1.1 请解释：计算机和计算机系统。

“计算机”指的是“现代计算机”，全名为“通用电子数字计算机（General-Purpose Electronic Digital Computer）”。核心部件包括 CPU 和存储器/内存。

计算机系统由硬件和软件两部分组成，硬件包括处理器、存储器和外部设备等，软件包括程序和文档。

1.2 有两种计算机 A 和 B：A 有乘法指令，而 B 没有；二者都有加法和减法指令；在其余方面，二者都相同。那么，对于 A 和 B，哪种计算机可以解决更多的问题？

所有的计算机（无论大小、快慢），如果给予足够的时间和足够的存储器，都可以做相同的计算。换句话说，所有的计算机都能做几乎完全相同的事情，只是计算速度上有差别。

B 计算机可以通过加/减法指令，编程实现乘法运算。

1.3 现代计算机为什么采用数字设计，而非模拟机？

模拟机的缺点是很难提高精度，采用数字机，只要增加数字位数就可以提高精度，非常容易实现。

1.4 现代计算机的核心部件有哪些？分别具有什么功能？

核心处理部件是 CPU（Central Processing Unit，中央处理器）。处理器完成的最重要的工作是：执行指令，即执行加法、乘法等计算工作。

核心部件除 CPU 外，还有存储器（Memory），又称内存。程序存储在存储器里，CPU 负责完成两项工作：指挥信息的处理和执行信息的实际处理。

1.5 当你将计算机升级（如更换 CPU）后，原来的软件（如操作系统）还能够使用吗？

能用理由：计算机升级升的是硬件，软件存储在磁盘中，只要升级的硬件提供与原有硬件相同的工作方式和功能，软件还是能够正常工作的。

不能用理由：计算机中的硬件升级前与升级后所需要的驱动可能不一样，软件无法通过原有方法调用底层硬件，因此也就不能使用了。

1.6 当你的计算机中需要安装软件时，这些软件在安装之前是以什么形式存在的？是高级语言还是目标机器 ISA 兼容的机器语言？

软件存在的方式可能是多种多样的：源代码或目标代码。源代码包括汇编语言和高级语言等。目标代码包括机器语言、解释型源代码等。

1.7 请解释：指令集结构是计算机硬件和软件之间的接口。

从计算机硬件设计者的视角来看，指令集结构是硬件设计的依据；从程序设计者的角度来看，指令集结构指明了在一台机器上编写程序时所要注意的全部信息。

1.8 你对计算机系统中的哪些抽象层比较熟悉？熟悉程度如何？

（略）

1.9 对于以下 C 语言中的问题，分别涉及计算机系统中的哪一个抽象层？

1）“sum=sum++ +i”类似的表达式，不是好的编程风格，为什么？

一条 C 语句需要经过编译，被编译为多条计算机指令，这就涉及到计算机的指令集结构相关知识（指令集结构层）；此外，编译结果与选择使用的编译器也有相关性（语言处理层）。

2）为什么 switch 语句中的表达式只能是整型，case 必须是常量？switch 语句与级联的 if-else 到底有什么区别？

一条 C 语句需要经过编译，被编译为多条计算机指令，这就涉及到计算机的指令集结构相关知识（指令集结构层）；对于 switch 语句与级联的 if-else，编译器编译的结果不同（语言处理层）。

1.10 高级语言的可移植性是指其代码是否可以在不同的目标机器（即不同的计算机系统）上运行。请问，C 语言和 Java 语言的可移植性如何？提示：Java 语言的翻译、执行过程为，首先翻译成字节码文件，然后在 Java 虚拟机上执行。

C 语言程序经过编译，得到目标机器的机器码，不能在其他计算机系统（不同的指令集结构、操作系统）上执行；Java 语言程序经过编译，得到与目标机器无关的字节码，在具体机器上执行时，再通过 Java 虚拟机翻译为目标机器的机器码执行，因此，可移植性好。

1.11 上机实践：

1）编写一个 C 程序：计算 n 的阶乘；并测试当 n 取值为多少时，计算结果出错？假设 n 是 int 类型变量；

程序样例：

```
#include <stdio.h>

int main(){
    int n;
    int result=1;
    int i=1;
    scanf("%d",&n);

    for(i=1;i<=n;i++){
      result=result*i;
      printf("%d ",result);
    }
}
```

以 32 位 int 为例，当输入 13 时，结果出错。

2）编写一个 C 程序：计算斐波那契数列的第 n 项；并测试当 n 取值为多少时，计算结果出错？假设 n 是 int 类型变量；

程序样例：

```
#include <stdio.h>

int main(){
    int n;
    int result1=1;
    int result2=1;
    int i=1;
    scanf("%d",&n);
    int temp;
```

```
    for(i=1;i<=n;i++){
      temp=result2;
      result2=result1+result2;
      result1=temp;

      printf("%d ",result2);
    }
}
```

以 32 位 int 为例，当输入 45 时，结果出错。

3）在 PC 机上，编写一个实现排序的 C 程序，实现排序的 Sort 函数如图 1.3 所示。使用 GCC 编译器的 gcc -S *.c 命令，将该程序编译为 X86 汇编文件，对比 C 程序和 X86 汇编程序；

（略）

4）在 PC 机上，编写两个 C 程序，Fibonacci 函数分别使用递归和循环实现斐波那契数列的计算。使用 GCC 编译器的 gcc -S *.c 命令，将两个程序分别编译为汇编文件，将两个汇编程序做对比。

（略）

2.1 请回答如下问题：

1）对于 8 位二进制原码、反码和补码整数类型，能够表示的最大的正数分别是多少？请分别以二进制和十进制形式写出结果；

2）对于 8 位二进制原码、反码和补码整数类型，能够表示的最小的负数分别是多少？请分别以二进制和十进制形式写出结果；

3）对于 *n* 位二进制原码、反码和补码整数类型，能够表示的最大的正数分别是多少？

4）对于 *n* 位二进制原码、反码和补码整数类型，能够表示的最小的负数分别是多少？

	原码		反码		补码	
8 位最大正数	0111 1111	127	0111 1111	127	0111 1111	127
8 位最小负数	1111 1111	-127	1000 0000	-127	1000 0000	-128
n 位最大正数	$2^{n-1}-1$		$2^{n-1}-1$		$2^{n-1}-1$	
n 位最小负数	$-2^{n-1}+1$		$-2^{n-1}+1$		-2^{n-1}	

2.2 将下列二进制数转化为十进制数，假设此二进制数为补码整数。

1）0111

2）1110

3）11111111

4）10000000

0111	7
1110	-2
11111111	-1
10000000	-128

2.3 将下列十进制数转化为 8 位二进制补码整数。

1）-86

2）85

3）-127

4）127

-86	10101010
85	01010101
-127	10000001
127	01111111

2.4 如果二进制补码整数最后一位是 0，表明该数是偶数，如果最后两位是 00，则表明该数的什么特点？

该数是 4 的倍数。

2.5 做下列二进制加法运算，给出二进制形式的结果：

1）1010+0101

2）0001+1111

3）1110+0001

4）0111+0110

1010+0101	1111

0001+1111	0000
1110+0001	1111
0111+0110	1101

2.6 对于一个二进制数，如果向右移一位，则意味着进行了什么运算？

n/2

2.7 做下列二进制补码整数加法运算，给出十进制形式的结果，并判断是否产生溢出。

1）1101+01010101

2）0111+0101

3）11111111+01

4）01+1110

5）0111+0001

6）1000+11

7）1100+00110011

8）1010+101

	结果	是否溢出
1)	82	
2)	-4	是
3)	0	
4)	-1	
5)	-8	是
6)	7	是
7)	47	
8)	7	是

2.8 请给出下列十进制数的 IEEE 754 32 位浮点数表示，结果以十六进制表示。

1）32.9375

2）$-32\frac{45}{128}$

3）-2^{-140}

4）65536

	浮点数	H
32.9375	0 10000100 0000 0111 1000 0000 0000 000	4203 C000
$-32\frac{45}{128}$	1 10000100 0000 0010 1101 0000 0000 000	C201 6800
-2^{-140}	1 00000000 0000 0000 0000 0100 0000 000	8000 0200
65536	0 10001111 0000 0000 0000 0000 0000 000	4780 0000

2.9 请给出下列 IEEE 浮点数的十进制数表示：

1）0 00000001 00000000000000000000000

2）0 00000000 00000000010000000000000

3）1 11111011 00000000000000000000000

4）1 10000001 10101000000000000000000

5）0　01111101　01010100000000000000000

浮点数	十进制
0 00000001 00000000000000000000000	2^{-126}
0 00000000 00000000010000000000000	2^{-136}
1 11111011 00000000000000000000000	-2^{124}
1 10000001 01010000000000000000000	-6.625 or -53/8
0 01111101 01010100000000000000000	0.33203125 or 85/256

2.10 如下代码分别输出哪些内容？

1）printf ("%c\n", 13 + 'A');

2）printf ("%x\n", 130);

答案：1) 输出字母N

2) 输出十六进制数：82

2.11 请解释如下代码段的作用。

```
char nextChar;
int x;
scanf ("%c", &nextChar);
printf ("%d\n", nextChar);
scanf ("%d", &x);
printf ("%c\n", x);
```

答案：输入一个字符，输出其 ASCII 的数值；

输入一个整数，输出其作为 ASCII 码代表的字符。

2.12 求满足条件 1+2+3+…+n≤2147483647 的最大整数 n，对于如下程序段，请解释：为什么会出现无限循环？

```
int n=1, sum=0;
while (sum <= 2147483647)
{
    sum+=n;
    n++;
}
printf ("n=%d\n", n-1);
```

