

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: 202222130195	姓名: 阎发祥	班级: 22.3
实验题目: 用 WinDLX 模拟器执行程序求最大公约数		
实验学时: 2	实验日期: 2025. 4. 23	

**实验目的:**

- (1) 通过本实验, 熟练掌握 WinDLX 模拟器的操作和使用, 清楚 WinDLX 五段流水线在执行 具体程序时的流水情况。  
(2) 熟悉 DLX 指令集结构及其特点。

**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) 用 WinDLX 模拟器执行程序 gcm.s。该程序从标准输入读入两个整数, 求他们的 greatest common measure, 然后将结果写到标准输出。该程序中调用了 input.s 中的输入子程序。  
(2) 给出两组数 6、3 和 6、1, 分别在 main+0x8(add r2,r1,r0)、gcm.loop(seg r3,r1,r2) 和 result+0xc(trap 0x0)设断点, 采用单步和连续混合执行的方法完成程序, 注意中间过程 和 寄存器的变化情况, 然后单击主菜单 execute/display dlx-i/0, 观察结果。

**实验步骤:**

1. 汇编代码分析

本实验要求两个数的最大公约数, 先从汇编代码的角度分析程序的运行过程, 如下所示: 在汇编代码中首先是对一些常量进行了定义, 例如 Prompt1, Prompt2 等。

```
.data

    ;*** Prompts for input
Prompt1: .asciiz "First Number:"
Prompt2: .asciiz "Second Number:

    ;*** Data for printf-Trap
PrintfFormat: .asciiz "gcM=%d\n\n"
                .align 2
PrintfPar:    .word    PrintfFormat
PrintfValue:  .space   4
```

之后是 main 函数对应的汇编代码。在本次实验中 main 函数的主要作用是提示用户输入数据, 计算最大公约数以及向显示器输出信息, 具体代码如下图所示:

```

        .text
        .global main
main:
    ;*** Read two positive integer numbers into R1 and R2
    addi    r1, r0, Prompt1
    jal     InputUnsigned ;read uns.-integer into R1
    add    r2, r1, r0 ;R2 <- R1
    addi    r1, r0, Prompt2
    jal     InputUnsigned ;read uns.-integer into R1

```

通过上面代码可以看到，在 main 函数中调用了 input.s 文件中的 InputUnsigned 函数，实现了数据的读入。当读入两个数之后，通过使用 Loop 和比较大小来求解两个数的最大公约数。

```

Loop:           ;*** Compare R1 and R2
    seq    r3, r1, r2 ;R1 == R2 ?
    bnez  r3, Result
    sgt   r3, r1, r2 ;R1 > R2 ?
    bnez  r3, r1Greater

r2Greater: ;*** subtract r1 from r2
    sub   r2, r2, r1
    j     Loop

r1Greater: ;*** subtract r2 from r1
    sub   r1, r1, r2
    j     Loop

```

最后计算出最大公约数后，需要将结果输出在显示器上，通过 trap 执行系统调用实现

```

Result: ;*** Write the result (R1)
    sw     PrintfValue, r1
    addi  r14, r0, PrintfPar
    trap   5

;*** end
    trap   0

```

## 2. 装入程序单步执行

WINDLX - [Code]			
		File	Window Execute Memory Configuration Code
\$TEXT		0x20011000	addi r1, r0, 0x1000
main+0x4		0x0c00003c	jal InputUnsigned
main+0x8	BID	0x00201020	add r2, r1, 0
main+0xc		0x2001100e	addi r1, r0, 0x100e
main+0x10		0x0c000030	jal InputUnsigned
gcmLoop	BID	0x00221828	seq r3, r1, r2
0x00000118		0x14600018	bnez r3, Result
0x0000011c		0x0022182b	sgt r3, r1, r2
0x00000120		0x14600008	bnez r3, r1Greater
r2Greater		0x00411022	sub r2, r2, r1
0x00000128		0x0bfffe8	j gcmLoop
r1Greater		0x00220822	sub r1, r1, r2
0x00000130		0x0bfffe0	j gcmLoop
Result		0xac01102c	sw PrintfValue(r0), r1
Result+0x4		0x200e1028	addi r14, r0, 0x1028
Result+0x8		0x44000005	trap 0x5
Result+0xc	BID	0x44000000	trap 0x0

