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：计算机科学与技术

指 导 教 师：刘刚

报告人：林宪亮　学号：2022150130　班级：国际班

实 验 时 间：2023 年 9 月 25 日

实验报告提交时间：2023 年 10 月 7 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

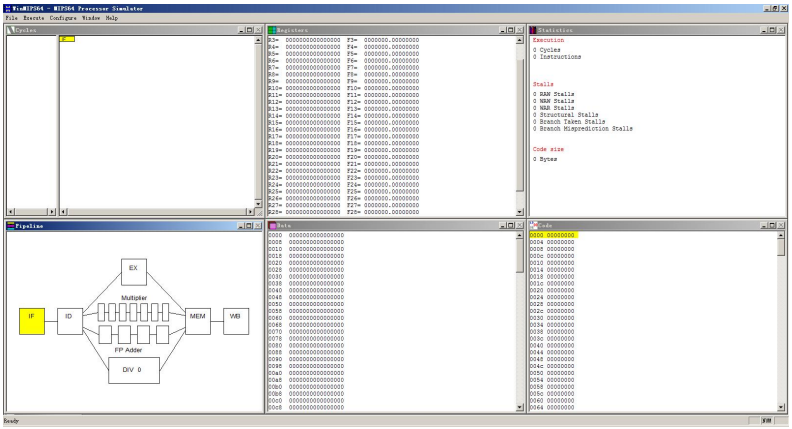
请按以下步骤在 Windows 下安装 WinMIPS64 ：

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

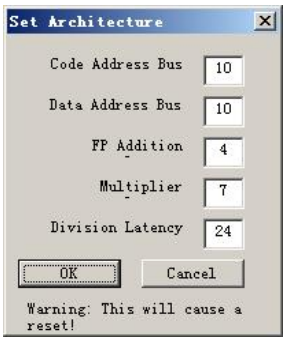
1. 开始和配置 WinMIPS64

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。

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。

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。

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。

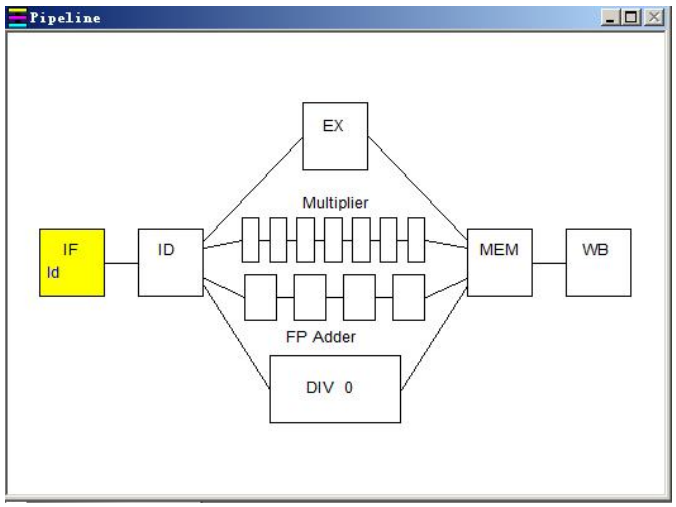
你可以在***CODE***窗口观察代码内容，可以在***DATE***窗口观察程序数据了。

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

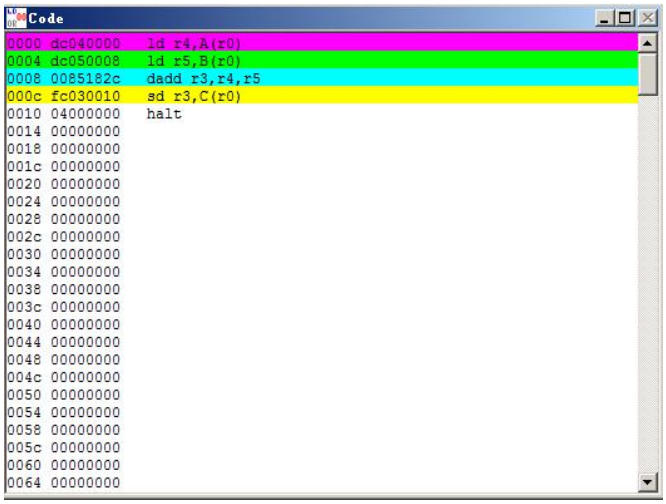
我们来看一下 ***Code*** 窗口。你将看到代表存储器内容的三栏信息，从左到右依次为：地址 (符号或数字)、命令的十六进制机器代码和汇编命令。

