**深 圳 大 学 实 验 报 告**

**课 程 名 称： 计算机系统(3)**

**实验项目名称： MIPS指令集实验**

**学 院： 计算机与软件学院**

**专 业： 计算机与软件学院所有专业**

**指 导 教 师： 罗秋明**

**报告人： 王曦 学号： 2021192010 班级： 数计**

**实 验 时 间： 2023年09月17日**

**实验报告提交时间： 2023年09月17日**

**教务处制**

**一、 实验目标：**

了解WinMIPS64的基本功能和作用；

熟悉MIPS指令、初步建立指令流水执行的感性认识；

掌握该工具的基本命令和操作，为流水线实验作准备。

**二、实验内容**

按照下面的实验步骤及说明，完成相关操作**记录实验过程的截图**：

1）下载WinMIPS64；运行样例代码并观察软件各个观察窗口的内容和作用，掌握软件的使用方法。（80分）

2）学会正确使用WinMIPS64的IO方法；（10分）

3）编写完整的排序程序；（10分）

**三、实验环境**

硬件：桌面PC

软件：Windows，WinMIPS64仿真器

**四、****实验步骤及说明**

WinMIPS64是一款指令集模拟器，它是基于WinDLX设计的，如果你对于WinDLX这款软件十分熟悉的话，那么对于WinMIPS64也会十分的容易上手。DLX 处理器 (发音为 "DeLuXe")是Hennessy 和Patterson合著一书《**Computer Architecture - A Quantitative Approach**》中流水线处理器的例子。WinDLX是一个基于Windows的模拟器。

本教程通过一个实例介绍WinMIPS64的使用方法。WinMIPS64模拟器能够演示MIPS64流水线是如何工作的。

本教程使用的例子非常简单，它并没有囊括WinMIPS64的各个方面，仅仅作为使用WinMIPS64的入门级介绍。如果你想自己了解更多的资料，在给出的winmips64.zip中，有WinMIPS64 — Documentation Summary.html和winmipstut.docx两个文件可以供你随时参考，其中涵盖了WinMIPS64的指令集和模拟器的组成与使用方法。

虽然我们将详细讨论例子中的各个阶段，但你应具备基本的使用Windows的知识。现假定你知道如何启动 Windows，使用滚动条滚动，双击执行以及激活窗口。

（一）、安 装

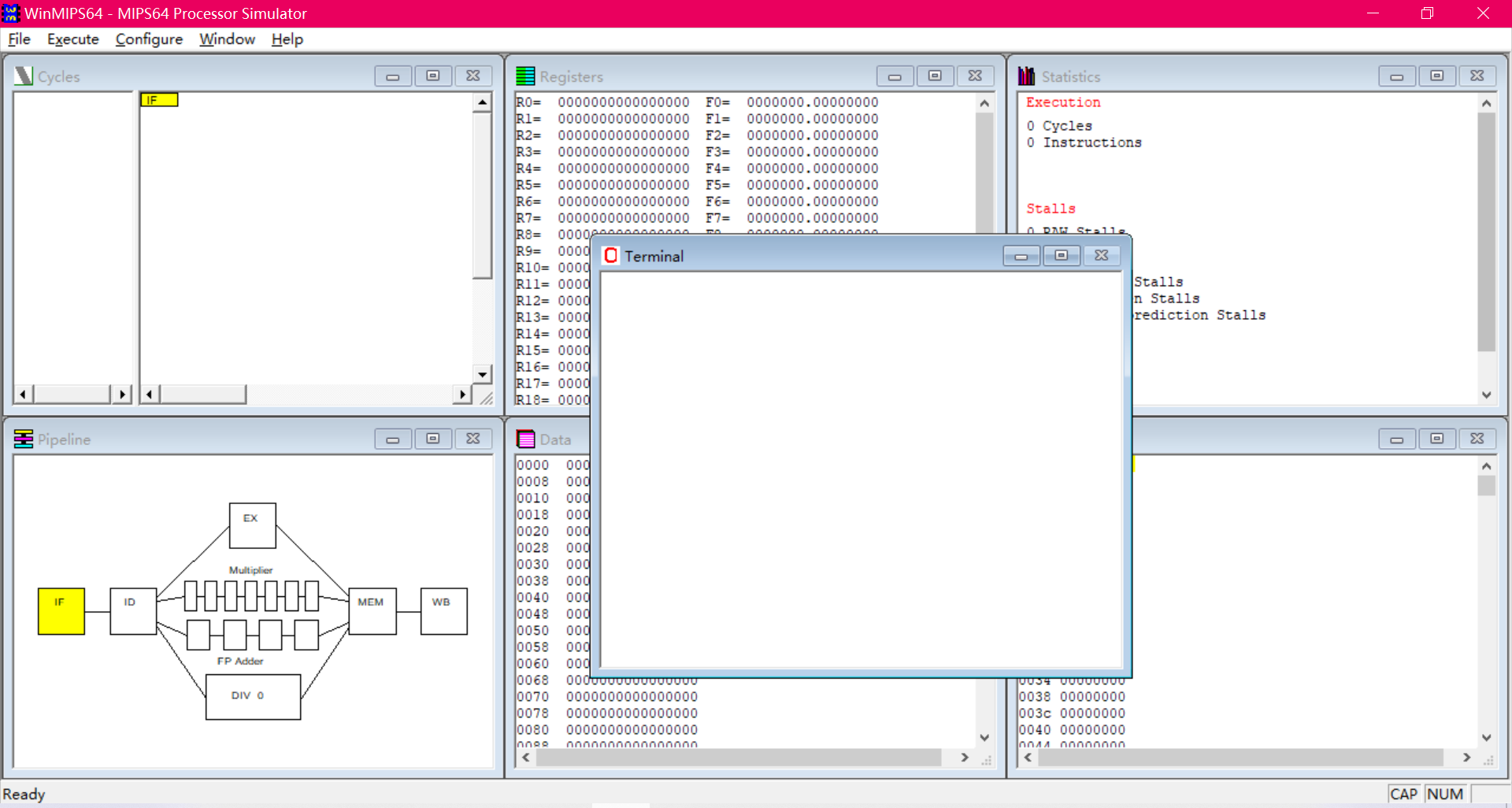
请按以下步骤在Windows下安装WinMIPS64 ：

1. 为WinMIPS64 创建目录，例如**D:\ WinMIPS64**
2. 解压给出的winmips64.zip压缩文件到创建的目录中。

（二）、一个完整的例子

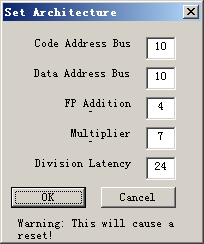
1. **开始和配置WinMIPS64**

在winmips64这个子目录下，双击winmips64.exe文件，即打开了WinMIPS64模拟器，其外观如下图：



为了初始化模拟器, 点击***File*** 菜单中的 ***Reset all*（Ctrl+R）** 菜单项即可。**OKOK**

WinMIPS64可以在多种配置下工作。你可以改变流水线的结构和时间要求、存储器大小和其他几个控制模拟的参数。点击 ***Configuration*** /***Floating Point Stages***（点击***Configuration***打开菜单，然后点击***Architecture***菜单项），选择如下标准配置：



如果需要，可以通过点击相应区域来改变设置。然后，点击***OK*** 返回主窗口。

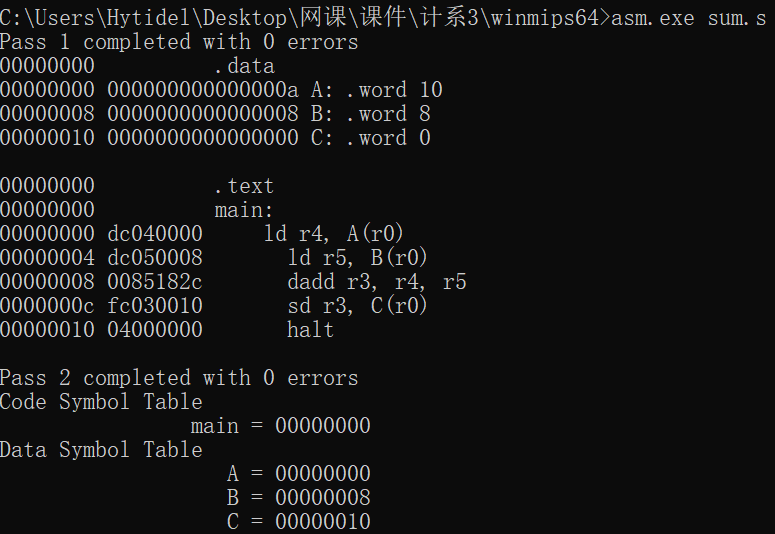
在 ***Configuration*** 菜单中的其他四个配置也可以设置，它们是：***Multi-Step, Enable Forwarding, Enable Branch Target Buffer*** 和 ***Enable Delay Slot***。 点击相应菜单项后， 在它的旁边将显示一个小钩。 本次实验要求不要勾选“Enable Forwarding”。