在运行程序之前首先在 main+0x8(add r2, r1, r0)、gcm.loop(seg r3,r1,r2)和result+0xc (trap 0x0)设置断点，之后单步执行指令，同时观察各个部件之间的变化

```

$TEXT      0x20011000 WB    addi r1,0x0x100
main+0x4   0x0c00003c MEM   jal InputUnsigned

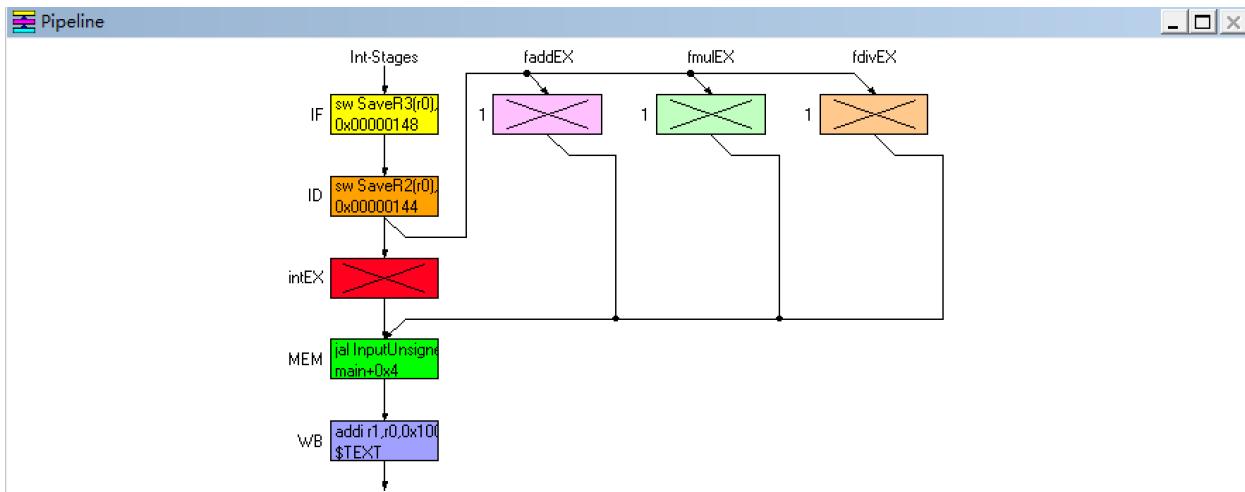
main+0x8   BID 0x00201020  add r2,r1,0
main+0xc   0x2001100e  addi r1,0,0x00e
main+0x10  0x0c000030  jal InputUnsigned
gcmLoop   BID 0x00221828  seq r3,r1,2
0x00000118 0x14600018  bnez r3,Result
0x0000011c 0x0022182b  sgt r3,r1,2
0x00000120 0x14600008  bnez r3,r1Greater
r2Greater 0x00411022  sub r2,r2,1
0x00000128 0xbffffe8   j gcmLoop
r1Greater 0x00220822  sub r1,r1,2
0x00000130 0xbffffe0   j gcmLoop
Result    0xac01102c  sw PrintValue(r0),1
Result+0x4 0x200e1028  addi r14,r0,0x1028
Result+0x8 0x44000005  trap 0x5
Result+0xc 0x44000000  trap 0x0

0x00000144 0xac021090 ID   sw SaveR2(r0),2
0x00000148 0xac031094 IF   sw SaveR3(r0),3
0x0000014c 0xac041098  sw SaveR4(r0),4
0x00000150 0xac05109c  sw SaveR5(r0),5
0x00000154 0xac01108c  sw input.PrintPar(r0),1
0x00000158 0x200e108c  addi r14,r0,0x108c
n:nnnnnnn5~ 0x44000005

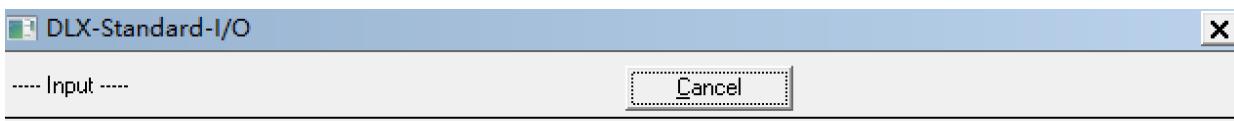
```

