

# 中山大学数据科学与计算机学院本科生实验报告

## (2017 学年春季学期)

课程名称: 计算机组成原理实验

任课教师: 郭雪梅

助教: 李声涛、王绍菊

年级&班级	1518	专业(方向)	软件工程(移动信息工程)
学号	15352408	姓名	张稼伟
电话	13531810182	Email	<a href="mailto:709075442@qq.com">709075442@qq.com</a>
开始日期	3.24	完成日期	4.2

### 一、实验题目

1. 下载 MARS 汇编语言编译器, 运行调试 2 个简单的 MIPS 汇编程序, 分别是求和和数组求和, 并按要求对两个程序作出相应修改。

三个代码的具体要求如下:

1. Sum.asm: 单步运行程序, 观察每一步的结果(0...4 这 5 个数字求和), 结果会存入寄存器 \$8 中。修改代码, 使其求 5-10 这六个数字的和
2. SumArray.asm: 单步运行程序, 观察结果(7, 8, 9, 10, 8 这五个在 a 数组中的数字求和), 结果会存在地址从 0x0 开始的 sum 变量中。将变量 a 改为 half word 类型, 修改程序, 比较结果, 观察内存。

### 2. 编写简单的 MIPS 程序

1. 编写 MIPS 代码完成: 在给定 \$s0 和 \$s1 的值的前提下, 将下列值放到 \$t? 寄存器中 (其中 ? 表示任意 0-7 之间的数):

```
$t0 = $s0
$t1 = $s1
$t2 = $t0 + $t1
$t3 = $t1 + $t2
...
$t7 = $t5 + $t6
```

换言之, 对 \$t2 到 \$t7 的每个寄存器, 都存储其前两个 \$t? 寄存器的值。寄存器 \$s0 和 \$s1 中包含初始值。

不要在代码中设置 \$s0 和 \$s1 的值。取而代之, 学会如何在 MARS 中手动设置它们的值。

2. 假定你想编写一个 MIPS 程序 foo, 该程序使用 5 个字的数组, 数组元素初始化为整数 1, ..., 5.

```
.data
```

```
foo: .word 1,2,3,4,5
```

你用程序来把数组 foo 中的每个数加 2 再写回数组 foo

3. 回答下列关于 MARS 的问题.

a. .data, .word, .text 指示器 (directives) 的含义是什么(即, 在每段中放入什么内

容)?

- b. 在 MARS 中如何设置断点 breakpoint?
- c. 在程序运行到断点处停止时, 如何继续执行? 如何单步调试代码?
- d. 如何知道某个寄存器 register 的值是多少? 如何修改寄存器的值.

## 二、实验目的

1. 学习使用汇编语言的编译器 MARS
2. 熟悉并掌握一些简单的汇编指令
3. 提高将课堂所学知识进行实际应用的能力。

## 三、实验内容

### 1. 实验原理

1. MARS 是 MIPS 汇编语言的模拟机, 它能运行 MIPS 汇编程序。其设置: Settings->Memory Configuration->Compact, Data at Address 0. 告诉了模拟机从哪个内存地址开始存数据和代码。

2. 汇编程序用到的寄存器

寄存器号	符号名	用途
0	始终为 0	看起来象浪费,其实很有用
1	at	保留给汇编器使用
2-3	v0,v1	函数返回值
4-7	a0-a3	前头几个函数参数
8-15	t0-t7	临时寄存器,子过程可以不保存就使用
24-25	t8,t9	同上
16-23	s0-s7	寄存器变量,子过程要使用它必须先保存 然后在退出前恢复以保留调用者需要的值
26,27	k0,k1	保留给异常处理函数使用
28	gp	global pointer;用于方便存取全局或者静态变量
29	sp	stack pointer
30	s8/fp	第 9 个寄存器变量;子过程可以用它做 frame pointer
31	ra	返回地址

3. 用到的汇编指令:

`.text address` 指定了后续编译出来的内容放在代码段(可执行), 地址从 address 开始。address 可以不写

`.data address` 指定读写数据段, 用法同.text

`.globl symbol` .globl 使得连接程序(ld)能够识别 symbol, 声明 symbol 是全局可见的。标号\_start 是 GNU 链接器用来指定第一个要执行指令所必须的, 同样的是全局可见的(并且只能出现在一个模块中) 例如: `.global _start` #定义 \_start 为外部程序可以访问的标签...

