Tomasulo 架构

2023.6.20

刘祎禹

简单执行流程

- 逐个指令执行

运算指令

1. 读取指令
2. 从寄存器读取数据
3. 计算
4. 存回寄存器

跳转指令

1. 读取指令
2. 从寄存器读取数据
3. 比较数据
4. 修改 PC

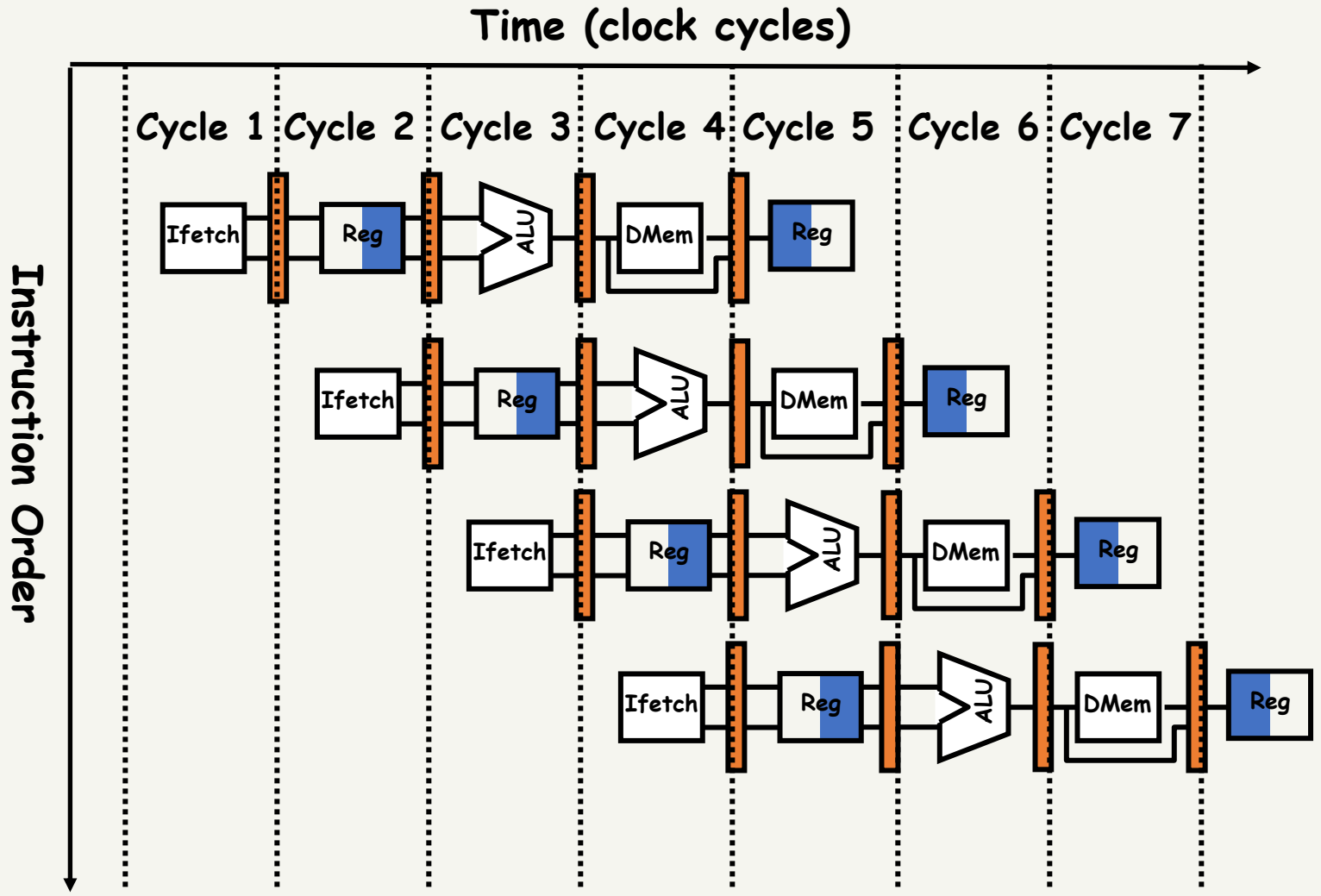
读取内存指令

1. 读取指令
2. 从内存读取数据
3. 将数据存到寄存器

写入内存指令

1. 读取指令
2. 从寄存器读取数据
3. 将数据写到内存

流水线－5 个阶段


Time (clock cycles)
Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5
Cycle 6
Cycle 7
Instruction Order
Ifetch
Reg
ALU
DMem
Reg