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。

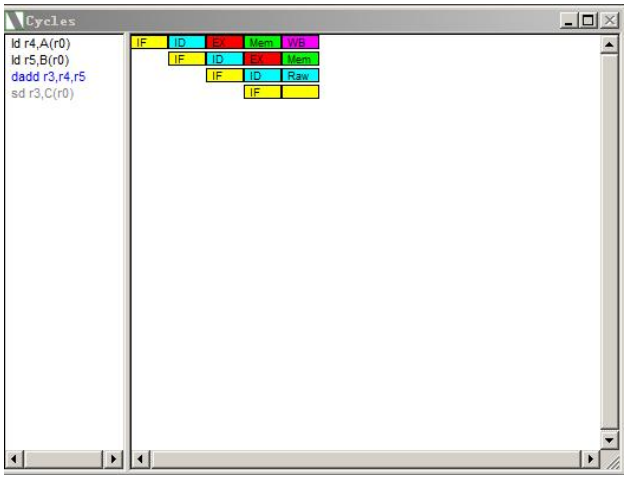
这时，第一行变成了了蓝色，第二行变成了黄色，这表示第一行指令处于“译码”阶段，而第二行指令处于“取指”阶段。这些不同的颜色代表指令分别处于不同的流水线阶段。黄色代表“取指”，蓝色代表“译码”，红色代表“执行”，绿色代表“内存数据读或写”，紫色代表“写回”。

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)”，思考一下这是为什么？



(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）。

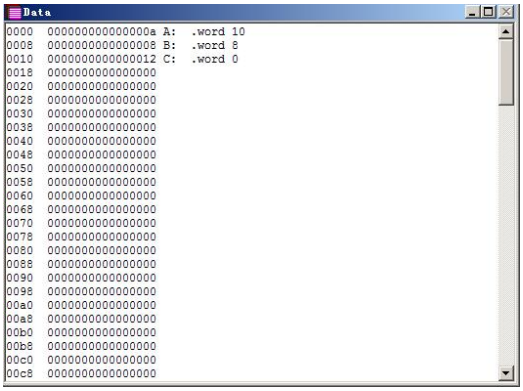
接着点击***F7***，到第五个时钟周期时，再次发生相关，造成停滞。接着点击***F7***，直至第十三个时钟周期全部指令执行结束。

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。

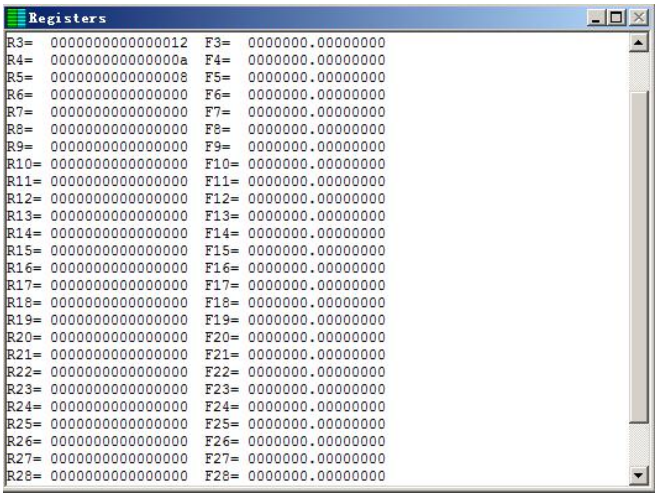


上图即为第十三个时钟周期的***data***窗口的图示，其中，左边一行即为用64位表示的内存地址，中间行为数据的内容，右边的一行为相关的代码。可以看出，在这个时钟周期，A与B的值分别为0xa和0x8，C的值为0x12，表明A与B的值之和已经相加并保存到了C中。

