

实验课程名称	数字逻辑与计算机组成实验	成绩	
实验项目名称	RISC-V 汇编语言设计实验	指导老师	

一、实验目的

- 1、掌握 RISC-V 基础指令集 RV32I 的指令格式和使用方法；
- 2、学会 RISC-V 汇编模拟器 RARS 的使用；
- 3、掌握用 RISC-V 汇编语言编写、调试和运行程序的方法。

二、实验要求

- 1、适当添加程序注释，便于理解；
- 2、做好程序测试，验证程序功能是否正确并做好测试记录；
- 3、根据本次实验内容的要求，将源程序（文本格式，不得为图片）、运行结果、测试记录、分析和思考等内容写入实验报告。
- 4、上交实验报告和打包的源程序压缩文件。其中，实验报告为 word 文档，文件名为学号，如 200640001.docx；5 个题目的 .asm 源程序压缩打包为 1 个文件，文件名为学号，如 200640001.zip。

三、实验内容

（一）安装并熟悉 RISC-V 汇编模拟器 RARS 的使用；

（二）完成以下程序的编写

1、顺序结构的编程

计算 $y=10a+6b-c$ ，其中 y 放在寄存器 $a0$ 中， a 、 b 、 c 三个变量存放的寄存器不限。要求不用乘法指令。

2、分支结构的编程

计算 C 语言表达式： $\text{if } (x>y) \text{ } z=x+5; \text{ else } z=y-5$ 。其中 z 存放在寄存器 $a1$ 中， x 、 y 存放的寄存器不限。

3、循环结构的编程

计算 $y=1+2+3+\dots+100$ ， y 存放在寄存器 $a2$ 中。

4、系统调用

输出提示信息“请输入姓名：”，从键盘输入本人姓名的字符串；输出提示信息“请输入学号：”，输入本人学号后 3 位（整数形式）。程序结束后调用 `exit` 功能退出。

提示：系统调用功能的使用方式：将功能号放入 a7 寄存器，参数放入 a0~a3 等系统要求的寄存器，运行 ecall 指令。常用的系统调用功能有：

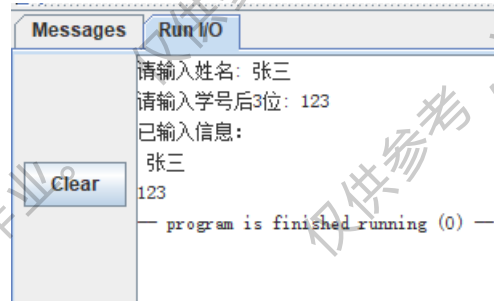
功能号	功能描述	输入值	输出值
1	输出 1 个整数	a0 = 要输出的整数	
4	输出字符串	a0 = 要输出的字符串首地址	
5	输入 1 个整数		a0 = 输入的整数
8	输入字符串	a0 = 输入字符串放置的地址 a1 = 最大的输入字符个数	
10	退出程序 exit		
11	输出 ascii 字符	a0 = 要输出的字符（只输出最低字节）	

更多的系统调用功能详见 RARS 系统 Help 中的 Syscalls 部分。

用法示例：

```
la a0,string    #假设字符串首地址标号为 string，放入 a0
li a7,4         #a7 中存放功能号 4
ecall          #开始系统调用，即可输出字符串到 Run/IO 区。
```

本题运行后，结果显示如下：



5、综合编程

利用数据结构课所学的任意一种排序方式，将数据区的 10 个数字按从小到大排序，并输出显示。要求程序的第一行注释里写明是采用什么排序方式，如冒泡排序。数据区为：

```
.data
array: .word -15,1024,12,60,19,26,-18,19,100,86
```

四、实验步骤及结果

1. 计算 $y=10a+6b-c$, 其中 y 放在寄存器 $a0$ 中, a 、 b 、 c 三个变量存放的寄存器不限。要求不用乘法指令。

- 源代码

第一题 顺序结构的编程

y 在 $a0$; a 在 $a1$, b 在 $a2$, c 在 $a3$

#####数据段#####

.data

StartMsg: .string "\n 计算 $y=10a+6b-c$ "

InputaMsg: .string "\n 请输入 a: "

InputbMsg: .string "请输入 b: "

InputcMsg: .string "请输入 c: "

ResultMsg: .string "\n 计算结果为: "

