

学号：201900130133	姓名：施政良	班级：四班
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：2022-	
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： Windows 7		
实验步骤与内容： 实验内容 本次实验主要涉及 xxxx, 具体的实验步骤可以划分为如下几个步骤 (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) 跟踪程序的运行状态 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。		
具体实验过程 1. 汇编代码分析 本次实验以求两个数的最大公约数为例，从汇编代码的角度分析程序的运行过程，并观察指令流水。需要对汇编代码进行分析。		

在汇编代码的 12-22 行首先对一些常量进行了定义，例如 *promtp1*, *promp2* 等。

```
1.      .data
2.
3.      ;*** Prompts for input
4. Prompt1:      .asciiiz      "First Number:"
5. Prompt2:      .asciiiz      "Second Number: "
6.
7.      ;*** Data for printf-Trip
8. PrintfFormat:  .asciiiz      "gcM=%d\n\n"
9.      .align      2
10. PrintfPar:    .word          PrintfFormat
11. PrintfValue:  .space         4
```

之后是 main 函数对应的汇编代码。在本次实验中 main 函数的主要作用是像显示器输出信息，提示用户输入数据，并且负责将用户输入的数据保存在相应的寄存器中。具体代码如下所示：

```
1. main:
2.      ;*** Read two positive integer numbers into R1 and R2
3.      addi      r1,r0,Prompt1
4.      jal      InputUnsigned ;read uns.-integer into R1
5.      add       r2,r1,r0      ;R2 <- R1
6.      addi      r1,r0,Prompt2
7.      jal      InputUnsigned ;read uns.-integer into R1
```

通过代码可以看到，在 main 函数中调用了 input.s 文件中的 read 函数，实现了数据的读入。

读入数据之后，通过使用 for 循环和 if 判断来求解两个数的最大公约数。

```
1. Loop:      ;*** Compare R1 and R2
2.      seq      r3,r1,r2      ;R1 == R2 ?
3.      bnez      r3,Result
4.      sgt      r3,r1,r2      ;R1 > R2 ?
5.      bnez      r3,r1Greater
```

在 Loop 循环中主要调用了 *r1Greate* 汇编代码段。

```
1. r1Greater: ;*** subtract r2 from r1
2.      sub      r1,r1,r2
3.      j         Loop
```

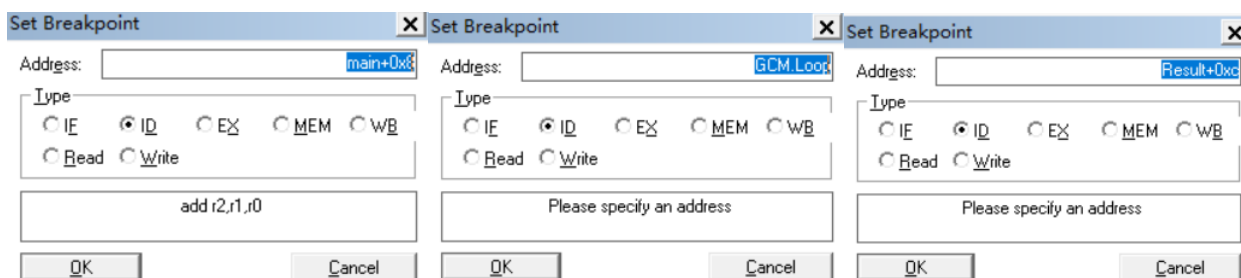
当求解出最大公约数之后，需要结果输出在显示器上，可以通过 Write 代码段实现。

```
1. Result:      ;*** Write the result (R1)
2.             sw      PrintfValue,r1
3.             addi     r14,r0,PrintfPar
4.             trap     5
5.
6.             ;*** end
7.             trap     0
```

