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：	姓名：	班级：
实验题目：数据相关		
实验学时：2	实验日期：2024.5.31	

实验目的：

通过本实验，加深对数据相关的理解，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(一个基于 Windows 的 DLX 模拟器)

软件环境：

VMware Workstation 16 Player

Windows 7

实验步骤与内容：

实验内容：

(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。

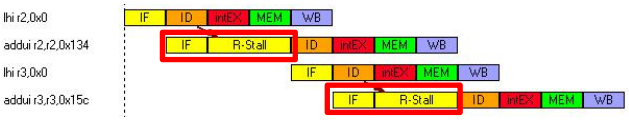
(5)根据上面记录的数据，计算采用定向技术后性能提高的倍数。

实验步骤：

（1）阅读汇编代码

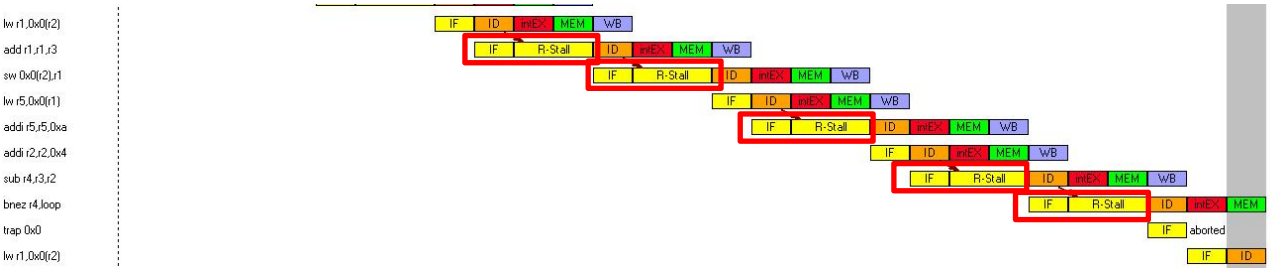
```
LHI R2, (A>>16) & 0xFFFF
ADDUI R2, R2, A & 0xFFFF
LHI R3, (B>>16)&0xFFFF
ADDUI R3, R3, B&0xFFFF
;将数组A的首地址加载到R2，数组B的首地址加载到R3
loop:
LW R1, 0 (R2)   ; 从R2地址指向的内存位置加载一个值到R1中。
ADD R1, R1, R3  ; 将R1中的值与R3地址指向的值相加
SW 0(R2), R1 ; 将结果存储回R2地址指向的位置
LW R5, 0 (R1)
ADDI R5, R5, #10 ; 将B数组每个元素 + 10 但不保存
ADDI R2, R2, #4  ; R2 + 4
SUB R4, R3, R2
BNEZ R4, loop
TRAP #0
A: .word 0, 4, 8, 12, 16, 20, 24, 28, 32, 36
B: .word 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
```

（2）关闭 Forwarding 选项，运行程序 data_d.s，分析数据相关：



可以看到的是在还未计算之前，程序初始化过程中存在两条 addui 指令发生数据相关，由于 lhi 指令在 WB 阶段才将值写回寄存器，这里可以理解为前一条指令在 WB 前半周器写回寄存器，下一条指令在后半周期读取寄存器。因此指令 addui r2，r2，0x134 和 addui r3，r3，0x15c 均无法正常执行译码阶段，在 IF 部件停留一个 Stalls。

继续执行程序



可以看到的是在一次循环过程中，会出现 5 次 R-stall，均是因为数据相关（即前一条指令的结果会被下一条指令使用到）

（3）查看 Statistics 窗口，分析指令执行的统计信息

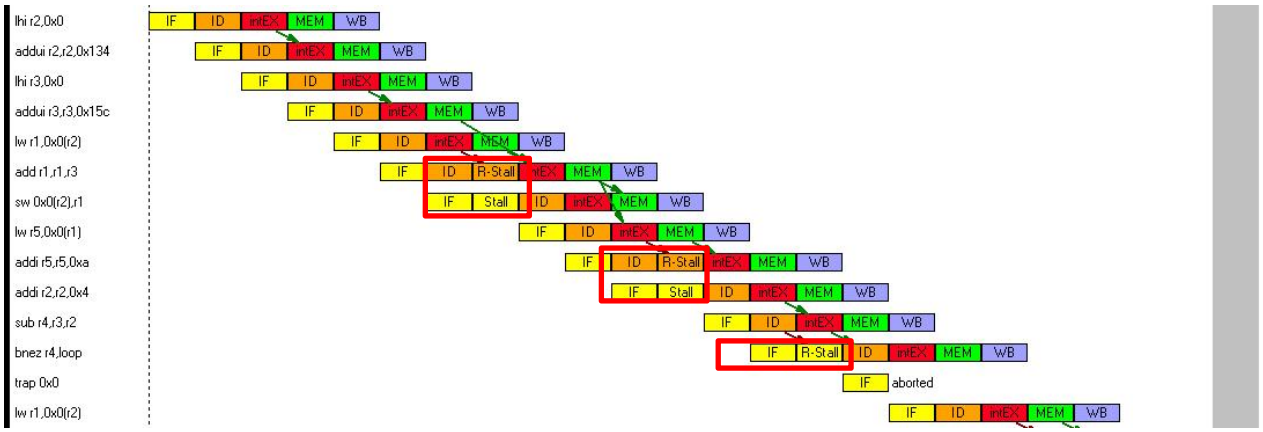
```
Total:
    202 Cycle(s) executed.
    ID executed by 85 Instruction(s).
    2 Instruction(s) currently in Pipeline.

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)
```

