第四章 ppt 更正报告

CSAPP

黎有琦
liyouqi@bit.edu.cn

计算机学院
北京理工大学

November 18, 2023

Outline

Outline

返回示例 Return Example

指令流如下：

```
0x000:  irmovq Stack,%rsp    # Intialize stack pointer
0x00a:  call p               # Procedure call
0x013:  irmovq $5,%rsi       # Return point
0x01d:  halt
0x020:  .pos 0x20
0x020:  p: irmovq $-1,%rdi   # procedure
0x02a:  ret
0x02b:  irmovq $1,%rax       # Should not be executed
0x035:  irmovq $2,%rcx       # Should not be executed
0x03f:  irmovq $3,%rdx       # Should not be executed
0x049:  irmovq $4,%rbx       # Should not be executed
0x100:  .pos 0x100
0x100:  Stack:               # Stack: Stack pointer
```

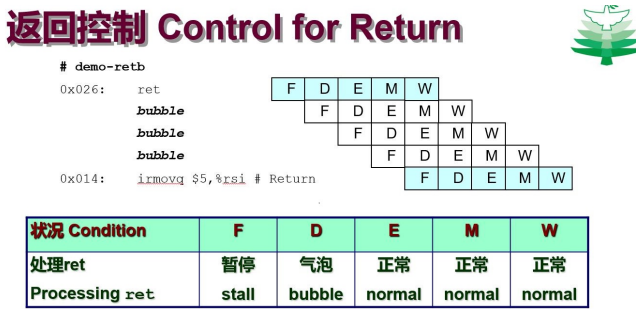


Figure: 图 1: 错误 ppt 页

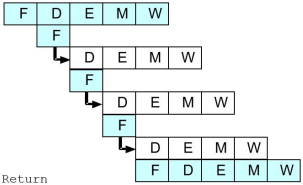
- 原 ppt 页码：211
- 错误原因：流水线图和下面下个周期流水线寄存器的控制逻辑动作不对应。
- 地址不对

更正

正确的 ppt 如下修改

返回控制 Control for Return

```
# demo-retb
0x02a:    ret
0x02b:    irmovq $1,%rax
          bubble
0x02b:    irmovq $1,%rax
          bubble
0x02b:    irmovq $1,%rax
          bubble
0x013:    irmovq $5,%rsi # Return
```



状况 Condition	F	D	E	M	W
处理ret Processing ret	暂停 stall	气泡 bubble	正常 normal	正常 normal	正常 normal

Figure: 图 2: 正确 ppt 页

如何理解 ret 流水线控制逻辑的动作表

我们先来看看课本上 (p316) 的这段话，尤其是画红线的地方。

2. 发现特殊控制条件

图 4-64 总结了需要特殊流水线控制的条件。它给出的表达式描述了在哪些条件下会出现这三种特殊情况。一些简单的组合逻辑块实现了这些表达式，为了在时钟上升开始下一个周期时控制流水线寄存器的活动，这些块必须在时钟周期结束之前产生出结果。在一个时钟周期内，流水线寄存器 D、E 和 M 分别保持着处于译码、执行和访存阶段中的指令的状态。在到达时钟周期末尾时，信号 d_srcA 和 d_srcB 会被设置为译码阶段中指令的源操作数的寄存器 ID。当 ret 指令通过流水线时，要想发现它，只要检查译码、执行和访存阶段中指令的指令码。发现加载/使用冒险要检查执行阶段中的指令类型(mrmovq 或 popq)，并把它的目的寄存器与译码阶段中指令的源寄存器相比较。当跳转指令在执行阶段时，流水线控制逻辑应该能发现预测错误的分支，这样当指令进入访存阶段时，它就能设置从错误预测中恢复所需要的条件。当跳转指令处于执行阶段时，信号 e_Cnd 指明是否要选择分支。通过检查访存和写回阶段中的指令状态值，就能发现异常指令。对于访存阶段，我们使用在这个阶段中计算出来的信号 m_stat，而不是使用流水线寄存器的 M_stat。这个内部信号包含着可能的数据内存地址错误。

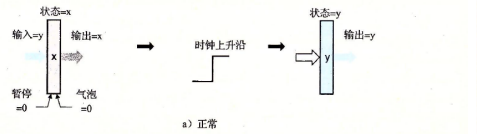
条件	触发条件
处理 ret	IRET∈{D_icode,E_icode,M_icode}
加载/使用冒险	E_icode∈{IMRMOVL,IPOPL} && E_dstM∈{d_srcA,d_srcB}
预测错误的分支	E_icode=IJXX && ! e_Cnd
异常	m_stat∈{SADR,SINS,SHLT}||W_stat∈{SADR,SINS,SHLT}

图 4-64 流水线控制逻辑的检查条件。四种不同的条件要求改变流水线，暂停流水线或者取消已经部分执行的指令

如何理解 ret 流水线控制逻辑的动作表'Con.

