

《计算机组成原理实验》

(第四次)

厦门大学信息学院软件工程系 曾文华

2022年4月28日

目录

一、验证实验

1. MIPS汇编语言程序的运行
2. RISC-V汇编语言程序的运行
3. MIPS单周期处理器数据通路

验证实验：有现成程序或电路

请同学们验证，并对实验结果进行分析

二、设计实验

1. MIPS汇编语言程序设计
 - ① 计算费波那契数列的程序
 - ② 冒泡排序算法的程序
2. RISC-V汇编语言程序设计
 - ① 计算费波那契数列的程序
 - ② 冒泡排序算法的程序
3. Intel x86汇编语言程序设计
 - 冒泡排序算法的程序

设计实验：需要同学们自己编写程序

实验电路设计文件：
第四次实验——CPU设计实验（单周期MIPS处理器）.circ

实验要求

- 实验结束后1周内（**5月5日晚上24点前**）提交实验报告（Word文档），同时提交相应的汇编语言程序（完成实验目录中“设计实验”的要求）。

一、验证实验

1、MIPS汇编语言程序的运行

- (1) 运行MIPS汇编仿真器

- 下载“jdk-8u121-windows-x64.exe”、“Mars4_5.jar”，并放到C盘\mips目录中

- 查看电脑上是否已有JDK?

- 打开“命令提示符”，执行：

- **java -version**



```
命令提示符
Microsoft Windows [版本 10.0.19044.1645]
(c) Microsoft Corporation。保留所有权利。

C:\Users\lily2002>java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)

C:\Users\lily2002>
```

- 如果没有，运行：jdk-8u121-windows-x64.exe，安装JDK

- 运行MIPS汇编仿真器Mars4_5.jar

- 打开“命令提示符”，执行：

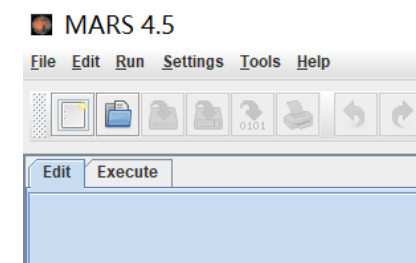
- **cd \mips**

- **java -jar Mars4_5.jar**



```
命令提示符 - java -jar Mars4_5.jar
C:\Users\lily2002>cd \mips

C:\mips>java -jar Mars4_5.jar
四月 24, 2022 1:09:34 下午 java.util.pre
fs.WindowsPreferences <init>
WARNING: Could not open/create prefs root
node Software\JavaSoft\Prefs at root 0
x80000002. Windows RegCreateKeyEx(...) r
eturned error code 5.
```

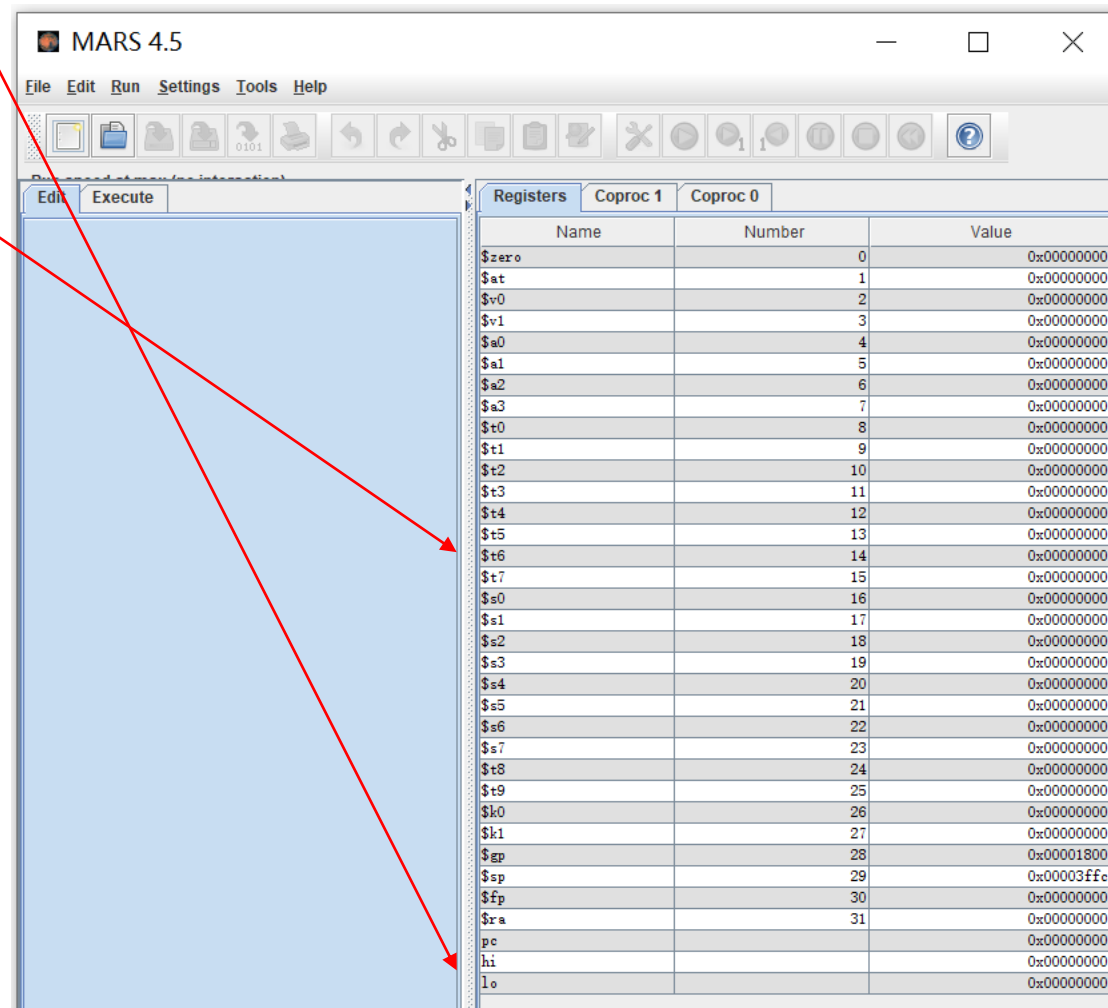


- MIPS处理器的32个寄存器，以及pc、hi、lo寄存器

PC (程序计数器)、HI (乘除结果高位寄存器) 和LO (乘除结果低位寄存器)

表5.7 MIPS的通用寄存器

寄存器#	助记符	释义
\$0	\$zero	固定值为0 硬件置位
\$1	\$at	汇编器保留，临时变量
\$2~\$3	\$v0~\$v1	函数调用返回值
\$4~\$7	\$a0~\$a3	4个函数调用参数
\$8~\$15	\$t0~\$t7	暂存寄存器，被调用者按需保存
\$16~\$23	\$s0~\$s7	save寄存器，调用者按需保存
\$24~\$25	\$t8~\$t9	暂存寄存器，同上
\$26~\$27	\$k0~\$k1	操作系统保留，中断异常处理
\$28	\$gp	全局指针 (Global Pointer)
\$29	\$sp	堆栈指针 (Stack Pointer)
\$30	\$fp	帧指针 (Frame Pointer)
\$31	\$ra	函数返回地址 (Return Address)



- (2) 运行第一个MIPS汇编语言程序

- **helloworld_mips.asm**为显示字符串“Hello, World!”的MIPS汇编语言程序。

helloworld.asm - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

#helloworld.asm

.data

out_string: .asciiz "\nHello, World!\n"

#数据段开始标志

定义ASCII码字符串

.text

main:

li \$v0,4

la \$a0,out_string

syscall

#代码段开始标志

#4号系统调用, 显示字符串

立即数4 -> \$v0

#获取字符串的地址

#系统调用

地址out_string -> \$a0

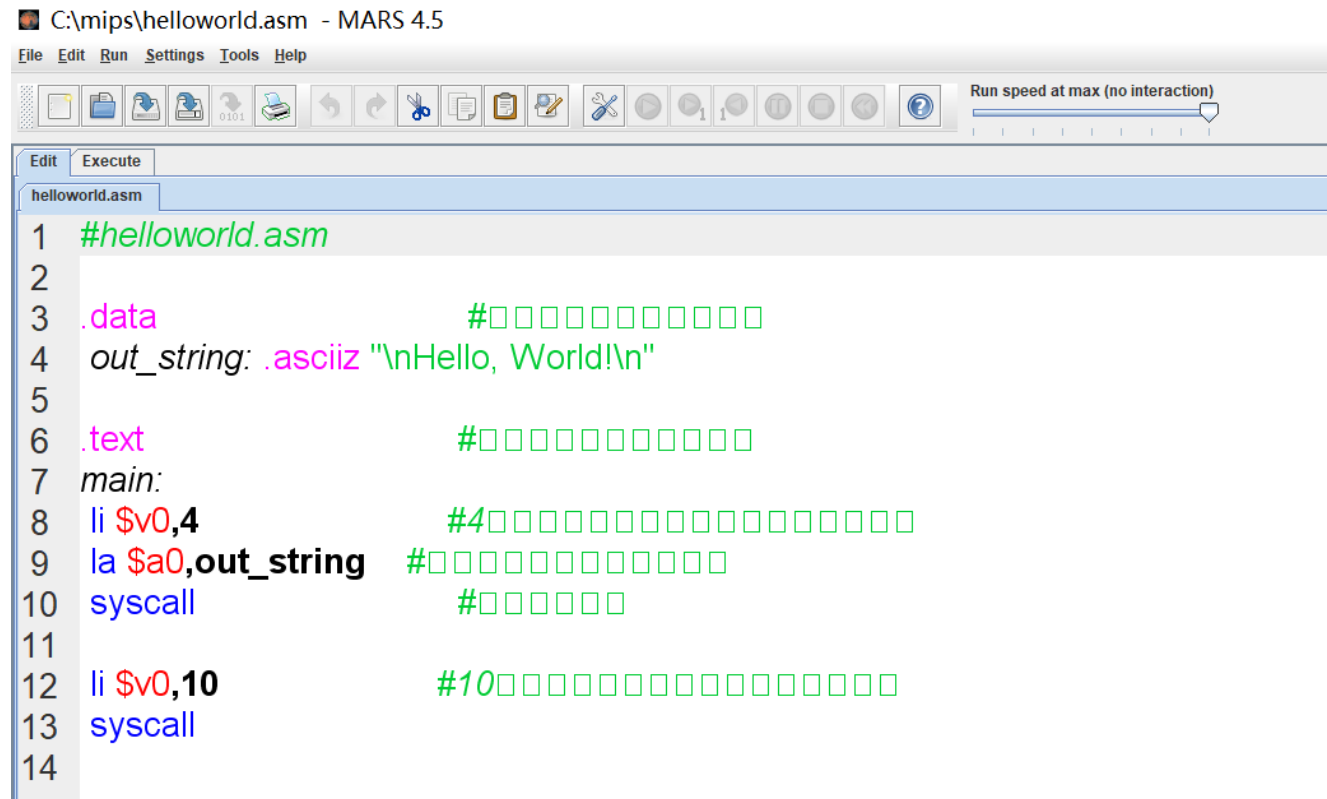
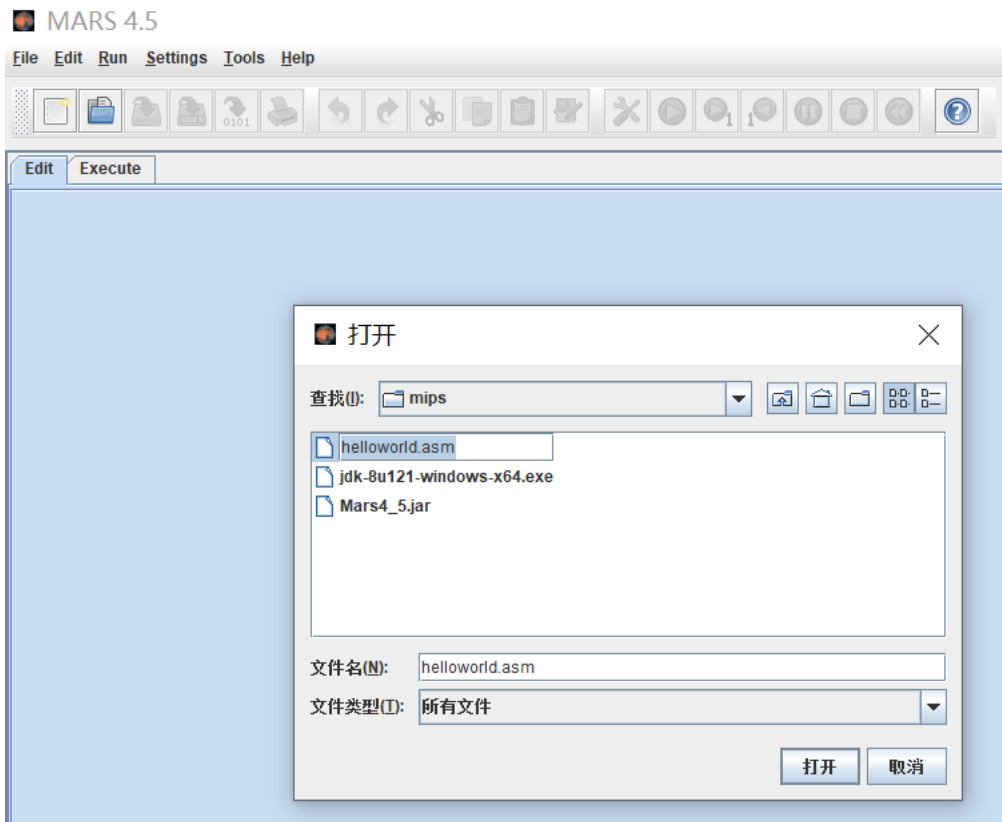
li \$v0,10

