

## 1、MIPS 汇编语言程序的运行

### (1) 运行 MIPS 汇编仿真器

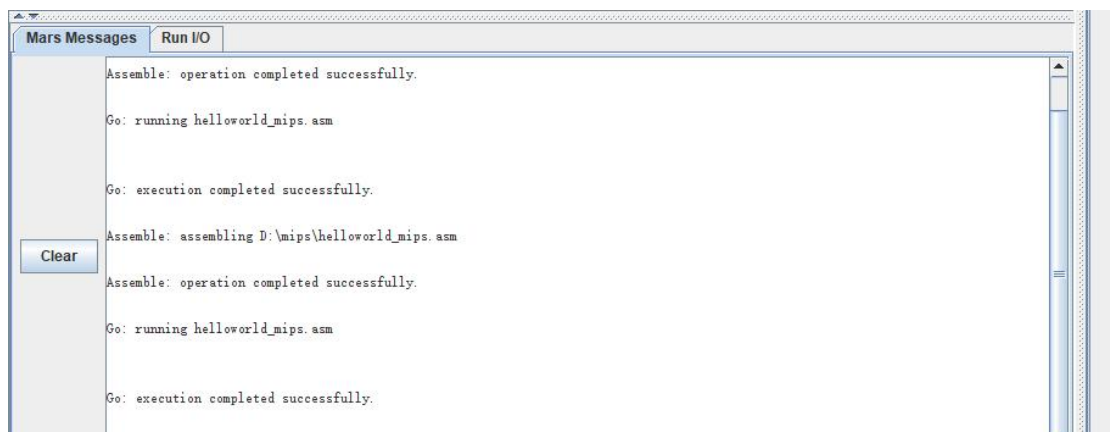
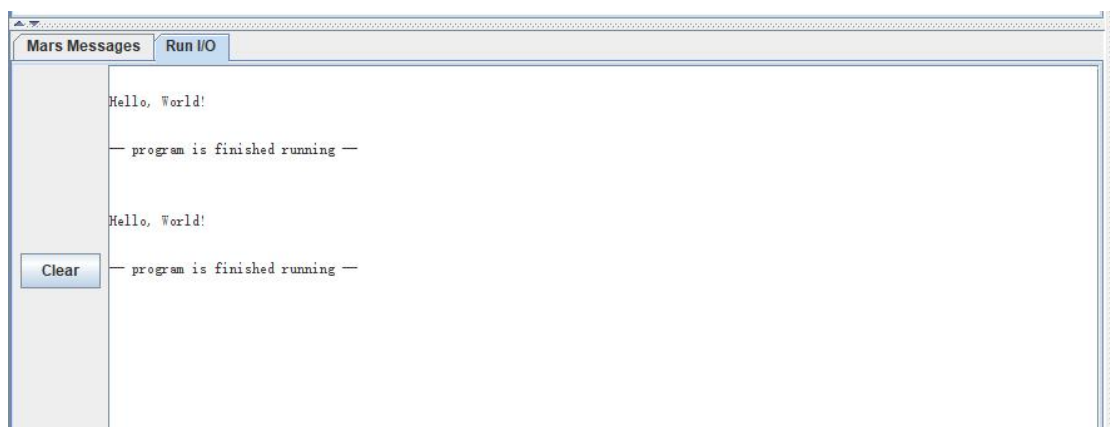
```
C:\WINDOWS\system32\cmd. X + v
Microsoft Windows [版本 10.0.22621.2861]
(c) Microsoft Corporation。保留所有权利。

C:\Users\86187>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)
```

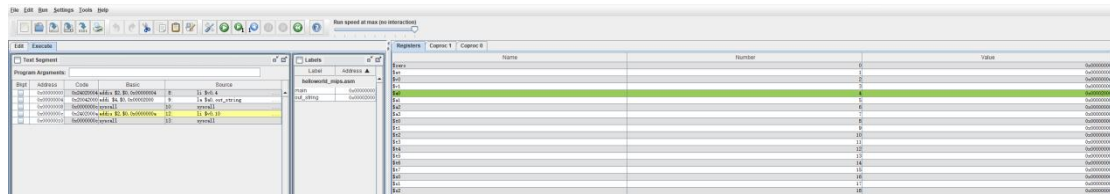
```
C:\>cd /d D:
D:\>cd \mips
D:\mips>java -jar Mars4_5.jar
十二月 27, 2023 3:03:54 下午 java.util.prefs.WindowsPreferences <init>
WARNING: Could not open/create prefs root node Software\JavaSoft\Prefs at root 0x80000002. Windows RegCreateKeyEx(...) r
eturned error code 5.
```

