Tomasulo 算法实现与可视化 实验报告

计 93 王哲凡 2019011200

实验环境

环境要求

- 测试平台：WSL2，利用 XLaunch 得到可视化 GUI。
- 可行平台：普通 Linux 主机，Mac OS，Windows（需要修改 `Makefile` 部分内容）。
- 语言：C++ 与 Python3
- 软件需求：gcc 编译工具，Python 3.8.0 及以上，Make 工具。
- 库需求：仅需要安装 `tkinter` 包。

代码使用

在 `sample` 文件夹下放置输入汇编文件，并以 `inputXXX.asm` 命名，如 `input5.asm`。

之后在根目录使用如下命令即可对 `input5.asm` 进行可视化展示：

```
make simulate TEST=5
```

如果只需要后端输出，可以使用以下命令：

```
make test TEST=5
```

代码分工

`.cpp` 与 `.hpp` 文件为封装后的 `template.c` 文件，也即后端部分。

`simulator.py` 为可视化前端，`assembler.py` 为汇编器。

实验设计思路

首先将实验分为前后端与汇编器三个部分：

- 汇编器用 Python 编写，只进行了简单的汇编到机器码的翻译。
- 前端使用 Python 编写，负责在后端给出指令运行结果的前提下，展示 Tomasulo 算法的可视化流程；
- 后端利用 `template.c` 改写，使用 C++ 语言，负责 Tomasulo 算法的主体逻辑模拟与实现。

后端实现约定

基本按照给定框架完成，其中利用面向对象，将原有的状态转移函数修改为了 `machineState` 的成员函数，省去了一些指针的传递麻烦。

为了方便检查，有以下约定：

- 其中未处理 `LOAD` 与 `STORE` 指令地址冲突，默认不存在。
- 规定每周期只可发射一条指令、只可提交一条指令，其余阶段无限制。
- `STORE` 指令由于在提交阶段执行写入，所以添加了 `storeQueue` 用于记录当前正在内存中写入的 `STORE` 指令信息，并在加入队列的两周期后，对其进行实际写入，用以模拟真实的写入延迟，并且在实际写入过程中，仍占用对应的保留站/缓冲区。
- `LOAD` 指令的执行阶段被设置为 3 个时钟周期，这是 1 个地址计算的整数计算周期和 2 个读内存的时钟周期相加得到的。
- 特别地，为了防止读取非法内存，在 `pc >= memorySize` 时，停止发射指令（这说明上一条发射的指令为 `halt`，有概率需要停机）。

后端实现思路

在每个时钟周期，按照倒序分别进行提交、写结果、执行、发射和尝试发射阶段：

- 提交：
  - 首先判断 ROB 头指令是否处于提交状态，如果处于且非分支非跳转指令，则只需将寄存器结果写入，或加入 `storeQueue` 队列。
  - 对于分支和跳转指令，先根据实际结果，更新 BTB 表项，再判断是否预测成功，如果失败则需清空当前 ROB，并设置 PC 为 ROB 中 `result` 值。
- 写结果：
  - 对于 `STORE` 指令，需要判断写入的寄存器是否就绪，如果未就绪，则不改变，如果就绪，则设置合适的 `result` 和 `storeAddress` 字段并进入提交阶段。
  - 对于其他指令，只须调用 `getResult()` 获得结果并进入提交阶段。
- 执行：
  - 只须将 `exTimeLeft` 减一，并判断是否为 0，如果为 0，即可进入写结果阶段。
- 发射：
  - 对于已经发射的指令，如果其 `QJ` 和 `QK` 均为 0，或者 `QJ` 为 0 且为 `STORE` 指令，则进入执行阶段。
- 尝试发射：
  - 对于当前 `pc`，如果地址合法（小于 `memorySize`）且存在空闲的保留站和 ROB，则发射指令。
  - 并根据 `getTarget` 获取下一条指令 PC 值的预测。
  - 发射时，特别注意需要将分支指令与跳转指令的 `VK` 设置为 `pc + 1`，并且对于部分指令需要修改寄存器文件结构。

选做：BTB 实现

我同时完成了选做的分支预测缓冲区 BTB 的实现。

为了方便实现，我在 ROB 的字段中添加了当条指令对应的 PC 值和前瞻执行的下条 PC 值，并实现了 FIFO 的 BTB 替换策略，为此我在 `machineState` 中添加了 `btPtr` 字段表示下一个可用于插入的编号。

- 对于 `getPrediction()` 函数，我将其改写，设计为传入 `pc` 查询 BTB 表中是否存在对应的表项并且合法，如果存在返回对应编号，否则返回 -1。

- 对于 `updateBTB()` 函数，我认为 `branchPC` 为分支指令的 PC 值，`targetPC` 为分支指令如果跳转其对应的跳转地址，`outcome` 代表分支指令本次是否跳转。
  - 首先尝试查找表项，如果存在，则根据 `outcome` 更新 2 位饱和计数器，即如果跳转，则 `branchPred` 加一（不超过 3），否则减一（不小于 0）。
  - 如果不存在，则根据 `btPtr` 来进行表项插入。
- 对于 `getTarget()` 函数，我利用 `getPrediction()` 快速判断表项是否存在，并根据结果与 `branchPred` 来确定返回 `branchTarget` 或者 `pc + 1`。

