计算机体系结构课程实验

实验平台（同济大学）

1. 本实验的实验环境在阿里云上提供，地址：

<https://developer.aliyun.com/adc/series/labs-tongji-css/>

1. 进入实验前，要先在阿里云上进行注册和认证。
2. 登录后，单击要进行的实验。
3. 按网页中右侧的说明“免费开通”ECS（单击“免费开通”），建立实验服务器。
4. 按说明连接实验服务器。
5. 连接成功后，屏幕上会显示远程桌面。
6. 进入桌面上“comparch-simulator” 目录，即可找到各个实验所需的模拟器。双击模拟器便可执行之。

目录

**实验一：MIPS指令系统和MIPS体系结构**

**实验二：流水线及流水线中的冲突**

**实验三：指令调度和延迟分支**

**实验四：Cache性能分析**

**附录：MIPSsim使用手册**

实验一 MIPS指令系统和MIPS体系结构

**1.1 实验目的**

1. 了解和熟悉指令级模拟器。
2. 熟练掌握MIPSsim模拟器的操作和使用方法。
3. 熟悉MIPS指令系统及其特点，加深对MIPS指令操作语义的理解。
4. 熟悉MIPS体系结构。

**1.2 实验平台**

实验平台采用指令级和流水线操作级模拟器MIPSsim（在目录MIPSsim里）。

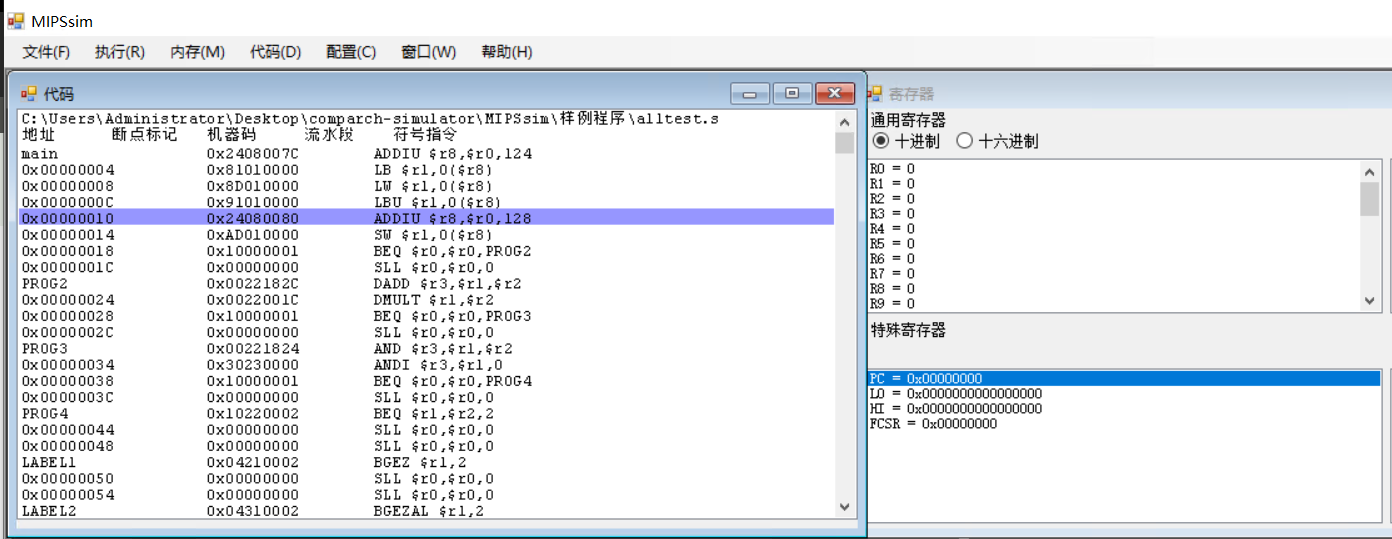
设计：张晨曦教授，版权所有。

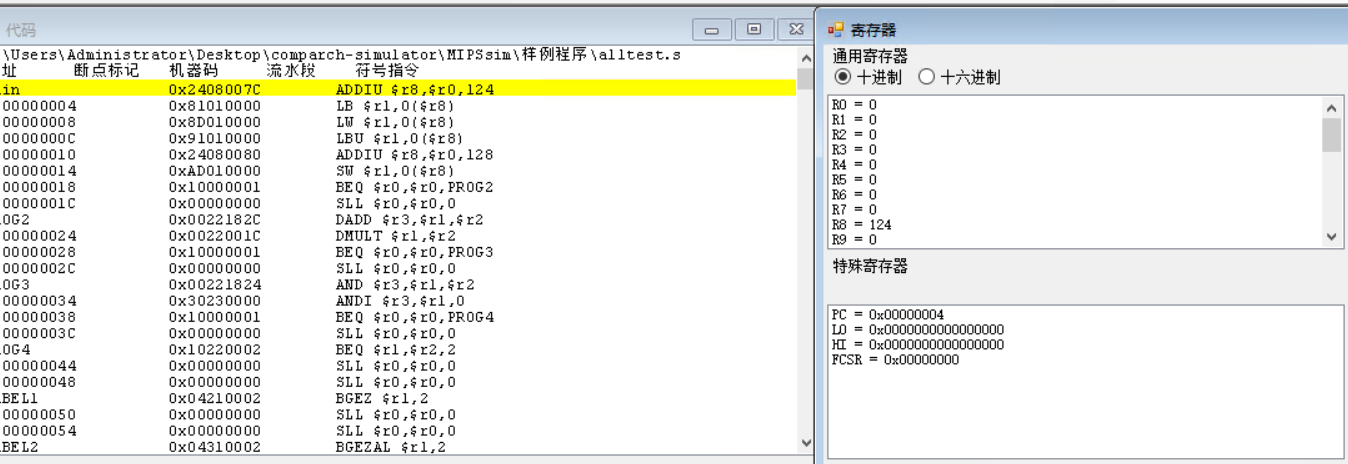
**1.3 实验内容和步骤**

首先要阅读MIPSsim模拟器的使用方法（见附录：MIPSsim使用手册），然后了解MIPSsim的指令系统。

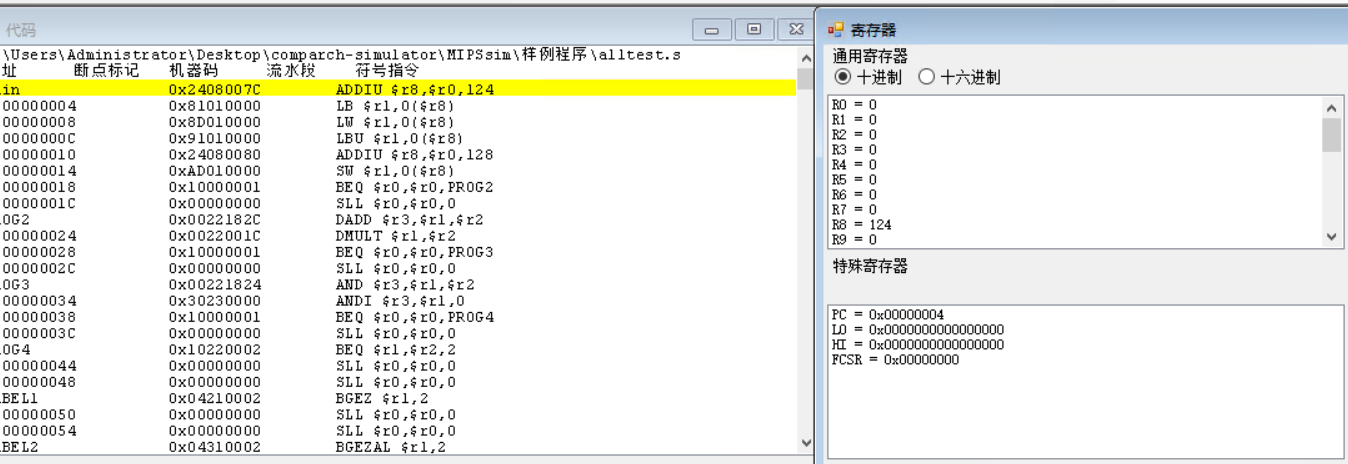
1. 启动MIPSsim（双击MIPSsim.exe）。
2. 点击“配置”→“流水方式”，使模拟器工作在非流水方式下。
3. 阅读附录，熟悉MIPSsim模拟器的操作和使用方法。

可以先载入一个样例程序（在本模拟器所在的文件夹下的“样例程序”文件夹中），然后分别以单步执行一条指令、执行多条指令、连续执行、设置断点等的方式运行程序，观察程序的执行情况，观察CPU中寄存器和存储器的内容的变化。

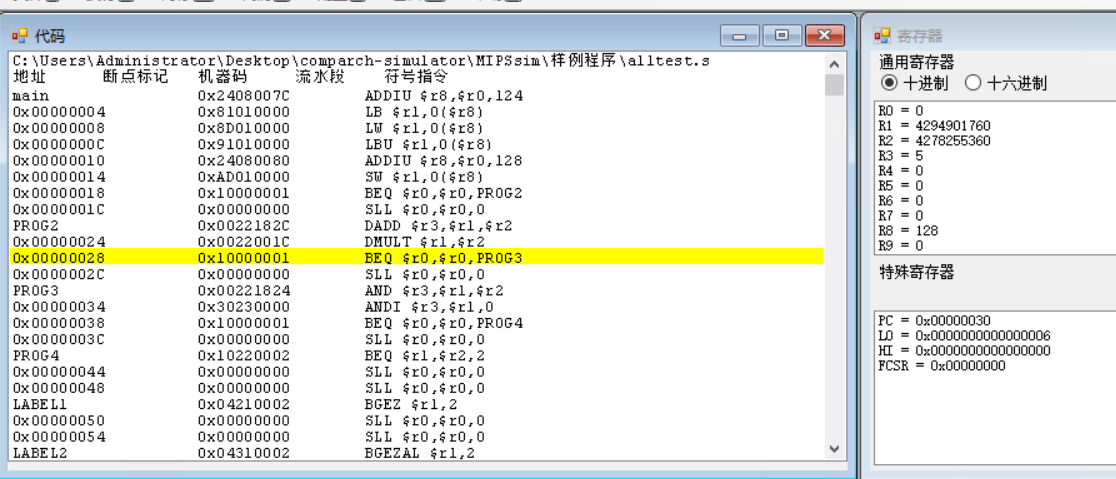
1. 选择“文件”→“载入程序”选项，加载样例程序alltest.asm，然后查看“代码”窗口，查看程序所在的位置（起始地址为0x00000100）。