从第一个画线的地方，我们得到一个观察：当检测到处理 ret 指令，必须在下一个周期开始前，即时钟沿上升时候，控制流水线寄存器的活动：暂停、气泡、正常，即用一个控制逻辑电路输出相应的比特信号，来控制相应的流水线寄存器，如下图所示：

都设为 0，使得寄存器加载它的输入作为新的状态。当暂停信号设为 1 时(图 4-65b)，禁止更新状态。相反，寄存器会保持它以前的状态。这使得它可以将指令阻塞在某个流水线阶段中。当气泡信号设置为 1 时(图 4-65c)，寄存器状态会设置成某个固定的复位配置(reset configuration)，得到一个等效于 nop 指令的状态。一个流水线寄存器的复位配置的 0、1 模式是由流水线寄存器中字段的集合决定的。例如，要往流水线寄存器 D 中插入一个气泡，我们要将 icode 字段设置为常数值 INOP(图 4-26)。要往流水线寄存器 E 中插入一个气泡，我们要将 icode 字段设为 INOP，并将 dstE、dstM、srcA 和 srcB 字段设为常数 RNONE。确定复位配置是硬件设计师在设计流水线寄存器时的任务之一。在此我们不讨论细节。我们会将气泡和暂停信号都设为 1 看成是出错。



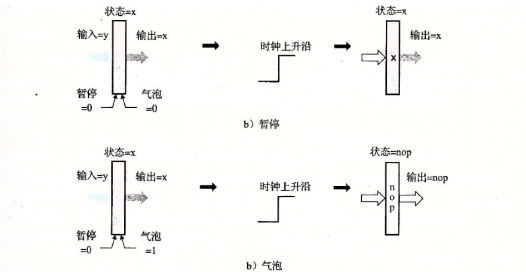
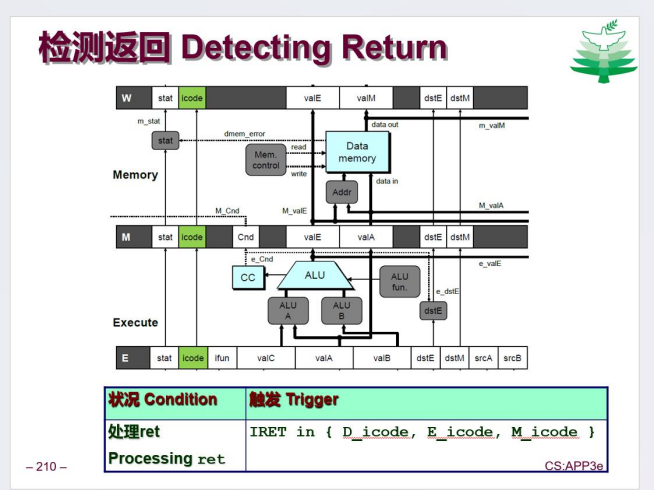


图 4-65 附加的流水线寄存器操作。a)在正常条件下，当时钟上升时，寄存器的状态和输出被设置成输入的值；b)当运行在暂停模式中时，状态保持为先前的值不变；c)当运行在气泡模式中时，会用 nop 操作的状态覆盖当前状态

如何理解 ret 流水线控制逻辑的动作表'Con.

- 回想一下，什么时候能检测到执行 ret 指令?
- 取值阶段完成就能检测到，因为已经确定 icode 和 ifun
- 那么，我们看看前两页，触发处理 ret 的条件是什么？
- IRET in { D_icode, E_icode, M_icode }
- 有点反直觉，怎么是 D_icode 开始检测到？
- **注意**：取值完后，icode 和 ifun 是写入 D 寄存器，是在 F 阶段完之后，写入 D 寄存器，D 阶段开始才读 D 寄存器。所以 D 阶段开始，触发处理 ret 错误控制条件。而此时，ret 下一条指令已进来，那么下一个周期的寄存器动作是，暂停 F 寄存器、往 D 寄存器插入气泡，E 寄存器正常（ret 到 E 阶段，已读的错误指令即将进入 D 阶段，为了不执行它，继续把它暂停在 F 阶段，即 F 寄存器暂停、D 寄存器插入气泡），所以就有了图 2 的寄存器控制动作表。

其他 ppt 错误



图画错，但是没有原图修改，原来材料的 ppt 截图来源不清晰。下面的触发 HCL 条件表是对的。

处理器 6 阶段的具体过程

取指 Fetch	icode,ifun rA,rB valC valP
译码 Decode	valA, srcA valB, srcB
执行 Execute	valE Cond code
访存 Memory	valE
写回 Write back	dstE dstM
更新 PC	PC

取指阶段确定 icode 和 ifun；译码确定寄存器来源，读寄存器