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：201900130133	姓名：施政良	班级： 四班
实验题目： 数据相关		
实验学时：2	实验日期： 2022-05-17	

实验目的：

（1）通过本实验，加深对数据相关的理解

（2）掌握如何使用定向技术来减少数据 相关带来的暂停。

硬件环境：

WinDLX (是一个基于 Windows 的 DLX 模拟器)

软件环境：

Windows 7

实验步骤与内容：

实验内容

本次实验主要涉及使用定向技术优化指令的执行，具体的实验步骤可以划分为如下几个步骤

（1） 在不采用定向技术的情况下（通过 Configuration 菜单中的 Enable Forwarding 选项设置），用 WinDLX 模拟器运行程序 data_d.s

（2） 记录数据相关引起的暂停时钟周期数以及程序执行的总时钟周期数，计 算暂停时钟周期数占总执行周期数的百分比。

（3） 在采用定向技术的情况下，用 WinDLX 模拟器再次运行程序 data_d.s。

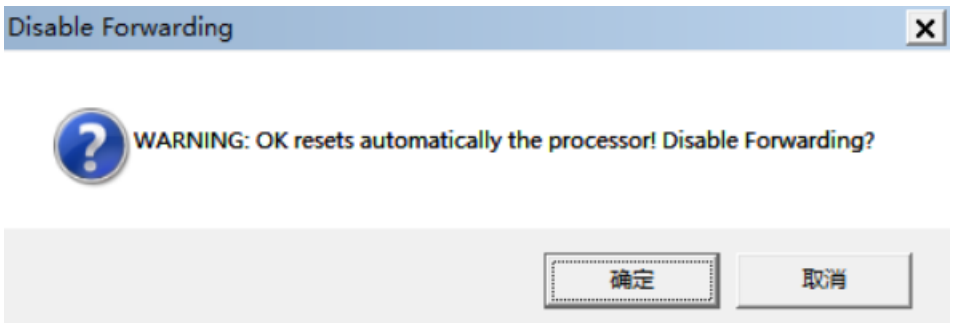
（4） 记录数据相关引起的暂停时钟周期数以及程序执行的总时钟周期数，计 算暂停时钟周期数占总执行周期数的百分比。

