系统结构第二次实验：设计实现 Tomasulo 调度算法

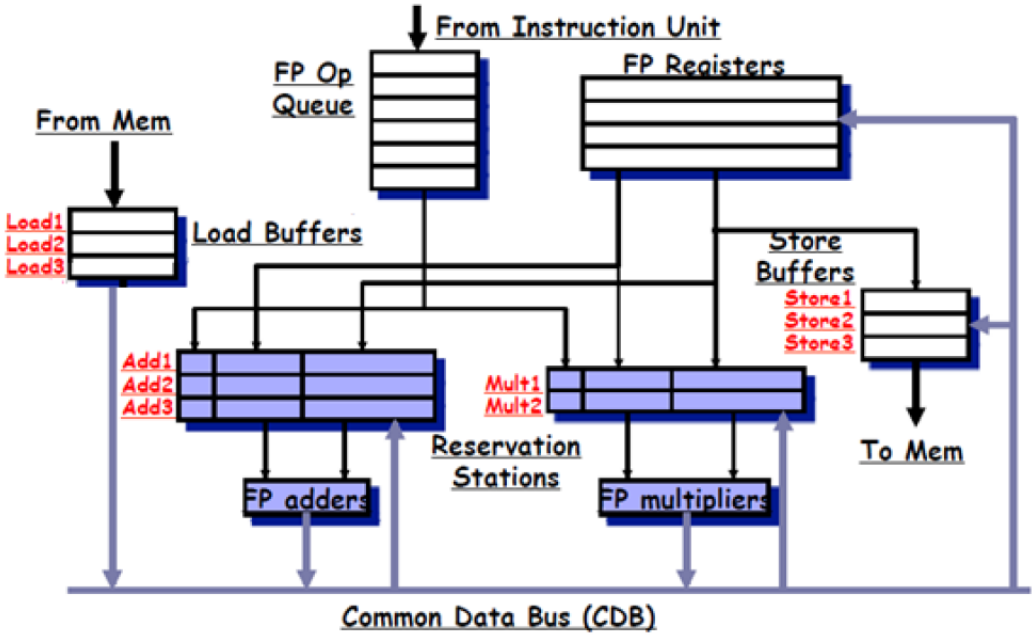
计 43 兰敏琪 2014011360

计 43 唐玉涵 2014011328

1. 实验原理

Tomasulo 算法以硬件方式实现了寄存器重命名，允许指令乱序执行，是提高流水线的吞吐率和效率的一种有效方式。该算法首先出现在 IBM360/91 处理机的浮点处理部件中，后广泛应用于现代处理器设计中。

假设浮点处理部件结构如下图所示。浮点处理部件从取指单元接收指令，存入浮点操作队列。浮点操作队列每拍最多发射 1 条指令给浮点加法器或浮点乘除法器。浮点处理部件包含一个浮点加法器和一个浮点乘除法器。浮点加法器为两段流水线，输入端有三个保留站 A1、A2、A3，浮点乘除法器为六段流水线，输入端有两个保留站 M1，M2。当任意一个保留站中的两个源操作数到齐后，如果对应的操作部件空闲，可以把两个操作数立即送到浮点操作部件中执行。Load Buffer 和 Store Buffer 各缓存三条访存操作。



2. 模拟器平台

用浏览器即可查看，通过 HTML+CSS+JS 实现，使用的库有 jQuery、Bootstrap（已内置）。

3. 模拟器功能

能执行浮点的加减乘除运算及 LOAD、STORE 操作。

操作具体描述如下：

指令格式	指令说明	指令周期	保留站/缓冲队列项数
ADDD F1,F2,F3	F1, F2, F3 为浮点寄存器寄存器至少支持（F0~F10）	2 个周期	3
SUBD F1, F2, F3	同上	2 个周期	
MULD F1, F2, F3	同上	10 个周期	2
DIVD F1, F2, F3	同上	40 个周期	
LD F1, ADDR	F1 为寄存器，ADDR 为地址，0<=ADDR<4096	2 个周期	3
ST F1, ADDR	同上	2 个周期	3

支持单步执行、多步执行、实时显示算法的运行状况；

显示指令执行的周期数及当前 PC 指向那条指令；

能够根据内存地址（0-4095）来实时查看与编辑内存值；

指令序列可通过文本框输入；

具体界面、功能介绍如下：

为了用户使用的便利，我们将主要功能都集中在一个页面上，并将页面分为了导航条、工作区域和底部版权信息三部分。工作区域按照功能划分为操作区、指令队列区、保留站区、浮点寄存器区和内存区五个功能区。整体界面风格简洁明快，直观大气，在完成基本显示和交互功能的前提下注重美观性与易用性的结合，并符合现代网页设计潮流与趋势。