### (2) 运行第一个 MIPS 汇编语言程序

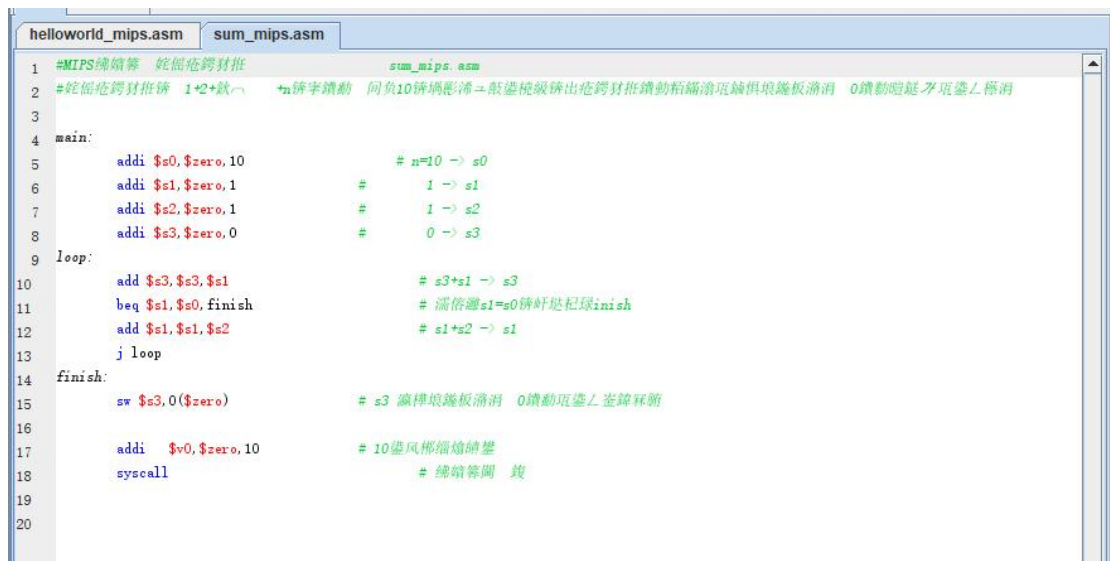
运行汇编后的程序，下面的界面中显示 “Hello, World!”



## 单步执行程序

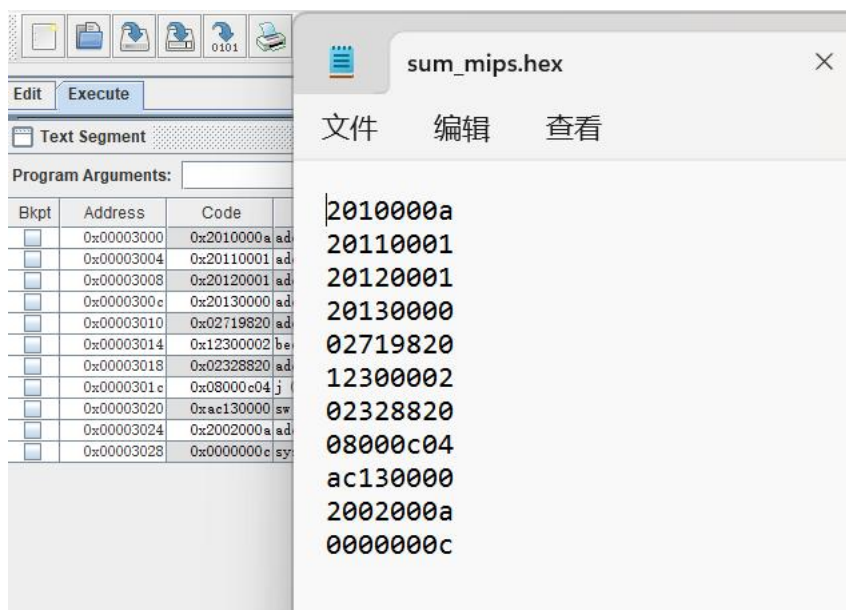


### (3) 求累加和的 MIPS 汇编语言程序



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



(4) 计算费波那契数列的 MIPS 汇编语言程序

The screenshot shows a MIPS simulator interface. The 'Text Segment' window displays the following assembly code:

Bkpt	Address	Code
	0x00003000	0x2012000a addi \$
	0x00003004	0x20110000 addi \$
	0x00003008	0xac110000 sw \$17
	0x0000300c	0x20110001 addi \$
	0x00003010	0xac110004 sw \$17
	0x00003014	0xac110008 sw \$17
	0x00003018	0x20130002 addi \$
	0x0000301c	0x20140001 addi \$
	0x00003020	0x20150001 addi \$
	0x00003024	0x0295b020 add \$2
	0x00003028	0x22b40000 addi \$
	0x0000302c	0x22d50000 addi \$
	0x00003030	0x22730001 addi \$
	0x00003034	0x0273b820 add \$2
	0x00003038	0x02f78020 add \$1
	0x0000303c	0xae160000 sw \$22
	0x00003040	0x12530001 beq \$1
	0x00003044	0x08000c09 j 0x00
	0x00003048	0x2002000a addi \$
	0x0000304c	0x0000000c syscall

The 'Data Segment' window shows the following memory values:

Address	Value (+0)	Value (+4)
0x00000000	0x00000000	0x0
0x00000020	0x00000000	0x0

A hex dump window is open, displaying the memory contents of 'sum\_mips.hex' in hexadecimal format:

```

2012000a
20110000
ac110000
20110001
ac110004
ac110008
20130002
20140001
20150001
0295b020
22b40000
22d50000
22730001
0273b820
02f78020
ae160000
12530001
08000c09
2002000a
0000000c
  
```

(5) 修改“sum\_mips.asm”程序，计算  $1+2+3+\dots+100=5050=13bah$ ，将该程序的机器码保存到 sum100\_mips.hex 文件中。

The screenshot shows a MIPS simulator interface. The 'Data Segment' window shows the following memory values:

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

The 'Text Segment' window displays the following assembly code:

Bkpt	Address	Code
	0x00003000	0x20100064 addi \$
	0x00003004	0x20110001 addi \$
	0x00003008	0x20120001 addi \$
	0x0000300c	0x20130000 addi \$
	0x00003010	0x02719820 addi \$
	0x00003014	0x12300002 beq \$
	0x00003018	0x02328820 addi \$
	0x0000301c	0x08000c04 j 0
	0x00003020	0xac130000 sw \$
	0x00003024	0x20020064 addi \$
	0x00003028	0x0000000c syscall

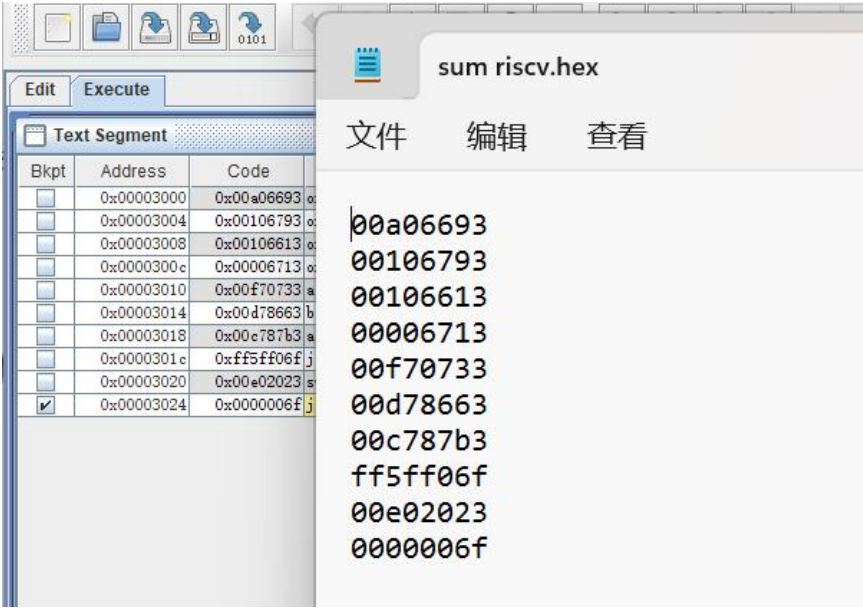
A hex dump window is open, displaying the memory contents of 'sum100\_mips.hex' in hexadecimal format:

```

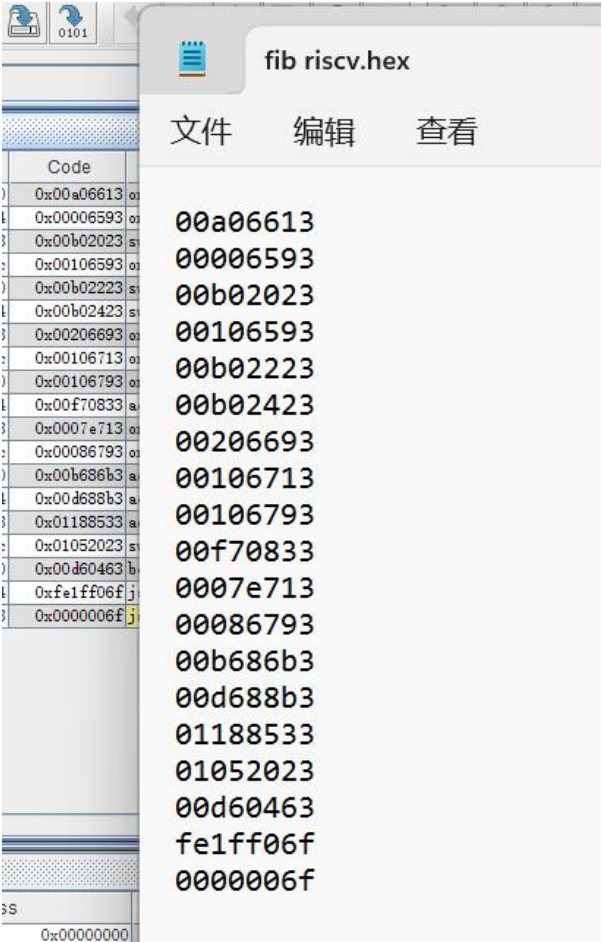
20100064
20110001
20120001
20130000
02719820
12300002
02328820
08000c04
ac130000
20020064
0000000c
  