syscall

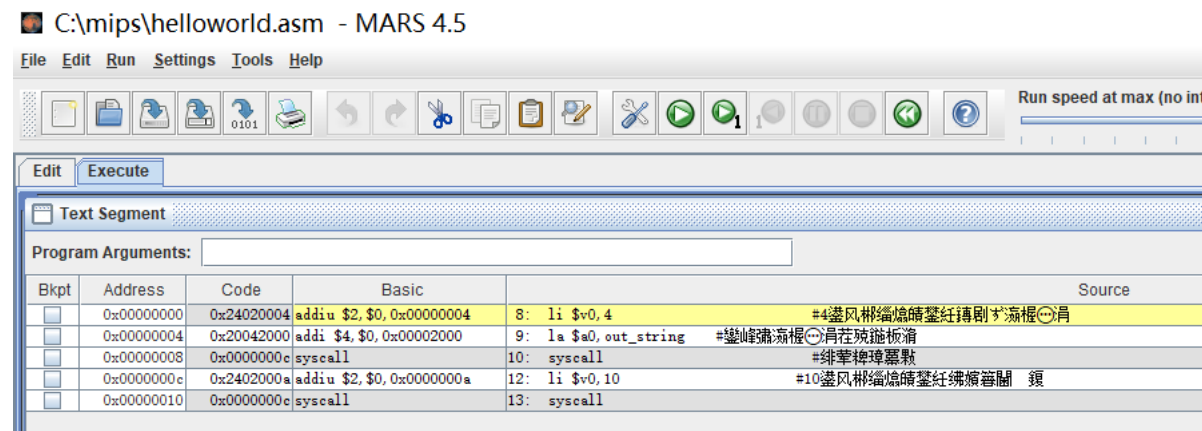
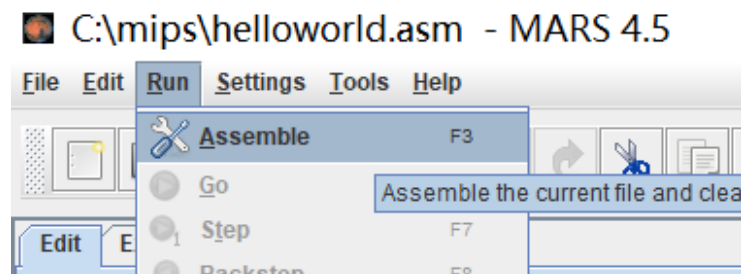
#10号系统调用, 程序退出

立即数10 -> \$v0

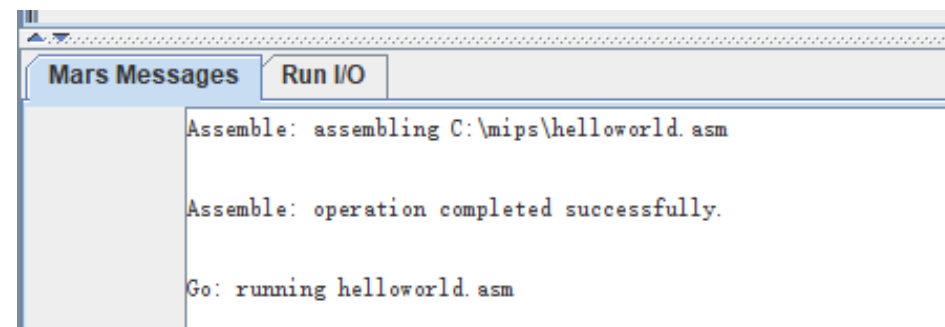
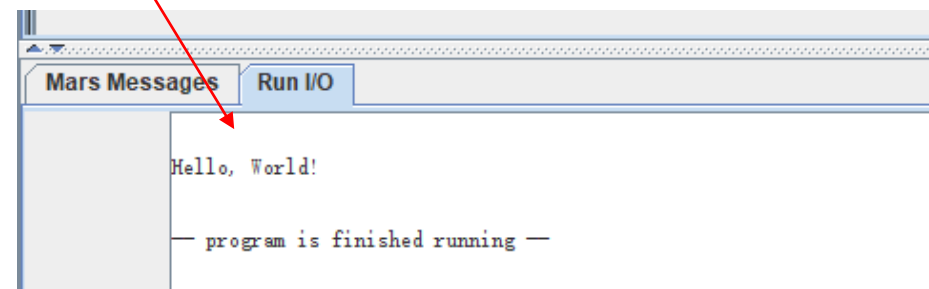
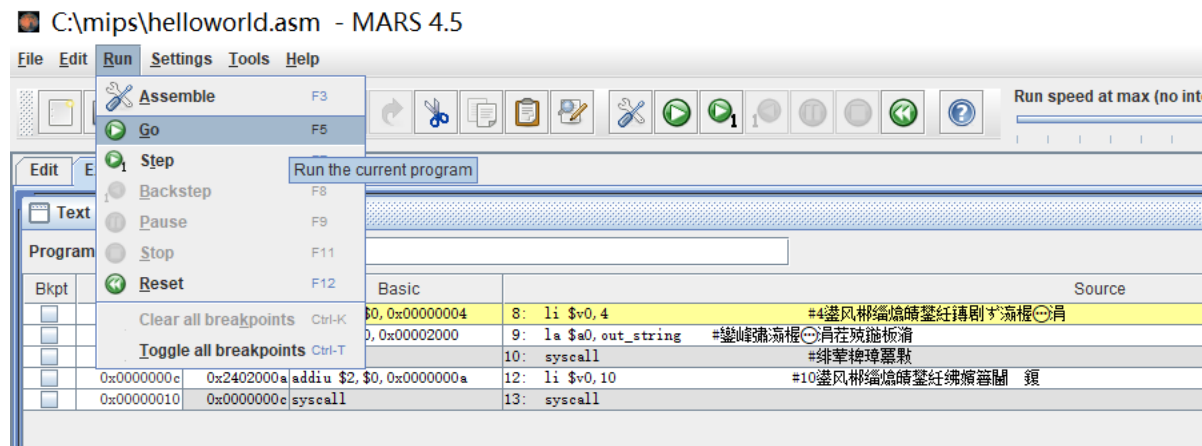
- 在MARS汇编仿真器中**打开**helloworld_mips.asm



- 对helloworld_mips.asm源程序进行汇编（点击：**Assemble**）

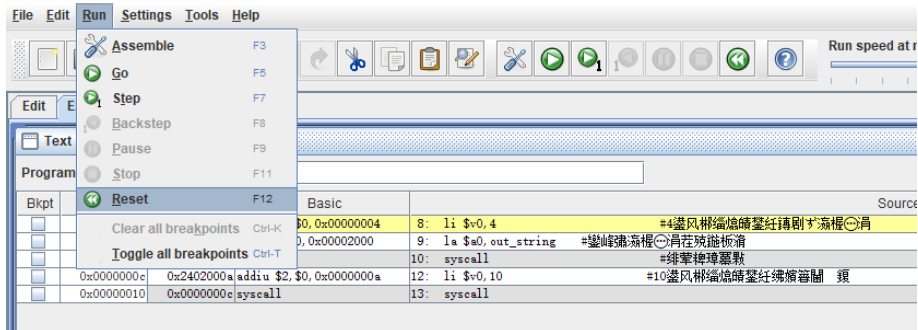


- 运行汇编后的程序（点击：Go），在下面的界面中显示“Hello, World!”



- 也可以**单步**执行程序：首先点击：**Reset**，此时将第一条指令背景置黄色，表示从第一条指令开始执行；然后点击：**Step**，每点击一次Step，执行一条指令；直到所有指令执行完毕。

C:\mips\helloworld.asm - MARS 4.5



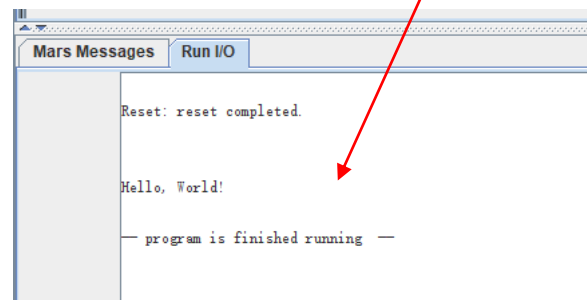
Program Arguments:				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00000000	0x24020004	addiu \$2,\$0,0x00000004	8: li \$v0,4 #4 盞风榑縕焔嶝堇纒縕刷ず源棍⊙肩
<input type="checkbox"/>	0x00000004	0x20042000	addi \$4,\$0,0x00002000	9: la \$a0,out_string #盞嶝嶝嶝嶝⊙肩荏玢玢板消
<input type="checkbox"/>	0x00000008	0x0000000c	syscall	10: syscall #排鞑揀璋慕歎
<input type="checkbox"/>	0x0000000c	0x2402000a	addiu \$2,\$0,0x0000000a	12: li \$v0,10 #10 盞风榑縕焔嶝堇纒縕刷ず源棍⊙肩
<input type="checkbox"/>	0x00000010	0x0000000c	syscall	13: syscall

Program Arguments:				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00000000	0x24020004	addiu \$2,\$0,0x00000004	8: li \$v0,4 #4 盞风榑縕焔嶝堇纒縕刷ず源棍⊙肩
<input type="checkbox"/>	0x00000004	0x20042000	addi \$4,\$0,0x00002000	9: la \$a0,out_string #盞嶝嶝嶝嶝⊙肩荏玢玢板消
<input type="checkbox"/>	0x00000008	0x0000000c	syscall	10: syscall #排鞑揀璋慕歎
<input type="checkbox"/>	0x0000000c	0x2402000a	addiu \$2,\$0,0x0000000a	12: li \$v0,10 #10 盞风榑縕焔嶝堇纒縕刷ず源棍⊙肩
<input type="checkbox"/>	0x00000010	0x0000000c	syscall	13: syscall

Program Arguments:				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00000000	0x24020004	addiu \$2,\$0,0x00000004	8: li \$v0,4 #4 盞风榑縕焔嶝堇纒縕刷ず源棍⊙肩
<input type="checkbox"/>	0x00000004	0x20042000	addi \$4,\$0,0x00002000	9: la \$a0,out_string #盞嶝嶝嶝嶝⊙肩荏玢玢板消
<input type="checkbox"/>	0x00000008	0x0000000c	syscall	10: syscall #排鞑揀璋慕歎
<input type="checkbox"/>	0x0000000c	0x2402000a	addiu \$2,\$0,0x0000000a	12: li \$v0,10 #10 盞风榑縕焔嶝堇纒縕刷ず源棍⊙肩
<input type="checkbox"/>	0x00000010	0x0000000c	syscall	13: syscall

