山东大学_____计算机科学与技术_____学院

____计算机体系结构____课程实验报告

学号：202000130143	姓名： 郑凯饶	班级： 计科 1 班
实验题目：实验六 指令调度		
实验学时：2	实验日期：5.17	

实验目的：

通过本实验，加深对指令调度的理解，了解指令调度技术对 CPU 性能改进的好处。

硬件环境：

Dell Latitude 5411
Intel(R) Core(TM) i5-10400H CPU @ 2.60GHz(8GPUs),~2.6GHz

软件环境：
VMware Workstation 16 Player
Windows 7

实验步骤与内容：

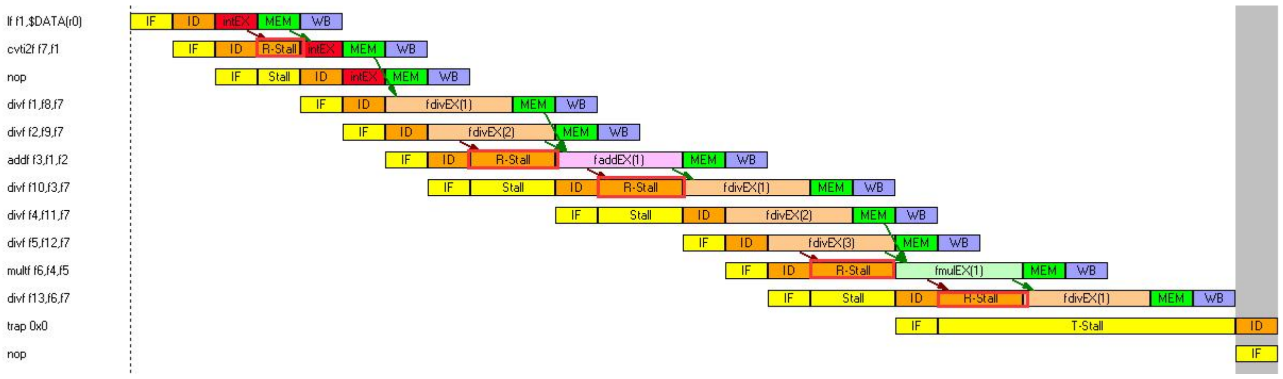
1. 阅读汇编程序，对比前面指令调度的调度顺序：

```
;-----------------------------------------------------------------
; Example to illustrate instruction scheduling
;-----------------------------------------------------------------
        .data
        .global     ONE
ONE:      .word       1
        .text
        .global main
main:
        lf          f1,ONE      ;turn divf into a move
        cvti2f       f7,f1      ;by storing in f7 1 in  ；将整数转换为浮点数存储于 f7
        nop                     ;floating-point format
        divf        f1,f8,f7    ;move Y=(f8) into f1
        divf        f2,f9,f7    ;move Z=(f9) into f2    ；除法运算
        addf         f3,f1,f2   ；加法运算
        divf        f10,f3,f7   ;move f3 into X=(f10)
        divf        f4,f11,f7   ;move B=(f11) into f4
        divf        f5,f12,f7   ;move C=(f12) into f5
        multf        f6,f4,f5   ；乘法运算
        divf        f13,f6,f7   ;move f6 into A=(f13)
Finish:
        trap        0
```

可以看到黄色高亮的两条指令被往后移动了。

```
        lf          f1,ONE       ;turn divf into a move
        cvti2f      f7,f1        ;by storing in f7 1 in
        nop                      ;floating-point format
        divf        f1,f8,f7     ;move Y=(f8) into f1
        divf        f2,f9,f7     ;move Z=(f9) into f2
        divf        f4,f11,f7    ;move B=(f11) into f4
        divf        f5,f12,f7    ;move C=(f12) into f5
        addf        f3,f1,f2
        multf       f6,f4,f5
        divf        f10,f3,f7    ;move f3 into X=(f10)
        divf        f13,f6,f7    ;move f6 into A=(f13)
```