**2. 装载测试程序**

用标准的text编辑器来新建一个名为sum.s的文件，这个文件的功能是，计算两个整数A、B之和，然后将结果传给C。程序如下：

|  |
| --- |
| .data  A: .word 10  B: .word 8  C: .word 0  .text  main:  ld r4,A(r0)  ld r5,B(r0)  dadd r3,r4,r5  sd r3,C(r0)  halt |

在将该程序装载进WinMIPS64之前，我们必须用asm.exe来检验该输入程序的语法正确性。asm.exe程序文件在所给的winmips压缩包里有，用命令行使用它。具体操作为，打开终端，利用cd命令进到**D:\ WinMIPS64**目录中，然后直接使用asm.exe sum.s命令，检查输出结果是否无误。注意检查前需将 sum.s 置于 asm.exe 同目录下，输出如下图所示。



在开始模拟之前，至少应装入一个程序到主存。为此，选择***File*** / ***OPEN***，窗口中会列出当前目录中所有汇编程序，包括sum.s。

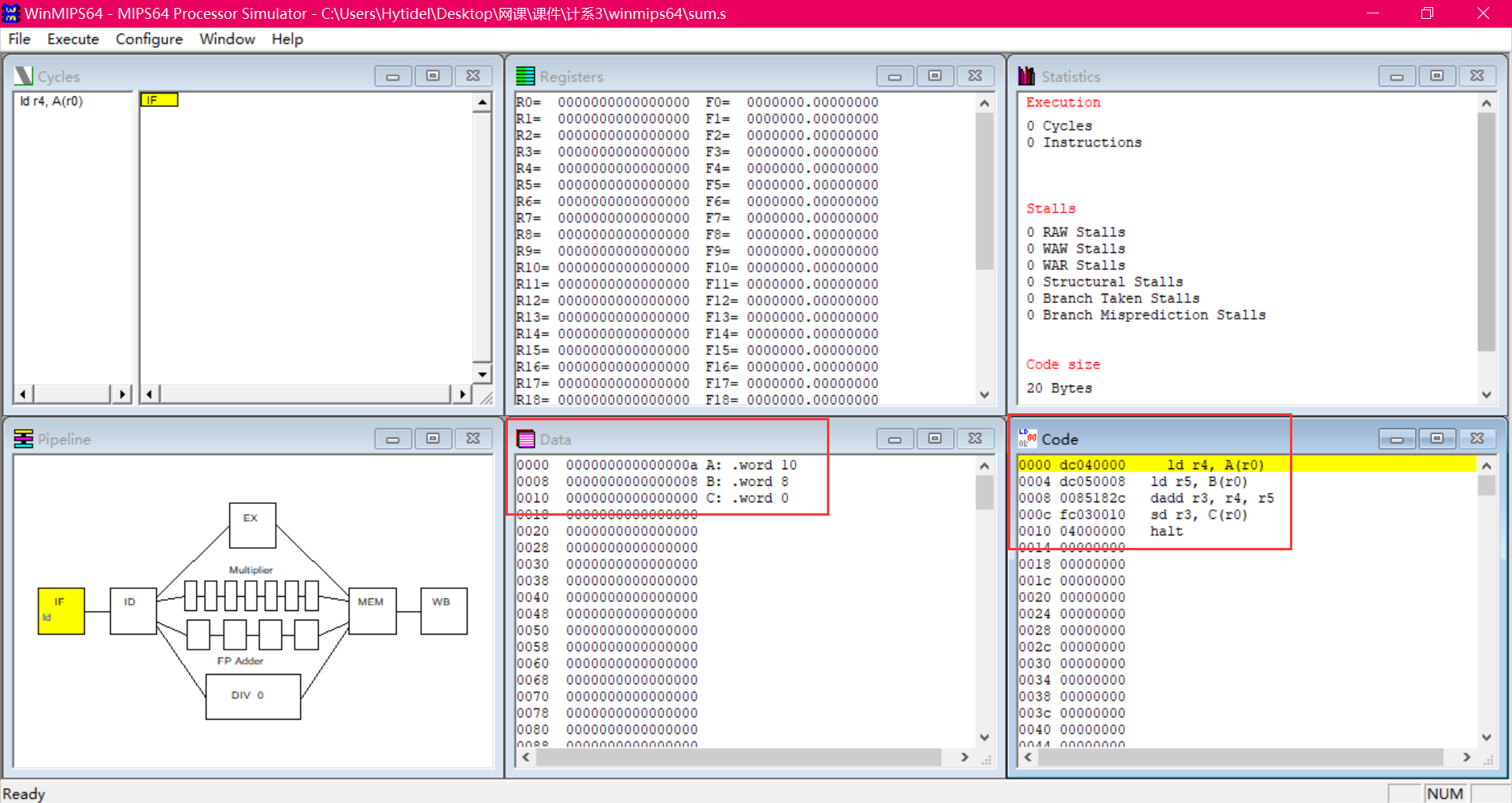
按如下步骤操作，可将这个文件装入主存。

点击 **sum.s**

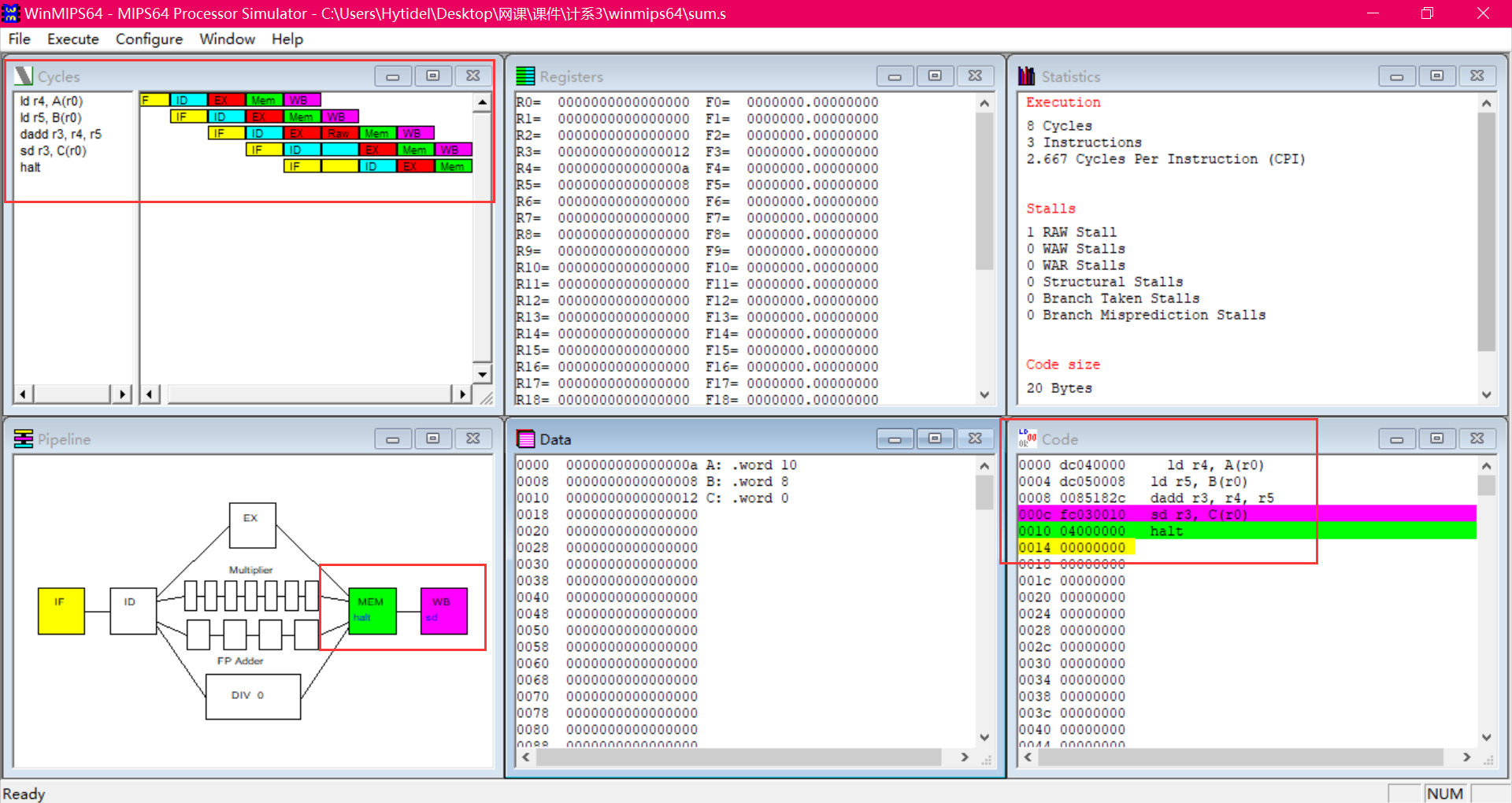
点击 ***open*** 按钮

现在，文件就已被装入到存储器中了，现在可以开始模拟工作了。

你可以在***CODE***窗口观察代码内容，可以在***DATE***窗口观察程序数据了。



按快捷键F7单步运行，观察到 Cycle 、Pipeline 、Code 窗口有变化。



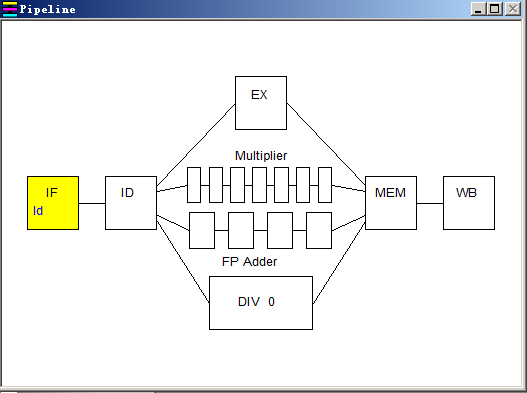
1. Pipleline 窗口展示处于不同流水阶段的指令。
2. Code 窗口展示指令所处的不同流水线阶段，其中黄色表示“取指”，蓝色表示“译码”，红色表示“执行”，绿色表示“内存数据读或写”，紫色表示“写回”。
3. Cycles 窗口展示流水线的时空图，在其中可观察到每条指令在 Pipeline 窗口中的路径，观察到这是典型的流水线模式。

**3. 模 拟**

在主窗口中，我们可以看见七个子窗口，和一条在底部的状态栏。这七个子窗口分别是**Pipeline**, **Code**, **Data**, **Registers**, **Statistics**, **Cycles**和**Terminal**。在模拟过程中将介绍每一个窗口的特性和用法。

#### (1) *Pipeline* 窗口

在***Pipeline***窗口中，展示了MIPS64处理器的内部结构，其中包括了MIPS64的五级流水线和浮点操作（加法/减法，乘法和除法）的单元。展示了处于不同流水段的指令。



#### (2) *Code* 窗口

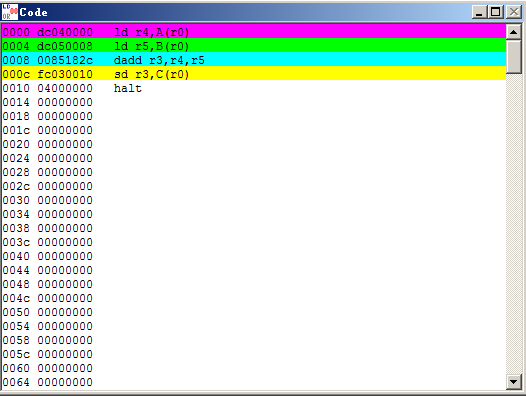