初始界面如下图所示。

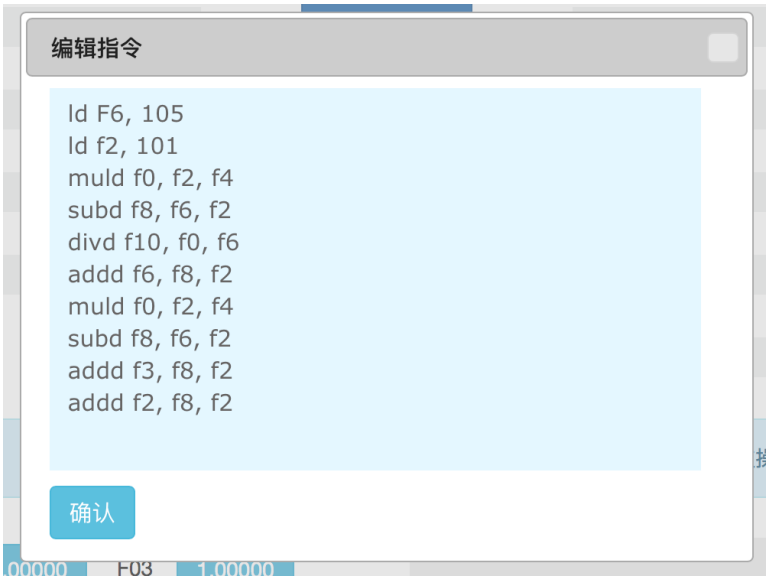
界面包括以下几块区域

1) 操作区

操作区分为两部分，上方为周期和 PC 的数值显示，下方为六个基本操作：单步执行、多步执行（弹框见下图）、开始运行、暂停运行、运行结果和清除数据（归零）。

2) 指令队列区

此部分具有编辑指令和显示指令运行状态两个功能，点击“编辑指令”处会弹出弹框（见下图），可在此处进行增删以及修改指令的操作，编辑



完成后点击确认即可回到主页面。编辑完成的指令即显示到下方的表格之中，表格中数据根据实际运行状态进行变化。

指令队列（共10条） 编辑指令

行号	指令	读取时刻	完成时刻	写回时刻
1	LD F6, 105	1	4	5
2	LD F2, 101	2	5	6
3	MULD F0, F2, F4	3	16	17
4	SUBD F8, F6, F2	4	8	9
5	DIVD F10, F0, F6	5		
6	ADDD F6, F8, F2	6	11	12
7	MULD F0, F2, F4	18		
8	SUBD F8, F6, F2	19	21	
9	ADDD F3, F8, F2	20		
10	ADDD F2, F8, F2	21		

3) 保留站区

此区域包括 Load Buffer 和 Store Buffer，将保留站和 Buffer 的各项状态以表格的形式呈现出来，包括剩余执行周期、对应指令行号、当前状态、输入和输出。

保留站（包含 Load Buffer 和 Store Buffer）

名称	剩余周期	指令行号	状态	输出	输入1	输入2
ADD_1	2	[4]	ISSUE	F8	105	LOAD_2
ADD_2			IDLE			
ADD_3			IDLE			
MUL_1	10	[3]	ISSUE	F0	LOAD_2	1
MUL_2	40	[5]	ISSUE	F10	MUL_1	105
LOAD_1			IDLE			
LOAD_2	0	[2]	WRITE	F2	101	-
LOAD_3			IDLE			
STORE_1			IDLE			
STORE_2			IDLE			
STORE_3			IDLE			

4) 浮点寄存器区

此区域将 F00~F10 这 11 个浮点寄存器内部的数值实时显示在页面上。

浮点寄存器

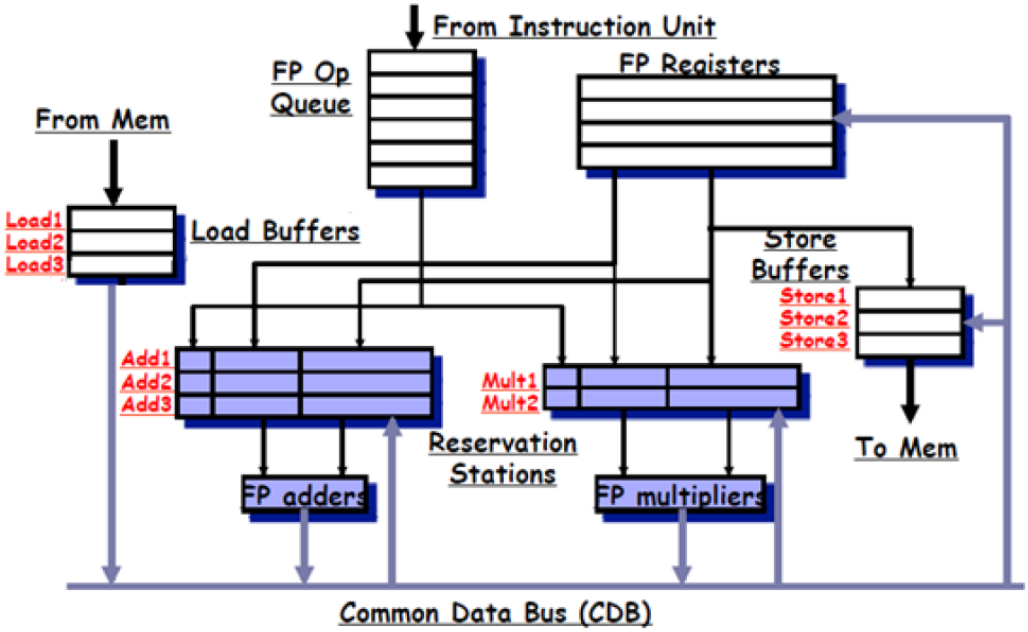
F00	1.00000	F01	1.00000	F02	1.00000	F03	1.00000
F04	1.00000	F05	1.00000	F06	105.00000	F07	1.00000
F08	1.00000	F09	1.00000	F10	1.00000		