#####代码段#####

.text

#####输入 a, b, c#####

-----输出 StartMsg-----

la a0,StartMsg # 将字符串 StartMsg 首址放入 a0

li a7,4 # 功能号放入 a7, 4 号表示字符串的输出

ecall # 系统调用

-----输出 InputaMsg-----

la a0,InputaMsg # 将字符串 InputaMsg 首址放入 a0

li a7,4 # 功能号放入 a7, 4 号表示字符串的输出

ecall # 系统调用

-----输入 a-----

li a7,5 # 功能号放入 a7, 5 号表示输入一个整数到 a0

ecall # 系统调用

mv a1,a0 # 将 a0 数据放入 a1; 即 $a1=a0$

-----输出 InputbMsg-----

la a0,InputbMsg # 将字符串 InputbMsg 首址放入 a0

li a7,4 # 功能号放入 a7, 4 号表示字符串的输出

ecall # 系统调用

-----输入 b-----

li a7,5 # 功能号放入 a7, 5 号表示输入一个整数到 a0

```

ecall          # 系统调用
mv a2,a0       # 将 a0 数据放入 a2; 即 a2=a0
# -----输出 InputbMsg-----
la a0,InputbMsg # 将字符串 InputbMsg 首址放入 a0
li a7,4        # 功能号放入 a7, 4 号表示字符串的输出
ecall          # 系统调用

# -----输入 c-----
li a7,5        # 功能号放入 a7, 5 号表示输入一个整数到 a0
ecall          # 系统调用
mv a3,a0       # 将 a0 数据放入 a3; 即 a3=a0

# -----输出 ResultMsg-----
la a0,ResultMsg # 将字符串 ResultMsg 首址放入 a0
li a7,4        # 功能号放入 a7, 4 号表示字符串的输出
ecall          # 系统调用

#####开始计算#####

# -----计算 10a, 10a=8a+2a(因为不可以用乘法, 采用移位)-----
slli t1,a1,3   # 计算 8a, 并放入 t1
slli t2,a1,1   # 计算 2a, 并放入 t2
add a1,t1,t2    # 计算 8a+2a, 并放回 a1

# -----计算 6b, 6b=4b+2b(原因同上)-----
slli t1,a2,2   # 计算 4b, 并放入 t1
slli t2,a2,1   # 计算 2b, 并放入 t2
add a2,t1,t2    # 计算 4b+2b, 并放回 a2

# -----计算 10a+6b-----
add t1,a1,a2    # 计算 a1+a2, 并放入 t1

# -----计算 10a+6b-c-----
sub a0,t1,a3     # 计算最终结果, 10a+6b-c 结果 y 放入 a0 中

# -----输出计算结果 y-----
li a7,1        # 功能号放入 a7, 1 号表示整数的输出
ecall          # 系统调用命令

# -----退出程序-----
addi a7,zero, 10 # 系统调用,功能号为 10。功能: 结束退出, 即 exit
ecall          # 系统调用命令

```

- 运行截图

```

计算y=10a+6b-c
请输入a: 10
请输入b: 20
请输入c: 100

计算结果为: 120
-- program is finished running (0) --

```

2. 计算 C 语言表达式: if (x>y) z=x+5; else z=y-5。其中 z 存放在寄存器 a1 中, x、y 存放的寄存器不限。

- 源代码

第二题 分支结构的编程

其中 z 存放在寄存器 a1 中, x 在 a2、y 在 a3。

#####数据段#####

.data

变量定义

x: .word 0

y: .word 0

z: .word 0

输出信息

startMsg: .string "计算C语言表达式 if (x>y) z=x+5; else z=y-5"

inputxMsg: .string "\n 请输入 x: "

inputyMsg: .string "请输入 y: "

resultMsg: .string "\nz 为: "

#####代码段#####

.text

#####x,y,z 赋值给寄存器 a1,2,3#####

main:

lw a1,z # 将 z 的值赋给 a1 寄存器

lw a2,x # 将 x 的值赋给 a2 寄存器

lw a3,y # 将 y 的值赋给 a3 寄存器

#####输入 x,y#####

input:

-----输出 startMsg-----

```

    la a0,startMsg      # 将字符串 startMsg 首址放入 a0
    li a7,4             # 功能号放入 a7, 4 号表示字符串的输出
    ecall              # 系统调用

# -----输出 inputxMsg-----
    la a0,inputxMsg     # 将字符串 InputaMsg 首址放入 a0
    li a7,4             # 功能号放入 a7, 4 号表示字符串的输出
    ecall              # 系统调用

# -----输入 x-----
    li a7,5             # 功能号放入 a7, 5 号表示输入一个整数到 a0
    ecall              # 系统调用
    sw a0,x,t1          # 借助 t1 储存 x 变量的地址, 然后将 a0 赋值给 x

# -----输出 inputyMsg-----
    la a0,inputyMsg     # 将字符串 InputbMsg 首址放入 a0
    li a7,4             # 功能号放入 a7, 4 号表示字符串的输出
    ecall              # 系统调用

# -----输入 y-----
    li a7,5             # 功能号放入 a7, 5 号表示输入一个整数到 a0
    ecall              # 系统调用
    sw a0,y,t1          # 借助 t1 储存 x 变量的地址, 然后将 a0 赋值给 y(伪指令)

# -----输出 resultMsg-----
    la a0,resultMsg     # 将字符串 resultMsg 首址放入 a0
    li a7,4             # 功能号放入 a7, 4 号表示字符串的输出
    ecall              # 系统调用

#####计算 if(x>y)表达式#####
calculate:
    # -----将变量 x,y 的值读取到寄存器 a2, a3 中-----
    lw a2,x             # 将 x 值放入寄存器 a2
    lw a3,y             # 将 y 值放入寄存器 a3

    bgt a2,a3,Yes       # a2>a3?, 即 x>y?
    j No                # 上述条件不成立, 跳转执行 No:

Yes:    # x>y 时执行
    addi a1,a2,5         # z=x+5
    sw a1,z,t1           # load adress t1,z      store word a0,(t1)(伪指令)
    j exit              # 表达式计算完毕, 跳转到 exit:

No:    # x<y 时执行
    addi a1,a3,-5        # z=y-5

```



```

sw a1,z,t1          # load address t1,z      store word a0,(t1)(伪指令)

#####输出结果#####
exit:
lw a0,z             # 将 z 的值放入寄存器 a0
li a7,1             # 功能号放入 a7, 1 号表示整数的输出
ecall               # 系统调用
li a7,10            # 系统调用,功能号为 10。功能: 结束退出,即 exit
ecall               # 系统调用命令

```

● 运行截图

```

计算c语言表达式if (x>y) z=x+5; else z=y-5
请输入x: 1
请输入y: 5

z为: 0
-- program is finished running (0) --

计算c语言表达式if (x>y) z=x+5; else z=y-5
请输入x: 9
请输入y: 5

z为: 14
-- program is finished running (0) --

```

3. 计算 $y=1+2+3+\dots+100$, y 存放在寄存器 $a2$ 中。

● 源代码

```

# 第三题 循环结构的编程

## 其中 a1 为每次循环自增 1, y 存放在 a2, iMax 存放在 a3
#####数据段#####
.data
# 数据定义
y:      .word 0      # 结果 y
iMax:   .word 100    # i 的最大值
# 输出信息
StartMsg: .string "计算 y=1+2+3...+100"
RusultMsg: .string "\n 结果为: y="

#####代码段#####

```

```

.text
# 变量分别加载到寄存器, i->a1;y->a2;iMax->a3
main:
    add a1,a1,zero
    lw a2,y
    lw a3,iMax

# 输出一些提示信息
Output:
    la a0,StartMsg      # 将字符串 StartMsg 首址放入 a0
    li a7,4              # 功能号放入 a7, 4 号表示字符串的输出
    ecall               # 系统调用
    la a0,RusultMsg     # 将字符串 StartMsg 首址放入 a0
    li a7,4              # 功能号放入 a7, 4 号表示字符串的输出
    ecall               # 系统调用

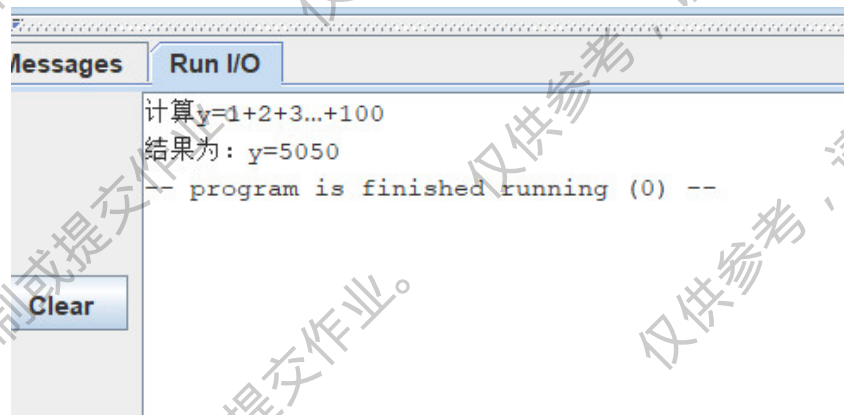
# 进行判断, 当 i<iMax(100)时进行跳转到循环; 当不满足时跳转到 exit 输出结果退出程序
judge:
    ble a1,a3,loop      # i<=100 Jump loop:
    j exit              # if(i>100) Jump exit:

# 循环
loop:
    lw a2,y              # 将 y 的值加载到寄存器 a2 中
    add t1,a2,a1         # t1=y+i
    sw t1,y,t2           # y=t1
    addi a1,a1,1         # i++
    j judge             # 跳转到判断 judge:

# 输出结果, 程序退出
exit:
    lw a0,y              # 将 y 的值读取到寄存器 a0 中 (为了系统调用输出)
    li a7,1              # 在 a7 寄存器设置功能号 1(整数输出)
    ecall               # 系统调用
    li a7,10             # 在 a7 寄存器设置功能号 10(exit(0))
    ecall               # 系统调用

