

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: 刘刚

报告人: __学号: __班级: __

实 验 时 间: 2025 年 10 月 1 日

实验报告提交时间: 2025 年 10 月 8 日

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 **E:\ WinMIPS64**

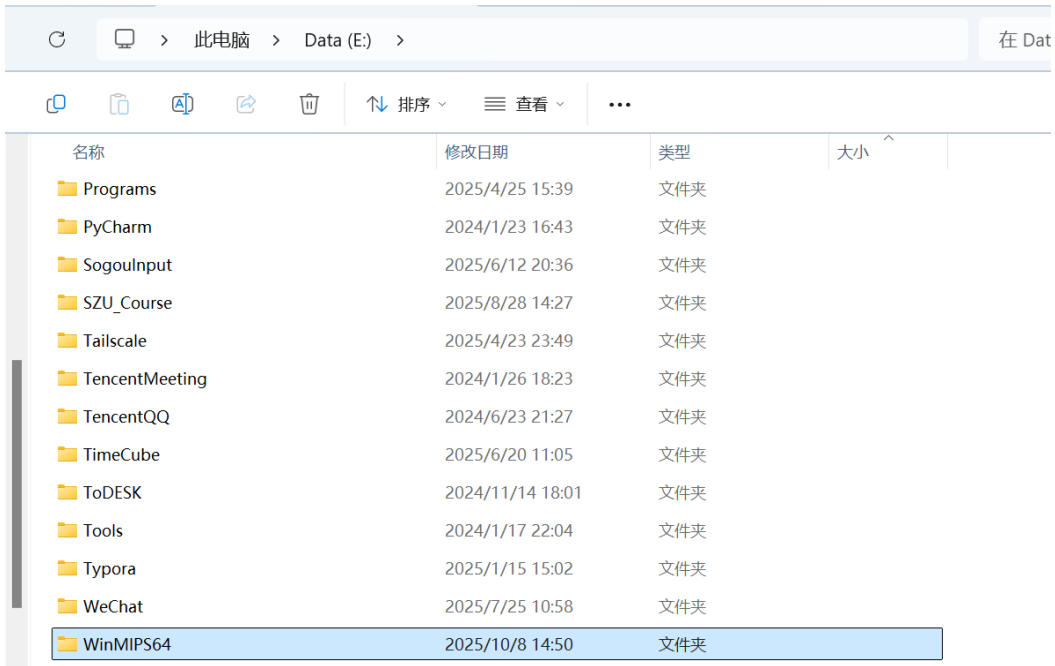


图 1

2. 解压给出的 winmips64.zip 压缩文件到创建的目录中。

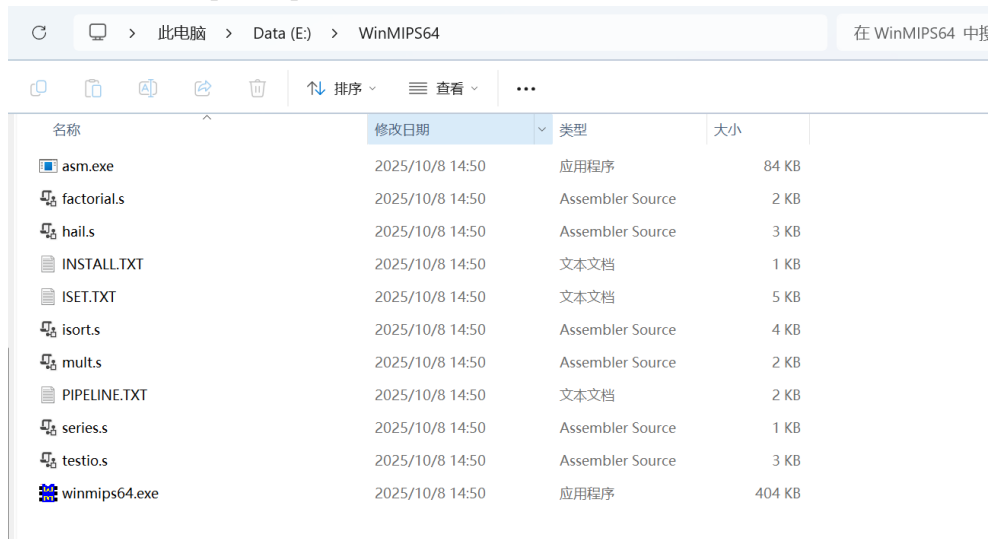
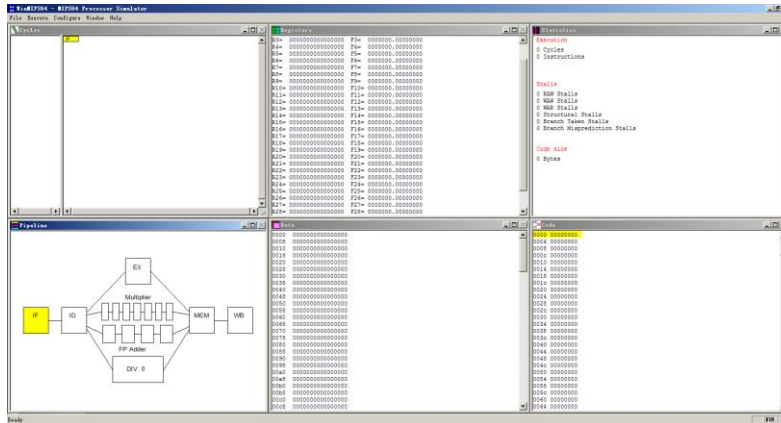