我们来看一下 ***Code*** 窗口。你将看到代表存储器内容的三栏信息，从左到右依次为：地址 (符号或数字)、命令的十六进制机器代码和汇编命令。

我们可以看到，初始时，第一行为黄色，表示该行指令处于“取指”阶段。

现在，点击主窗口中的 ***Execution***开始模拟。在出现的下拉式菜单中，点击***Single Cycle***或按 ***F7***键。

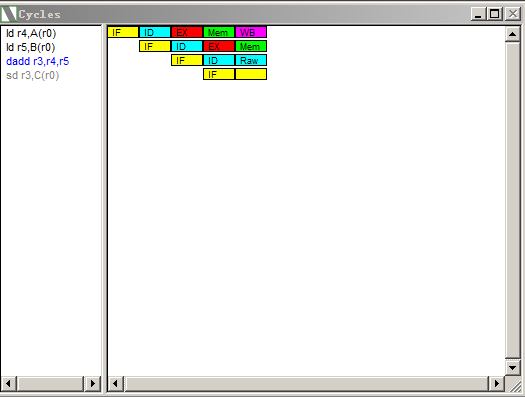
这时，第一行变成了了蓝色，第二行变成了黄色，这表示第一行指令处于“译码”阶段，而第二行指令处于“取指”阶段。这些不同的颜色代表指令分别处于不同的流水线阶段。黄色代表“取指”，蓝色代表“译码”，红色代表“执行”，绿色代表“内存数据读或写”，紫色代表“写回”。

接着按F7，直到第五个时钟周期的时候，有趣的事情发生了，“dadd r3，r4，r5”指令没有从“译码”跳到其下一个流水阶段“执行”，并且“sd r3，C(r0)”指令，仍然停留在“取指”阶段，同时在***terminal***窗口显示一行信息“RAW Stall in ID (RS)”，这是因为 dadd 指令需用到寄存器 r5 的值，但此时 r5 的值不可用。



#### (3) *Cycls*窗口

我们将注意力放到***Cycls***窗口上。它显示流水线的时空图。



在窗口中，你将看到模拟正在第五时钟周期，第一条指令正在WB段，第二条命令在MeM段，第四条命令在处于暂停状态（installed），第五条指令也因此停滞不前。这是因为发生了数据相关（第四条指令的dadd命令需要用到寄存器r5的值，但是r5的值并不可用）。在 Cycles 窗口中观察到，第四条指令的ID和IF模块未停止工作，EX模块处于等待中。

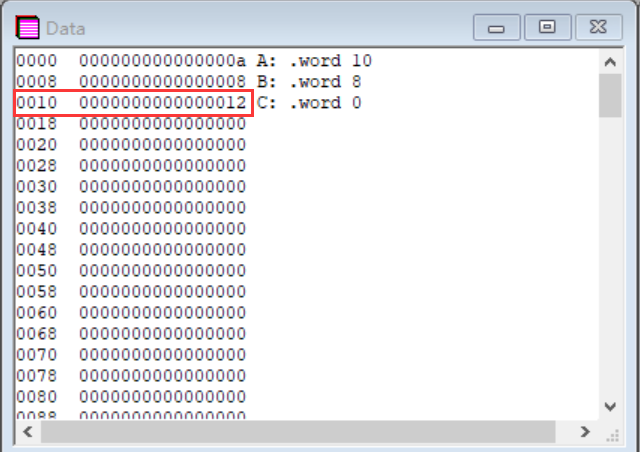
接着点击***F7***，到第五个时钟周期时，再次发生相关，造成停滞。接着点击***F7***，直至第十三个时钟周期全部指令执行结束。

值得一提的是，***Cycls***窗口是分为两个子窗口的，左边的子窗口是一系列的指令，右边的窗口是图示的指令执行过程。其中，左边子窗口的命令是动态出现的，当一条指令在进行“取指”时，该指令才出现，而且，当出现了数据相关的时候，所涉及到的指令会变色，暂停的指令会变成蓝色，而被其影响的后续指令会变成灰色。

#### (4) *Data* 窗口

在***Data***中，我们可以观察到内存中的数据，包括数据内容和地址两个方面，其中地址使用64位表示。

如果想改变一个整型的数据的值，左键双击该值所在的行，如果是想改变一个浮点类型的数据的值，那么请右键双击该值所在的行。

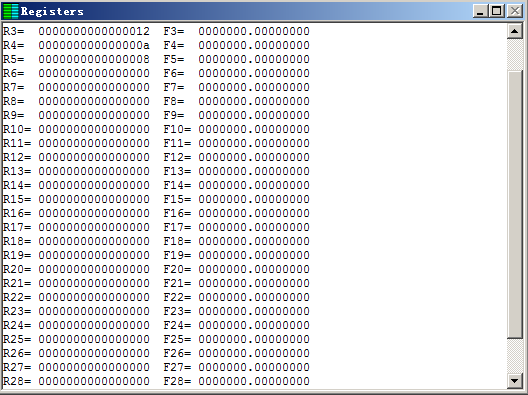


上图即为第十三个时钟周期的***data***窗口的图示，其中，左边一行即为用64位表示的内存地址，中间行为数据的内容，右边的一行为相关的代码。可以看出，在这个时钟周期，A与B的值分别为0xa和0x8，C的值为0x12，表明A与B的值之和已经相加并保存到了C中。