此时，执行过程中从 jal InputUnsigned 跳转到 0x00000144 的位置，说明正在调用 input.s 中的函数读入数据。

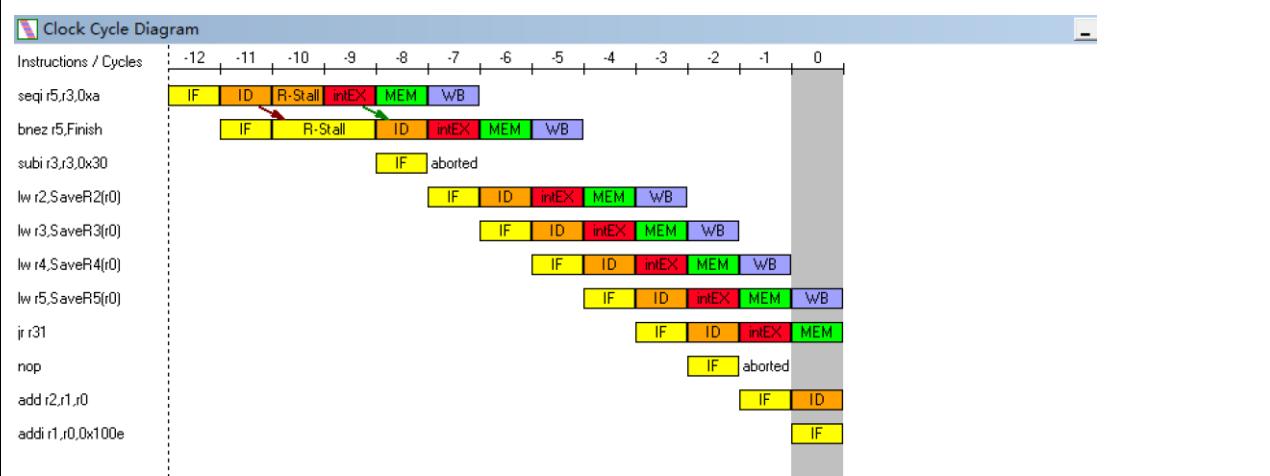
观察到 Pipeline 中 DLX 的五段流水线如下所示：



继续单步执行，直到提示输出 First number，此处我们输入 6。



此时查看 Clock Cycle Diagram 模块，发现出现了红和绿两种颜色的箭头，红色箭头是因为指令之间发生了冲突，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。此处是因为第二条指令 lbu r3, 0x0[r2]和第三条指令 seqi r5, r3, 0xa，前者在 EX 执行阶段时要向 r3 中写入数据，而后者要在 ID 阶段取 r3 寄存器中的值作为源操作数，此时发生数据相关。而绿色箭头表示的是定向技术的使用，即在第二条指令访存后，立刻将 r3 寄存器中的值送入到第三条指令 ID 阶段，减少等待时间。