(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。

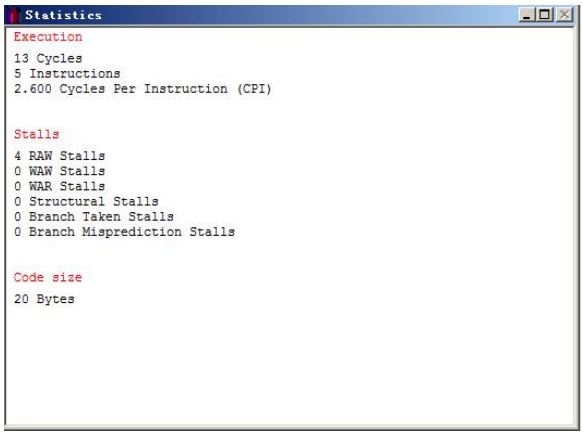


上图即为第十三个时钟周期的***Registers***窗口的图示，很显然，其中可以很清楚的看出每个寄存器的值是什么。

⑹ *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13个时钟周期，5条指令，CPI为2.600，有4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。

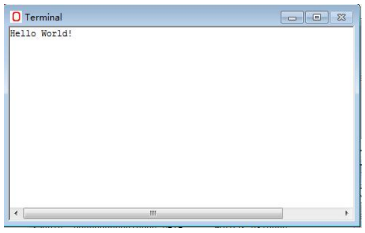
下图是 I/O 区域的内存映射，一个是控制字，一个是数据字：

```
CONTROL: .word32 0x10000
DATA:    .word32 0x10008

Set CONTROL = 1, Set DATA to Unsigned Integer to be output
Set CONTROL = 2, Set DATA to Signed Integer to be output
Set CONTROL = 3, Set DATA to Floating Point to be output
Set CONTROL = 4, Set DATA to address of string to be output
Set CONTROL = 5, Set DATA+5 to x coordinate, DATA+4 to y coordinate,
and DATA to RGB colour to be output
Set CONTROL = 6, Clears the terminal screen
Set CONTROL = 7, Clears the graphics screen
Set CONTROL = 8, read the DATA (either an integer or a floating-point)
from the keyboard
Set CONTROL = 9, read one byte from DATA, no character echo.
```

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。最终还要截取 Terminal 窗口，图如下：



证明你的程序结果输出了“Hello World！”。

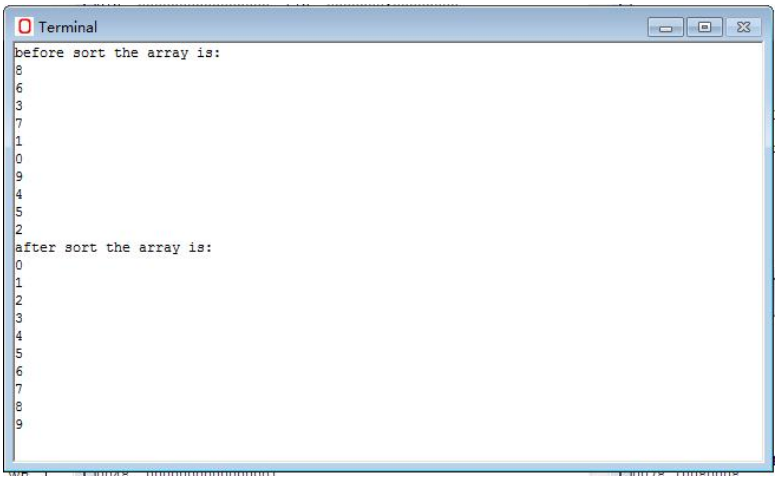
（五）、编写排序算法

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 10，十个数进行了排序，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。在 WinMIPS64 运行上述程序，将得到如下结果（这里只给出 Terminal 窗口的截图即可）：