变量声明: 例如:

```
.data 0x0
sum:    .space 4
i:      .space 4
```

a: .word 7,8,9,10,8 # (数组)

MIPS 指令集(共31条)										
助记符	指令格式						示例	示例含义	操作及其解释	
Bit #	31..26	25..21	20..16	15..11	10..6	5..0				
R-type	op	rs	rt	rd	shamt	imm				
add	000000	rs	rt	rd	00000	100000	add \$1,\$2,\$3	\$1=\$2+\$3	rd <- rs + rt	；其中rs=\$2, rt=\$3, rd=\$1
addu	000000	rs	rt	rd	00000	100001	addu \$1,\$2,\$3	\$1=\$2+\$3	rd <- rs + rt	；其中rs=\$2, rt=\$3, rd=\$1,无符号数
sub	000000	rs	rt	rd	00000	100010	sub \$1,\$2,\$3	\$1=\$2-\$3	rd <- rs - rt	；其中rs=\$2, rt=\$3, rd=\$1
subu	000000	rs	rt	rd	00000	100011	subu \$1,\$2,\$3	\$1=\$2-\$3	rd <- rs - rt	；其中rs=\$2, rt=\$3, rd=\$1,无符号数
and	000000	rs	rt	rd	00000	100100	and \$1,\$2,\$3	\$1=\$2 & \$3	rd <- rs & rt	；其中rs=\$2, rt=\$3, rd=\$1
or	000000	rs	rt	rd	00000	100101	or \$1,\$2,\$3	\$1=\$2   \$3	rd <- rs   rt	；其中rs=\$2, rt=\$3, rd=\$1
xor	000000	rs	rt	rd	00000	100110	xor \$1,\$2,\$3	\$1=\$2 ^ \$3	rd <- rs xor rt	；其中rs=\$2, rt=\$3, rd=\$1(异或)
nor	000000	rs	rt	rd	00000	100111	nor \$1,\$2,\$3	\$1=~(\$2   \$3)	rd <- not(rs   rt)	；其中rs=\$2, rt=\$3, rd=\$1(或非)
slt	000000	rs	rt	rd	00000	101010	slt \$1,\$2,\$3	if(\$2<\$3) \$1=1 else \$1=0	if (rs < rt) rd=1 else rd=0	；其中rs=\$2, rt=\$3, rd=\$1
sltu	000000	rs	rt	rd	00000	101011	sltu \$1,\$2,\$3	if(\$2<\$3) \$1=1 else \$1=0	if (rs < rt) rd=1 else rd=0	；其中rs=\$2, rt=\$3, rd=\$1 (无符号数)
sll	000000	00000	rt	rd	shamt	000000	sll \$1,\$2,10	\$1=\$2<<10	rd <- rt << shamt	；shamt存放移位的位数,也就是指令中的立即数,其中rt=\$2, rd=\$1
srl	000000	00000	rt	rd	shamt	000010	srl \$1,\$2,10	\$1=\$2>>10	rd <- rt >> shamt	；(logical), 其中rt=\$2, rd=\$1