Program Arguments:				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00000000	0x24020004	addiu \$2,\$0,0x00000004	8: li \$v0,4 #4 盞风榑縕焔嶝堇纒縕刷ず源棍⊙肩
<input type="checkbox"/>	0x00000004	0x20042000	addi \$4,\$0,0x00002000	9: la \$a0,out_string #盞嶝嶝嶝嶝⊙肩荏玢玢板消
<input type="checkbox"/>	0x00000008	0x0000000c	syscall	10: syscall #排鞑揀璋慕歎
<input type="checkbox"/>	0x0000000c	0x2402000a	addiu \$2,\$0,0x0000000a	12: li \$v0,10 #10 盞风榑縕焔嶝堇纒縕刷ず源棍⊙肩
<input type="checkbox"/>	0x00000010	0x0000000c	syscall	13: syscall

Program Arguments:				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00000000	0x24020004	addiu \$2,\$0,0x00000004	8: li \$v0,4 #4 盞风榑縕焔嶝堇纒縕刷ず源棍⊙肩
<input type="checkbox"/>	0x00000004	0x20042000	addi \$4,\$0,0x00002000	9: la \$a0,out_string #盞嶝嶝嶝嶝⊙肩荏玢玢板消
<input type="checkbox"/>	0x00000008	0x0000000c	syscall	10: syscall #排鞑揀璋慕歎
<input type="checkbox"/>	0x0000000c	0x2402000a	addiu \$2,\$0,0x0000000a	12: li \$v0,10 #10 盞风榑縕焔嶝堇纒縕刷ず源棍⊙肩
<input type="checkbox"/>	0x00000010	0x0000000c	syscall	13: syscall



• helloworld_mips.asm源程序对应的指令和机器码：

机器码在存储器中的地址	指令的机器码	源程序对应的MIPS指令	源程序
Address	Code	Basic	
0x00000000	0x24020004	addiu \$2, \$0, 0x00000004	8: li \$v0, 4
0x00000004	0x20042000	addi \$4, \$0, 0x00002000	9: la \$a0, out_string
0x00000008	0x0000000c	syscall	10: syscall
0x0000000c	0x2402000a	addiu \$2, \$0, 0x0000000a	12: li \$v0, 10
0x00000010	0x0000000c	syscall	13: syscall

0+4 -> \$2 4 -> \$v0

0+2000 -> \$4 2000 -> \$a0

0+0a -> \$2 10 -> \$v0

表5.7 MIPS的通用寄存器

寄存器#	助记符	释义
\$0	\$zero	固定值为0 硬件置位
\$1	\$at	汇编器保留，临时变量
\$2~\$3	\$v0~\$v1	函数调用返回值
\$4~\$7	\$a0~\$a3	4个函数调用参数
\$8~\$15	\$t0~\$t7	暂存寄存器，被调用者按需保存
\$16~\$23	\$s0~\$s7	save寄存器，调用者按需保存
\$24~\$25	\$t8~\$t9	暂存寄存器，同上
\$26~\$27	\$k0~\$k1	操作系统保留，中断异常处理
\$28	\$gp	全局指针 (Global Pointer)
\$29	\$sp	堆栈指针 (Stack Pointer)
\$30	\$fp	帧指针 (Frame Pointer)
\$31	\$ra	函数返回地址 (Return Address)

指令在存储器中的起始地址



数据在存储器中的起始地址

源程序

```
.data
out_string: .asciiz "\nHello, World!\n"

.text
main:
li $v0,4
la $a0,out_string
syscall

li $v0,10
syscall
```

- helloworld_mips.asm源程序数据段的内容:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x00002000	0x6c65480a	0x202c6f6c	0x6c726f57	0x000a2164

0x0a, 0x48, 0x65, 0x6C, 0x6C, 0x6F, 0x2c, 0x20, 0x57, 0x6F, 0x72, 0x6C, 0x64, 0x21, 0x0a


\n H e l l o , 空格 W o r l d ! \n

```
.data
```

```
out_string: .asciiz "\nHello, World!\n"
```

• (3) 求累加和的MIPS汇编语言程序

- 求累加和: $result = 1 + 2 + \dots + n$
 - $n=10=0ah$, $result=55=37h$
 - $n=100=64h$, $result=5050=13bah$

 sum_mips.asm - 记事本

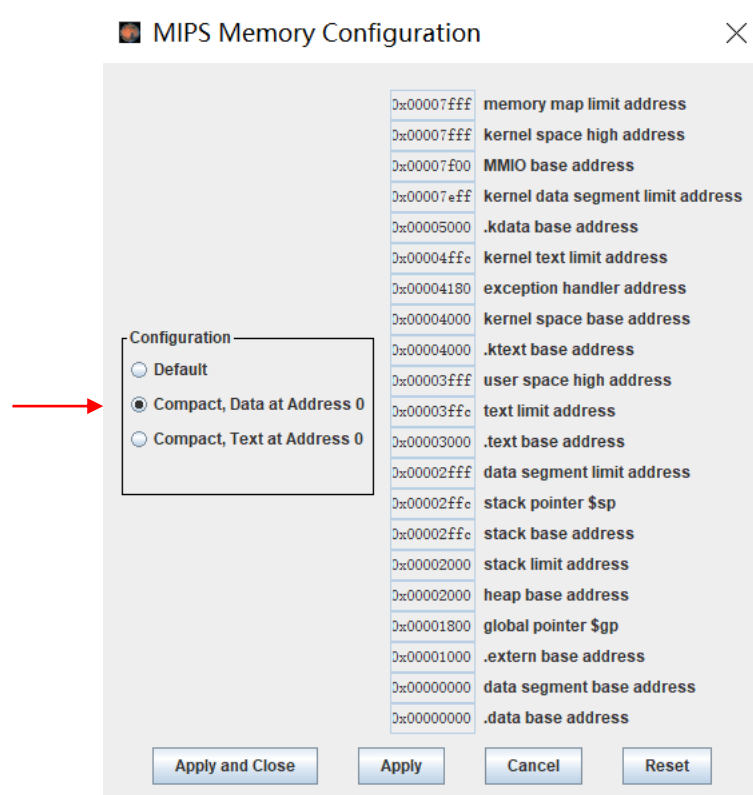
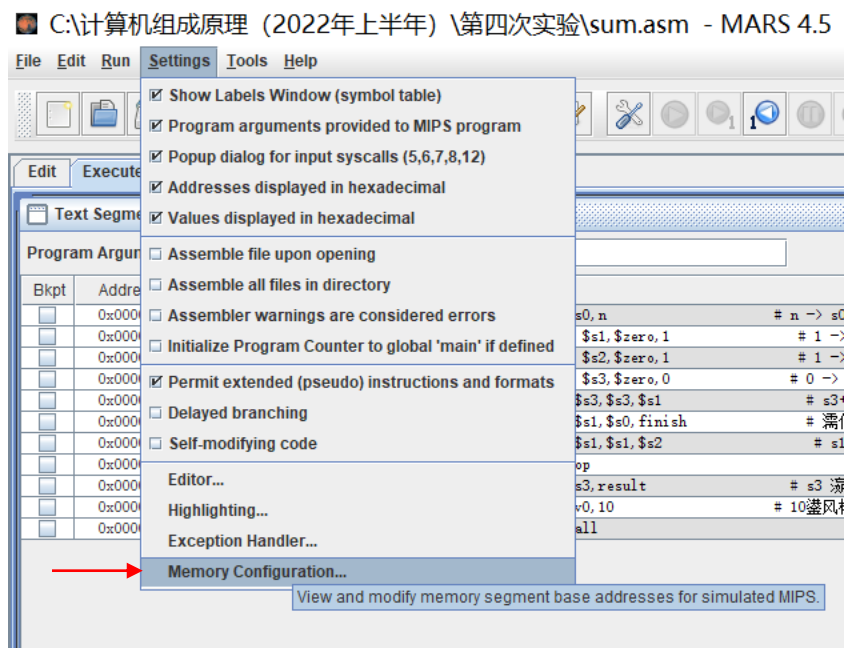
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
#sum.asm                                #求累加和的程序: 1+2+.....+n, n的值为10 (可以改变), 累加和的结果存放到地址为0的存储单元中

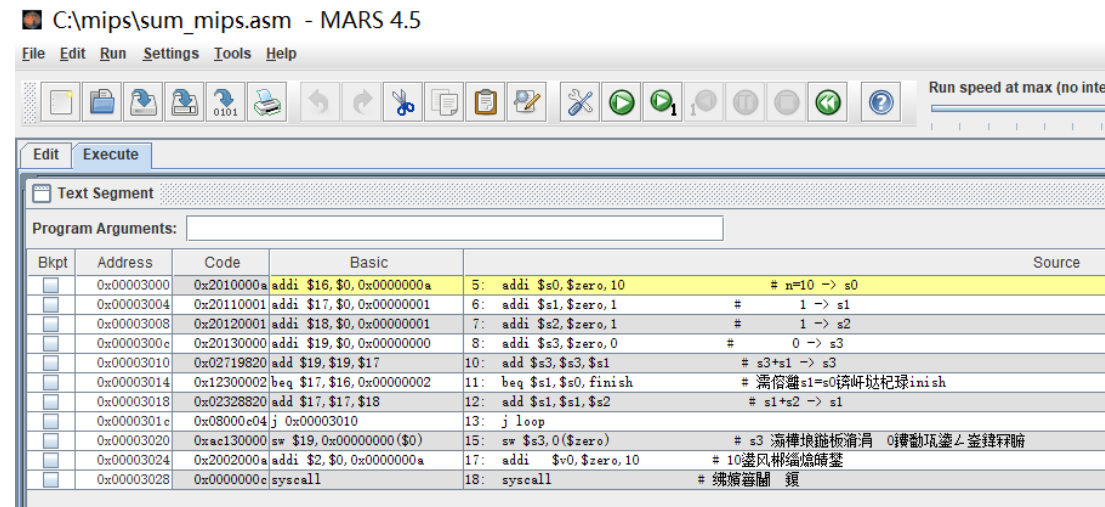
.text                                    #代码段开始标志
main:
    addi $s0,$zero,10                   # n=10 -> s0
    addi $s1,$zero,1                    # 1 -> s1
    addi $s2,$zero,1                    # 1 -> s2
    addi $s3,$zero,0                    # 0 -> s3
loop:
    add $s3,$s3,$s1                     # s3+s1 -> s3
    beq $s1,$s0,finish                 # 如果s1=s0, 则转finish
    add $s1,$s1,$s2                     # s1+s2 -> s1
    j loop
finish:
    sw $s3,0($zero)                    # s3 存到地址为0的存储单元中

    addi $v0,$zero,10                  # 10号系统调用
    syscall                            # 程序退出
```

- 将MARS汇编仿真器的**数据段**的开始地址设置为0



- 在MARS汇编仿真器中**打开**sum_mips.asm
- 对sum_mips.asm源程序进行汇编（点击：**Assemble**）
- 运行汇编后的程序（点击：**Go**）
- 观看数据段中的内容：

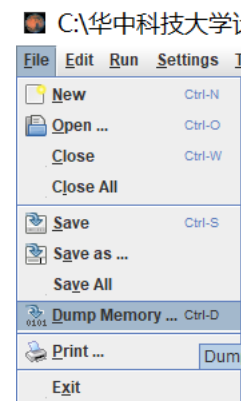


Address	Value (+0)
0x00000000	0x00000037
0x00000020	0x00000000

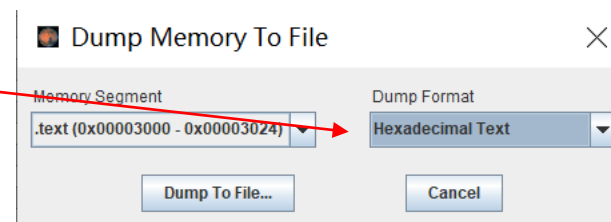
result=55=37h

- 导出汇编后的机器码：

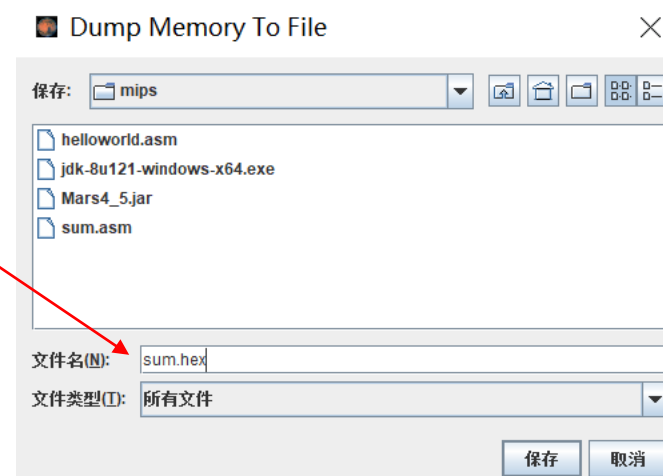
- “File” -> “Dump Memory”



- 保存为十六进制文本格式



- 保存到sum_mips.hex文件中



Code
0x2010000a
0x20110001
0x20120001
0x20130000
0x02719820
0x12300002
0x02328820
0x08000c04
0xac130000
0x2002000a
0x0000000c

sum_mips.hex
文件(F) 编辑(E) 1
2010000a
20110001
20120001
20130000
02719820
12300002
02328820
08000c04
ac130000
2002000a
0000000c

2、RISC-V汇编语言程序的运行

- (1) 运行RISC-V汇编仿真器

- 下载“jdk-8u121-windows-x64.exe”、“**rars1_5.jar**”，并放到C盘\riscv目录中

- 查看电脑上是否已有JDK？

- 打开“命令提示符”，执行：

- **java -version**

- 如果没有，运行：jdk-8u121-windows-x64.exe，安装JDK



