

计算机体系结构 课程实验报告

学号: 202222130195	姓名: 阎发祥	班级: 22.3
实验题目: 用 WinDLX 模拟器完成求素数程序		
实验学时: 2		实验日期: 2024. 4. 30
<b>实验目的:</b> 通过实验, 熟练掌握 WINDLX 的操作方法, 特别注意在单步执行 WinDLX 程序中, 流水线中指令的节拍数。		
<b>硬件环境:</b> WinDLX(一个基于 Windows 的 DLX 模拟器)		
<b>软件环境:</b> VMware Workstation 16 Player		
<b>实验步骤与内容:</b>		
<b>实验内容:</b>		
(1) 用 WinDLX 模拟器执行求素数程序 prim.s。这个程序计算若干个整数的素数。		
(2) 单步执行两轮程序, 求出素数 2 和 3。		
(3) 在执行程序过程中, 注意体验单步执行除法和乘法指令的节拍数, 并和主菜单 configuration/floating point stages 中的各指令执行拍数进行比较。		
<b>实验步骤:</b>		
1. 汇编代码分析 求解素数的汇编代码在 prim.s 中, 经过分析可知, 该代码主要由 6 个部分组成, 分别是:		
(1) main 函数部分 (2) NextValue 代码部分 (3) Loop 循环部分 (4) IsPrim 代码部分 (5) IsNoPrim 代码部分 (6) Finish 代码部分		
其中 main 函数部分是整个程序的入口, 在此处还对后续使用到的寄存器进行了初始化, 具体代码如下所示:		
<pre> 21      main: 22          ;*** Initialization 23          addi    r1, r0, 0      ;Index in Table 24          addi    r2, r0, 2      ;Current value </pre>		
之后, 定义 NextValue 代码段, 这部分主要作用是将 r3 置 0, 用于后面的 Loop 循环。		

```
27      NextValue: addi    r3, r0, 0 ; Help index in Table
```

在 Loop 循环中，先将 r3 与 r1 进行比较，判断素数表是否遍历完。之后每一次循环判断一个数是否是素数，需要调用 IsPrim 函数和 IsNoPrim 函数，而判断的依据就是将一个数先除一个素数再乘上看是否还等于其本身，因为如果一个数不能被另一个数整除的话，与它相除就会有精度丢失，如果相等的话就说明它能被整除，既不是素数。

```
28  ▼Loop:  
29      seq r4, r1, r3 ; 比较 r1 和 r3 是否相等 (Table 结束了)  
30      bnez r4, IsPrim ; 如果相等，说明没有能整除 r2 的素数 → r2 是素数  
31  
32      lw r5, Table(r3) ; r5 = Table[r3]，取出之前找到的素数  
33      divu r6, r2, r5 ; r6 = r2 / r5  
34      multu r7, r6, r5 ; r7 = (r2 / r5) * r5  
35      subu r8, r2, r7 ; r8 = r2 - r7，如果为0，说明整除  
36  
37      beqz r8, IsNoPrim ; 如果能整除，r2 不是素数 → 跳到 IsNoPrim  
38  
39      addi r3, r3, 4 ; 否则继续检查下一个素数  
40      j Loop
```

如果判断一个数为素数时，就需要跳转到 IsPrim 代码段。在 IsPrim 代码段中主要是将一个素数存入到素数表中，同时需要修改 r1，因为 r1 表示的是素数表的长度。还需判断素数表长度是否为 count，如果为 count 就说明已经满了，结束即可。

```
43  ▼IsPrim:  
44      sw Table(r1), r2 ; 把素数 r2 保存到 Table[r1]  
45      addi r1, r1, 4 ; 索引+4 (下一个位置)  
46      ;*** 'Count' reached?  
47      lw r9, Count ; r9 = Count  
48      srl r10, r1, 2 ; r10 = r1 / 4 → 当前找到的素数个数  
49      sge r11, r10, r9 ; 比较 r10 >= r9  
50      bnez r11, Finish ; 如果已经找到 Count 个素数，结束程序
```