2. 查看“寄存器”窗口PC寄存器的值：[PC] = 0x00000000 。
3. 
4. 执行load和store指令。步骤如下：
   * 1. 单步执行一条指令（F7）。
     2. 下一条指令地址为0x00000004 ， 是一条 有 （有，无）符号载入 字节 （字节，半字，字）指令。



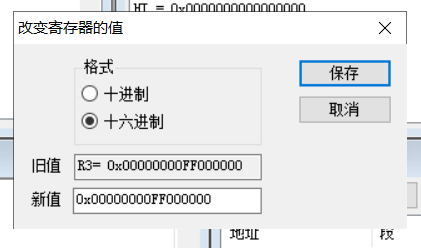
* + 1. 单步执行1条指令（F7）。
    2. 查看R1的值，[R1]= 0xFFFFFFFFFFFFFF80\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。
    3. 下一条指令地址为0x00000008 ，是一条 有 （有，无）符号载入 字 （字，半字，字）指令。
    4. 单步执行1条指令。
    5. 查看R1的值，[R1]= 0x\_\_00000000000080\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_。
    6. 下一条指令地址为0x0000000C ，是一条 有 （有，无）符号载入 字 （字，半字，字）指令。
    7. 单步执行1条指令。
    8. 查看R1的值，[R1]= 0x00000000000080\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。
    9. 单步执行1条指令。
    10. 下一条指令地址为0x0x00000014 ，是一条保存 字 （字，半字，字）指令。
    11. 单步执行1条指令。
    12. 查看内存BUFFER处字的值，值为0x0x00000080\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 。



1. 执行算术运算类指令。步骤如下：
2. 双击“寄存器”窗口里的R1，将其值修改为2。
3. 双击“寄存器”窗口里的R2，将其值修改为3。
4. 单步执行1条指令。
5. 下一条指令地址为0x0x00000020 ， 是一条加法指令。
6. 单步执行1条指令。
7. 查看R3的值，[R3]= 0x0000000000000005\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_。