```
命令提示符
Microsoft Windows [版本 10.0.19044.1645]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\lily2002>java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)

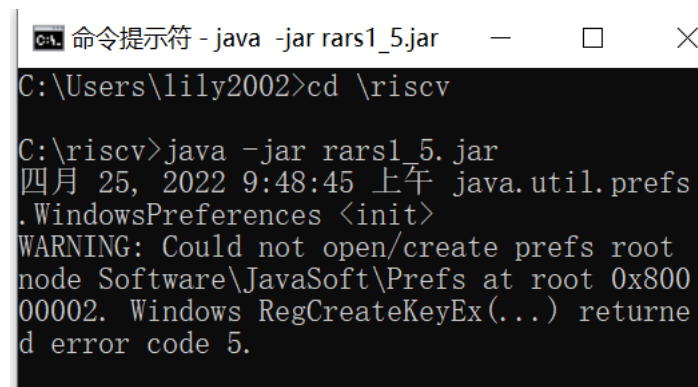
C:\Users\lily2002>
```

- 运行RISC-V汇编仿真器rars1_5.jar

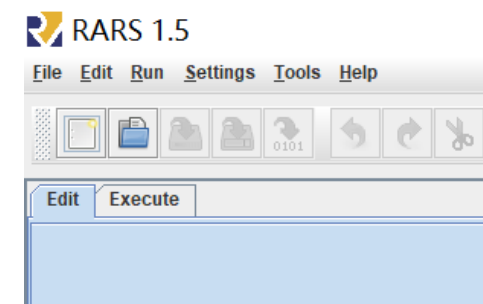
- 打开“命令提示符”，执行：

- **cd \riscv**

- **java -jar rars1_5.jar**



```
命令提示符 - java -jar rars1_5.jar
C:\Users\lily2002>cd \riscv
C:\riscv>java -jar rars1_5.jar
四月 25, 2022 9:48:45 上午 java.util.prefs
.WindowsPreferences <init>
WARNING: Could not open/create prefs root
node Software\JavaSoft\Prefs at root 0x800
00002. Windows RegCreateKeyEx(...) returne
d error code 5.
```



- RISC-V的寄存器：32个寄存器，以及pc

表5.12 RISC-V的通用寄存器

编号	助记符	英文全称	功能描述
x0	zero	zero	恒零值，可用0号寄存器参与的加法指令实现MOV指令
x1	ra	Return Address	返回地址
x2	sp	Stack Pointer	栈指针，指向栈顶
x3	gp	Global Pointer	全局指针
x4	tp	Thread Pointer	线程寄存器
x5 ~ x7	t0 ~ t2	Temporaries	临时变量，调用者保存寄存器
x8	s0/fp	Saved Register/Frame Pointer	通用寄存器，被调用者保存寄存器，在子程序使用时必须先压栈保存原值，使用后应出栈恢复原值
x9	s1	Saved Registers	通用寄存器，被调用者保存寄存器
x10 ~ x11	a0 ~ a1	Arguments/Return values	用于存储子程序参数或返回值
x12 ~ x17	a2 ~ a7	Arguments	用于存储子程序参数
x18 ~ x27	s2 ~ s11	Saved Registers	通用寄存器，被调用者保存寄存器
x28 ~ x31	t3 ~ t6	Temporaries	临时变量


Registers	Floating Point	Control and Status
Name	Number	
zero	0	
ra	1	
sp	2	
gp	3	
tp	4	
t0	5	
t1	6	
t2	7	
s0	8	
s1	9	
a0	10	
a1	11	
a2	12	
a3	13	
a4	14	
a5	15	
a6	16	
a7	17	
s2	18	
s3	19	
s4	20	
s5	21	
s6	22	
s7	23	
s8	24	
s9	25	
s10	26	
s11	27	
t3	28	
t4	29	
t5	30	
t6	31	
pc		

- RISC-V核心指令集RV32I的9条指令（我们设计的RISC-V单周期处理器仅支持该9条指令）：

- ① add rd,rs1,rs2 ; rs1+rs2->rd
- ② slt rd,rs1,rs2 ; 带符号数比较指令 if rs1<rs2 1->rd else 0->rd
- ③ sltu rd,rs1,rs2 ; 无符号数比较指令 if rs1<rs2 1->rd else 0->rd
- ④ ori rd,rs1,imm12 ; rs1 或 imm12 -> rd
- ⑤ lw rd rs1,imm12 ; M[rs1+imm12] -> rd
- ⑥ lui rd,imm20 ; imm20 -> rd
- ⑦ sw rs2,rs1,imm12 ; rs2 -> M[rs1+imm12]
- ⑧ beq rs1,rs2,imm12 ; if rs1=rs2 then goto imm12
- ⑨ jal rd,imm20 ; goto imm20

- (2) RISC-V求累加和程序

- 利用前面的RISC-V的9条指令，编写一个计算 $1+2+\dots+n$ 的累加和程序，从数据存储器地址为0的存储单元中读入参数 n ，通过循环累加的算法计算累加和，结果保存到数据存储器地址为4的存储单元中。
- 该程序的汇编代码如下： **sum_riscv.asm**

 sum_riscv.asm - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

#计算累加和程序

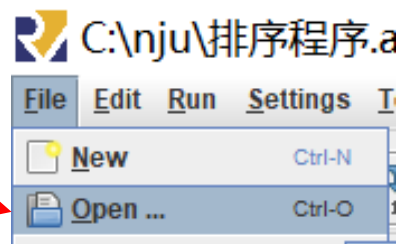
result=1+2+...+n

n存放在主存地址为0的单元，结果result存放在主存地址为4的单元

```
        lw a3, 0(zero)           # 读取主存0号单元（地址为0）的值（n）到a3
        ori a5, zero, 1          # a5内容（循环变量i）为1
        ori a2, zero, 1          # a2内容（循环增量）为1
        ori a4, zero, 0          # a4内容为0
loop:    add a4, a4, a5            # 将a5加到a4（累加和）
        beq a5, a3, finish       # 若a5=n，则跳出循环
        add a5, a5, a2           # a5=a5+1
        jal zero, loop           # 无条件跳转到loop执行
finish:  sw a4, 4(zero)           # 将累加结果保存到主存1号单元（地址为4）
end:    jal zero, end            # 无条件跳转到end执行
```

- 在RARS中打开“sum_riscv.asm”

- File -> Open



C:\nju\RISC-V汇编语言程序\累加程序1.asm* - RARS 1.5

```
File Edit Run Settings Tools Help
[Icons] Run speed at max (no interaction)