图 2

(二)、一个完整的例子

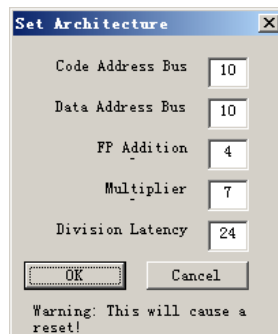
1. 开始和配置 WinMIPS64

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。 **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E:\ 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。

输出如下图所示



```
命令提示符
(c) Microsoft Corporation. 保留所有权利。
C:\Users\OutIn>E:
E:\>cd WinMIPS64
E:\WinMIPS64>asm.exe sum.s.txt
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
E:\WinMIPS64>
```

图 3

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。

如下图所示

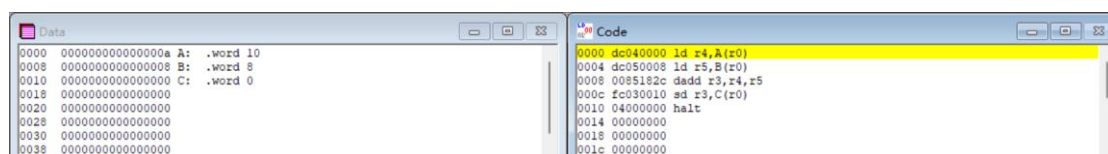


图4

按F7或者点击左上角Execute->Single Cycle单步运行，观察Cycle、Pipeline、Code、Statistics都有变化

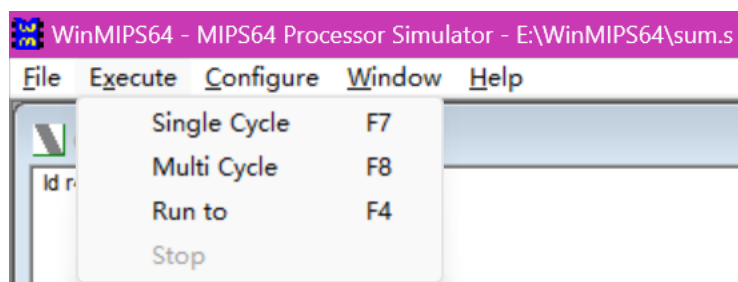


图5

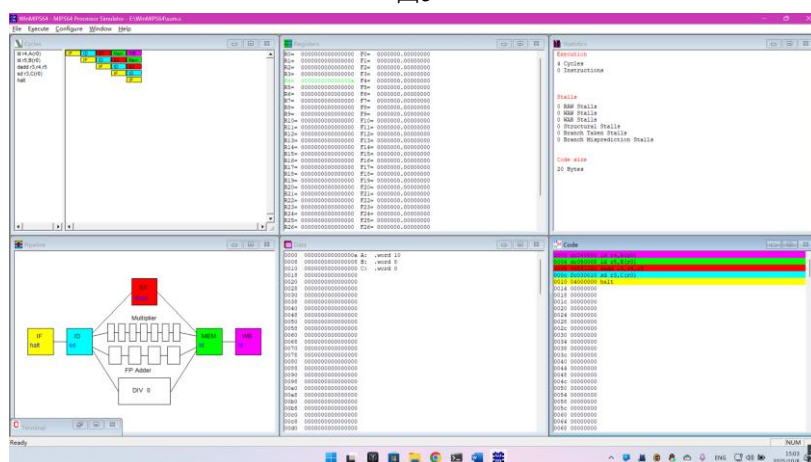


图6

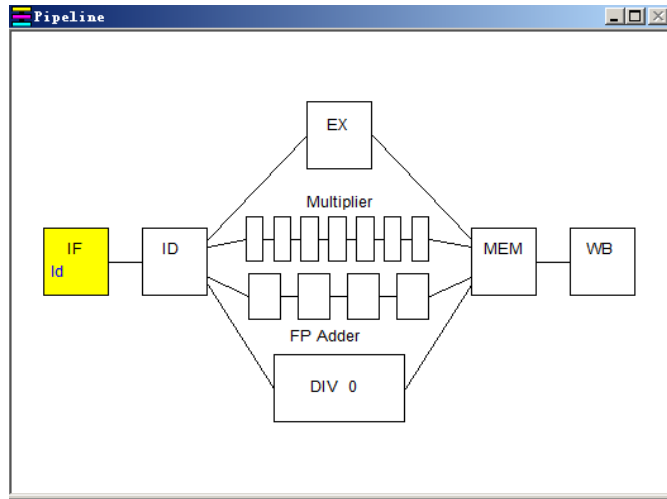
- (1) Pipeline 窗口显示了 MIPS64 处理器的 5 个流水线级的示意图
