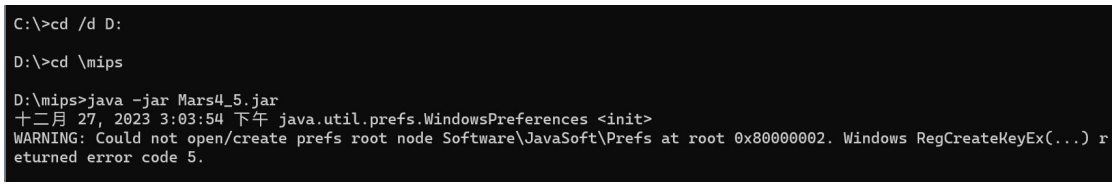
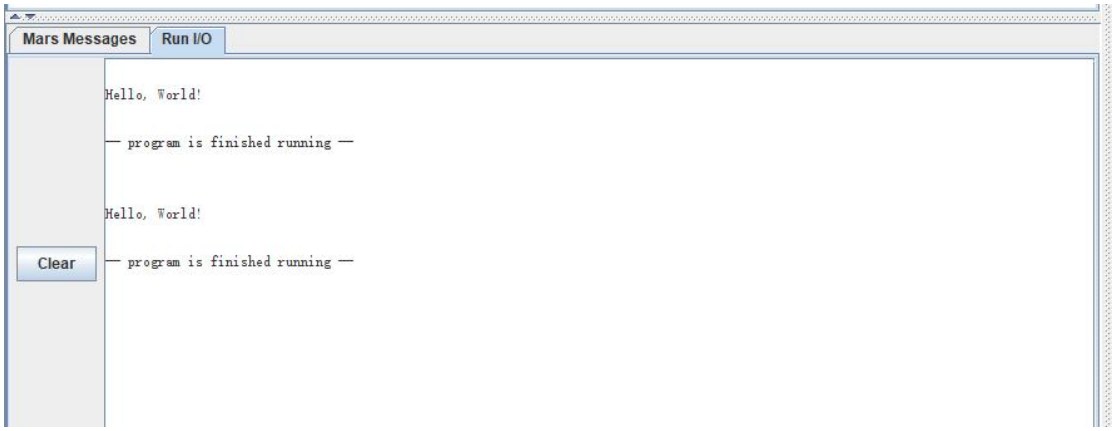
1、MIPS 汇编语言程序的运行

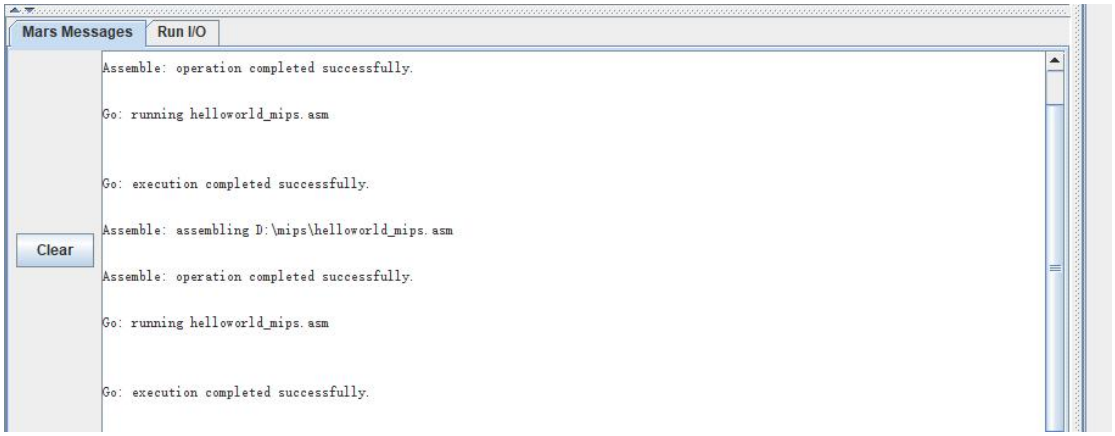
（1）运行 MIPS 汇编仿真器



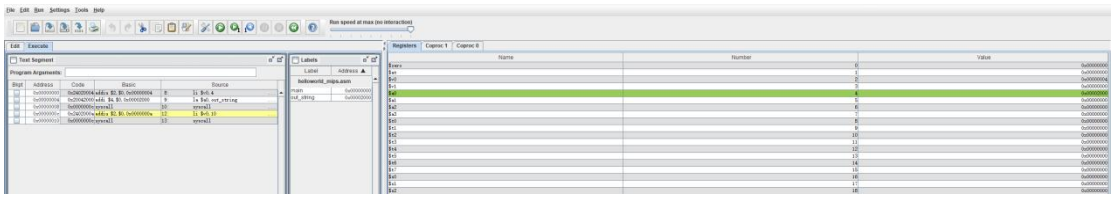
（2）运行第一个 MIPS 汇编语言程序

运行汇编后的程序，下面的界面中显示“Hello, World!”