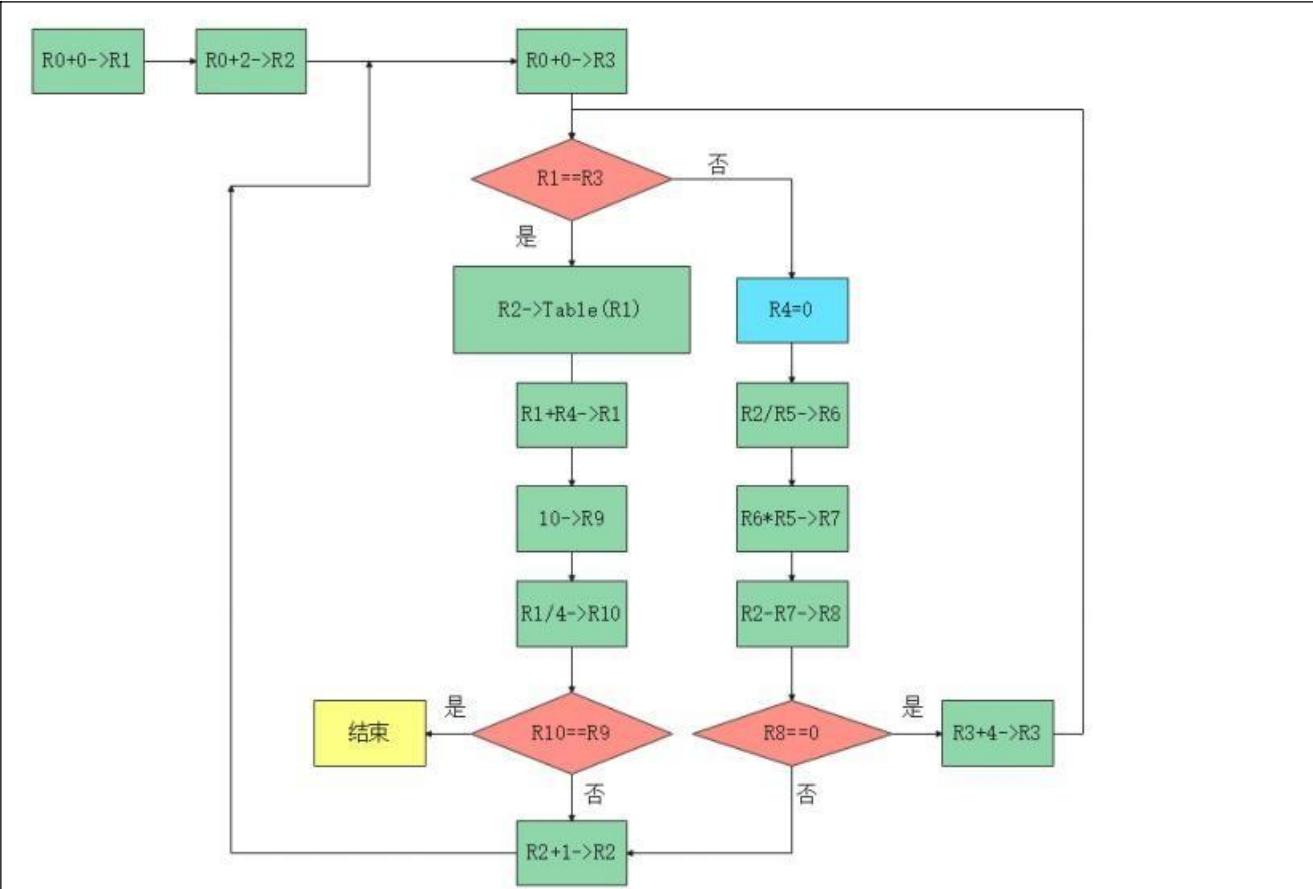
当在 Loop 循环中已经判断出一个数不为素数时，即可从 Loop 循环中跳出到 IsNoPrim 代码段中，在该段主要是将 r2 的值加 1，即判断下一个数是否为素数。

```
48  ▼IsNoPrim: ;*** Check next value  
49      addi r2, r2, 1 ; increment R2  
50      j NextValue
```