最终实现了最大公约数的求解。

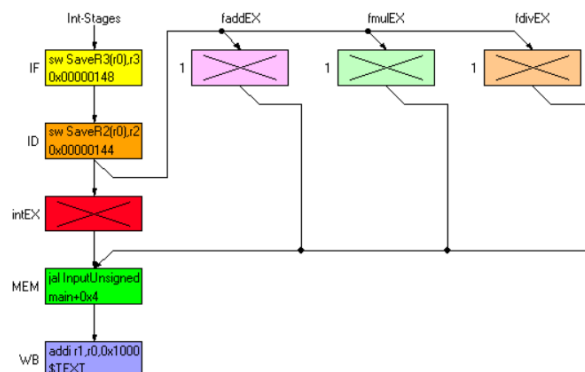
2. 具体实验过程展示

根据实验指导书的提示，在运行程序的指令之前需要首先在 main+0x8(add r2, r1, r0)、gcm. loop(seg r3, r1, r2)和 result+0xc(trap 0x0) 设断点



之后单步运行指令，同时观察寄存器和各个执行部件的变化。

\$TEXT	0x20011000	WB	addi r1,r0,0x1000
main+0x4	0x0c00003c	MEM	jal InputUnsigned
main+0x8	0x00201020		add r2,r1,r0
main+0xc	0x2001100e		addi r1,r0,0x100e
main+0x10	0x0c000030		jal InputUnsigned
GCM.Loop	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	0x0bffffe8		j GCM.Loop
r1Greater	0x00220822		sub r1,r1,r2
0x00000130	0x0bffffe0		j GCM.Loop
Result	0xac01102c		sw PrintfValue(r0),r1
Result+0x4	0x200e1028		addi r14,r0,0x1028
Result+0x8	0x44000005		trap 0x5
Result+0xc	0x44000000		trap 0x0
0x00000144	0xac021090	ID	sw SaveR2(r0),r2
0x00000148	0xac031094	IF	sw SaveR3(r0),r3
0x0000014c	0xac041098		sw SaveR4(r0),r4
0x00000150	0xac05109c		sw SaveR5(r0),r5
0x00000154	0xac0610a0		sw SaveR6(r0),r6



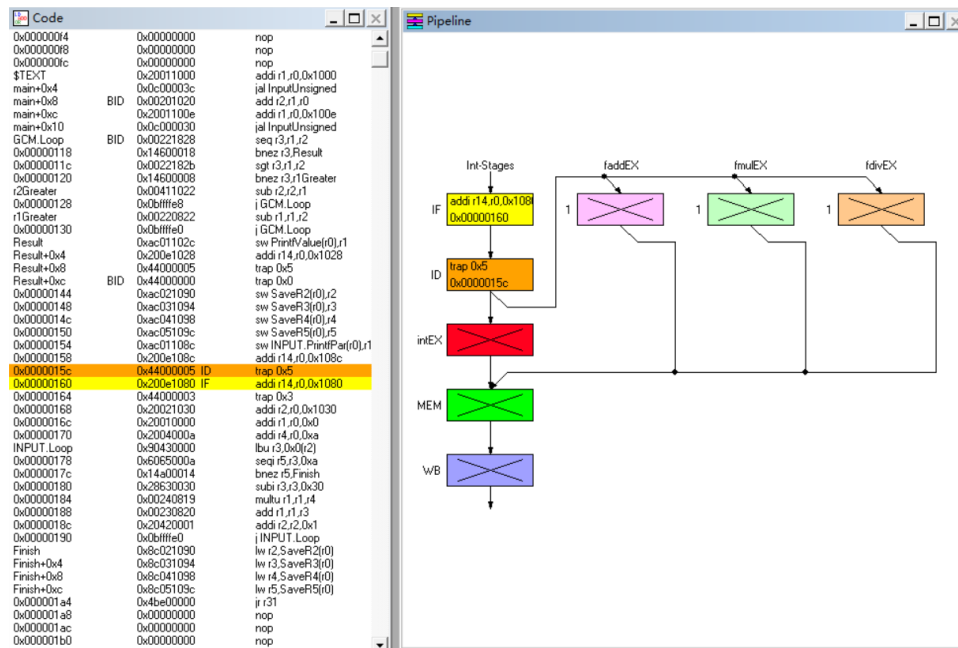
可以看到，当前程序jal inputUnsigned指令跳转到地址0x0000148位置，说明正在调用 input. s 中的数据读入函数。