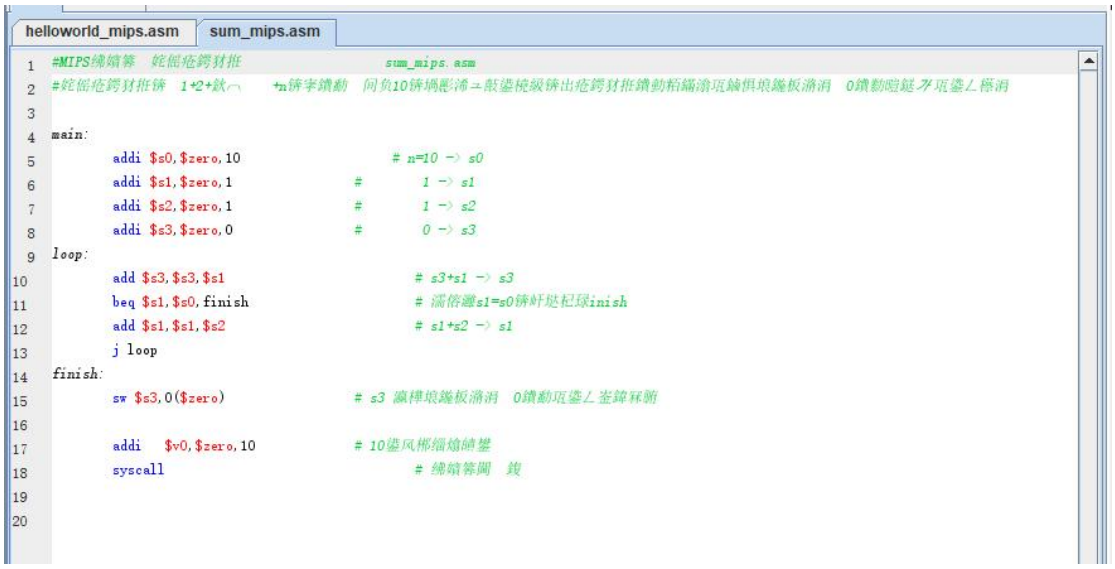




单步执行程序



（3）求累加和的 MIPS 汇编语言程序

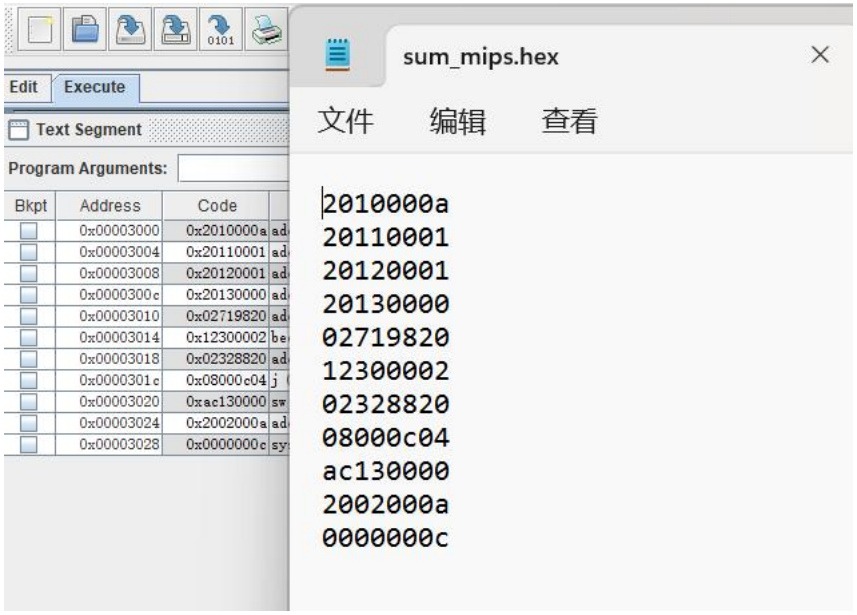


```
helloworld_mips.asm  sum_mips.asm

1  #MIPS编辑等   姣倔疮锣狩推                    sum_mips.asm
2  #姣倔疮锣狩推铢  1+2+鈥﹀    +n铢宰鐨勫   闾负10铢塢彫浠ュ敵鍙梳级铢出疮锣狩推鐨勫粘鍚滀项鑰俱埌鍚板浠涓   0鐨勫暄鎹ㄥ项鍙ㄥ樣涓
3
4  main:
5          addi $s0,$zero,10                  # n=10 -> s0
6          addi $s1,$zero,1                   #      1 -> s1
7          addi $s2,$zero,1                   #      1 -> s2
8          addi $s3,$zero,0                   #      0 -> s3
9  loop:
10         add $s3,$s3,$s1                    # s3+s1 -> s3
11         beq $s1,$s0,finish                 # 濡傛灉s1=s0铢屽垯杞琹inish
12         add $s1,$s1,$s2                    # s1+s2 -> s1
13         j loop
14 finish:
15         sw $s3,0($zero)                # s3 瀛樻埌鍚板浠涓   0鐨勫项鍙ㄥ宓鍏冪
16
17         addi   $v0,$zero,10            # 10鍙风郴缁熺畝鍙
18         syscall                            # 绋嬪簭閫   鍑
19
20
```

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

导出汇编后的机器码，保存到 sum_mips.hex 文件中



Edit Execute

Text Segment

Program Arguments:

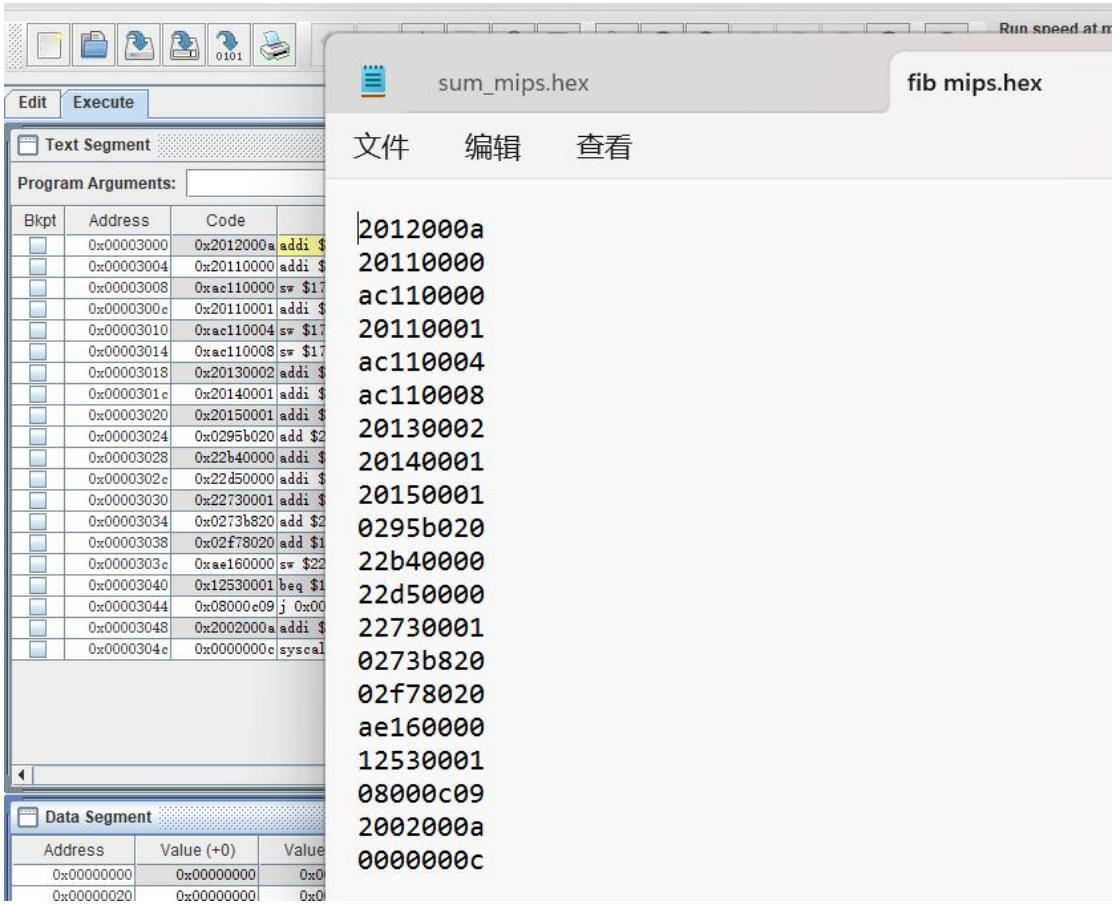
Bkpt	Address	Code
	0x00003000	0x2010000a
	0x00003004	0x20110001
	0x00003008	0x20120001
	0x0000300c	0x20130000
	0x00003010	0x02719820
	0x00003014	0x12300002
	0x00003018	0x02328820
	0x0000301c	0x08000c04
	0x00003020	0xac130000
	0x00003024	0x2002000a
	0x00003028	0x0000000c

sum_mips.hex

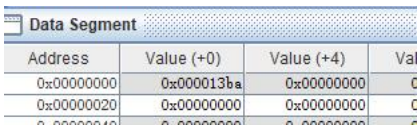
文件 编辑 查看

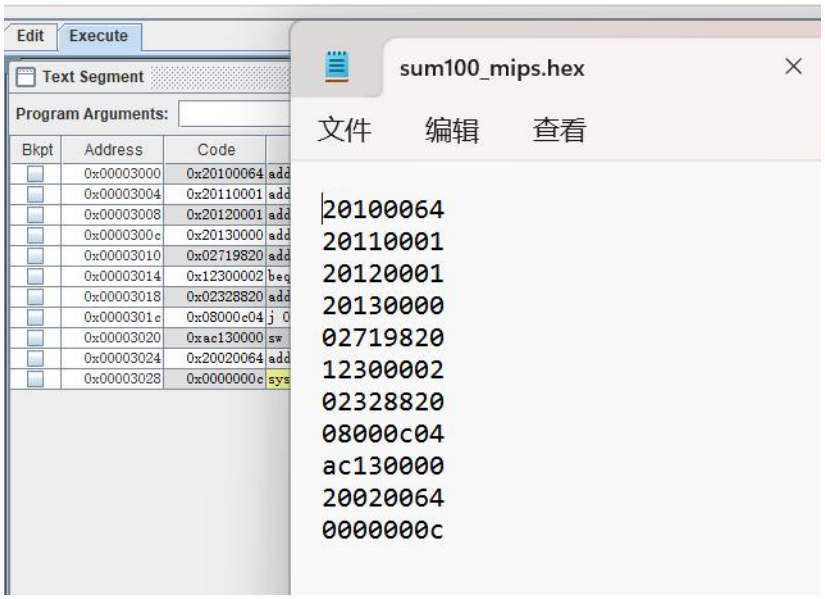
```
2010000a
20110001
20120001
20130000
02719820
12300002
02328820
08000c04
ac130000
2002000a
0000000c
```