sra	000000	00000	rt	rd	shamt	000011	sra \$1,\$2,10	\$1=\$2>>10	rd <- rt >> shamt	(arithmetic) 注意符号位保留其中rt=\$2, rd=\$1
sliv	000000	rs	rt	rd	00000	000100	sliv \$1,\$2,\$3	\$1=\$2<<\$3	rd <- rt << rs	其中rs=\$3, rt=\$2, rd=\$1
srlv	000000	rs	rt	rd	00000	000110	srlv \$1,\$2,\$3	\$1=\$2>>\$3	rd <- rt >> rs	(logical)其中rs=\$3, rt=\$2, rd=\$1
srav	000000	rs	rt	rd	00000	000111	srav \$1,\$2,\$3	\$1=\$2>>\$3	rd <- rt >> rs	(arithmetic) 注意符号位保留其中rs=\$3, rt=\$2, rd=\$1
jr	000000	rs	00000	00000	00000	001000	jr \$31	goto \$31	PC <- rs	
I-type	op	rs	rt	immediate						
addi	001000	rs	rt	immediate		addi \$1,\$2,100	\$1=\$2+100	rt <- rs + (sign-extend)immediate	其中rt=\$1,rs=\$2	
addiu	001001	rs	rt	immediate		addiu \$1,\$2,100	\$1=\$2+100	rt <- rs + (zero-extend)immediate	其中rt=\$1,rs=\$2	
andi	001100	rs	rt	immediate		andi \$1,\$2,10	\$1=\$2 & 10	rt <- rs & (zero-extend)immediate	其中rt=\$1,rs=\$2	
ori	001101	rs	rt	immediate		ori \$1,\$2,10	\$1=\$2   10	rt <- rs   (zero-extend)immediate	其中rt=\$1,rs=\$2	
xori	001110	rs	rt	immediate		xori \$1,\$2,10	\$1=\$2 ^ 10	rt <- rs xor (zero-extend)immediate	其中rt=\$1,rs=\$2	
lui	001111	00000	rt	immediate		lui \$1,100	\$1=100*65536	rt <- immediate*65536	将16位立即数放到目标寄存器高16位, 目标寄存器的 低16位填0	
lw	100011	rs	rt	immediate		lw \$1,10(\$2)	\$1=memory[\$2+10]	rt <- memory[rs + (sign-extend)immediate]	rt=\$1,rs=\$2	
sw	101011	rs	rt	immediate		sw \$1,10(\$2)	memory[\$2+10]=\$1	memory[rs + (sign-extend)immediate] <- rt	rt=\$1,rs=\$2	
beq	000100	rs	rt	immediate		beq \$1,\$2,10	if(\$1==\$2) goto PC+4+40	if (rs == rt) PC <- PC+4 + (sign-extend)immediate<<2		
bne	000101	rs	rt	immediate		bne \$1,\$2,10	if(\$1!=\$2) goto PC+4+40	if (rs != rt) PC <- PC+4 + (sign-extend)immediate<<2		
slti	001010	rs	rt	immediate		slti \$1,\$2,10	if(\$2<10) \$1=1 else \$1=0	if (rs < (sign-extend)immediate) rt=1 else rt=0	其中rs=\$2, rt=\$1	
sltiu	001011	rs	rt	immediate		sltiu \$1,\$2,10	if(\$2<10) \$1=1 else \$1=0	if (rs < (zero-extend)immediate) rt=1 else rt=0	其中rs=\$2, rt=\$1	

J-type	op	address			
j	000010	address	j 10000	goto 10000	PC <- (PC+4)[31..28], address, 0, 0 ; address=10000/4
jal	000011	address	jal 10000	\$31<-PC+4; goto 10000	\$31<-PC+4; PC <- (PC+4)[31..28], address, 0, 0 ; address=10000/4