继续单步运行，直到显示器输出First number，此时说明需要输入第一个数据。



First Number:

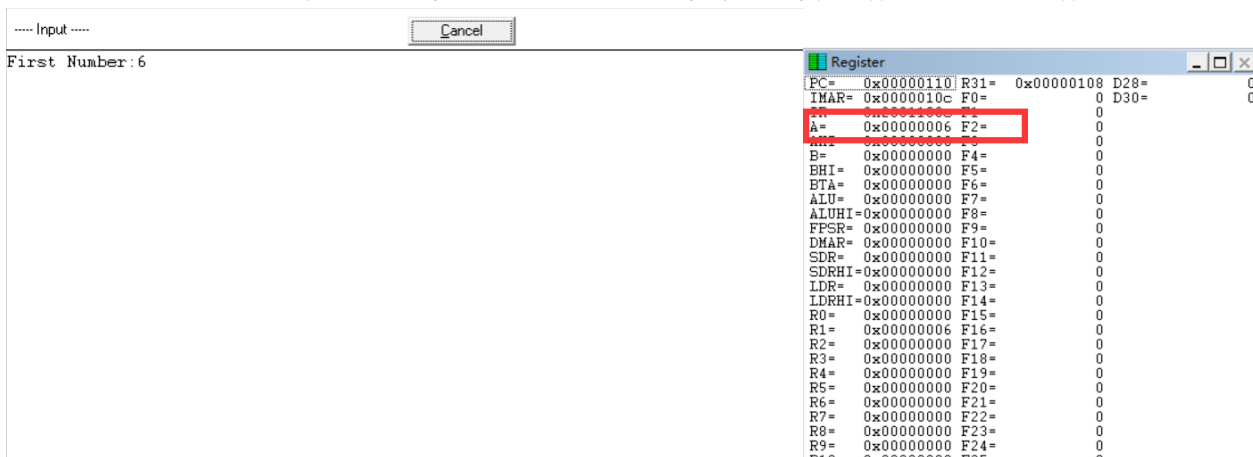
此时 PC 所在的位置和流水线的执行情况如下所示:



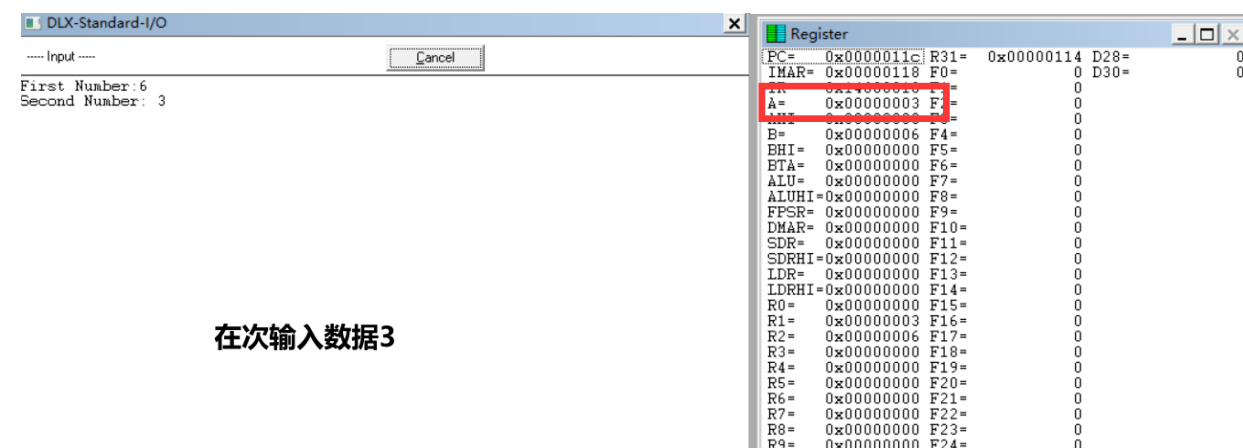
为了加快程序跟踪的速度, 采用连续跟踪的方式, 直接运行到断点 1 所在的位置。此时程序通过系统调用, 陷入内核并执行相应的 I/O 操作, 负责数据的读入。



