

计算机体系结构 第三次实验报告——Tomasulo模拟器实现

本次实验中，参照教材附带模拟器和课堂幻灯片的动态演示，我自主设计了一款支持LOAD、STORE、ADD、SUB、MUL、DIV一共六种指令的Tomasulo模拟器，支持单步或多步执行、查看保留站内容、指令状态表、寄存器状态表、设置不同指令功能部件的执行时间、查看上一周期执行后的保留站内容、指令状态表、寄存器状态表。

代码和编译

源码如下所示：

```
#include<iostream>
#include<vector>
#include<string.h>
#include<stdio.h>
using namespace std;

enum state {unready = 4, issue = 0, execmpl = 1, writebk = 2};

struct RS{
    char * name;
    int time;
    bool busy;
    char * op;
    char * Qj, * Qk;
    char * Vj, * Vk;
    int inst_index;
    RS(char * Sname, int Stime, bool Sbusy, char * Sop, char * SQj, char * SQk,
    char * SVj, char * SVk, int Sinst_index=0):
        name(Sname), time(Stime), busy(Sbusy), op(Sop), Qj(SQj), Qk(SQk),
        Vj(SVj), Vk(SVk), inst_index(Sinst_index){}
};

struct Load{
    char * name;
    int time;
    bool busy;
    char * address;
    Load(char * Sname, int Stime, bool Sbusy, char * Saddress):
        name(Sname), time(Stime), busy(Sbusy), address(Saddress){}
};

struct inst{
    char * op;
    int rs, rt, rd;
    int state;
    int RS_index;
    inst(char * opcode, int regd, int regs, int regt, int prev_state=unready, int
MRS_index=0):
        op(opcode), rs(regs), rt(regt), rd(regd), state(prev_state),
        RS_index(MRS_index){}
```

```

};

vector<inst> insts;
vector<RS> RSs;
vector<Load> Loads;
char * regstatus[30];
char * regvalue[30];

int inst_state[200][3];
int my_clock = 0;

int exec_time[6] = {2, 2, 10, 2, 2, 40};
int reg_count = 20;

char *functs[9] = {"Load1", "Load2", "Load3", "Add1", "Add2", "Add3", "Mult1",
"Mult2", ""};
char *opCode[7] = {"LOAD", "STORE", "MUL", "SUB", "ADD", "DIV", ""};
char *tmp_buffer = new char[1000];
char *reg_buffer = new char[1000];
int buffer_ptr = 0;
int reg_ptr = 0;

char * get_regvalue(int idx){
    if(regvalue[idx] == nullptr){
        sprintf(reg_buffer + reg_ptr, "%-2d", idx);
        // printf("%s-----", reg_buffer+reg_ptr);
        char * ret = reg_buffer + reg_ptr;
        reg_ptr += 4;
        return ret;
    }
    else{
        return regvalue[idx];
    }
}

// 将保留站、LoadBuffer、寄存器状态、指令状态拷贝一份，需要在next()进入的时候调用
// 保存上次状态的数据结构：
vector<RS> RSs_old;
vector<Load> Loads_old;
char * regstatus_old[30];
int inst_state_old[200][3];

void save_old(){
    RSs_old.clear();
    Loads_old.clear();
    for(int i = 0; i < RSs.size(); i++){
        RSs_old.push_back(RS(RSs[i].name, RSs[i].time, RSs[i].busy, RSs[i].op,
RSs[i].Qj, RSs[i].Qk, RSs[i].Vj, RSs[i].Vk, RSs[i].inst_idx));
    }
    for(int i = 0; i < Loads.size(); i++){
        Loads_old.push_back(Load(Loads[i].name, Loads[i].time, Loads[i].busy,
Loads[i].address));
    }
    for(int i = 0; i < 30; i++){
        regstatus_old[i] = regstatus[i];
    }
}

