1093322 PROJECT3 Tomasulo Algorithm

1.Global Variable

a.

```
const int ADDcode = 0;
const int ADDIcode = 1;
const int SUBcode = 2;
const int MULcode = 3;
const int DIVcode = 4;
```

將不同**operation**定義一個數字來辨識

b.

```
const int ADDlatency = 2;
const int MULlatency = 6;
const int DIVlatency = 8;
```

定義不同**operation**所需的**latency**
其中**ADDlatency**包含**ADD**、**SUB**、**ADDI**

c.

```
const int ADD_Nums = 3;
const int MULandDIV_Nums = 2;
```

定義每種**operation**所能使用的**reservation station**數
RS1~3能給**ADD**、**SUB**、**ADDI**
RS4~5能給**MUL**、**DIV**

d.

```
int cycleNum = 0;
int WRnums = 0;
int InstIssue = 0;
```

cycleNum:計算**cycle**總數
WRnums:計算**WriteBack**總數
InstIssue:判別當前**Issue**的**Instruction**

```
const int REGzero = 10000;
const int RATempty = 10001;
const int QperandAva = 10002;
const int OperandInit = 10003;
```

用來初始化以、判別對應容器是否為空、判斷變數是否可取得

2.Data Structure

大部分資料結構中的變數都直接採用**Tamasulo Algorithm**裡定義的變數名稱

a.Instruction

```
class Inst {
public:
    int rd;
    int rs;
    int rt;
    int constant;//for immediate inst
    int operation;
    int issue;

    Inst();
    Inst(int, int, int, int, int);
};

#endif //_INST_H
```

int constant:專門給**immediate instruction(ex. ADDI)**來存取常數值以便跟**rt**區隔

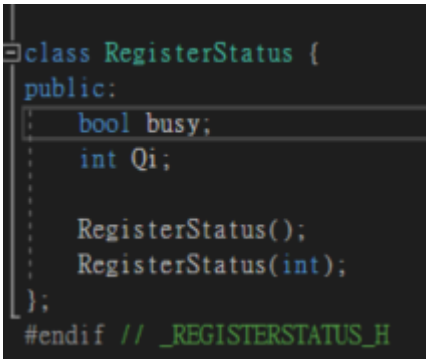
b.ReservationStation

```
class ReservationStation {
public:
    bool busy;
    int Qj;
    int Qk;
    int Vj;
    int Vk;
    int latency;
    int operation;
    int result;
    bool complete;
    int InstNum;
    int IssueLatency;
    int WriteBackLatency;
    int unCompleteResultJ;
    int unCompleteResultK;
    ReservationStation();
    ReservationStation(int, int);

};
#endif //_RS_H
```

int unCompleteResultJ、int unCompleteResultK:專門用來儲存尚未完成運算的變數所在的**ReservationStation**裡的編號

c.RegisterStatus



```
class RegisterStatus {
public:
    bool busy;
    int Qi;

    RegisterStatus();
    RegisterStatus(int);
};
#endif // _REGISTERSTATUS_H
```

d.Array

Instruction用**vector<Inst> Instructions**來儲存

ReservationStation用**vector<ReservationStation>ResStations**來儲存

RegisterStatus用**vector<RegisterStatus>RATarr**來儲存

3.Program Function

a.main

```
ifstream file;
file.open("input.txt");
string s;
while (getline(file, s)) {
    stringstream ss(s);
    string word;
    vector<string>instruction;
    while (ss >> word) {
        if (word[word.size() - 1] == ',') word.pop_back();
        if (word[0] == 'F') word.erase(word.begin());
        instruction.push_back(word);
    }
    int rd = atoi(instruction[1].c_str()), rs = atoi(instruction[2].c_str()), rt = atoi(instruction[3].c_str());
    if (instruction[0] == "ADD") {
        Inst I(rd, rs, rt, 0, ADDcode);
        Instructions.push_back(I);
    }
    else if (instruction[0] == "ADDI") {
        Inst I(rd, rs, 0, rt, ADDIcode);
        Instructions.push_back(I);
    }
    else if (instruction[0] == "SUB") {
        Inst I(rd, rs, rt, 0, SUBcode);
        Instructions.push_back(I);

    }
    else if (instruction[0] == "MUL") {
        Inst I(rd, rs, rt, 0, MULcode);
        Instructions.push_back(I);

    }
    else if (instruction[0] == "DIV") {
        Inst I(rd, rs, rt, 0, DIVcode);
        Instructions.push_back(I);
    }
}
```