#### (5) *Registers* 窗口

这个窗口显示存储在寄存器中的值。

如果该寄存器为灰色，那么它正处于被一条指令写入的过程，如果它用一种颜色表示，那么就代表，该颜色所代表的的流水线阶段的值可以用来进行前递（forwarding）。同时，这个窗口允许你交互式的该变寄存器的值，但是前提是该寄存器不能处于被写入或者前递的阶段。如果想改变一个整型的数据的值，左键双击该值所在的行，如果是想改变一个浮点类型的数据的值，那么请右键双击该值所在的行，然后按***OK***来进行确定。

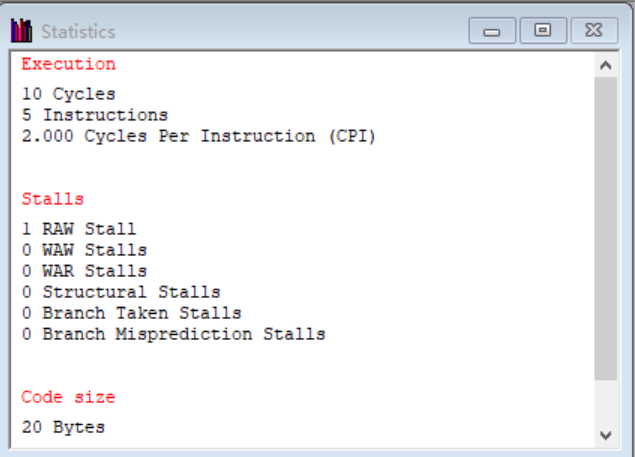


上图即为第十三个时钟周期的***Registers***窗口的图示，很显然，其中可以很清楚的看出每个寄存器的值是什么。

#### (6) *Statistics* 窗口

最后我们来看一下***Statistics*** 窗口。

这个窗口是用来记录一些模拟周期的统计数据。其中包括***Execution***，***Stalls***，和***Code Size***三个大项。其中，***Execution***用来显示模拟周期中指令数，执行周期数和CPI（没条指令所用周期数），***Stalls***用来表示暂停的周期数，并且分门别类的进行了统计，其中包括***RAW Stalls***，***WAW Stalls***，***WAR Stalls***, ***Structural Stalls***, ***Branch Taken Stalls***和***Branch misprediction Stalls***。***Code Size***表示了代码的大小，用***byte***表示。



上图即为***Statistics***窗口的图示，其中表示了该程序有10时钟周期，5条指令，CPI

为2.000，有11个***RAW Stalls***，代码大小为20个Bytes。

（三）、更多操作

首先，点击File/Reset MIPS64（ctrl + R）进行重置。如果你点击File/Full Reset，你将删除内存中的数据，这样你就不得不重新装载文件，所以点击File/Reload（F10）是一个很方便的重置的方法。

你可以一次推进多个时钟周期，方法是点击Execute/Multi cycle（F8），而多个时钟周期数是在Configure/Multi-step中设置的。

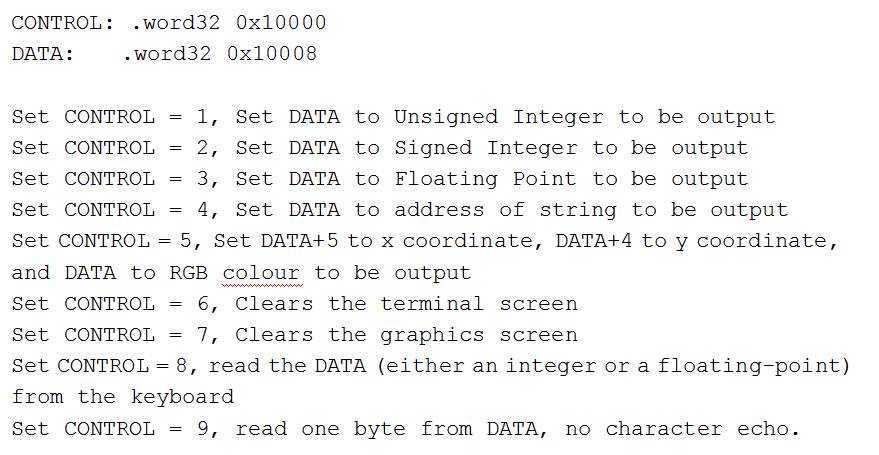
你也可以通过按F4一次完成整个程序的模拟。同时，你可以设置断点，方法是，在Code窗口中左键双击想要设置断点的指令，该指令会变成蓝色，然后点击F4，程序就会停在这条指令执行“ 取指”的阶段，如果想要清除断点，再次左键双击改行指令。

（四）、终端I/O的简单实例

通过上面对WinMIPS64的了解，我们可以开始简单的使用该工具了。

这里，需要我们编写一个简单的终端输出“Hello World！！”的小程序，运行并且截图。所以，我们需要了解如何将数据在终端中输出输入。

下图是I/O区域的内存映射，一个是控制字，一个是数据字：



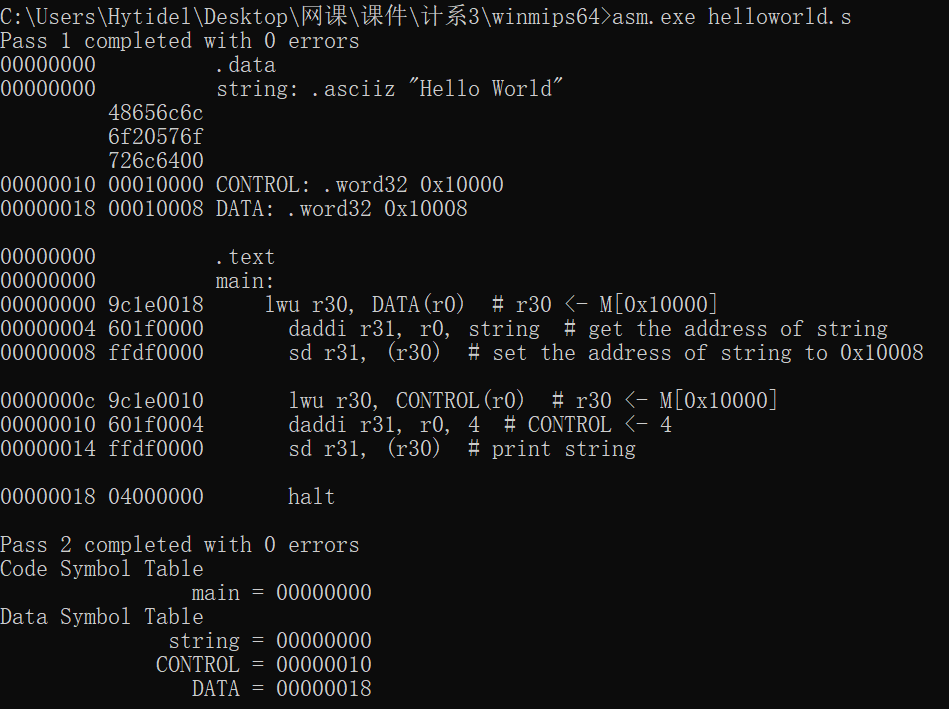
所以我们需要先将CONTROL和DATA地址读取到寄存器，然后分别在这两个区域内存储相应的序列号（如上图所示）和要显示在Terminal窗口的数据，同时，设置CONTROL为9，我们能对其进行读取数据。

请编写完整程序，输出“Hello World!”字符串。然后通过asm.exe来检验该程序的语法正确性，然后在WinMIPS64中的File栏中open打开文件。最后一步步按F7，同时观察各个窗口。