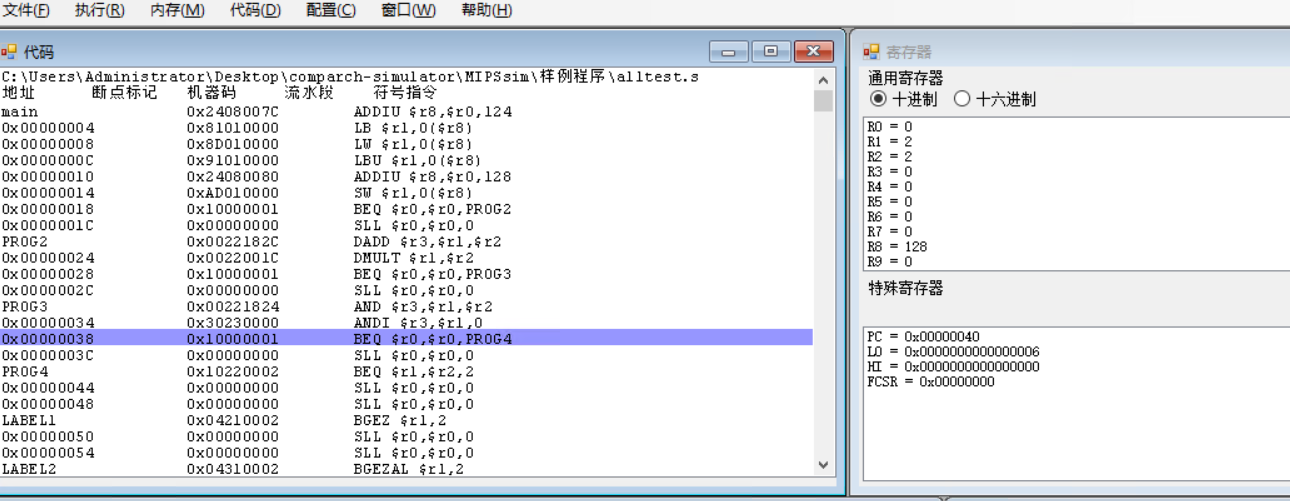


1. 下一条指令地址为0x0x00000024 ，是一条乘法指令。
2. 单步执行1条指令。
3. 查看LO、HI的值，[LO]= 0x000000000000006\_\_\_，[HI]=0x0000000000000000 。
4. 执行逻辑运算类指令。步骤如下：
   1. 双击“寄存器”窗口里的R1，将其值修改为0xFFFF0000。
   2. 双击“寄存器”窗口里的R2，将其值修改为0xFF00FF00。
   3. 单步执行1条指令。
   4. 下一条指令地址为0x00000030 ，是一条逻辑与运算指令，第二个操作数寻址方式是 寄存器直接寻址 （寄存器直接寻址，立即数寻址）。
   5. 单步执行1条指令。
   6. 查看R3的值，[R3]= 0x00000000FF000000\_。



* 1. 下一条指令地址为：0x00000034 ，是一条逻辑与指令，第二个操作数寻址方式是 立即数寻址 （寄存器直接寻址，立即数寻址）。
  2. 单步执行1条指令。
  3. 查看R3的值，[R3]= 0x0000000000000000 \_\_\_\_\_\_\_\_。

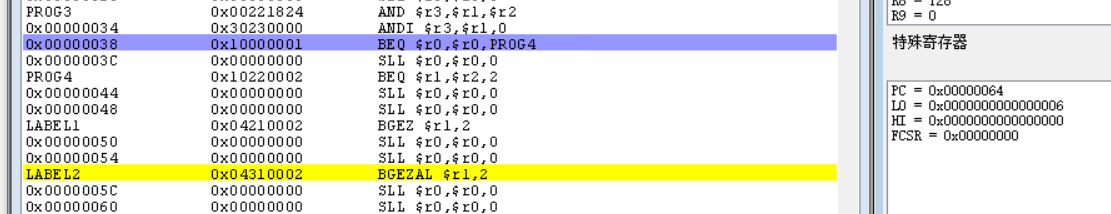
1. 执行控制转移类指令。步骤如下：
2. 双击“寄存器”窗口里的R1，将其值修改为2。
3. 双击“寄存器”窗口里的R2，将其值修改为2。
4. 单步执行1条指令。
5. 下一条指令地址为0x00000040 ，是一条BEQ指令，其测试条件是： Regs[R1]==Regs[R2] ，目标地址为0x0000004C 。



1. 单步执行1条指令。
2. 查看PC的值，[PC]=0x0000004C ，表明分支 成功 （成功，失败）。
3. 下一条指令地址是一条BGEZ指令，其测试条件是 Regs[R1]>=0 ，目标地址为0x00000058 。

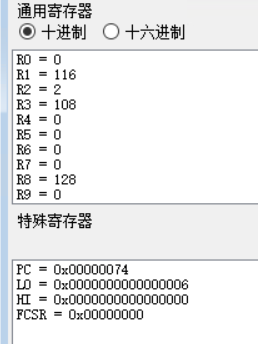


1. 单步执行1条指令。
2. 查看PC的值，[PC]=0x00000058 ，表明分支 成功 （成功，失败）。
3. 下一条指令地址是一条BGEZAL指令，其测试条件是 Regs[R1]>=0 ，目标地址为0x00000064 。
4. 单步执行1条指令。
5. 查看PC的值，[PC]=0x00000064 ，表明分支 成功 （成功，失败）；查看R31的值，[R31]= 0x000000000000005C \_\_\_\_\_\_\_。



1. 单步执行1条指令。
2. 查看R1的值，[R1]= 0x0000000000000074 \_\_\_\_\_\_\_。
3. 下一条指令地址为0x00000068 ，是一条JALR指令，保存目标地址的寄存器为R1 \_，保存返回地址的目标寄存器为R3 。
4. 单步执行1条指令。
5. 查看PC和R3的值，[PC]=0x00000074 ，

[R3]= 0x000000000000006C \_\_\_\_\_\_\_。



实验二 流水线及流水线中的冲突

**2.1 实验目的**

1. 加深对计算机流水线基本概念的理解。
2. 理解MIPS结构如何用5段流水线来实现，理解各段的功能和基本操作。
3. **加深对**数据冲突、结构冲突**的理解，理解**这两类冲突对CPU性能的影响。
4. 进一步理解解决数据冲突的方法，**掌握如何应用定向技术来减少数据冲突引起的停顿。**

**2.2 实验平台**

实验平台采用指令级和流水线操作级模拟器MIPSsim。

设计：张晨曦教授，版权所有。

**2.3 实验内容和步骤**

首先要掌握MIPSsim模拟器的使用方法。详见附录：MIPSsim使用手册。

1. 启动MIPSsim。

2．根据预备知识中关于流水线各段操作的描述，进一步理解流水线窗口中各段的功能，掌握各流水寄存器的含义。（双击各段，就可以看到各流水寄存器的内容）

3. 熟悉MIPSsim模拟器的操作和使用方法。

可以先载入一个样例程序（在本模拟器所在的文件夹下的“样例程序”文件夹中），然后分别以单步执行一个周期、执行多个周期、连续执行、设置断点等的方式运行程序，观察程序的执行情况，观察CPU中寄存器和存储器的内容的变化，特别是流水寄存器内容的变化。

4. 选择配置菜单中的“流水方式”，使模拟器工作于流水方式下。

5．观察程序在流水线中的执行情况，步骤如下：

1. 选择MIPSsim的“文件”→“载入程序”选项来加载pipeline.s（在模拟器所在文件夹下的“样例程序”文件夹中）。
2. 关闭定向功能。这是通过在“配置”→“定向”（使该项前面没有“√”号）来实现的。
3. 用单步执行一周期的方式（“执行”菜单中）或用F7执行该程序，观察每一周期中，各段流水寄存器内容的变化、指令的执行情况（“代码”窗口）以及时钟周期图。
4. 当执行到第13个时钟周期时，各段分别正在处理的指令是：