```



```

        if(cur_RS.Qj == nullptr && cur_RS.Qk == nullptr){
            if(cur.op == opCode[2]) cur_RS.time = exec_time[2];
            else cur_RS.time = exec_time[5];
        }
        regstatus[cur.rd] = cur_RS.name;
        issue_flag = true;
        break;
    }
}

else if(cur.op == opCode[3] || cur.op == opCode[4]){ // SUB, ADD
    for(int j = 0; j < RSs.size(); j++){
        auto & cur_RS = RSs[j];
        if(!cur_RS.busy && (cur_RS.name == functs[3] || cur_RS.name
== functs[4] || cur_RS.name == functs[5]) && !RS_hit[j]){
            RS_hit[j] = true;
            inst_state[i][0] = my_clock;
            cur_RS_index = j;
            cur.state = issue;
            cur_RS.busy = true;
            cur_RS.op = cur.op;
            cur_RS.inst_idx = i;
            if(regstatus[cur.rs] == functs[8])
                cur_RS.Vj = get_regvalue(cur.rs);
            else
                cur_RS.Qj = regstatus[cur.rs];
            if(regstatus[cur.rt] == functs[8])
                cur_RS.Vk = get_regvalue(cur.rt);
            else
                cur_RS.Qk = regstatus[cur.rt];
            if(cur_RS.Qj == nullptr && cur_RS.Qk == nullptr){
                if(cur.op == opCode[3]) cur_RS.time = exec_time[3];
                else cur_RS.time = exec_time[4];
            }
            regstatus[cur.rd] = cur_RS.name;
            issue_flag = true;
            break;
        }
    }
}

if(cur.state == issue){ // 尝试执行
    if(!Load_hit[cur_RS_index] && (cur.op == opCode[0] || cur.op ==
opCode[1])){ // Load or Store
        auto & cur_load = Loads[cur_RS_index];
        Load_hit[cur_RS_index] = true;
        cur_load.time -= 1;
        if(cur_load.time == 0){
            cur.state = execmpl;
            inst_state[i][1] = my_clock;
        }
    }
    else if(!RS_hit[cur_RS_index] && (cur.op == opCode[2] || cur.op ==
opCode[3] || cur.op == opCode[4] || cur.op == opCode[5])){ // MUL, DIV, SUB, ADD
        auto & cur_RS = RSs[cur_RS_index];
        if(cur_RS.Qj == nullptr && cur_RS.Qk == nullptr){

```

```

        RS_hit[cur.RS_index] = true;
        cur_RS.time -= 1;
        if(cur_RS.time == 0){
            cur.state = execmpl;
            inst_state[i][1] = my_clock;
        }
    }
}

if(cur.state == execmpl){ // 尝试写回
    if(!Load_hit[cur.RS_index] && (cur.op == opCode[0] || cur.op ==
opCode[1])){ // Load or Store
        auto & cur_load = Loads[cur.RS_index];
        Load_hit[cur.RS_index] = true;
        // 写入REG和需要这个数字的保留站
        sprintf(tmp_buffer + buffer_ptr, "M[%s]", cur_load.address);
        regvalue[cur.rd] = tmp_buffer + buffer_ptr;
        for (int j = 0; j < RSs.size(); j++){
            auto & cur_RS = RSs[j];
            if(cur_RS.Qj == cur_load.name){
                cur_RS.Vj = regvalue[cur.rd];
                cur_RS.Qj = nullptr;
                RS_hit[j] = true;
                if (cur_RS.op == opCode[2]) cur_RS.time = exec_time[2];
                else if (cur_RS.op == opCode[5]) cur_RS.time =
exec_time[5];

                else if (cur_RS.op == opCode[3]) cur_RS.time =
exec_time[3];

                else if (cur_RS.op == opCode[4]) cur_RS.time =
exec_time[4];
            }
            if(cur_RS.Qk == cur_load.name){
                cur_RS.Vk = regvalue[cur.rd];
                cur_RS.Qk = nullptr;
                RS_hit[j] = true;
                if (cur_RS.op == opCode[2]) cur_RS.time = exec_time[2];
                else if (cur_RS.op == opCode[5]) cur_RS.time =
exec_time[5];

                else if (cur_RS.op == opCode[3]) cur_RS.time =
exec_time[3];

                else if (cur_RS.op == opCode[4]) cur_RS.time =
exec_time[4];
            }
        }
        buffer_ptr += strlen(cur_load.address) + 4;
        regstatus[cur.rd] = functs[8];
        cur_load.busy = false;
        cur_load.address = nullptr;
        cur.state = writebk;
        inst_state[i][2] = my_clock;
    }
    else if(!RS_hit[cur.RS_index] && (cur.op == opCode[2] || cur.op ==
opCode[3] || cur.op == opCode[4] || cur.op == opCode[5])){ // MUL, DIV, SUB, ADD
        auto & cur_RS = RSs[cur.RS_index];
        RS_hit[cur.RS_index] = true;
        // 写入REG和需要这个数字的保留站