2. 运行两个程序，观察程序执行过程中各种相关发生并对比：



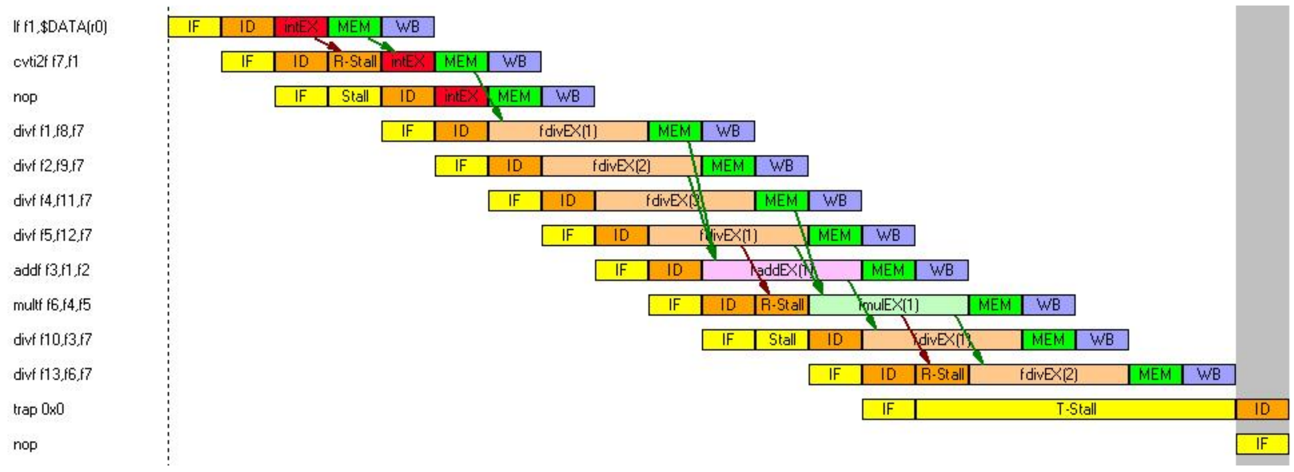
从时钟周期视图中可以看到，初始化出现了 1 次读写相关，之后在执行运算的过程中出现了 4 次读写相关，每次相关引起了 2 个 stall。

