

Lab 3 汇编程序设计

姓名:傅申 学号: PB20000051 实验日期: 2022-4-6

1 实验题目

汇编程序设计

2 实验目的

- 熟悉 RISC-V 汇编指令的格式
- 熟悉 CPU 仿真软件 Ripes, 理解汇编指令执行的基本原理 (数据通路和控制器的协调工作过程)
- 熟悉汇编程序的基本结构, 掌握简单汇编程序的设计
- 掌握汇编仿真软件 RARS (RISC-V Assembler & Runtime Simulator) 的使用方法, 会用该软件进行汇编程序的仿真, 调试以及生成 CPU 测试需要的指令和数据文件 (COE)
- 理解 CPU 调试模块 PDU 的使用方法

3 实验平台

- Windows 11 PC + OpenJDK 17.0.2
- Ripes: RISC-V graphical processor simulator, v2.2.4
- RARS: RISC-V Assembler and Runtime Simulator, v1.5

4 实验内容

4.1 理解并仿真 Ripes 示例汇编程序

consolePrinting.s 的输出结果为

```
A string
-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
3.14159
!, ", #, $, %, &, ', (, ), *, +, ,, -, ., /, 0, 1, 2, 3, 4, 5,
Program exited with code: 0
```

4.2 设计汇编程序, 验证 6 条指令功能

设计的汇编程序如下:

```

1  .data
2  array: .word  0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08
3
4  .text
5      la      a0, array
6      lw      a1, 4(a0)
7      addi    a0, a0, 16
8      lw      a2, 0(a0)
9      add     a2, a2, a1
10     sw      a2, 0(a0)
11     addi    a0, a0, -16
12     lw      a1, 0(a0)
13 branch:
14     beq     a1, x0, case1
15     jal     case2
16 case1:
17     addi    a1, a1, 1
18     jal     x0, branch
19 case2:
20     addi    a7, x0, 10
21     ecall

```

执行完成后的内存情况:

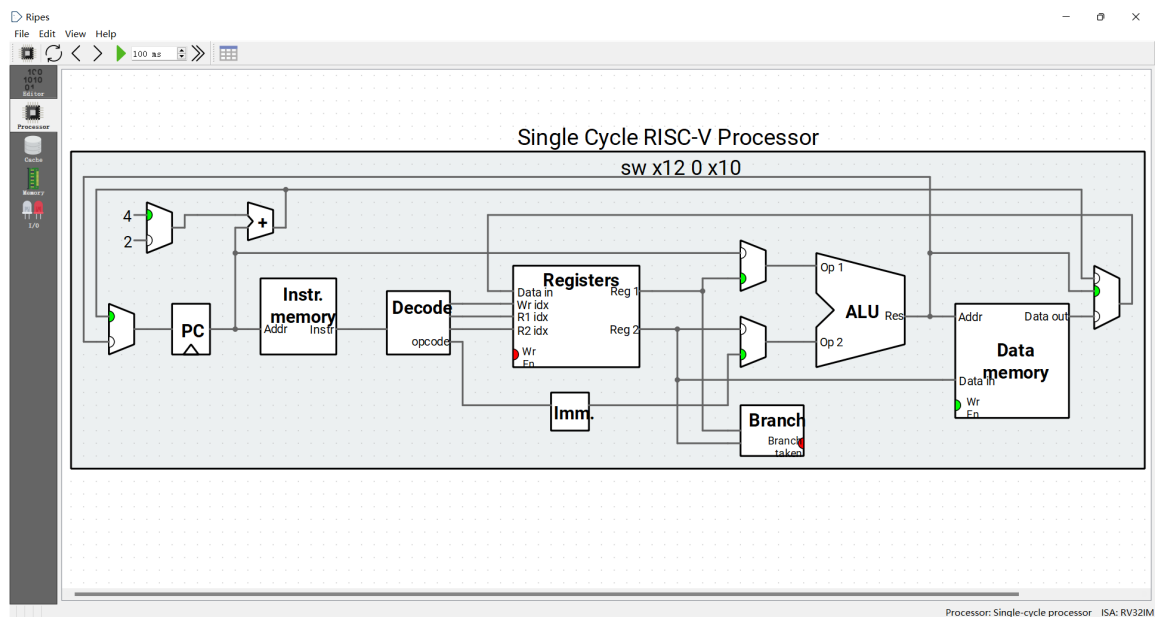
Address	Word	Byte 0	Byte 1	Byte 2	Byte 3
0x10000020	0x00000008	0x08	0x00	0x00	0x00
0x1000001c	0x00000007	0x07	0x00	0x00	0x00
0x10000018	0x00000006	0x06	0x00	0x00	0x00
0x10000014	0x00000005	0x05	0x00	0x00	0x00
0x10000010	0x00000005	0x05	0x00	0x00	0x00
0x1000000c	0x00000003	0x03	0x00	0x00	0x00
0x10000008	0x00000002	0x02	0x00	0x00	0x00
0x10000004	0x00000001	0x01	0x00	0x00	0x00
0x10000000	0x00000000	0x00	0x00	0x00	0x00
0x0fffffc	X	X	X	X	X
0x0fffffb	X	X	X	X	X
0x0fffff4	X	X	X	X	X
0x0fffff0	X	X	X	X	X
0x0ffffec	X	X	X	X	X
0x0ffffe8	X	X	X	X	X
0x0ffffe4	X	X	X	X	X
0x0ffffe0	X	X	X	X	X
0x0ffffdc	X	X	X	X	X
0x0ffffd8	X	X	X	X	X
0x0ffffd4	X	X	X	X	X
0x0ffffd0	X	X	X	X	X
0x0ffffcc	X	X	X	X	X