```

```

        if(cur_RS.op == opCode[2]) sprintf(tmp_buffer + buffer_ptr,
"%s*%s", cur_RS.Vj, cur_RS.Vk);
        else if(cur_RS.op == opCode[3]) sprintf(tmp_buffer + buffer_ptr,
"%s-%s", cur_RS.Vj, cur_RS.Vk);
        else if(cur_RS.op == opCode[4]) sprintf(tmp_buffer + buffer_ptr,
"%s+%s", cur_RS.Vj, cur_RS.Vk);
        else if(cur_RS.op == opCode[5]) sprintf(tmp_buffer + buffer_ptr,
"%s/%s", cur_RS.Vj, cur_RS.Vk);
        regvalue[cur.rd] = tmp_buffer + buffer_ptr;
        for (int j = 0; j < RSs.size(); j++){
            auto & cur_RS_j = RSs[j];
            if(cur_RS_j.Qj == cur_RS.name){
                cur_RS_j.Vj = regvalue[cur.rd];
                cur_RS_j.Qj = nullptr;
                RS_hit[j] = true;
                if(cur_RS_j.op == opCode[2]) cur_RS_j.time =
exec_time[2];
                else if(cur_RS_j.op == opCode[5]) cur_RS_j.time =
exec_time[5];
                else if(cur_RS_j.op == opCode[3]) cur_RS_j.time =
exec_time[3];
                else if(cur_RS_j.op == opCode[4]) cur_RS_j.time =
exec_time[4];
            }
            if(cur_RS_j.Qk == cur_RS.name){
                cur_RS_j.Vk = regvalue[cur.rd];
                cur_RS_j.Qk = nullptr;
                RS_hit[j] = true;
                if(cur_RS_j.op == opCode[2]) cur_RS_j.time =
exec_time[2];
                else if(cur_RS_j.op == opCode[5]) cur_RS_j.time =
exec_time[5];
                else if(cur_RS_j.op == opCode[3]) cur_RS_j.time =
exec_time[3];
                else if(cur_RS_j.op == opCode[4]) cur_RS_j.time =
exec_time[4];
            }
        }
        buffer_ptr += strlen(cur_RS.Vj) + strlen(cur_RS.Vk) + 2;
        regstatus[cur.rd] = functs[8];
        cur_RS.busy = false;
        cur_RS.Vj = nullptr;
        cur_RS.Vk = nullptr;
        cur_RS.op = opCode[6];
        cur.state = writebk;
        inst_state[i][2] = my_clock;
    }
}
}
bool ret = false;
for(int i = 0; i < insts.size(); i++){
    if(insts[i].state != writebk){
        ret = true;
        break;
    }
}
}