以求解 6 和 3 的最大公约数为例，键盘键入数字 6，并保存在相应的寄存器中。

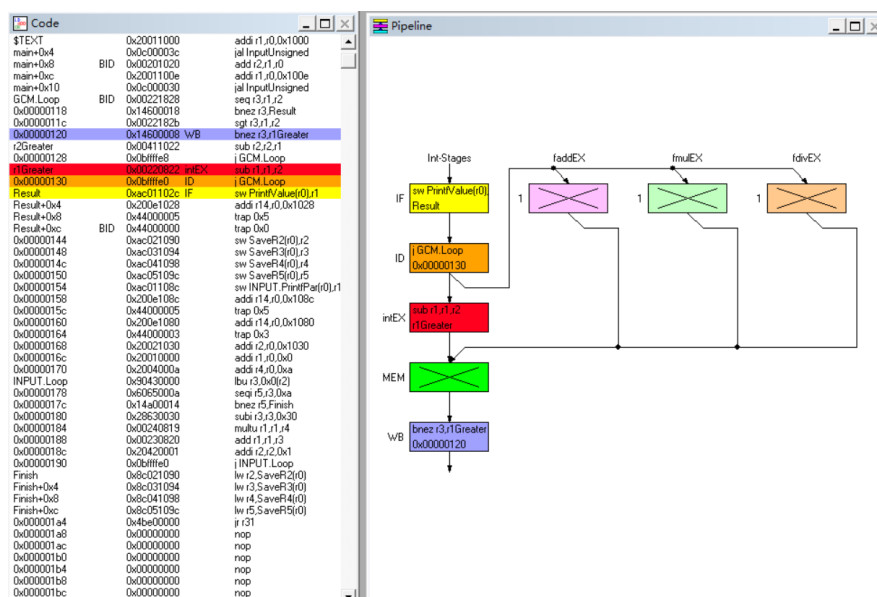


之后，根据上述对汇编代码的分析，需要再次输入第二个数字，且指令执行的逻辑同上。