Name	Size	Range
.text	60	0x00000000 - 0x0000003c
.data	36	0x10000000 - 0x10000024
.bss	0	0x11000000 - 0x11000000

Display type: Hex Go to register: Go to section: Processor: Single-cycle processor ISA: RV32IM

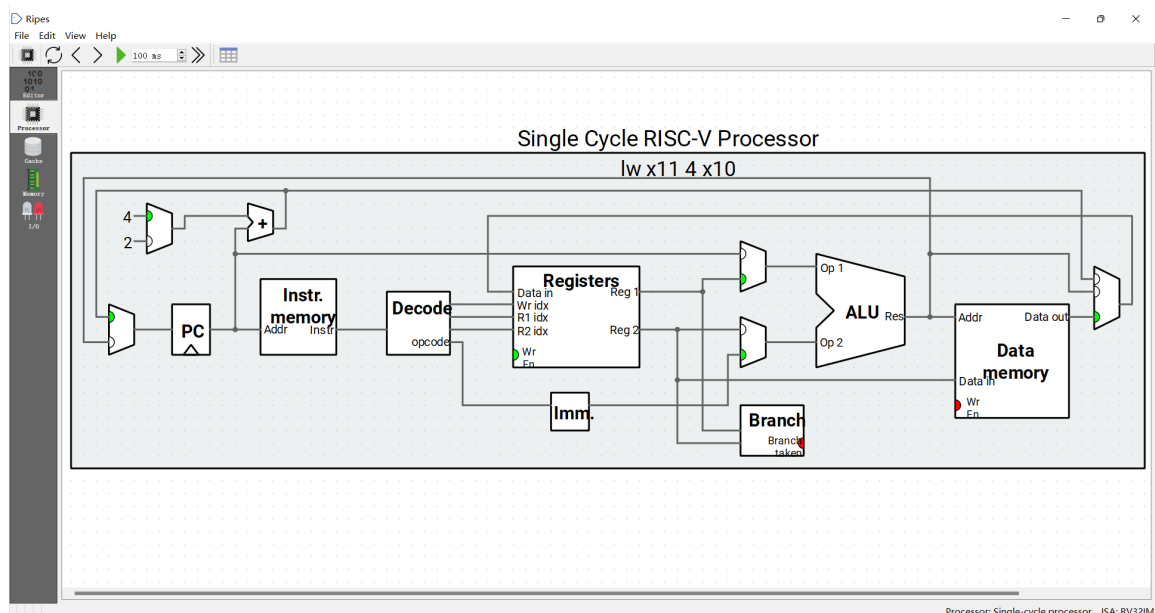
下面简要说明 `sw`, `lw`, `add`, `addi`, `beq`, `jal` 的执行过程.