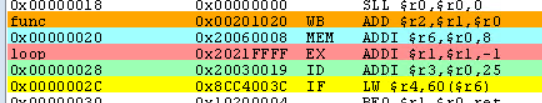
IF： LW $r4,60($r6)

ID： ADDI $r3,$r0,25

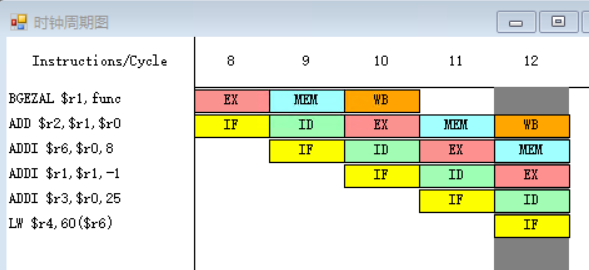
EX： ADDI $r1,$r1,-1

MEM： ADDI $r6,$r0,8

WB： ADD $r2,$r1,$r0



画出这时的时钟周期图。



6. 这时各流水寄存器中的内容为：

IF/ID.IR：\_0x8CC4003C\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_

IF/ID.NPC：\_0x00000030\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

ID/EX.A：\_0x00000000000000\_\_\_\_\_\_\_\_\_\_\_\_

ID/EX.B：\_0x00000000000000\_\_\_\_\_\_\_\_\_\_\_\_ \_

ID/EX.Imm：\_0x00000000000019\_\_\_\_\_\_\_\_\_\_\_\_

ID/EX.IR：\_0x20030019\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_

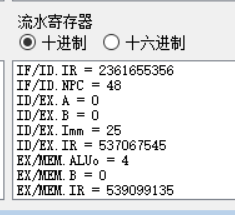
EX/MEM.ALUo：\_0x0000000000000004\_\_\_\_\_\_\_\_\_\_\_\_

EX/MEM.IR：\_0x2021FFFF\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_

MEM/WB.LMD：\_0x0000000000000000\_\_\_\_\_\_\_\_\_\_\_\_

MEM/WB.ALUo：\_0x0000000000000008\_\_\_\_\_\_\_\_\_\_\_\_

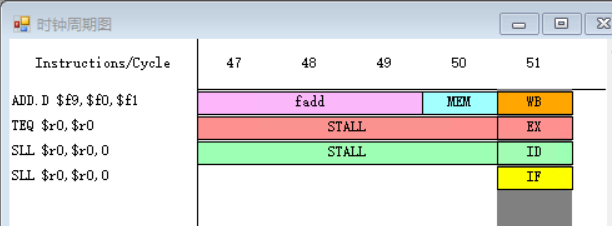
MEM/WB.IR：\_0x20060008\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_



7. 观察和分析结构冲突对CPU性能的影响，步骤如下：

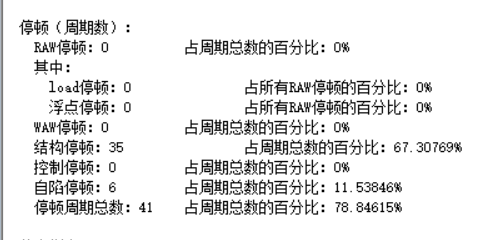
（1）加载structure\_hz.s（在模拟器所在文件夹下的“样例程序”文件夹中）。

（2）执行该程序，找出存在结构冲突的指令对以及导致结构冲突的部件。



(3) 记录由结构冲突引起的停顿时钟周期数，计算停顿时钟周期数占总执行周期数的百分比；

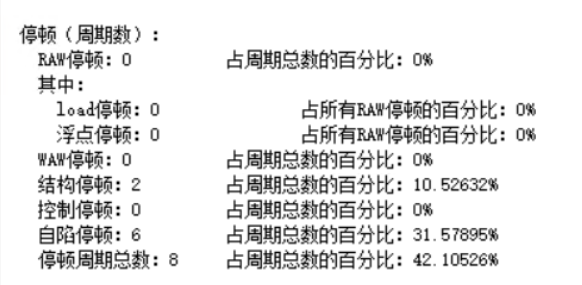
总周期数52个，结构停顿周期数35个，占总执行周期数的67.30769%



（4）把浮点加法器的个数改为4个。



（5）再次重复上述（1）～（3）的工作。



（6）分析结构冲突对CPU性能的影响，讨论解决结构冲突的方法。

发生结构冲突时，流水线会停顿，造成CPU性能下降。

解决方法：（1） 暂停一个时钟周期，取后一条指令操作;（2）设置两个独立的存储器分别存放操作数和指令；（3）采取指令预存技术

8. 观察数据冲突并用定向技术来减少停顿，步骤如下：

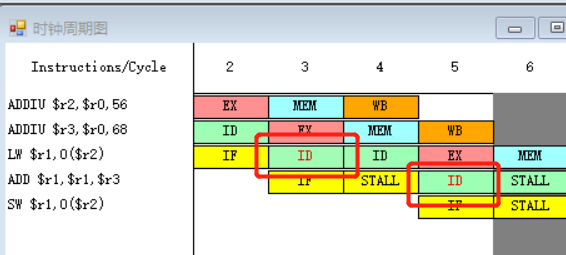
（1）全部复位。

（2）加载data\_hz.s（在模拟器所在文件夹下的“样例程序”文件夹中）。

（3）关闭定向功能。这是通过在“配置”→“定向”（使该项前面没有“√”号）来实现的。

（4）用单步执行一个周期的方式（F7）执行该程序，同时查看时钟周期图，列出在什么时刻发生了RAW（先写后读）冲突。

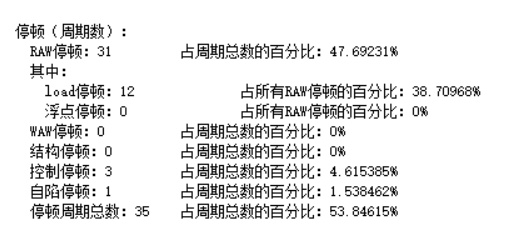
RAW冲突发生时刻：4、6、7、9、10、13、14、17、18、20、21、25、26、28、29、32、33、36、37、39、40、44、45、47、48、51、52、55、56、58、59