当求出 10 个素数后即可结束，执行 trap 0 触发系统调用结束即可。

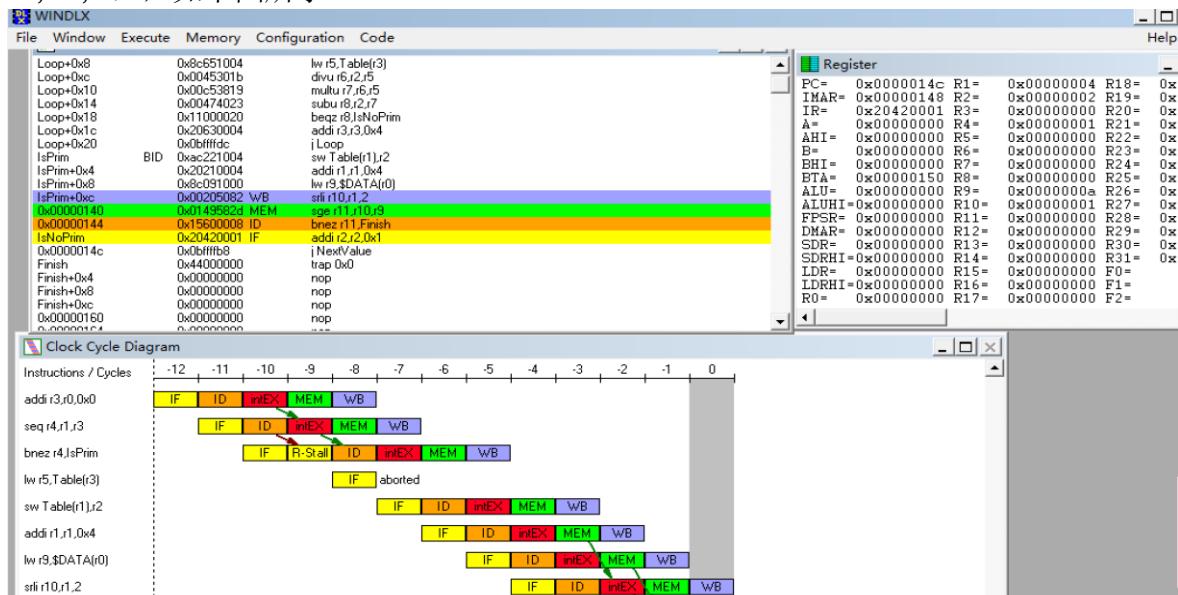
```
52  ▼Finish: ;*** end  
53      trap 0
```

## 2. 指令执行流程



### 3. 装载程序具体执行

**求第一个素数 2:** 在isPrim处设置断点，继续单步执行，直到设置断点的指令执行完毕。此时，素数2已被写入table中，继续单步执行，此时table的下标+“1”，可以看到r1的值变成了4（addi r1, r1, 4），如下图所示



逐过程分析：

- (1) 首先是将 r1 和 r3 都置为 0，然后将 r2 置为 2（最小的素数），之后执行指令 seq r4, r1, r3 判断 r1 是否等于 r3，在此处 r1 等于 r3 成功跳转，在 WinDLX 模拟器的 5 级流水线中跳转指令在 ID 阶段确定是否成功跳转，成功跳转后下一条读入的指令被 aborted。

(2) 由于前面执行了跳转指令，于是接下来就执行指令 `sw Table[r1]`，`r2` 将 `r2 = 2` 写入素数表中，同时 `r1 = r1 + 4`，因为 `r1` 记录的是素数表的长度，`int` 类型占 4 个字节。

(3) `lw r9, $DATA[0]` 是将数据区的 0 号元素也就是 `count = 10` 存入到 `r9` 中，指令 `srli r10, r1, 2` 是将 `r1` 右移 2 位，相当于除 4，即为当前素数表的长度，之后判断 `r10` 与 `r9` 的大小关系，如果 `r10` 等于 `r9` 即找到 10 个素数，程序结束。