答案：n 的值为 65536 时，计算 sum 溢出，sum 小于 0，则循环条件仍然满足，不会跳出循环。

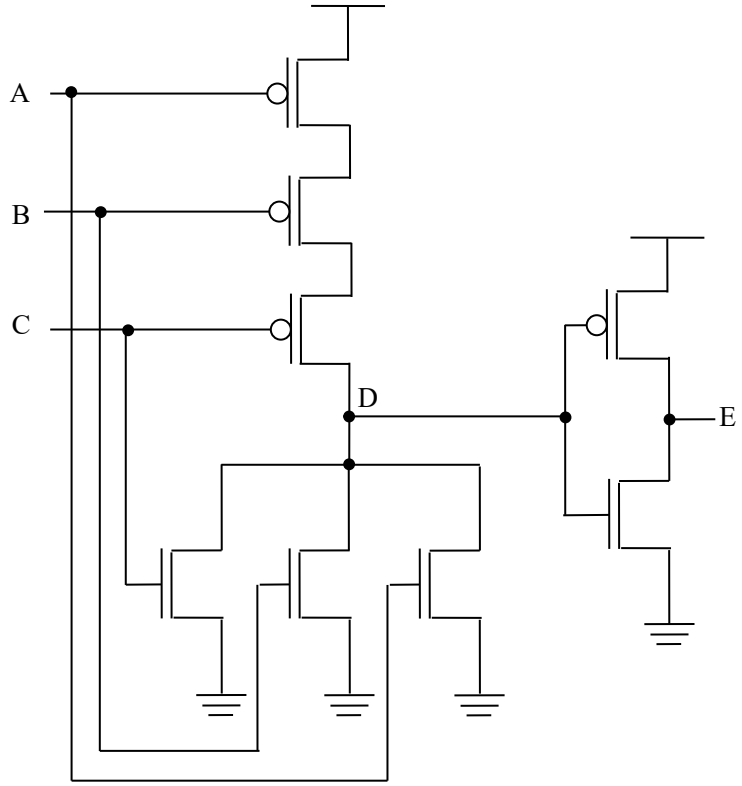
2.13 使用“printf("%.16f\n", 3.14);”语句，输出 3.14 的值，为什么输出结果是“3.1400000000000001”，即小数末尾为什么会出现一个 1？提示：IEEE 754 64 位浮点数标准的分数域为 52 位。

答案：在计算机中，3.14 的二进制表示为 0　10000000000　1001 0001 1110 1011 1000 0101 0001 1110 1011 1000 0101 0001 1111，与 float 类型相比，拥有更高的精度，但是仍然存在误差。计算到小数点后 16 位的结果是 3.1400000000000001。

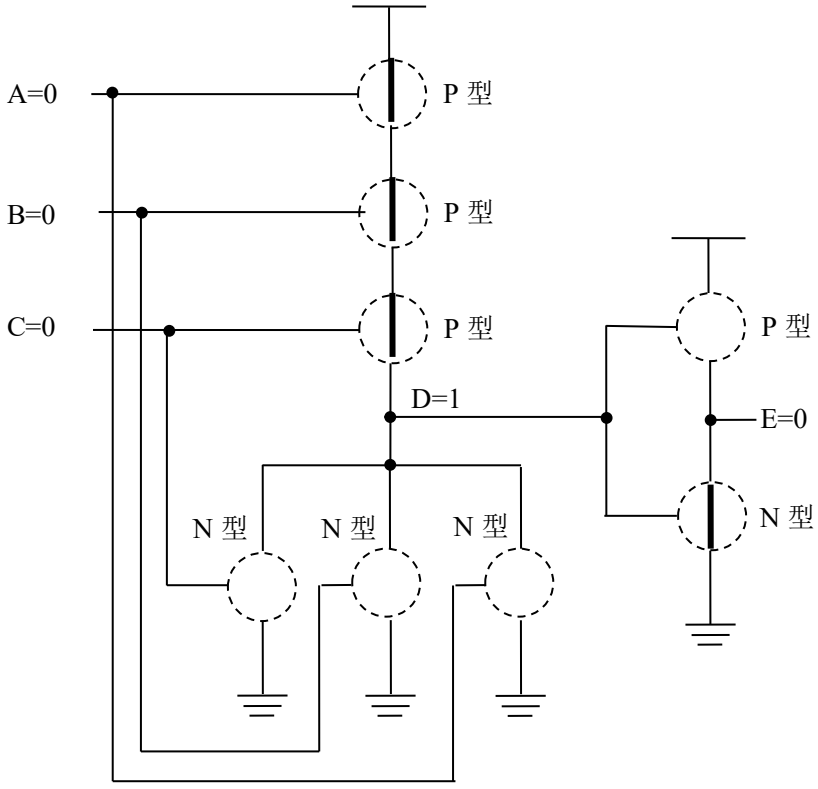
2.14 上机实践：在你的计算机上，编写一个 C 程序，输出 int 类型整数的最大值和最小值。

```c
#include <stdio.h>

int main(){
	int n=1;

	while(n>0){
		n++;
	}
	printf("%d\n",n);	//最小値
	printf("%d\n",n-1);	//最大値
}
```

3.1 1）请分别画出三个输入的与门和三个输入的或门的晶体管级电路图。
以或门为例：



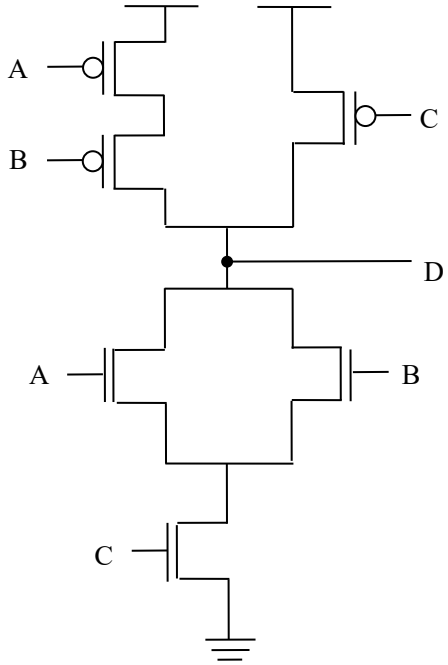
2）对于如下输入，请分别在其与门和或门晶体管级电路图中标出其表现。
I. A=0, B=0, C=0 ； II. A=0, B=0, C=1； III. A=1, B=1, C=1