```


- 运行截图



4. 输出提示信息“请输入姓名:”，从键盘输入本人姓名的字符串；输出提示信息“请输入学号:”，输入本人学号后 3 位（整数形式）。程序结束后调用 exit 功能退出。

- 源代码

第四题 系统调用

#####数据段#####

.data

数据定义

num: .word 000 # 学号

strLen: .word 10 # 姓名字符串的最大长度

输出信息

NameMsg: .string "请输入姓名: "

NumMsg: .string "请输入学号: "

#####代码段#####

.text

输出 NameMsg

la a0,NameMsg

li a7,4

ecall

把 strLen 加载到 a1，调用系统调用功能 8，输入字符串

lw a1,strLen

li a7,8

ecall

把 a0 内容放入 t1 寄存器

add t1,a0,zero # t1 = a0 + 0

输出 NumMsg

la a0,NumMsg

li a7,4

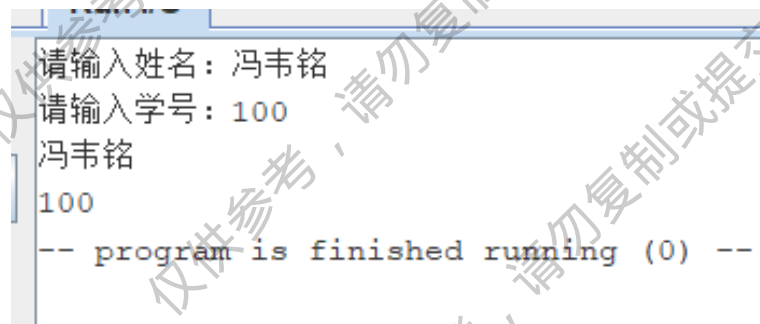
ecall

```

# 输入整数
li a7,5
ecall
sw a0,num,t2          # num = a0
# 输出输入的姓名字符串
add a0,t1,zero
li a7,4
ecall
# 输出 ascii 字符\n
addi a0,zero,10      # ascii 10 = '\n'
li a7,11
ecall
# 输出学号整数
lw a0,num
li a7,1
ecall
# exit 退出
addi a7, zero, 10     # 系统调用,功能号为 10。功能: 结束退出,即 exit
ecall                 # 系统调用命令

```

● 运行截图



5. 利用数据结构课所学的任意一种排序方式，将数据区的 10 个数字按从小到大排序，并输出显示。要求程序的第一行注释里写明是采用什么排序方式，如冒泡排序。

● 源代码

```

# 冒泡排序
#####数据段#####
.data
# 数据定义
array:      .word   -15,1024,12,60,19,26,-18,19,100,86 # 需要进行排序的数组
arrayLen:   .word   10 # 数组长度
# 输出信息
startMsg:   .string "从小到大冒泡排序: -15,1024,12,60,19,26,-18,19,100,86"
endMsg:     .string "\n 排序好后的数组为: "