数据相关引起的暂停时钟周期数为 104（对应的是 RAW stalls，因为一次暂停会产生 2 个时钟周期的 R-Stall， 初始化有 2 次暂停，一次循环有 5 次暂停，一共循环 10 次）。程序执行的总时钟周期数为 202，暂停时钟周期数占总执行周期数的百分比为 104/202=51.48%

（4）采用定向技术，再次执行

首先通过 configuration 下拉单中的 Enable Forwarding 选项设置，设置流水线执行使用定

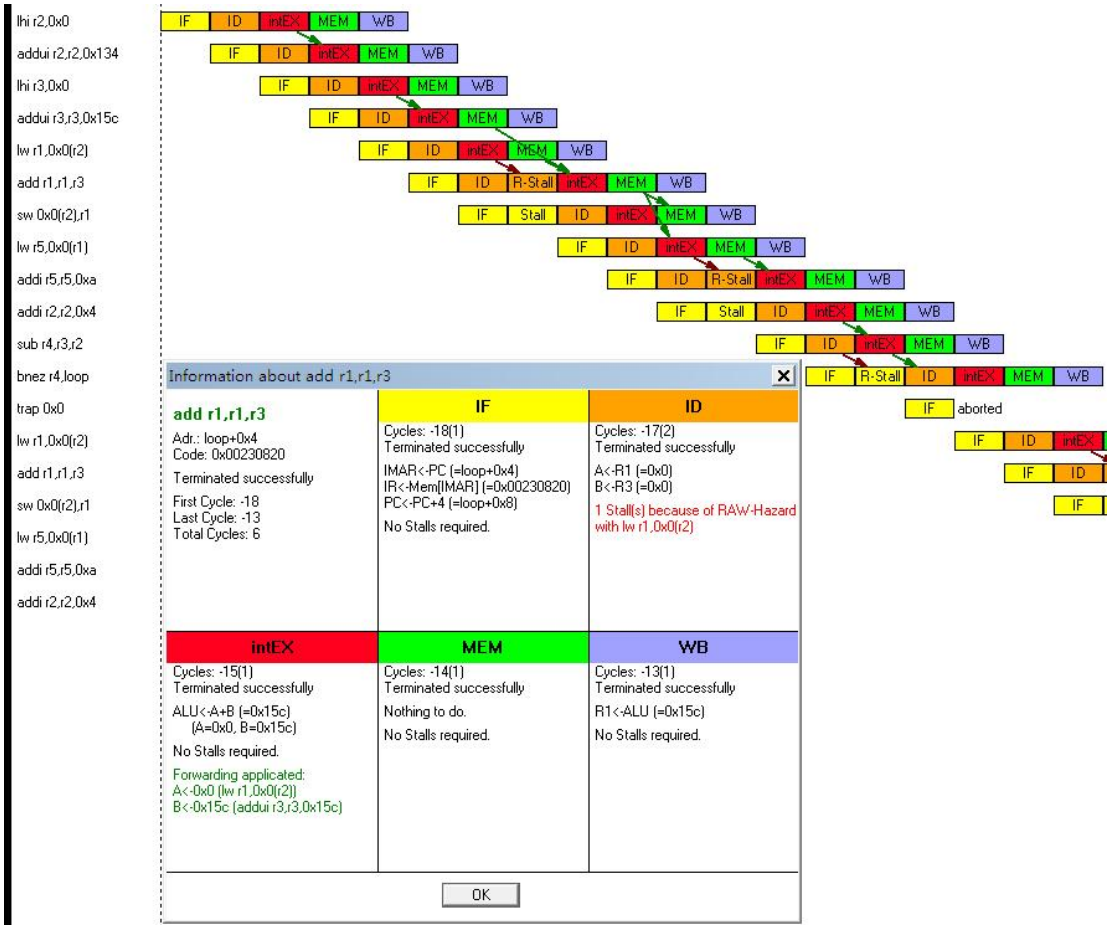
向技术，然后再次执行：



此时采用了定向技术，可以看到在初始化阶段产生的数据相关通过定向技术被消除了，lhi 指令的结果在 intEx 阶段计算得到后定向到 addui 指令的 intEX 阶段，也就是又重新送回到 intEx 部件。

