1、指令解析

要求：查看**可执行程序**的反汇编文件，在算术运算、移位运算、访存、分支跳转四种指令中，每种分别选出一条机器指令，并参照示例对它们进行解析。

示例：

指令 1：10094: c3418513 addi a0,gp,-972

机器码	16 进制	2 进制			
	c3418513	1100 0011 0100 0001 1000 0101 0001 0011			
opcode	001 0011	funct3	000	funct7	-
rd	0101 0（x10/a0）	rs1	0001 1（x3/gp）	rs2	-
imm	1100 0011 0100（-972 的补码）				
指令功能	(a0) ← (gp) - 972				

指令 1：10080: 54850513 addi a0,a0,1352

机器码	16 进制	2 进制			
	54850513	0101 0100 1000 0101 0000 0101 0001 0011			
opcode	001 0011	funct3	000	funct7	-
rd	0101 0（x10/a0）	rs1	0101 0（x10/a0）	rs2	-
imm	0101 0100 1000（1352）				
指令功能	(a0) ← (a0) +1352				

指令 2：10380: 00269693 slli a3,a3,0x2

机器码	16 进制	2 进制			
	00269693	0000 0000 0010 0110 1001 0110 1001 0011			
opcode	001 0011	funct3	001	funct7	-
rd	0110 1（x13/a3）	rs1	0110 1（x13/a3）	rs2	-
imm	0000 0000 0010				
指令功能	(a3) ← (a3) 左移两位				

指令 3：10150: fca42e23 sw a0,-36(s0)

机器码	16 进制	2 进制			
	fca42e23	1111 1100 1010 0100 0010 1110 0010 0011			
opcode	010 0011	funct3	010	funct7	-
rd	-	rs1	0100 0（x8/s0）	rs2	0 1010（x10/a0）
imm	1111 1101 1100（-36 的补码）				
指令功能	将 a0 寄存器中的数据存储到 s0 寄存器所指向的内存地址减去 36 字节的位置				

指令 4：

105e8: 02d50463 beq a0,a3,10610 <__register_exitproc+0x6c>

机器码	16 进制	2 进制			
	02d50463	0000 0010 1101 0101 0000 0100 0110 0011			
opcode	110 0011	funct3	000	funct7	-
rd	-	rs1	0101 0（x10/a0）	rs2	0 1101（x13/a3）
imm	0000 0001 0100				
指令功能	如果 a0 中的值等于 a3 中的值，跳转到 10610 <__register_exitproc+0x6c>				

2、汇编程序实验结果

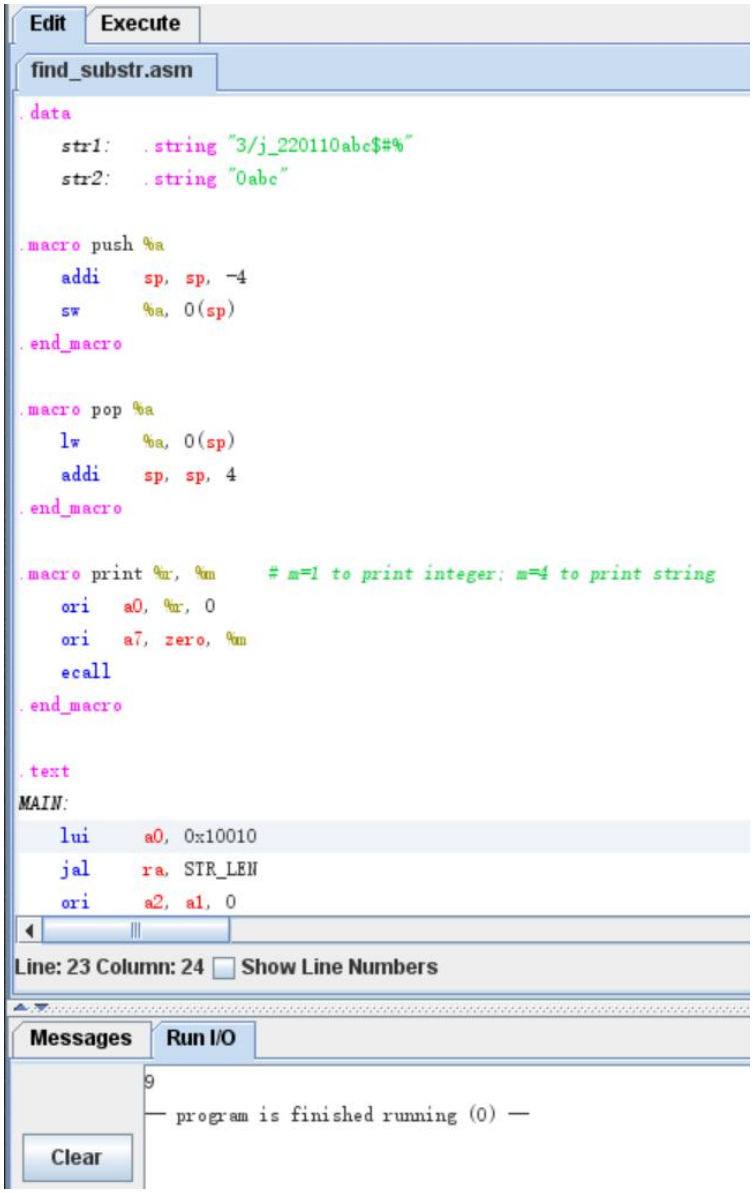
要求：自行构造 5 个测试用例，对自行编写的字符串匹配汇编程序进行测试，并记录运行结果：

