",F" + Inst[i].rt;
                j = 25 - Instruction_output1.Length;
            }
            sw.WriteLine(Instruction_output1 +
string.Empty.PadRight(j, ' ') + Instruction_output2 + string.Empty.PadRight(k,
' ') +
                Instruction_output3 + string.Empty.PadRight(k, ' ') +
Instruction_output4 + string.Empty.PadRight(k, ' ') + Instruction_output5);
        }
        sw.WriteLine(" -----");
    };
}

//保存保留站状态表
sw.WriteLine("      Name      " + "      Busy      " + "      Op      "
+ "      Vj      " + "      Vk      " + "      Qj      " + "      Qk      " + "      A
");
for(int i = 0; i < ResStation_size; i++)
{

```

```

        string str1 = " ";
        string str2, str3, str4;

        if (ResStation[i].busy)
        {
            switch (ResStation[i].op)
            {
                case 0:
                    str1 = "ADD.D";
                    break;
                case 1:
                    str1 = "SUB.D";
                    break;
                case 2:
                    str1 = "MUL.D";
                    break;
                case 3:
                    str1 = "DIV.D";
                    break;
                case 4:
                    str1 = "L.D";
                    break;
                default:
                    break;
            }
            if (ResStation[i].Qj == OperandAvailable)
                str2 = ResStation[i].Vj.ToString();
            else str2 = "      ";
            if (ResStation[i].Qk == OperandAvailable)
                str3 = ResStation[i].Vk.ToString();
            else str3 = "      ";
        }
        else
        {
            str1 = "      ";
            str2 = "      ";
            str3 = "      ";
        }

        if (ResStation[i].Qj == OperandInit) str4 = ""
        ;
        else if (ResStation[i].Qj == OperandAvailable) str4 = ""
        ;
        else str3 = ResStation[ResStation[i].Qj].name;

        if (ResStation[i].Qk == OperandInit) str4 = "      ";
        else if (ResStation[i].Qk == OperandAvailable) str4 = "      ";
        else str4 = ResStation[ResStation[i].Qk].name;
    }
}

```

```

        sw.WriteLine(ResStation[i].name + "      " +
ResStation[i].busy.ToString() + "      "+ str1 + "      " + str2 + "      "
+ str3 + "      " + str4);

    }

//保存寄存器状态表
sw.WriteLine("-----");
-----");

string final = " ";
for (int i = 0; i < RegisterStatus_size; i++)
{
    if (RegisterStatus[i].Qi == RegStatusEmpty) final += " ";
    else final += ResStation[RegisterStatus[i].Qi].name + "(F"
+ i + ")");
}
sw.WriteLine("      Qi      " + final);
}

}

```

6. EDIT键——设置

设置上面的配置信息，包括功能部件的延迟和功能部件的个数。由于时间原因并没有完善容错性，导致只能在所有textbox均填写完成才不会出错。填写完成后点击OK便更改完成，点击BACK返回到当前界面。

(1) 界面展示

The screenshot shows a software interface for managing a reservation station and registers. The top section displays the 'Content of the ReservationStation' table:

Instruction		State				Name								Content of the ReservationStation							
Instruction	Issue	Exe-begin	Exe-end	WriteBack	Name	Busy	Op	Vj	Vk	Qj	Qk	A									
L.D F6,34(R2)	1	2	4	5	LOAD1	False															
L.D F2,45(R3)	2	3	5	6	LOAD2	False															
MUL.D F0,F2,F4	3	7	16	17	ADD1	False															
SUB.D F0,F2,F6	4	7	11	12	ADD2	False															
DIV.D F10,F0,F6	18	0	0	0	ADD3	False															
ADD.D F6,F0,F2	0	0	0	0	ADD4	False															
					MULT1	True	DIV.D	Mem[45]+Regs[R3...]	Regs[F6]												

Control buttons include START, AUTO, PAUSE/CONTINUO, STOP, and RESET.

The bottom section displays the 'Register' table:

ResStation		Register												
Field	Q _i	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
												MULT1		

A large blue 'EDIT' button is highlighted in the center of the interface.

↓注释：只能在所有textbox均填写完成才不会出错

Form2

The number of ResvertonStation		Latency of Instruction			
LOAD	<input type="text"/>	L.D	<input type="text"/>		
ADD	<input type="text"/>	ADD.D	<input type="text"/>		
MULT	<input type="text"/>	MULT.D	<input type="text"/>		
OK		BACK		DIV.D	<input type="text"/>

*TIP1: Only when you finish all the boxes can you press the "OK" button or you will get back error message
TIP2: "BACK" is for you to go back to the former page*

Form2

The number of ResvertonStation		Latency of Instruction			
LOAD	<input type="text"/> 2	L.D	<input type="text"/> 3		
ADD	<input type="text"/> 4	ADD.D	<input type="text"/> 4		
MULT	<input type="text"/> 4	MULT.D	<input type="text"/> 12		
OK		BACK		DIV.D	<input type="text"/> 41

*TIP1: Only when you finish all the boxes can you press the "OK" button or you will get back error message
TIP2: "BACK" is for you to go back to the former page*

↓返回Form1界面后，保留站数目、指令的延迟数均发生相应改变

Form1

Instruction	Issue	State			Name	Content of the ReservationStation					
		Exe-begin	Exe-end	WriteBack		Busy	Op	Vj	Vk	Qj	Qk
L.D F6,F4,R2	1	2	4	5	False						
L.D F2,F4,R3	2	3	5	6	False						
MUL.D F0,F2,F4	3	7	18	19	False						
SUB.D F8,F2,F6	4	7	10	11	False						
DIV.D F10,F0,F6	20	0	0	0	False						
ADD.D F4,F8,F2	0	0	0	0	True	DIV.D	Mem[45] & Regs[R3...]	Regs[F6]			

START Clock: 20

AUTO Clock: 0

PAUSE/CONTINU

STOP

RESET

EDIT

ResStation		Register													
Field	Q _i	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	
												MULT1			

(2) 思路以及源代码

- 通过Form2中的输入将改变的变量传入Form1中

```

//在Form2.cs中
private void Button2_Click(object sender, EventArgs e)
{
    Num_LOAD_RS = int.Parse(textBox1.Text);
    Num_ADD_RS = int.Parse(textBox2.Text);
    Num_MULT_RS = int.Parse(textBox3.Text);
    Num_LOAD_lat = int.Parse(textBox4.Text);
    Num_ADD_lat = int.Parse(textBox5.Text);
    Num_MULT_lat = int.Parse(textBox6.Text);
    Num_DIV_lat = int.Parse(textBox7.Text);
    parent.SetResStation(Num_LOAD_RS, Num_ADD_RS, Num_MULT_RS,
    Num_LOAD_lat, Num_ADD_lat, Num_MULT_lat, Num_DIV_lat);
}

```

- Form1接受变量

```

Num_LOAD_RS = num_load;
Num_ADD_RS = num_add;
Num_MULT_RS = num_mult;
LOAD_Lat = load_lat;
ADD_Lat = add_lat;
MULT_Lat = mult_lat;
DIV_Lat = div_lat;

```

- Form1重新初始化保留站（以Load为例）

```

//update the reservationStation
for (int i = 0; i < Num_LOAD_RS; i++)
{
    ReservationStation LOAD = new ReservationStation(LoadOp, OperandInit,
    "LOAD" + (i + 1).ToString());
    ResStation[i] = LOAD;
}

```

- 初始化其他所有变量，并更新所有表

四、思路以及源代码

见附件

五、参考文献

使用部分思想：github.com/milleraj66/ECE585_TomasuloAlgorithm