如果遇到这样的代码会发生什么?

```
lw x10, x2(0)
sub x10, x10, x3
add x11, x12, x13
```

如果遇到这样的代码会发生什么?

```
lw x10, x2(0)
sub x10, x10, x3
add x11, x12, x13
```

- 第二行会卡住
- 如果认为访存速度是现实情况（至少 10 个周期），那么会卡非常长的时间
- 但第三行其实可以执行（对前面的指令没有依赖关系）

如果遇到这样的代码会发生什么?

```
lw x10, x2(0)
sub x10, x10, x3
add x11, x12, x13
```

- 第二行会卡住
- 如果认为访存速度是现实情况（至少 10 个周期），那么会卡非常长的时间
- 但第三行其实可以执行（对前面的指令没有依赖关系）

如何解决?

如果遇到这样的代码会发生什么?

```
lw x10, x2(0)
sub x10, x10, x3
add x11, x12, x13
```

- 第二行会卡住
- 如果认为访存速度是现实情况（至少 10 个周期），那么会卡非常长的时间
- 但第三行其实可以执行（对前面的指令没有依赖关系）

如何解决? → 乱序执行

乱序执行会有什么问题?

乱序执行会有什么问题?

- 如果互相间存在次序依赖，则乱序可能会破坏程序的顺序，导致错误
- 请想一种解决方案

Tomasulo 的解决方案

乱序执行 – 总体思路

- 设计一个队列用于顺序提交指令 – Reorder Buffer (RoB)
  - 保证结果是顺序的，但过程可以是乱序的
  - 解决 Write after Read 和 Write after Write
- 记录依赖 – Register File
  - Register File 的名字比较奇怪，实际上就是包含寄存器的模块
  - 在 Register File 里记录每个 Register 的依赖
- 解决依赖问题 – Renaming
  - Renaming 实际上是标记依赖
  - 解决 Read after Write

Reorder Buffer (RoB)

- 一个 buffer (FIFO)
- 提交表示真正执行了这个指令
- 当队列顶端的元素可以提交 (通常用 ready 表示) 时真正完成操作（如写回寄存器、写回内存、跳转到某个地址）
- 当数据更新时，将元素设为 ready，表示可以提交

				Reorder buffer		
Entry	Busy	Instruction		State	Destination	Value
1	No	`fld`	`f6,32(x2)`	Commit	`f6`	`Mem[32 + Regs[x2]]`
2	No	`fld`	`f2,44(x3)`	Commit	`f2`	`Mem[44 + Regs[x3]]`
3	Yes	`fmul.d`	`f0,f2,f4`	Write result	`f0`	#2 × `Regs[f4]`
4	Yes	`fsub.d`	`f8,f2,f6`	Write result	`f8`	#2 – #1
5	Yes	`fdiv.d`	`f0,f0,f6`	Execute	`f0`	
6	Yes	`fadd.d`	`f6,f8,f2`	Write result	`f6`	#4 + #2

记录依赖 – Register File

- Register File 实际上就是包含寄存器的模块
- 记录每个 Register 的依赖
- 依赖为 RoB 的位置，是一个数字
- Instruction Decode 阶段更新依赖
  - 注意需要 forward 给下次 Instruction Decode 的指令
- Instruction Decode 阶段同时获取依赖
- Commit 的时候更新依赖

解决依赖 - Renaming

- 在 Instruction Decode 阶段
- 如果不存在依赖，直接读取寄存器数据
- 如果存在依赖，则换成 RoB 上依赖的指令对应的位置
- 当收到更新数据，去除自己的依赖

传输更新信息 – CDB

- Common Data Bus – CDB
- 当有数据更新的时候，将数据放到 CDB 上广播
- 实际情况下，CDB 的阻塞情况较为严重，可以加大 CDB 的数据并行个数，也可以采取更简单的实现

并行运算 – Reservation Station

- Reservation Station – RS
- 用于连接 ALU (Arithmetic Logic Unit)
- 对于不再依赖其他指令的结果的行，可以将其放到 ALU 中运算

	Reservation stations							
Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	No							
Load2	No							
Add1	No							
Add2	No							
Add3	No							
Mult1	No	`fmul.d`	`Mem[44 + Regs[x3]]`	`Regs[f4]`			#3	
Mult2	Yes	`fdiv.d`		`Mem[32 + Regs[x2]]`	#3		#5	