```

```

        return ret;
    }

    // 支持old参数的print函数，用于打印上一次的状态

    void printRS(bool old=false){
        if(old){
            printf("Time Name      Busy    Op      Qj      Qk      Vj\n");
            vk\n");
            for(int i = 0; i < RSs_old.size(); i++){
                printf("%-5d", RSs_old[i].time);
                printf("%-10s", RSs_old[i].name);
                if(RSs_old[i].busy) printf("Yes    ");
                else printf("No      ");
                printf("%-8s ", RSs_old[i].op);
                if(RSs_old[i].Qj == nullptr) printf("          ");
                else printf("%-7s ", RSs_old[i].Qj);
                if(RSs_old[i].Qk == nullptr) printf("          ");
                else printf("%-7s ", RSs_old[i].Qk);
                if(RSs_old[i].Vj == nullptr) printf("          ");
                else printf("%-19s ", RSs_old[i].Vj);
                if(RSs_old[i].Vk == nullptr) printf("          ");
                else printf("%-19s ", RSs_old[i].Vk);
                puts("");
            }
        }
        else{
            printf("Time Name      Busy    Op      Qj      Qk      Vj\n");
            vk\n");
            for(int i = 0; i < RSs.size(); i++){
                printf("%-5d", RSs[i].time);
                printf("%-10s", RSs[i].name);
                if(RSs[i].busy) printf("Yes    ");
                else printf("No      ");
                printf("%-8s ", RSs[i].op);
                if(RSs[i].Qj == nullptr) printf("          ");
                else printf("%-7s ", RSs[i].Qj);
                if(RSs[i].Qk == nullptr) printf("          ");
                else printf("%-7s ", RSs[i].Qk);
                if(RSs[i].Vj == nullptr) printf("          ");
                else printf("%-19s ", RSs[i].Vj);
                if(RSs[i].Vk == nullptr) printf("          ");
                else printf("%-19s ", RSs[i].Vk);
                puts("");
            }
        }
    }

    void printLoad(bool old=false){
        if(old){
            printf("Time Name      Busy    Address\n");
            for(int i = 0; i < Loads_old.size(); i++){
                printf("%-5d", Loads_old[i].time);
                printf("%-10s", Loads_old[i].name);
                if(Loads_old[i].busy) printf("Yes ");
                else printf("No  ");
            }
        }
    }
}

```

```

        if(Loads_old[i].address == nullptr) printf("          ");
        else printf("%9s ", Loads_old[i].address);
        puts("");
    }
}
else{
    printf("Time Name      Busy   Address\n");
    for(int i = 0; i < Loads.size(); i++){
        printf("%-5d", Loads[i].time);
        printf("%-10s", Loads[i].name);
        if(Loads[i].busy) printf("Yes ");
        else printf("No  ");
        if(Loads[i].address == nullptr) printf("          ");
        else printf("%9s ", Loads[i].address);
        puts("");
    }
}
}

void printInst(bool old=false){
    if(old){
        for(int i = 0; i < insts.size(); i++){
            printf("%-6s", insts[i].op);
            if(insts[i].op == "LOAD"){
                printf("F%-2d ", insts[i].rd);
                printf("%-3d ", insts[i].rs);
                printf("R%-2d   ", insts[i].rt);
                for(int k = 0; k < 3; k++){
                    printf("%3d", inst_state_old[i][k]);
                }
                puts("");
            }
            else{
                printf("F%-2d ", insts[i].rd);
                printf("F%-2d ", insts[i].rs);
                printf("F%-2d   ", insts[i].rt);
                for(int k = 0; k < 3; k++){
                    printf("%3d", inst_state_old[i][k]);
                }
                puts("");
            }
        }
    }
    else{
        for(int i = 0; i < insts.size(); i++){
            printf("%-6s", insts[i].op);
            if(insts[i].op == "LOAD"){
                printf("F%-2d ", insts[i].rd);
                printf("%-3d ", insts[i].rs);
                printf("R%-2d   ", insts[i].rt);
                for(int k = 0; k < 3; k++){
                    printf("%3d", inst_state[i][k]);
                }
                puts("");
            }
            else{

