基于 Tomasulo 算法的乱序执行 CPU 的实现

颜彬　　王永锋

16337269 16337237

中山大学教务三班

2017 年 1 月 3 日

摘要

本文主要描述了一种基于 *Tomasolu* 算法的支持乱序执行的 *CPU* 的实现。网络上很少关于该算法的 *verilog* 实现，本文创新性地根据该算法的理论，从模块设计，时序设计等不同角度，独立进行分析、设计该算法的硬件实现。在最后，本文还对该算法的优点及不足及可能的提高方法做出了进一步的阐述，并为之后该 *CPU* 效率的提高从分支预测，前瞻执行等方面做出了适当的预测和展望。

关键词：乱序执行 动态调度

I. 选题背景

Tomasulo 算法是一种在乱序执行流水线 CPU 中使用的，用于对指令的顺序进行动态调度的算法。在流水线处理器中，先后执行的指令往往具有相关性，如上一条指令将一个数字写进寄存器中，下一条指令马上就要用这一个寄存器中存有的值来进行下一步的计算。这样的数据相关会导致流水线处理器中运行的冲突，这个时候可能可以通过旁路解决，但更多的时候，处理器只能够通过插入一个气泡，堵塞处理器的运行，才能够解决这一种数据冲突。由此，我们可以看出，传统顺序执行的流水线处理器，在处理具有极多数据相关的代码的时候，只能以较低的效率进行计算。

为了降低数据相关带来的处理器停顿时间，一般有两种处理方式。[1]

静态调度 静态调度的流水线依靠编译器对代码进行静态调度，以减少相关冲突。此类调度方式，是通过程序在进行编译的时候就把相关的指令拉开距离，来减少可能产生的停顿。由于是在编译期间进行的指令调度，在程序执行阶段，指令的顺序不能够进行改变，“静态”由此而来。

动态调度 动态的指令调度是在程序的执行过程中，依靠专门的硬件对指令进行调度。该种调度方式能够处理一些编译时情况不明的相关，还能够让代码的执行效率与产生指令的编译器解构。目前许多现代的处理器都采用了这种技术。

在本文中，我们关注对指令的动态调度的算法实现。首先探讨一下指令动态调度算法的可行性，要实现指令的动态调度，也就是说我们需要做到在运行过程中，由 CPU 自行判断哪些指令能够提前运行，同时还能够做到保持数据流和控制异常行为。为了实现这个目的，一个典型的算法是记分牌算法，该算法能够做到与前文无关的指令可以尽早进入执行阶段，从而让处理器的停顿时间减少，但此算法并不能真正解决指令中常常存在的反相关和输出相关，反而，还有可能会让原本的伪相关在乱序执行下变为真相关，从而又从另一个角度增加了处理器的停顿时间。

另一类实现动态调度的算法是 Tomasulo 算法，该算法解决了记分牌算法的缺陷。相比起记分牌算法，Tomasulo 算法中有两个重要的突破[2] 让它不仅能够最大限度的减少真相关带来的处理器停顿时间，同时直接解决了反相关和输出相关。

- 寄存器换名技术
- CDB 公共数据总线

II. 主要工作

本次课程设计，我们主要完成了以下工作

- 对 Tomasulo 算法，进行模块设计及时序设计，从而使用硬件的方式实现
- 将 Tomasulo 算法的实现，与 CPU 的设计结合，整合为一个能够做到支持乱序执行的流水线 CPU
- 编写机器代码样例，测试 CPU 的可用性
- 分析 Tomasulo 算法优点及不足，并提出可行的改进方案

III. 技术路线

1. CPU 支持指令集的简要介绍 2. 数据通路图设计 3. 关键模块设计（放模块端口图及部分代码）4. 测试样例及仿真效果（放上仿真波形图）

IV. 项目亮点

1. 采用了多周期 ALU 2. 访存阶段模拟真实情况，需要多个周期才能够完成访存任务 3. 硬件队列的实现

传统 Tomasulo 教材资料只给出了算法的软件模拟实现或伪代码实现。具体的硬件实现会遇到许多瓶颈。本项目对其中的一些难点做了突破，体现了一些创新性。

I. Architecture

框架流水线，局部并行化，部件多周期。

流水线 总框架大致可以分为 ‘指令发射 ‘,‘执行 ‘,‘广播 ‘等三个阶段。各个阶段流水执行。即每个时钟周期（几乎）保证有一条指令被执行, 一份数据被广播。

并行化 多个 ALU 并行地执行数据。一旦指令的操作数准备完毕，即可从保留站发射到 ALU 处。各个 ALU 的运算独立进行，互不干涉。

多周期 部件多周期更符合实际情况，本设计中各个执行单元都有 ‘state‘部件用于控制状态。所有执行和存储器件都在各个周期内分步骤完成。

II. mALU

利用阵列乘法器加速定点数乘法。采用 $32 + 16 + ... + 1 = 63$ 个简易的加法电路，按 5 层的方式排布成阵列，并行地计算乘法。将乘法的运行时间缩短至 5 个 CPU 时钟。

III. Queue in Hardware

利用硬件实现队列。注意到并解决了所有的所有的难点。包括

1. **计算空余位置号** 利用组合电路正确计算队列中的空余位置号
2. **分配保留站号** 每次新指令进队时，正确地分配唯一的保留站号
3. **处理广播冲突** 当进队的指令中的保留站号恰好为正在广播的保留站号时，队列能正确地将广播中的数据替换指令的数据，再写进队列里
4. **正确判断“伪满”** 若队列已满，但下一个周期到来时队列能发射一条指令，则队列实质上仍可以接受指令, 并没有处于真正满的状态。本设计能正确识别“伪满”现象，最大限度保证指令流动。

IV. Passing Extreme Testcase

通过了所有的边界条件测试样例。在算法的实际设计中，受到硬件时序的约束，会产生极其多的边界条件。例如

1. 指令队列流出 指令队列需要判断当前指令所在的保留站是否满。当满时，指令无法流出。

2. 广播与写入 当前广播的信号，恰好对应着当前写入信号的保留站号。此时器件应能正确捕获广播，避免遗漏。

3. 执行单元的状态转换 当执行单元（例如 ALU）将运算执行完毕时，它需要考虑以下几种状况：‘CDB‘总线是否忙碌，保留站是否仍发来请求。ALU 的附属器件 ‘state‘模块需要对其进行分析，判断其接下来进入的状态。

4. CDB 繁忙 CDB 是所有“写”操作的唯一总线。当多个器件同时企图写总线时，将会引发冲突，此时需要一个优先译码器决定哪个执行器件的输出可以被广播。被拒绝广播的器件必须阻塞等待，直到 CDB 总线接受广播。

V. Good Coding Style

良好的代码风格。

采用宏定义增强可读性 将所有常量写入头文件中，便于管理。所有常数都用宏定义代替，增强可读性。

generate 语法 当大量产生相同器件，或进行相同的连线时，采用 verilog 2001 标准中加入的 generate 语法, 以达到效率、准确地描述硬件的效果。

```
1       generate
2           genvar i
3           for (i = 0; i < n; i = i + 1) begin: loop
4               // codes here
5           end
6       endgenerate
```

V. 效果评价

1. 实现该算法带来的性能提升 2. 该算法存在的不足及改进方案

VI. 项目前景

参考文献

[1] 张晨曦, 王志英, 张春元, and 戴葵. 计算机体系结构, 2000.

[2] Wikipedia contributors. Tomasulo algorithm — wikipedia, the free encyclopedia, 2018. [Online; accessed 7-January-2018].