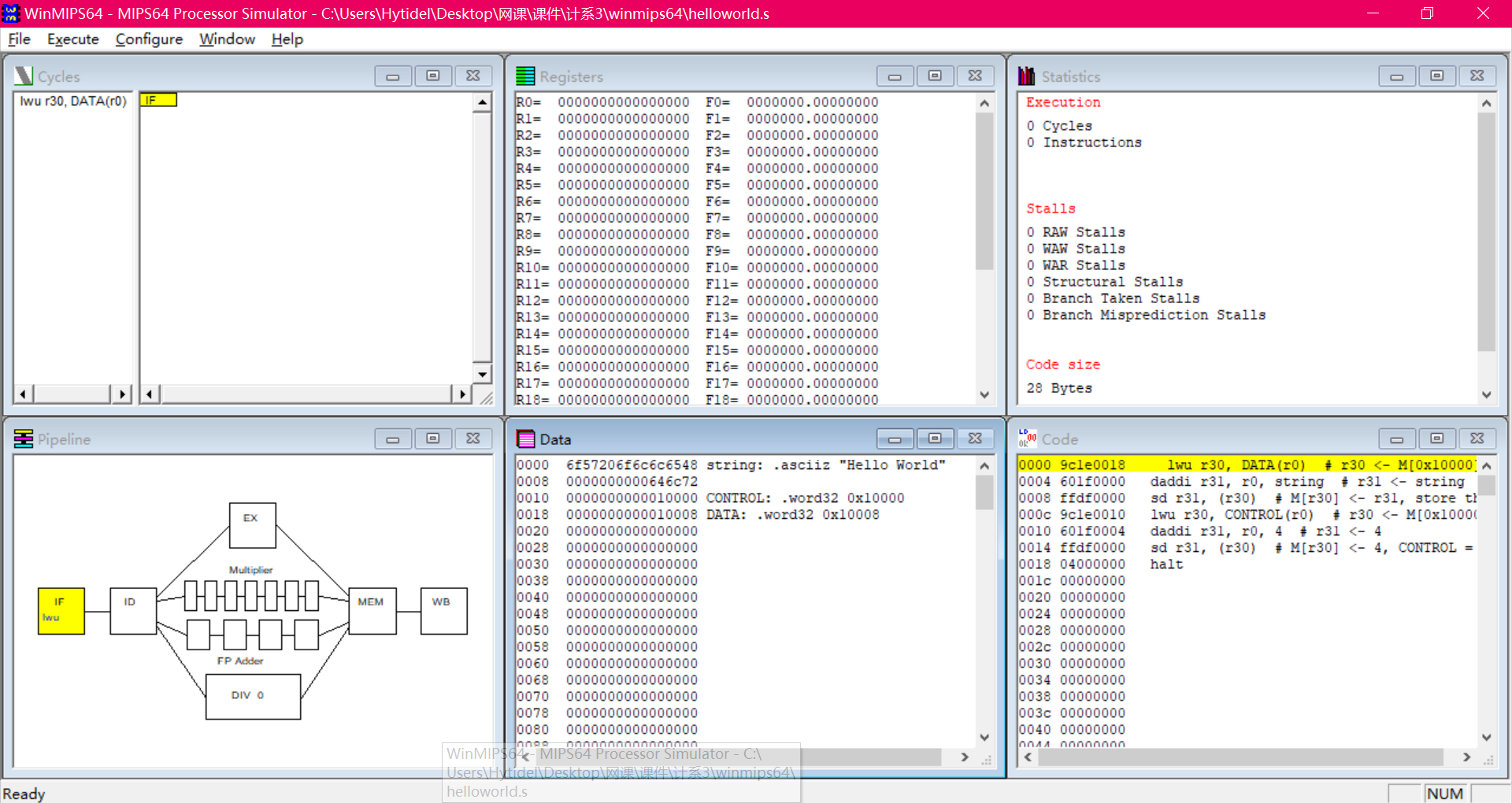
编写的程序如下。

|  |
| --- |
| .data  string: .asciiz "Hello World"  CONTROL: .word32 0x10000  DATA: .word32 0x10008  .text  main:  lwu r30, DATA(r0) # r30 <- M[0x10000]  daddi r31, r0, string # r31 <- string  sd r31, (r30) # M[r30] <- r31, store the string    lwu r30, CONTROL(r0) # r30 <- M[0x10000]  daddi r31, r0, 4 # r31 <- 4  sd r31, (r30) # M[r30] <- 4, CONTROL = 4    halt |

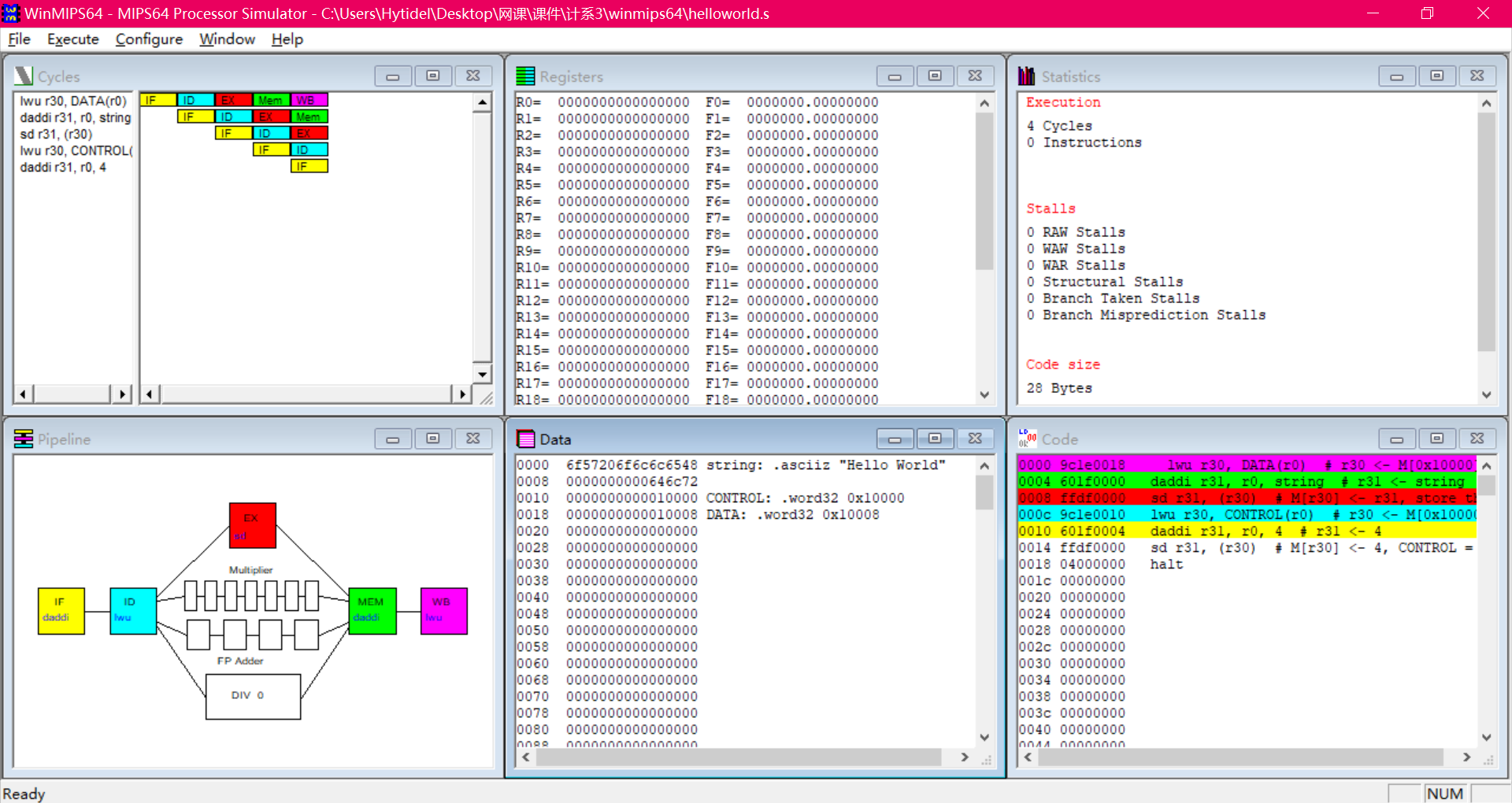
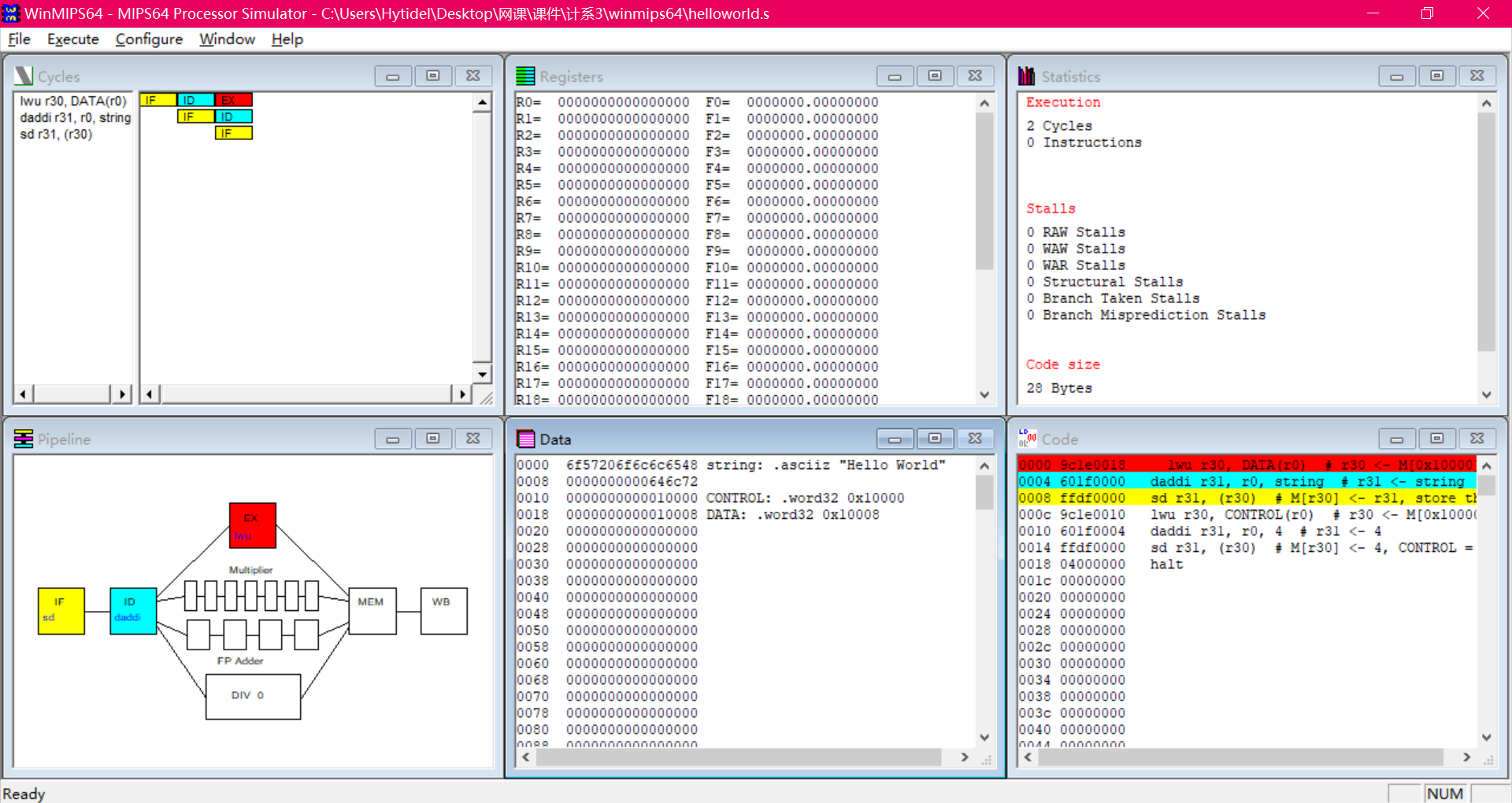
用 asm.exe 检验该程序的语法，发现没有错误。