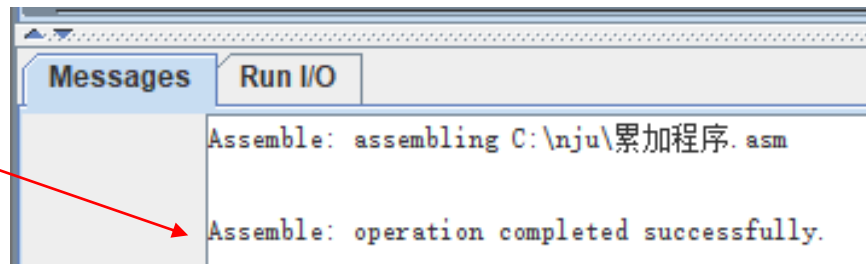
Edit Execute
累加程序1.asm*
1 #计算累加和程序
2     lw a3, 0(zero)           # 读取主存0号单元的值(n)到a3
3     ori a5, zero, 1         # a5内容(循环变量i)为1
4     ori a2, zero, 1         # a2内容(循环增量)为1
5     ori a4, zero, 0         # a4内容为0
6 loop:
7     add a4, a4, a5           # 将i加到a4(累加和)
8     beq a5, a3, finish       # 若a5=n, 则跳出循环
9     add a5, a5, a2           # a5=a5+1
10    jal zero, loop           # 无条件跳转到loop执行
11 finish:
12    sw a4, 4(zero)           # 将累加结果保存到主存1号单元
13 end:
14    jal zero, end            # 无条件跳转到end执行
15
```

- 在RARS中汇编“sum_riscv.asm”

- Run -> **Assemble**



- 左下角窗口中显示汇编成功!



- 以下为汇编后的机器码:

Text Segment					
Bkpt	Address	Code	Basic		
<input type="checkbox"/>	0x00003000	0x00002683	lw x13, 0x00000000(x0)	2:	lw a3, 0(zero)
<input type="checkbox"/>	0x00003004	0x00106793	ori x15, x0, 0x00000001	3:	ori a5, zero, 1
<input type="checkbox"/>	0x00003008	0x00106613	ori x12, x0, 0x00000001	4:	ori a2, zero, 1
<input type="checkbox"/>	0x0000300c	0x00006713	ori x14, x0, 0x00000000	5:	ori a4, zero, 0
<input type="checkbox"/>	0x00003010	0x00f70733	add x14, x14, x15	7:	add a4, a4, a5
<input type="checkbox"/>	0x00003014	0x00d78663	beq x15, x13, 0x00000006	8:	beq a5, a3, finish
<input type="checkbox"/>	0x00003018	0x00c787b3	add x15, x15, x12	9:	add a5, a5, a2
<input type="checkbox"/>	0x0000301c	0xff5ff06f	jal x0, 0xffffffffa	10:	jal zero, loop
<input type="checkbox"/>	0x00003020	0x00e02223	sw x14, 0x00000004(x0)	12:	sw a4, 4(zero)
<input type="checkbox"/>	0x00003024	0x0000006f	jal x0, 0x00000000	14:	jal zero, end

- 在RARS中运行“sum_riscv.asm”

- Run -> Reset

- 设置存储器的第 0x0 单元的内容 (例如=5)

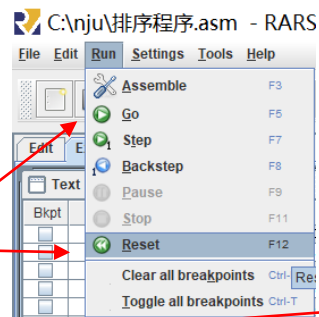
- 在最后一行指令上设置断点

- Run -> Go

- 运行结束后，程序将停在断点处

- 此时可以看到程序运行的结果在第2个存储单元中

- 结果 = f = 15 = 1+2+3+4+5



The screenshot shows the 'Data Segment' window. It displays a table with 'Address' and 'Value (+0)' columns. A red arrow points to the value '0x00000005' at address '0x00000000'.

Address	Value (+0)
0x00000000	0x00000005
0x00000020	0x00000000
0x00000040	0x00000000

The screenshot shows the 'Text Segment' window with the 'Execute' tab selected. A breakpoint is set on the last instruction of the program. The assembly code is displayed in a table.

Bkpt	Address	Code	Basic
<input type="checkbox"/>	0x00003000	0x00002683	lw x13, 0x00000000(x0)
<input type="checkbox"/>	0x00003004	0x00106793	ori x15, x0, 0x00000001
<input type="checkbox"/>	0x00003008	0x00106613	ori x12, x0, 0x00000001
<input type="checkbox"/>	0x0000300c	0x00006713	ori x14, x0, 0x00000000
<input type="checkbox"/>	0x00003010	0x00f70733	add x14, x14, x15
<input type="checkbox"/>	0x00003014	0x00d78663	beq x15, x13, 0x00000006
<input type="checkbox"/>	0x00003018	0x00e787b3	add x15, x15, x12
<input type="checkbox"/>	0x0000301c	0xff5ff06f	jal x0, 0xffffffffa
<input checked="" type="checkbox"/>	0x00003020	0x00e02223	sw x14, 0x00000004(x0)
<input type="checkbox"/>	0x00003024	0x0000006f	jal x0, 0x00000000

The screenshot shows the 'Data Segment' window. It displays a table with 'Address', 'Value (+0)', and 'Value (+4)' columns. The value '0x0000000f' is shown at address '0x00000000'.

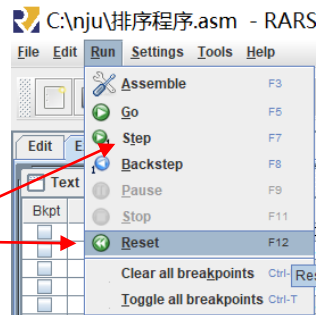
Address	Value (+0)	Value (+4)
0x00000000	0x00000005	0x0000000f
0x00000020	0x00000000	0x00000000

The screenshot shows the 'Text Segment' window with the 'Execute' tab selected. The assembly code is displayed in a table. The last instruction is highlighted.

Bkpt	Address	Code	Basic
<input type="checkbox"/>	0x00003000	0x00002683	lw x13, 0x00000000(x0)
<input type="checkbox"/>	0x00003004	0x00106793	ori x15, x0, 0x00000001
<input type="checkbox"/>	0x00003008	0x00106613	ori x12, x0, 0x00000001
<input type="checkbox"/>	0x0000300c	0x00006713	ori x14, x0, 0x00000000
<input type="checkbox"/>	0x00003010	0x00f70733	add x14, x14, x15
<input type="checkbox"/>	0x00003014	0x00d78663	beq x15, x13, 0x00000006
<input type="checkbox"/>	0x00003018	0x00e787b3	add x15, x15, x12
<input type="checkbox"/>	0x0000301c	0xff5ff06f	jal x0, 0xffffffffa
<input type="checkbox"/>	0x00003020	0x00e02223	sw x14, 0x00000004(x0)
<input checked="" type="checkbox"/>	0x00003024	0x0000006f	jal x0, 0x00000000

• 单步执行程序:

- Run -> Reset
- 设置存储器的第 0x0 单元的内容 (例如=3)
- Run -> Step
- 每按一次Step, 执行一条指令
- 当程序运行到最后一条指令时, 表示程序运行完毕
- 此时, 查看第2个存储单元, 内容为6=1+2+3



Text Segment				
Bkpt	Address	Code	Basic	
<input type="checkbox"/>	0x00003000	0x00002183	lw x3, 0x00000000(x0)	2: lw x3, 0x0(x0)
<input type="checkbox"/>	0x00003004	0x00106293	ori x5, x0, 0x00000001	3: ori x5, x0, 0x1
<input type="checkbox"/>	0x00003008	0x00106113	ori x2, x0, 0x00000001	4: ori x2, x0, 0x1
<input type="checkbox"/>	0x0000300c	0x00520233	add x4, x4, x5	6: add x4, x4, x5
<input type="checkbox"/>	0x00003010	0x00328663	beq x5, x3, 0x00000006	7: beq x5, x3, finish
<input type="checkbox"/>	0x00003014	0x002282b3	add x5, x5, x2	8: add x5, x5, x2
<input type="checkbox"/>	0x00003018	0xff5ff06f	jal x0, 0xffffffffa	9: jal x0, loop
<input type="checkbox"/>	0x0000301c	0x00402223	sw x4, 0x00000004(x0)	11: sw x4, 0x4(x0)
<input checked="" type="checkbox"/>	0x00003020	0x0000006f	jal x0, 0x00000000	13: jal x0, end

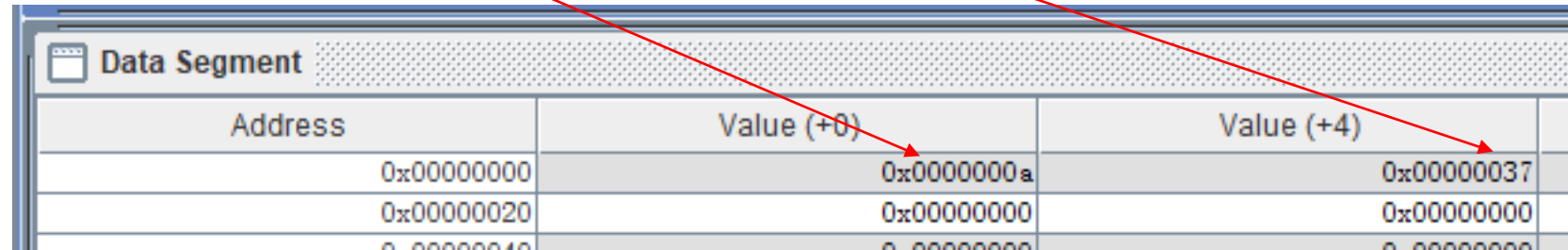
Text Segment				
Bkpt	Address	Code	Basic	
<input type="checkbox"/>	0x00003000	0x00002183	lw x3, 0x00000000(x0)	2: lw x3, 0x0(x0)
<input type="checkbox"/>	0x00003004	0x00106293	ori x5, x0, 0x00000001	3: ori x5, x0, 0x1
<input type="checkbox"/>	0x00003008	0x00106113	ori x2, x0, 0x00000001	4: ori x2, x0, 0x1
<input type="checkbox"/>	0x0000300c	0x00520233	add x4, x4, x5	6: add x4, x4, x5
<input type="checkbox"/>	0x00003010	0x00328663	beq x5, x3, 0x00000006	7: beq x5, x3, finish
<input type="checkbox"/>	0x00003014	0x002282b3	add x5, x5, x2	8: add x5, x5, x2
<input type="checkbox"/>	0x00003018	0xff5ff06f	jal x0, 0xffffffffa	9: jal x0, loop
<input type="checkbox"/>	0x0000301c	0x00402223	sw x4, 0x00000004(x0)	11: sw x4, 0x4(x0)
<input checked="" type="checkbox"/>	0x00003020	0x0000006f	jal x0, 0x00000000	13: jal x0, end

Text Segment				
Bkpt	Address	Code	Basic	
<input type="checkbox"/>	0x00003000	0x00002183	lw x3, 0x00000000(x0)	2: lw x3, 0x0(x0)
<input type="checkbox"/>	0x00003004	0x00106293	ori x5, x0, 0x00000001	3: ori x5, x0, 0x1
<input type="checkbox"/>	0x00003008	0x00106113	ori x2, x0, 0x00000001	4: ori x2, x0, 0x1
<input type="checkbox"/>	0x0000300c	0x00520233	add x4, x4, x5	6: add x4, x4, x5
<input type="checkbox"/>	0x00003010	0x00328663	beq x5, x3, 0x00000006	7: beq x5, x3, finish
<input type="checkbox"/>	0x00003014	0x002282b3	add x5, x5, x2	8: add x5, x5, x2
<input type="checkbox"/>	0x00003018	0xff5ff06f	jal x0, 0xffffffffa	9: jal x0, loop
<input type="checkbox"/>	0x0000301c	0x00402223	sw x4, 0x00000004(x0)	11: sw x4, 0x4(x0)
<input checked="" type="checkbox"/>	0x00003020	0x0000006f	jal x0, 0x00000000	13: jal x0, end

Text Segment				
Bkpt	Address	Code	Basic	
<input type="checkbox"/>	0x00003000	0x00002183	lw x3, 0x00000000(x0)	2: lw x3, 0x0(x0)
<input type="checkbox"/>	0x00003004	0x00106293	ori x5, x0, 0x00000001	3: ori x5, x0, 0x1
<input type="checkbox"/>	0x00003008	0x00106113	ori x2, x0, 0x00000001	4: ori x2, x0, 0x1
<input type="checkbox"/>	0x0000300c	0x00520233	add x4, x4, x5	6: add x4, x4, x5
<input type="checkbox"/>	0x00003010	0x00328663	beq x5, x3, 0x00000006	7: beq x5, x3, finish
<input type="checkbox"/>	0x00003014	0x002282b3	add x5, x5, x2	8: add x5, x5, x2
<input type="checkbox"/>	0x00003018	0xff5ff06f	jal x0, 0xffffffffa	9: jal x0, loop
<input type="checkbox"/>	0x0000301c	0x00402223	sw x4, 0x00000004(x0)	11: sw x4, 0x4(x0)
<input checked="" type="checkbox"/>	0x00003020	0x0000006f	jal x0, 0x00000000	13: jal x0, end

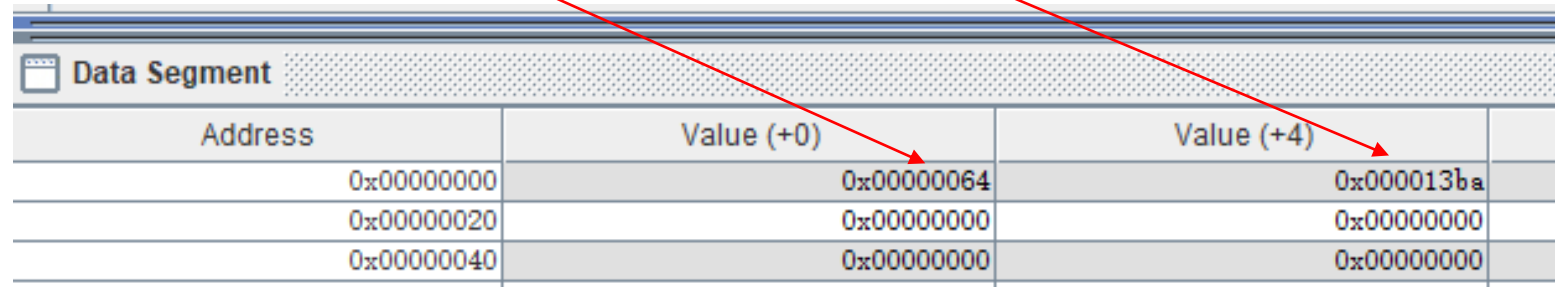
Data Segment			
Address	Value (+0)	Value (+4)	
0x00000000	0x00000003	0x00000006	
0x00000020	0x00000000	0x00000000	

- $n=10=0xa$ 累加和= $0x37=55$



Address	Value (+0)	Value (+4)
0x00000000	0x0000000a	0x00000037
0x00000020	0x00000000	0x00000000
0x00000040	0x00000000	0x00000000

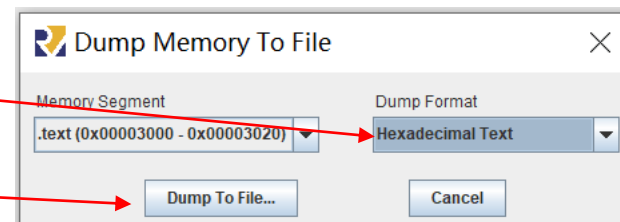
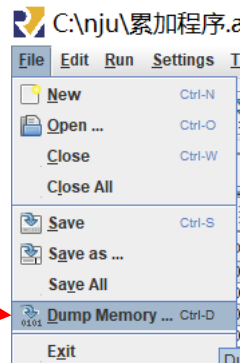
- $n=100=0x64$ 累加和= $0x13ba=5050$



Address	Value (+0)	Value (+4)
0x00000000	0x00000064	0x000013ba
0x00000020	0x00000000	0x00000000
0x00000040	0x00000000	0x00000000

- 将汇编后的机器码，导出到一个文本文件中：

- File -> Dump Memory To File
- 设置对话框中的“Dump Format”为“Hexadecimal Text”
- 点击“Dump To File”
- 输入文件名：sum_riscv.hex，保存

A screenshot of a Notepad window titled 'sum_riscv.hex - 记事本'. The menu bar shows '文件(F)', '编辑(E)', and '格式(O)'. The text content is a list of hexadecimal values:

```
00002683
00106793
00106613
00006713
00f70733
00d78663
00c787b3
ff5ff06f
00e02223
0000006f
```

A red arrow points from the text '输入文件名：sum_riscv.hex，保存' to the Notepad window.

3、MIPS单周期处理器数据通路

- (1) MIPS单周期处理器支持的24指令

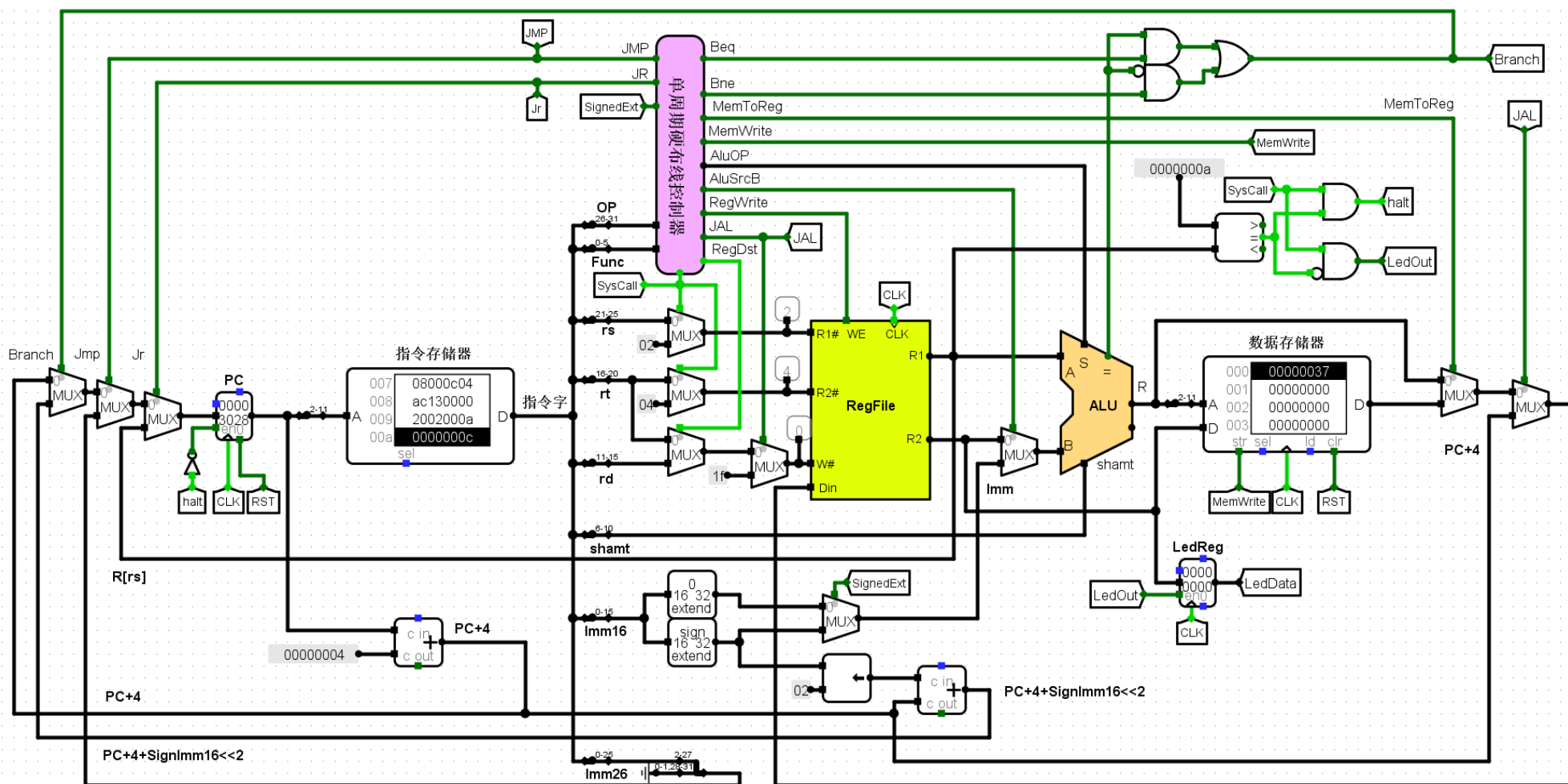
核心指令（8条）

序号	MIPS指令	RTL描述	功能说明
1	add \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] + R[rt]$	寄存器加法指令，不考虑溢出
2	slt rd,rs,rt	$R[rd] \leftarrow (R[rs] < R[rt]) ? 1:0$	小于置1指令，有符号数比较
3	addi rt,rs,imm	$R[rt] \leftarrow R[rs] + \text{SignExt}(imm)$	立即数加法指令，不考虑溢出
4	lw rt,imm(rs)	$R[rt] \leftarrow M[R[rs] + \text{SignExt}(imm)]$	取数指令
5	sw rt,imm(rs)	$M[R[rs] + \text{SignExt}(imm)] \leftarrow R[rt]$	存数指令
6	beq rs,rt,imm	$\text{if}(R[rs] == R[rt]) \quad PC \leftarrow PC + 4 + \text{SignExt}(imm) \ll 2$	条件分支指令：如果rs == rt，则跳转
7	bne rs,rt,imm	$\text{if}(R[rs] != R[rt]) \quad PC \leftarrow PC + 4 + \text{SignExt}(imm) \ll 2$	条件分支指令：如果rs != rt，则跳转
8	syscall	系统调用指令，用于停机	系统调用指令，用于停机

基础指令（16条）

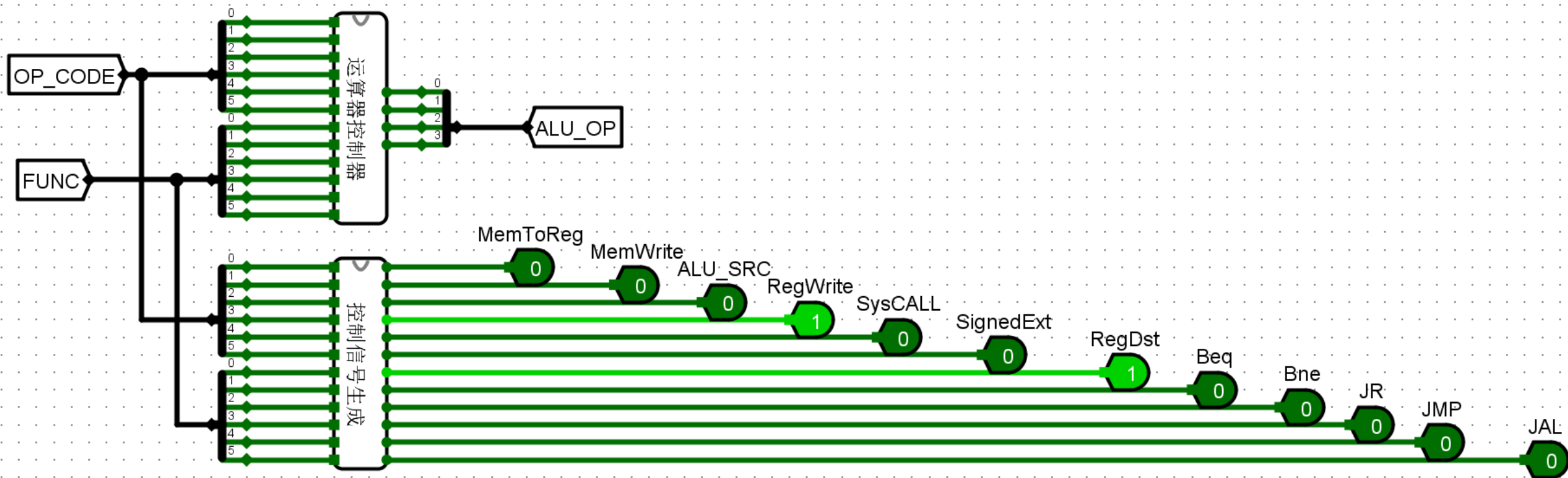
序号	MIPS指令	RTL描述	功能说明
1	addu \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] + R[rt]$	无符号寄存器加法指令
2	addiu \$rt,\$rs,imm	$R[rt] \leftarrow R[rs] + \text{SignExt}(imm)$	无符号立即数加法指令
3	and \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] \& R[rt]$	逻辑与指令
4	andi \$rt,\$rs,imm	$R[rt] \leftarrow R[rs] \& \{16\text{个}0, imm\}$	立即数逻辑与指令
5	sll \$rd,\$rt,imm	$R[rd] \leftarrow R[rt] \ll \text{shamt}$	逻辑左移
6	srl \$rd,\$rt,imm	$R[rd] \leftarrow R[rt] \gg \text{shamt}$	逻辑右移
7	sra \$rd,\$rt,imm	$R[rd] \leftarrow R[rt] \gg \text{shamt}$	算术右移
8	sub \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] - R[rt]$	寄存器减法指令
9	or \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] R[rt]$	逻辑或指令
10	ori \$rt,\$rs,imm	$R[rt] \leftarrow R[rs] \{16\text{个}0, imm\}$	立即数逻辑或指令
11	nor \$rd,\$rs,\$rt	$R[rd] \leftarrow \neg(R[rs] R[rt])$	逻辑或非指令
12	sltu \$rd,\$rs,\$rt	$R[rd] \leftarrow (R[rs] < R[rt]) ? 1:0$	小于置1指令，无符号数比较
13	slti \$rt,\$rs,imm	$R[rd] \leftarrow (R[rs] < \text{SignExt}(imm)) ? 1:0$	小于置1指令，立即数比较
14	j address	$PC \leftarrow \{(PC+4)_{31:28}, address<<2\}$	无条件分支指令
15	jal address	$R[31] \leftarrow PC+4 \quad PC \leftarrow \{(PC+4)_{31:28}, address<<2\}$	子程序调用指令
16	jr \$rs	$PC \leftarrow R[rs]$	寄存器跳转指令

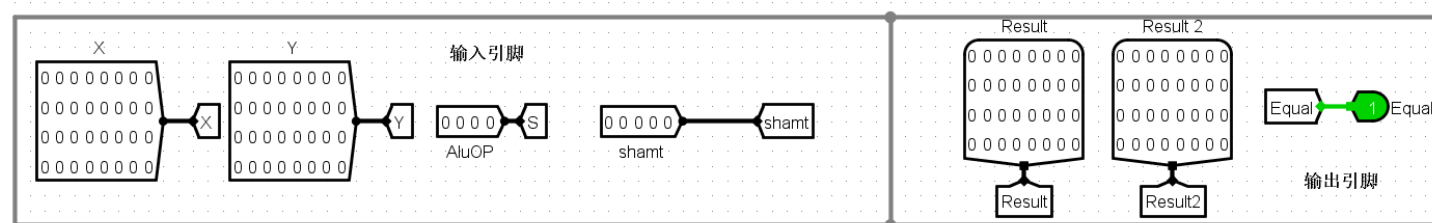
- (2) MIPS单周期处理器的数据通路



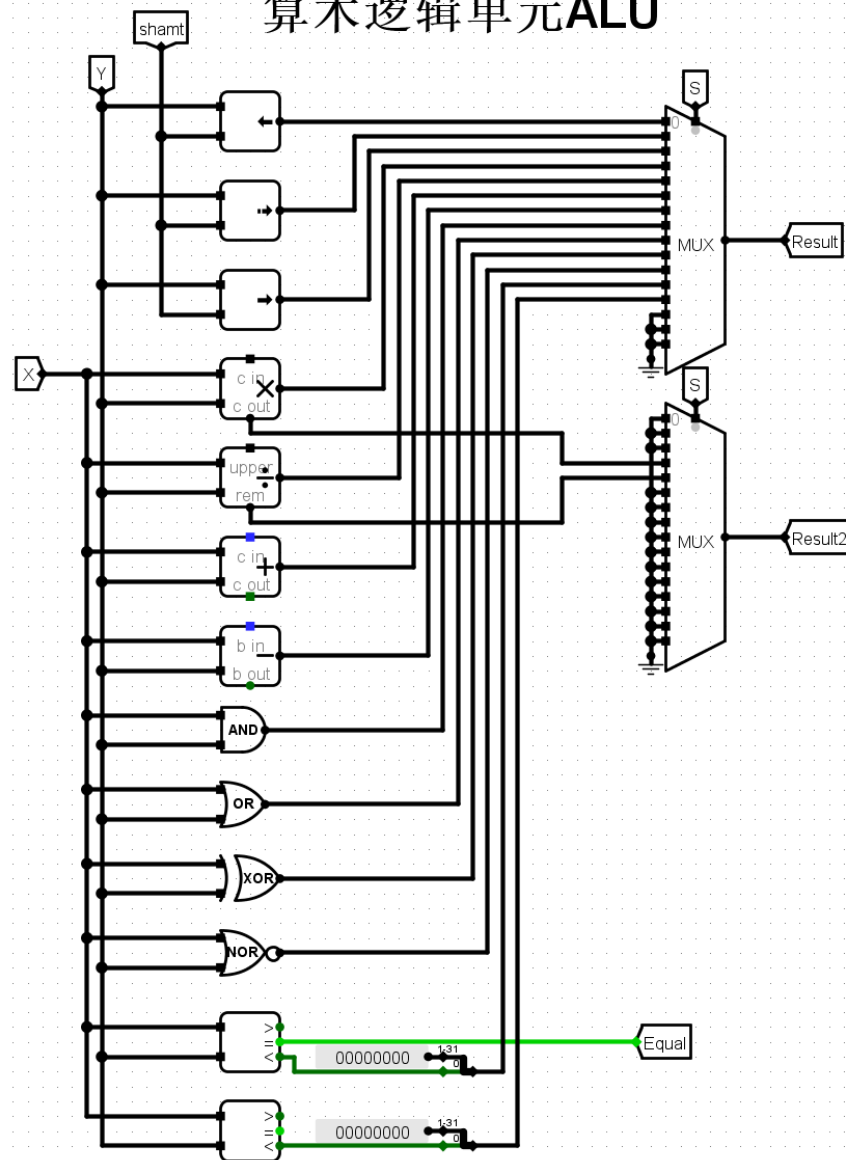
{PC+4的高4位, imm26<<2} 拼接操作 假设PC+4的高4位=0000

单周期硬布线控制器

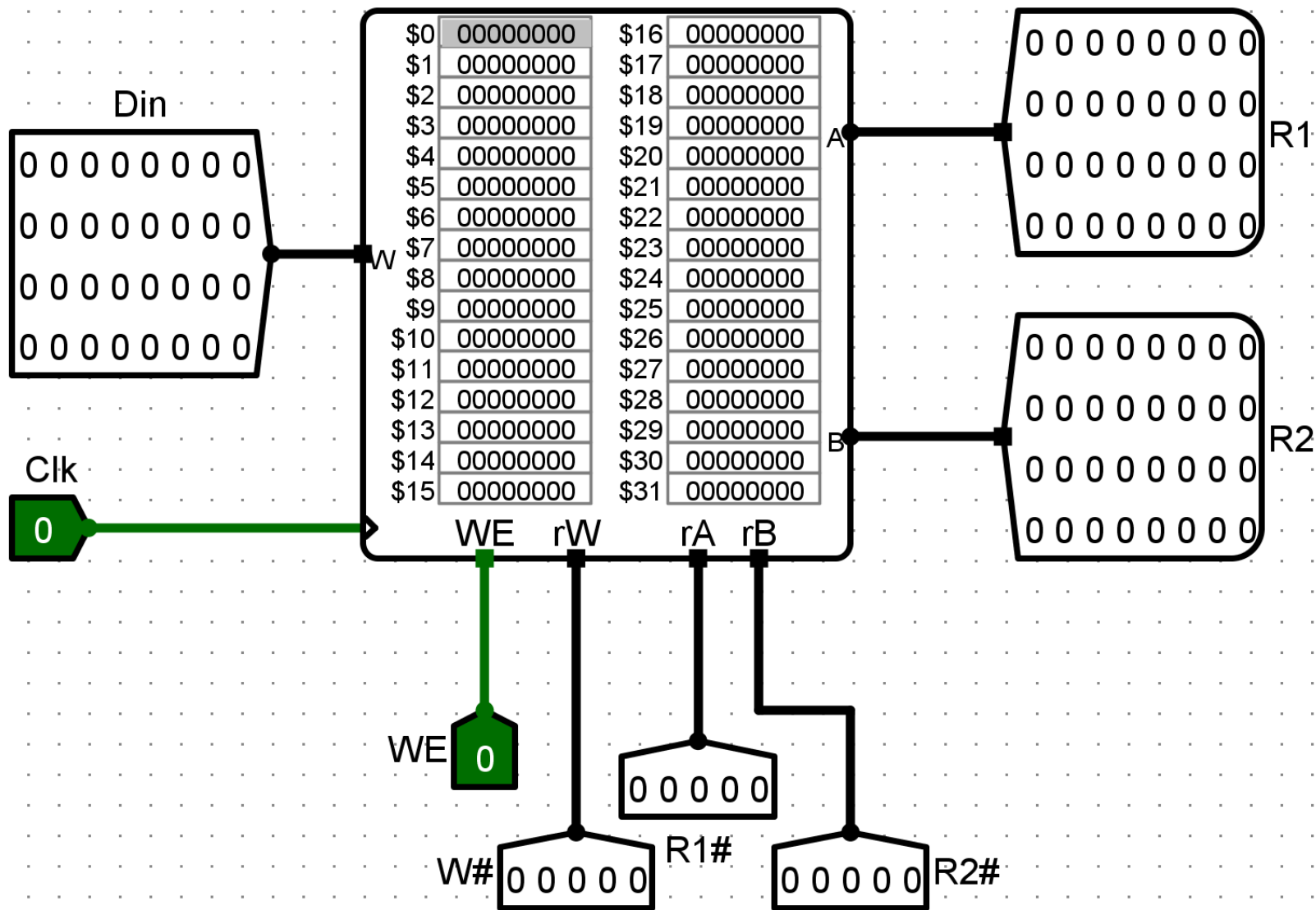




算术逻辑单元ALU



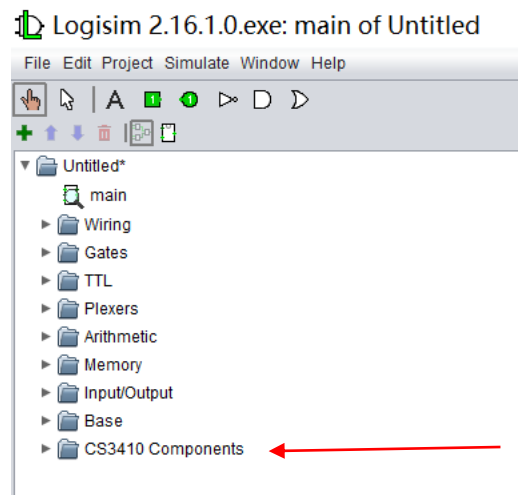
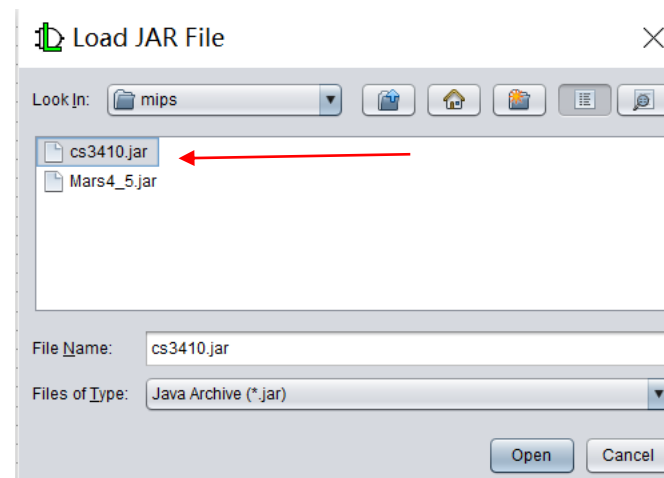
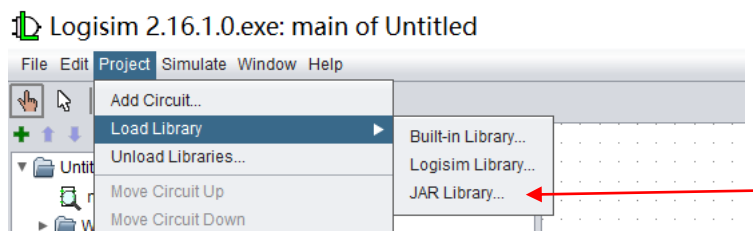
寄存器堆 Regfile

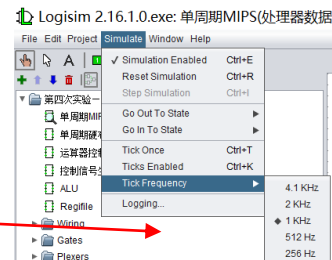


• (3) 在单周期MIPS处理器的数据通路上运行程序

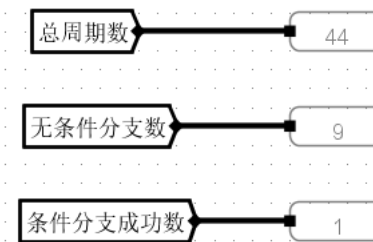
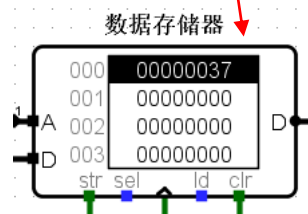
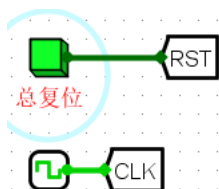
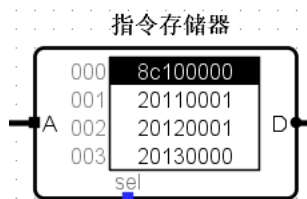
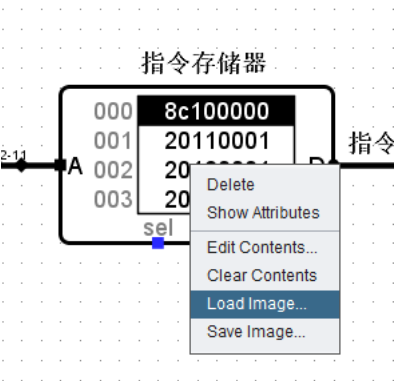
- 第一步：修改前面导出的机器码文件sum_mips.hex，增加：v2.0 raw
- 第二步：打开Logisim，在Logisim中增加寄存器文件库“cs3410.jar”
- 第三步：打开设计文件“第四次实验——CPU设计实验（单周期MIPS处理器）.circ”

sum.hex - 记事本
文件(F) 编辑(E) 格式
v2.0 raw
8c100000
20110001
20120001
20130000
02719820
12300002
02328820
08000c04
ac130004
08000c09





- 第四步：在设计文件的指令存储器中装入机器码文件sum_mips.hex
- 第五步：设置时钟频率=1KHz，按Ctrl+K启动时钟（Ticks Enabled），程序开始执行
- 第六步：按总复位，程序重新执行
- 第七步：等程序执行完毕，观察数据存储器第0号单元的值（累加和，正确值为37h=55），以及程序运行的总周期数、无条件分支数、条件分支成功数
- 第八步：修改sum_mips.asm程序，计算 $1+2+\dots+100=5050$ ，并在单周期MIPS处理器数据通路上运行，观察结果是不是13bah (5050) ？