之后查看 Register 模块，可以发现 R1 寄存器中值变为 6，即我们刚才输入的值。

Register

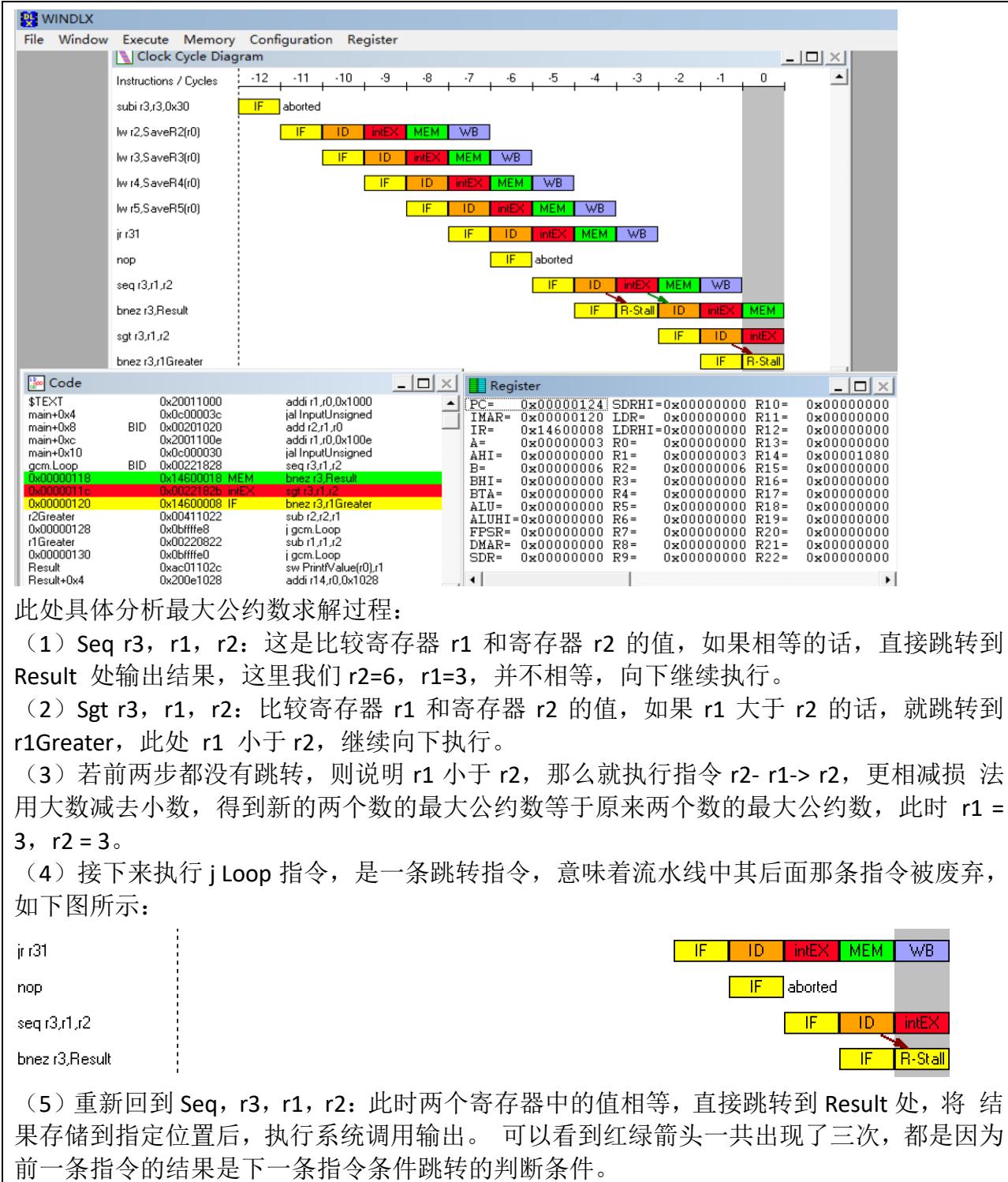
PC= 0x000000110	R5= 0x000000000	R26= 0x000000000	F15=
IMAR= 0x00000010c	R6= 0x000000000	R27= 0x000000000	F16=
IR= 0x2001100e	R7= 0x000000000	R28= 0x000000000	F17=
A= 0x000000006	R8= 0x000000000	R29= 0x000000000	F18=
AHI= 0x000000000	R9= 0x000000000	R30= 0x000000000	F19=
B= 0x000000000	R10= 0x000000000	R31= 0x000000108	F20=
BHI= 0x000000000	R11= 0x000000000	F0= 0	F21=
BTA= 0x000000000	R12= 0x000000000	F1= 0	F22=
ALU= 0x000000000	R13= 0x000000000	F2= 0	F23=
ALUHI=0x000000000	R14= 0x00001080	F3= 0	F24=
FPSR= 0x000000000	R15= 0x000000000	F4= 0	F25=
DMAR= 0x000000000	R16= 0x000000000	F5= 0	F26=
SDR= 0x000000000	R17= 0x000000000	F6= 0	F27=
SDRHI=0x000000000	R18= 0x000000000	F7= 0	F28=
LDR= 0x000000000	R19= 0x000000000	F8= 0	F29=
LDRHI=0x000000000	R20= 0x000000000	F9= 0	F30=
R0= 0x000000000	R21= 0x000000000	F10= 0	F31=
R1= 0x000000006	R22= 0x000000000	F11= 0	D0=
R2= 0x000000000	R23= 0x000000000	F12= 0	D2=
R3= 0x000000000	R24= 0x000000000	F13= 0	D4=
R4= 0x000000000	R25= 0x000000000	F14= 0	D6=