观察实验截图， 证明你的程序确实做到了对数组排序的效果。

（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

1. 软件安装

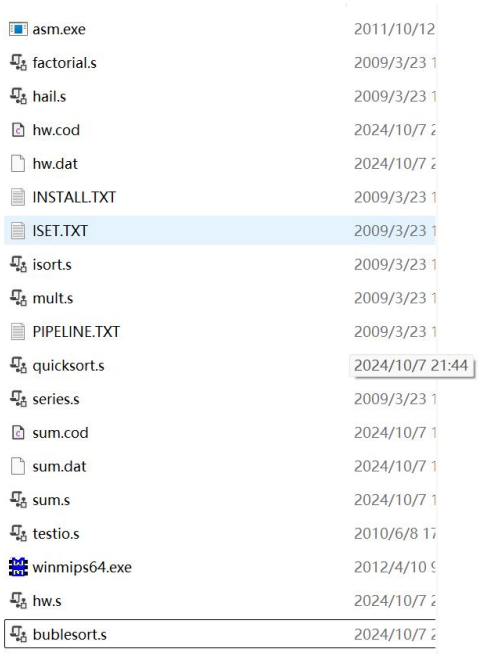


图 1 软件安装

如图 1，我成功安装了所需要的软件。

2. 示例代码运行

```
D:\APP\WinMIPS64>asm.exe sum.s
Pass 1 completed with 0 errors
00000000          .data
00000000 000000000000000a A:     .word 10
00000008 0000000000000008 B:     .word 8
00000010 0000000000000000 C:     .word 0

00000000          .text
00000000          main:
00000000 dc040000 ld r4,A(r0)
00000004 dc050008 ld r5,B(r0)
00000008 0085182c dadd r3,r4,r5
0000000c fc030010 sd r3,C(r0)
00000010 04000000 halt

Pass 2 completed with 0 errors
Code Symbol Table
                  main = 00000000
Data Symbol Table
                   A = 00000000
                   B = 00000008
                   C = 00000010
```

图 2 检测

如图，测试的代码通过了检测。
下面使用软件进行模拟运行。

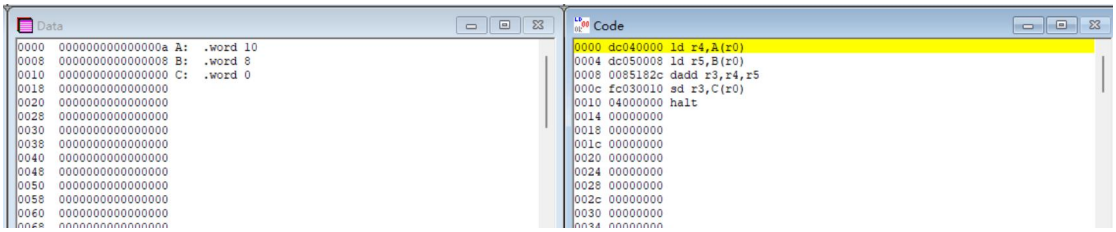


图 3 模拟

如图为进行模拟的过程。由于过程示例中已经写的足够详细，就不再赘述。

3. 输出“Hello World”

```
.data
CONTROL:  .word32 0x10000
DATA:     .word32 0x10008
HW: .asciiz "Hello World!\n"     //字符串

.text
lwu $t8,DATA($zero) ;将DATA的值加载到t8
lwu $t9,CONTROL($zero);将CONTROL的值加载到t9
daddi $v0,$zero,4    ;设置$v0为ASCII输出
daddi $t1,$zero,HW   ;将字符串的地址加载到t1
sd $t1,0($t8)    ;将字符串的地址写入DATA
sd $v0,0($t9)    ;打印输出

halt   ;停止程序
```