（5） 根据上面记录的数据，计算采用定向技术后性能提高的倍数。

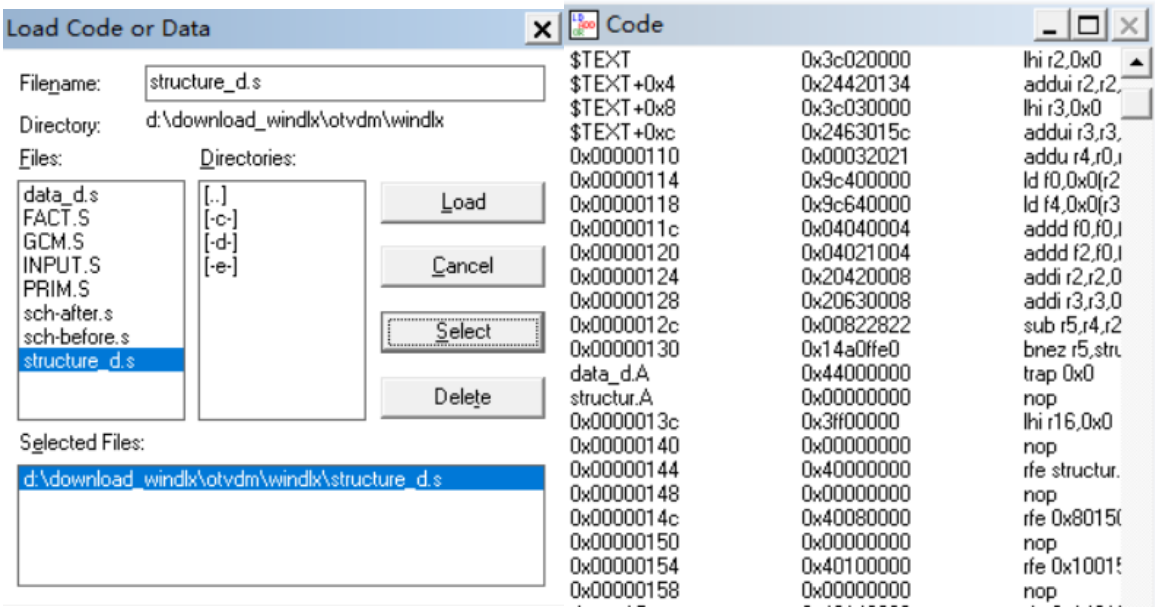
具体实验过程

一、 不采用定向技术

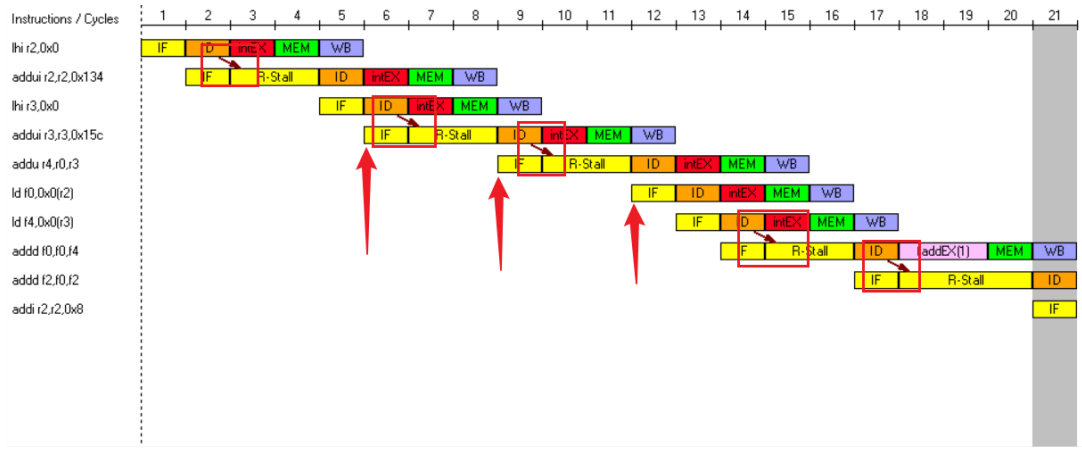
1. 首先通过 configuration 下拉单中的 Enable Forwarding 选项设置，设置流水线执行不适用定向技术。如下图所示



2. 之后加载运行对应的程序文件 data_d.s

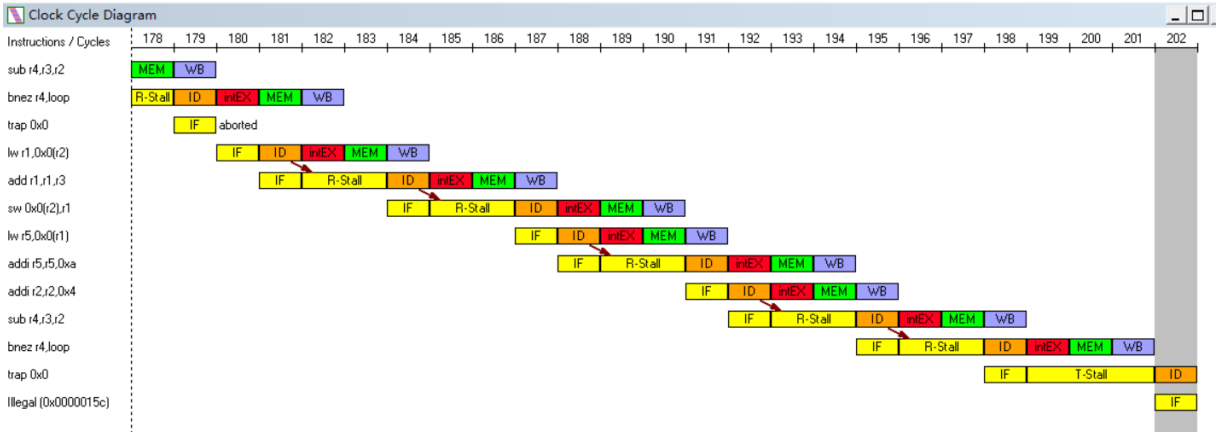


3. 之后单步跟踪程序的执行，观察指令执行时的流水线变化



可以看到，由于不适用定向技术，当后续执行中需要用到之前指令的执行结果时必须进行等待，导致流水线断流。上图显示的红色箭头则表示由于数据相关带来的延迟。

4. 按下 F5 连续执行，观察执行完毕所需要的指令周期数量。



可以看到，由于不适用定向技术，整个程序执行完毕需要花费 202 个始终周期。

进一步的，查看 statistics 窗口，分析指令执行的统计信息。

Total:
202 Cycle(s) executed.
ID executed by 85 Instruction(s).
2 Instruction(s) currently in Pipeline.