```

```

#####代码段#####
# s10    数组入口地址
# s11    数组结束地址

# t0     工具人
# t1     外部循环下标最大值
# t2     内部循环下标最大值

# a1     交换数 1 地址
# a2     交换数 2 地址
# a3     交换数 1
# a4     交换数 2
# a5     打印数组元素地址

# s1     外部循环下标
# s2     内部循环下标
.text
main:
# 输出提示信息 startMsg
la a0,startMsg
li a7,4
ecall
# 输出提示信息 endMsg
la a0,endMsg
li a7,4
ecall
# 初始化寄存器, 把数组放进去
la s10, array          # s10    数组入口地址
lw t0, arrayLen        # t0     temp
addi t0, t0, -1
slli t0, t0, 2          # t0     temp*4; 因为一个.word 4bit, 所以左移 2 位
add s11, s10, t0        # s11    数组结束地址; s11=s10+arryLen*4;
# 初始化内外循环下标
addi s1, zero, 0        # s1     外部循环下标
addi s2, zero, 0        # s2     内部循环下标
lw t0, arrayLen
addi t1, t0, -2         # t1     外层循环下标的最大值 = arrayLen + -2
# 进入外层循环
j extloop               # jump to extloop

# 外层循环下标++
addExtPos:
addi s1, s1, 1          # s1 = s1 + 1
j extloop               # jump to extloop

```

```

# 内层循环下标++
addInPos:
    addi s2, s2, 1          # s2 = s2 + 1
    j inloop                # jump to inloop

# 冒泡（交换）两数
swap:
    # a1    交换数 1 地址
    # a2    交换数 2 地址
    # a3    交换数 1
    # a4    交换数 2
    sw a4, (a1)             # a4 = *a1; 交换数 2 放入数 1 地址
    sw a3, (a2)             # a3 = *a2; 交换数 1 放入数 2 地址
    j addInPos              # jump to addInPos

# 外层循环
extloop:
    bge s1, t1, initprint   # if s1 >= t1 then initprint
    j initIn                # jump to initIn

# 初始化内层循环
initIn:
    # s1    外部循环下标
    # s2    内部循环下标
    sub t2, t1, s1          # t2 = t1 - s1; 内层循环的最大值
    addi s2, zero, 0
    j inloop                # jump to inloop

# 内层循环
inloop:
    # a1    交换数 1 地址（内层循环下标所指）
    # a2    交换数 2 地址（内层循环下标+1所指）
    # a3    交换数 1
    # a4    交换数 2
    # t0    工具人（temp）
    # s2    内部循环下标
    # s10   数组入口地址

    bgt s2, t2, addExtPos   # if s2 > t2 then addExtPos; 内层循环执行完毕
    # a1 = 数组首地址 + 4*内循环下标
    slli t0, s2, 2          # t0 = 4*内循环下标 s2
    add a1, s10, t0         # a1 = s10 + t0
    lw a3, (a1)             # a3 = *a1

```

```

# a2 = a1 + 4
addi a2, a1, 4
lw a4, (a2)

# 判断是否需要冒泡
bgt a3, a4, swap      # if a1 > a2 then swap
# 不需要冒泡直接内层循环下标++
j addInPos            # jump to addInPos

# 初始化数组打印
initprint:
# s10 数组入口地址
mv a5, s10            # a5 = s10; (a5 此时为数组首地址)
j printloop           # jump to printloop

# 数组打印循环
printloop:
# a5 打印数组元素地址
# s11 数组结束地址
lw a0, (a5)           # a0 = *a5
li a7, 1              # 系统调用功能 1, 输出整数
ecall                # 系统调用

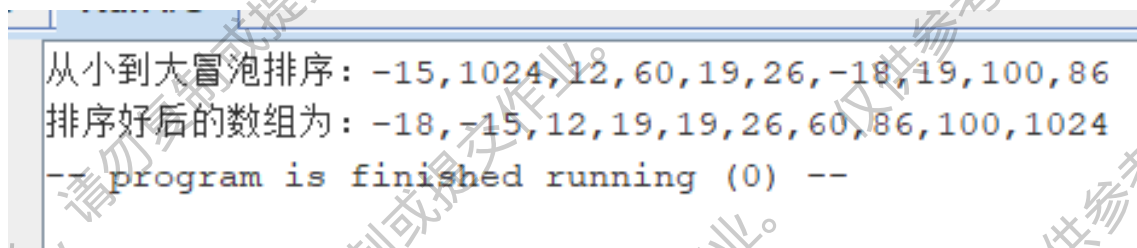
sub t0, s11, a5        # t0 = s11 - a5, 地址相减 = 剩余元素个数
beqz t0, exit          # 剩余元素个数 t0 = 0, 打印完了, 下班
addi a5, a5, 4         # 没打印完, 地址后移 4bit 打印下一个元素~

addi a0, zero, 44      # ascii 44 = ',', 打印个, 分隔
li a7, 11              # 系统调用功能 11, 输出 ascii 字符
ecall                # 系统调用
j printloop           # jump to printloop

# 下班~ (退出程序)
exit:
li a7, 10              # ...终于下班了
ecall