### 3.系统调用 System calls

参数所使用的寄存器：\$v0, \$a0, \$a1

返回值使用：\$v0

下表给出了系统调用中对应功能，代码，参数机返回值

Service	Codein \$v0 对应功能的调用码	Arguments 所需参数	Results 返回值
print_int 打印一个整型	\$v0 = 1	\$a0 = integer to be printed 将要打印的整型赋值给 \$a0	
print_float 打印一个浮点	\$v0 = 2	\$f12 = float to be printed 将要打印的浮点赋值给 \$f12	
print_double 打印双精度	\$v0 = 3	\$f12 = double to be printed 将要打印的双精度赋值给 \$f12	
print_string	\$v0 = 4	\$a0 = address of string in memory 将要打印的字符串的地址赋值给 \$a0	
read_int	\$v0 = 5		integer returned in \$v0 将读取的整型赋值给 \$v0
read_float 读取浮点	\$v0 = 6		float returned in \$v0 将读取的浮点赋值给 \$v0
read_double 读取双精度	\$v0 = 7		double returned in \$v0 将读取的双精度赋值给 \$v0
read_string 读取字符串	\$v0 = 8	\$a0 = memory address of string input buffer 将读取的字符串地址赋值给	

		\$a0 \$a1 = length of string buffer (n) 将读取的字符串长度赋值给 \$a1	
sbrk 应该同 C 中的 sbrk() 函数 动态分配内存	\$v0 = 9	\$a0 = amount 需要分配的空间大小 (单位目测是字节 bytes)	address in \$v0 将分配好的空间首地址给 \$v0
exit 退出	\$v0 = 10		

## 2. 实验步骤

1. 下载并打开 MARS, 并做好他的配置: Settings->Memory Configuration->Compact, Data at Address 0.
2. 分别打开 Sum.asm 和 Sum.array 进行运行调试, 观察每一步各个寄存器的变化, 同时更好地理解每一条 MIPS 指令的意义

### 3. Sum.array 的修改:

原代码本质是 `for(i=0;i<5;i++)sum+=i;`

我们的目标是改成 `for(i=5;i<11;i++)sum+=i;`

main 中初始化: `add $9,$0,$0 -> add addi $9,$0,5` #这一步初始化 i=5

loop 中: `slti $10,$9,5 -> slti $10,$9,11` #这一步将 i<5 改为 i<11

其余代码不变, 完整代码见附件。

### 4. SumArray.asm 的修改:

题目要求我们将 word 改成 .half。这里我们要明白一点, halfword 所占内存是 word 的一半。除了将声明变量时的 word 改为 halfword 和将所有 sw, lw

改为 sh, lh 外, loop 中第一句 `sll $10, $9, 2` 要改为 `sll $10, $9, 1`。这一句的意思在于寻找得到 a[i] 的地址, 原本 word 类型每个 a[i] 占 4 个字节, 改为 half 之后每个 a[i] 占 2 个字节。

### 5. 编写程序的第一题:

将题目给出的代码按照顺序补充完整即可, 详见附件。MARS 中手动设置 \$s0 和 \$s1 的值的方法是, 先将代码编译和可以看到如下界面:

Registers	Coproc 1	Coproc 0
Name	Number	Value
zero	0	0x00000000
at	1	0x00000000
v0	2	0x00000000
v1	3	0x00000000
a0	4	0x00000000
a1	5	0x00000000
a2	6	0x00000000
a3	7	0x00000000
t0	8	0x00000000
t1	9	0x00000000
t2	10	0x00000000
t3	11	0x00000000
t4	12	0x00000000
t5	13	0x00000000
t6	14	0x00000000
t7	15	0x00000000
s0	16	0x00000000
s1	17	0x00000000

在运行前先双击后面的 value 就可以手动设置值。

#### 6. 编写程序第二题:

这题有我采取用循环去写。

声明一个变量 i 变初始化为 0:

```
2 i: .space 4
    sw $0, 0($0) #i=0
```

运行时将 i 存储在 \$9 中:

```
lw $9, 0($0)
```

循环中 i\*4 得到 a[i] 真正的地址:

```
sll $10, $9, 2 #转化i to word offset
```

取出 foo[i] 并+2 再存回去:

```
lw $8, 4($10) #load foo[i]
addiu $8, $8, 2 #foo[i]+=2
sw $8, 4($10)
```

i+1 并判断是否推出循环:

```
addiu $9, $9, 1 #i=i+1
slti $10, $9, 5 #i>=5 则$10=0, 反之=1
bne $10, $0, loop # $10不等于0则继续循环
```

#### 3. 回答下列关于 MARS 的问题:

a. .data, .word, .text 指示器 (directives) 的含义是什么(即, 在每段中放入什么内容)?