Hardware configuration:
Memory size: 32768 Bytes
faddEX-Stages: 1, required Cycles: 2
fmulEX-Stages: 1, required Cycles: 5
fdivEX-Stages: 1, required Cycles: 19
Forwarding disabled.

Stalls:
RAW stalls: 104 (51.48% of all Cycles)
WAW stalls: 0 (0.00% of all Cycles)
Structural stalls: 0 (0.00% of all Cycles)
Control stalls: 9 (4.46% of all Cycles)
Trap stalls: 3 (1.48% of all Cycles)
Total: 116 Stall(s) (57.42% of all Cycles)

Conditional Branches):
Total: 10 (11.76% of all Instructions), thereof:
taken: 9 (90.00% of all cond. Branches)
not taken: 1 (10.00% of all cond. Branches)

Load-/Store-Instructions:
Total: 30 (35.29% of all Instructions), thereof:
Loads: 20 (66.67% of Load-/Store-Instructions)
Stores: 10 (33.33% of Load-/Store-Instructions)

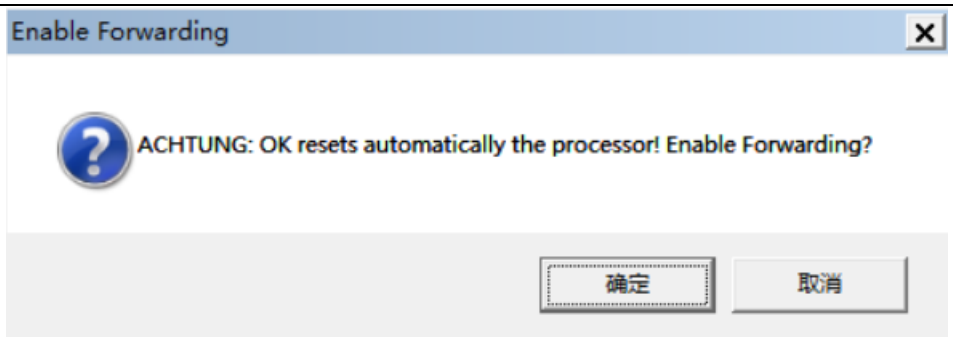
Floating point stage instructions:
Total: 0 (0.00% of all Instructions), thereof:
Additions: 0 (0.00% of Floating point stage inst.)
Multiplications: 0 (0.00% of Floating point stage inst.)
Divisions: 0 (0.00% of Floating point stage inst.)

Traps:
Traps: 1 (1.18% of all Instructions)

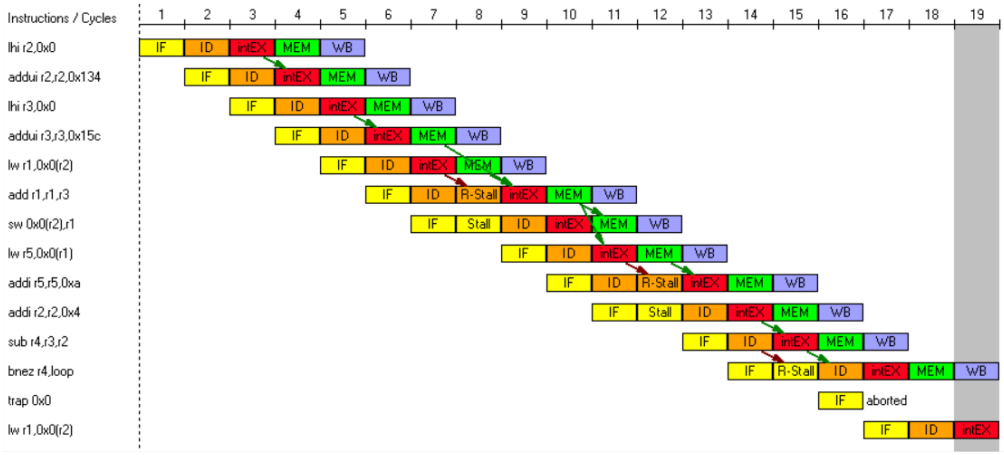
可以看到，因为数据相关而导致的周期延迟为 104 个周期，占总周期数的 51.48%。