（4）计算费波那契数列的 MIPS 汇编语言程序



（5）修改“sum_mips.asm”程序，计算 1+2+3+……+100=5050=13bah，将该程序的机器码保存到 sum100_mips.hex 文件中。





3、RISC-V 汇编语言程序的运行

（1）运行 RISC-V 汇编仿真器

```
D:\mips>cd ..

D:\>cd \riscv

D:\riscv>java -jar rars1_5.jar
十二月 27, 2023 3:23:31 下午 java.util.prefs.WindowsPreferences <init>
WARNING: Could not open/create prefs root node Software\JavaSoft\Prefs at root 0x80000002. Windows RegCreateKeyEx(...) r
eturned error code 5.
```

（2）RISC-V 求累加和程序

Edit Execute

sum_riscv.asm

```
1  # RISC-V计算累加和程序          sum_riscv.asm          result=1+2+...+n
2  # 结果result存放在地址为0的数据存储器中
3
4  main:
5          ori a3,zero,10          # a3=10=n
6      ori a5, zero, 1             # a5=1
7      ori a2, zero, 1             # a2=1
8          ori a4, zero, 0                     # a4=0
9  loop:
10     add a4, a4, a5              # a4+a5 -> a4
11     beq a5, a3, finish          # 判断a5=a3?        若a5=a3=n，则跳出循环
12     add a5, a5, a2              # a5+a2 -> a5
13     jal zero, loop          # 无条件跳转到loop执行
14 finish:
15     sw a4, 0(zero)              # a4 -> (0)              将累加结果result保存到存储器中
16 end:
17     jal zero, end           # 无条件跳转到end执行
18
```

在 RARS 中运行“sum_riscv.asm”,在最后一条指令上设置断点

Text Segment

Bkpt	Address	Code	Basic		Source
	0x00003000	0x00a06693	ori x13,x0,0x0000000a	5: ori a3,zero,10	# a3=10=n
	0x00003004	0x00106793	ori x15,x0,0x00000001	6: ori a5, zero, 1	# a5=1
	0x00003008	0x00106613	ori x12,x0,0x00000001	7: ori a2, zero, 1	# a2=1
	0x0000300c	0x00006713	ori x14,x0,0x00000000	8: ori a4, zero, 0	# a4=0
	0x00003010	0x00f70733	add x14,x14,x15	10: add a4, a4, a5	# a4+a5 -> a4
	0x00003014	0x00d78663	beq x15,x13,0x00000006	11: beq a5, a3, finish	# 判断a5=a3? 若a5=a3=n，则跳出循环
	0x00003018	0x00c787b3	add x15,x15,x12	12: add a5, a5, a2	# a5+a2 -> a5
	0x0000301c	0xff5ff06f	jal x0,0xfffffffa	13: jal zero, loop	# 无条件跳转到loop执行
	0x00003020	0x00e02023	sw x14,0x00000000(x0)	15: sw a4, 0(zero)	# a4 -> (0) 将累加结果result保存到存储器中
✔	0x00003024	0x0000006f	jal x0,0x00000000	17: jal zero, end	# 无条件跳转到end执行

结果 = 37h = 55 = 1+2+3+…+10

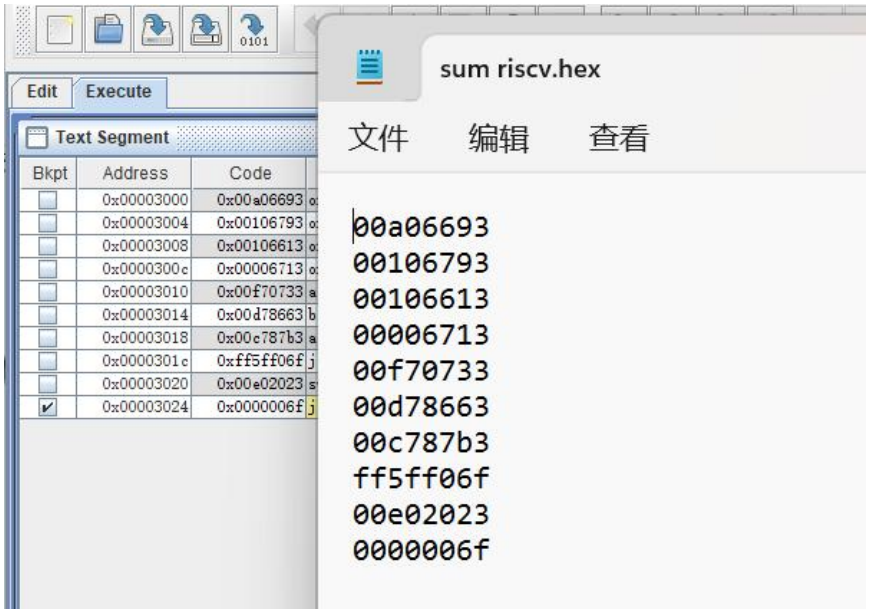
Data Segment

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

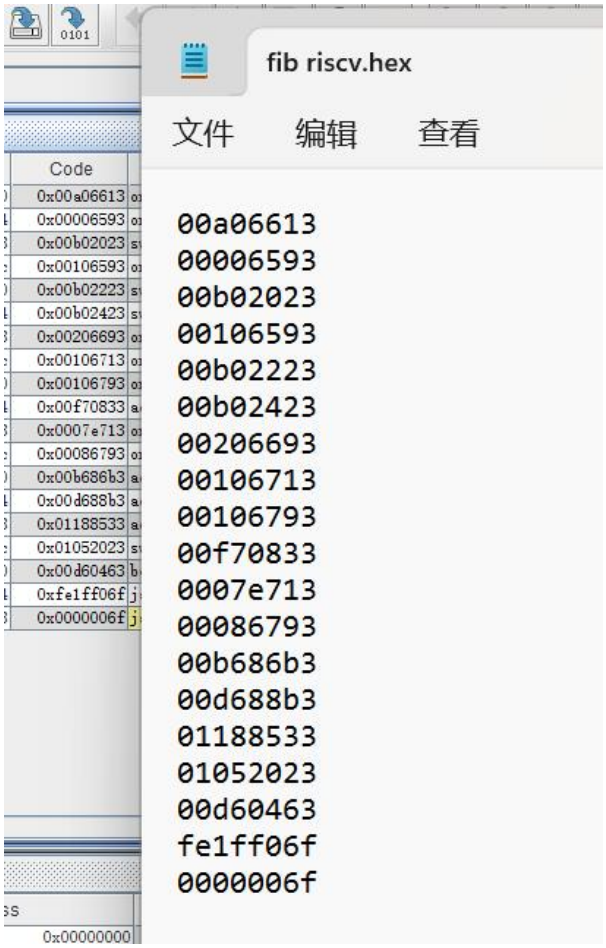
Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00000000	0x00000000	0x00000001	0x00000001	0x00000002	0x00000003	0x00000005	0x00000008	0x0000000d
0x00000020	0x00000015	0x00000022	0x00000037	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