（5）记录数据冲突引起的停顿时钟周期数以及程序执行的总时钟周期数，计算停顿时钟周期数占总执行周期数的百分比。

RAW停顿：31   总周期数：65

 占周期总数的百分比：47.69231%

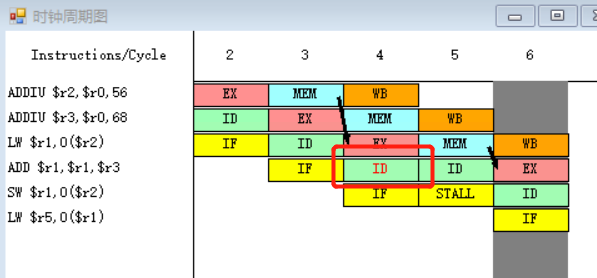


（6）复位CPU。

（7）打开定向功能。这是通过在“配置”→“定向”（使该项前面有一个“√”号）来实现的。

（8）用单步执行一周期的方式（F7）执行该程序，同时查看时钟周期图，列出在什么时刻发生了RAW（先写后读）冲突，并与（3）的结果进行比较；

RAW冲突发生时刻：5、10、13、18、22、25、30、34、37

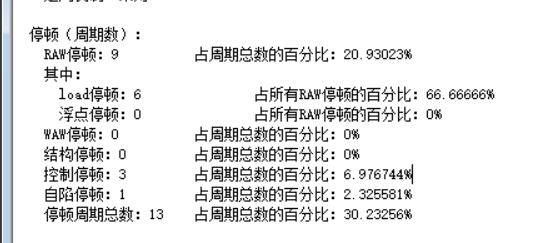


（9）记录数据冲突引起的停顿时钟周期数以及程序执行的总时钟周期数。计算采用定向技术后性能提高的倍数。

RAW停顿：9    总周期数：43

占周期总数的百分比：20.93023%

性能提升倍数=65/43=1.512



实验三 指令调度和延迟分支

**3.1 实验目的**

1. **加深对指令调度技术的理解。**
2. **加深对延迟分支技术的理解。**
3. 熟练掌握用**指令调度技术来解决流水线中的**数据冲突的方法**。**
4. **进一步理解指令调度技术对CPU性能的改进。**
5. **进一步理解延迟分支技术对CPU性能的改进。**

**3.2 实验平台**

实验平台采用指令级和流水线操作级模拟器MIPSsim。

设计：张晨曦教授，版权所有。

**3.3 实验内容和步骤**

首先要掌握MIPSsim模拟器的使用方法。详见附录：MIPSsim使用手册。

1. 启动MIPSsim。

2. 根据教材中关于流水线各段操作的描述，进一步理解流水线窗口中各段的功能，掌握各流水寄存器的含义（双击各段，就可以看到各流水寄存器的内容）。

3. 选择“配置”→“流水方式”选项，使模拟器工作于流水方式下。

4. 用指令调度技术解决流水线中的结构冲突与数据冲突。

1. 启动MIPSsim。
2. 用MIPSsim的“文件”→“载入程序”选项来加载schedule.asm（在模拟器所在文件夹下的“样例程序”文件夹中）。
3. 关闭定向功能。这是通过在“配置”菜单中关闭“定向”（使该项前面没有“√”号）来实现的。
4. 执行所载入的程序。通过查看统计数据和时钟周期图，找出并记录程序执行过程中各种冲突发生的次数、发生冲突的指令组合，以及程序执行的总时钟周期数。

总时钟周期数：33

各种冲突发生的次数：

RAW停顿：16，占周期总数的百分比：48.48485%，其中load停顿：6，占所有RAW停顿的百分比：37.5%

自陷停顿：1，占周期总数的百分比：3.030303%

停顿周期总数：17，占周期总数的百分比：51.51515%

发生冲突的指令组合有：

[ADDIU $r1,$r0,56，LW $r2,0($r1)],

[LW $r2,0($r1) ， ADD $r4,$r0,$r2],

[ADD $r4,$r0,$r2 ， SW $r4,0($r1)],

[SW $r4,0($r1) ， LW $r6,4($r1)],

[LW $r6,4($r1)， ADD $r8,$r6,$r1],

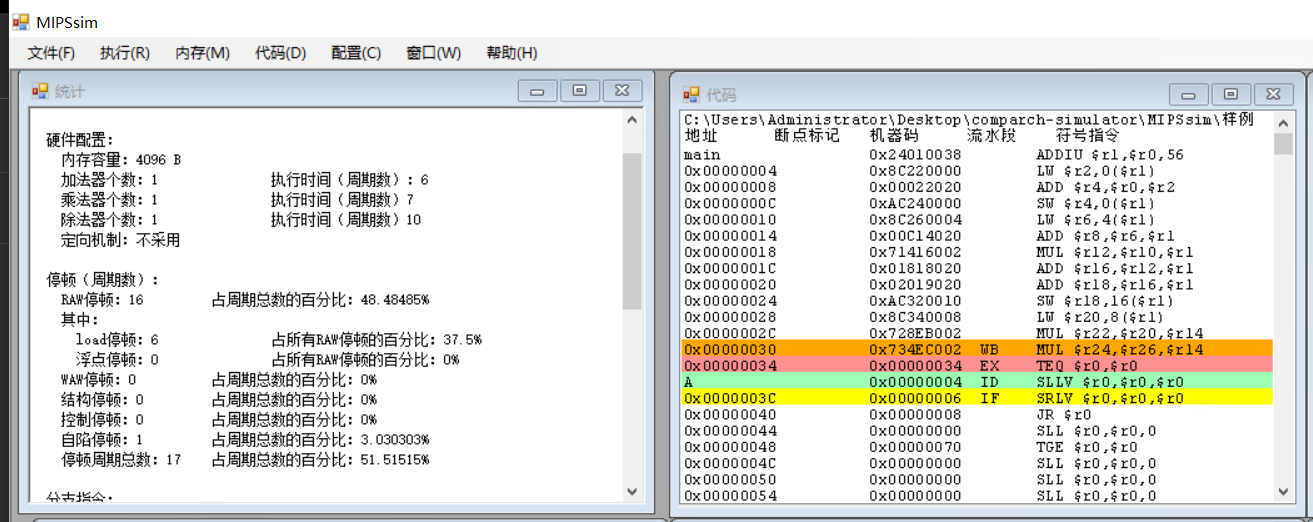
[MUL $r12,$r10,$r1，ADD $r16,$r12,$r1],

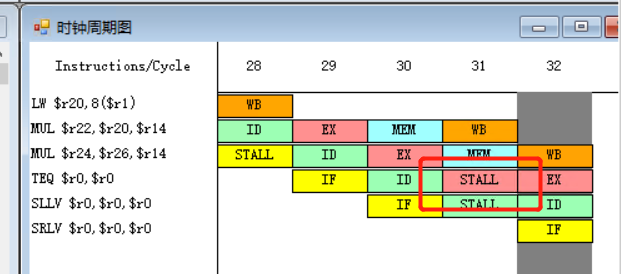
[ADD $r16,$r12,$r1 ， ADD $r18,$r16,$r1],