- (4) 要求:

- ①分析单周期MIPS处理器数据通路**电路原理**:

- 与教材图6.25相比, 该单周期MIPS处理器数据通路**增加了哪些部件**? 为什么要增加这些部件?
- 与教材表6.9相比, 该单周期MIPS处理器数据通路的控制器**增加了哪几个控制信号**? 为什么要增加这些控制信号?

- ②累加和程序 ($1+2+\cdots+10$) 运行后, 总周期数、无条件分支数、条件分支成功数分别是44、9、1, 请**分析为什么是这个值**?

- ③累加和程序 ($1+2+\cdots+100$) 运行后, 总周期数、无条件分支数、条件分支成功数分别是多少? 并**分析为什么是这个值**?

二、设计实验

1、MIPS汇编语言程序设计实验

- (1) 请使用单周期MIPS处理器的24条指令，编写计算**费波那契数列**的程序
 - 斐波那契数列（Fibonacci sequence），又称黄金分割数列，指的是这样一个数列：0、1、1、2、3、5、8、13、21、34、……；在数学上，斐波那契数列通常以递推的方法定义： $F(0)=0$ ， $F(1)=1$ ， $F(n)=F(n-1)+F(n-2)$ ($n \geq 2$ ， $n \in N^*$)。
 - 现要计算n+1个费波那契数列（n=3时，费波那契数列为：0、1、1、2）；n在程序中设置（可以修改），计算好的费波那契数列存放在数据存储器地址为0、4、8、…、4*n的存储单元中。
 - 要求，编写好的程序先在**MIPS汇编器上运行通过**；然后再到**单周期MIPS处理器数据通路上运行通过**，观察程序运行的总周期数、无条件分支数、条件分支成功数分别是多少，并进行分析。

核心指令（8条）

序号	MIPS指令	RTL描述	功能说明
1	add \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] + R[rt]$	寄存器加法指令，不考虑溢出
2	slt \$rd,\$rs,\$rt	$R[rd] \leftarrow (R[rs] < R[rt]) ? 1:0$	小于置1指令，有符号数比较
3	addi \$rt,\$rs,imm	$R[rt] \leftarrow R[rs] + \text{SignExt}(imm)$	立即数加法指令，不考虑溢出
4	lw \$rt,imm(\$rs)	$R[rt] \leftarrow M[R[rs] + \text{SignExt}(imm)]$	取数指令
5	sw \$rt,imm(\$rs)	$M[R[rs] + \text{SignExt}(imm)] \leftarrow R[rt]$	存数指令
6	beq \$rs,\$rt,imm	if($R[rs] == R[rt]$) PC $\leftarrow PC + 4 + \text{SignExt}(imm) << 2$	条件分支指令：如果rs == rt，则跳转
7	bne \$rs,\$rt,imm	if($R[rs] != R[rt]$) PC $\leftarrow PC + 4 + \text{SignExt}(imm) << 2$	条件分支指令：如果rs != rt，则跳转
8	syscall	系统调用指令，用于停机	系统调用指令，用于停机

基础指令（16条）

序号	MIPS指令	RTL描述	功能说明
1	addu \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] + R[rt]$	无符号寄存器加法指令
2	addiu \$rt,\$rs,imm	$R[rt] \leftarrow R[rs] + \text{SignExt}(imm)$	无符号立即数加法指令
3	and \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] \& R[rt]$	逻辑与指令
4	andi \$rt,\$rs,imm	$R[rt] \leftarrow R[rs] \& (16\text{个}0, imm)$	立即数逻辑与指令
5	sll \$rd,\$rt,imm	$R[rd] \leftarrow R[rt] << \text{shamt}$	逻辑左移
6	srl \$rd,\$rt,imm	$R[rd] \leftarrow R[rt] >> \text{shamt}$	逻辑右移
7	sra \$rd,\$rt,imm	$R[rd] \leftarrow R[rt] >> \text{shamt}$	逻辑左移
8	sub \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] - R[rt]$	寄存器减法指令
9	or \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] R[rt]$	逻辑或指令
10	ori \$rt,\$rs,imm	$R[rt] \leftarrow (R[rs] (16\text{个}0, imm))$	立即数逻辑或指令
11	nor \$rd,\$rs,\$rt	$R[rd] \leftarrow \neg(R[rs] R[rt])$	逻辑或非指令
12	sltu \$rd,\$rs,\$rt	$R[rd] \leftarrow (R[rs] < R[rt]) ? 1:0$	小于置1指令，无符号数比较
13	slti \$rt,\$rs,imm	$R[rt] \leftarrow (R[rs] < \text{SignExt}(imm)) ? 1:0$	小于置1指令，立即数比较
14	j address	PC $\leftarrow \{(PC+4)_{20,28}, address << 2\}$	无条件分支指令
15	jal address	$R[31] \leftarrow PC+4$ PC $\leftarrow \{(PC+4)_{20,28}, address << 2\}$	子程序调用指令
16	jr \$rs	PC $\leftarrow R[rs]$	寄存器跳转指令

- (2) 请使用单周期MIPS处理器的24条指令，编写**冒泡排序算法**的程序

- 假设有10个数据，这10个数据的顺序是随机的，例如：9、1、2、5、10、8、4、7、6、3；这10个数据是通过指令存放在数据存储器的第0、4、8、…、36号地址的存储单元中。
- 要求对这10个数据按照**从小到大**的顺序进行排序，排序后的数据仍然放在数据存储器的第0、4、8、…、36号地址的存储单元中。
- 要求对这10个数据按照**从大到小**的顺序进行排序，排序后的数据仍然放在数据存储器的第0、4、8、…、36号地址的存储单元中。
- 要求，编写好的2个程序先在**MIPS汇编器上运行通过**；然后再到**单周期MIPS处理器数据通路上运行通过**，观察程序运行的总周期数、无条件分支数、条件分支成功数分别是多少，并进行分析。

核心指令（8条）

序号	MIPS指令	RTL描述	功能说明
1	add \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] + R[rt]$	寄存器加法指令，不考虑溢出
2	slt \$rd,\$rs,\$rt	$R[rd] \leftarrow (R[rs] < R[rt]) ? 1:0$	小于置1指令，有符号数比较
3	addi \$rt,\$rs,imm	$R[rt] \leftarrow R[rs] + \text{SignExt}(imm)$	立即数加法指令，不考虑溢出
4	lw \$rt,imm(\$rs)	$R[rt] \leftarrow M[R[rs] + \text{SignExt}(imm)]$	取数指令