- (2) Code 窗口展示每条指令分别处于流水线的哪个阶段，黄色表示“取指”，蓝色表示“译码”，红色表示“执行”，绿色表示“内存数据读或写”，紫色表示“写回”
- (3) Cycles 窗口此窗口表示 pipeline 的 时空 行为。它记录指令进出管道时的历史记录

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？

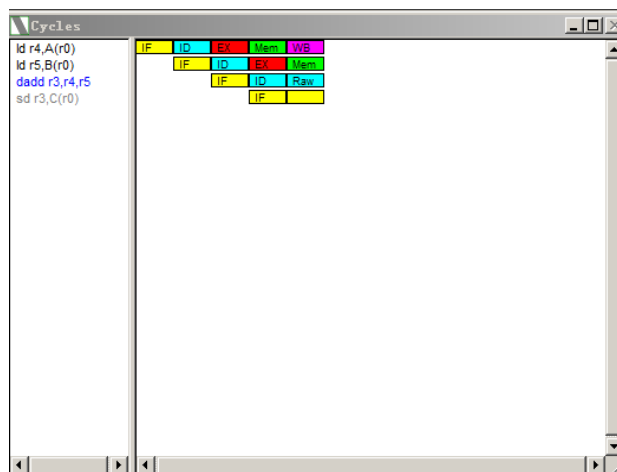
```

Code
0000 dc040000 ld r4,A(r0)
0004 dc050008 ld r5,B(r0)
0008 0085182c dadd r3,r4,r5
000c fc030010 sd r3,C(r0)
0010 04000000 halt
0014 00000000
0018 00000000
001c 00000000
0020 00000000
0024 00000000
0028 00000000
002c 00000000
0030 00000000
0034 00000000
0038 00000000
003c 00000000
0040 00000000
0044 00000000
0048 00000000
004c 00000000
0050 00000000
0054 00000000
0058 00000000
005c 00000000
0060 00000000
0064 00000000

```

(3) *Cycles* 窗口

我们将注意力放到*Cycles*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，*Cycles*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。