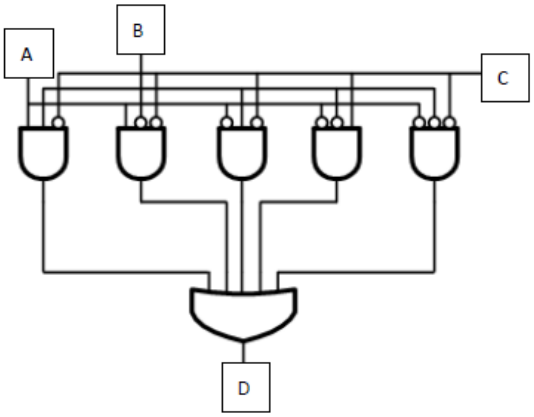
以 A=0,B=0,C=0 为例

3.2 1）给出下图所示的晶体管级电路的真值表。

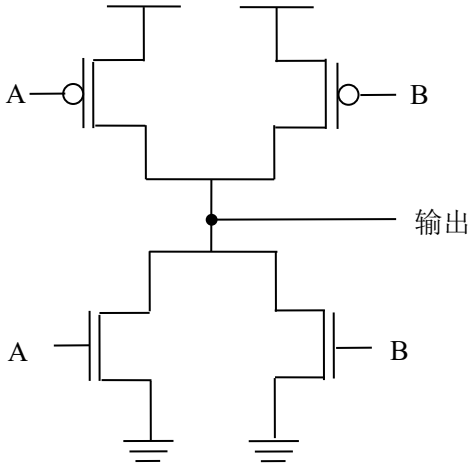
2）使用与、或、非门，给出该真值表的门级电路图。



A	B	C	D
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

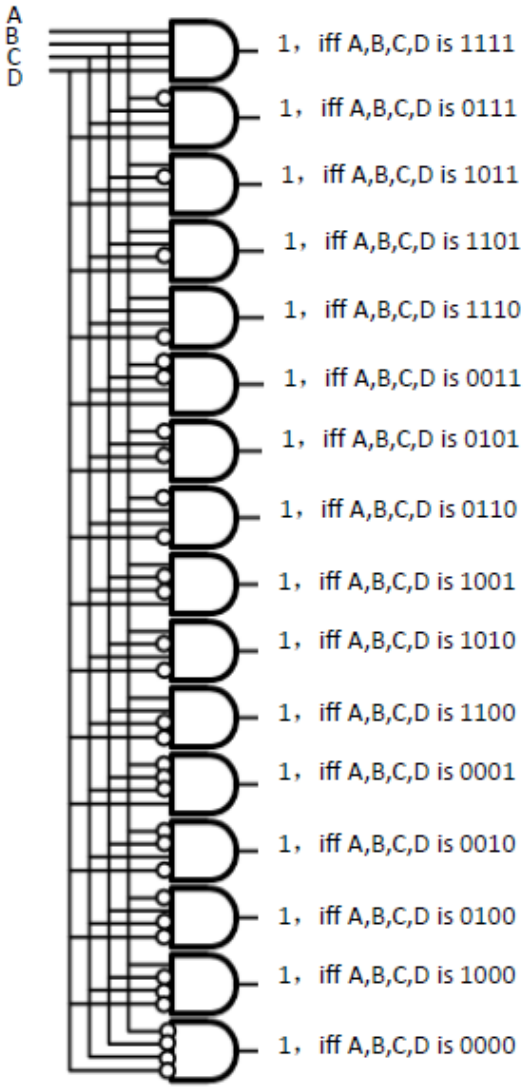


3.3 如下图所示的电路有一个缺陷，请指出该缺陷。

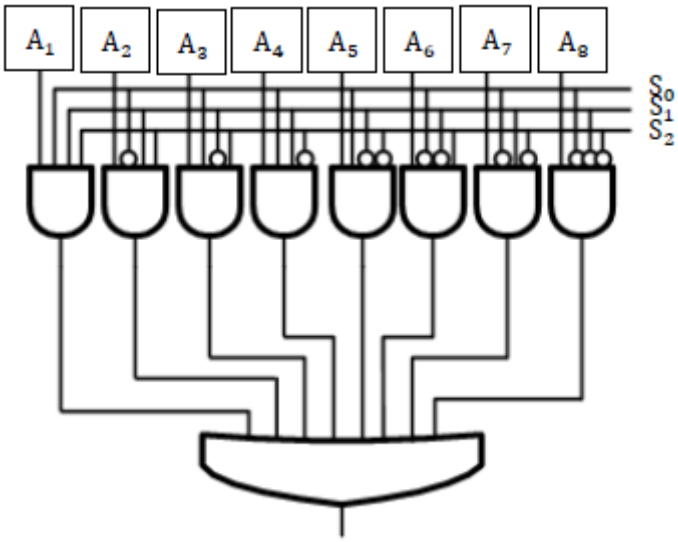


缺陷：若 A=0,B=1 或 A=1,B=0，则同时接到了电源正极和地，出现短路。

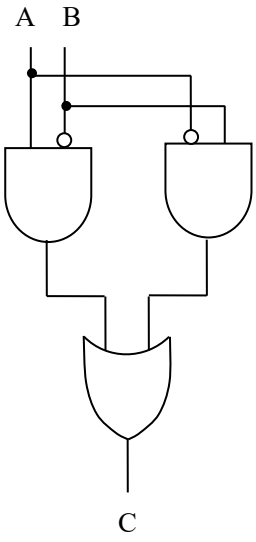
3.4 请画出有 4 个输入的译码器的门极电路图，并注明各输出为 1 的条件。



3.5 请画出有 8 个输入的多路选择器的门极电路图。

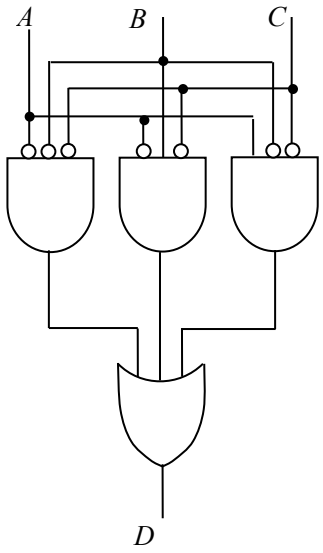


3.6 使用与、或、非门，给出异或函数的门级电路图。

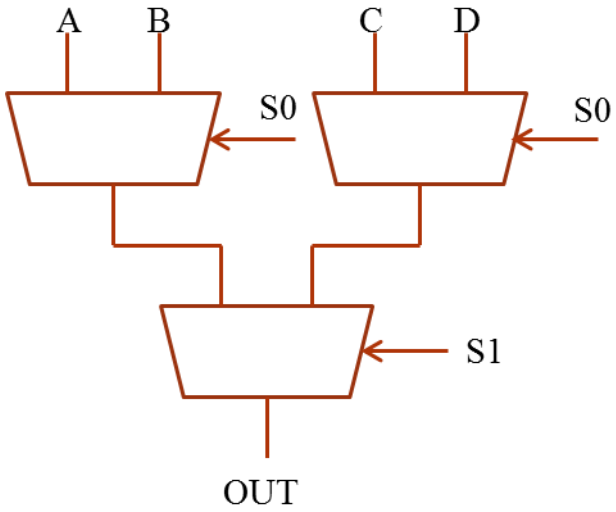


3.7 对于如下真值表，请使用 3.4.4 节给出的算法（可编程逻辑阵列），生成其门级逻辑电路。

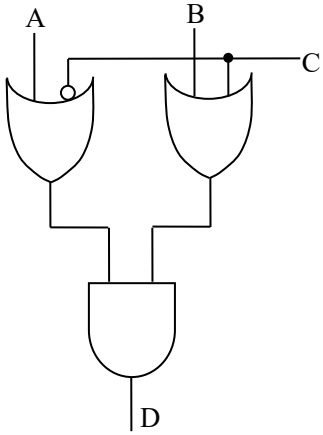
A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0



3.8 只使用 2 选 1 的多路选择器，就可以实现 4 选 1 的多路选择器，给出其电路图。



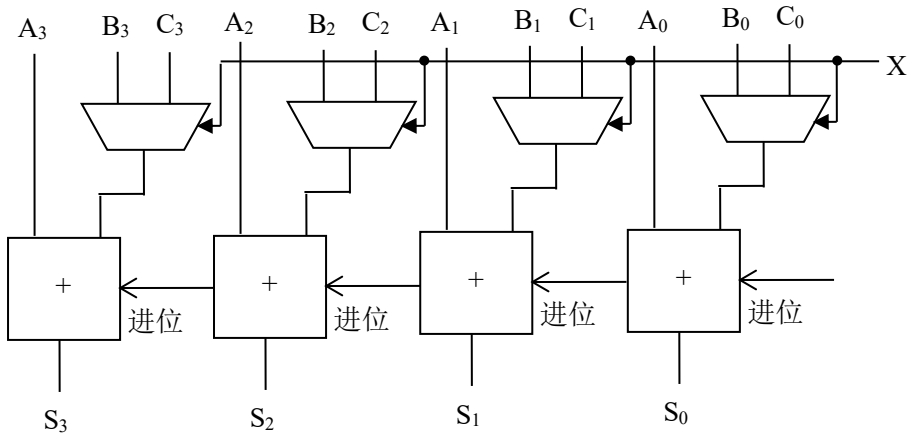
3.9 根据下图所示的逻辑电路图，写出相应的真值表。



A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

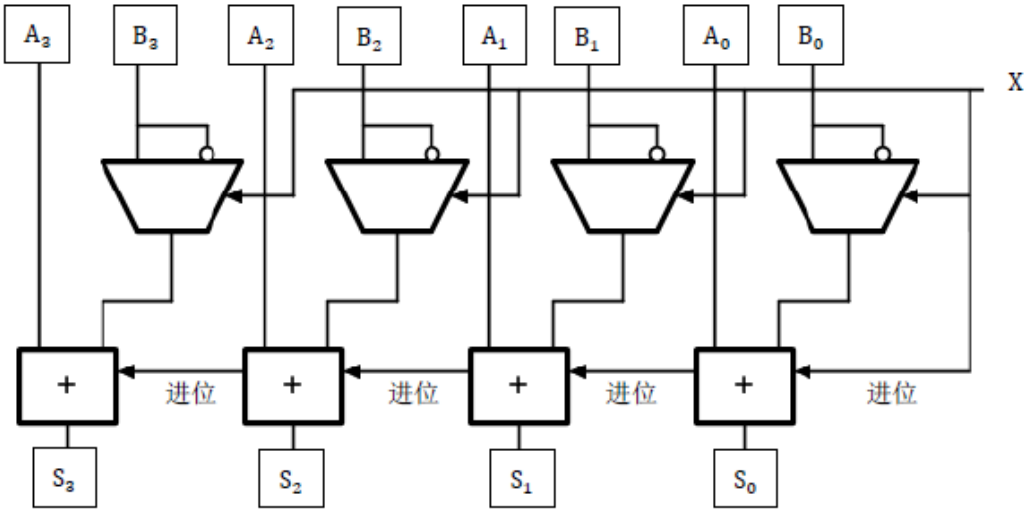
3.10 1）下图中的每个矩形都表示一个全加法器，当 X=0 和 X=1 时，电路的输出分别是什么？

2）在该电路图的基础上，构建一个可以实现加法/减法运算的逻辑电路图。提示：*X*=0 时，*S* 的值是 *A*+*B* 的值；*X*=1 时，*S* 的值是 *A*-*B* 的值。



1) 答：分别为 S＝A＋B，S＝A＋C

2) 答：当 X=0 时，计算 A+B；当 X=1 时，计算 A-B



3.11 一个逻辑结构的速度与从输入到达输出，需传递经过的逻辑门的最长路径有关。假设与、或、非门都被计为一个门延迟，例如，两个输入的译码器的传递延迟等于 2（参照

图 3.9），这是因为有些输出需经过两个门的传递。

1）两个输入的多路选择器的传递延迟是多少（参照图 3.10）？

2）1 位的全加法器的传递延迟是多少（参照图 3.12b）？

3）4 位的全加法器的传递延迟是多少（参照图 3.12c）？

4）32 位的全加法器的传递延迟是多少？

1) 答：3

2) 答：3

3) 答：4 * 3 = 12

4) 答：32 * 3 = 96

3.12 设计一个 1 位的比较器，该比较器电路有两个 1 位的输入 A 和 B，有 3 个 1 位的输出 G（greater，大于）、E（equal，等于）和 L（less，小于）。当 A＞B 时，G 为 1，否则，G 为 0；当 A＝B 时，E 为 1，否则，E 为 0；当 A＜B 时，L 为 1，否则，L 为 0。

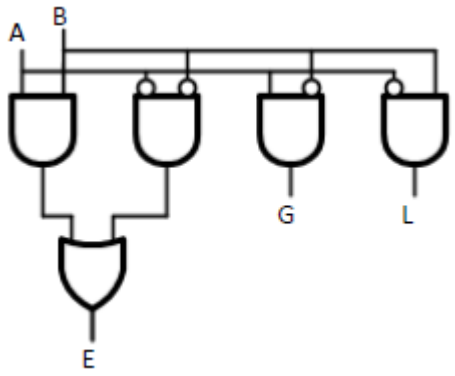
1）给出此 1 位比较器的真值表。

2）使用与、或、非门实现此比较器电路。

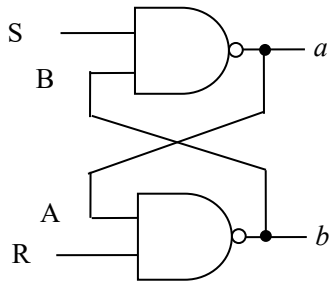
1) 真值表

A	B	G	E	L
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

2) 比较器



3.13 参照下图，回答问题：



1）当 S 和 R 都为 1 时，此逻辑电路的输出是什么？
2）如果 S 从 1 转换到 0，输出是什么？
3）此逻辑电路是存储元件吗？

1) 答：A=0/a=0 时 B=1/b=1, A=1/a=1 时 b=0/B=0，即 a 的值可以为 0，也可以为 1；
2) 答：a 的值为 1，b 为 0；
3) 答：是