```



```

        printf("F%-2d ", insts[i].rd);
        printf("F%-2d ", insts[i].rs);
        printf("F%-2d ", insts[i].rt);
        for(int k = 0; k < 3; k++){
            printf("%3d", inst_state[i][k]);
        }
        puts("");
    }
}

}

}

void printRegs(bool old=false){
    if(old){
        for(int i = 0; i < reg_count; i+=2){
            printf("F%-10d", i);
        }
        puts("");
        for(int i = 0; i < reg_count; i+=2){
            printf("%-10s ", regstatus_old[i]);
        }
        puts("");
    }
    else{
        for(int i = 0; i < reg_count; i+=2){
            printf("F%-10d", i);
        }
        puts("");
        for(int i = 0; i < reg_count; i+=2){
            printf("%-10s ", regstatus[i]);
        }
        puts("");
    }
}

void printmy(bool old=false){
    if(old){
        printInst(true);
        puts("");
        printRS(true);
        puts("");
        printLoad(true);
        puts("");
        printRegs(true);
        puts("");
    }
    else{
        printInst();
        puts("");
        printRS();
        puts("");
        printLoad();
        puts("");
        printRegs();
        puts("");
    }
}

```

```

}

int main(){
    for(int i = 0; i < 30; i++){
        // regstatus[i] = nullptr;
        regstatus[i] = functs[8];
        regvalue[i] = nullptr;
    }
    Loads.push_back(Load(functs[0], 0, false, nullptr));
    Loads.push_back(Load(functs[1], 0, false, nullptr));
    Loads.push_back(Load(functs[2], 0, false, nullptr));
    RSs.push_back(RS(functs[3], 0, false, opCode[6], nullptr, nullptr, nullptr,
    nullptr));
    RSs.push_back(RS(functs[4], 0, false, opCode[6], nullptr, nullptr, nullptr,
    nullptr));
    RSs.push_back(RS(functs[5], 0, false, opCode[6], nullptr, nullptr, nullptr,
    nullptr));
    RSs.push_back(RS(functs[6], 0, false, opCode[6], nullptr, nullptr, nullptr,
    nullptr));
    RSs.push_back(RS(functs[7], 0, false, opCode[6], nullptr, nullptr, nullptr,
    nullptr));
    // 没有任何冲突的演示:
    insts.push_back(inst(opCode[0], 2, 21, 3));
    insts.push_back(inst(opCode[0], 4, 16, 4));
    insts.push_back(inst(opCode[4], 6, 8, 10));
    insts.push_back(inst(opCode[3], 12, 14, 16));
    int ne = 1;
    int chk = 0;
    bool live = true;
    while(ne){
        while(ne--){
            my_clock += 1;
            live = next();
        }
        cout<<"Clock: "<<my_clock<<endl;
        printmy(0);
        if(!live){
            cout<<"All instructions have been executed."<<endl;
            break;
        }
        cout<<"want to check last status? 1 for yes, 0 for no: ";
        cin>>chk;
        if(chk){
            printmy(1);
        }
        cout<<"1 for 1 cycle, n for n cycles: ";
        cin>>ne;
    }
    return 0;
}

```

编译运行的方法是：

在linux或Windows环境下，使用支持C++11或以上的编译器编译文件，执行所得到的可执行文件，即可运行程序。

模拟器测试要执行的程序需要在main函数中直接输入，程序运行起来后，需要在询问执行周期数时输入要一次性执行的周期，若输入1就是要执行一个周期，即单步执行，若输入大于1的整数即执行多个周期，即多步执行，输入0退出程序。

设计思路

在核心的单步执行代码void next()中，我按照要执行的代码的顺序遍历指令，对每个指令检查他的状态，查看是否可以进行下一步操作，例如当前状态是未流出时，存储指令的数据结构中的state设置为0，枚举类型映射为unready；当流出后，设置为1，映射为issue；当执行完成之后，state设置为2，代表execmpl；写回步骤完成之后，设置为3，代表writebk。在遍历到每个指令时，需要看它能否进行下一步操作，操作的细节和动作按照课本的描述进行：当要尝试流出时，查看是否有空的保留站，如果存在空闲的保留站，就将指令流出，流出后，检查操作数是否就绪，如果就绪，将操作数的值读入到保留站，若未就绪，将产生操作数的保留站编号读入；当两个操作数就绪，即可在下一周期开始执行；执行完成后，尝试写回，写回时需要将得到的数放到CDB上，即将数据送入所有需要的地方，包括寄存器或者保留站或者内存空间。