图 4 输出代码

代码思路：

首先，在数据段中定义了两个内存地址 CONTROL 和 DATA，分别用于控制输出和存储字符串的地址。代码段中首先将 DATA 的值加载到寄存器 $t8，并将 CONTROL 的值加载到 $t9。

接着，通过 daddi 指令将数字 4 加载到寄存器 $v0，这表示进行 ASCII 字符串输出的系统调用。同时，字符串 "Hello World!\n" 的地址被加载到寄存器 $t1。随后，代码使用 sd 指令将字符串的地址存储到 DATA 指向的内存位置，并将系统调用编号存储到 CONTROL 指向的内存地址。

```
D:\APP\WinMIPS64>asm.exe hw.s
Pass 1 completed with 0 errors
00000000           .data
00000000 00010000 CONTROL: .word32 0x10000
00000008 00010008 DATA: .word32 0x10008
00000010           HW:    .asciiz "Hello World!\n"    //STRING
         48656c6c
         6f20576f
         726c6421
         0a00

00000000           .text
00000000 9c180008 lwu $t8,DATA($zero) ;store DATA in t8
00000004 9c190000 lwu $t9,CONTROL($zero);store CONTROL in t9
00000008 60020004 daddi $v0,$zero,4   ;set  for ascii output
0000000c 60090010 daddi $t1,$zero,HW  ;load String in t1
00000010 ff090000 sd $t1,0($t8)   ;write address of the string to data
00000014 ff220000 sd $v0,0($t9)   ;print

00000018 04000000 halt

Pass 2 completed with 0 errors
Code Symbol Table
Data Symbol Table
            CONTROL = 00000000
               DATA = 00000008
                 HW = 00000010
```