3.14 某个计算机有 4 个字节的寻址能力，访问其存储器的一个单元需要 64 位，该存储器的大小是多少（以字节为单位）？此存储器共存储多少位？
答：$2^{64} * 4 = 2^{66}$(byte)，　$2^{66} * 8 = 2^{69}$(bit)

3.15 8 位被称为一个字节（byte），4 位被称为一个单元组（nibble）。一个字节可寻址的存储器使用 14 位的地址，那么，此存储器共存储了多少单元组？
答：$2^{14} * 8 / 4 = 2^{15}$(nibble)

3.16 对于图 3.29 所示的 4×2 位大小的存储器，回答以下问题：
1）如果向单元 3 存储数值，A[1:0]和 WE 必须被设置为什么值？
2）如果将此存储器的单元数目从 4 增长到 10，需要多少条地址线？存储器的寻址能力是否发生变化？

1) 答：A[1:0]必须被设置为 11，WE 必须被设置为 1。
2) 答：需要 4 条地址线，存储器的寻址能力不变。

3.17 设计一个简单的交通灯控制器。与 3.6.2 节的系统背景类似，在东西向大街和南北向大街相交的十字路口，设有一组交通灯 1、2、3 和 4。当没有行人时，在第一个时钟周期，1 号灯和 4 号灯亮；下一周期，2 号灯和 3 号灯亮；然后，重复这个顺序。与 3.6.2 节不同的是，如果有行人按下按钮，1 号灯和 3 号灯总是立即亮起，而不是等到当前时钟周期结束。
1）画出状态图。
2）写出输出函数表和状态转换表。
3）给出此交通灯控制器的时序逻辑电路。

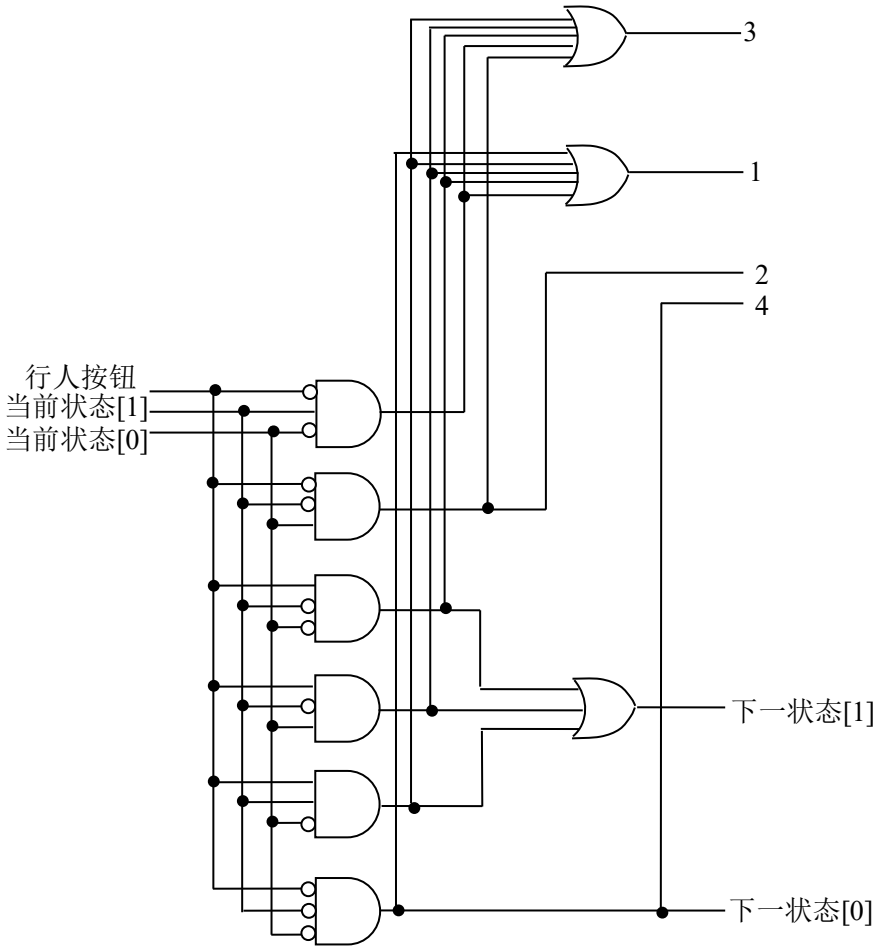
1）与图 3.22 相同
2)

输出函数表

行人按钮	当前状态[1]	当前状态[0]	1 灯	2 灯	3 灯	4 灯
0	0	0	1	0	0	1
0	0	1	0	1	1	0
0	1	0	1	0	1	0
1	0	0	1	0	1	0
1	0	1	1	0	1	0
1	1	0	1	0	1	0

状态转换表与 3.6.2 节相同

3）（寄存器与 3.6.2 节相同）



3.18 上机实践：编写 C 程序，从键盘输入一个十进制整数，输出该数值的二进制表示（32 位）。例如，如果输入是 10，则输出为 00000000000000000000000000001010。提示：使用按位运算符。

```
#include <stdio.h>

int main(){
    int n;
    int i=32;
```

```
	int mask=0x80000000;

	scanf("%d", &n);
	for(i=32;i>=1;i--){
		if((n&mask)!=0){
			putchar('1');
		}else{
			putchar('0');
		}
		n=n<<1;
	}
}
```

4.1 下图显示了某内存的部分情况，请回答以下问题。注意：地址和数据给出的均为二进制表示。

地址	数据
……	……
00001011	0000 0011
00001010	0010 1000
00001001	0110 0100
00001000	0001 0011
00000111	0100 0000
00000110	1000 0000
00000101	1100 0000
00000100	0100 0010
00000011	0111 1111
00000010	1000 0000
00000001	1111 1110
00000000	0000 0000

1）单元 0 和单元 4 包含的二进制数值分别是什么？

2）每个单元内的二进制数值可以以不同的方式被解释，如可以表示为无符号整数、补码整数、浮点数、ASCII 码等。

I. 将单元 0 和单元 1 解释为一个 8 位补码整数，请以十进制形式写出结果；

II. 将单元 2 和单元 3 解释为一个 8 位无符号整数，请以十进制形式写出结果；

III. 将单元 4 解释为 ASCII 码值；

IV. 将单元 4、5、6 和 7 解释为一个 IEEE 浮点数（32 位），请分别以大尾端和小尾端的方式解释，给出十进制结果。

3）存储单元的内容也可以是一条指令，将单元 8、9、10 和 11 解释为一条 RV32I 指令，请分别以大尾端和小尾端的方式解释，该指令表示什么？

4）一个二进制数值也可以被解释为一个存储单元的地址，如果存储在单元 11 中的数值是一个地址，它指的是哪个单元？那个单元里包含的二进制数值是什么？

答：

(1) 单元 0：00000000　单元 4：01000010

(2) I. 单元 0： 0　　单元 1：-2

II. 单元 2: 128　　单元 3： 127

III. 单元 4： B

IV. 大尾端：0x42C08040, 96.25048828125

小尾端：0x4080C042, 4. 0234689712524414

(3) 大尾端：0x13642803, lw x16, 0x136(x8) ，注：为便于解释，本章提供的答案都是汇编语言指令（见第 5 章）

小尾端：0x03286413, ori x8, x16, 0x32

(4) 单元 3，01111111

4.2 假设一个 16 位的指令采取如下格式：

操作码	源寄存器	目标寄存器	补码整数

如果共有 12 个操作码和 8 个寄存器，那么，“补码整数”能够表示的数值范围是什么？

答：操作码需 4 位，寄存器需 3 位

16-4-3-3=6

-2^5~2^5-1，即-32~31

4.3 假设一个 32 位的指令采取如下格式：

操作码	目标寄存器	源寄存器 1	源寄存器 2	无符号整数

如果共有 200 个操作码和 60 个寄存器，“无符号整数”能够表示的最大数是多少？

答：操作码需 8 位，寄存器需 6 位

32-8-6-6-6=6

2^6-1，即 63

4.4 对于 RV32I 的 I-类型指令，请回答以下问题：

1）立即数的范围是多少？

2）如果重新定义 RV32I 的 I-类型指令，使得立即数表示无符号整数，那么，立即数的范围是多少？

3）如果重新定义 RV32I 指令集，将寄存器的数量从 32 个降低到 16 个，那么，在 I-类型的指令中能够表示的立即数的最大值是多少？假设立即数仍表示补码整数。

答：

1）-2^{11}~2^{11}-1

2）0~2^{12}-1

3）32-7-3-4-4=14，2^{13}-1

4.5 对于 RV32I 的 R-类型指令，如果重新定义 RV32I 指令集，将寄存器的数量从 32 个增加至 128 个，是否可行？

答：32-7-3-7=15，每个寄存器最多使用 5 位表示，最多只能表示 32 个寄存器，不能增加。

4.6 假设某计算机的内存包括 65536 个单元，每个单元包含 16 位的内容，请回答以下问题：

1）需要多少位表示地址？

2）假设每条指令都由 16 位组成，操作码占 4 位，寄存器占 3 位。其中一条指令与 RV32I 的 jal 指令工作机制类似，那么，PC 相对偏移范围是多少？

答：

1）16

2）16-4-3=9，-2^9~2^9-1

4.7 假设寄存器 x5 中存储的位组合的最右边两位有特殊的重要性，请使用一条 RV32I 指令，将这两位数值读取出来。提示：读取出的数值的左边 30 位均为 0，右边两位为 x5 中的位组合的最右边两位。

答：

andi x6, x5, 3

4.8 1） 将 x5 乘 8，并将结果存于 x6 中，一条 RV32I 指令（图 4.4 给出的指令）可以实现吗？

2）将 x5 除以 8，并将结果存于 x6 中，一条 RV32I 指令可以实现吗？

3）使用一条 RV32I 指令，可以将 x5 中的值移至 x6 中吗？

答：

1）slli x6, x5, 3

2）srai x6, x5, 3

3）addi x6, x5, 0

4.9 写一段 RV32I 指令序列，将数据在内存的单元之间移动。以移动一个字（32 位）为例，假设该字所在的地址位于寄存器 x5 中，将其移动至寄存器 x6 保存的地址中。

lw x7, 0(x5)

sw x7, 0(x6)