同时指令 lw r1,0x0[r2]和指令 add r1,r1,r3 原有的数据相关被消除了，指令 add 不在 IF 阶段停留 2 个 Stall，但是有新的数据相关产生：指令 lw r1，0x0[r2]在 ID 阶段读取寄存器的值，在 EX 阶段计算访问地址，在 MEM 阶段访存获取数据，因此后面的指令 add 需在 ID 阶段停留 1 个 stall，等待指令 lw 的 MEM 阶段完成。另外 addui 指令通过定向得到寄存器 r3 的值。

可以通过点击指令 add r1，r1，r3 查看具体情况：



（5）执行结束后，查看 Statistics 界面

```
Total:
    128 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 enabled.

Stalls:
    RAW stalls: 30 (23.44% of all Cycles), thereof:
        LD stalls: 20 (66.67% of RAW stalls)
        Branch/Jump stalls: 10 (33.33% of RAW stalls)
        Floating point stalls: 0 (0.00% of RAW stalls)
    WAW stalls: 0 (0.00% of all Cycles)
    Structural stalls: 0 (0.00% of all Cycles)
    Control stalls: 9 (7.03% of all Cycles)
    Trap stalls: 3 (2.34% of all Cycles)
    Total: 42 Stall(s) (32.81% 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)
```

此时数据相关引起的暂停时钟周期数为 30（对应的是 RAW stalls，一次循环产生 3 个 Stall，一共循环 10 次）。而程序执行的总时钟周期数为 128，暂停时钟周期数占总执行周期数的百分比：30/128=23.44%。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。

（6）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，可以计算定向技术对系统性能优化的情况。具体公式为

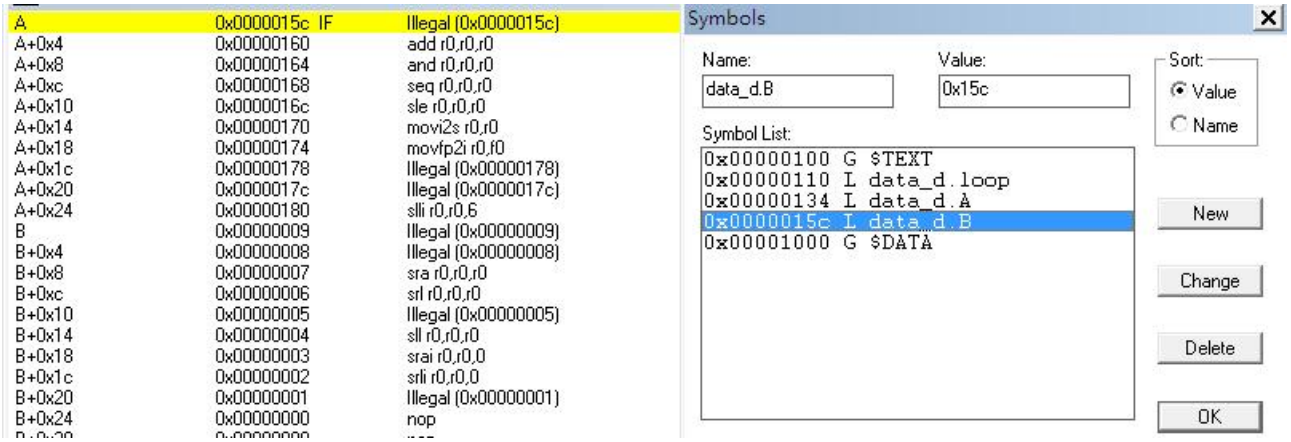
$$\text{优化倍数} = \frac{\text{不采用定向技术所需执行周期数} - \text{采用定向技术所需执行周期数}}{\text{采用定向技术所需执行周期数}}$$

带入数据可得

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

由此可见，采用定向技术后，减少了数据相关，缩短了程序的执行周期，大大提高了执行性能。

（7）查看程序结果



可以看到数组 A 存储了数组 B 各个元素对应的内存地址，数组 B 的内容没有改变。

结论分析与体会：

结论分析：

1. 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的操作数是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（如前面的 ADD 指令）产生一个计算结果之前，其它指令（如前面 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。

2. 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

3.实验中的汇编代码

在本实验中首先为寄存器 R2 和 R3 赋初值，将数组 A 的首地址存到 R2，数组 B 的首地址存到 R3。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，每一次增加 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。而当整个执行结束后，数组 A 存储了数组 B 各个元素对应的内存地址，数组 B 的内容没有改变。

体会：

通过本次实验，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，进而将导致数据相关。而在程序的执行时，数据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，我们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。总的来说，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。