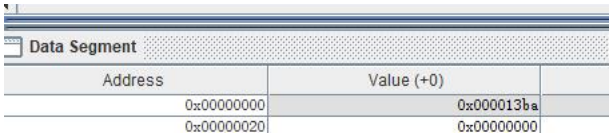
汇编后的机器码，导出到 sum_riscv.hex 中

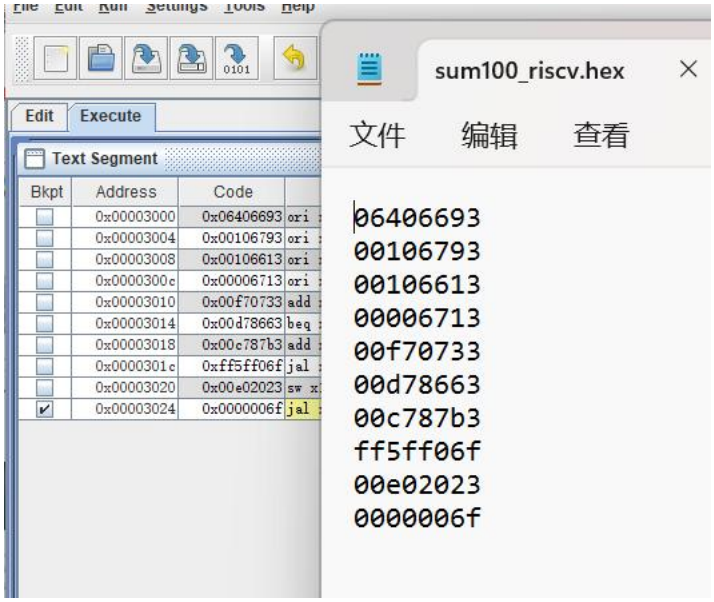


（3）RISC-V 计算费波那契数列的程序
该程序的机器码保存到 fib_riscv.hex 文件中



（4）计算 1+2+3+……+100=5050=13bah,将该程序的机器码保存到 sum100_riscv.hex 文件中。



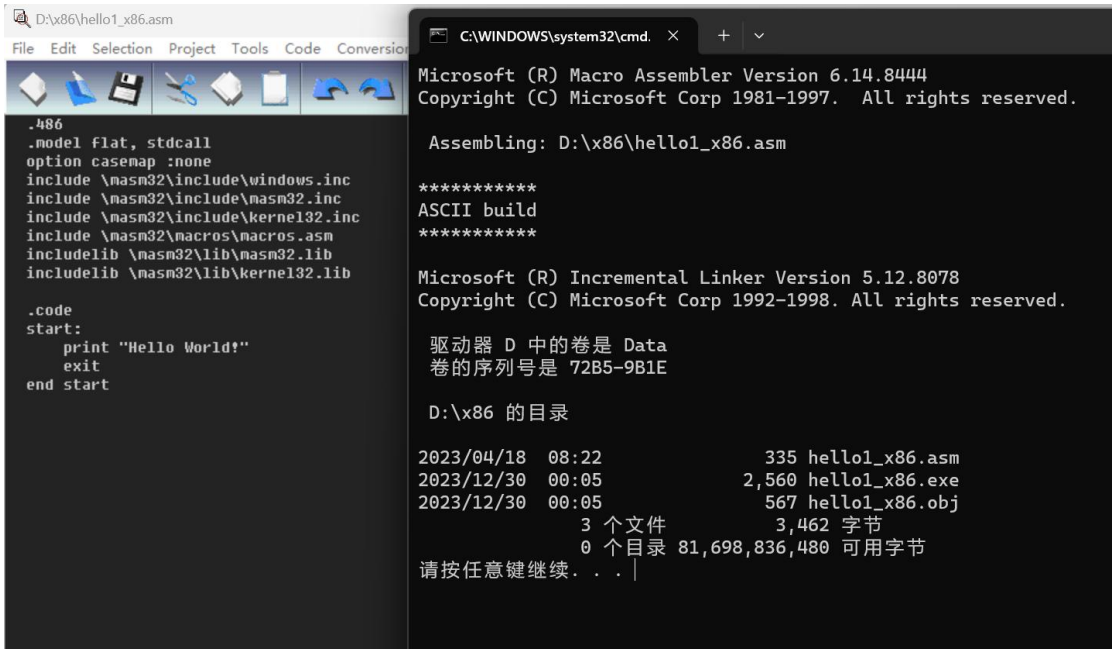


5、Intel x86 汇编语言程序的运行

（1）Intel x86 汇编工具,masm32.rar

（2）运行 Intel x86 汇编语言程序

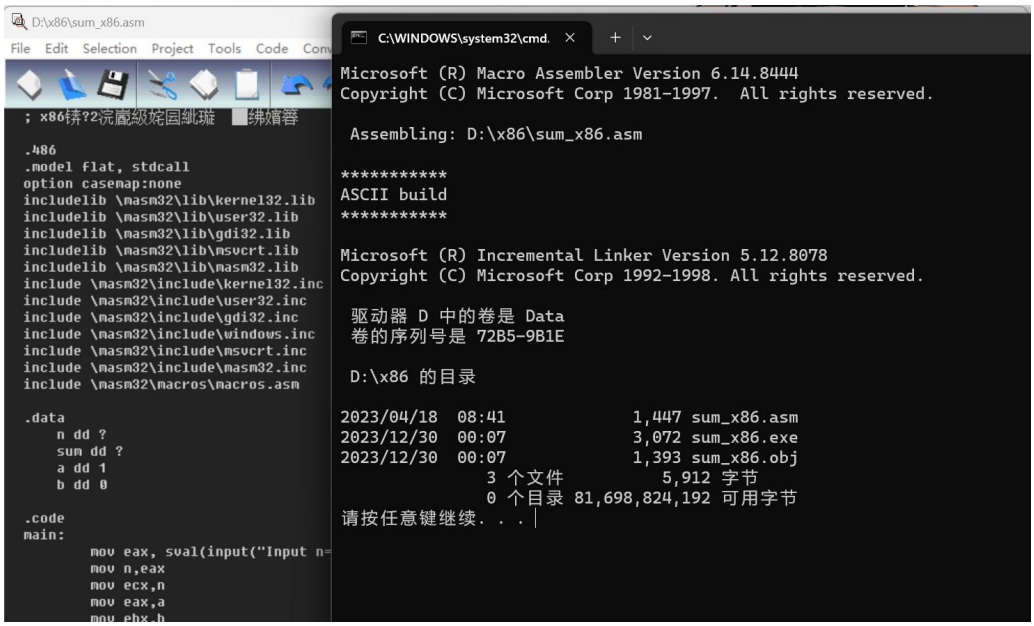
在 masm32 汇编工具中打开源程序“hello1_x86.asm”