```

Data Segment										0
Address	Value (+0)	Value (+4)	Value (+8)	Value (+C)	Value (+10)	Value (+14)	Value (+18)	Value (+1C)		
0x00000000	0x00000000	0x00000001	0x00000001	0x00000001	0x00000002	0x00000003	0x00000006	0x0000000A	0x0000000A	
0x00000020	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007	0x00000008	0x00000009	
0x00000040	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007	0x00000008	0x00000009	0x0000000A	

汇编后的机器码，导出到 sum\_riscv.hex 中

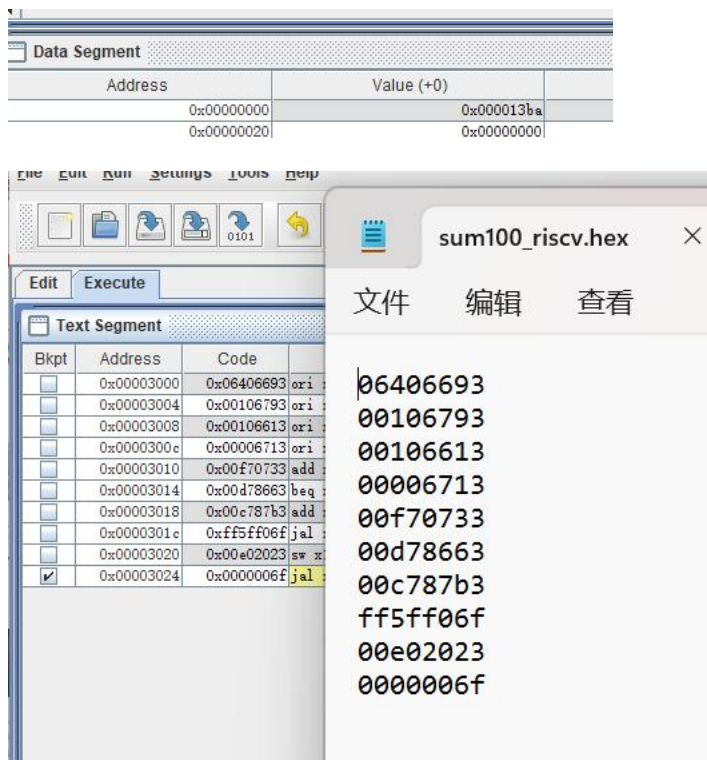


(3) RISC-V 计算费波那契数列的程序  