实现时的技巧包括：设计时采用了字符串指针来指示存储表示保留站内容的字符数组的位置，使用Vector等工具存储指令和保留站结构。

没有任何冲突的演示

第三个周期执行结束的场景：

```
Clock: 3
LOAD  F2  21  R3      1  3  0
LOAD  F4  16  R4      2  0  0
ADD   F6  F8  F10     3  0  0
SUB   F12 F14 F16     0  0  0

Time Name    Busy   Op      Qj      Qk      Vj      Vk
2    Add1     Yes    ADD     Qj      Qk      Vj      Vk
0    Add2     No
0    Add3     No
0    Mult1    No
0    Mult2    No

Time Name    Busy   Address
0    Load1   Yes    21+R3
1    Load2   Yes    16+R4
0    Load3   No

F0      F2      F4      F6      F8      F10     F12     F14     F16     F18
      Load1   Load2   Add1

1 for 1 cycle, n for n cycles: █
```

第五个周期执行结束的场景：

```
Clock: 5
LOAD  F2  21  R3      1  3  4
LOAD  F4  16  R4      2  4  5
ADD   F6  F8  F10     3  5  0
SUB   F12 F14 F16     4  0  0

Time Name      Busy  Op      Qj      Qk      Vj      Vk
0    Add1      Yes  ADD      Qj      Qk      F8      F10
1    Add2      Yes  SUB      Qk      Vj      F14     F16
0    Add3      No
0    Mult1     No
0    Mult2     No

Time Name      Busy  Address
0    Load1    No
0    Load2    No
0    Load3    No

F0      F2      F4      F6      F8      F10     F12     F14     F16     F18
          Add1          Add2

1 for 1 cycle, n for n cycles: █
```

第七个周期执行结束的场景：

```
Clock: 7
LOAD  F2  21  R3      1  3  4
LOAD  F4  16  R4      2  4  5
ADD   F6  F8  F10     3  5  6
SUB   F12 F14 F16     4  6  7

Time Name      Busy  Op      Qj      Qk      Vj      Vk
0    Add1      No
0    Add2      No
0    Add3      No
0    Mult1     No
0    Mult2     No

Time Name      Busy  Address
0    Load1    No
0    Load2    No
0    Load3    No

F0      F2      F4      F6      F8      F10     F12     F14     F16     F18

1 for 1 cycle, n for n cycles: █
```

没有冲突时，指令执行不会前后相互等待，Vj和Vk能够立即读入到保留站，不会存在读入其他保留站编号的情况。

包含RAW冲突：

下图中的第二条指令LOAD与第三条指令ADD关于寄存器F2存在RAW冲突、第三条指令ADD与第四条指令SUB关于寄存器F0存在RAW冲突。

第4个周期执行完：

Clock: 4									
LOAD	F6	34	R2	1	3	4			
LOAD	F2	45	R3	2	4	0			
ADD	F0	F2	F4	3	0	0			
SUB	F8	F6	F0	4	0	0			
Time	Name	Busy	Op	Qj	Qk	Vj		Vk	
0	Add1	Yes	ADD	Load2				F4	
0	Add2	Yes	SUB		Add1	M[34+R2]			
0	Add3	No							
0	Mult1	No							
0	Mult2	No							
Time	Name	Busy	Address						
0	Load1	No							
0	Load2	Yes	45+R3						
0	Load3	No							
F0		F2	F4	F6	F8	F10	F12	F14	F16
Add1		Load2			Add2				