5	sw \$rs,imm(\$rs)	$M[R[rs] + \text{SignExt}(imm)] \leftarrow R[rs]$	存数指令
6	beq \$rs,\$rt,imm	if($R[rs] == R[rt]$) PC \leftarrow PC + 4 + $\text{SignExt}(imm) < 2$	条件分支指令：如果rs == rt，则跳转
7	bne \$rs,\$rt,imm	if($R[rs] != R[rt]$) PC \leftarrow PC + 4 + $\text{SignExt}(imm) < 2$	条件分支指令：如果rs != rt，则跳转
8	syscall	系统调用指令，用于停机	系统调用指令，用于停机

基础指令（16条）

序号	MIPS指令	RTL描述	功能说明
1	addu \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] + R[rt]$	无符号寄存器加法指令
2	addiu \$rt,\$rs,imm	$R[rt] \leftarrow R[rs] + \text{SignExt}(imm)$	无符号立即数加法指令
3	and \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] \& R[rt]$	逻辑与指令
4	andi \$rt,\$rs,imm	$R[rt] \leftarrow R[rs] \& (16\text{个}0, imm)$	立即数逻辑与指令
5	sll \$rd,\$rs,imm	$R[rd] \leftarrow R[rs] \ll \text{shamt}$	逻辑左移
6	srl \$rd,\$rs,imm	$R[rd] \leftarrow R[rs] \gg \text{shamt}$	逻辑右移
7	sra \$rd,\$rs,imm	$R[rd] \leftarrow R[rs] \gg \text{shamt}$	逻辑左移
8	sub \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] - R[rt]$	寄存器减法指令
9	or \$rd,\$rs,\$rt	$R[rd] \leftarrow R[rs] R[rt]$	逻辑或指令
10	ori \$rt,\$rs,imm	$R[rt] \leftarrow R[rs] (16\text{个}0, imm)$	立即数逻辑或指令
11	nor \$rd,\$rs,\$rt	$R[rd] \leftarrow \neg(R[rs] R[rt])$	逻辑或非指令
12	sltu \$rd,\$rs,\$rt	$R[rd] \leftarrow (R[rs] < R[rt]) ? 1:0$	小于置1指令，无符号数比较
13	slti \$rt,\$rs,imm	$R[rd] \leftarrow (R[rs] < \text{SignExt}(imm)) ? 1:0$	小于置1指令，立即数比较
14	j address	PC \leftarrow $\{(PC+4)_{24}, address < 2\}$	无条件分支指令
15	jal address	$R[31] \leftarrow PC+4$ PC \leftarrow $\{(PC+4)_{24}, address < 2\}$	子程序调用指令
16	jr \$rs	PC $\leftarrow R[rs]$	寄存器跳转指令

2、RISC-V汇编语言程序设计实验

- (1) 请使用RISC-V处理器的9条指令，编写计算**费波那契数列**的程序
 - 斐波那契数列 (Fibonacci sequence)，又称黄金分割数列，指的是这样一个数列：0、1、1、2、3、5、8、13、21、34、……，在数学上，斐波那契数列通常以递推的方法定义： $F(0)=0$, $F(1)=1$, $F(n)=F(n-1)+F(n-2)$ ($n \geq 2$, $n \in \mathbb{N}^*$)。
 - 现要计算 $n+1$ 个费波那契数列 ($n=3$ 时，费波那契数列为：0、1、1、2)， n 存放在数据存储器地址为0的单元中（可以修改），计算好的费波那契数列存放在数据存储器地址为4、8、…、 $4*(n+1)$ 的存储单元中。
 - 要求，编写好的程序在RISC-V汇编器上运行通过，并保存该程序的汇编机器码到文件中（下次实验用）。

• RISC-V核心指令集RV32I的9条指令（我们设计的RISC-V单周期处理器仅支持该9条指令）：

- | | |
|---------------------|--|
| ① add rd,rs1,rs2 | ; rs1+rs2->rd |
| ② slt rd,rs1,rs2 | ; 带符号数比较指令 if rs1<rs2 1->rd else 0->rd |
| ③ sltu rd,rs1,rs2 | ; 无符号数比较指令 if rs1<rs2 1->rd else 0->rd |
| ④ ori rd,rs1,imm12 | ; rs1 或 imm12 -> rd |
| ⑤ lw rd rs1,imm12 | ; M[rs1+imm12] -> rd |
| ⑥ lui rd,imm20 | ; imm20 -> rd |
| ⑦ sw rs2,rs1,imm12 | ; rs2 -> M[rs1+imm12] |
| ⑧ beq rs1,rs2,imm12 | ; if rs1=rs2 then goto imm12 |
| ⑨ jal rd,imm20 | ; goto imm20 |

- (2) 请使用RISC-V处理器的9条指令，编写**冒泡排序算法**的程序

- 假设有n个数据，这n个数据的顺序是随机的，例如10个数据（n=10）：9、1、2、5、10、8、4、7、6、3；n存放在数据存储器地址为0的单元中，n个原始数据存放在数据存储器的第4、8、…、4*n号地址的存储单元中。
- 要求对这10个数据按照**从小到大**的顺序进行排序，排序后的数据仍然放在数据存储器的第4、8、…、4*n号地址的存储单元中。
- 要求对这10个数据按照**从大到小**的顺序进行排序，排序后的数据仍然放在数据存储器的第4、8、…、4*n号地址的存储单元中。
- 要求，编写好的程序在RISC-V汇编器上运行通过，并保存该程序的汇编机器码到文件中（下次实验用）。

• RISC-V核心指令集RV32I的9条指令（我们设计的RISC-V单周期处理器仅支持该9条指令）：

- ① add rd,rs1,rs2 ; rs1+rs2->rd
- ② slt rd,rs1,rs2 ; 带符号数比较指令 if rs1<rs2 1->rd else 0->rd
- ③ sltu rd,rs1,rs2 ; 无符号数比较指令 if rs1<rs2 1->rd else 0->rd
- ④ ori rd,rs1,imm12 ; rs1 或 imm12 -> rd
- ⑤ lw rd,rs1,imm12 ; M[rs1+imm12] -> rd
- ⑥ lui rd,imm20 ; imm20 -> rd
- ⑦ sw rs2,rs1,imm12 ; rs2 -> M[rs1+imm12]
- ⑧ beq rs1,rs2,imm12 ; if rs1=rs2 then goto imm12
- ⑨ jal rd,imm20 ; goto imm20

3、Intel x86汇编语言程序设计实验

- 请使用32位的Intel x86的指令，编写计算**冒泡排序算法**的程序（从小到大排序、从大到小排序）；并在32位的Intel x86汇编语言环境下运行通过。

Thanks