该程序的机器码保存到 fib\_riscv.hex 文件中





(4) 计算  $1+2+3+\dots+100=5050=13\text{bah}$ , 将该程序的机器码保存到 sum100\_riscv.hex 文件中。

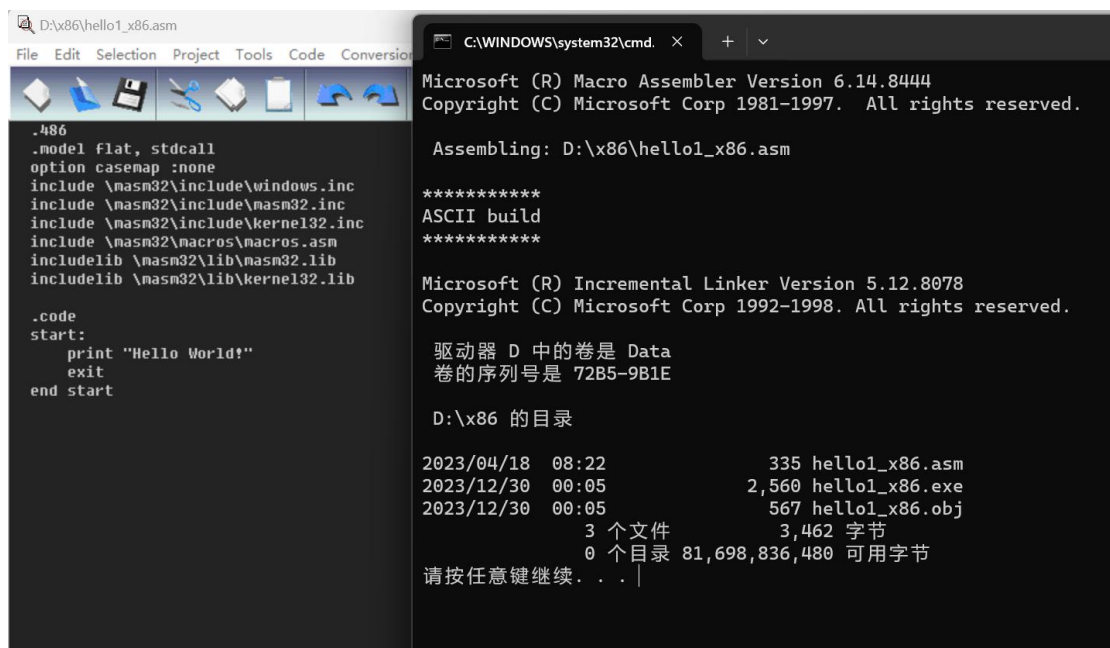


## 5、Intel x86 汇编语言程序的运行

(1) Intel x86 汇编工具, masm32.rar

(2) 运行 Intel x86 汇编语言程序

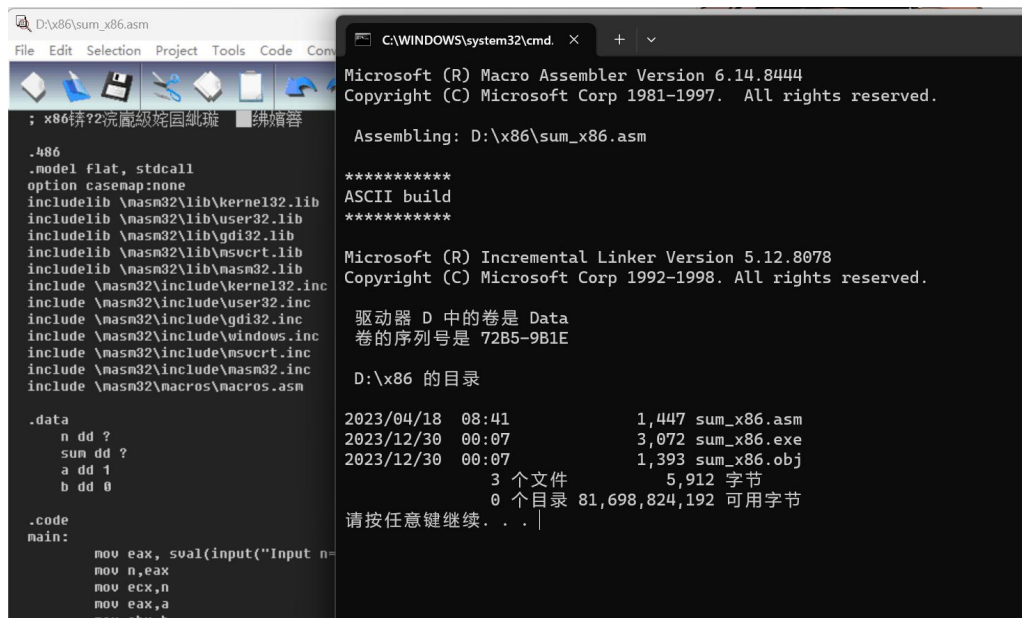
在 masm32 汇编工具中打开源程序 “hello1\_x86.asm”



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



The screenshot shows the assembly environment with the source code for `sum_x86.asm` on the left and the command prompt output on the right.