为了修复分支预测错误，我在提交分支指令和跳转指令时，统一进行了分支预测验证，如果预测失败，则清空 ROB 及其对应的保留站和寄存器结果表。

前端实现

利用 `tkinter` 包可视化。

首先读取后端输出的信息文件，并将其解析为 Python 格式的字典留待使用。

在可视化过程中，维护一个全局变量 `CYCLES`，表示当前所在的时钟周期数，则对应的各表内容即可根据当前周期的输出得到。

为了方便的查看不同周期的 CPU 状态，我设计了四个按钮分别用于：

- 进入下一个时钟周期。
- 回到上一个时钟周期。
- 跳过连续 10 个时钟周期。
- 重置回到初始状态。

实验结果

初始状态


Tomasulo Display
上一个周期
下一个周期
下十个周期
重置
PC: 16
周期数: 0
寄存器表
编号 值 有效 ROB 编号 值 有效 ROB
0 0 是 16 0 是
1 0 是 17 0 是
2 0 是 18 0 是
3 0 是 19 0 是
4 0 是 20 0 是
5 0 是 21 0 是
6 0 是 22 0 是
7 0 是 23 0 是
8 0 是 24 0 是
9 0 是 25 0 是
10 0 是 26 0 是
11 0 是 27 0 是
12 0 是 28 0 是
13 0 是 29 0 是
14 0 是 30 0 是
15 0 是 31 0 是
ROB
编号 指令 状态 单元 有效 结果 地址
保留站
保留站名 Busy 指令 Vj Vk Qj Qk 剩余时间 ROB编号
LOAD1 0
LOAD2 0
STORE1 0
STORE2 0
INT1 0
INT2 0
BTB
PC PC指令 目标 目标指令 历史
内存
地址 数据 地址 数据 地址 数据 地址 数据 地址 数据 地址 数据 地址 数据
0 0 1 0 2 0 3 0 4 0 5 0 6 0
7 0 8 0 9 0 10 0 11 0 12 0 13 0
14 0 15 0

中间状态


Tomasulo Display
上一个周期
下一个周期
下十个周期
重置
PC: 28
周期数: 170
寄存器表
编号 值 有效 ROB 编号 值 有效 ROB
0 0 是 16 0 是
1 2 是 17 0 是
2 8 是 18 0 是
3 0 否 4 19 0 是
4 0 是 20 0 是
5 0 是 21 0 是
6 0 是 22 0 是
7 0 是 23 0 是
8 34 是 24 0 是
9 34 否 3 25 0 是
10 55 是 26 0 是
11 0 是 27 0 是
12 0 是 28 0 是
13 0 是 29 0 是
14 0 是 30 0 是
15 0 是 31 0 是
ROB
编号 指令 状态 单元 有效 结果 地址
2 sw r10,r2,0 提交 STORE1 否 8
3 lw r9,r2,0 执行 LOAD1 否 0
4 sub r3,r9,r10 发射 INT1 否 0
5 j 4 发射 INT2 否 1
保留站
保留站名 Busy 指令 Vj Vk Qj Qk 剩余时间 ROB编号
LOAD1 1 lw r9,r2,0 8 0 2 3
LOAD2 0
STORE1 1 sw r10,r2,0 8 55 0 2
STORE2 0
INT1 1 sub r3,r9,r10 55 0 1 4
INT2 1 j 4 0 24 1 5
BTB
PC PC指令 目标 目标指令 历史
23 j 4 28 beqz r3,-5 STRONGTAKEN
28 beqz r3,-5 24 addi r2,r2,1 STRONGTAKEN
26 beqz r1,2 29 halt STRONGNOT
27 j -10 18 add r10,r8,r9 STRONGTAKEN
内存
地址 数据 地址 数据 地址 数据 地址 数据 地址 数据 地址 数据 地址 数据
0 1 1 2 2 3 3 5 4 8 5 13 6 21
7 34 8 0 9 0 10 0 11 0 12 0 13 0
14 0 15 0

结束状态


Tomasulo Display
上一个周期
下一个周期
下十个周期
重置
PC: 30
周期数: 208
寄存器表
编号 值 有效 ROB 编号 值 有效 ROB
0 0 是 16 0 是
1 0 是 17 0 是
2 10 是 18 0 是
3 0 是 19 0 是
4 0 是 20 0 是
5 0 是 21 0 是
6 0 是 22 0 是
7 0 是 23 0 是
8 55 是 24 0 是
9 89 是 25 0 是
10 89 是 26 0 是
11 0 是 27 0 是
12 0 是 28 0 是
13 0 是 29 0 是
14 0 是 30 0 是
15 0 是 31 0 是
ROB
编号 指令 状态 单元 有效 结果 地址
8 halt 提交 INT1 是 0 0
保留站
保留站名 Busy 指令 Vj Vk Qj Qk 剩余时间 ROB编号
LOAD1 0
LOAD2 0
STORE1 0
STORE2 0
INT1 0
INT2 0
BTB
PC PC指令 目标 目标指令 历史
23 j 4 28 beqz r3,-5 STRONGTAKEN
28 beqz r3,-5 24 addi r2,r2,1 STRONGTAKEN
26 beqz r1,2 29 halt WEAKNOT
27 j -10 18 add r10,r8,r9 STRONGTAKEN
内存
地址 数据 地址 数据 地址 数据 地址 数据 地址 数据 地址 数据 地址 数据
0 1 1 2 2 3 3 5 4 8 5 13 6 21
7 34 8 55 9 89 10 0 11 0 12 0 13 0
14 0 15 0