运行“hello1_x86.exe”

```
D:\x86>hello1_x86.exe
Hello World!
```

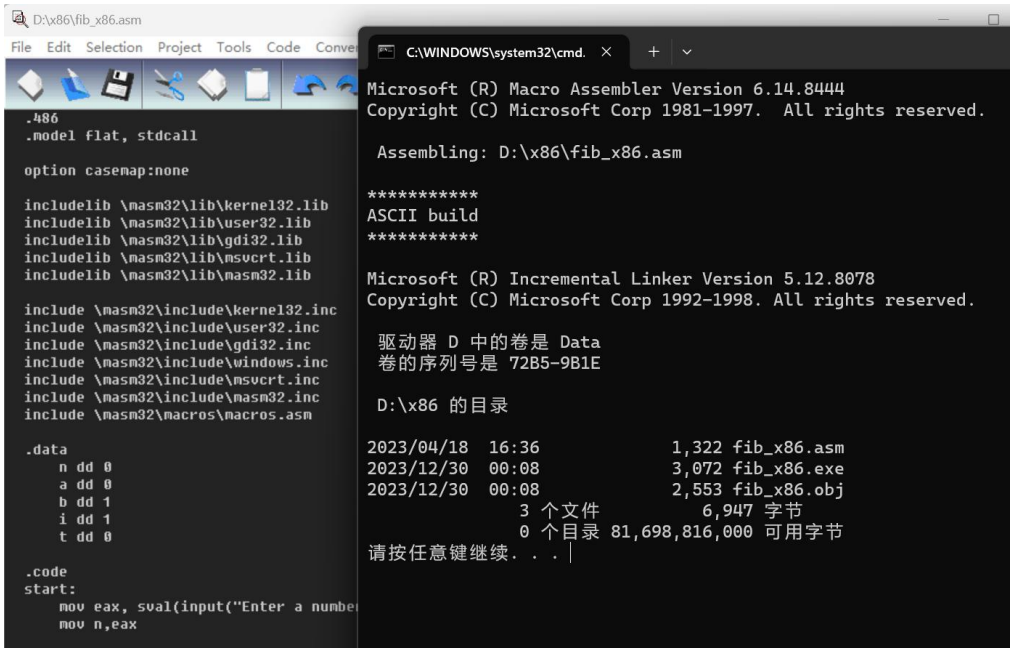
(3）运行求累加和的 Intel x86 汇编语言程序



运行“sum_x86.exe”

```
D:\x86>sum_x86.exe
Input n= 10
Sum= 55
```

（4）运行计算费波那契数列的 Intel x86 汇编语言程序



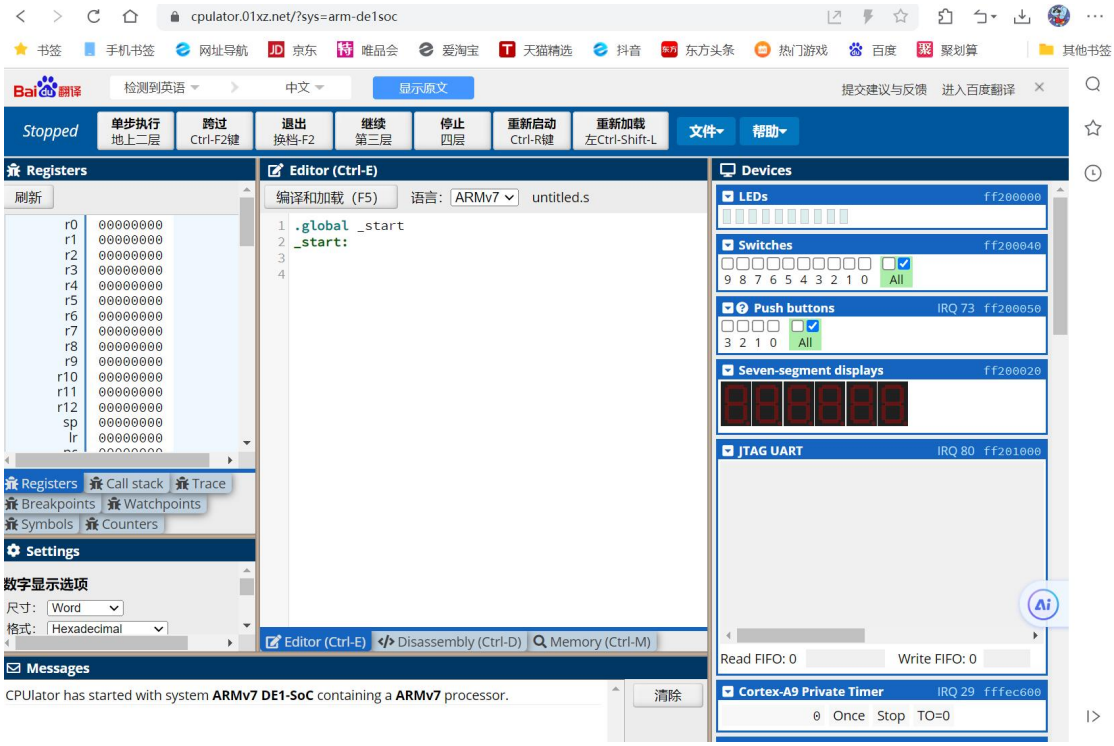
运行“fib_x86.exe”

```
D:\x86>fib_x86.exe
Enter a number : 10
Fibonacci_number0 is 0
Fibonacci_number1 is 1
Fibonacci_number2 is 1
Fibonacci_number3 is 2
Fibonacci_number4 is 3
Fibonacci_number5 is 5
Fibonacci_number6 is 8
Fibonacci_number7 is 13
Fibonacci_number8 is 21
Fibonacci_number9 is 34
Fibonacci_number10 is 55
```

6、ARMv7 汇编语言程序的运行

（1）ARMv7 汇编工具

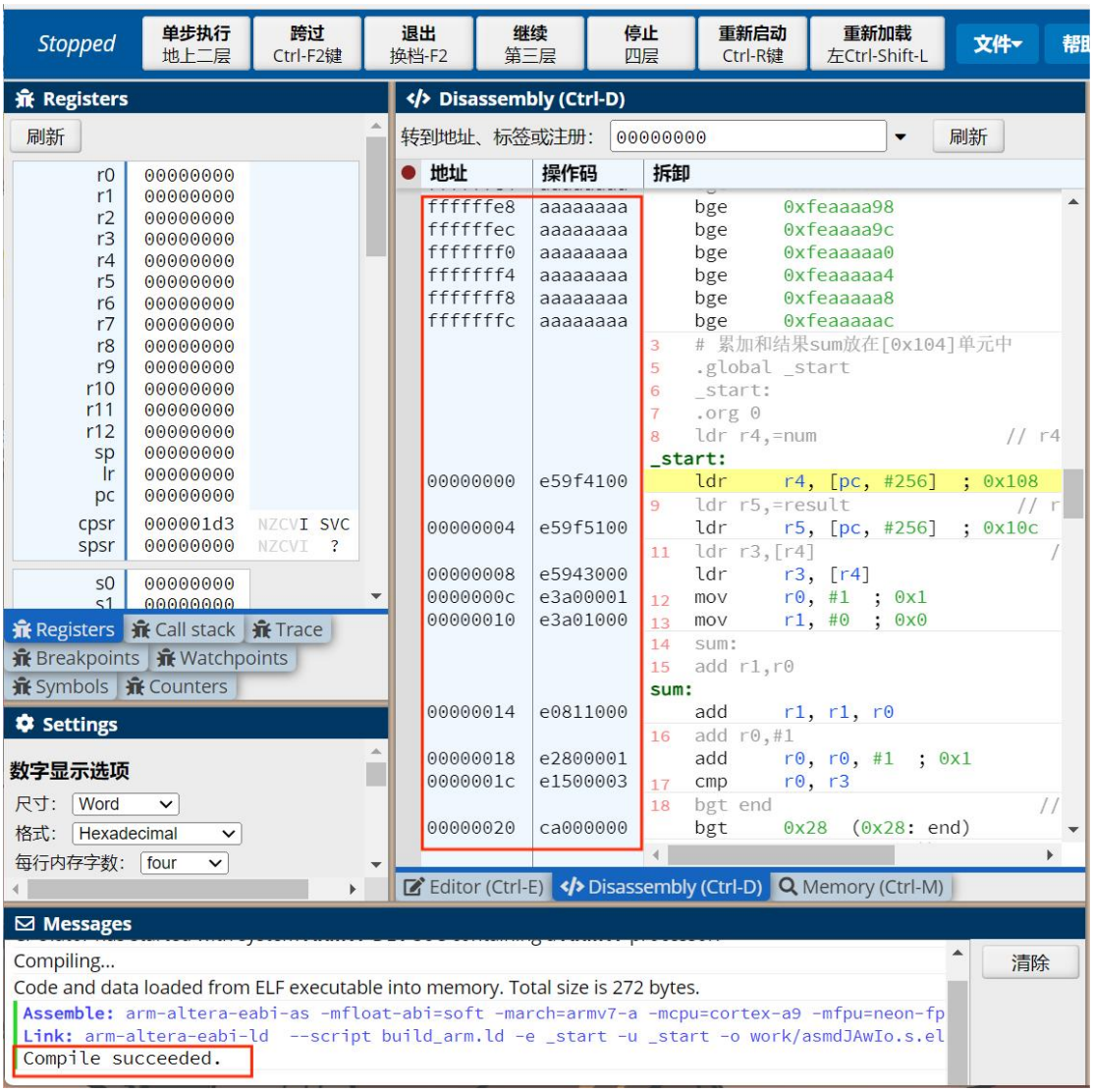
在浏览器中打开 ARMv7 汇编工具：https://cpulator.01xz.net/?sys=arm-de1soc