**Source Code (sum\_x86.asm):**

```
.486
.model flat, stdcall
option casemap:none
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib
includelib \masm32\lib\gdi32.lib
includelib \masm32\lib\msvcrt.lib
includelib \masm32\lib\masm32.lib
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
include \masm32\include\gdi32.inc
include \masm32\include\windows.inc
include \masm32\include\msvcrt.inc
include \masm32\include\masm32.inc
include \masm32\macros\macros.asm

.data
n dd ?
sum dd ?
a dd 1
b dd 0

.code
main:
    mov eax, sval(input("Input n="
    mov n,eax
    mov ecx,n
    mov eax,a
    mov ebx,b
```

**Command Prompt Output:**

```
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: D:\x86\sum_x86.asm

*****
ASCII build
*****

Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

驱动器 D 中的卷是 Data
卷的序列号是 72B5-9B1E

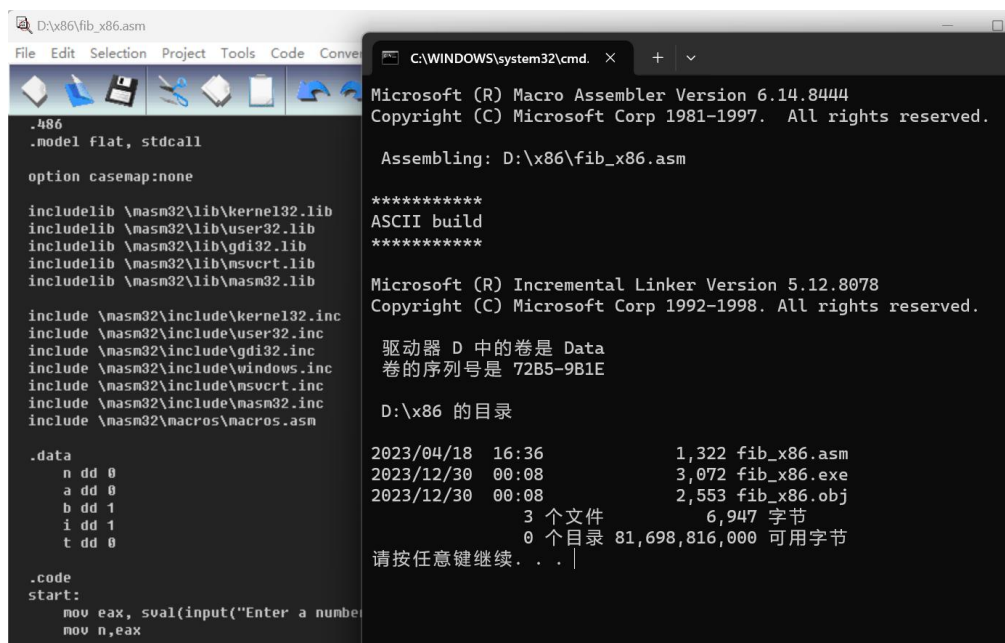
D:\x86 的目录

2023/04/18 08:41 1,447 sum_x86.asm
2023/12/30 00:07 3,072 sum_x86.exe
2023/12/30 00:07 1,393 sum_x86.obj
3 个文件 5,912 字节
0 个目录 81,698,824,192 可用字节
请按任意键继续. . .
```

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



The screenshot shows the assembly environment with the source code for `fib_x86.asm` on the left and the command prompt output on the right.

**Source Code (fib\_x86.asm):**

```
.486
.model flat, stdcall
option casemap:none

includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib
includelib \masm32\lib\gdi32.lib
includelib \masm32\lib\msvcrt.lib
includelib \masm32\lib\masm32.lib

include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
include \masm32\include\gdi32.inc
include \masm32\include\windows.inc
include \masm32\include\msvcrt.inc
include \masm32\include\masm32.inc
include \masm32\macros\macros.asm

.data
n dd 0
a dd 0
b dd 1
i dd 1
t dd 0

.code
start:
    mov eax, sval(input("Enter a number
    mov n,eax
```

**Command Prompt Output:**

```
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: D:\x86\fib_x86.asm

*****
ASCII build
*****

Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

驱动器 D 中的卷是 Data
卷的序列号是 72B5-9B1E

D:\x86 的目录

2023/04/18 16:36 1,322 fib_x86.asm
2023/12/30 00:08 3,072 fib_x86.exe
2023/12/30 00:08 2,553 fib_x86.obj
3 个文件 6,947 字节
0 个目录 81,698,816,000 可用字节
请按任意键继续. . .
```