5) 内存区

此区域可进行内存数值的查询与修改操作。在上方的内存地址框中填入要查询的内存地址，点击“查询”按钮，该内存值便会显示在下方框中。如有修改需求，则在内存数值框中进行编辑，再点击“修改”按钮即完成操

作。(由于内存地址过多，全部显示在界面上既不方便查看，也会使得页面变得十分杂乱，所以我们采取了定向查询的方式进行呈现)。

4. 设计思路



各个类（JS 中对应为函数）设计参靠上图，FP Registers、From Mem、Buffers、ReservationStations、Common_Data_Bus、Instruction 均对应了一个类，这样数据的传递简单地对应为他们在相应类之间的传递。

各个类的介绍：

1) ReservationStation

ReservationStation 是保留站模块，成员变量包括名称、状态、流入该

保留站的指令对象、指令参数、指令在指令序列中的位置（行数）。定义了保留站状态的取值：

Buffer.STATE_IDLE = 1;

Buffer.STATE_ISSUE = 2;

Buffer.STATE_EXECUTE = 3;

Buffer.STATE_WRITE_BACK = 4;

保留站有上述四种状态，用来对应指令从发射到执行完成的阶段，详见后面 Tomasulo 算法的描述。

2） Buffer

Load/Store 缓冲器，成员变量包括保留站的所有成员变量，以及一个表示内存的数组。

3） Instruction

指令类，用来描述指令的特征信息。成员变量包括：指令在指令序列中的位置，指令类型，指令剩下的周期数，指令的参数。

4） InstructionType

指令类型，用来描述一类指令的特征信息。成员变量包括 :指令名称，执行所需的周期数，各参数类型，对应的保留站类型。

5） Memory

内存类，成员变量包括：内存块大小，内存块数组。

6） Register

寄存器堆类，成员变量包括：寄存器数组，寄存器数量。

7） CommonDataBus

公共数据总线类，成员变量为两个字典，分别表示正在计算与已经完成计算的指令的结果。

定义了 5 个函数，用于从外部操作数据类的成员变量。

```
CommonDataBus.prototype.getBusy = function(type, name) {
    var result = this._busy[type + '.' + name];
    if (typeof result === 'undefined') {
        return null;
    } else {
        return result;
    }
};

CommonDataBus.prototype.setBusy = function(type, name, instruction) {
    this._busy[type + '.' + name] = instruction;
};

CommonDataBus.prototype.getResult = function(station) {
    var result = this._result['' + station.instruction.id];
    if (typeof result === 'undefined') {
        return null;
    } else {
```

```
            return result;
        }
    };

    CommonDataBus.prototype.setResult = function(station, value) {
        this._result['' + station.instruction.id] = value;
    };

    CommonDataBus.prototype.clearResult = function() {
        this._result = {};
};
```

Tomasulo 算法流程介绍（代码见 tomasulo.js）：

一个时钟周期的操作可分为三个阶段：ISSUE，EXCUTE，WRITEBACK。

ISSUE 阶段：

若有空闲保留站：发射浮点数指令，检测第一第二操作数是否就绪，若对应操作数没有就绪则进行寄存器换名，否则更新寄存器值到保留站，将当前保留站编号放入指令的目的寄存器的状态表项，以便目的寄存器接受结果。

若有空闲 Buffer：发射 Load/Store 指令，检测第一操作数，若没有就绪进行寄存器换名，否则更新寄存器值到 Buffer，对于 Load 指令，将当前 Buffer 编号放入指令的目的寄存器的状态表项，对于 Store 指令，检测要存储的数据是否就绪，若没有就绪进行寄存器换名，否则将数据取到 Buffer 的缓冲器单元。

EXCUTE 阶段：

遍历保留站，如果两个源操作数就绪，执行指令，产生结果。若 Buffer 的头部对应的基址寄存器就绪，则计算有效地址，对于 load 指令，还要从存储器读取数据。执行结束进入 WRITEBACK 阶段。

WRITEBACK 阶段：

对于非 Store 指令，若 CDB 就绪，写入结果到正在等待该结果的浮点寄存器及正在等待该结果作为操作数的保留站，释放当前保留站（Buffer）；对于 Store 指令，若存储的数据就绪，将数据写入存储器，释放当前 Buffer。