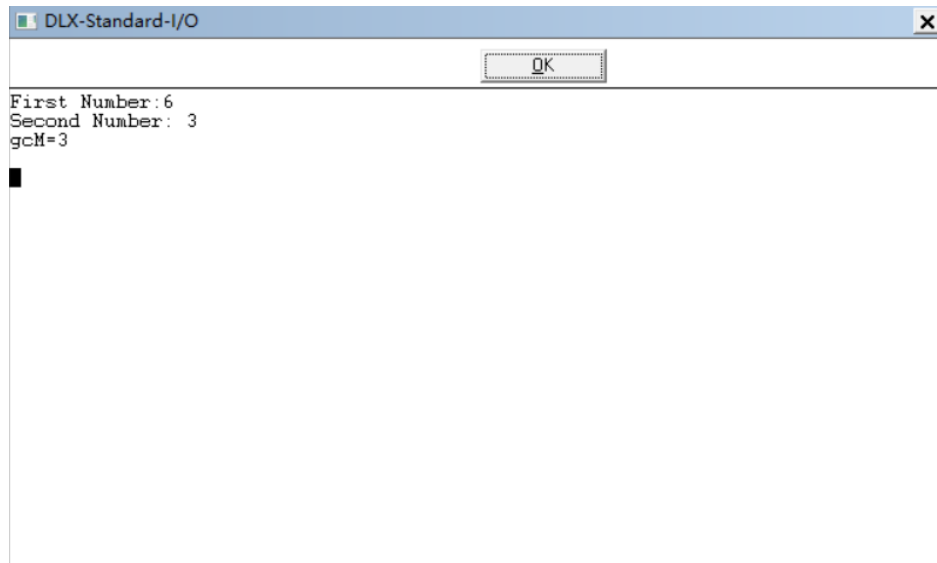


在次输入数据3

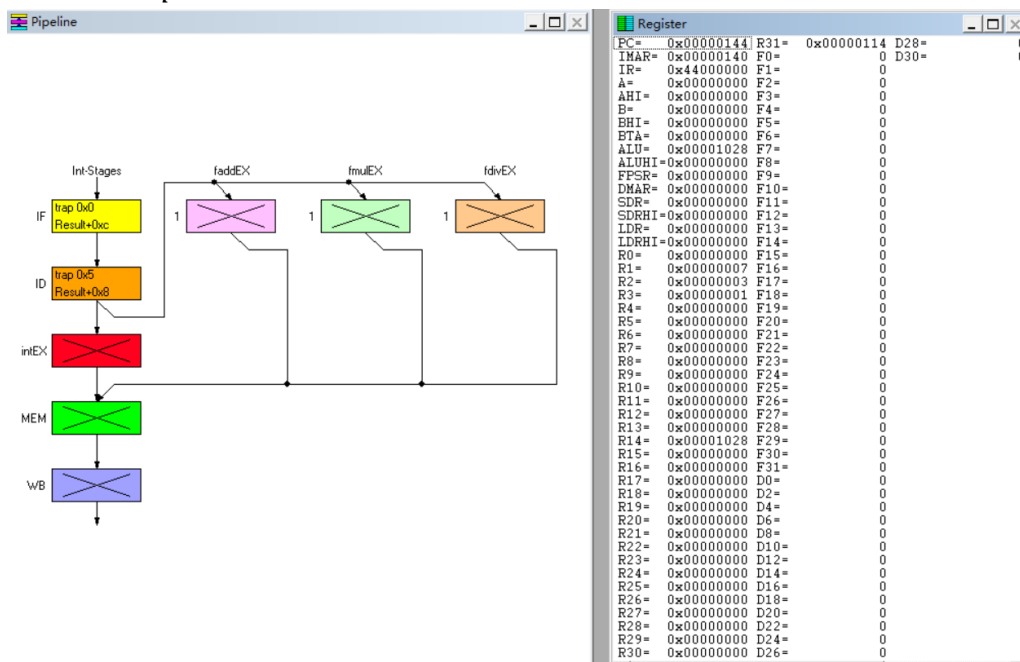
此时，完成数据的读入，进入 Loop 循环求解最大公约数。执行过程中流水线和 PC 指向的变化如下所示：