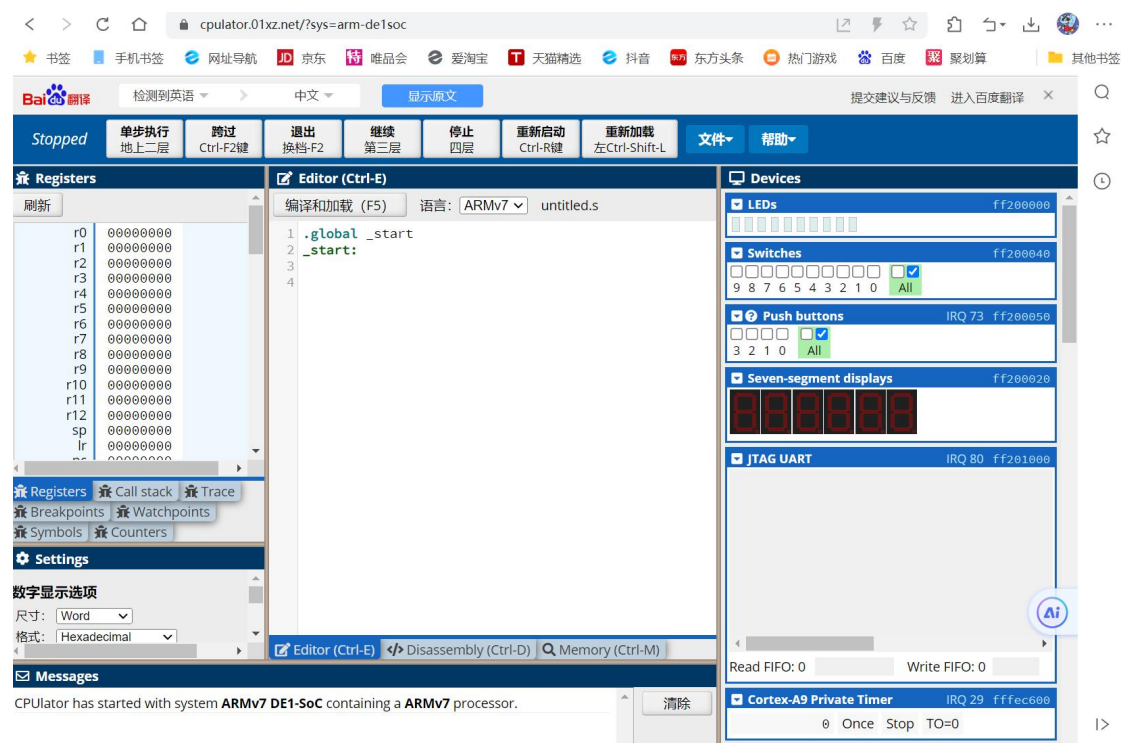
运行“fib\_x86.exe”

```
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程序

第一步: 打开源程序 (sum\_armv7.s)

第二步: 汇编



Stopped 单步执行 跨过 退出 继续 停止 重新启动 重新加载 文件 帮助  
地上二层 Ctrl-F2键 换档-F2 第三层 第四层 Ctrl-R键 左Ctrl-Shift-L

**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
s1	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
ffffffe8	aaaaaaaa	bge 0xfeaaaa98
ffffffec	aaaaaaaa	bge 0xfeaaaa9c
fffffff0	aaaaaaaa	bge 0xfeaaaaa0
fffffff4	aaaaaaaa	bge 0xfeaaaaa4
fffffff8	aaaaaaaa	bge 0xfeaaaaa8
fffffffc	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
00000004	e59f5100	ldr r5,=result // r
9 ldr r5, [pc, #256] ; 0x10c		
00000008	e5943000	ldr r3, [r4]
0000000c	e3a00001	mov r0, #1 ; 0x1
00000010	e3a01000	mov r1, #0 ; 0x0
14 sum:		
15 add r1,r0		
sum:		
00000014	e0811000	add r1, r1, r0
00000018	e2800001	add r0,#1
0000001c	e1500003	add r0, r0, #1 ; 0x1
00000020	ca000000	cmp r0, r3
18 bgt end //		
bgt 0x28 (0x28: end)		

Editor (Ctrl-E) Disassembly (Ctrl-D) Memory (Ctrl-M)

**Messages**

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 -mfp=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

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

地址	内存内容和ASCII				
00000000	e59f4100	e59f5100	e5943000	e3a00001	•A••••Q••••0••••••••
00000010	e3a01000	e0811000	e2800001	e1500003	••••••••••••••P•
00000020	ca000000	ea000000	e5851000	ea000000	••••••••••••••••
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	••••7•••~•~•~•~•~•~•~•~•
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	••••~•~•~•~•~•~•~•~•
000001d0	aaaaaaaa	aaaaaaaa	aaaaaaaa	aaaaaaaa	••••~•~•~•~•~•~•~•~•

第四步：单步执行程序

Address	Opcode	Disassembly
00000000	e59f4100	ldr r4, [pc, #256] ; 0x108
00000004	e59f5100	ldr r5, [pc, #256] ; 0x10c
00000008	e5943000	ldr r3, [r4]
0000000c	e3a00001	mov r0, #1 ; 0x1
00000010	e3a01000	mov r1, #0 ; 0x0
00000014		sum:
00000018		add r1, r0
0000001c		add r1, r1, r0
00000020		add r0, #1
00000024		add r0, r0, #1 ; 0x1
00000028		cmp r0, r3
0000002c		bgt end // 如果r0 > r3 则转 e
00000030		b sum // 转 sum
00000034		end:
00000038		str r1, [r5]
0000003c		stop:
00000040		b stop // 转 stop
00000044		stop:
00000048		b 0x2c (0x2c: stop)
0000004c		andeq r0, r0, r0

**Registers**

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	00000024
cpsr	800001d3 NZCVI SVC
spsr	00000000 NZCVI ?

**Disassembly (Ctrl-D)**

Address	Opcode	Disassembly