4.10 写一段 RV32I 指令序列，将以下常数写入 x5 中：

1）20

2）0x12345678

3）0xBFFFFFF0

答：

1）addi x5, x0, 10

2）lui x5, 0x12345

addi x5, x5, 0x678

3）lui x5, 0xC0000

addi x5, x5, -16

4.11 当一段起始于单元 x0040 0000 的 RV32I 程序结束执行后，寄存器 x18~x23 的值分别是多少？

地址	31 25	24 20	19 15	14 12	11 7	6 0	解释
x2000 1234	0000 0000 0000 0000 1000 0000 0000 0000						
x2000 1230	0000 0000 0100 0000 0000 0000 0000 0000						
.....	……						.....
x1000 0004	0010 0000 0000 0000 0001 0010 0011 0000						
x1000 0000	……						.....
.....	……						.....
x0040 0020	0000 0000 0000		10111	010	10111	0000011	lw
x0040 001C	0000 0000 0000		10010	010	10111	0000011	lw
x0040 0018	0000 000	10011	10011	000	00011	0100011	sb
x0040 0014	0000 0000 0001		10010	000	10110	0000011	lb
x0040 0010	0000 0000 0100		10011	010	10101	0000011	lw
x0040 000C	0000 000	10011	10010	010	00000	0100011	sw
x0040 0008	0000 0000 0100		10010	000	10100	0000011	lb
x0040 0004	0000 0000 0100		10010	010	10011	0000011	lw
x0040 0000	0001 0000 0000 0000 0000				10010	0110111	lui

答：

x18	0x1000 0000
x19	0x2000 1230
x20	0x0000 0030

x21	0x0000 8000
x22	0x0000 0012
x23	0x3040 0000

另：内存地址 0x1000 0000~0x1000 0003 中的内容为 0x2000 1230；0x2000 1230~0x2000 1233 中的内容为 0x3040 0000。

4.12 下表显示了 RV32I 内存的一部分情况：

地址	31 25	24 20	19 15	14 12	11 7	6 0	解释
x0040 000C	1 111111	00000	00111	000	1010 1	1100011	beq
x0040 0008	1111 1111 1111		00111	100	00111	0010011	xori
x0040 0004	0000000	00110	00101	000	00111	0110011	add
x0040 0000	1111 1111 1111		00101	100	00101	0010011	xori

如果条件分支指令将控制转移到 x0400 0000 单元，那么 x5 和 x6 有什么特点？

答：-x6 的值为 x5 取反加 1 的结果时，即 x6==x5 时，分支跳转。

4.13 当如下 RV32I 指令序列执行至 x0040 0024 时，x5 中存储的值为 7，由此可推知 x6 的什么信息？

地址	31 25	24 20	19 15	14 12	11 7	6 0	解释
x0040 0024	.....						.....
x0040 0020	1 111111	00000	00111	001	0110 1	1100011	bne
x0040 001C	1111 1111 1111		00111	000	00111	0010011	addi
x0040 0018	0000000	00001	01000	001	01000	0010011	slli
x0040 0014	0000 0000 0001		00101	000	00101	0010011	addi
x0040 0010	0 000000	00000	01001	000	0100 0	1100011	beq
x0040 000C	0000000	01000	00110	111	01001	0110011	and
x0040 0008	0000 0000 0001		00000	000	01000	0010011	addi
x0040 0004	0000 0001 0000		00000	000	00111	0010011	addi
x0040 0000	0000 0000 0000		00000	000	00101	0010011	addi

答：x6 的低 16 位中有 7 个 1

4.14 RISC-V 指令处理包括哪些阶段？每个阶段的主要工作是什么？

取指令：IR ← Mem[PC]；NPC ← PC+4；

译码/取寄存器：A ← Regs[rs1]; B ← Regs[rs2]; Imm ← SEXT(imm[11:0]);

执行/计算地址：

R-类型指令：ALUOut ← A func B；

lw/sw 指令：ALUOut ← A + Imm；

beq 指令：Cond ← (A == B)；PCOffest ← PC + (Imm << 1)；

访问内存/完成分支：

lw 指令：MDR ← Mem[ALUOut]；PC ← NPC；

sw 指令：Mem[ALUOut] ← B；PC ← NPC；

R-类型指令：PC ← NPC；

beq 指令：if (Cond && Branch)　PC ← PCOffset；

else　PC ← NPC；

写回：Regs[rd] ← ALUOut；或 Regs[rd] ← MDR；

4.15 关于指令处理，请回答以下问题：

1）如果一个时钟周期需要 3 纳秒（即，3×10^{-9} 秒），那么，每秒可以包含多少个时钟周期？

2）如果某种计算机处理每条指令平均需要 5 个时钟周期，那么，在 1 秒内能够处理多少条指令？

3）在当今的微处理器中，采用流水线技术增加每秒处理的指令数。使用该技术，在每个时钟周期里，都从内存中取出一条指令，在时钟周期结束交给译码器，在下一个时钟周期进行译码，同时取下一条指令。

I. RISC-V 将指令执行过程划分为 5 个阶段，也就意味着采用流水线技术，在任意时钟周期内，最多会有多少条指令被执行？

II. 假设执行每条指令均需要从取指令开始到写回的 5 个阶段，每个阶段需要 1 个时钟周期，采用流水线技术执行一段程序，该程序顺序执行，不包含分支跳转指令，那么，每秒可以执行多少条指令？

答：

1）$1/(3\times10^{-9})$, 约为 0.3×10^{9}，约 300M；

2）60M，约为 6 千万条指令

3）I. 5 条指令；II. 最多 300M，约为 3 亿条指令

4.16 图 4.7 数据通路中的“生成立即数”逻辑电路，对来自 IR 中的立即数进行符号扩展。请给出该逻辑电路的真值表。

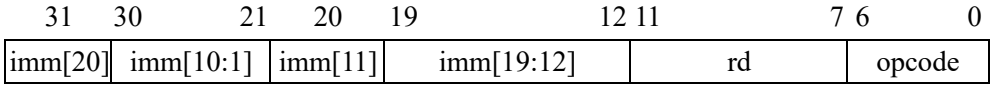
IR [6:0]	IR [31]	Imm
1100011（beq 指令）	1	0xFFFFF [31|7|30:25|11:8]
1100011	0	0x00000 [31|7|30:25|11:8]
0000011（lw 指令）	1	0xFFFFF [31:20]
0000011	0	0x00000 [31:20]
0100011（sw 指令）	1	0xFFFFF [31:25|11:7]
0100011	0	0x00000 [31:25|11:7]

4.17 图 4.7 数据通路中的“ALU 控制器”逻辑电路，根据来自 IR 中的函数码和控制器中的 ALUOp 控制信号，控制 ALU 选择何种运算。请给出该逻辑电路的真值表。

IR [30]	IR [14:12]	ALUOp	运算
0	000	10（R-类型）	add
1	000	10	sub
0	001	10	sll
0	010	10	slt
0	011	10	sltu
0	100	10	xor
0	101	10	srl
1	101	10	sra
0	110	10	or
0	111	10	and

/	/	00	add（加）
/	/	01	sub（减）

4.18 请基于图 4.7 给出的数据通路，对 RV32I 子集的数据通路重新设计，使其能够执行 J-类型指令。



执行/计算地址阶段：

jal 指令： PCOffest ← PC + (Imm << 1)；

注：“生成立即数”逻辑电路，对 jal 指令，增加 SEXT([31|19:12|20|30:21])的逻辑

访问内存阶段：

jal 指令：PC ← PCOffset；

写回阶段：

jal 指令：Regs[rd] ← NPC；

注：增加 NPC 到“写回阶段的选择器”的连线

5.1 在图 5.1 的循环结构中，条件分支指令中的立即数/偏移量是多少？无条件跳转指令中的立即数/偏移量是多少？

答：条件分支指令中的立即数：(子任务的指令数目+2)×4

无条件跳转指令中的立即数：- (子任务的指令数目+生成条件的指令数目+1)×4

5.2 如下 RV32I 机器语言程序片段实现了什么？请给出实现这一功能的 C 程序片段和 RISC-V 汇编语言程序片段。

地址	31 25	24 20	19 15	14 12	11 7	6 0	解释
x0040 002C	……						.....
x0040 0028	1 111111	00000	01000	001	0010 1	1100011	bne x8,x0,-28
x0040 0024	1111 1111 1111		01000	000	01000	0010011	addi x8,x8,-1
x0040 0020	0000 0000 0100		00110	000	00110	0010011	addi x6,x6,4
x0040 001C	0000 0000 0100		00101	000	00101	0010011	addi x5,x5,4
x0040 0018	0000000	11100	00101	010	00000	0100011	sw x28,0(x5)
x0040 0014	0000000	11101	11100	000	11100	0110011	add x28,x28,x29
x0040 0010	0000 0000 0000		00110	010	11101	0000011	lw x29,0(x6)
x0040 000C	0000 0000 0000		00101	010	11100	0000011	lw x28,0(x5)
x0040 0008	0000 0000 0101		00000	000	01000	0010011	addi x8,x0,5
x0040 0004	0000 0001 0100		00101	000	00110	0010011	addi x6,x5,20
x0040 0000	0001 0000 0000 0000 0000				00101	0110111	lui x5,0x10000

两个数组，分别起始于 0x10000000 和 0x10000014，各包含 5 个整数；计算两个数组之和，结果存储与第一个数组中。

```
for(i=0; i<5; i++){
    x[i]=x[i]+y[i];     //x，y，两个整数数组
}
```

```
        lui   x5, 0x10000
        addi x6, x5, 20
        addi x8, x0, 5
loop:   lw    x28, 0(x5)
        lw    x29, 0(x6)
        add   x28, x28, x29
        sw    x28, 0(x5)
        addi x5, x5, 4
        addi x6, x6, 4
        addi x8, x8,-1
        bne   x8, x0, loop
```