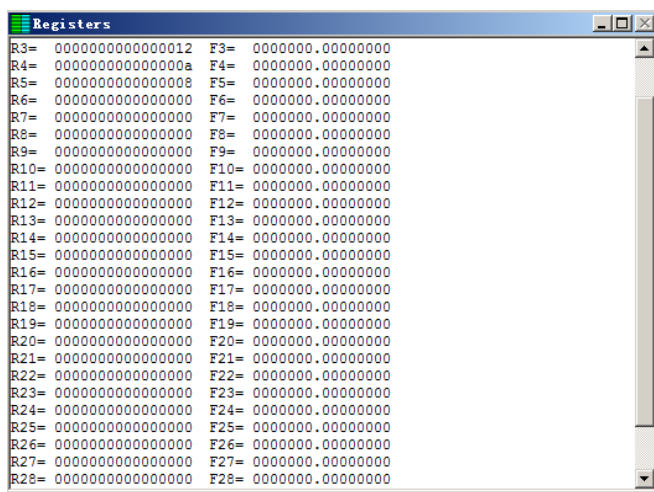
Address	Data
0000	0000000000000000 A: .word 10
0008	0000000000000000 B: .word 8
0010	0000000000000012 C: .word 0
0018	0000000000000000
0020	0000000000000000
0028	0000000000000000
0030	0000000000000000
0038	0000000000000000
0040	0000000000000000
0048	0000000000000000
0050	0000000000000000
0058	0000000000000000
0060	0000000000000000
0068	0000000000000000
0070	0000000000000000
0078	0000000000000000
0080	0000000000000000
0088	0000000000000000
0090	0000000000000000
0098	0000000000000000
00a0	0000000000000000
00a8	0000000000000000
00b0	0000000000000000
00b8	0000000000000000
00c0	0000000000000000
00c8	0000000000000000

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改变寄存器的值，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。

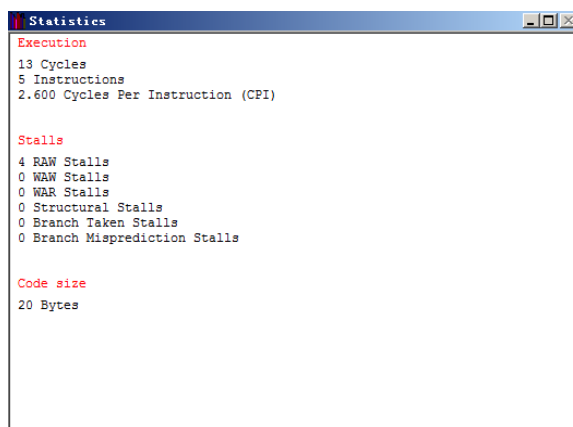


上图即为第十三个时钟周期的`Registers`窗口的图示，很显然，其中可以很清楚的看出每个寄存器的值是什么。

(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：

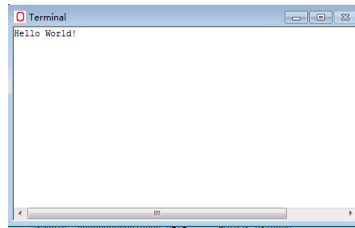
```
CONTROL: .word32 0x10000
DATA:    .word32 0x10008

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!”。

编写程序如下：

```
.data
message: .asciiz "Hello World!"
CONTROL: .word32 0x10000
DATA:    .word32 0x10008

.text
main:
lwu r12,DATA(r0)    ; M[DATA] -> R12
daddi r21,r0,message ; MESSAGE -> M[R21]
sd r21,0(r12)        ; MESSAGE -> M[R12]

lwu r13,CONTROL(r0) ; M[CONTROL]->R13
daddi r20,r0,4       ; 4->M[R20]
sd r20,0(r13)        ; SET CONTROL = 4

halt
```

用 asm.exe 检验程序，没有错误

```
C:\Users\OutIn>E:
E:\>cd WinMIPS64
E:\WinMIPS64>asm.exe hello.s.txt
Pass 1 completed with 0 errors
00000000 .data
00000000 message: .asciiz "Hello World!"
         48656c6c
         6f20576f
         726c6421
         00
00000010 00010000 CONTROL: .word32 0x10000
00000018 00010008 DATA:    .word32 0x10008

00000000 .text
00000000 main:
00000000 9c0c0018 lwu r12,DATA(r0)    ; M[DATA] -> R12
00000004 60150000 daddi r21,r0,message ; MESSAGE -> M[R21]
00000008 fd950000 sd r21,0(r12)        ; MESSAGE -> M[R12]

0000000c 9c0d0010 lwu r13,CONTROL(r0) ; M[CONTROL]->R13
00000010 60140004 daddi r20,r0,4       ; 4->M[R20]
00000014 fdb40000 sd r20,0(r13)        ; SET CONTROL = 4