图 5 测试

如图，我们的代码通过了测试。

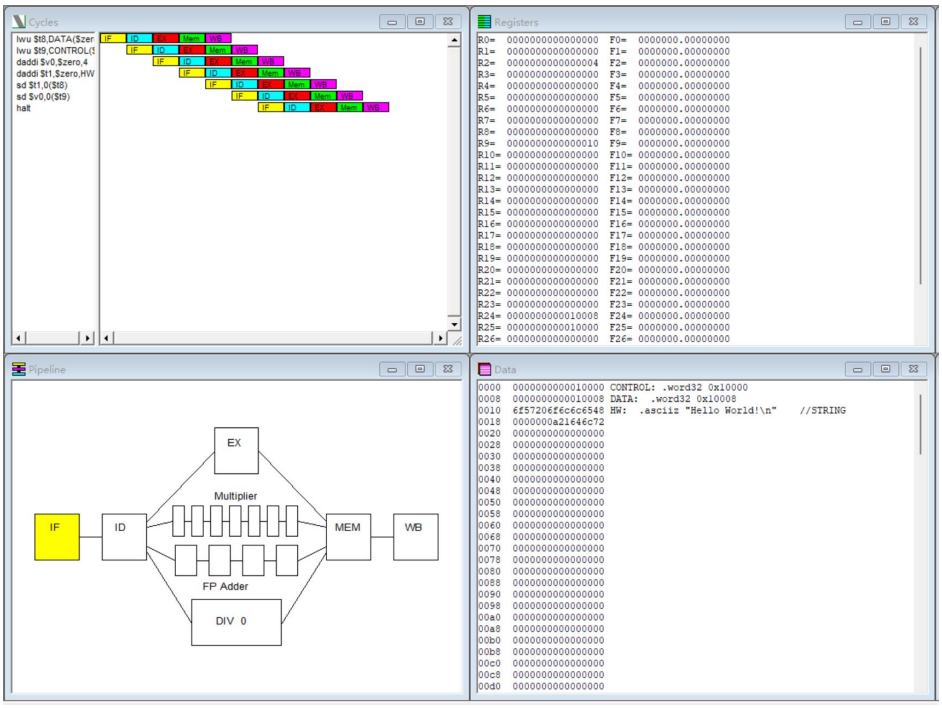


图 6 执行

将代码载入软件并逐步执行。

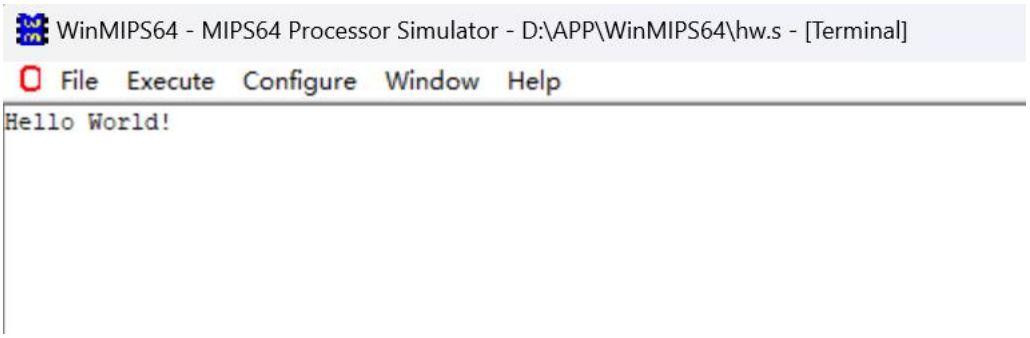


图 7 输出

可以看到我们的代码成功输出“Hello World！”

4. 冒泡排序

```
LOOPi:
    daddi $t1, $t1, 1          # Increment outer loop index (i++)
    daddi $t2, $t1, 1          # Set inner loop index (j = i + 1)

    LOOPj:
        dadd $a0, $zero, $t1  # Set a0 to i (index of first element in comparison)
        dadd $a1, $zero, $t2  # Set a1 to j (index of second element in comparison)
        daddi $t0, $zero, 8   # Set t0 to 8 (word size in bytes)
        dmul $a0, $t0, $a0    # Multiply i by word size to get offset in the array
        dmul $a1, $t0, $a1    # Multiply j by word size to get offset in the array
        jal swap              # Call the swap function to compare and swap elements if nee

        # Increment inner loop index (j++)
        BACK:
        daddi $t2, $t2, 1
        slti $t5, $t2, 10     # Check if j < 10
        bne $t5, $zero, LOOPj # Continue inner loop if j < 10

    slti $t5, $t1, 9          # Check if i < 9
    bne $t5, $zero, LOOPi     # Continue outer loop if i < 9
```

图 8 冒泡排序部分代码

首先，在数据段中定义了几个变量：CONTROL 和 DATA 用于存储内存地址，array 是一个包含待排序整数的数组，before 和 after 是用于输出的字符串，指示数组排序前后的状态。

在代码段中，首先加载 DATA 和 CONTROL 的值到寄存器 $t8 和 $t9 中。接下来，将字符串 before 的地址加载到 $t3 中，并通过 sd 指令将其存储到 DATA 指向的地址，随后通过设置 $v0 为 4 进行字符串输出，打印出排序前的数组状态。

然后，代码设置循环变量，准备打印数组的每个元素。通过一个循环，依次加载 array 中的值并打印，每次打印后更新循环计数器，直到打印完所有元素。接下来，进入冒泡排序的实现部分。外层循环控制当前已排序的元素位置，内层循环用于比较相邻元素并进行必要的交换。通过调用 swap 函数，判断两个元素的大小并执行交换操作，直到所有元素按升序排列。