二、 采用定向技术

1. 步骤同上，首先通过 configuration 下拉单中的 Enable Forwarding 选项设置，设置流水线执行适用定向技术。如下图所示



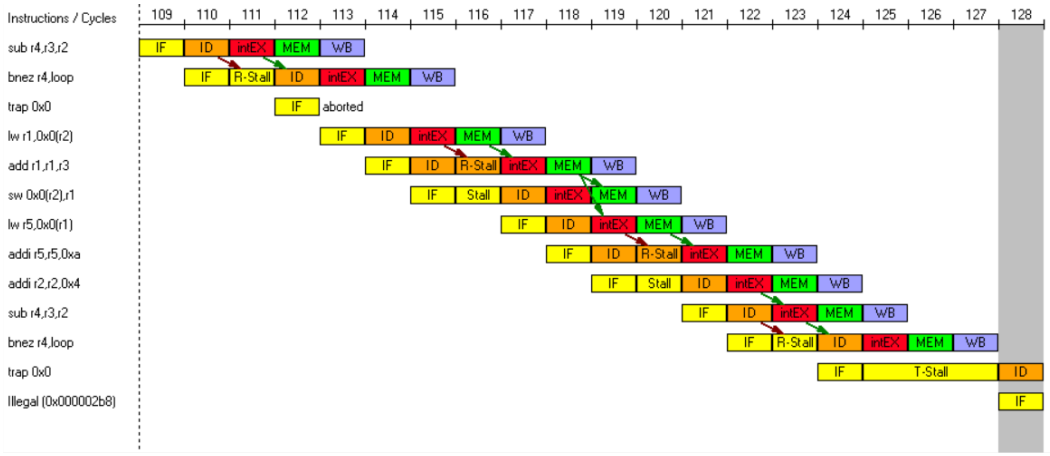
2. 之后单步跟踪，观察指令执行时流水线的执行情况。



可以看到，由于此时使用了定向技术，因此对于部分数据相关可以通过设置旁路的方法直接将数据进行传送。如上图中绿色的箭头所示。

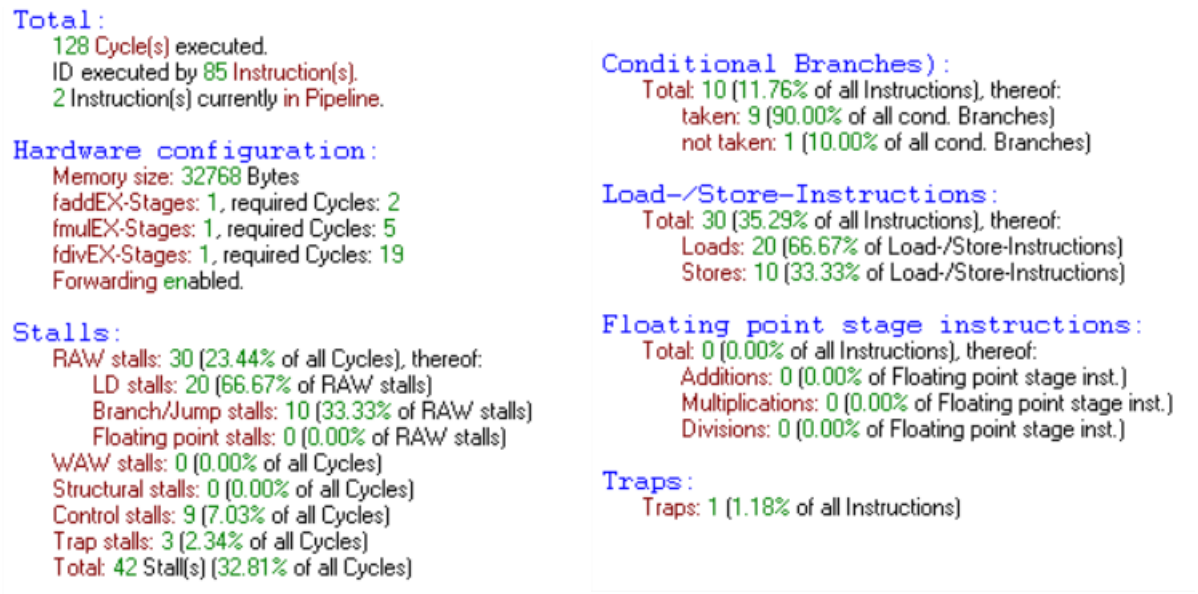
相比于不采用定向技术，采用定向技术减少了周期的暂停，加速了指令的执行

3. 按下 F5 连续执行，观察执行完毕所需要的指令周期数量。



可以看到，采用定向技术之后，整个程序只需要执行 128 个周期即可。相比于不采用定向技术节约了 76 个周期。在一定程度上减少了流水线的断流。

4. 点击 statistics 窗口，分析指令执行的统计信息。



可以看到，此时只有 23.44%的周期暂停，说说明定向技术确实可以提高指令执行的效率，提高系统的性能。

三、 定量计算

通过对比采用定向技术和不采用定向数的实验结果，可以定量的计算定向技术对系统优化的倍数。具体公式为

$$\text{优化倍数} = \frac{\text{不采用定向技术所需执行周期数} - \text{采用定向技术序偶所需执行周期数}}{\text{采用定向技术序偶所需执行周期数}}$$