答: 在汇编中, .data 是数据段的标识, 即其后面可以声明变量。 .word 是“双字节”的标识, 可以声明双字节变量。 .text 是“文本”的标识, 后面跟真正的程序代码。


b. 在 MARS 中如何设置断点 breakpoint?

答: 编译后在如下界面左边 Bkpt 列上选择需要加断点的行打勾即可

Text Segment				
Bkpt	Address	Code	Basic	
<input type="checkbox"/>	0x00003000	0xac000000	sw \$0, 0x00000000(\$0)	7: sw \$0, 0(\$0) #i=0
<input checked="" type="checkbox"/>	0x00003004	0x8c090000	lw \$9, 0x00000000(\$0)	8: lw \$9, 0(\$0)
<input type="checkbox"/>	0x00003008	0x00095080	sll \$10, \$9, 0x00000002	10: sll \$10, \$9, 2 #转化i to word offset
<input type="checkbox"/>	0x0000300c	0x8d480004	lw \$8, 0x00000004(\$10)	11: lw \$8, 4(\$10) #load foo[i]
<input type="checkbox"/>	0x00003010	0x25080002	addiu \$8, \$8, 0x00000002	12: addiu \$8, \$8, 2 #foo[i]+=2
<input type="checkbox"/>	0x00003014	0xad480004	sw \$8, 0x00000004(\$10)	13: sw \$8, 4(\$10)
<input type="checkbox"/>	0x00003018	0x25290001	addiu \$9, \$9, 0x00000001	14: addiu \$9, \$9, 1 #i=i+1
<input type="checkbox"/>	0x0000301c	0x292a0005	slti \$10, \$9, 0x00000005	15: slti \$10, \$9, 5 #i>=5 则\$10=0, 反之=1
<input type="checkbox"/>	0x00003020	0x1540fff9	bne \$10, \$0, 0xfffffffff9	16: bne \$10, \$0, loop # \$10不等于0则继续循环
<input type="checkbox"/>	0x00003024	0x3402000a	ori \$2, \$0, 0x0000000a	18: ori \$v0, \$0, 10
<input type="checkbox"/>	0x00003028	0x0000000c	syscall	19: syscall

c. 在程序运行到断点处停止时, 如何继续执行? 如何单步调试代码?

答: 点  可以继续运行, 同时这个按钮也是单步运行, 点一次运行一次代码。

单步调试的时候, 首先让程序运行到断点处, 然后点上面这个按钮一步一步继续运行。  这个按钮可以实现单步回退, 返回上一句代码。

d. 如何知道某个寄存器 register 的值是多少? 如何修改寄存器的值。

答: 在运行界面右边可以查看 register 的值, 如下左图。修改寄存器的值只需要在下左图这个界面, 双击寄存器对应的 value 栏, 就可以手动修改寄存器的值。如下右图, 修改 \$s1 的值。

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x00001800
\$sp	29	0x00002ffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00003000
hi		0x00000000
lo		0x00000000

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000022
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x00001800
\$sp	29	0x00002ffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00003000
hi		0x00000000
lo		0x00000000

## 四、实验结果

### 1.Sum.asm:

源代码:

Text Segment					Labels	
Bkpt	Address	Code	Basic	Source	Label	Address
	0x00003000	0x00004020	add \$8,\$0,\$0	5: add \$8,\$0,\$0	(global)	
	0x00003004	0x00004820	add \$9,\$0,\$0	6: add \$9,\$0,\$0	main	0x00003000
	0x00003008	0x01094021	addu \$8,\$8,\$9	8: addu \$8,\$8,\$9		
	0x0000300c	0x21290001	addi \$9,\$9,0x00000001	9: addi \$9,\$9,1	mips1.asm	
	0x00003010	0x292a0005	slti \$10,\$9,0x00000005	10: slti \$10,\$9,5	loop	0x00003008
	0x00003014	0x1540ffff	bne \$10,\$0,0xffffffff	11: bne \$10,\$0,loop	end	0x00003018
	0x00003018	0x3402000a	ori \$2,\$0,0x0000000a	13: ori \$v0,\$0,10		
	0x0000301c	0x0000000c	syscall	14: syscall		

\$t0	8	0x0000000a
------	---	------------

修改后:

Text Segment					Labels	
Bkpt	Address	Code	Basic	Source	Label	Address
	0x00003000	0x00004020	add \$8,\$0,\$0	5: add \$8,\$0,\$0 #sum=0	(global)	
	0x00003004	0x20090005	addi \$9,\$0,0x00000005	6: addi \$9,\$0,5 # i=5	main	0x00003000
	0x00003008	0x01094021	addu \$8,\$8,\$9	8: addu \$8,\$8,\$9 #sum=sum+i		
	0x0000300c	0x21290001	addi \$9,\$9,0x00000001	9: addi \$9,\$9,1 # i++	mips1.asm	
	0x00003010	0x292a0005	slti \$10,\$9,0x00000005	10: slti \$10,\$9,5 # if(\$9<5) \$10=1 else \$10=0	loop	0x00003008
	0x00003014	0x1540ffff	bne \$10,\$0,0xffffffff	11: bne \$10,\$0,loop #if(\$10!=0) goto loop	end	0x00003018
	0x00003018	0x3402000a	ori \$2,\$0,0x0000000a	13: ori \$v0,\$0,10		
	0x0000301c	0x0000000c	syscall	14: syscall #这两步是系统调用，\$v0=10 代表系统调用退出功能		

\$t0	8	0x0000002d
------	---	------------

### 2.Sumarray.asm:

源代码:

Text Segment					Labels	
Bkpt	Address	Code	Basic	Source	Label	Address
	0x00003000	0xac000000	sw \$0,0x00000000(\$0)	10: sw \$0,0(\$0) # sum = 0;	(global)	
	0x00003004	0xac000004	sw \$0,0x00000004(\$0)	11: sw \$0,4(\$0) # for (i = 0;	main	0x00003000
	0x00003008	0x8c090004	lw \$9,0x00000004(\$0)	12: lw \$9,4(\$0) # allocate register for i		
	0x0000300c	0x8c000000	lw \$8,0(\$0)	13: lw \$8,0(\$0) # choose register \$8 to hold value for sum		
	0x00003010	0x00090080	sll \$10,\$9,2	15: sll \$10,\$9,2 # covert 'i' to word offset		
	0x00003014	0x844a0008	lw \$10,0x00000008(\$10)	16: lw \$10,8(\$10) # load a[i]	sum	0x00000000
	0x00003018	0x010a0021	addu \$8,\$8,\$10	17: addu \$8,\$8,\$10 # sum = sum + a[i];	i	0x00000004
	0x0000301c	0xac080000	sw \$8,0x00000000(\$0)	18: sw \$8,0(\$0) # update variable in memory	a	0x00000008
	0x00003020	0x21290001	addi \$9,\$9,0x00000001	19: addi \$9,\$9,1 # for (...): i++	loop	0x00003010
	0x00003024	0xac090004	sw \$9,0x00000004(\$0)	20: sw \$9,4(\$0) # update memory	end	0x00003030
	0x00003028	0x292a0005	slti \$10,\$9,0x00000005	21: slti \$10,\$9,5 # for (...): i<5;		
	0x0000302c	0x1540ffff	bne \$10,\$0,0xffffffff	22: bne \$10,\$0,loop		
	0x00003030	0x3402000a	ori \$2,\$0,0x0000000a	24: ori \$v0,\$0,10 # system call 10 for exit		
	0x00003034	0x0000000c	syscall	25: syscall # we are out of here.		