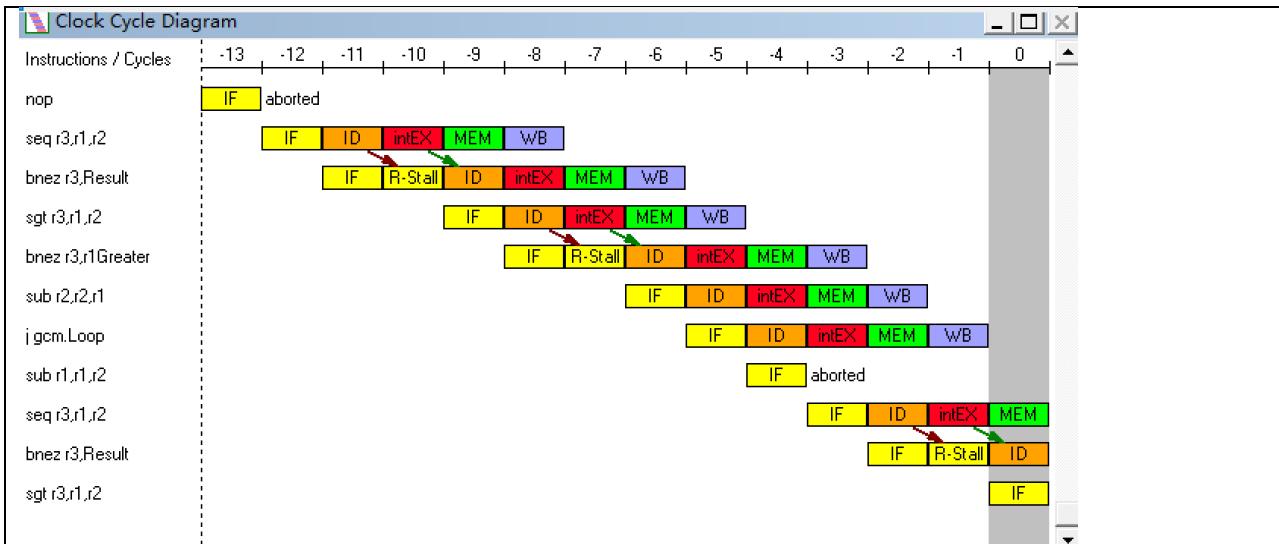
继续按下 F5，输入第二个数 3，此时输入完成。可以看到的是第一个数被保存在 R2，第二个数被保存在 R1，准确来说调用 InputUnsigned 函数的返回值会被保存到 R1 寄存器中。