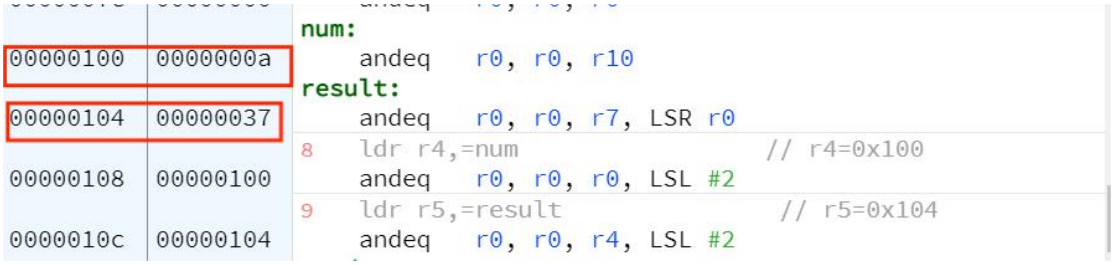
（2）在 ARMv7 汇编工具中运行求累加和程序

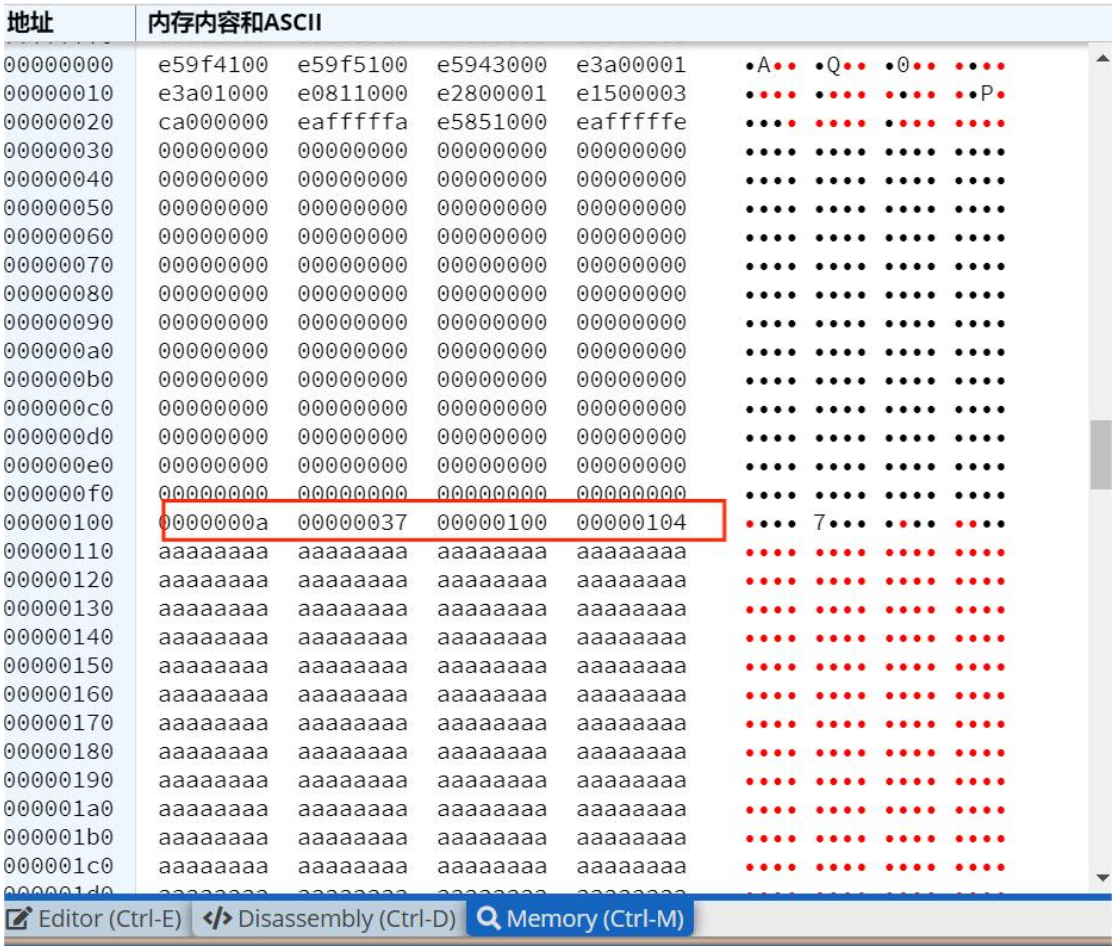
第一步：打开源程序（sum_armv7.s）

第二步：汇编

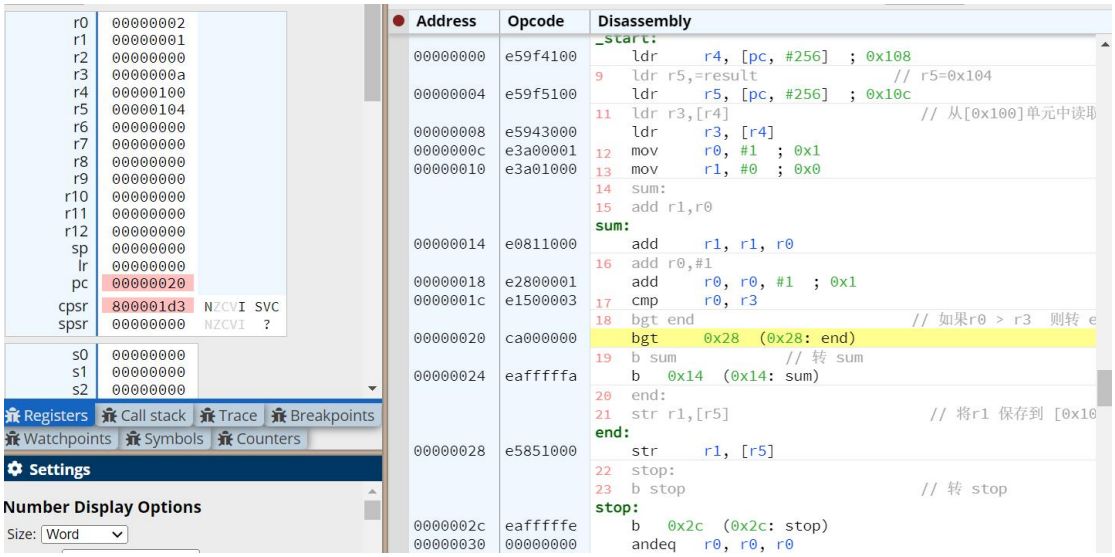

Stopped
单步执行 地上二层
跨过 Ctrl-F2键
退出 换档-F2
继续 第三层
停止 四层
重新启动 Ctrl-R键
重新加载 左Ctrl-Shift-L
文件
Registers
刷新
r0 00000000
r1 00000000
r2 00000000
r3 00000000
r4 00000000
r5 00000000
r6 00000000
r7 00000000
r8 00000000
r9 00000000
r10 00000000
r11 00000000
r12 00000000
sp 00000000
lr 00000000
pc 00000000
cpsr 000001d3 NZCVI SVC
spsr 00000000 NZCVI ?
s0 00000000
Registers
Call stack
Trace
Breakpoints
Watchpoints
Symbols
Counters
Settings
数字显示选项
尺寸: Word
格式: Hexadecimal
每行内存字数: four
Disassembly (Ctrl-D)
转到地址、标签或注册: 00000000
刷新
地址
操作码
拆卸
fffffff8 aaaaaaaa bge 0xfeaaaa98
fffffffc aaaaaaaa bge 0xfeaaaa9c
ffffffff0 aaaaaaaa bge 0xfeaaaaa0
ffffffff4 aaaaaaaa bge 0xfeaaaaa4
ffffffff8 aaaaaaaa bge 0xfeaaaaa8
ffffffffc aaaaaaaa bge 0xfeaaaaac
3 # 累加和结果sum放在[0x104]单元中
5 .global _start
6 _start:
7 .org 0
8 ldr r4,=num // r4
_start:
00000000 e59f4100 ldr r4, [pc, #256] ; 0x108
9 ldr r5,=result // r
00000004 e59f5100 ldr r5, [pc, #256] ; 0x10c
11 ldr r3,[r4] /
00000008 e5943000 ldr r3, [r4]
0000000c e3a00001 12 mov r0, #1 ; 0x1
00000010 e3a01000 13 mov r1, #0 ; 0x0
14 sum:
15 add r1,r0
sum:
00000014 e0811000 add r1, r1, r0
16 add r0,#1
00000018 e2800001 add r0, r0, #1 ; 0x1
0000001c e1500003 17 cmp r0, r3
18 bgt end //
00000020 ca000000 bgt 0x28 (0x28: end)
Editor (Ctrl-E)
Disassembly (Ctrl-D)
Memory (Ctrl-M)
Messages
清除
Compiling...
Code and data loaded from ELF executable into memory. Total size is 272 bytes.
Assemble: arm-altera-eabi-as -mfloat-abi=soft -march=armv7-a -mcpu=cortex-a9 -mfpu=neon-fp
Link: arm-altera-eabi-ld --script build_arm.ld -e _start -u _start -o work/asmdJAwIo.s.el
Compile succeeded.

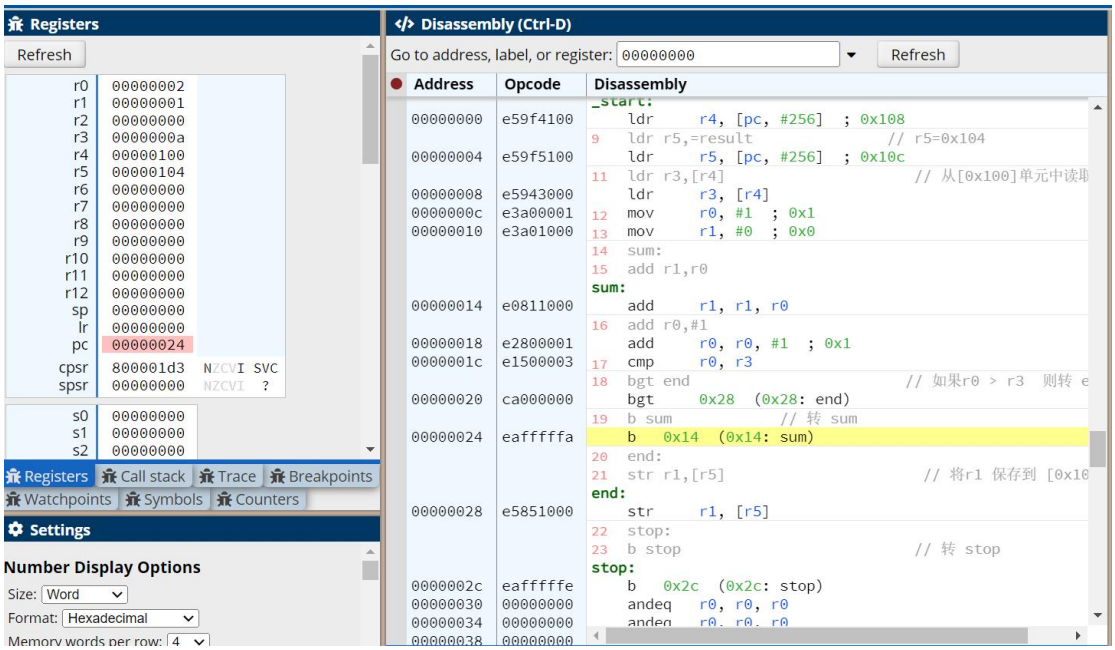