(4) 若 `r10` 小于 `r9` 的话，将 `r2` 自增，重复上述过程，直到 `r10` 等于 `r9` 为止。

当求出素数 2 以后可以查看 register 窗口，发现此时 `r1 = 4` 表示当前素数表中有一个元素，`r2` 等于 2 说明当前正在判断 2 是否为素数。

PC= 0x00000150	R12= 0x00000000	F8= 0 D8= 0
IMAR= 0x0000014c	R13= 0x00000000	F9= 0 D10= 0
IR= 0xbfffffb8	R14= 0x00000000	F10= 0 D12= 0
A= 0x00000002	R15= 0x00000000	F11= 0 D14= 0
AHI= 0x00000000	R16= 0x00000000	F12= 0 D16= 0
B= 0x00000000	R17= 0x00000000	F13= 0 D18= 0
BHI= 0x00000000	R18= 0x00000000	F14= 0 D20= 0
ETA= 0x00000000	R19= 0x00000000	F15= 0 D22= 0
ALU= 0x00000000	R20= 0x00000000	F16= 0 D24= 0
ALUHI= 0x00000000	R21= 0x00000000	F17= 0 D26= 0
FPSR= 0x00000000	R22= 0x00000000	F18= 0 D28= 0
DMAR= 0x00000000	R23= 0x00000000	F19= 0 D30= 0
SDR= 0x00000000	R24= 0x00000000	F20= 0
SDRHI= 0x00000000	R25= 0x00000000	F21= 0
LDR= 0x00000000	R26= 0x00000000	F22= 0
LDRHI= 0x00000000	R27= 0x00000000	F23= 0
R0= 0x00000000	R28= 0x00000000	F24= 0
R1= 0x00000004	R29= 0x00000000	F25= 0
R2= 0x00000002	R30= 0x00000000	F26= 0
R3= 0x00000000	R31= 0x00000000	F27= 0
R4= 0x00000001	F0= 0	F28= 0
R5= 0x00000000	F1= 0	F29= 0
R6= 0x00000000	F2= 0	F30= 0
R7= 0x00000000	F3= 0	F31= 0
R8= 0x00000000	F4= 0	D0= 0
R9= 0x0000000a	F5= 0	D2= 0
R10= 0x00000001	F6= 0	D4= 0
R11= 0x00000000	F7= 0	D6= 0