第一次：divf f2, f9, f7 & addf f3, f1, f2

第二次：addf f3, f1, f2 & divf f10, f3, f7

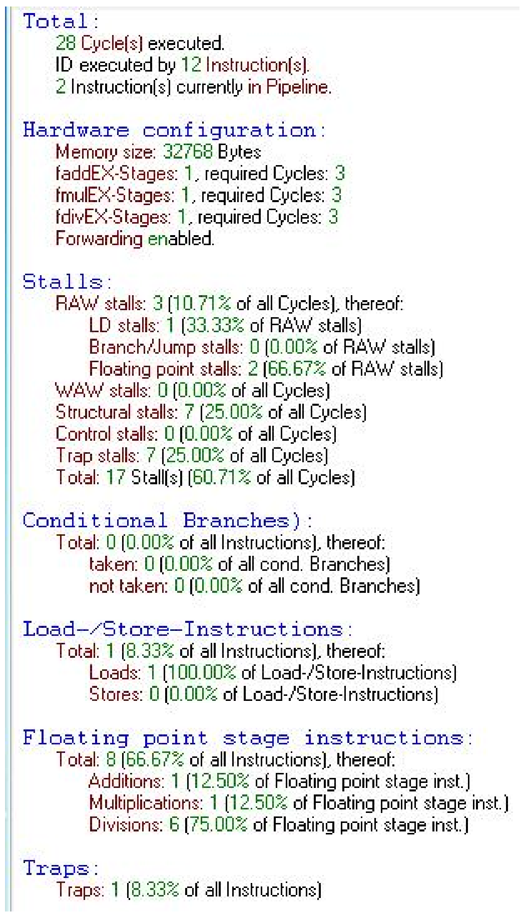
第三次：divf f5, f12, f7 & multf f6, f4, f5

第四次：multf f6, f4, f5 & divf f13, f6, f7

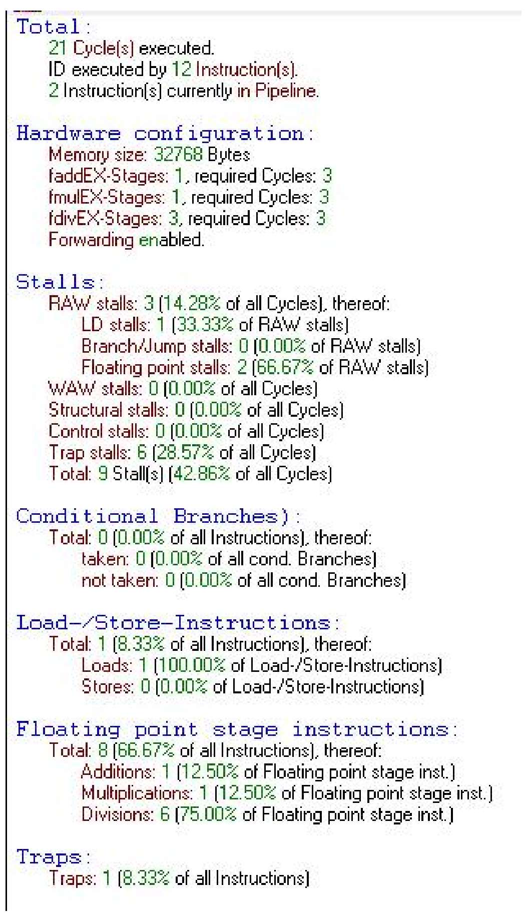


初始化 1 次，最后出现了两次读写相关，每次停 1 个 stall。

打开统计视图：



```
Total:
    28 Cycle(s) executed.
    ID executed by 12 Instruction(s).
    2 Instruction(s) currently in Pipeline.

Hardware configuration:
    Memory size: 32768 Bytes
    faddEX-Stages: 1, required Cycles: 3
    fmulEX-Stages: 1, required Cycles: 3
    fdivEX-Stages: 1, required Cycles: 3
    Forwarding enabled.

Stalls:
    RAW stalls: 3 (10.71% of all Cycles), thereof:
        LD stalls: 1 (33.33% of RAW stalls)
        Branch/Jump stalls: 0 (0.00% of RAW stalls)
        Floating point stalls: 2 (66.67% of RAW stalls)
    WAW stalls: 0 (0.00% of all Cycles)
    Structural stalls: 7 (25.00% of all Cycles)
    Control stalls: 0 (0.00% of all Cycles)
    Trap stalls: 7 (25.00% of all Cycles)
    Total: 17 Stall(s) (60.71% of all Cycles)

Conditional Branches):
    Total: 0 (0.00% of all Instructions), thereof:
        taken: 0 (0.00% of all cond. Branches)
        not taken: 0 (0.00% of all cond. Branches)

Load-/Store-Instructions:
    Total: 1 (8.33% of all Instructions), thereof:
        Loads: 1 (100.00% of Load-/Store-Instructions)
        Stores: 0 (0.00% of Load-/Store-Instructions)

Floating point stage instructions:
    Total: 8 (66.67% of all Instructions), thereof:
        Additions: 1 (12.50% of Floating point stage inst.)
        Multiplications: 1 (12.50% of Floating point stage inst.)
        Divisions: 6 (75.00% of Floating point stage inst.)

Traps:
    Traps: 1 (8.33% of all Instructions)
```



```
Total:
    21 Cycle(s) executed.
    ID executed by 12 Instruction(s).
    2 Instruction(s) currently in Pipeline.

Hardware configuration:
    Memory size: 32768 Bytes
    faddEX-Stages: 1, required Cycles: 3
    fmulEX-Stages: 1, required Cycles: 3
    fdivEX-Stages: 3, required Cycles: 3
    Forwarding enabled.

Stalls:
    RAW stalls: 3 (14.28% of all Cycles), thereof:
        LD stalls: 1 (33.33% of RAW stalls)
        Branch/Jump stalls: 0 (0.00% of RAW stalls)
        Floating point stalls: 2 (66.67% of RAW stalls)
    WAW stalls: 0 (0.00% of all Cycles)
    Structural stalls: 0 (0.00% of all Cycles)
    Control stalls: 0 (0.00% of all Cycles)
    Trap stalls: 6 (28.57% of all Cycles)
    Total: 9 Stall(s) (42.86% of all Cycles)

Conditional Branches):
    Total: 0 (0.00% of all Instructions), thereof:
        taken: 0 (0.00% of all cond. Branches)
        not taken: 0 (0.00% of all cond. Branches)

Load-/Store-Instructions:
    Total: 1 (8.33% of all Instructions), thereof:
        Loads: 1 (100.00% of Load-/Store-Instructions)
        Stores: 0 (0.00% of Load-/Store-Instructions)

Floating point stage instructions:
    Total: 8 (66.67% of all Instructions), thereof:
        Additions: 1 (12.50% of Floating point stage inst.)
        Multiplications: 1 (12.50% of Floating point stage inst.)
        Divisions: 6 (75.00% of Floating point stage inst.)

Traps:
    Traps: 1 (8.33% of all Instructions)
```

可以看到通过指令调度将加法运算指令以及相关一条除法指令延后执行，减少了读写相关，从原来的 9stalls 变为 3stalls，使程序的总执行周期从 27 优化为 21。使得性能提升到原来的 27/21=1.286 倍。

还需要关注一点，VM 拥有 3 个除法单元，之前的调度并没有充分利用所有的运算部件，若只有 2 个除法单元，调度之后会有新的结构相关问题，并不能优化程序。

结论分析与体会：

恰好正在学习编译原理的代码优化，优化一个基本块，通过构建基本块中四元式的 DAG，合并已知量，删除公共的子表达式以及删除无用赋值，达到代码优化的效果。这个思路可以应用到指令的执行调度中，构建指令 DAG 图，在不改变 DAG 图（不改变程序的执行逻辑）的情况下，分析程序局部各个硬件资源的使用情况，通过调度使得它们充分利用。