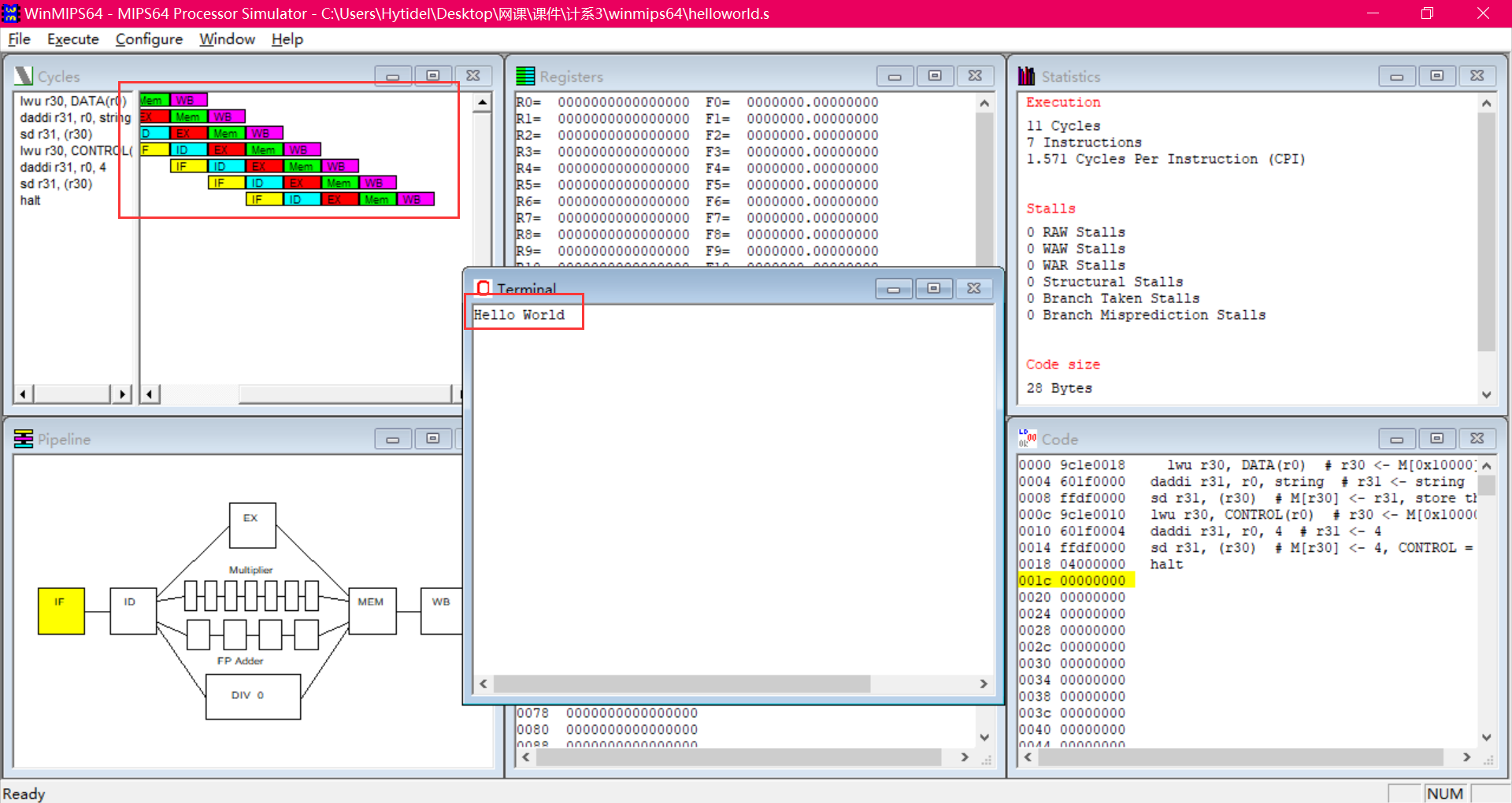
载入 helloworld.s ，如下图所示。



按 F7 单步运行，观察运行过程中各窗口的变化，观察到指令依次经过“取指”、“解码”、“执行”、“访存”、“写回”的过程，不同指令的该过程以流水线的形式执行，效率高。



执行完所有指令后，终端上出现“Hello World”。



（五）、编写排序算法

在这一部分，我们要求编写一个排序算法，对一组int型数据进行排序。该算法使用冒泡排序法，并且在其中嵌入一个swap函数过程（该算法在课本上有完整的程序，但是其中的数据初始化、寄存器映射、命令的映射以及I/O部分还需要自己手动编写）。编写完成后，在asm.exe中进行检测，然后运行。

初始数据要求为：“array: .word 8,6,3,7,1,0,9,4,5,2”

该程序需要对0到10（左闭右开），十个数进行了排序，其中使用了sort和swap两个函数过程，并且swap是嵌套在sort中的，在编写程序的时候一定要注意使用栈来保留寄存器的值，嵌套时还额外需要保存$ra的值。

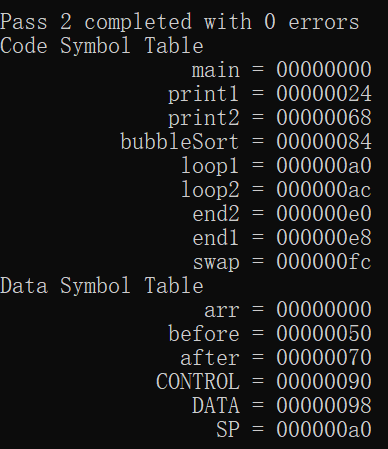
先写出该程序的 C/C++ 代码，如下。

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  int n = 10;  int a[] = { 8, 6, 3, 7, 1, 0, 9, 4, 5, 2 };  string before = "Before sort the array is:\n";  string after = "After sort the array is:\n";  void swap(int& a, int& b) {  int tmp = a;  a = b;  b = tmp;  }  void bubbleSort() {  for (int i = 0; i < n; i++) {  for (int j = i - 1; j >= 0; j--)  if (a[j] > a[j + 1]) swap(a[j], a[j + 1]);  }  }  int main() {  cout << before;  for (int i = 0; i < n; i++) cout << a[i] << '\n';  bubbleSort();  cout << after;  for (int i = 0; i < n; i++) cout << a[i] << '\n';  return 0;  } |