Register

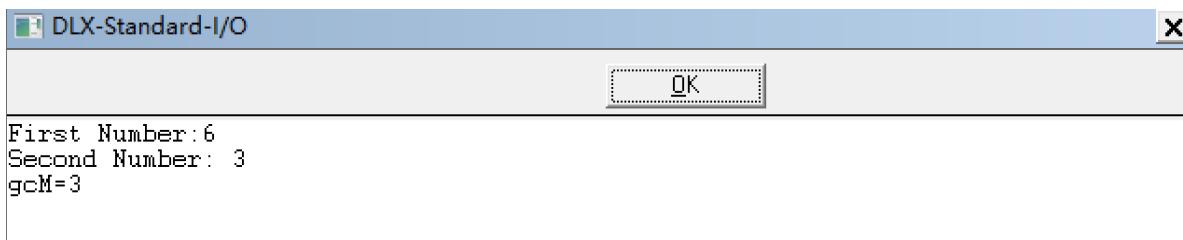
PC= 0x00000011c	R5= 0x000000000	R26= 0x000000000	F15=
IMAR= 0x000000118	R6= 0x000000000	R27= 0x000000000	F16=
IR= 0x14600018	R7= 0x000000000	R28= 0x000000000	F17=
A= 0x000000003	R8= 0x000000000	R29= 0x000000000	F18=
AHI= 0x000000000	R9= 0x000000000	R30= 0x000000000	F19=
B= 0x000000006	R10= 0x000000000	R31= 0x000000114	F20=
BHI= 0x000000000	R11= 0x000000000	F0= 0	F21=
BTA= 0x000000000	R12= 0x000000000	F1= 0	F22=
ALU= 0x000000000	R13= 0x000000000	F2= 0	F23=
ALUHI=0x000000000	R14= 0x00001080	F3= 0	F24=
FPSR= 0x000000000	R15= 0x000000000	F4= 0	F25=
DMAR= 0x000000000	R16= 0x000000000	F5= 0	F26=
SDR= 0x000000000	R17= 0x000000000	F6= 0	F27=
SDRHI=0x000000000	R18= 0x000000000	F7= 0	F28=
LDR= 0x000000000	R19= 0x000000000	F8= 0	F29=
LDRHI=0x000000000	R20= 0x000000000	F9= 0	F30=
R0= 0x000000000	R21= 0x000000000	F10= 0	F31=
R1= 0x000000003	R22= 0x000000000	F11= 0	D0=
R2= 0x000000006	R23= 0x000000000	F12= 0	D2=
R3= 0x000000000	R24= 0x000000000	F13= 0	D4=
R4= 0x000000000	R25= 0x000000000	F14= 0	D6=

接下来进入 Loop 循环中求解最大公约数，执行过程中流水线和 PC 值变化如下：





最后输出结果为 3，如下所示：



此时程序运行结束，查看 Statistics 窗口，程序执行的统计信息如下：

File Window Execute Memory Configuration Statistics

**Total:**  
110 Cycle(s) executed.  
ID executed by 69 Instruction(s).  
3 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: 23 (20.91% of all Cycles), thereof:  
LD stalls: 4 (17.39% of RAW stalls)  
Branch/Jump stalls: 7 (30.43% of RAW stalls)  
Floating point stalls: 12 (52.17% of RAW stalls)  
WAW stalls: 0 (0.00% of all Cycles)  
Structural stalls: 0 (0.00% of all Cycles)  
Control stalls: 10 (9.09% of all Cycles)  
Trap stalls: 12 (10.91% of all Cycles)  
Total: 45 Stall(s) (40.91% of all Cycles)

**Conditional Branches:**  
Total: 7 (10.14% of all Instructions), thereof:  
taken: 3 (42.86% of all cond. Branches)  
not taken: 4 (57.14% of all cond. Branches)

**Load-/Store-Instructions:**  
Total: 22 (31.88% of all Instructions), thereof:  
Loads: 12 (54.54% of Load-/Store-Instructions)  
Stores: 10 (45.45% of Load-/Store-Instructions)

**Floating point stage instructions:**  
Total: 2 (3.00% of all Instructions), thereof:  
Additions: 0 (0.00% of Floating point stage inst.)  
Multiplications: 2 (100.00% of Floating point stage inst.)  
Divisions: 0 (0.00% of Floating point stage inst.)

**Traps:**  
Traps: 4 (5.80% of all Instructions)

### 3. 重复性实验

上述实验中以 6 和 3 为例分析了指令执行的具体过程。为了进一步感受指令执行时寄存器中值的变化，以 6 和 2 作为程序的输入，重复试验，对比两个实验结果中寄存器的区别。