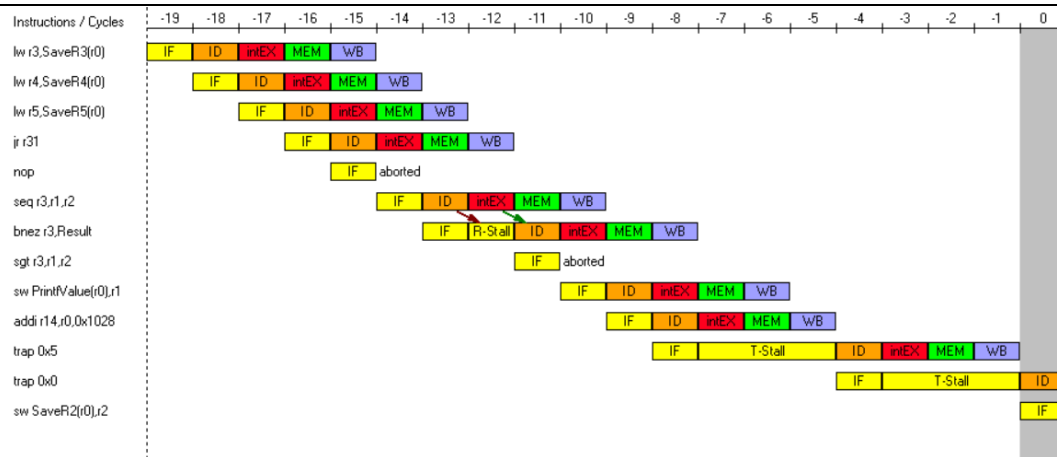


待执行到地址 $Result + 0xc$ 对应的指令时，运行结束，此时程序将最大公约数的结果写入显示器。



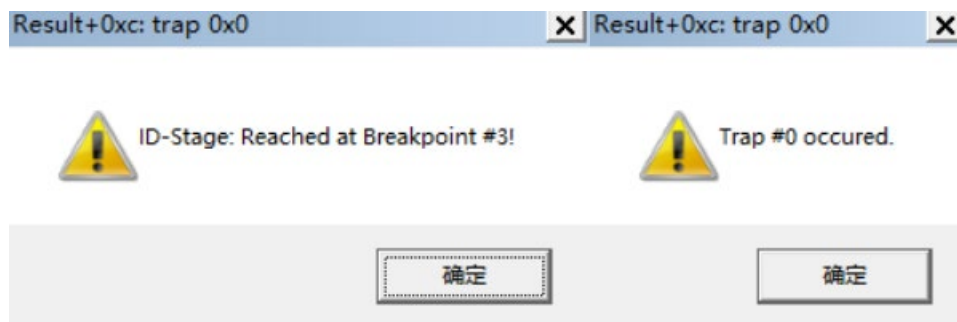
且此时寄存器和流水线的执行状态如下所示，可以发现，由于程序此时正在进行 I/O 操作，因此通过 $trap$ 指令陷入中断。





各个执行部件流水线示意图

再次执行连续跟踪，程序在第三个断点处停下，同时提示*Trap #0 occurred*



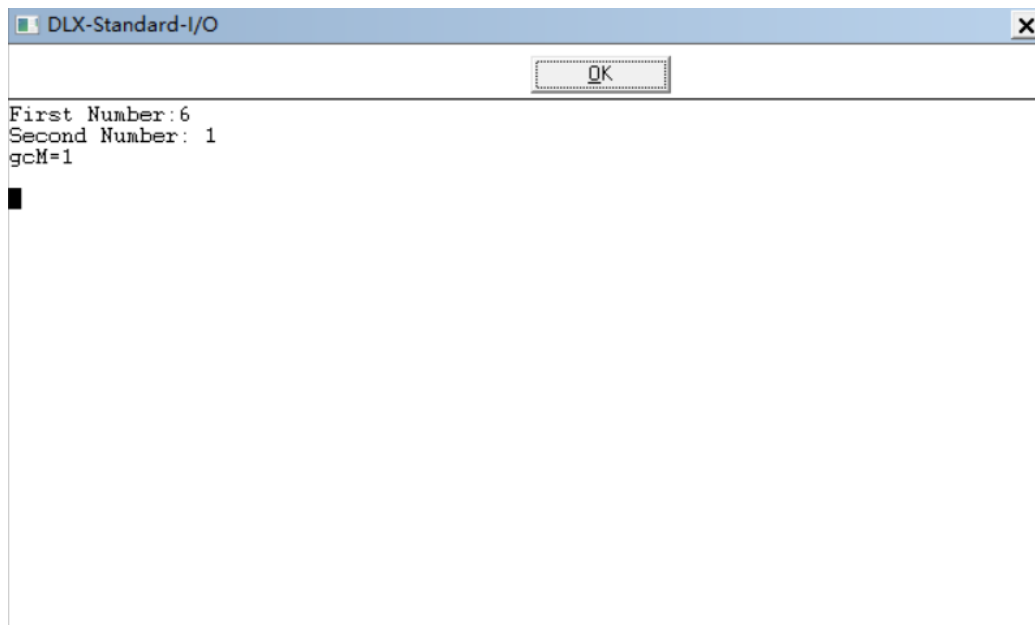
最终程序运行结束。程序执行的统计信息如下：

Statistics	
Total:	
117 Cycle(s) executed.	
ID executed by 72 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: 23 (19.66% 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 (8.55% of all Cycles)	
Trap stalls: 15 (12.82% of all Cycles)	
Total: 48 Stall(s) (41.02% of all Cycles)	
Conditional Branches):	
Total: 7 (9.72% 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: 23 (31.94% of all Instructions), thereof:	
Loads: 12 (52.17% of Load-/Store-Instructions)	
Stores: 11 (47.83% of Load-/Store-Instructions)	
Floating point stage instructions:	
Total: 2 (2.78% 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: 5 (6.94% of all Instructions)	

3. 重复性实验

上述实验中以数字 6 和数字 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的变化，可以以数字 6 和数字 1 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。