## 求第二个素数 3

此次 register 窗口如下：

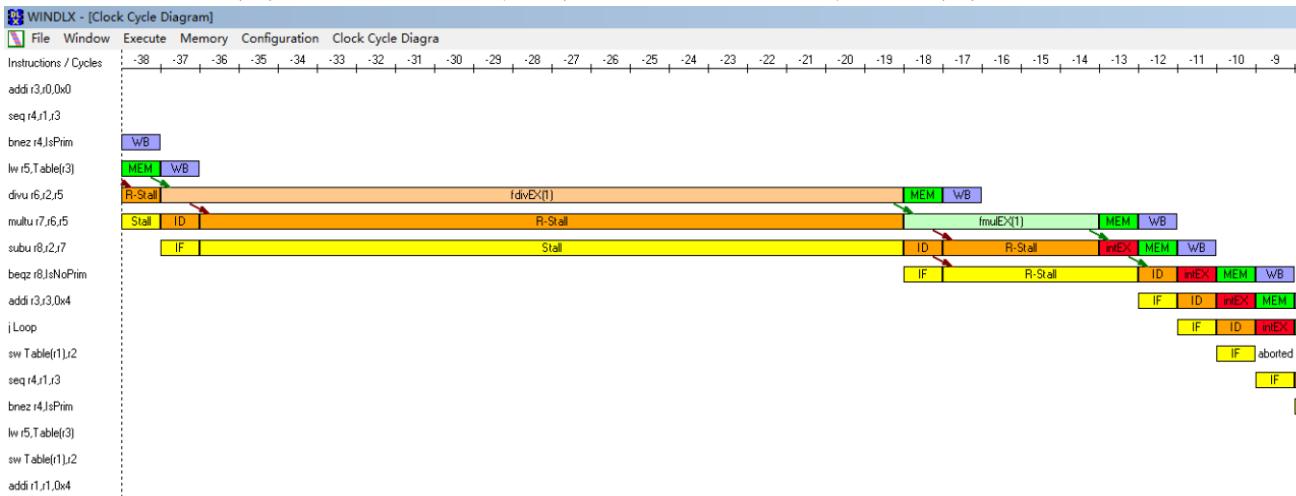
PC= 0x00000128	R12= 0x00000000	F8= 0 D8= 0
IMAR= 0x00000124	R13= 0x00000000	F9= 0 D10= 0
IR= 0x11000020	R14= 0x00000000	F10= 0 D12= 0
A= 0x00000003	R15= 0x00000000	F11= 0 D14= 0
AHI= 0x00000000	R16= 0x00000000	F12= 0 D16= 0
B= 0x00000000	R17= 0x00000000	F13= 0 D18= 0
BHI= 0x00000000	R18= 0x00000000	F14= 0 D20= 0
ETA= 0x00000000	R19= 0x00000000	F15= 0 D22= 0
ALU= 0x00000000	R20= 0x00000000	F16= 0 D24= 0
ALUHI= 0x00000000	R21= 0x00000000	F17= 0 D26= 0
FPSR= 0x00000000	R22= 0x00000000	F18= 0 D28= 0
DMAR= 0x00001004	R23= 0x00000000	F19= 0 D30= 0
SDR= 0x00000000	R24= 0x00000000	F20= 0
SDRHI= 0x00000000	R25= 0x00000000	F21= 0
LDR= 0x00000000	R26= 0x00000000	F22= 0
LDRHI= 0x00000000	R27= 0x00000000	F23= 0
R0= 0x00000000	R28= 0x00000000	F24= 0
R1= 0x00000004	R29= 0x00000000	F25= 0
R2= 0x00000003	R30= 0x00000000	F26= 0
R3= 0x00000000	R31= 0x00000000	F27= 0
R4= 0x00000000	F0= 0	F28= 0
R5= 0x00000002	F1= 0	F29= 0
R6= 0x00000001	F2= 0	F30= 0
R7= 0x00000000	F3= 0	F31= 0
R8= 0x00000000	F4= 0	D0= 0
R9= 0x0000000a	F5= 0	D2= 0
R10= 0x00000001	F6= 0	D4= 0
R11= 0x00000000	F7= 0	D6= 0

逐过程分析：

(1) 当判断完 2 后，接着判断 3 是否为素数。先比较 r1 和 r3 的值是否相等，此时  $r1 = 4$  而  $r3 = 0$ ，二者并不相等。

(2) 之后取出在素数表中位置为  $r3$  的数（即为 2），然后判断  $(3 / 2) * 2$  是否等于 3，由于 3 无法被 2 整除，因此  $3 / 2$  会存在精度损失，即  $(3 / 2) * 2 \neq 3$ 。