完成排序后，再次加载 DATA 和 CONTROL 的值，准备输出字符串 after，并打印排序后的数组状态。最后，通过一个循环打印已排序的数组。

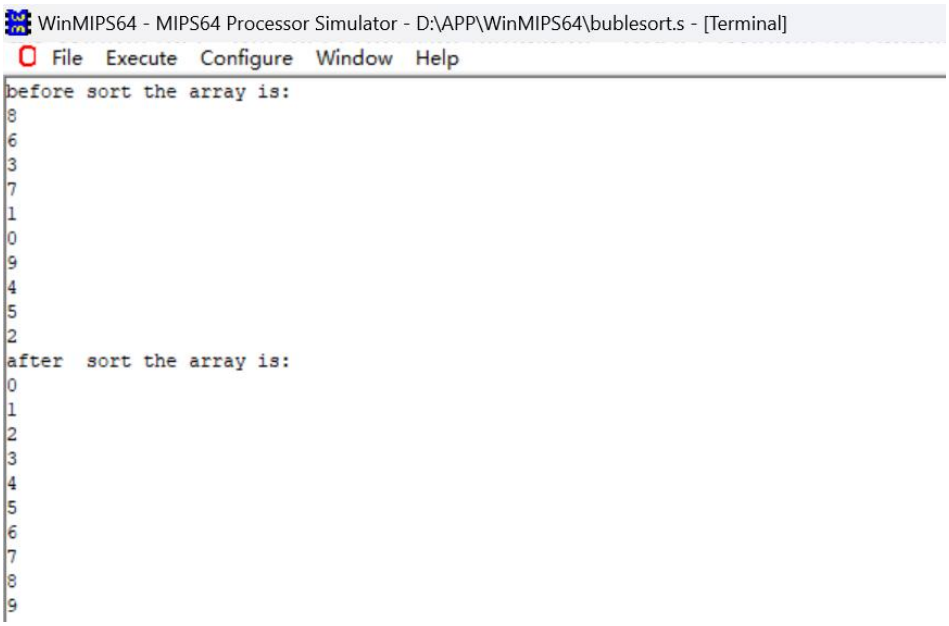


图 9 运行结果图

如图 9，我成功使用冒泡排序对数组进行排序。

5. 快速排序

```
QSORT:
    daddi $sp, $sp, -24       # Create stack frame
    sd $ra, 0($sp)            # Save return address
    sd $a0, 8($sp)            # Save low index (l)
    sd $a1, 16($sp)           # Save high index (h)

    slt $t0, $a0, $a1         # Check if l < h (low < high)
    beq $t0, $zero, OUT       # If not, exit function

    dadd $s0, $zero, $a0      # Set $s0 = i (start index)
    dadd $s1, $zero, $a1      # Set $s1 = j (end index)
    daddi $s2, $zero, 0       # Set flag = 0

    # Partition loop: while i < j
    WHILE:
        slt $t0, $s0, $s1     # Check if i < j
        beq $t0, $zero, FUNC  # If not, partition is done

        daddi $t7, $zero, 8   # Set word size to 8 (array element size)
        dmul $t1, $s0, $t7    # Calculate address for a[i]
        dmul $t2, $s1, $t7    # Calculate address for a[j]
        ld $t4, array($t1)    # Load a[i] into $t4
        ld $t5, array($t2)    # Load a[j] into $t5
        slt $t0, $t5, $t4     # Check if a[j] < a[i]
        beq $t0, $zero, else2 # If not, continue
```

图 10 快速排序部分代码

代码解释：

首先，在数据段中定义了一些变量，包括用于控制输出的地址和存储待排序整数的数组。然后，代码加载这些地址并输出字符串“before sort the array is:\n”，提示用户即将显示排序前的数组。接着，通过一个循环，依次打印数组中的每个元素。

随后，代码进入快速排序的实现部分。它通过递归调用 QSORT 函数对数组进行排序。在 QSORT 函数中，首先保存当前的返回地址和参数，然后判断数组的边界条件。如果左边界小于右边界，则开始排序。在内部，使用双指针法比较

和交换元素，通过调用 SWAP 函数来交换数组中的元素。经过一系列的比较和交换，数组最终被排序。

完成排序后，代码再次加载数据地址，并输出字符串“after sort the array is:\n”，提示用户显示排序后的数组。最后，通过另一个循环打印已排序的数组。