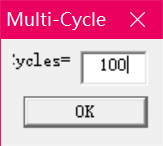
将其转化为MIPS指令集的代码，如下：

|  |
| --- |
| .data  arr: .word 8, 6, 3, 7, 1, 0, 9, 4, 5, 2  before: .asciiz "Before sort the array is:\n"  after: .asciiz "After sort the array is:\n"  CONTROL: .word 0x10000  DATA: .word 0x10008  SP: .word 0x300 # stack pointer  .text  main:  # print before  ld r16, CONTROL(r0) # r16 <- M[CONTROL]  ld r17, DATA(r0) # r17 <- M[DATA]  daddi r8, r0, 4 # r8 <- 4, CONTROL = 4  daddi r9, r0, before # r9 <- before  sd r9, (r17) # M[r17] <- r9  sd r8, (r16) # M[r16] <- r8, print before  # print the array  daddi r8, r0, 2 # r8 <- 2, CONTROL = 2  daddi r2, r0, 10 # r2 <- 10, n = 10  daddi r1, r0, 0 # r1 <- 0, i = 0  print1:  dsll r3, r1, 3 # r3 <- (r1 << 3, logically), r3 = i \* 8  ld r9, arr(r3) # r9 <- M[arr + r3], r9 = arr[i]  sd r9, (r17) # M[r17] <- r9, M[r17] = arr[i]  sd r8, (r16) # print arr[i]    daddi r1, r1, 1 # r1 <- r1 + 1, i++  bne r1, r2, print1 # goto print1 if r1 != r2, i != n  # bubbleSort()  ld r29, SP(r0) # r29 <- M[SP], use r29 to maintain the stack pointer  daddi r4, r0, arr # para1: a[]  daddi r5, r0, 10 # para2: n  jal bubbleSort # goto bubbleSort  # print after  daddi r8, r0, 4 # r8 <- 4, CONTROL = 4  daddi r9, r0, after # r9 <- after  sd r9, (r17) # M[r17] <- r9  sd r8, (r16) # M[r16] <- r8, print after    # print the array  daddi r8, r0, 2 # r8 <- 2, CONTROL = 2  daddi r2, r0, 10 # r2 <- 10, n = 10  daddi r1, r0, 0 # r1 <- 0, i = 0  print2:  dsll r3, r1, 3 # r3 <- (r1 << 3), r3 = i \* 8  ld r9, arr(r3) # r9 <- M[arr + i \* 8]  sd r9, (r17) # M[r17] <- r9  sd r8, (r16) # M[r16] <- r8, print arr[i]    daddi r1, r1, 1 # r1 <- r1 + 1, i++  bne r1, r2, print2 # goto print2 if r2 != r1, loop if i != 10  halt  # ----------------------------------------  bubbleSort:  # backup  daddi r29, r29, -24 # r29 <- r29 - 24, allocate stack frames for 3 ints  sd $ra, 16(r29) # store PC  sd r16, 8(r29) # store r16  sd r17,0(r29) # store r17  dadd r22, r4, r0 # r22 <- r4, r2 = a[]  daddi r23, r5, 0 # r23 <- r5, r23 = n  and r18, r18, r0 # r18 <- r18 & r0, i = 0  loop1:  # for (int i = 0; i < n; i++)  slt r10, r18, r23 # set r10 = 1 if r18 < r23, loop1 if i < n  beq r10, r0, end1 # goto end1 if r10 == r0, break if i >= n    daddi r19, r18, -1 # r19 <- r18 - 1, j = i - 1  loop2:  # for (int j = i - 1; j >= 0; j--)  slti r10, r19, 0 # set r10 = 1 if r19 < 0, not loop2 if j < 0  bne r10, r0, end2 # goto end2 if r10 != r0, break if j < 0    dsll r11, r19, 3 # r11 <- (r19 << 3), r11 = j \* 8  dadd r12, r22, r11 # r12 = r22 + r11, r12 = a + j \* 8  ld r13, 0(r12) # r13 <- M[r12], r13 = a[j]  ld r14, 8(r12) # r14 <- M[r12 + 8], r14 = a[j + 1]    # if (a[j] > a[j + 1]) swap(a[j], a[j + 1]);  slt r10, r14, r13 # set r10 = 1 if r14 < r13, swap if a[j + 1] < a[j]  beq r10, r0, end2 # goot end2 if r10 == r0, continue if a[j + 1] >= a[j]    #swap(a[j], a[j + 1])  dadd r4, r0, r12 # r4 <- r12, para1: a + j \* 8  daddi r5, r12, 8 # r5 <- r12 + 8, para2: a + (j + 1) \* 8  jal swap # goto swap    daddi r19, r19, -1 # r19 <- r19 - 1, j--  j loop2 # goto loop2    end2:  daddi r18,r18,1 # r18 <- r18 - 1, i--  j loop1 # goto loop1  end1:  # restore  ld r17, 0(r29) # restore r17  ld r16, 8(r29) # restore r16  ld $ra, 16(r29) # restore PC  daddi r29, r29, 24 # recover the stack frames  jr $ra # return to M[PC]  # ----------------------------------------  swap:  # swap(a[i], a[j])  ld r9, 0(r4) # r9 <- M[r4], r9 = a[i]  ld r10, 0(r5) # r10 <- M[r5], r10 = a[j]  sd r10, 0(r4) # M[r4] <- r10, a[i] = r10  sd r9, 0(r5) # M[r5] <- r9, a[j] = r9  jr $ra # return to M[PC] |