[ADD $r18,$r16,$r1 ， SW $r18,16($r1)],

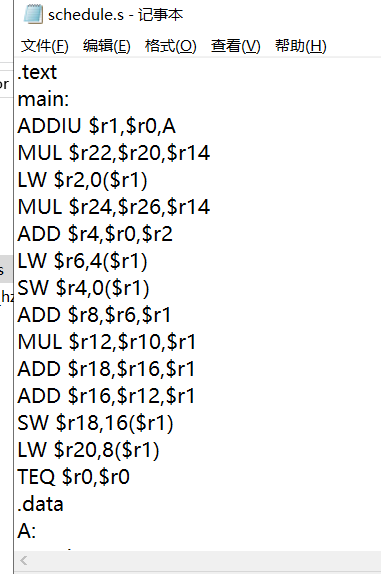
[LW$r20,8($r1)，MUL$r22,$r20,$r14],

[MUL $r22,$r20,$r14 ， MUL $r24,$r26,$r14]

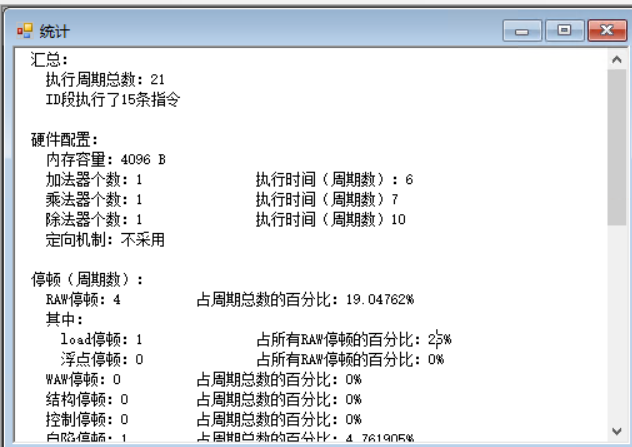




1. 采用指令调度技术对程序进行指令调度，消除冲突。将调度后的程序保持到after-schedule.asm中。



1. 载入after-schedule.asm。
2. 执行该程序，观察程序在流水线中的执行情况，记录程序执行的总时钟周期数；



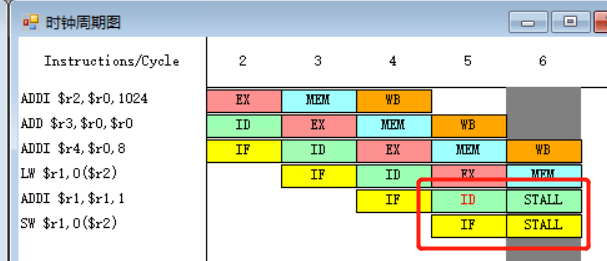
1. 根据记录结果，比较调度前和调度后的性能。论述指令调度对于提高CPU性能的作用。

调度前执行总周期数为33个周期，调度后执行总周期数为21周期，减少了12周期，证明调度后提高了CPU使用率，减少了冲突次数，提高了CPU性能。33/21=1.57倍性能提升。

5. 用延迟分支减少分支指令对性能的影响。

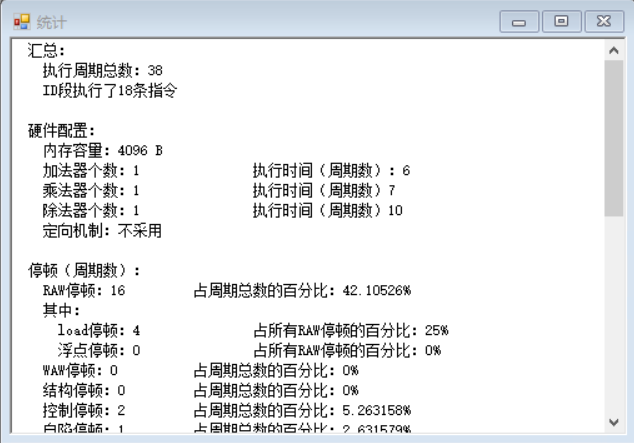
1. 启动MIPSsim。
2. 载入branch.asm。
3. 关闭延迟分支功能。这是通过在“配置”→“延迟槽”选项来实现的。
4. 执行该程序。观察并记录发生分支延迟的时刻~~。~~

分支延迟的时刻：Cycle 6, 9, 13, 21, 24, 28.

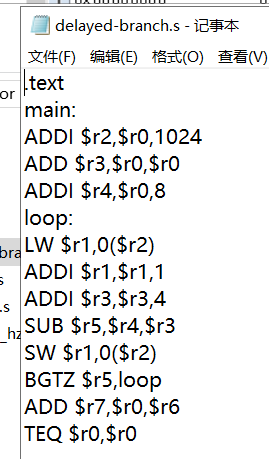


1. 记录执行该程序所花的总时钟周期数。

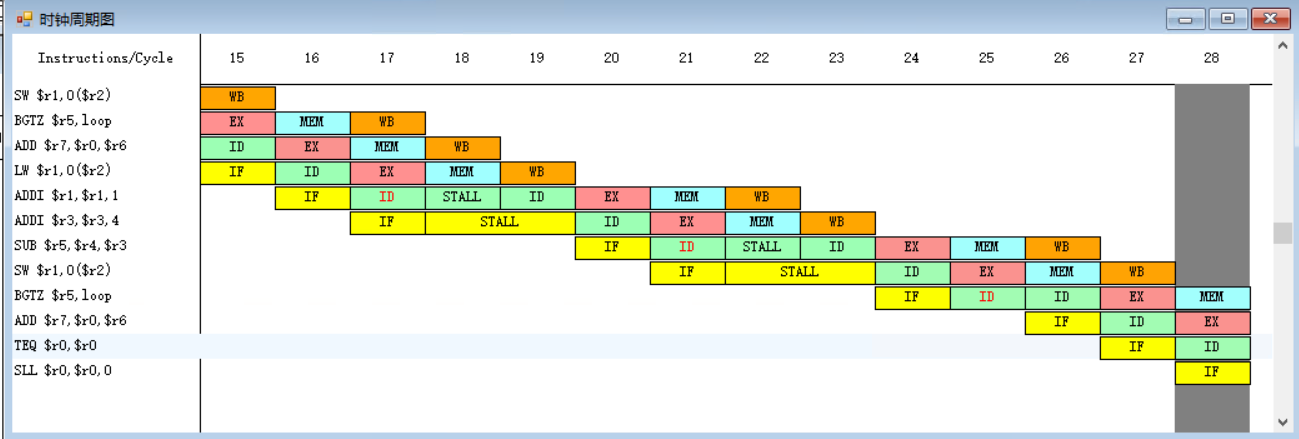
执行周期总数：38，ID段执行了18条指令



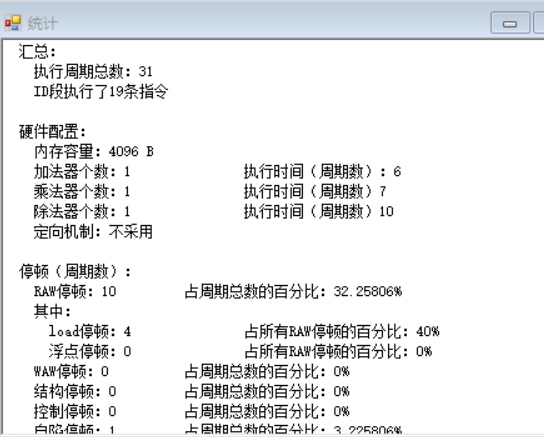
1. 假设延迟槽为一个，对branch.asm进行指令调度，然后保存到“delayed-branch.asm”中。