00000018 04000000 halt

Pass 2 completed with 0 errors
Code Symbol Table
main = 00000000
```

图 7

打开 hello.s, 初始情况如下

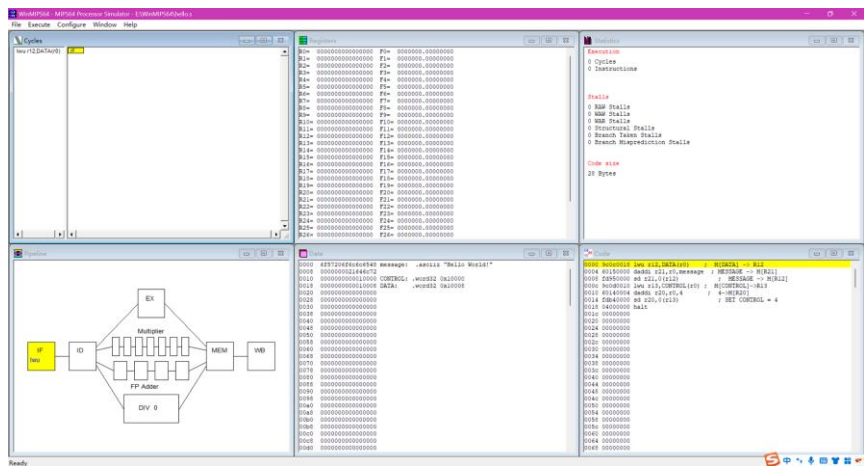


图 8

按 F4 运行

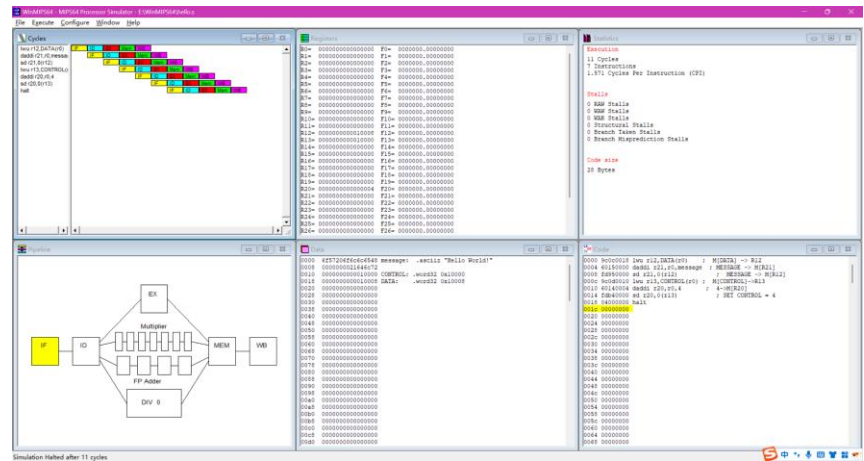


图 9

终端成功打印 Hello World

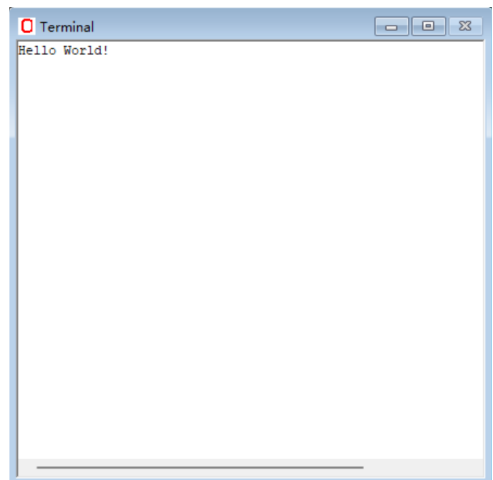


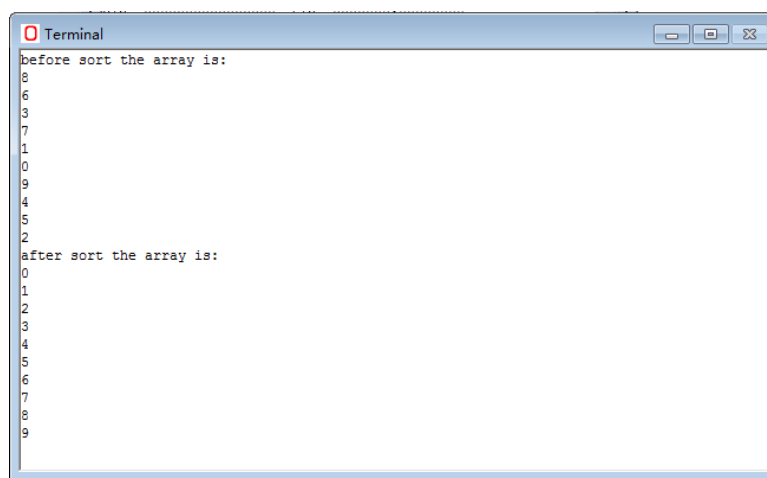
图 10

(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）：