- `sw rs2, offset(rs1)`



1. 指令从指令存储器取出后经译码器得到 $rs1$ 和 $rs2$ 的下标, 分别传入寄存器堆的 $R1\ idx$ 和 $R2\ idx$ 的端口, 输出两个寄存器中的值.
2. 指令中的 $offset$ 由译码器传入的信号通过立即数生成单元生成.
3. ALU 前的两个 Mux 分别选择 $rs1$ 和 $offset$ 的值, 传入 ALU 后做加法运算得到地址.
4. 数据存储器的写使能打开, $rs2$ 的值被写入到地址所指向的内存位置.
5. 寄存器堆的写使能关闭. PC 前的 Mux 选择 $PC + 4$ 的值.

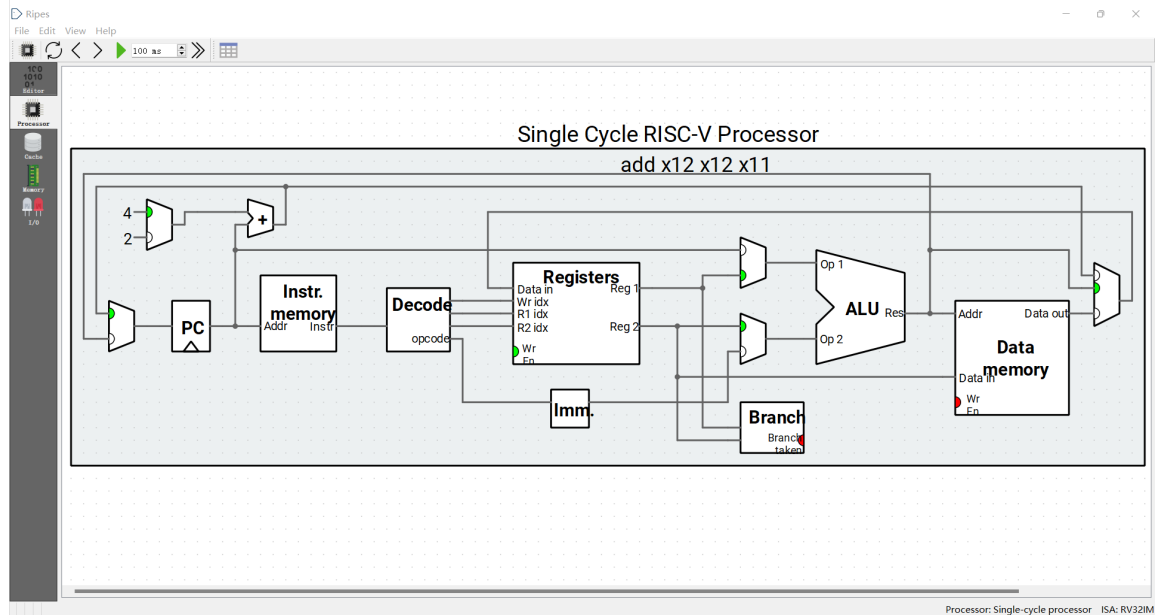
- **lw rd, offset(rs1)**



1. 指令从指令存储器取出后经译码器得到 $rs1$ 和 rd 的下标, 分别传入寄存器堆的 $R1\ idx$ 和 $Wr\ idx$ 端口. 寄存器堆输出 $rs1$ 的值.
2. 指令中的 $offset$ 由译码器传出的信号通过立即数生成单元生成.
3. ALU 前的两个 Mux 分别选择 $rs1$ 和 $offset$ 的值, 传入 ALU 后做加法运算得到地址.