第5个周期执行完：

Clock: 5									
LOAD	F6	34	R2	1	3	4			
LOAD	F2	45	R3	2	4	5			
ADD	F0	F2	F4	3	0	0			
SUB	F8	F6	F0	4	0	0			
Time	Name	Busy	Op	Qj	Qk	Vj		Vk	
2	Add1	Yes	ADD			M[45+R3]		F4	
0	Add2	Yes	SUB		Add1	M[34+R2]			
0	Add3	No							
0	Mult1	No							
0	Mult2	No							
Time	Name	Busy	Address						
0	Load1	No							
0	Load2	No							
0	Load3	No							
F0		F2	F4	F6	F8	F10	F12	F14	F16
Add1					Add2				

第8个周期执行完：

Clock: 8									
LOAD	F6	34	R2	1	3	4			
LOAD	F2	45	R3	2	4	5			
ADD	F0	F2	F4	3	7	8			
SUB	F8	F6	F0	4	0	0			
Time	Name	Busy	Op	Qj	Qk	Vj		Vk	
0	Add1	No							
2	Add2	Yes	SUB			M[34+R2]		M[45+R3]+F4	
0	Add3	No							
0	Mult1	No							
0	Mult2	No							
Time	Name	Busy	Address						
0	Load1	No							
0	Load2	No							
0	Load3	No							
F0		F2	F4	F6	F8	F10	F12	F14	F16
					Add2				

第11个周期执行结束：

Clock: 11									
LOAD	F6	34	R2	1	3	4			
LOAD	F2	45	R3	2	4	5			
ADD	F0	F2	F4	3	7	8			
SUB	F8	F6	F0	4	10	11			
Time	Name	Busy	Op	Qj	Qk	Vj		Vk	
0	Add1	No							
0	Add2	No							
0	Add3	No							
0	Mult1	No							
0	Mult2	No							
Time	Name	Busy	Address						
0	Load1	No							
0	Load2	No							
0	Load3	No							
F0	F2	F4	F6	F8	F10	F12	F14	F16	F18

Tomasulo算法解决写后读冲突的方法是将能产生（写）未准备好的操作数的保留站编号存入要读的指令保留站中，当上一个写指令完成后，将数据同时写入寄存器和需要它的保留站。这能够减少指令由于无法读操作数而不断地从寄存器尝试读取数据的次数，减轻了寄存器组的数据访问压力。CDB是增加的数据通路，能够加速RAW冲突的解决速度，读取操作数的指令能够再最快的时间内获得数据。这样的时间上的高效性是数据通路增加、控制逻辑变复杂所换来的。

包含WAR冲突：

下图中的第三条指令DIV与第四条指令ADD关于寄存器F8存在WAR冲突，而由于DIV与之前的MUL存在RAW，MUL执行速度特别慢，而流出条件是只要有空位就流出，ADD预计在第4个周期流出，在第7个周期完成，就会写入F8，那么DIV会不会受到影响呢？

第1个周期执行完：

Clock: 1									
LOAD	F10	21	R3	1	0	0			
MUL	F2	F16	F4	0	0	0			
DIV	F12	F2	F8	0	0	0			
ADD	F8	F10	F4	0	0	0			
Time	Name	Busy	Op	Qj	Qk	Vj		Vk	
0	Add1	No							
0	Add2	No							
0	Add3	No							
0	Mult1	No							
0	Mult2	No							
Time	Name	Busy	Address						
2	Load1	Yes	21+R3						
0	Load2	No							
0	Load3	No							
F0	F2	F4	F6	F8	F10	F12	F14	F16	F18
					Load1				

第4个周期末：

ADD已流出，但是DIV的操作数还没有就绪，因为MUL较慢。

实验总结：

Tomasulo算法通过**改名技术解决了WAR冲突**，通过增加CDB和保留站结构**最小化了RAW带来的性能损耗**，只考虑数据冲突情况下的指令并行技术，Tomasulo到达了很优秀的程度。