【用例 1】母串：用任意字符将学号填充至 16 位；子串：学号后 4 位。
【用例 2】母串包含子串的前几个字符，但不构成匹配。
【用例 3】母串不包含子串，但将母串首尾衔接后包含子串。
【用例 4】母串包含多个子串。
【用例 5】母串包含子串的逆序串。

示例：

【用例 1】母串：3/j_220110abc$#%　　子串：0abc

运行结果：



【用例 1】母串：220110401KYC11　　子串：0401

运行结果：

Edit　Execute

riscv1.asm

```
1  .data
2      str1: .string "220110401KYC11"     # 母串
3      str2: .string "0401"               # 子串
4
5  .macro push %a
6      addi     sp, sp, -4
7      sw       %a, 0(sp)
8  .end_macro
9
10 .macro pop %a
11     lw       %a, 0(sp)
12     addi     sp, sp, 4
13 .end_macro
14
15 .macro print %r, %m
16     ori      a0, %r, 0
17     ori      a7, zero, %m
18     ecall
19 .end_macro
20
21 .text
22 .globl MAIN
23 MAIN:
24     lui      a1, %hi(str1)
25     addi     a1, a1, %lo(str1)        # 加载母串地址到a1
26
27     lui      a2, %hi(str2)
28     addi     a2, a2, %lo(str2)        # 加载子串地址到a2
29     addi     a3, zero, 16             # 母串长度
```

Line: 3 Column: 23　Show Line Numbers

Messages　Run I/O

```
5
-- program is finished running (0) --
```

Clear

【用例 2】母串：asdhhlkan6dlkn　　子串：n6d2

运行结果：

Edit　Execute

riscv1.asm

```
1  .data
2      str1: .string "asdhhlkan6dlkn"     # 母串
3      str2: .string "n6d2"               # 子串
4
5  .macro push %a
6      addi    sp, sp, -4
7      sw      %a, 0(sp)
8  .end_macro
9
10 .macro pop %a
11     lw      %a, 0(sp)
12     addi    sp, sp, 4
13 .end_macro
14
15 .macro print %r, %m
16     ori     a0, %r, 0
17     ori     a7, zero, %m
18     ecall
19 .end_macro
20
21 .text
22 .globl MAIN
23 MAIN:
24     lui     a1, %hi(str1)
25     addi    a1, a1, %lo(str1)        # 加载母串地址到a1
26
27     lui     a2, %hi(str2)
28     addi    a2, a2, %lo(str2)        # 加载子串地址到a2
29     addi    a3, zero, 16          # 母串长度
```

Line: 4 Column: 1　Show Line Numbers

Messages　Run I/O

```
-1
-- program is finished running (0) --
```

Clear

【用例 3】母串：agushajsajbcshdh　　子串：dhag

运行结果：

Edit　Execute

riscv1.asm

```
1   .data
2       str1: .string "agushajsajbcshdh"    # 母串
3       str2: .string "dhag"                # 子串
4
5   .macro push %a
6       addi    sp, sp, -4
7       sw      %a, 0(sp)
8   .end_macro
9
10  .macro pop %a
11      lw      %a, 0(sp)
12      addi    sp, sp, 4
13  .end_macro
14
15  .macro print %r, %m
16      ori     a0, %r, 0
17      ori     a7, zero, %m
18      ecall
19  .end_macro
20
21  .text
22  .globl MAIN
23  MAIN:
24      lui     a1, %hi(str1)
25      addi    a1, a1, %lo(str1)       # 加载母串地址到a1
26
27      lui     a2, %hi(str2)
28      addi    a2, a2, %lo(str2)       # 加载子串地址到a2
29      addi    a3, zero, 16            # 母串长度
```

Line: 3 Column: 43　Show Line Numbers

Messages　Run I/O

```
-1
-- program is finished running (0) --
```

Clear

【用例 4】母串：22011040122011　　子串：2201

运行结果：

Edit　Execute

riscv1.asm

```
1  .data
2      str1: .string "22011040122011"     # 母串
3      str2: .string "2201"               # 子串
4
5  .macro push %a
6      addi    sp, sp, -4
7      sw      %a, 0(sp)
8  .end_macro
9
10 .macro pop %a
11     lw      %a, 0(sp)
12     addi    sp, sp, 4
13 .end_macro
14
15 .macro print %r, %m
16     ori     a0, %r, 0
17     ori     a7, zero, %m
18     ecall
19 .end_macro
20
21 .text
22 .globl MAIN
23 MAIN:
24     lui     a1, %hi(str1)
25     addi    a1, a1, %lo(str1)        # 加载母串地址到a1
26
27     lui     a2, %hi(str2)
28     addi    a2, a2, %lo(str2)        # 加载子串地址到a2
29     addi    a3, zero, 16          # 母串长度
```