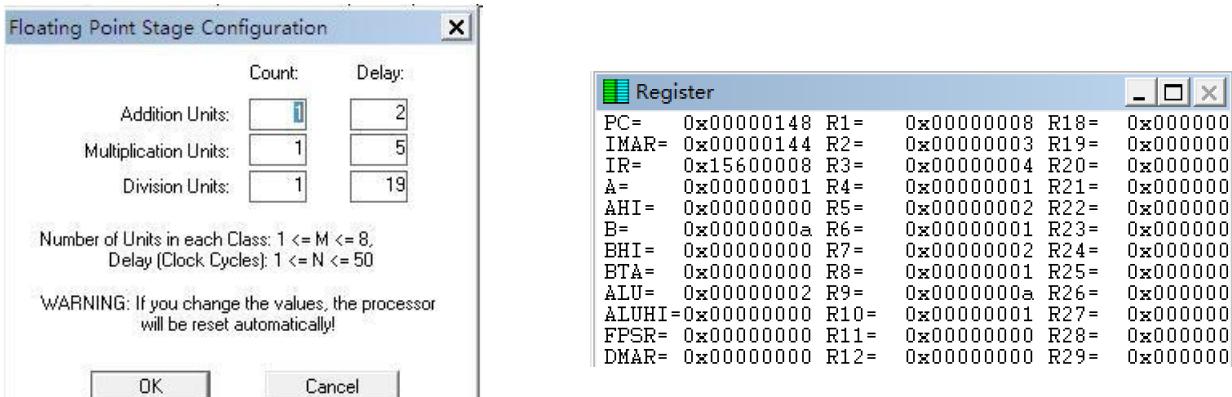
(3) 由此目前素数表中只有 2 一个元素，所以此时即可判断 3 为素数。



通过观察上图不难发现，乘法指令的执行周期为 5，除法指令的执行周期为 19。在上述过程中，流水线中几乎只有除法指令在执行，由此可见复杂指令对流水线的工作表现有很重要的影响。

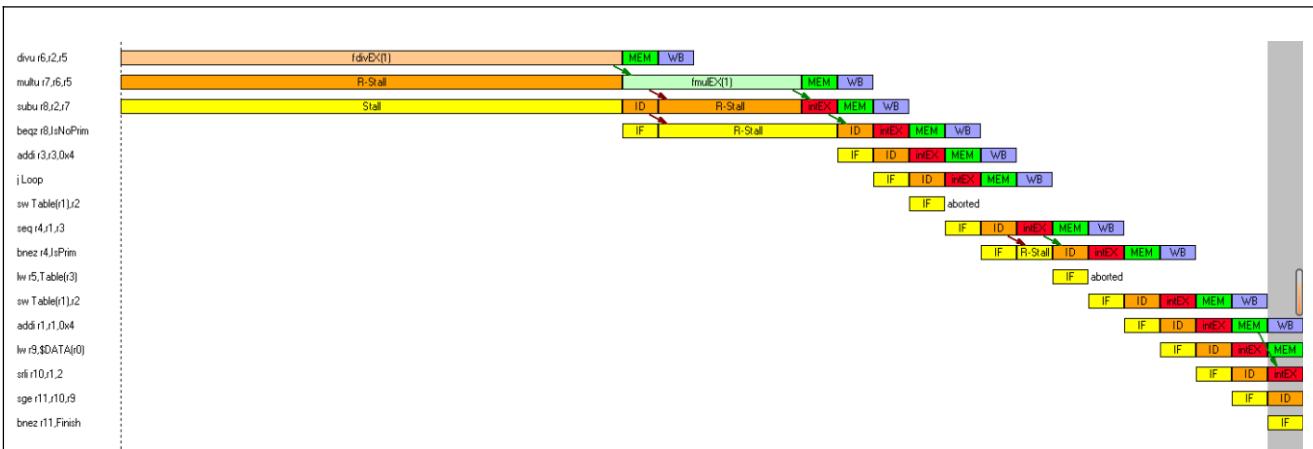
在 WinDLX 模拟器中，可以在 configuration/floating point stages 中设置乘除指令的时延，可以看到与上图中的实验结果是一样的。

(4) 当判断 3 为素数后，再将 r2 自增，同时将 3 存入素数表中，此时寄存器内容如下：



至此，素数 3 求取完毕。可以看到 R1 寄存器的内容变成了 8，即数组的下标实现了+1，也就是说，现在已经进入求取下一个素数的阶段。

通过 Clock Cycle Diagram 窗口可以详细看到流水线执行过程。



继续执行此函数到最后

发现此函数没有输出结果，于是我们在内存中查看结果。查看Symbols 找到Table 的地址为0x1004，在内存中找到指定内存单元，可以看到求出的素数表内容：

\$DATA	0x0000000a	Illegal (0x0000000a)
Table	0x00000002	srl r0,r0,0
Table+0x4	0x00000003	srai r0,r0,0
Table+0x8	0x00000005	Illegal (0x00000005)
Table+0xc	0x00000007	sra r0,r0,r0
Table+0x10	0x0000000b	Illegal (0x0000000b)
Table+0x14	0x0000000d	Illegal (0x0000000d)
Table+0x18	0x00000011	sneu r0,r0,r0
Table+0x1c	0x00000013	sgtu r0,r0,r0
Table+0x20	0x00000017	Illegal (0x00000017)
Table+0x24	0x0000001d	Illegal (0x0000001d)
Table+0x28	0x00000000	nop
Table+0x2c	0x00000000	nop