5.3 如下 RV32I 机器语言程序片段实现了什么？x10 的初始值是什么时，可以使得 x9 的最终结果为 7？请给出实现这一功能的 C 程序片段和 RISC-V 汇编语言程序片段。

地址	31 25	24 20	19 15	14 12	11 7	6 0	解释
x0040 0020	……						.....
x0040 001C	1 1111110110 1 11111111				00000	1101111	jal x0, -20

x0040 0018	0 000000	00000	01010	000	0100 0	1100011	beq x10,x0,8
x0040 0014	0000000	01010	01010	000	01010	0110011	add x10,x10,x10
x0040 0010	0000 0000 0001		01001	000	01001	0010011	addi x9,x9,1
x0040 000C	0 000000	00000	00101	000	0100 0	1100011	beq x5,x0,8
x0040 0008	0000000	01000	01010	111	00101	0110011	and x5,x10,x8
x0040 0004	0000 0000 0000		00000	000	01001	0010011	addi x9,x0,0
x0040 0000	1000 0000 0000 0000 0000				01000	0110111	lui x8,0x80000

x10 中的数有 7 个 1；

```
int result=0;
int mask=0x80000000;
while (x!=0){
    if((x&mask)!=0)
        result++;
    x=x+x;          //或 x=x<<1;
}
```

```
        lui   x8, 0x80000
        addi x9, x0, 0
loop:   and  5, x10, x8
        beq  x5, x0, next
        addi x9, x9, 1
next:   add  x10, x10, x10
        beq  x10, x0, exit
        j     loop
exit:   #省略
```

5.4 编写 RISC-V 汇编程序片段，并给出实现这一功能的 C 程序片段。

1）比较 x5 和 x6 中的两个数，并将较大的数放入 x7 中。

2）判断存储在 x5 中的数是否是偶数。如果是偶数，则 x6=1，否则，x6=0。

3）统计一列正整数中的奇数和偶数个数，并将结果分别保存于 x8 和 x9 中。假设这一列整数存储于从 x1000 0000 开始的一段连续的存储单元之中，以-1 结束。

4）统计 10 个整数中的负数的个数，并将结果保存于 x5 中。假设这 10 个整数存储于从 x1000 0000 开始的一段连续的存储单元之中。

5）将 10 个整数中的正数乘以 2，负数除以 2，并存回原存储单元之中。假设这一列整数存储于从 x1000 0000 开始的一段连续的存储单元之中。

6）从存储单元 A 到存储单元 B 中存储的是整数，统计这些整数中 5 出现的次数。假设地址 A 和地址 B 位于存储单元 x1000 0000~x1000 0003 和 x1000 0004~x1000 0007 中。

7）统计某个字符在一个文档中出现的次数。文档由 ASCII 字符构成，假设文档的起始地址为存储单元 x1000 0000，文档的末尾有一个表示结束的字符 EOT（x04），要统计的字符位于 x5 中，统计结果存储于 x6 中。

8）将一个文档中的英文小写字母转换为相应的大写字母，并存回原文档中。假设文档

的起始地址为存储单元 x1000 0000，文档的末尾有一个表示结束的字符 EOT（x04）。

汇编程序见编程题——5-4-*.s

C 程序片段：

1）

```
int x=1;
int y=10;
int max=x;
if(x<y)    max=y;
```

2）

```
int x=10;
int y=0;
if(x%2==0) y=1;
```

3）

```
int num[]={1,2,3,4,5,-1};
int numOfOdd=0;
int numOFEven=0;
int i=0;
while(num[i]!=-1){
    if(num[i]%2==0)
        numOFEven++;
    else
        numOfOdd++;
    i++;
}
```

4）

```
int num[]={1,2,3,4,5,-1,-2,3,2,1};
int numOfNeg=0;
int i;
for(i=0;i<10;i++){
    if(num[i]<0)
        numOfNeg++;
}
```

5）

```
int num[]={1,2,3,4,5,-5,-6,3,2,1};

int i;
for(i=0;i<10;i++){
      if(num[i]>0)
            num[i]=num[i]*2;
      else
            num[i]=num[i]/2;
}
```

6）

```
int num[8]={1,2,3,4,5,-5,-6,5};
int *p1=num;
int *p2=num+8;
int count=0;
int i=0;

while(p1!=p2){
    if(num[i]==5)
        count++;
    i++;
    p1++;
}
```

7）

```
char file[]={'H','e','l','l','o',',',' ','w','o','r','l','d','!','\4'};
char c='o';
int count=0;
int i=0;

while(file[i]!=4){
    if(file[i]==c)
        count++;
    i++;
}
```

8）

```
char file[]={'H','e','l','l','o',',',' ','w','o','r','l','d','!','\4'};
int i=0;

while(file[i]!=4){
    if(file[i]>='a' && file[i]<='z')
        file[i]=file[i]+'A'-'a';
    i++;
}
```

5.5 ~ 5.7

见编程题——5-5-1.s, 5-5-2.s, 5-6.s, 5-7.s

5.8 假设已知从 x1000 0000 开始的存储单元中存储了 10 个整数，计算这些整数的和。实现的程序如下，但是存在 bug，请一一找出并纠正过来。

```
01    #
02    # 计算一列数之和
03    #
04                    .data
05                    .align        2
06    numbers:        .word         6, 3, 4, 6, 8, -2, 45, 5, 8, 5
07    #
```

```
08                  .text
09                  .align      2
0A                  .globl      main
0B    main:         la          x5, numbers          # x5，整数地址
0C                  andi        x8, x0, 0            # x8 清零，一列数之和
0D                  addi        x6, x5, 10
0E    again:        beq         x6, x5, exit
0F                  lw          x7, 0(x5)
10                  add         x8, x7, x8
11                  addi        x5, x5, 1            # 跟踪下一个整数地址
12                  j           again
13    exit:         ......                           #下一个任务
```

答：0D 行 改为：addi x6, x5, 40
11 行 改为：addi x5, x5, 4

5.9 对于如下程序：

```
                    .data
                    .align      2
HelloWorld:         .string     "Hello, World!"
#
                    .text
                    .align      2
                    .globl      main
main:               la          x5, HelloWorld
loop:               lb          x6, 0(x5)
                    beqz        x6, exit
                    addi        x6, x6, 1
                    sb          x6, 0(x5)
                    addi        x5, x5, 1
                    j           loop
exit:               ......                           #下一个任务
```

1）构建符号表；
2）该程序实现了什么？
3）给出实现这一功能的 C 程序片段。

答：
1)

HelloWorld	0x1000 0000
main	0x0001 0000
loop	0x0001 0008
exit	0x0001 0020

2)将“Hello, World!”字符串中的每个字符加 1 后再存回去，即“Ifmmp-!Xpsme"”

3)

```
char str[]="Hello, World!";
int i=0;
while(str[i]!=0){
    str[i]=str[i]+1;
    i++;
}
```

5.10 对于如下程序：

```
                .data
                .align      2
num:            .word       ......              #一个正整数
#
                .text
                .align      2
                .globl      main
main:           la          x5, num
                lw          x6, 0(x5)
                addi        x8, x0, 0
                andi        x7, x6, 1
                bnez        x7, again
                addi        x6, x6, -1
again:          add         x8, x8, x6
                addi        x6, x6, -2
                bgez        x6, again
                sw          x8, 4(x5)
                ......                          #下一个任务
```

1）构建符号表；

2）将该程序翻译为机器语言程序；

3）该程序实现了什么？

4）给出实现这一功能的 C 程序片段。

答：

1)

num	0x1000 0000
main	0x0001 0000
again	0x0001 001C

2)

地址	31 25	24 20	19 15	14 12	11 7	6 0	解释
x1000 0004	sum						sum
x1000 0000	x						x
.....	……						.....
x0001 0028	0000000	01000	00101	010	00100	0100011	sw

x0001 0024	1 111111	00000	00110	101	1100 1	1100011	bge
x0001 0020	1111 1111 1110		00110	000	00110	0010011	addi
x0001 001C	0000000	00110	01000	000	01000	0110011	add
x0001 0018	1111 1111 1111		00110	000	00110	0010011	addi
x0001 0014	0 000000	00000	00111	001	0100 0	1100011	bne
x0001 0010	0000 0000 0001		00110	111	00111	0010011	andi
x0001 000C	0000 0000 0000		00000	000	01000	0010011	addi
x0001 0008	0000 0000 0000		00101	010	00110	0000011	lw
x0001 0004	0000 0000 0000		00101	000	00101	0010011	addi
x0001 0000	0000 1111 1111 1111 0000				00101	0010111	auipc

3) 计算 1+3+5+.....+(2n-1) ，即不大于 x 的正奇数和，结果存入整数 x 后的存储单元

4)

```
int sum=0;
if (x%2==0)
    x=x-1;
while(x>=0){
    sum+=x;
    x=x-2;
}
```

5.11 对于如下程序：

```
                .data
                .align      2
num:            .word       ......              #一个字
mask:           .word       0xFFFF0000
#
                .text
                .align      2
                .globl      main
main:           la          x5, num
                lw          x5, 0(x5)
                la          x6, mask
                lw          x6, 0(x6)
                slli        x7, x5, 16
                and         x8, x5, x6
                bne         x7, x8, else
                addi        x8, x0, 1
                j           exit
else:           andi        x8, x0, 0
exit:           ......                          #下一个任务
```

1）构建符号表；

2）该程序实现了什么？

3）给出实现这一功能的 C 程序片段。

1)

num	0x1000 0000
mask	0x1000 0004
main	0x0001 0000
else	0x0001 002C
exit	0x0001 0030

2) 判断 num 中的数，高 16 位与低 16 位是否相同，结果（1 或 0）在 x8 中

3)

```
int result=0;
int temp1,temp2;
int mask=0xffff0000;
temp1=num<<16;
temp2=num&mask;
if(temp1==temp2)
    result=1;
```