讀取輸入指令，分割後再將其宣告成**Inst**結構的變數放入**vector<Inst> Instructions**指令集

```
ReservationStation ADD(ADDcode, OperandInit), MUL(MULcode, OperandInit);
for (int i = 0; i < 3; i++) ResStations.push_back(ADD);
for (int i = 0; i < 2; i++)ResStations.push_back(MUL);
RegisterStatus F(RATempty);
for (int i = 0; i < 6; i++)RATarr.push_back(F);
```

初始化**ResStation**和**RATarr**

```
while (1) {
    isChange = false;
    cycleNum++;
    issue();
    execute();
    writeback();
    if (WRnums == Instructions.size())
        break;
    if (isChange) {
        cout << "Cycle " << cycleNum << ":\n" << endl;
        printRF();
        printRAT();
        printRS();
        cout << endl;
    }
}
for (int i = 0; i < 6; i++)
    RF[i] = Register[i];
cout << "Cycle " << cycleNum << ":\n" << endl;
printRF();
printRAT();
printRS();
```

每一次迴圈等於跑一次**cycle**來模擬**Tamasulo Algorithm**

b.

issue:

```
if (InstIssue >= Instructions.size())
    return;
```

先判別**Issue**數是否超出輸入的指令數

```
int order = Instructions[InstIssue].operation;
if (order == ADDcode || order == SUBcode || order == ADDIcode) {
    for (int i = 0; i < ADD_Nums; i++) {
        if (!ResStations[i].busy) {
            isChange = true;
            InstIssue++;
            RSfree = 1;
            if (order == ADDcode)
                ResStations[i].operation = ADDcode;
            else if (order == SUBcode)
                ResStations[i].operation = SUBcode;
            else if (order == ADDIcode)
                ResStations[i].operation = ADDIcode;
            order = i;
            break;
        }
    }
}
else {
    for (int i = ADD_Nums; i < ADD_Nums + MULandDIV_Nums; i++) {
        if (!ResStations[i].busy) {
            isChange = true;
            InstIssue++;
            if(order==MULcode)
                ResStations[i].operation = MULcode;
            else
                ResStations[i].operation = DIVcode;
            RSfree = 1;
            order = i;
            break;
        }
    }
}
```

再判斷進入**Issue**的指令的**operation**類型，再判斷有沒有空著的**RS**，再放入放入適當的**ReservationStation**

```
if (RATarr[Instructions[InstIssue - 1].rs].Qi == RATempty) {
    ResStations[order].Vj = Register[Instructions[InstIssue - 1].rs];
    ResStations[order].Qj = QperandAva;
    ResStations[order].unCompleteResultJ = -1;
}
else {
    ResStations[order].Qk = RATarr[Instructions[InstIssue - 1].rs].Qi;
    ResStations[order].unCompleteResultJ = RATarr[Instructions[InstIssue - 1].rs].Qi;
}
if (Instructions[InstIssue-1].operation == ADDIcode) {
    ResStations[order].Vk = Instructions[InstIssue - 1].constant;
    ResStations[order].Qk = QperandAva;
}
else if (RATarr[Instructions[InstIssue - 1].rt].Qi == RATempty) {
    ResStations[order].Vk = Register[Instructions[InstIssue - 1].rt];
    ResStations[order].Qk = QperandAva;
    ResStations[order].unCompleteResultK = -1;
}
else {
    ResStations[order].Qk = RATarr[Instructions[InstIssue - 1].rt].Qi;
    ResStations[order].unCompleteResultK = RATarr[Instructions[InstIssue - 1].rt].Qi;
}
ResStations[order].busy = true;
ResStations[order].IssueLatency = 0;
ResStations[order].InstNum = InstIssue-1;
RATarr[Instructions[InstIssue - 1].rd].Qi = order;
```