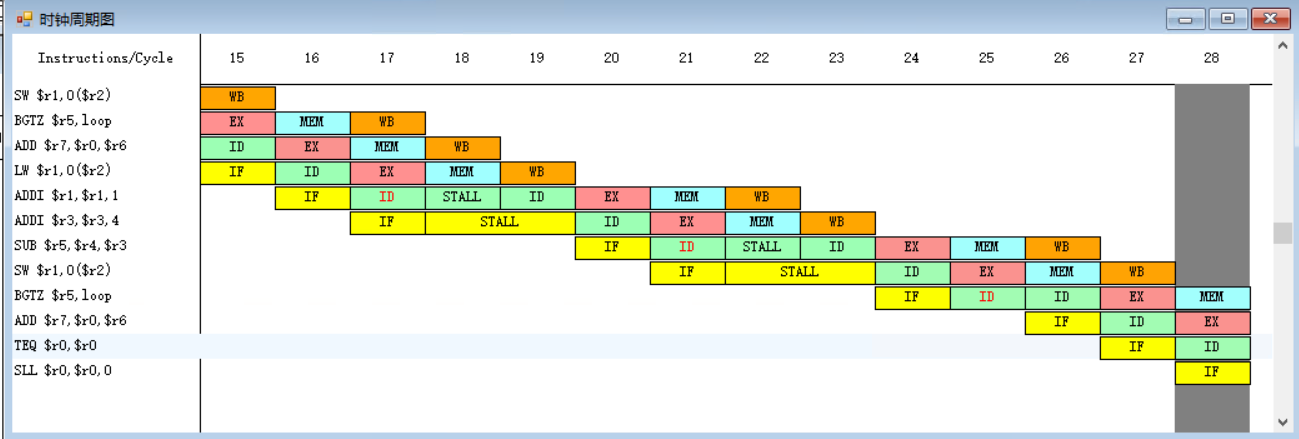
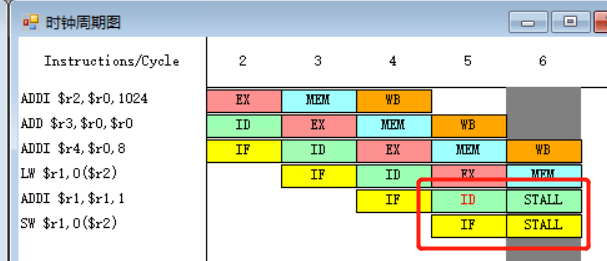
1. 载入delayed-branch.asm。
2. 打开延迟分支功能。
3. 执行该程序，观察其时钟周期图。



1. 记录执行该程序所花的总时钟周期数。



1. 对比上述两种情况下的时钟周期图。



1. 根据记录结果，比较没采用延迟分支和采用了延迟分支的性能。论述延迟分支对于提高CPU性能的作用。

比较两种情况的时钟周期总数，可知：在使用延迟槽后，指令在运行到跳转指令时，不会出现延迟等待，则能够提高CPU的性能。

实验四 Cache性能分析

**4.1 实验目的**

1. 加深对Cache的基本概念、基本组织结构以及基本工作原理的理解。
2. 掌握Cache容量、相联度、块大小对Cache性能的影响。
3. 掌握降低Cache不命中率的各种方法以及这些方法对提高Cache性能的好处。
4. 理解LRU与随机法的基本思想以及它们对Cache性能的影响。

**4.2 实验平台**

实验平台采用Cache模拟器：myCache模拟器（在目录MyCache里）。

设计：张晨曦教授，版权所有。

**4.3 实验内容及步骤**

首先要掌握myCache模拟器的使用方法（见4.4节）。

**4.3.1 Cache容量对不命中率的影响**

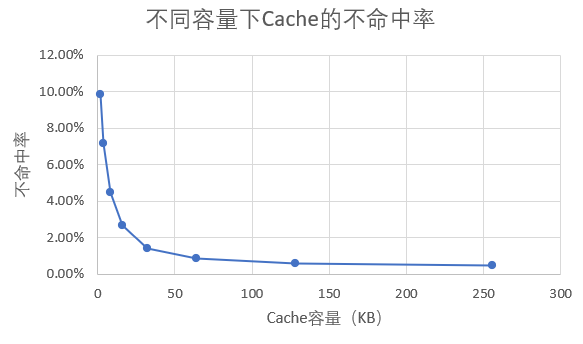
1. 双击启动myCache模拟器。
2. 用鼠标点击“复位”按钮，把各参数设置为默认值。
3. 选择一个地址流文件。方法：选择“访问地址”→“地址流文件”选项，然后点击“浏览”按钮，从本模拟器所在的文件夹下的“地址流”文件夹中选取。
4. **选择不同的Cache容量**，包括：2KB，4KB，8KB，16KB，32KB，64KB，128KB，256KB，分别执行模拟器（单击“执行到底”按钮即可执行），然后在表4.1中记录各种情况下的不命中率。

表4.1 不同容量下Cache的不命中率

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cache容量（KB） | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
| 不命中率 | 9.87% | 7.19% | 4.48% | 2.65% | 1.42% | 0.89% | 0.60% | 0/49% |

地址流文件名： all.din

1. 以容量为横坐标，画出不命中率随Cache容量变化而变化的曲线。并指明地址流文件名。



1. 根据该模拟结果，你能得出什么结论？

Cache的不命中率随着Cache容量的增大而降低，降低幅度随Cache的增大而递减，当容量增大到一定程度后，再增大Cache容量的不命中率的变化较不显著。

**4.3.2 相联度对不命中率的影响**

1. 用鼠标单击“复位”按钮，把各参数设置为默认值。此时的Cache容量为64KB。
2. 选择一个地址流文件。方法：选择“访问地址”→“地址流文件”选项，然后单击“浏览”按钮，从本模拟器所在的文件夹下的“地址流”文件夹中选取。
3. **选择不同的Cache相联度**，包括：直接映象，2路，4路，8路，16路，32路，分别执行模拟器（单击“执行到底”按钮即可执行），然后在表4.2中记录各种情况下的不命中率。

表4.2 当容量为64KB时，不同相联度下Cache的不命中率

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 相联度 | 1 | 2 | 4 | 8 | 16 | 32 |
| 不命中率 | 0.89% | 0.53% | 0.47% | 0.45% | 0.44% | 0.44% |