第三步：连续执行程序


num:
00000100 0000000a andeq r0, r0, r10
result:
00000104 00000037 andeq r0, r0, r7, LSR r0
8 ldr r4,=num // r4=0x100
00000108 00000100 andeq r0, r0, r0, LSL #2
9 ldr r5,=result // r5=0x104
0000010c 00000104 andeq r0, r0, r4, LSL #2


地址
内存内容和ASCII
00000000 e59f4100 e59f5100 e5943000 e3a00001
00000010 e3a01000 e0811000 e2800001 e1500003
00000020 ca000000 eafffffa e5851000 eafffffe
00000030 00000000 00000000 00000000 00000000
00000040 00000000 00000000 00000000 00000000
00000050 00000000 00000000 00000000 00000000
00000060 00000000 00000000 00000000 00000000
00000070 00000000 00000000 00000000 00000000
00000080 00000000 00000000 00000000 00000000
00000090 00000000 00000000 00000000 00000000
000000a0 00000000 00000000 00000000 00000000
000000b0 00000000 00000000 00000000 00000000
000000c0 00000000 00000000 00000000 00000000
000000d0 00000000 00000000 00000000 00000000
000000e0 00000000 00000000 00000000 00000000
000000f0 00000000 00000000 00000000 00000000
00000100 0000000a 00000037 00000100 00000104
00000110 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
00000120 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
00000130 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
00000140 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
00000150 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
00000160 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
00000170 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
00000180 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
00000190 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
000001a0 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
000001b0 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
000001c0 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
Editor (Ctrl-E)
Disassembly (Ctrl-D)
Memory (Ctrl-M)