带入数据可得

$$\text{优化倍数} = \frac{204 - 128}{128} = 59.4\%$$

计算采用定向技术后，减少了数据相关，缩短了程序的执行周期，性能提高的倍数约为 59.4%

结论分析与体会：

结论分析

1. 分析实验中的汇编代码

分析：汇编代码中首先为寄存器 R2 和 R3 赋初值，之后进入 LOOP 循环。在循环中寄存器 R2 的作用类似于循环变量，每一次递增 4 个字节，指向下一个地址单元。在 LOOP 循环中的第

一条指令，通过 LW 指令，实现了基址寻址，以（0）地址单元为基址，以 R2 寄存器的内容为偏移量取出对应地址单元的内容。在整个循环中寄存器 R4 用来控制循环的终止。根据代码*benz R4 LOOP*可知，当 R4 取值为零时，整个循环停止执行。

2. 对数据相关和定向技术的理解

分析：数据相关指的是当一条指令需要用到前面指令的执行结果，而这些指令均在流水线中重叠执 行时，就可能引起数据相关。而定向技术的主要思想是：在某条指令（如图 3.3.4 中的 ADD 指令）产生一个计算结果之前，其 它指令（如图 3.3.4 中的 SUB 和 AND 指令）并不真正需要该计算结果，如果能够将该计算结果从其产生的地方（寄存器文件 EX/MEM）直接送到其它指令需要它的地方（ALU 的输入寄存 器），那么就可以避免暂停。

3. 定向技术总结

分析：在本次实验中，定向技术的要点可以归纳为：

(1) 寄存器文件 EX/MEM 中的 ALU 的运算结果总是回送到 ALU 的输入寄存器。

(2) 当定向硬件检测到前一个 ALU 运算结果的写入寄存器就是当前 ALU 操作的源寄存器时，那么控制逻辑将前一个 ALU 运算结果定向到 ALU 的输入端，后一个 ALU 操作就不必从 源寄存器中读取操作数。

尽管采用定向技术可以缓解数据相关问题，从而提高 CPU 效率。但是不是所有的数据相关都可以用定向技术解 决，通过调整指令顺序和插入空指令也可以解决部分数据相关问题，在实际应用中需要根据使用场景选择合适的优化策略。

体会

通过本次实验，我进一步掌握了数据相关的基本知识，当指令在流水线中重叠执行时，流水线有可能改变指令读/写操作数的顺序，使得读/写操作 顺序不同于它们非流水实现的顺序，将导致数据相关。在程序的执行时，相关会导致指令的执行暂停，从而引起流水线的断流，减低系统的性能，通常可以采用插入空指令，编译优化调整执行的执行顺序或者定向技术等优化策略。

在本次实验中，主要采用了定向技术缓解数据相关的问题，通过设置旁路直接将数据进行传送。根据实验结果可知，采用定向技术后，程序执行的周期数从 204 缩短至 128，一定程度上的提升了系统的性能。

作为最后的总结，本次实验通过程序执行的具体案例分析了数据相关对指令执行的影

响，并且定量计算了优化前后系统的性能，使我更加深刻的感受到了使用定向技术优势。

附录

```
1.        LHI     R2, (A>>16) & 0xFFFF
2.        ADDUI   R2, R2, A & 0xFFFF
3.        LHI     R3, (B>>16)&0xFFFF
4.        ADDUI   R3, R3, B&0xFFFF
5. loop:
6.        LW      R1, 0 (R2)
7.        ADD     R1, R1, R3
8.        SW      0(R2), R1
9.        LW      R5, 0 (R1)
10.       ADDI    R5, R5, #10
11.       ADDI    R2, R2, #4
12.       SUB     R4, R3, R2
13.       BNEZ    R4, loop
14.       TRAP    #0
15.A: .word 0, 4, 8, 12, 16, 20, 24, 28, 32, 36
16.B: .word 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
```