重复上述的操作，输入数据 6 和 1，最终显示器输出结果如下所示：



对比两次试验结束时寄存器的状态，可以发现，大部分寄存器的都未被使用到，取值为 0，而由于数据输入的不同，最终程序计数器 PC，以及寄存器 IMAR 和 IR 都有所不同。

Register										Register									
PC=	0x00000164	R31=	0x00000114	D28=	0	0	0	0	0	PC=	0x00000144	R31=	0x00000114	D28=	0	0	0	0	0
IMAR=	0x00000160	F0=	0	D30=	0	0	0	0	0	IMAR=	0x00000140	F0=	0	D30=	0	0	0	0	0
IR=	0x200e1080	F1=	0							IR=	0x44000000	F1=	0						
A=	0x00000000	F2=	0							A=	0x00000000	F2=	0						
AHI=	0x00000000	F3=	0							AHI=	0x00000000	F3=	0						
B=	0x00000000	F4=	0							B=	0x00000000	F4=	0						
BHI=	0x00000000	F5=	0							BHI=	0x00000000	F5=	0						
BT=	0x00000000	F6=	0							BT=	0x00000000	F6=	0						
ALU=	0x0000108c	F7=	0							ALU=	0x00001028	F7=	0						
ALUHI=	0x00000000	F8=	0							ALUHI=	0x00000000	F8=	0						
FPSR=	0x00000000	F9=	0							FPSR=	0x00000000	F9=	0						
DMAR=	0x00000000	F10=	0							DMAR=	0x00000000	F10=	0						
SDR=	0x00000000	F11=	0							SDR=	0x00000000	F11=	0						
SDRHI=	0x00000000	F12=	0							SDRHI=	0x00000000	F12=	0						
LDR=	0x00000000	F13=	0							LDR=	0x00000000	F13=	0						
LDRHI=	0x00000000	F14=	0							LDRHI=	0x00000000	F14=	0						
R0=	0x00000000	F15=	0							R0=	0x00000000	F15=	0						
R1=	0x00000000	F16=	0							R1=	0x00000007	F16=	0						
R2=	0x00000003	F17=	0							R2=	0x00000001	F17=	0						
R3=	0x00000001	F18=	0							R3=	0x00000001	F18=	0						
R4=	0x00000000	F19=	0							R4=	0x00000000	F19=	0						
R5=	0x00000000	F20=	0							R5=	0x00000000	F20=	0						
R6=	0x00000000	F21=	0							R6=	0x00000000	F21=	0						
R7=	0x00000000	F22=	0							R7=	0x00000000	F22=	0						
R8=	0x00000000	F23=	0							R8=	0x00000000	F23=	0						
R9=	0x00000000	F24=	0							R9=	0x00000000	F24=	0						
R10=	0x00000000	F25=	0							R10=	0x00000000	F25=	0						
R11=	0x00000000	F26=	0							R11=	0x00000000	F26=	0						
R12=	0x00000000	F27=	0							R12=	0x00000000	F27=	0						
R13=	0x00000000	F28=	0							R13=	0x00000000	F28=	0						
R14=	0x0000108c	F29=	0							R14=	0x00001028	F29=	0						
R15=	0x00000000	F30=	0							R15=	0x00000000	F30=	0						
R16=	0x00000000	F31=	0							R16=	0x00000000	F31=	0						
R17=	0x00000000	D0=	0							R17=	0x00000000	D0=	0						