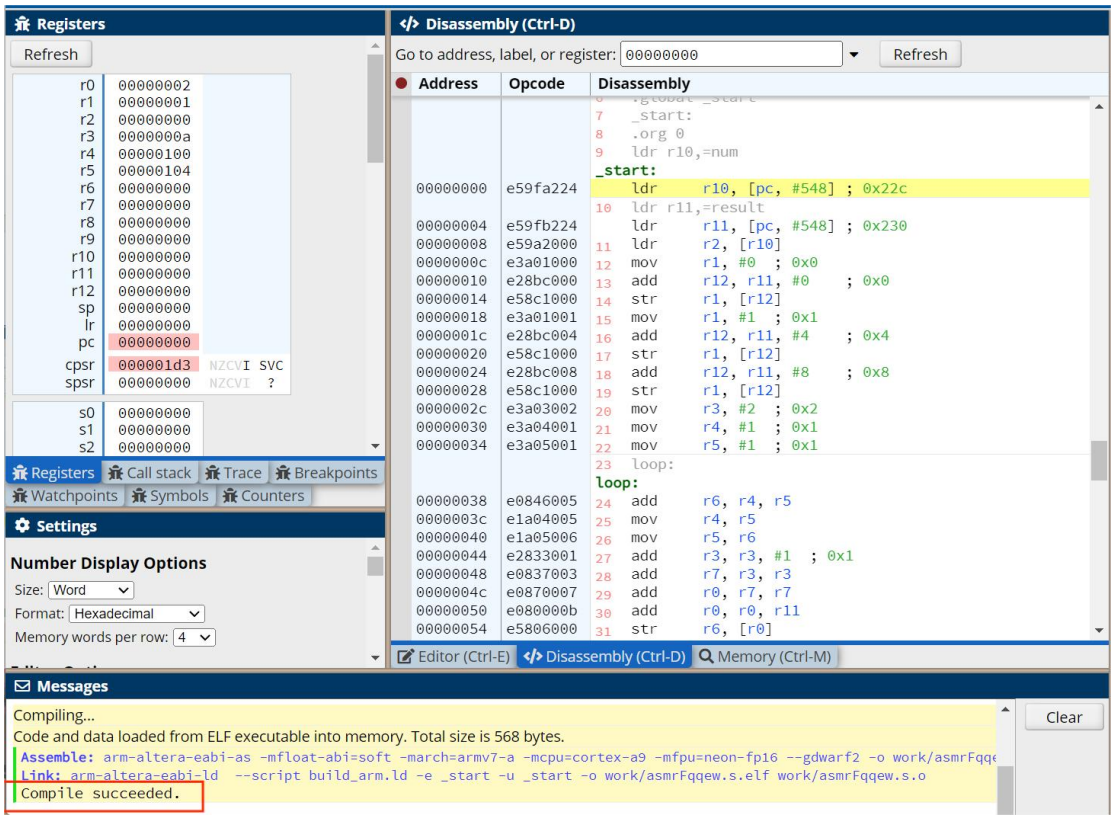
第四步：单步执行程序


r0 00000002
r1 00000001
r2 00000000
r3 0000000a
r4 00000100
r5 00000104
r6 00000000
r7 00000000
r8 00000000
r9 00000000
r10 00000000
r11 00000000
r12 00000000
sp 00000000
lr 00000000
pc 00000020
cpsr 800001d3 NZCVI SVC
spsr 00000000 NZCVI ?
s0 00000000
s1 00000000
s2 00000000
Registers
Call stack
Trace
Breakpoints
Watchpoints
Symbols
Counters
Settings
Number Display Options
Size: Word
Address
Opcode
Disassembly
00000000 e59f4100 ldr r4, [pc, #256] ; 0x108
ldr r5,=result // r5=0x104
00000004 e59f5100 ldr r5, [pc, #256] ; 0x10c
ldr r3,[r4] // 从[0x100]单元中读取
00000008 e5943000 ldr r3, [r4]
0000000c e3a00001 mov r0, #1 ; 0x1
00000010 e3a01000 mov r1, #0 ; 0x0
sum:
add r1,r0
sum:
00000014 e0811000 add r1, r1, r0
add r0,#1
00000018 e2800001 add r0, r0, #1 ; 0x1
0000001c e1500003 cmp r0, r3
bgt end // 如果r0 > r3 则转 e
00000020 ca000000 bgt 0x28 (0x28: end)
b sum // 转 sum
00000024 eafffffa b 0x14 (0x14: sum)
end:
str r1,[r5] // 将r1 保存到 [0x10
end:
00000028 e5851000 str r1, [r5]
stop:
b stop // 转 stop
stop:
0000002c eafffffe b 0x2c (0x2c: stop)
00000030 00000000 andeq r0, r0, r0

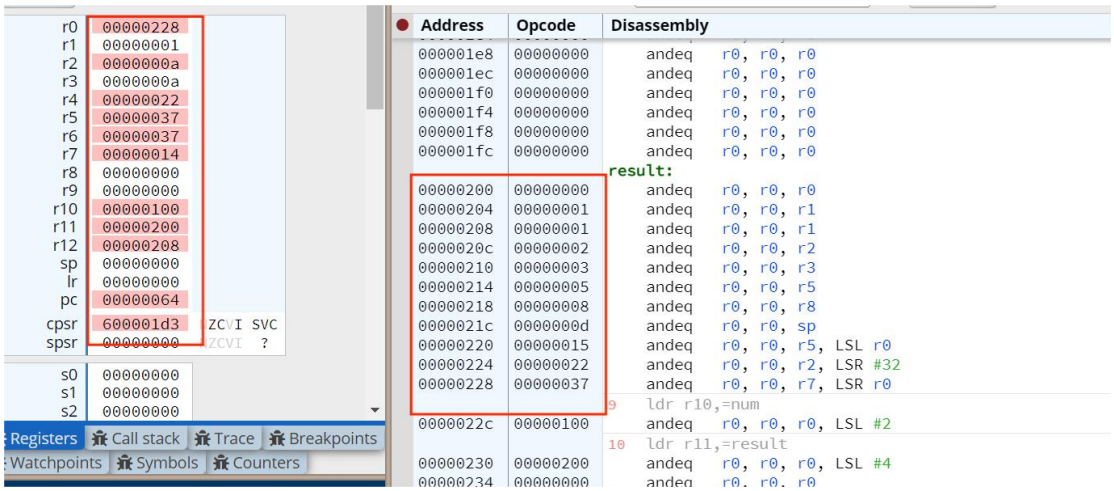


（3）在 ARMv7 汇编工具中运行计算费波那契数列程序

汇编



连续执行程序


r0 00000228
r1 00000001
r2 0000000a
r3 0000000a
r4 00000022
r5 00000037
r6 00000037
r7 00000014
r8 00000000
r9 00000000
r10 00000100
r11 00000200
r12 00000208
sp 00000000
lr 00000000
pc 00000064
cpsr 600001d3 nZC\I SVC
spsr 00000000 nZCVI ?
s0 00000000
s1 00000000
s2 00000000
Registers
Call stack
Trace
Breakpoints
Watchpoints
Symbols
Counters
Address
Opcode
Disassembly
000001e8 00000000 andeq r0, r0, r0
000001ec 00000000 andeq r0, r0, r0
000001f0 00000000 andeq r0, r0, r0
000001f4 00000000 andeq r0, r0, r0
000001f8 00000000 andeq r0, r0, r0
000001fc 00000000 andeq r0, r0, r0
result:
00000200 00000000 andeq r0, r0, r0
00000204 00000001 andeq r0, r0, r1
00000208 00000001 andeq r0, r0, r1
0000020c 00000002 andeq r0, r0, r2
00000210 00000003 andeq r0, r0, r3
00000214 00000005 andeq r0, r0, r5
00000218 00000008 andeq r0, r0, r8
0000021c 0000000d andeq r0, r0, sp
00000220 00000015 andeq r0, r0, r5, LSL r0
00000224 00000022 andeq r0, r0, r2, LSR #32
00000228 00000037 andeq r0, r0, r7, LSR r0
9 ldr r10,=num
0000022c 00000100 andeq r0, r0, r0, LSL #2
10 ldr r11,=result
00000230 00000200 andeq r0, r0, r0, LSL #4