5.12 对于图 5.8 中的 switch 语句：

1）将此 switch 语句转化为级联的 if-else 语句，并给出级联的 if-else 语句的汇编代码；

2）将级联的 if-else 语句与 switch 语句的汇编代码进行对比：当 x 的值分别为 0、1、2、3、4、5 和 6 时，执行的指令数目。

答：

1）

```
int result = 0;
if (x==1) {
    result += 1;
}else if(x==2){
    result += 2;
    result += 3;
} else if(x==3){
    result += 3;
} else if(x==4||x==5){
    result += 5;
}else{
    result = 0;
}
```

```
addi    x9, x0, 0       # result = 0;
addi    x5, x0, 1
beq     x18, x5, L1     # x==1
addi    x5, x0, 2
beq     x18, x5, L2     # x==2
addi    x5, x0, 3
```

```
                beq     x18, x5, L3     # x==3
                addi    x5, x0, 4
                beq     x18, x5, L4     # x==4
                addi    x5, x0, 5
                beq     x18, x5, L4     # x==5
                j       exit
L1:             addi    x9, x9, 1       # result += 1;
                j       exit
L2:             addi    x9, x9, 2       # result += 2;
                addi    x9, x9, 3       # result += 3;
                j       exit
L3:             addi    x9, x9, 3       # result += 3;
                j       exit
L4:             addi    x9, x9, 5       # result += 5;
exit:           #省略
```

2）

x	级联的 if-else	switch
0	12	10
1	5	11
2	7	12
3	9	11
4	9	11
5	11	11
6	12	10

5.13 如下程序比较两个字符串，如果两个字符串相同，以 x8 的值为 1 结束，否则，x8 的值为 0。请填空，将程序补充完整。

```
                .data
First:          .string     "string1"
Second:         .string     "string2"
#
                .text
                .align      2
                .globl      main
main:           addi        x8, x0, 0
                la          x5, First
                la          x6, Second
loop:           lb          x7, 0(x5)
                lb          x28, 0(x6)
                bne         x7, x28, done
                beqz        x7, exit
                addi        x5, x5, 1
                addi        x6, x6, 1
```

```
                j           loop
exit:           addi        x8, x0, 1
done:           ......                          #下一个任务
```

5.14 如下程序判断一个字符串是否是“回文”（正向读和反向读都相同的字符串），例如，“strts”就是回文。如果字符串是回文，程序以 x8 的值为 1 结束，否则 x8 为 0。请填空，将程序补充完整。

```
                .data
chars:          .string     "strts"
#
                .text
                .align      2
                .globl      main
main:           addi        x8, x0, 0
                la          x5, chars
loop1:          lb          x6, 0(x5)
                beqz        x6, next
                addi        x5, x5, 1
                j           loop1
next:           addi        x7, x5, -1
                la          x5, chars
loop2:          beq         x5, x7, exit
                lb          x6, 0(x5)
                lb          x28, 0(x7)
                bne         x6, x28, done
                addi        x5, x5, 1
                addi        x7, x7, -1
                blt         x7, x5, exit
                j           loop2
exit:           addi        x8, x0, 1
done:           ......                          #下一个任务
```

6.1 ~ 6.3

见编程题——6-1.s, 6-2-1.s, 6-2-2.s, 6-3-1.s，以及 6-2.c, 6-3.c

6.4 如果在子例程中又调用了子例程，是否可以采用 callee-save（被调用者保存）的策略，保存/恢复返回地址 x1？

答：无法采用 callee-save（被调用者保存）的策略，实现返回地址的保存。

6.5 见编程题——6-5.s

6.6 假设位于存储单元 num 中的值是一个大于 2 的正整数。

1）如下程序实现了什么？程序执行到 end 时，x11 中的值代表什么？如果 x5 的值为 1，代表什么？

2）Mod 子例程的功能是什么？参数和返回值分别是哪些寄存器？

3）给出实现 Mod 子例程功能的 C 函数：int Mod(int x, int y)。

```
                .data
num:            .word       ......      #一个正整数
#
                .text
                .globl      main
main:           addi        x5, x0, 0
                addi        x11, x0, 2
                la          x7, num
                lw          x10, 0(x7)
#
loop:           call        Mod
                beqz        x9, exit0
                addi        x11, x11, 1
                beq         x11, x10, exit1
                j           loop
#
exit1:          addi        x5, x0, 1
exit0:          j           end
#
Mod:            addi        x9, x10, 0
again:          sub         x9, x9, x11
                bltz        x9, exit
                beqz        x9, exit
                j           again
exit:           ret
#
end:            ......                          # 下一个任务
```

答：

1）找出 num 中的数的最小因数；x11 中的数是最小因数；如果是质数，x5 中的值为 1；

2）Mod 子例程，x10/x11，是否可以整除，x10 和 x11 是参数，x9 是返回值，0 表示可以整除；

3）

```
int Mod(int x, int y){
        int result=0;
        if(x%y){
                result = -1;
        }
        return result;
}
```

6.7 假设从存储单元 Data 开始存储的 10 个值可以是任意的 10 个整数。

1）如下程序实现了什么？

2）Cmp 子例程的功能是什么？参数是哪些寄存器？有返回值吗？

3）Swap 子例程的功能是什么？参数是哪些寄存器？有返回值吗？

```
                .data
Data:           .word   3, 14, 35, 47, 5, 20, 12, 14, -6, 22
SaveReg:        .word   0
#
                .text
                .globl  main
main:           addi    x6, x0, 10
OutLoop:        addi    x5, x0, 1
                beq     x6, x5, exit
                addi    x7, x6, -1
                la      x5, Data
InnerLoop:      beqz    x7, exit1
                mv      x12, x5
                jal     x1, Cmp
                addi    x5, x5, 4
                addi    x7, x7, -1
                j       InnerLoop
exit1:          addi    x6, x6, -1
                j       OutLoop
exit:           j       end
#
Cmp:            la      x28, SaveReg
                sw      x1, 0(x28)
                lw      x10, 0(x12)
                lw      x11, 4(x12)
                beq     x10, x11, Return
                blt     x10, x11, Return
                jal     x1,Swap
Return:         la      x28, SaveReg
                lw      x1, 0(x28)
                ret
#
```

```
Swap:       sw      x11, 0(x12)
            sw      x10, 4(x12)
            ret
#
end:        ......
```

答：

1）冒泡排序；

2）Cmp 子例程，比较 x12 所指的 8 个单元中的 2 个整数，x10 和 x11，如果 x10>x11，则调用 Swap 子例程（即，实现交换）；参数是 x12，没有返回值；

3）Swap 子例程，将 x11 和 x10 的值，依次存储于 x12 所指的 8 个单元中；参数是 x10，x11 和 x12，没有返回值。

6.8 如下程序将造成无限循环，请找出原因。

```
            jal     SubA
            j       end
#
SubA:       jal     SubB
            ret
#
SubB:       addi    x11, x0, 48
            ret
#
end:        ......
```

答：SubA 子例程没有保存返回地址 x1，在调用 SubB 子例程后，无法返回。

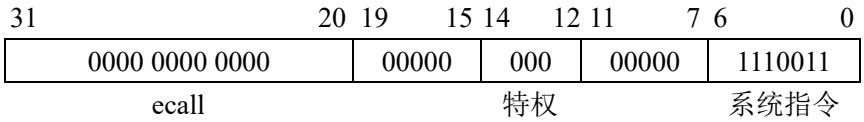
6.9 如下程序实现了什么？

```
            .data
Data:       .string "Hello, World!"
#
            .text
            .globl  main
main:       lui     x2, 0xc0000
            addi    x2, x2, -16
            addi    x11, x0, 0
            la      x5, Data
Input:      lb      x10, 0(x5)
            beqz    x10, Output
            addi    x11, x11, 1
            call    push
            addi    x5, x5, 1
            j       Input
#
Output:     la      x5, Data
loop:       beqz    x11, Done
```

```
                call        pop
                addi        x11, x11, -1
                sb          x10, 0(x5)
                addi        x5, x5, 1
                j           loop
        #
        Done:   j           end
        #
        push:   addi        x2, x2, -1
                sb          x10, 0(x2)
                ret
        #
        pop:    lb          x10, 0(x2)
                addi        x2, x2, 1
                ret
        #
        end:    ......
```

答：利用栈，实现字符串逆序。

7.1 基于图 4.7，对 RV32I 子集的数据通路重新设计，使其能够执行 ecall 指令。提示，与 jalr 指令类似。



增加几个特殊寄存器：mepc，mtvec，mstatus；mtvec 是一个硬连线的寄存器，值即操作系统自陷处理例程的起始地址；

执行/计算地址阶段：

ecall 指令： mepc ← PC；

访问内存阶段：

ecall 指令：PC ← mtvec；

写回阶段：

ecall 指令：mstatus[12:11] ← 11；

注：增加相应的连线

7.2 对于如下汇编语言程序：

```
                .data
Char:           .word       0x61626364
HelloWorld:     .string     "Hello, World!"
#
                .text
                .globl      main
main:           la          x5, Char
loop:           lb          x11, 0(x5)
                beqz        x11, Exit
                addi        x10, x0, 11
                ecall
                addi        x5, x5, 4
                j           loop
#
Exit:           .....                   #下一个任务
```

1）程序的输出是什么？

2）在“addi x5, x5, 4”指令被执行之前，执行的是哪条指令？

答：

1）“dHoo!”