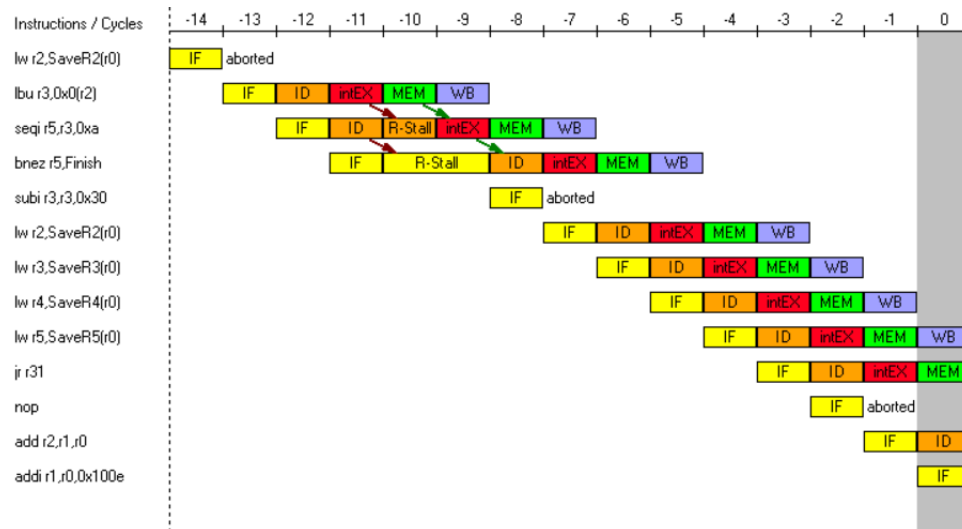
（注：（左图为数据 6、1 对应的结果，右图为数据 6、1 对应的结果）

结论分析与体会：

结论分析

1. 在上述程序执行的过程中是否发生了“流水线冒险”，即流水线相关分析：

根据执行部件的指令流水可知，在程序执行的过程中发生了流水线冒险。例如，当执行第二条和第三条指令时，会发生数据相关，如下图所示：



在指令之间出现了红和绿的箭头。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。

2. 分析实验中最大公约数的计算方法

分析：

实验中使用更相减损术实现最大公约数的求解。

第一步：任意给定两个正整数；判断它们是否都是偶数。若是，则用 2 约简；若不是则执行第二步。

第二步：以较大的数减较小的数，接着把所得的差与较小的数比较，并以大数减小数。继续这个操作，直到所得的减数和差相等为止。则第一步中约掉的若干个 2 的积与第二步中等数的乘积就是所求的最大公约数。即如下四步：

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

体会

本次实验通过单步跟踪最大公约数的求解，从汇编层面了解了指令执行的基本步骤。实验中的 GCM.s 代码在逻辑上可以大致划分为 3 个部分。首先是数据的读入，涉及到 I/O 操作，之后通过 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。

附录

GCM.S 完整代码

```
1  ;***** WINDLX Ex.1: Greatest common measure *****
2  ;***** (c) 1991 G 騰 ther Raidl *****
3  ;***** Modified 1992 Maziar Khosravipour *****
4
5  ;-----
6  ; Program begins at symbol main
7  ; requires module INPUT
8  ; Read two positive integer numbers from stdin, calculate the gcm
9  ; and write the result to stdout
10 ;-----
11
12 .data
13
14 ;*** Prompts for input
15 Prompt1: .asciiz "First Number:"
16 Prompt2: .asciiz "Second Number: "
17
18 ;*** Data for printf-Trip
19 PrintfFormat: .asciiz "gcm=%d\n\n"
20 .align 2
21 PrintfPar: .word PrintfFormat
22 PrintfValue: .space 4
23
24
25 .text
26 .global main
27 main:
28 ;*** Read two positive integer numbers into R1 and R2
29 addi r1,r0,Prompt1
30 jal InputUnsigned ;read uns.-integer into R1
31 add r2,r1,r0 ;R2 <- R1
32 addi r1,r0,Prompt2
```

```

33  jal  InputUnsigned ;read uns.-integer into R1
34
35  Loop:  ;*** Compare R1 and R2
36      seq  r3,r1,r2 ;R1 == R2 ?
37      bnez r3,Result
38      sgt  r3,r1,r2 ;R1 > R2 ?
39      bnez r3,r1Greater
40
41  r2Greater: ;*** subtract r1 from r2
42      sub  r2,r2,r1
43      j   Loop
44
45  r1Greater: ;*** subtract r2 from r1
46      sub  r1,r1,r2
47      j   Loop
48
49  Result:  ;*** Write the result (R1)
50      sw  PrintfValue,r1
51      addi r14,r0,PrintfPar
52      trap 5
53
54      ;*** end
55      trap 0

```