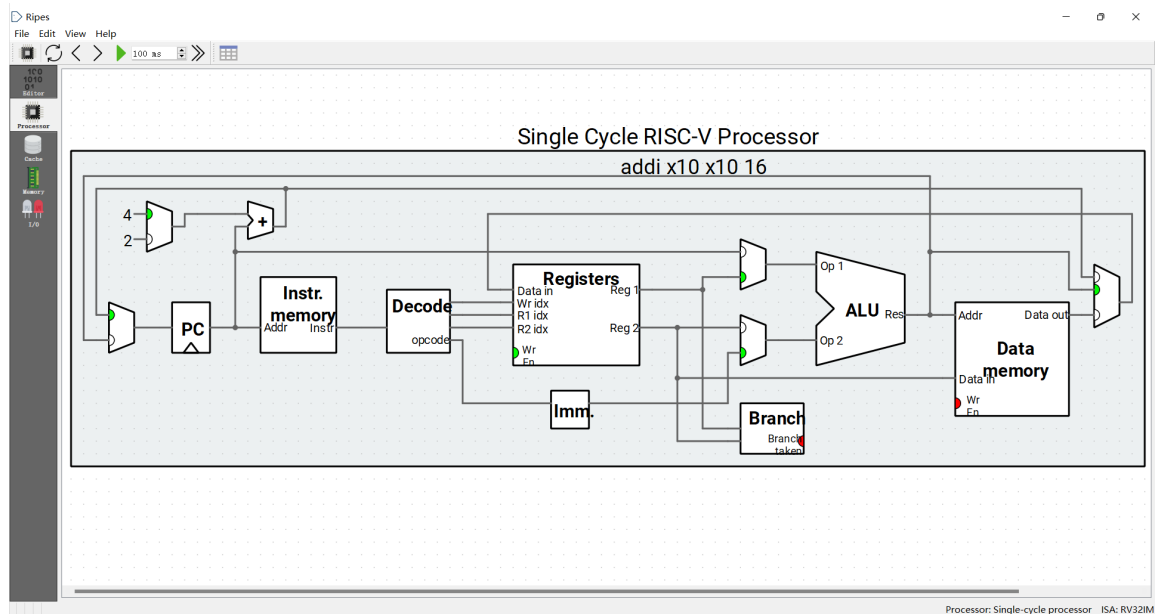
4. 数据存储器的写使能关闭, 取出地址所指向的内存位置的值, 由后面的 Mux 选择并传入到寄存器堆的 Data in 端口.
5. 寄存器堆的写使能打开, 数据被存到 rd 中.
6. PC 前的 Mux 选择 $PC + 4$ 的值.

- add rd, rs1, rs2



1. 指令从指令存储器取出后经译码器得到 rs1, rs2 和 rd 的下标, 分别传入寄存器堆的 R1 idx, R2 idx 和 Wr idx 端口, 寄存器堆输出 rs1 和 rs2 的值.
2. ALU 前的两个 Mux 分别选择 rs1 和 rs2 的值, 传入 ALU 后做加法运算得到结果并被后面的 Mux 选择.
3. 寄存器堆的写使能打开, 结果被存到 rd 中.
4. 数据存储器的写使能关闭. PC 前的 Mux 选择 $PC + 4$ 的值.