Line: 2 Column: 34　Show Line Numbers

Messages　Run I/O

```
0
-- program is finished running (0) --
```

Clear

【用例 5】母串：jssss7890111111　　子串：0987

运行结果：

Edit　Execute

riscv1.asm

```
1   .data
2       str1: .string "jssss7890111111"     # 母串
3       str2: .string "0987"                # 子串
4
5   .macro push %a
6       addi    sp, sp, -4
7       sw      %a, 0(sp)
8   .end_macro
9
10  .macro pop %a
11      lw      %a, 0(sp)
12      addi    sp, sp, 4
13  .end_macro
14
15  .macro print %r, %m
16      ori     a0, %r, 0
17      ori     a7, zero, %m
18      ecall
19  .end_macro
20
21  .text
22  .globl MAIN
23  MAIN:
24      lui     a1, %hi(str1)
25      addi    a1, a1, %lo(str1)       # 加载母串地址到a1
26
27      lui     a2, %hi(str2)
28      addi    a2, a2, %lo(str2)       # 加载子串地址到a2
29      addi    a3, zero, 16            # 母串长度
```

Line: 3 Column: 24　Show Line Numbers

Messages　Run I/O

```
-1
-- program is finished running (0) --
```

Clear

3、思考与讨论

（1）用自己的语言描述子程序的工作流程。

1. 保护现场：将 ra 和 t1 寄存器的值压入堆栈。

2. 初始化：将 t3（位置索引）设为-1，t5（外循环索引 i）设为 0。

3. 进入外层循环（LOOP）：

首先检查 i 是否大于等于母串长度，如果是，则跳出循环；如果不是，接着计算母串当前检查位置的地址。

4. 进入内层循环（INNER_LOOP）：

首先检查 j 是否大于等于子串长度，如果是，匹配成功，跳转到 MATCH_FOUND；如果不是，则加载母串和子串当前字符，比较字符是否相等，如果不相等，跳转到 UPDATE_I；如果相等，则移动到下一个字符并继续内循环。

5. 匹配成功（MATCH_FOUND）：设置位置索引 t3 为当前外循环索引 i，跳出主循环。

6. 更新外循环索引（UPDATE_I）：将 i 加 1，并继续外部循环。

7. 子程序执行完毕，退出（EXIT）：首先检查 t3 是否等于 s10（15），如果是，表示找不到子串，索引设为-1；如果不是，则将 t3 的值传递给 a0 作为 FUNC 的返回值。并恢复现场（从堆栈中弹出 t1 和 ra），最后跳转回到主程序（MAIN）。

（2）进入子程序时为何要保护现场，以及子程序返回前为何要恢复现场？

进入子程序时保护现场是为了保持调用者（主程序）的状态，保护主程序已在寄存器中存储的值，避免出现数据的丢失或错误；同时，防止子程序返回后造成主程序执行出错，保证子程序的独立性。

子程序返回前恢复现场是为了恢复调用者（主程序）进入子程序前的状态，以确保主程序可以继续正确执行；同时，由于进入子程序时进行了保护现场的操作（返回地址寄存器（ra）通常在调用子程序之前保存调用者的返回地址）。子程序执行完毕后，需要跳转回调用者的返回地址。如果子程序不恢复现场（保存并恢复 ra 的值），程序可能无法正确返回调用者（主程序），导致程序跳转到错误的地址。

（3）试对比分析编译器生成的汇编程序与自行编写的汇编程序各有什么异同。

1. 相同点：

①功能相同：都是在母串中查找子串的起始位置，并返回索引；

②两者都通过保护和恢复寄存器现场，确保子程序执行不会影响调用者（主程序）的状态；

③都涉及字符串操作和循环遍历字符比较。

2.不同点：

①编译器生成的汇编程序更加结构化和优化——使用了更多的跳转标签（如.L2, .L3, .L5 等）来组织代码，也在保护和恢复现场时使用了更多的局部变量（如 s0）来简化堆栈操作；

②编译器生成的汇编程序在初始化时，使用了更多的局部变量来存储函数参数和局部变量，同时，对堆栈空间的管理也更加严谨和系统（预先分配了足够的空间）；

③编译器生成的汇编程序在处理循环和条件跳转时，更加精细和优化（如在比较字符时，使用了更多的临时变量和条件跳转标签，确保了程序执行的效率和正确性）；

④编译器生成的汇编程序在退出时，通过多个标签和条件检查确保返回值的正确性和寄存器的恢复。而自行编写的汇编程序在退出时相对简单，没有那么复杂、全面。