```
Terminal
before sort the array is:
8
6
3
7
1
0
9
4
5
2
after sort the array is:
0
1
2
3
4
5
6
7
8
9
```

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。

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 `FFFFFFF`，该地址将超出 `winmips` 默认的内存空间。

C++冒泡排序函数代码如下：

```
swap( a, b) {
    tmp = a;
    a = b;
    b = tmp;
}

sort() {
    for (i = 0; i < 10 i++) {
        for (j = i - 1; j >= 0; j--)
            if (a[j] > a[j + 1]) {
                swap(a[j], a[j + 1]);
            }
    }
}
```

```

    }
}
}

```

编写程序代码如下：

```

.data
arr: .word 1, 3, 4, 2, 5, 7, 0, 8,6,9
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

.text
main:
    #cout  "before sort the array is"
    ld r13, CONTROL(r0)
    ld r14, DATA(r0)
    daddi r9, r0, 4
    daddi r8, r0, before
    sd r8, (r14)
    sd r9, (r13)

    daddi r9, r0, 2
    daddi r2, r0, 10
    daddi r1, r0, 0
firstCout:
    dsll r3, r1, 3
    ld r8, arr(r3)
    sd r8, (r14)
    sd r9, (r13)

    daddi r1, r1, 1
    bne r1, r2, firstCout
    # sort()
    ld r29, SP(r0)
    daddi r4, r0, arr
    daddi r5, r0, 10
    jal sort

    #cout  "after sort the array is"
    daddi r9, r0, 4

```

```

    daddi r8, r0, after
    sd r8, (r14)
    sd r9, (r13)

    # print the array
    daddi r9, r0, 2
    daddi r2, r0, 10
    daddi r1, r0, 0
secCout:
    dsll r3, r1, 3
    ld r8, arr(r3)
    sd r8, (r14)
    sd r9, (r13)

    daddi r1, r1, 1 # , i++
    bne r1, r2, secCout
    halt

sort:
    daddi r29, r29, -24
    sd $ra, 16(r29)
    sd r13, 8(r29)
    sd r14, 0(r29)

    dadd r22, r4, r0
    daddi r23, r5, 0

    and r18, r18, r0
loop1:
    # for (i = 0; i < n; i++)
    slt r10, r18, r23 # if i < n
    beq r10, r0, end1 # if i >= n
    daddi r19, r18, -1

loop2:
    # for (j = i - 1; j >= 0; j--)
    slti r10, r19, 0 # j < 0
    bne r10, r0, end2

    dsll r11, r19, 3
    dadd r12, r22, r11
    ld r16, 0(r12)
    ld r17, 8(r12)

```

```

        # if (a[j] > a[j + 1])
        slt r10, r17, r16
        beq r10, r0, end2
        dadd r4, r0, r12
daddi r5, r12, 8
# swap(a[j], a[j + 1])
        jal swap

        daddi r19, r19, -1 # j--
        j loop2

end2:
        daddi r18, r18, 1 # i--
        j loop1

endl:
        # restore
        ld r14, 0(r29)
        ld r13, 8(r29)
        ld $ra, 16(r29)
        daddi r29, r29, 24
        jr $ra

swap:
        # swap( a, b)
        ld r8, 0(r4)
        ld r10, 0(r5)
        sd r10, 0(r4)
        sd r8, 0(r5)
        jr $ra

```

用 asm.exe 检验程序，没有错误