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x0000000a	0x00000005	0x00000007	0x00000008	0x00000009	0x0000000a	0x00000008	0x00000000



\$t0	8	0x0000002a
------	---	------------

修改后：

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00003000	0x40000000	sh \$0,0x00000000(\$0)	8: sh \$0,0(\$0) #sum=0;
	0x00003004	0x40000004	sh \$0,0x00000004(\$0)	9: sh \$0,4(\$0) # for (i = 0;
	0x00003008	0x84090004	lh \$9,0x00000004(\$0)	10: lh \$9,4(\$0) # allocate register for i
	0x0000300c	0x84090000	lh \$8,0x00000000(\$0)	11: lh \$8,0(\$0) # choose register \$8 to hold value for
	0x00003010	0x00095040	slt \$10,\$9,0x00000001	13: slt \$10, \$9, 1 # covert "i" to halfword offset
	0x00003014	0x854a0008	lh \$10,0x00000008(\$10)	14: lh \$10, 8(\$10) # load a[i]
	0x00003018	0x010a4021	addu \$8,\$8,\$10	15: addu \$8,\$8,\$10 # sum = sum + a[i];
	0x0000301c	0x40800000	sh \$8,0x00000000(\$0)	16: sh \$8, 0(\$0) # update variable in memory
	0x00003020	0x21290001	addi \$9,\$9,0x00000001	17: addi \$9, \$9, 1 # for (...; i++
	0x00003024	0x40900004	sh \$9,0x00000004(\$0)	18: sh \$9, 4(\$0) # update memory
	0x00003028	0x292a0005	slti \$10,\$9,0x00000005	19: slti \$10, \$9, 5 # for (...; i<5;
	0x0000302c	0x1540fff8	bne \$10,\$0,0xfffffff8	20: bne \$10, \$0, loop
	0x00003030	0x3402000a	ori \$v0,\$0,0x0000000a	22: ori \$v0, \$0, 10
	0x00003034	0x0000000c	syscall	23: syscall

Labels

Label	Address
(global)	
main	0x00003000
Sumarray.asm	
sum	0x00000000
i	0x00000004
a	0x00000008
loop	0x00003010
end	0x00003030

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x0000002a	0x00000005	0x00080007	0x000a0009	0x00000008	0x00000000	0x00000000	0x00000000

### 3.编写程序第一题：

\$s0	16	0x00000001
\$s1	17	0x00000002

\$t0	8	0x00000001
\$t1	9	0x00000002
\$t2	10	0x00000003
\$t3	11	0x00000005
\$t4	12	0x00000008
\$t5	13	0x0000000d
\$t6	14	0x00000015
\$t7	15	0x00000022

### 4.编写程序第二题：

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x00000000	0x00000000	0x00000003	0x00000004	0x00000005	0x00000006	0x00000000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

## 五、实验感想

通过这次实验，我对 MIPS 汇编代码的基本格式有了了解。我们在理论课上学习了一些汇编指令，然后我一直对他们一知半解，掌握地不好，做理论作业的时候很迷茫。但是通过这次实验，通过对每个代码的跟踪运行，对数据的观察，以及到后面自己动手编写代码，我觉得自己对这些指令有了一个较清晰的理解。

## 附录：

### 1.Sum.asm：

原代码：



```

1  #Add the first five integers
2  .text 0x3000
3  .globl main
4  main:
5      add $8,$0,$0 #sum=0
6      add $9,$0,$0 # i=0
7  loop:
8      addu $8,$8,$9 #sum=sum+i
9      addi $9,$9,1 # i++
10     slti $10,$9,5 # if($9<5) $10=1 else $10=0
11     bne $10,$0,loop #if($10!=0)goto loop
12 end:
13     ori $v0,$0,10
14     syscall      #这两步是系统调用, $v0=10 代表系统调用退出功能