這邊用於判斷一條指令中所需的兩個數**rs**、**rt**是否已經運算完或可從**Register**中取得

c.execute:

```
for (int i = 0; i < ResStations.size(); i++) {
    if (ResStations[i].busy) {
        if (ResStations[i].IssueLatency >= ISSUElatency) {
            if (ResStations[i].Qj == QperandAva && ResStations[i].Qk == QperandAva) {
                bool exe = true;
                if (ResStations[i].operation < 3) {
                    if (buffer[0] == 50000)
                        buffer[0] = i;
                    else if (buffer[0] != i)
                        exe = false;
                }
                else {
                    if (buffer[1] == 50000)
                        buffer[1] = i;
                    else if (buffer[1] != i)
                        exe = false;
                }
```

先判斷該**ReservationStation**是否**busy**再判斷該**ReservationStation**的**Issue**時間有沒有經過一個**cycle**再判斷該**ReservationStation**裡的**Instruction**裡的**rs**、**rt**是否已經可取得，最後再判斷**BUFFER**裡是否有空位可以**execute**。

```
if (exe) {
    ResStations[i].latency++;
    switch (ResStations[i].operation) {
    case ADDcode:
        if (ResStations[i].latency == ADDlatency) {
            isChange = true;
            ResStations[i].result = ResStations[i].Vj + ResStations[i].Vk;
            ResStations[i].complete = true;
            ResStations[i].latency = 0;
            ResStations[i].IssueLatency = 0;
            buffer[0] = 50000;
            break;
        }
    case ADDIcode:
        if (ResStations[i].latency == ADDlatency) {
            isChange = true;
            ResStations[i].result = ResStations[i].Vj + ResStations[i].Vk;
            ResStations[i].complete = true;
            ResStations[i].latency = 0;
            ResStations[i].IssueLatency = 0;
            buffer[0] = 50000;
            break;
        }
    case SUBcode:
        if (ResStations[i].latency == ADDlatency) {
            isChange = true;
            ResStations[i].result = ResStations[i].Vj - ResStations[i].Vk;
            ResStations[i].complete = true;
            ResStations[i].latency = 0;
            ResStations[i].IssueLatency = 0;
            buffer[0] = 50000;
            break;
        }
    case MULcode:
        if (ResStations[i].latency == MULlatency) {
            isChange = true;
            ResStations[i].result = ResStations[i].Vj * ResStations[i].Vk;
            ResStations[i].complete = true;
            ResStations[i].latency = 0;
            ResStations[i].IssueLatency = 0;
            buffer[1] = 50000;
            break;
        }
    case DIVcode:
        if (ResStations[i].latency == DIVlatency) {
            isChange = true;
            ResStations[i].result = ResStations[i].Vj / ResStations[i].Vk;
            ResStations[i].complete = true;
            ResStations[i].latency = 0;
            ResStations[i].IssueLatency = 0;
            buffer[1] = 50000;
            break;
        }
    default:
        break;
    }
}
```

根據不同**operation**再跑完該種類**operation**所需的**latency**後根據**operation**的類型進行對應的運算式

d.writebcak:

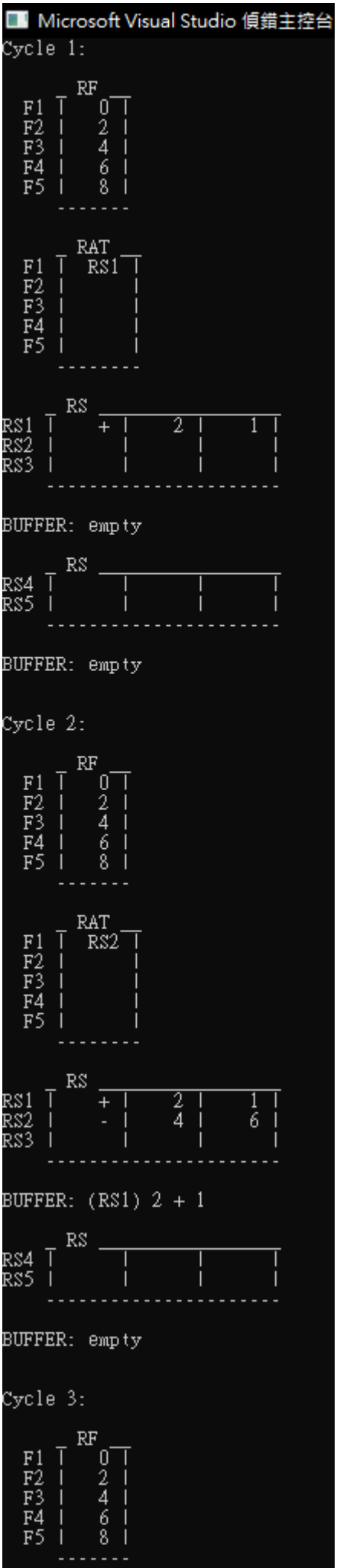
```
for (int i = 0; i < ResStations.size(); i++) {
    if (ResStations[i].complete) {
        if (ResStations[i].WriteBackLatency == WRITEBACKlatency) {
            isChange = true;
            for (int j = 0; j < Register.size(); j++) {
                if (RATarr[j].Qi == i)
                    Register[j] = ResStations[i].result, RATarr[j].Qi = RATempty;
            }
            for (int j = 0; j < ResStations.size(); j++) {
                if (ResStations[j].Qj == i)
                    ResStations[j].Vj = ResStations[i].result, ResStations[j].Qj = QperandAva;
                if (ResStations[j].Qk == i)
                    ResStations[j].Vk = ResStations[i].result, ResStations[j].Qk = QperandAva;
            }
            ResStations[i].busy = false;
            ResStations[i].complete = false;
            ResStations[i].Qj = QperandAva;
            ResStations[i].Qk = QperandAva;
            ResStations[i].Vj = 0;
            ResStations[i].Vk = 0;
            ResStations[i].latency = 0;
            WRnums++;
        }
        else
            ResStations[i].WriteBackLatency++;
    }
}
```

完成對應的運算再經過一個**cycle**後再將**result**寫入一開始指定的地點，再將該**ReservationStation**初始化空出來

4.Execute

將**.h**檔和**.cpp**檔放入專案再將想輸入的指令打在**input.txt**裡並儲存後執行即可

以下是一部分的輸出:


Microsoft Visual Studio 偵錯主控台
Cycle 1:
RF
F1 | 0 |
F2 | 2 |
F3 | 4 |
F4 | 6 |
F5 | 8 |
RAT
F1 | RS1 |
F2 |
F3 |
F4 |
F5 |
RS
RS1 | + | 2 | 1 |
RS2 |
RS3 |
BUFFER: empty
RS
RS4 |
RS5 |
BUFFER: empty
Cycle 2:
RF
F1 | 0 |
F2 | 2 |
F3 | 4 |
F4 | 6 |
F5 | 8 |
RAT
F1 | RS2 |
F2 |
F3 |
F4 |
F5 |
RS
RS1 | + | 2 | 1 |
RS2 | - | 4 | 6 |
RS3 |
BUFFER: (RS1) 2 + 1
RS
RS4 |
RS5 |
BUFFER: empty
Cycle 3:
RF
F1 | 0 |
F2 | 2 |
F3 | 4 |
F4 | 6 |
F5 | 8 |