00000000	e59f4100	_start: ldr r4, [pc, #256] ; 0x108
00000004	e59f5100	ldr r5, result // r5=0x104 ldr r5, [pc, #256] ; 0x10c
00000008	e5943000	ldr r3, [r4] // 从 [0x100] 单元中读取
0000000c	e3a00001	mov r0, #1 ; 0x1
00000010	e3a01000	mov r1, #0 ; 0x0
00000014	e0811000	sum: add r1, r0
00000018	e2800001	add r0, r0, #1 ; 0x1
0000001c	e1500003	cmp r0, r3
00000020	ca000000	bgt end // 如果 r0 > r3 则转 e
00000024	ea000000	b sum // 转 sum
00000028	e5851000	str r1, [r5] // 将 r1 保存到 [0x10
0000002c	ea000000	end: stop: b 0x2c (0x2c: stop)
00000030	00000000	andeq r0, r0, r0
00000034	00000000	andeq r0, r0, r0
00000038	00000000	

(3) 在 ARMv7 汇编工具中运行计算费波那契数列程序  
汇编

**Registers**

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	00000024
cpsr	800001d3 NZCVI SVC
spsr	00000000 NZCVI ?

**Disassembly (Ctrl-D)**

Address	Opcode	Disassembly
00000000	e59fa224	_start: ldr r10, [pc, #548] ; 0x22c
00000004	e59fb224	ldr r11, result
00000008	e59a2000	ldr r2, [r10] ; 0x230
0000000c	e3a01000	mov r1, #0 ; 0x0
00000010	e28bc000	add r12, r11, #0 ; 0x0
00000014	e58c1000	str r1, [r12]
00000018	e3a01001	mov r1, #1 ; 0x1
0000001c	e28bc004	add r12, r11, #4 ; 0x4
00000020	e58c1000	str r1, [r12]
00000024	e28bc008	add r12, r11, #8 ; 0x8
00000028	e58c1000	str r1, [r12]
0000002c	e3a03002	mov r3, #2 ; 0x2
00000030	e3a04001	mov r4, #1 ; 0x1
00000034	e3a05001	mov r5, #1 ; 0x1
00000038	e0846005	loop: add r6, r4, r5
0000003c	e1a04005	mov r4, r5
00000040	e1a05006	mov r5, r6
00000044	e2833001	add r3, r3, #1 ; 0x1
00000048	e0837003	add r7, r3, r3
0000004c	e0870007	add r0, r7, r7
00000050	e080000b	add r0, r0, r11
00000054	e5806000	str r6, [r0]

**Messages**

Compiling..  
Code and data loaded from ELF executable into memory. Total size is 568 bytes.  
Assemble: arm-altera-eabi-as -mfloat-abi=soft -march=armv7-a -mcpu=cortex-a9 -mcpu=neon-fp16 -gdwarf2 -o work/asmrFqqw.s.o work/asmrFqqw.s  
Link: arm-altera-eabi-ld --script build\_arm.ld -e \_start -u \_start -o work/asmrFqqw.s.elf work/asmrFqqw.s.o  
Compile succeeded.

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	ZCVI SVC
spsr	00000000	ZCVI ?
s0	00000000	
s1	00000000	
s2	00000000	

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
00000200	00000000	result: 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
0000022c	00000100	9 ldr r10,=num andeq r0, r0, r0, LSL #2
00000230	00000200	10 ldr r11,=result
00000234	00000000	andeq r0, r0, r0, LSL #4 andeq r0, r0, r0