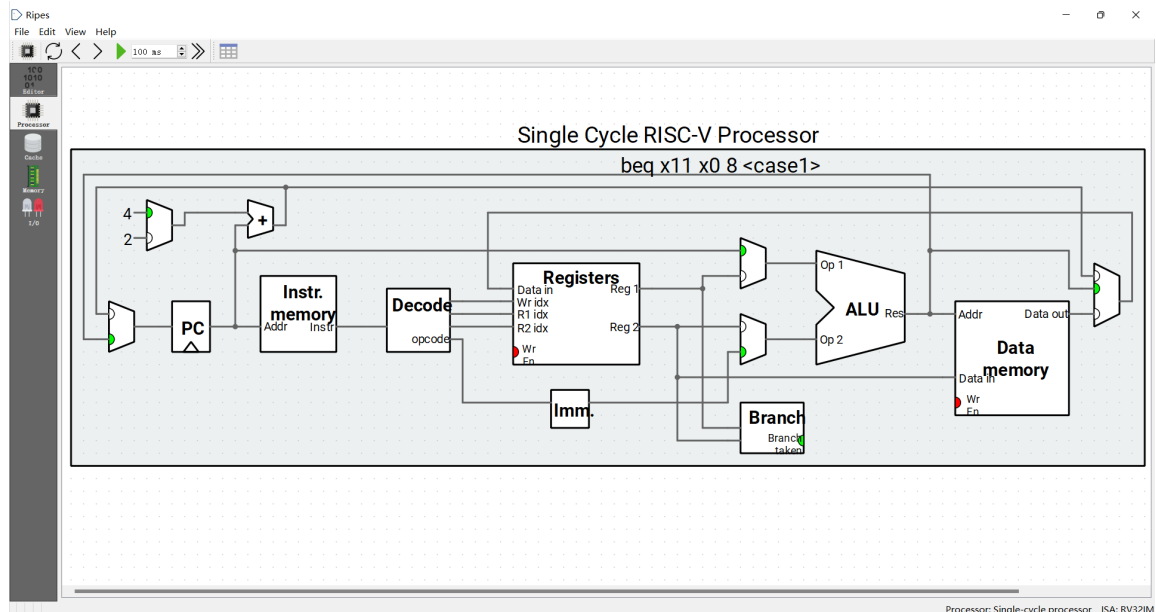
- addi rd, rs1, imm



1. 指令从指令存储器取出后经译码器得到 rs1 和 rd 的下标, 分别传入寄存器堆的 R1 idx 和 Wr idx 端口. 寄存器堆输出 rs1 的值.

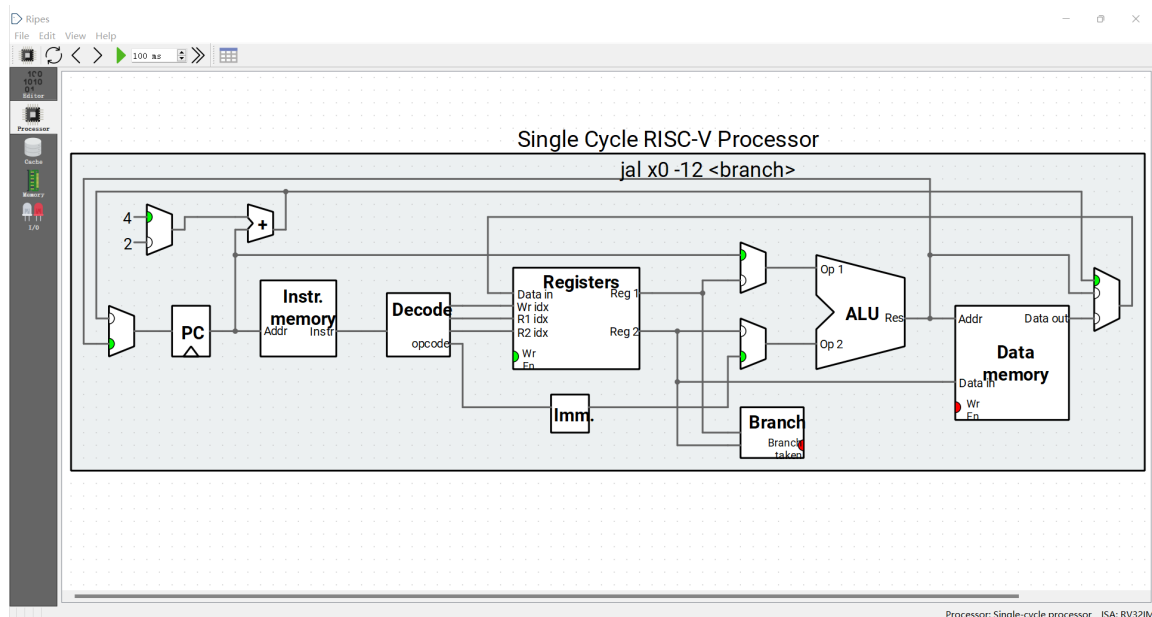
- 指令中的 `imm` 由译码器传出的信号通过立即数生成单元生成.
- ALU 前的两个 Mux 分别选择 `rs1` 和 `imm` 的值, 传入 ALU 后做加法运算得到结果并被后面的 Mux 选择.
- 寄存器堆的写使能打开, 结果被存到 `rd` 中.
- 数据存储器的写使能关闭. PC 前的 Mux 选择 `PC + 4` 的值.