最后查看 Statistics 窗口

```
Total:
 63 Cycle(s) executed.
 ID executed by 30 Instruction(s).
 5 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: 32 (50.79% of all Cycles), thereof:
    LD stalls: 1 (3.12% of RAW stalls)
    Branch/Jump stalls: 5 (15.62% of RAW stalls)
    Floating point stalls: 26 (81.25% of RAW stalls)
  WAW stalls: 0 (0.00% of all Cycles)
  Structural stalls: 0 (0.00% of all Cycles)
  Control stalls: 4 (6.35% of all Cycles)
  Trap stalls: 0 (0.00% of all Cycles)
  Total: 36 Stall(s) (57.14% of all Cycles)

Conditional Branches:
  Total: 5 (16.67% of all Instructions), thereof:
    taken: 2 (40.00% of all cond. Branches)
    not taken: 3 (60.00% of all cond. Branches)

Load-/Store-Instructions:
  Total: 5 (16.67% of all Instructions), thereof:
    Loads: 3 (60.00% of Load-/Store-Instructions)
    Stores: 2 (40.00% of Load-/Store-Instructions)

Floating point stage instructions:
  Total: 2 (6.67% of all Instructions), thereof:
    Additions: 0 (0.00% of Floating point stage inst.)
    Multiplications: 1 (50.00% of Floating point stage inst.)
    Divisions: 1 (50.00% of Floating point stage inst.)

Traps:
  Traps: 0 (0.00% of all Instructions)
```

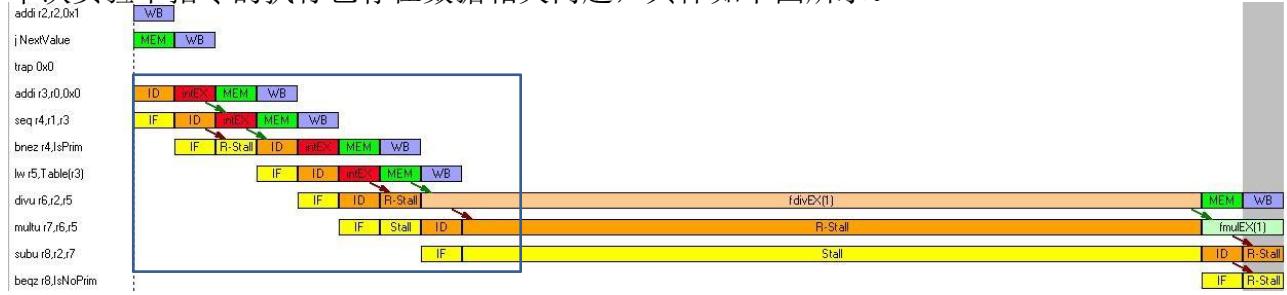
在此处也可以看到乘法、除法指令执行周期数。

### 结论分析与体会：

#### 结论分析：

1. 本次实验中是否出现流水线相关（流水线冒险）的问题？

本次实验中指令的执行也存在数据相关问题，具体如下图所示。



从上图中可以看到，红色箭头表示需要一个暂停，箭头指向处显示了暂停的原因。R-Stall (R-暂停) 表示引起暂停的原因是 RAW。绿色箭头表示定向技术的使用。定向技术的主要思想是：在某条指令（如上图）写回一个计算结果之前，其它指令需要该计算结果，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，那么就可以避免暂停。

基于上述分析，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。

#### 体会：

通过本次实验我进一步掌握了 WinDLX 模拟器的基本操作和使用，熟悉了 WinDLX 五段流水线在执行具体程序时的流水情况，更加深入的了解计算机系统流水线的工作过程。同时，通过单步跟踪指令的执行以及分析指令节拍，我进一步熟悉了 DLX 指令集结构及其特点，了解了乘法和除法在流水线中的节拍数。