快速排序解释：

• 初始化和参数设置

入口参数: 快速排序的入口为 QSORT 函数，它接受两个参数 $a0 和 $a1，分别表示当前排序的子数组的左边界和右边界。

栈空间分配: 在 QSORT 中，首先通过 daddi $sp,$sp,-24 指令为局部变量分配空间，并保存返回地址和参数，以便在递归调用后恢复。

• 边界条件检查

条件判断: 使用 slt 指令判断左边界 $a0 是否小于右边界 $a1。如果不满足条件（即 l >= h），则直接跳转到 OUT，结束递归。

• 分区过程

初始化指针: 设置两个指针 $s0（左指针）和 $s1（右指针），初始值分别为 $a0 和 $a1。

双指针法: 进入 WHILE 循环，通过比较 $s0 和 $s1 的值，继续进行分区。内部的比较逻辑如下：通过 dmul 指令将当前指针的索引转化为字节地址，然后读取数组元素进行比较。

如果左指针的元素小于右指针的元素，就调用 SWAP 函数交换这两个元素。

指针更新：如果交换发生，左指针加 1，右指针保持不变；否则，左指针保持不变，右指针减 1。

• 递归调用

分区完成后递归: 在分区完成后，左指针 $s0 的值被用作新的分界点。首先递归调用 QSORT 处理左半部分，即 QSORT(l, i - 1)，然后再递归调用处理右半部分，即 QSORT(i + 1, h)。

基于分区点的递归: 这样做确保了每个子数组都在正确的边界内进行排序。

• 退出条件

返回处理: 当所有的递归调用完成后，代码会通过 OUT 标签恢复栈中的返回地址，并返回到上一个调用点。daddi $sp,$sp,24 恢复栈指针，确保不影响其他调用。

快速排序的核心思想是选择一个基准元素（这里隐含在分区过程中），并通过双指针法将数组划分为小于基准和大于基准的两个部分。通过递归地对这两个部分进行排序，快速排序实现了高效的排序，平均时间复杂度为 O(n log n)。

五、实验总结与体会

在本次 MIPS 指令集实验中，我深入学习了 MIPS 架构下的基本操作和排序算法的实现，主要包括“Hello World”输出、冒泡排序和快速排序。这些实践不仅帮助我加深了对 MIPS 指令的理解，也增强了我对计算机系统底层操作的认识。

基础知识的应用：通过实现“Hello World”程序，我体会到在低级编程中如何通过加载和存储操作与寄存器交互。这一过程让我了解了计算机如何处理字符串输出。

冒泡排序的实现：在冒泡排序部分，我不仅复习了排序算法的基本原理，还通过

MIPS 指令实现了其核心逻辑。代码中通过循环和条件判断有效地展示了冒泡排序的过程，并加深了我对循环结构和函数调用的理解。
快速排序的递归特性：快速排序的实现让我领悟到了递归算法的力量。在这一部分中，我学习了如何使用栈进行参数传递与返回地址的保存，掌握了分治法的思想。同时，双指针法的应用让我理解了排序过程中的高效性与优雅性。
调试与测试的技巧：在编写和测试代码的过程中，我提高了调试能力，通过观察输出结果及时修正了潜在的问题。这不仅锻炼了我的逻辑思维能力，也让我意识到细节在编程中的重要性。
MIPS 指令集的学习：通过实践，我对 MIPS 指令集有了更深刻的理解，尤其是在如何使用不同的指令实现复杂功能上。对寄存器、存储器和控制流的操作让我明白了计算机是如何高效执行任务的。
总的来说，这次实验让我在理论与实践之间架起了一座桥梁，增强了我对计算机体系结构的理解和编程能力。未来，我将继续深入学习 MIPS 指令集和其他计算机系统知识，求在底层编程和系统设计方面取得更大进

指导教师批阅意见：

成绩评定：

指导教师签字： 刘刚

年 月 日

备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。