实验 4：记分板调度方法仿真

背景知识

Scoreboarding is a centralized method, used in the CDC 6600 computer, for dynamically scheduling a pipeline so that the instructions can execute out of order when there are no conflicts and the hardware is available. In a scoreboard, the data dependencies of every instruction are logged. Instructions are released only when the scoreboard determines that there are not conflicts with previously issued and incomplete instructions. If an instruction is stalled because it is unsafe to continue, the scoreboard monitors the flow of executing instructions until all dependencies have been resolved before the stalled instruction is issued.

1. Stages

Instructions are decoded in order and go through the following four stages.

1. **Issue:** The system checks which register will be read and written by this instruction. This information is remembered as it will be needed in the following stages. In order to avoid output dependencies (WAW – Write after Write) the instruction is stalled until instructions intending to write to the same register are completed. The instruction is also stalled when required functional units are currently busy.
2. **Read operands:** After an instruction has been issued and correctly allocated to the required hardware module, the instruction waits until all operands become available. This procedure resolves read dependencies (RAW – Read after Write) because registers which are intended to be written by another instruction are not considered available until they are actually written.
3. **Execution:** When all operands have been fetched, the functional unit starts its execution. After the result is ready, the scoreboard is notified.
4. **Write Result:** In this stage the result is about to be written to its destination register. However, this operation is delayed until earlier instructions – which intend to read registers this instruction wants to write to – have completed their read operands stage. This way, so called data dependencies (WAR – Write after Read) can be addressed.

2. Data structure

To control the execution of the instructions, the scoreboard maintains three status tables:

- **Instruction Status:** Indicates, for each instruction being executed, which of the four stages it is in.
- **Functional Unit Status:** Indicates the state of each functional unit. Each functional unit maintains 9 fields in the table:

- Busy: Indicates whether the unit is being used or not
- Op: Operation to perform in the unit (e.g. MUL, DIV or MOD)
- F_i: Destination register
- F_j, F_k: Source-register numbers
- Q_j, Q_k: Functional units that will produce the source register F_j, F_k
- R_j, R_k: Flags that indicates when F_j, F_k are ready.

- **Register Status:** Indicates, for each register, which functional unit will writes results into it.

3. The algorithm

The detailed algorithm for the scoreboard control is described below:

```
Function issue(op, dst, src1, src2)
    Wait until (!Busy[FU] AND !Result[dst]); // FU can be any functional unit that can execute operation op
    Busy[FU] ← Yes;
    Op[FU] ← op;
    Fi[FU] ← dst;
    Fj[FU] ← src1;
    Fk[FU] ← src2;
    Qj[FU] ← Result[src1];
    Qk[FU] ← Result[src2];
    Rj[FU] ← not Qj;
    Rk[FU] ← not Qk;
    Result[dst] ← FU;
```

```
Function read_operands(FU)
    Wait until (Rj[FU] AND Rk[FU]);
    Rj[FU] ← No;
    Rk[FU] ← No;
```

```
Function execute(FU)
    // Execute whatever FU must do
```

```
Function write_back(FU)
    Wait until $\left(\forall f\left\{\left(F_j[f] \neq F_i[FU]\ \text{OR}\ R_j[f] = \text{No}\right)\ \text{AND}\ \left(F_k[f] \neq F_i[FU]\ \text{OR}\ R_k[f] = \text{No}\right)\right\}\right)$
    for each f do
        if  Qj[f]=FU  then  Rj[f] ← Yes;
        if  Qk[f]=FU  then  Rk[f] ← Yes;
    Result[Fi[FU]] ← 0;
    Busy[FU] ← No;
```

4. Remarks

The scoreboarding method must stall the issue stage when there is no functional unit available. In this case, future instructions that could potentially be executed will wait until the structural hazard is resolved. Some other techniques like Tomasulo algorithm can avoid the structural hazard and also resolve WAR and WAW dependencies with Register renaming.

实验目的

- 了解动态调度的基本思路
- 了解记分板技术的基本原理和特点
- 掌握记分板技术的实现框架和运行机制
- 掌握记分板技术中指令乱序执行的特点

实验内容

1. 定义 NK-CPU 指令流水线记分板仿真的基本数据结构
2. 使用 C 语言实现 NK-CPU 指令流水线记分板仿真程序
3. 使用 NK-CPU 汇编语言编写记分板仿真程序的测试程序
4. 获得测试结果并与实验 3 仿真测试结果进行比较

实验要求

1. 同实验 3
2. 同实验 3
3. 指令流水线的功能单元包括 1 个 ALU、2 个浮点乘法器、1 个浮点加法器和 1 个浮点除法器。