Busy - 当前 station 是否被使用

Op - 操作类型

Vj, Vk - source 的 value

Qj, Qk - source 的依赖

- 如果存在依赖关系并且依赖的数据未得到，则 Qj/Qk 非 0；若两者都为 0，则说明该指令 ready

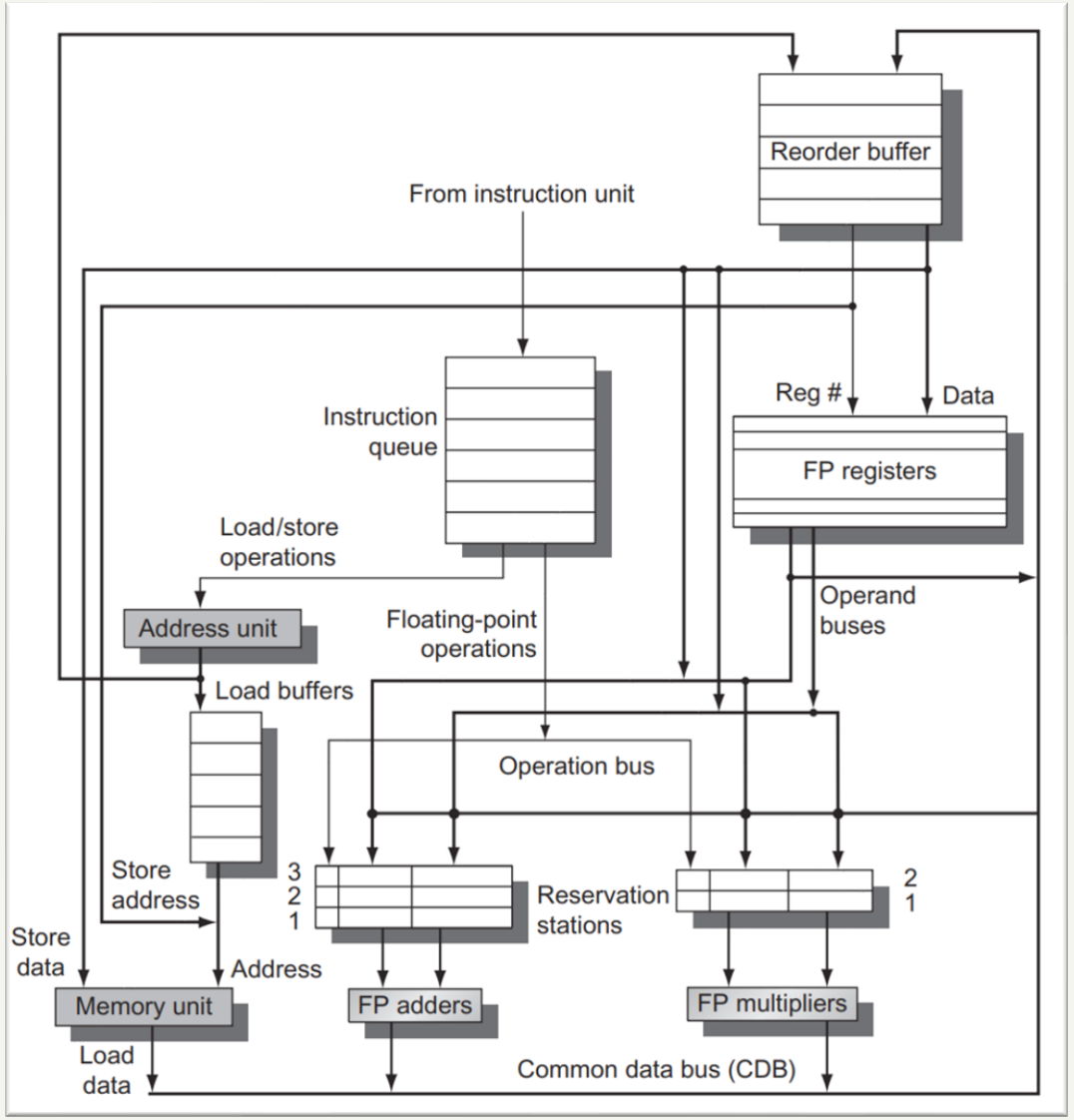
Dest - 目的寄存器

A - 立即数

分支指令

- RoB 保证了最终提交指令时下面的指令可以轻易取消执行
- 如果实现了分支预测，那么在 RoB commit 分支指令时检查预测是否正确
  - 如果正确，继续执行
  - 如果不正确，修改 PC，清空所有未 commit 的数据

参考架构 (From CAAQA)



注意

- Tomasulo 对于 Load/Store 指令有特别的处理（本质上和 RS 差不多），请自行考虑
- 分清楚哪些部分是时序逻辑，哪些部分是组合逻辑
- 请一定要把所有情况都想清楚之后再写
- 注意各种边界情况
- 可参考 CAAQA: 3.4 开始的一部分内容

谢谢