```

E:\WinMIP564>asm.exe sort.s.txt
Pass 1 completed with 0 errors
00000000 .data
00000000 0000000000000001 arr: .word 1, 3, 4, 2, 5, 7, 0, 8, 6, 9
00000004 0000000000000002
00000008 0000000000000005
0000000c 0000000000000007
00000010 0000000000000008
00000014 0000000000000006
00000018 0000000000000009
00000050 before: .asciiz "before sort the array is:\n"
6265666f
72652073
6f727428
74686520
61727261
79206973
3a8a00
00000070 after: .asciiz "after sort the array is:\n"
61667465
7220736f
72742074
68652061
72726179
2069733a
0a00
00000090 0000000000010000 CONTROL: .word 0x10000

```

图 11

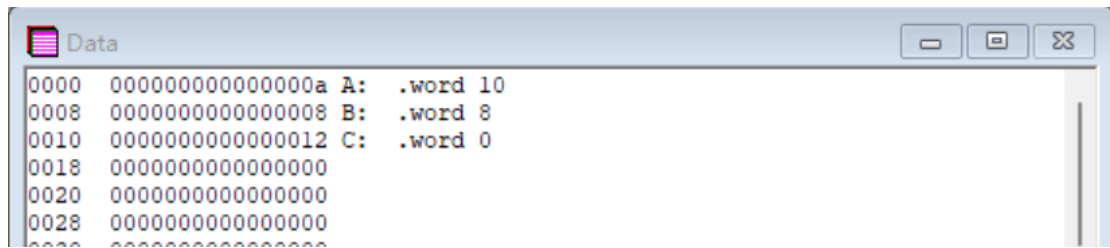


图 14

②hello.s

直接点击 Run to，程序运行完毕，终端打印 Hello World

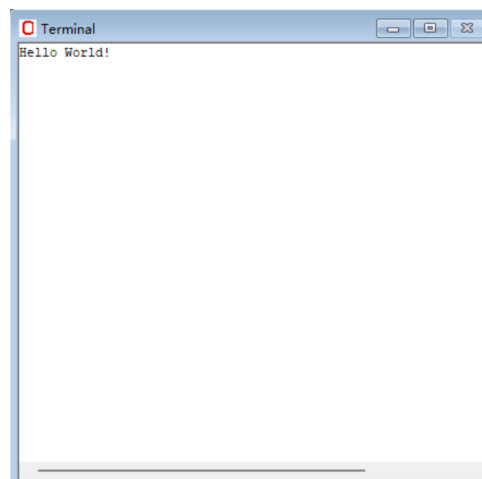


图 15

③sort.s

直接点击 Run to，程序运行完毕，成功排序

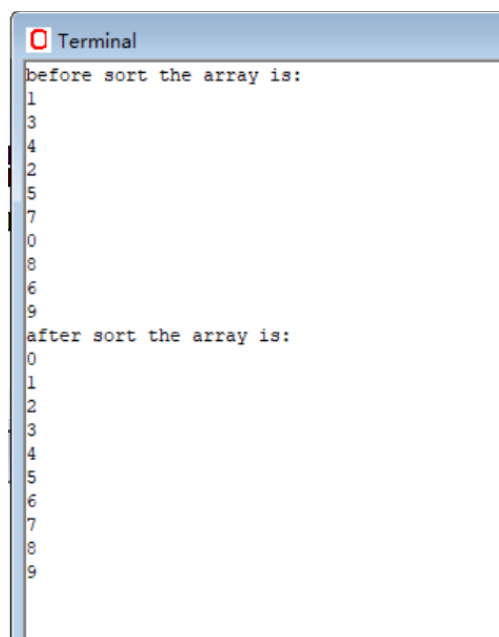


图 16

五、实验总结与体会

在完成了 MIPS 指令集实验后，我对计算机系统有了更深入的了解。通过使用 WinMIPS64 仿真器，我学会了如何操作 MIPS 指令集，建立了对指令流水执行的基本认识。在实验过程中，我熟悉了 WinMIPS64 的基本功能和操作方式，掌握了 IO 方法，并成功编写了排序程序。这些实践让我更加熟悉计算机系统的工作原理，为以后的流水线实验打下了基础。

指导教师批阅意见：	
成绩评定：	
	指导教师签字： 年 月 日

指导教师批阅意见：	
成绩评定：	
	指导教师签字： 年 月 日

指导教师批阅意见：	
成绩评定：	
	指导教师签字： 年 月 日

指导教师批阅意见：	
成绩评定：	
	指导教师签字： 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。

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。