```

修改后:

```

1  #Add the first five integers
2  .text 0x3000
3  .globl main
4  main:
5      add $8,$0,$0 #sum=0
6      addi $9,$0,5 # i=5
7  loop:
8      addu $8,$8,$9 #sum=sum+i
9      addi $9,$9,1 # i++
10     slti $10,$9,11 # if($9<11) $10=1 else $10=0
11     bne $10,$0,loop #if($10!=0)goto loop
12 end:
13     ori $v0,$0,10
14     syscall      #这两步是系统调用, $v0=10 代表系统调用退出功能

```

## 2.SumArray.asm:

原代码:

```

1  # Add the numbers in an array
2  .data 0x0
3  sum: .space 4
4  i: .space 4
5  a: .word 7,8,9,10,8
6
7  .text 0x3000
8  .globl main
9  main:
10     sw $0, 0($0) # sum = 0;
11     sw $0, 4($0) # for (i = 0;
12     lw $9, 4($0) # allocate register for i
13     lw $8, 0($0) # choose register $8 to hold value for sum
14 loop:
15     sll $10, $9, 2 # covert "i" to word offset
16     lw $10, 8($10) # load a[i]
17     addu $8, $8, $10 # sum = sum + a[i];
18     sw $8, 0($0) # update variable in memory
19     addi $9, $9, 1 # for (...; ...; i++
20     sw $9, 4($0) # update memory
21     slti $10, $9, 5 # for (...; i<5;
22     bne $10, $0, loop
23 end:
24     ori $v0, $0, 10 # system call 10 for exit
25     syscall # we are out of here.

```

修改后:

```
1  .data 0x0
2  sum: .space 4
3  i: .space 4
4  a: .half 7, 8, 9, 10, 8
5  .text 0x3000
6  .globl main
7  main:
8      sh $0, 0($0) #sum=0;
9      sh $0, 4($0) # for (i = 0;
10     lh $9, 4($0) # allocate register for i
11     lh $8, 0($0) # choose register $8 to hold value for
12 loop:
13     sll $10, $9, 1 # covert "i" to halfword offset
14     lh $10, 8($10) # load a[i]
15     addu $8, $8, $10 # sum = sum + a[i];
16     sh $8, 0($0) # update variable in memory
17     addi $9, $9, 1 # for (...; ...; i++
18     sh $9, 4($0) # update memory
19     slti $10, $9, 5 # for (...; i<5;
20     bne $10, $0, loop
21 end:
22     ori $v0, $0, 10
23     syscall
```

### 3.编写程序第一题:

```
1  .text 0x3000
2  .globl main
3  main:
4      addu $t0, $0, $s0
5      addu $t1, $0, $s1
6      addu $t2, $t0, $t1
7      addu $t3, $t1, $t2
8      addu $t4, $t2, $t3
9      addu $t5, $t3, $t4
10     addu $t6, $t4, $t5
11     addu $t7, $t5, $t6
12 end:
13     ori $v0, $0, 10
14     syscall
```

### 4.编写程序第二题

```
1  .data 0x0
2  i: .space 4
3  foo: .word 1, 2, 3, 4, 5
4  .text 0x3000
5  .globl _start
6  _start:
7      sw $0, 0($0) #i=0
8      lw $9, 0($0)
9  loop:
10     sll $10, $9, 2 #转化i to word offset
11     lw $8, 4($10) #load foo[i]
12     addiu $8, $8, 2 #foo[i]+=2
13     sw $8, 4($10)
14     addiu $9, $9, 1 #i=i+1
15     slti $10, $9, 5 #i>=5 则$10=0, 反之=1
16     bne $10, $0, loop # $10不等于0则继续循环
17 end:
18     ori $v0, $0, 10
19     syscall
```