地址流文件名： all.din

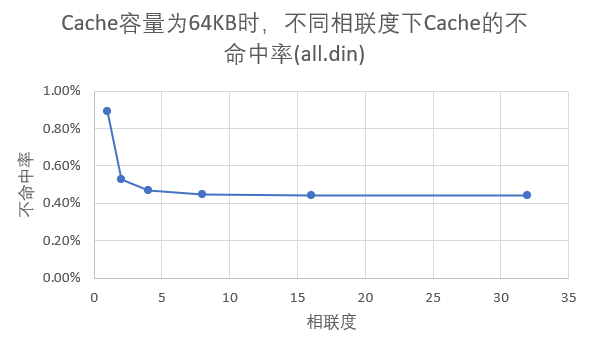
1. 把Cache的容量设置为256KB，重复(3)的工作，并填写表4.3。

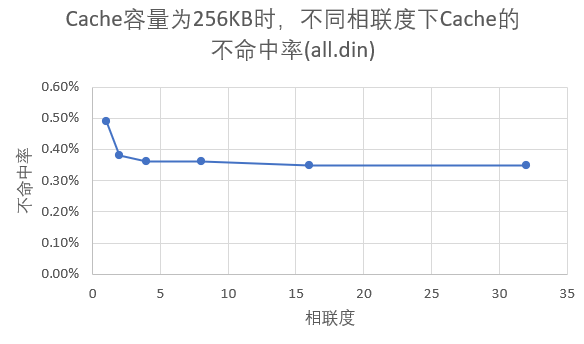
表4.3 当容量为256KB时，不同相联度下Cache的不命中率

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 相联度 | 1 | 2 | 4 | 8 | 16 | 32 |
| 不命中率 | 0.49% | 0.38% | 0.36% | 0.36% | 0.35% | 0.35% |

地址流文件名： all.din

1. 以相联度为横坐标，画出在64KB和256KB的情况下不命中率随Cache相联度变化而变化的曲线。并指明地址流文件名。





1. 根据该模拟结果，你能得出什么结论？

Cache容量大小相同的情况下，提高Cache的相联度可以降低Cache不命中率，同时，Cache容量越小，相联度对命中率效果的提升就越显著。

提升幅度随Cache相联度的增大而递减，当相联度增大到一定程度后，再增大Cache相联度的不命中率的变化较不显著。

**4.3.3 Cache块大小对不命中率的影响**

1. 用鼠标单击“复位”按钮，把各参数设置为默认值。
2. 选择一个地址流文件。方法：选择“访问地址”→“地址流文件”选项，然后单击“浏览”按钮，从本模拟器所在的文件夹下的“地址流”文件夹中选取。
3. **选择不同的Cache块大小**，包括：16B，32B，64B，128B，256B，**对于Cache的各种容量**，包括：2KB，8KB，32KB，128KB，512KB，分别执行模拟器（单击“执行到底”按钮即可执行），然后在表4.4中记录各种情况下的不命中率。

表4.4 各种块大小情况下Cache的不命中率

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 块大小  （B） | Cache容量（KB） | | | | |
| 2 | 8 | 32 | 128 | 512 |
| 16 | 12.02% | 5.79% | 1.86% | 0.95% | 0.71% |
| 32 | 9.87% | 4.48% | 1.42% | 0.60% | 0.42% |
| 64 | 9.36% | 4.03% | 1.20% | 0.43% | 0.27% |
| 128 | 10.49% | 4.60% | 1.08% | 0.35% | 0.20% |
| 256 | 13.45% | 5.35% | 1.19% | 0..34% | 0.16% |

0.

地址流文件名： all.din

1. 分析Cache块大小对不命中率的影响。

Cache容量大小相同的情况下，当Cache容量小于128KB，随着Cache块大小的增大，不命中率先降低后提高，当Cache容量大于等于128KB，随着Cache块大小的增大，不命中率一直降低。

增加块的大小会产生两重作用：（1）增加空间局部性，减少强制性不命中；（2）增加块大小会减少Cache中块的数目所以有可能会增加冲突不命中。

Cache容量较大时，（1）的作用超过（2）的作用，使不命中率持续下降；当Cache容量较小时，（2）的作用超过（1）的作用，使得不命中率上升。

**4.3.4 替换算法对不命中率的影响**

1. 用鼠标单击“复位”按钮，把各参数设置为默认值。
2. 选择一个地址流文件。方法：选择“访问地址”→“地址流文件”选项，然后单击“浏览”按钮，从本模拟器所在的文件夹下的“地址流”文件夹中选取。
3. **对于不同的替换算法、Cache容量和相联度**，分别执行模拟器（单击“执行到底”按钮即可执行），然后在表4.5中记录各种情况下的不命中率。

表4.5 LRU和随机替换法的不命中率的比较

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Cache  容量 | 相 联 度 | | | | | |
| 2 路 | | 4 路 | | 8 路 | |
| LRU | 随机算法 | LRU | 随机算法 | LRU | 随机算法 |
| 16KB | 1.71 | 2.30% | 1.33% | 2.37% | 1.21% | 2.81% |
| 0.64KB | 0.53 | 0.78% | 0.47% | 0.82% | 0.45% | 0.91% |
| 256KB | 0.38 | 0.41% | 0.36% | 0.37% | 0.36% | 0.36% |
| 1MB | 0.35 | 0.35% | 0.35% | 0.35% | 0.35% | 0.35% |

地址流文件名： all.din

1. 分析不同的替换算法对Cache不命中率的影响。

Cache容量较小时，LRU法的不命中率均低于随机算法，但当Cache容量大于等于256KB之后，LRU法的不命中率与随机算法的不命中率相差无几。相联度对于两种算法的不命中率影响不大。

**4.4 myCache模拟器使用方法**

1. 启动模拟器：用鼠标双击myCache模拟器.exe。

2. 系统会打开一个操作界面。该界面的左边为设置模拟参数区域，右边为模拟结果显示区域。如图4.1所示。

3. 可以设置的参数包括：是统一Cache还是分离Cache，Cache的容量，块大小，相联度，替换算法，预取策略，写策略，写不命中时的调块策略。可以直接从列表里选择。

4. 访问地址可以选择来自地址流文件，也可以选择手动输入。如果是前者，则可以通过点击“浏览”按钮，从模拟器所在文件夹下面的“地址流”文件夹中选取地址流文件（.din文件），然后进行执行。执行的方式可以是步进，也可以是一次执行到底。如果选择手动输入，就可以在“执行控制”区域中输入块地址，然后点击“访问”按钮。系统会在界面的右边显示访问类型、地址、块号以及块内地址。

5. 模拟结果包括：

（1）访问总次数，总的不命中次数，总的不命中率；

（2）读指令操作的次数，其不命中次数及其不命中率；

（3）读数据操作的次数，其不命中次数及其不命中率；

（4）写数据操作的次数，其不命中次数及其不命中率；

（5）手动输入单次访问的相关信息。



图4.1 MyCache模拟器的操作界面示意图