- `beq rs1, rs2, offset`



- 指令从指令存储器取出后经译码器得到 `rs1` 和 `rs2` 的下标, 分别传入寄存器堆的 `R1 idx` 和 `R2 idx` 端口. 寄存器堆输出 `rs1` 和 `rs2` 的值.
- 指令中的 `offset` 由译码器传出的信号通过立即数生成单元生成.
- ALU 前的两个 Mux 分别选择 PC 和 `offset` 的值, 传入 ALU 后做加法运算得到下一条指令 (可能) 的地址, 它可能被 PC 前的 Mux 选择.
- Branch 单元比较 `rs1` 和 `rs2` 的值, 如果相等, 则 PC 前的 Mux 选择计算出的地址, 否则选择 `PC + 4`.
- 寄存器堆和数据存储器的写使能关闭.

- `jal rd, offset`



1. 指令从指令存储器取出后经译码器得到 rd 的下标, 传入寄存器堆的 Wr idx 端口.
2. 数据存储器后面的 Mux 选择当前的 PC 值, 传入寄存器堆的 Data in 端口. 寄存器堆的写使能打开, 当前 PC 被存到 rd 中.
3. 指令中的 offset 由译码器传出的信号通过立即数生成单元生成.
4. ALU 前的两个 Mux 分别选择 PC 和 offset 的值, 传入 ALU 后做加法运算得到下一条指令的地址.
5. PC 前的 Mux 选择计算出的地址.
6. 数据存储器的写使能关闭.

4.3 设计汇编程序, 计算 FLS

设计的汇编程序如下:

```

1  .data
2  newline:
3      .string "\n"
4  array:
5      .word    0x0001 0x0002
6
7  .text
8  load: # load the arguments
9      la      a0, array    # a0 <- pointer to array
10     addi    a1, x0, 10    # a1 <- number of elements
11     lw      a2, 0(a0)
12     lw      a3, 4(a0)
13     addi    a0, a0, 8
14     addi    a1, a1, -2
15
16     # print first two elements
17     addi    a4, a2, 0
18     jal     x1, print
19     addi    a4, a3, 0
20     jal     x1, print
21

```

```

22  loop: # generate, store and print the array
23      bge      x0, a1, exit
24      add      a4, a2, a3
25      sw       a4, 0(a0)
26      jal      x1, print
27      addi     a0, a0, 4
28      addi     a2, a3, 0
29      addi     a3, a4, 0
30      addi     a1, a1, -1
31      jal      x0, loop
32
33  print:
34      addi     t0, a0, 0
35      addi     a0, a4, 0
36      addi     a7, x0, 1
37      ecall
38      la       a0, newline
39      addi     a7, x0, 4
40      ecall
41      addi     a0, t0, 0
42      jalr     x1
43
44  exit: # quit program
45      addi     a7, x0, 10
46      ecall