用 asm.exe 检查 sort.s 的语法，未发现错误，如下图所示。

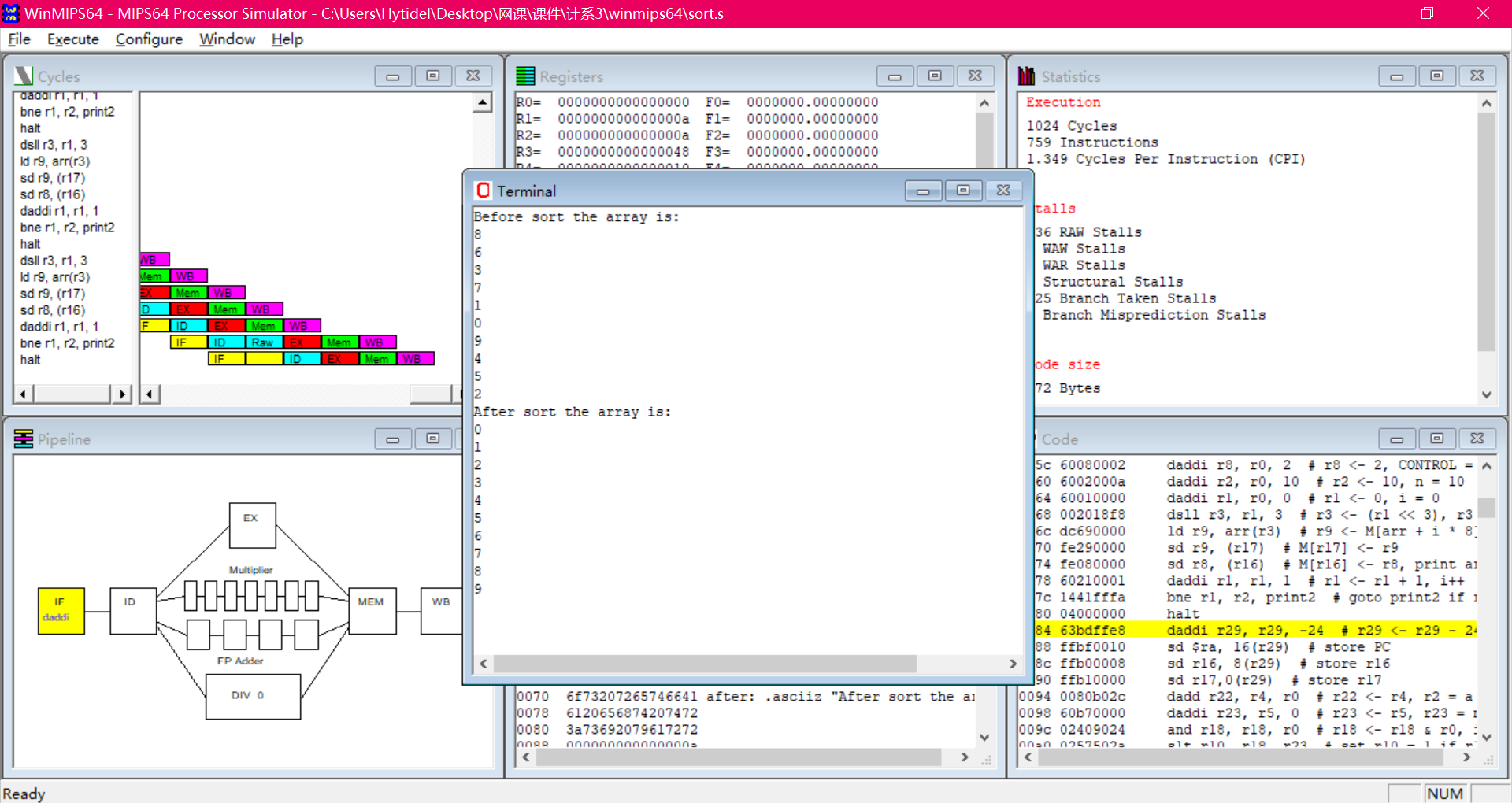


因冒泡排序的指令数较多，所需的时钟周期较多，可在Configure – Multi-step 中设置一次推进的时钟周期数，本次实验中将其设为100。



载入 sort.s 后，用 F8 一次推进多个时钟周期，观察各窗口和终端的变化。

结果如下图所示，发现序列已升序排列。



注意：需要将SP初始化为内存最高地址，否则为初始化SP为0，SP-1将指向FFFFFFFF,该地址将超出winmips默认的内存空间。

但事实上，为该程序分配足够的栈帧即可。上述代码中将SP初始化为0x300。

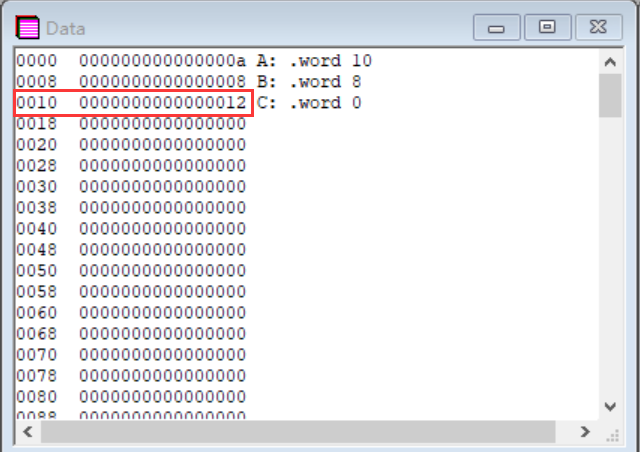
（六）、结束语

本实验通过一个例子介绍了WinMIPS64的重要特性，使你对流水线和MIPS64的操作类型有了一定的了解。当然，你还必须学习更多的知识，才能更深入地了解WinMIPS64。请参阅在winmips.zip压缩文件中的相关资料。

**五、实验结果**

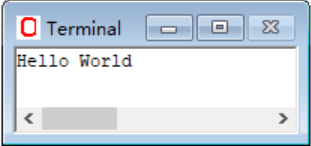
5.1 sum.s

观察到 地址A的值（十进制10） + 地址B 的值（十进制8）存放在地址C（十六进制12，即十进制18）中。



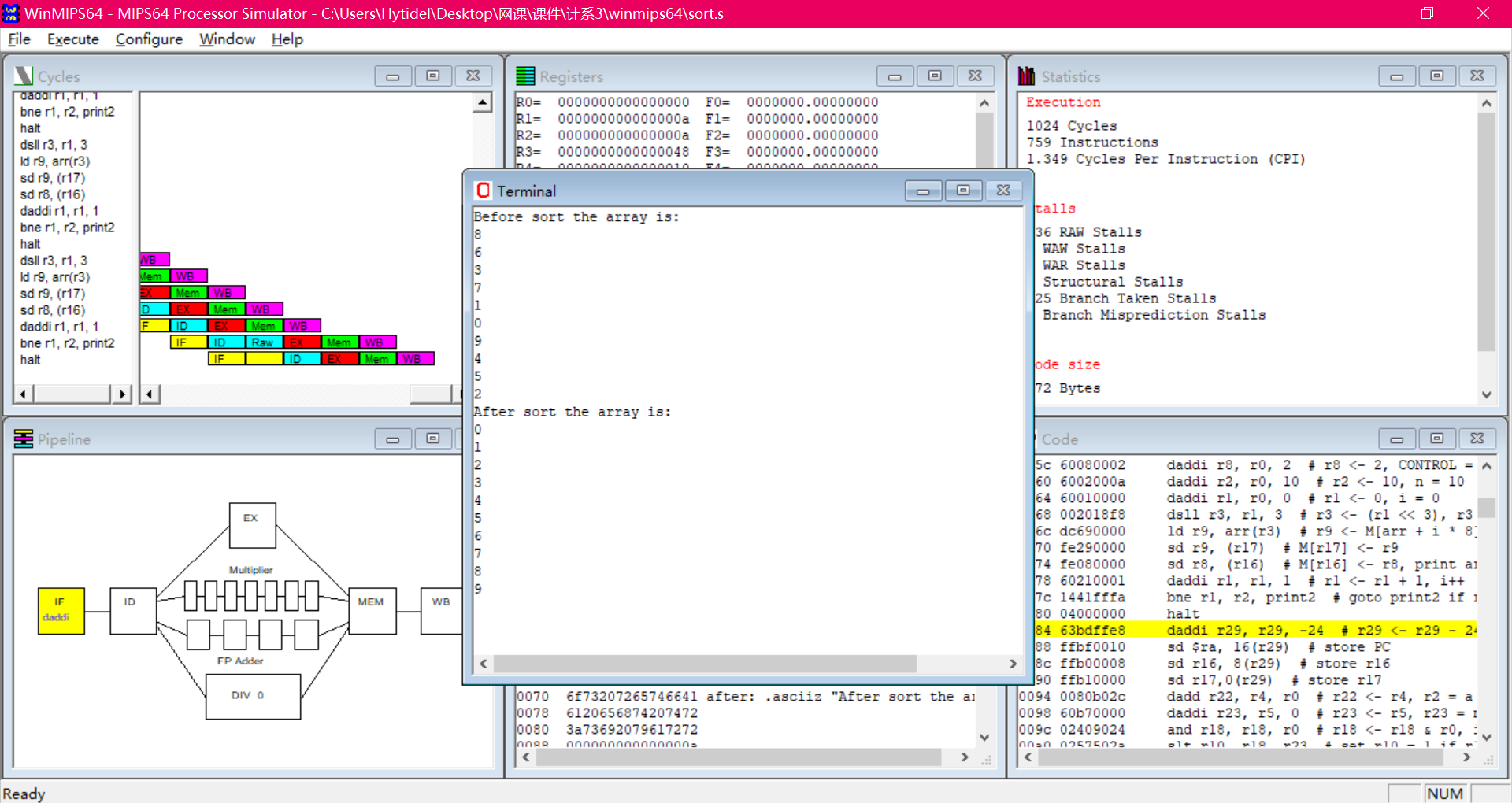
5.2 helloworld.s

执行完所有指令后，观察到终端上出现“Hello World”。



5.3 sort.s

执行完所有指令后，观察到序列已升序排列。



**五、实验总结与体会**

在MIPS指令集实验中，我学习并掌握了WinMIPS64软件的使用方法，以及指令的流水线阶段和终端I/O操作。通过实验，我成功地实现了一个在终端上输出"Hello World"的程序，并利用冒泡排序算法编写了一个能够将序列升序排列的程序。

在学习WinMIPS64软件方面，我了解了如何载入程序、运行程序以及进行调试。我学会了如何设置断点和单步执行指令，这对于调试程序中的错误非常有帮助。我也熟悉了各个窗口的作用，如指令监视窗口、寄存器监视窗口和内存监视窗口等。这些窗口提供了对程序执行过程中寄存器和内存的实时监控，方便我进行调试和观察程序运行结果。

了解指令的流水线阶段是该实验的重要一部分。我学习了指令的取值、译码、执行、内存数据读或写、写回等阶段，并了解了每个阶段的作用和所需的控制信号。对于理解指令执行过程和优化程序性能非常有帮助。

在终端I/O方面，我学会了使用系统调用指令来进行输入输出操作。通过编写程序，我成功地实现了在终端上输出"Hello World"的功能。这让我更好地理解了程序与用户之间的交互，并加深了对指令的理解。

最后，通过编写冒泡排序程序，我有效地应用了所学的知识。冒泡排序算法是经典的排序算法，通过比较和交换相邻元素实现排序。我使用MIPS指令集的加载、存储和比较指令，实现了将输入序列升序排列的功能。这个过程对我来说是一个巩固知识和提高编程技巧的机会。

通过这次实验，我不仅学会了WinMIPS64软件的使用和指令的流水线阶段，还掌握了在终端上进行输入输出操作和利用指令集实现算法逻辑的能力。这次实验让我更深入地了解了计算机体系结构和指令级编程的基本原理。我相信这些所学对于我的计算机科学学习和职业发展将大有裨益。

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  指导教师签字：  年 月 日 |
| 备注： |