Register	
PC=	0x0000001c
SDRHI=	0x00000000
R10=	0x00000000
IMAR=	0x00000018
LDR=	0x00000000
R11=	0x00000000
IR=	0x14600018
LDRHI=	0x00000000
R12=	0x00000000
A=	0x00000002
R0=	0x00000000
R13=	0x00000000
AHI=	0x00000000
R1=	0x00000002
R14=	0x00001080
B=	0x00000006
R2=	0x00000006
R15=	0x00000000
BHI=	0x00000000
R3=	0x00000000
R16=	0x00000000
BTA=	0x00000000
R4=	0x00000000
R17=	0x00000000
ALU=	0x00000000
R5=	0x00000000
R18=	0x00000000
ALUHI=	0x00000000
R6=	0x00000000
R19=	0x00000000
FPSR=	0x00000000
R7=	0x00000000
R20=	0x00000000
DMAR=	0x00000000
R8=	0x00000000
R21=	0x00000000
SDR=	0x00000000
R9=	0x00000000
R22=	0x00000000

可以看到此时两个寄存器的值分别变为了 2 和 6，最终显示器输出结果如下所示：



查看 Statistics 窗口，程序执行的统计信息如下：

File		Window	Execute	Memory	Configuration	Statistics
<b>Total:</b>						
145 Cycle(s) executed. ID executed by 88 Instruction(s). 2 Instruction(s) currently in Pipeline.						
<b>Hardware configuration:</b>						
Memory size: 32768 Bytes faddXStages: 1, required Cycles: 2 fmulXStages: 1, required Cycles: 5 fdvEXStages: 1, required Cycles: 19 Forwarding enabled.						
<b>Stalls:</b>						
RAW stalls: 25 (17.24% of all Cycles), thereof: LD stalls: 4 (16.00% of RAW stalls) Branch/Jump stalls: 9 (36.00% of RAW stalls) Floating point stalls: 12 (48.00% of RAW stalls) WAW stalls: 0 (0.00% of all Cycles) Structural stalls: 0 (0.00% of all Cycles) Control stalls: 11 (7.60% of all Cycles) Trap stalls: 24 (16.55% of all Cycles) Total: 60 Stall(s)(41.38% of all Cycles)						
<b>Conditional Branches:</b>						
Total: 9 (10.23% of all Instructions), thereof: taken: 3 (33.33% of all cond. Branches) not taken: 6 (66.67% of all cond. Branches)						
<b>Load-/Store-Instructions:</b>						
Total: 28 (31.02% of all Instructions), thereof: Loads: 12 (42.86% of Load-/Store-Instructions) Stores: 16 (57.14% of Load-/Store-Instructions)						
<b>Floating point stage instructions:</b>						
Total: 2 (2.27% of all Instructions), thereof: Additions: 0 (0.00% of Floating point stage inst.) Multiplications: 2 (100.00% of Floating point stage inst.) Divisions: 0 (0.00% of Floating point stage inst.)						
<b>Traps:</b>						
Traps: 8 (9.09% of all Instructions)						

结论分析与体会：

结论分析：

1. 从两次实验的 Statistics 窗口中不难发现，第二次实验执行的时钟周期数比第一次多不少，这是跟两次实验输入数据有关的。第一次输入的 6 和 3，只做 1 次减法就结束了，而第二次输入的 6 和 1，做了 5 次减法才结束，因此第二次实验执行的时钟周期数多出不少。

2. 实验中计算最大公约数的办法

实验中使用的是更相减损法实现最大公约数的求解，具体过程如下：设第一个输入的数是 a，第二个输入的数是 b

- (1) 若  $a == b$ ，则  $a$  (或  $b$ ) 即为最大公约数
- (2) 若  $a > b$ ，则  $a = a - b$
- (3) 若  $b > a$ ，则  $b = b - a$
- (4) 若  $a != b$ ，则回到 1

也可以使用辗转相除法来求解最大公约数。

体会：

本次实验通过单步跟踪最大公约数的求解过程，从汇编层面更好地理解了指令执行的基本步骤。实验中的 GCM.s 代码在逻辑上可以划分为 3 个部分。首先是数据的读入，涉及到 I/O 操作，之后就是通过 Loop 循环进行具体的计算，最终再次进行 I/O 操作，将计算的结果输出在显示器上。通过本次的实验操作，我进一步掌握了 WinDLX 模拟器的基本操作和使用，并且通过读源码基本上进一步熟悉了 DLX 指令集结构和特点