```

● 运行截图



```

从小到大冒泡排序: -15,1024,12,60,19,26,-18,19,100,86
排序好后的数组为: -18,-15,12,19,19,26,60,86,100,1024
-- program is finished running (0) --

```

五、分析与思考

● 实验中遇到的问题分析

1) lw 指令使用错误。

分析：lw 指令为 load word，即加载一个 word 数据到指定寄存器中
在下图中 a0 储存的是一个 word 数据，但是使用 lw 指令时需要另一个寄存器里存放的是地址，故错误

解决：将 lw 改为 mv

```
39  ecall
40  # -----输入c-----
41  li a7,5
42  ecall
43  lw a3,a0
44  # -----输出ResultMsg-----
```

2) 跳转逻辑错误

分析：下图中 $a2 > a3$ 时，跳转到 Yes:，但是 Yes: 段语句执行完毕后又直接顺序执行 No: 语句，违背逻辑

解决：将条件改为判断 $a2 < a3$ 跳转 No:

或者在 Yes: 语句里加上跳转 j exit

```
1  #####计算if表达式#####
2  bgt a2,a3,Yes # a2>a3?,即x>y?
3  Yes:          # z=x+5
4  add a1,a2,5
5  No:           # z=y-5
6  sub a1,a2,-5
```

● 思考

1) lw 指令的用法

lw a0,table # 将变量 table 值放入 a0 寄存器

lw a0,(t1) # 此时 t1 应该储存了地址（不是值），指令将地址所指的
 值取出来存放到 a0。相当于：a0=*t1

2) sw 指令的用法

sw a0,(t1) # *t1=a0

sw a0,table,t0 #伪指令 相当于 la t0,table sw a0,(t0)，用于更新变量值

3) 拷贝寄存器的值方法


```
add a0,a1,zero
```

```
addi a0,a1,0
```

```
mv a0,a1
```

4) 汇编编写循环 tips

- 所有循环一律用死循环的格式处理，在循环体中加入分支语句即可。
- 条件表达式取反处理

5) 寄存器与内存

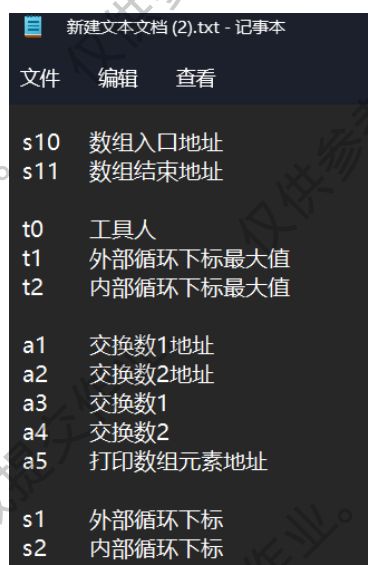
寄存器是 CPU 里的东西，内存是在 CPU 外面的数据总线上。

寄存器是中央处理器内的组成部份。寄存器是有限存贮容量的高速存贮部件，它们可用来暂存指令、数据和位址。内存是计算机中重要的部件之一，它是与 CPU 进行沟通的桥梁。计算机中所有程序的运行都是在内存中进行的。

访问内存时要在 CPU 的寄存器填上地址，再执行相应的汇编指令。这时 CPU 会在数据总线上生成读取或者写入内存数据的时钟信号，最后内存的内容会被 CPU 寄存器的内容更新(写入)或被读入 CPU 的寄存器(读取)。

● 体会

汇编是非常底层的语言，使用汇编编写程序需要有非常清晰的思维以及良好的注释习惯（没有注释我自己看自己刚刚写的代码都看懂了），并且需要列出使用的寄存器（用来干什么的，每个功能有个专用寄存器）。在编写第五题综合编程时，因为冒泡排序程序需要双层嵌套循环，并且因为不懂 jal 语句的 ra 如何使用（最后也没用上），花费了非常多的时间调试。由衷感叹我们在前人的大树下乘凉的快乐。



文件	编辑	查看
s10	数组入口地址	
s11	数组结束地址	
t0	工具人	
t1	外部循环下标最大值	
t2	内部循环下标最大值	
a1	交换数1地址	
a2	交换数2地址	
a3	交换数1	
a4	交换数2	
a5	打印数组元素地址	
s1	外部循环下标	
s2	内部循环下标	