```

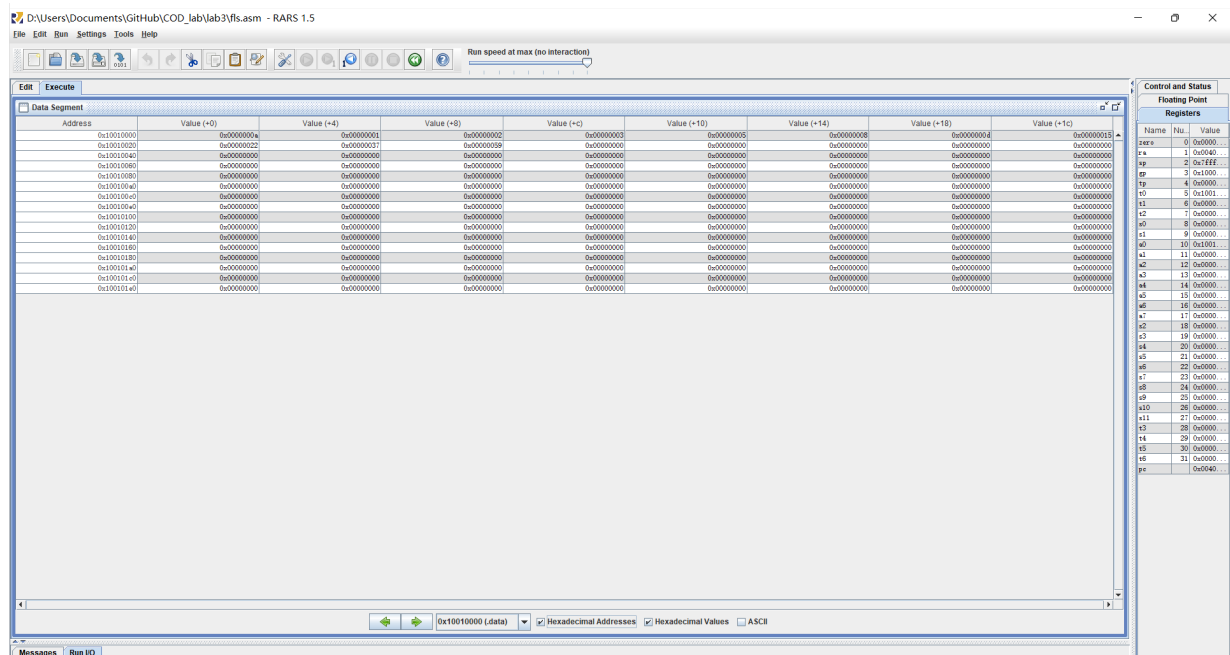
其中第 11 行的 `addi` 指令决定了程序会计算多少项。
程序的输出为

```

1
2
3
5
8
13
21
34
55
89

```

内存中部分值为



即下表

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x0000000a	0x00000001	0x00000002	0x00000003	0x00000005	0x00000008	0x0000000d	0x00000015
0x10010020	0x00000022	0x00000037	0x00000059	0x00000080	0x000000a0	0x000000c0	0x000000e0	0x00000100
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

.text 段生成的 coe 文件如下

1	0fc10517
2	00450513
3	00a00593
4	00052603
5	00452683
6	00850513
7	ffe58593
8	00060713
9	030000ef
10	00068713
11	028000ef
12	04b05663
13	00d60733
14	00e52023
15	018000ef
16	00450513
17	00068613
18	00070693
19	fff58593
20	fe1ff06f
21	00050293
22	00070513
23	00100893
24	00000073
25	0fc10517
26	fa050513
27	00400893
28	00000073

29	00028513
30	000080e7
31	00a00893
32	00000073

5 心得体会

本次实验比较简单.