2）操作系统自陷处理例程的 mret 指令，从自陷返回。

7.3 执行如下汇编语言程序，输出是什么？

```
                .data
Char:           .word       0x61626364
HelloWorld:     .string     "Hello, World!"
#
                .text
                .globl      main
```

```
main:       addi    x5, x0, 0
            la      x11, Char
            sw      x5, 4(x11)
            addi    x10, x0, 4
            ecall
#
Exit:       ......                  #下一个任务
```

答：“dcba”

7.4 编写 RISC-V 汇编程序片段（使用 ecall 指令），并给出实现这一功能的 C 程序（使用库函数）。

1）在屏幕上显示 26 个大写英文字母 A~Z。

2）从键盘输入一个字符，如果该字符是字母表中的小写英文字母，则转化成大写字母输出，否则，原样输出。

3）从键盘输入一行字符（以回车结束），并将该行字符存储于以标记 park 开头的一段存储单元之中，然后回显除空格之外的所有字符。例如，如果输入是"Let's go to the park at 4:00pm."，输出为"Let'sgototheparkat4:00pm."。

汇编程序见编程题——7-4-1.s, 7-4-2.s, 7-4-3.s

1）

```
char c='A';
while(c<='Z'){
    putchar(c);
    c++;
}
```

2）

```
char c;
c=getchar();
if(c>='a' && c<='z'){
    putchar(c+'A'-'a');
}else{
    putchar(c);
}
```

3）

```
char park[100];
int i=0;
while((park[i]=getchar())!='\n'){
    i++;
}
i=0;
while(park[i]!='\n'){
      if(park[i]!=' '){
            putchar(park[i]);
      }
```

```
    i++;
}
```

7.5 设计一个新的服务例程，该服务例程读入一个字符并回显到屏幕上，读入的字符存于 x11 中。

```
kbcr:       .word   0xFFFF0000      # KBCR 的内存映射地址
kbdr:       .word   0xFFFF0004      # KBDR 的内存映射地址
dcr:        .word   0xFFFF0008      # DCR 的内存映射地址
ddr:        .word   0xFFFF000C      # DDR 的内存映射地址
……                                  #  省略
#  输入字符并回显服务例程
getCNew:    addi    x2, x2, -4      # x2，栈指针
            sw      x7, 0(x2)       # callee-save

            la      x5, kbcr
InPoll:     lw      x6, 0(x5)       #  测试是否有字符被输入
            lw      x7, 0(x6)
            andi    x6, x7, 1
            beqz    x6, InPoll      #  如果 KBCR[0]==0，轮询
            la      x5, kbdr
            lw      x6, 0(x5)
            lw      x11, 0(x6)      #将 KBDR 中的数据加载到 x11 中
            la      x5, dcr
OutPoll:    lw      x6, 0(x5)       #  测试显示是否就绪
            lw      x7, 0(x6)
            andi    x6, x7, 1
            beqz    x6, OutPoll     #  如果 DCR[0]==0，轮询
            la      x5, ddr
            lw      x6, 0(x5)
            sw      x11, 0(x6)      #  将 x11 中的数据写到 DDR 中
            csrrw   x5, mepc, x0
            addi    x5, x5, 4
            csrrw   x0, mepc, x5    # mepc<-mepc+4
            lw      x7, 0(x2)       #  恢复寄存器
            addi    x2, x2, 4
            lw      x6, 0(x2)
            addi    x2, x2, 4
            lw      x5, 0(x2)
            addi    x2, x2, 4
            mret                    #  从自陷返回
```

7.6 重新定义操作系统自陷机制：

1）如果 mtvec 中的值，是操作系统的自陷向量表的起始地址，那么，如何定义 ecall 指令？

2）如果存储单元 x0000 0000~x0000 1000 被分配给操作系统：通过将系统调用号左移 8 位，形成相应服务例程的起始地址，那么，如何定义 ecall 指令？每个服务例程最多可以占用多少存储单元？ 最多可以有多少个服务例程？

3）如果系统调用号就是相应服务例程的起始地址，那么，如何定义 ecall 指令？

答：

1）先将 PC 的值写入 mepc；

计算 mtvec+x10*4，加载到 PC 中；注：x10*4: x10<<2

将 mstatus 设置为机器模式（如 11）。

2）ecall 指令：先将 PC 的值写入 mepc；

计算 x10<<8，加载到 PC 中；

将 mstatus 设置为机器模式（如 11）。

每个服务例程最多可以占用 256 个存储单元；

最多可以有 256 个服务例程；

3）先将 PC 的值写入 mepc；

将 x10 的值，加载到 PC 中；

将 mstatus 设置为机器模式（如 11）。

7.7 对于如下程序：

```
#include <stdio.h>

int main() {
    int x = 0;
    int y = 0;
    char a = 'a';
    char b = 'b';
    a = getchar ();
    scanf ("%d%d", &x, &y);
    b = getchar ();

    printf ("%d %d %c %c", x, y, a, b);
}
```

1）如果输入流是 10 20 C，程序的输出是什么？

2）如果输入流是 C 10 20D，程序的输出是什么？

3）如果输入流是 10 C，程序的输出是什么？

答：

1）0 20 1

2）10 20 C D

3）0 0 1 C

7.8 请给出 putchar 库函数的底层实现。提示：putchar 将输出的字符存入 stdout 中，如果存入的是换行符，或者 stdout 已满，就自陷进入操作系统处理例程，由操作系统将 stdout 中的字符串输出。

```
stdout:     .byte     0, 0, .....          # 标准输出流，共 size 个字节，初值均为 0/null
outPt:      .word     .....                # 指针，初值为 stdout
```

```
num:        .word   0               # stdout 中的字符数，初值为 0
size:       .word   ......          # stdout 的大小，如 1024
......                              #省略代码
putchar:    ......                  # 省略寄存器的保存代码
            la      x5, outPt
            lw      x7, 0(x5)       # 字符指针
            la      x6, num
            lw      x28, 0(x6)      # 字符数
            la      x29, size
            lw      x29, 0(x29)     # 输出流的大小
            sb      x11, 0(x7)      # 将 x11 中的字符存入输出流
            addi    x7, x7, 1
            sw      x7, 0(x5)       # 指针指向下一个字符
            addi    x28, x28, 1
            sw      x28, 0(x6)      # 字符数加 1
            beq     x28, x29, trap  # 输入流已满，自陷
            addi    x30, x0, 10
            beq     x11, x30, trap  # 存入的字符是换行符，自陷
            j       exit
trap:       la      x11, stdout     # x11，字符串首地址
            addi    x10, x0, 4      # puts，输出字符串服务例程
            ecall                   # 系统调用
            la      x5, outPt
            sw      x11, 0(x5)      # 字符指针指向 stdout
            la      x5, num
            sw      x0, 0(num)      # 字符数清零
exit:       ......                  # 省略寄存器的恢复代码
            ret
```

8.1 对于如下程序：

```
#include <stdio.h>

int Sub (int x, int y);

int main ()
{
    int x = 2;
    int y = 3;
    int z;

    z = Sub (x, y);
    x = Sub (y, z);
    y = Sub (x, y);
    printf ("%d %d %d\n", x, y, z);
}

int Sub (int y, int x)
{
    return y - x;
}
```

1）程序的输出是什么？

2）请写出这段 C 程序的 RISC-V 汇编代码（忽略 printf 库函数的调用）。

提示：main 函数将局部变量 x，y 和 z 分配给寄存器 s1，s2 和 s3；Sub 函数将形参 y 和 x 分配给参数寄存器 a0 和 a1，返回值被分配给 a0。

答：

1）4 1 -1

2）

```
main:   addi    sp, sp, -20     # 为 main 函数分配栈帧
        sw      ra, 16(sp)      # x1（返回地址）
        sw      fp, 12(sp)      # x8（帧指针）
        addi    fp, sp, 16      # 调整帧指针
        sw      s1, 8(sp)       # x9（局部变量 x）
        sw      s2, 4(sp)       # x18（局部变量 y）
        sw      s3, 0(sp)       # x19（局部变量 z）
        li      s1, 2           # x = 2;
        li      s2, 3           # y = 3;
        mv      a0, s1          # 实参 x->形参 y
        mv      a1, s2          # 实参 y->形参 x
        jal     ra, Sub1
        mv      s3, a0          # z = Sub (x, y);
        mv      a0, s2          # 实参 y->形参 y
        mv      a1, s3          # 实参 z->形参 x
        jal     ra, Sub1
```

```
                mv      s1, a0          # x = Sub (y, z);
                mv      a0, s1          #  实参 x->形参 y
                mv      a1, s2          #  实参 y->形参 x
                jal     ra, Sub1
                mv      s2, a0          # y = Sub (x, y);
#省略 printf 函数调用
                li      a0, 0           # return 0;
                lw      s3, 0(sp)       #  恢复 x19
                lw      s2, 4(sp)       #  恢复 x18
                lw      s1, 8(sp)       #  恢复 x9
                lw      fp, 12(sp)      #  恢复 x8
                lw      ra, 16(sp)      #  恢复 x1
                addi    sp, sp, 20      #  弹出 main 函数的栈帧
                ret
# Sub 函数
Sub1:           addi    sp, sp, -8      #  为 Sub 函数分配栈帧
                sw      ra, 4(sp)       # x1（返回地址）
                sw      fp, 0(sp)       # x8（帧指针）
                addi    fp, sp, 4       #  调整帧指针
                sub     t0, a0, a1      # y-z
                addi    a0, t0, 0       # return y-z;
                lw      fp, 0(sp)       #恢复  x8
                lw      ra, 4(sp)       #恢复  x1
                addi    sp, sp, 8       #  弹出 Sub 函数的栈帧
                ret
```

8.2 对于如下程序：

```
#include <stdio.h>

int Multiply (int x, int y);

int z = 4;                      //全局变量

int main ()
{
    int x = 2;
    int y = 3;
    int z;                      //局部变量

    z = Multiply (x, y);
    x = Multiply (y, z);
    y = Multiply (x, z);
    printf ("%d %d %d \n ", z, x, y);
}
```

```
int Multiply (int x, int y)
{
    return x * y * z;
}
```

1）程序的输出是什么？

2）请写出这段 C 程序的 RISC-V 汇编代码（忽略 printf 库函数的调用）。

提示：main 函数将局部变量 x，y 和 z 分配给寄存器 s1，s2 和 s3；Multiply 函数将形参 x 和 y 分配给参数寄存器 a0 和 a1，返回值被分配给 a0。

答：

1）24 288 27648

2）

```
                .data
z:              .word   4

                .text
                .globl  main
# main 函数，# 注：运行时，在 return 0；设置断点，运行至断点，查看 x,y 和 z 的值
main:
                addi    sp, sp, -20     # 为 main 函数分配栈帧
                sw      ra, 16(sp)      # x1（返回地址）
                sw      fp, 12(sp)      # x8（帧指针）
                addi    fp, sp, 16      # 调整帧指针
                sw      s1, 8(sp)       # x9（局部变量 x）
                sw      s2, 4(sp)       # x18（局部变量 y）
                sw      s3, 0(sp)       # x19（局部变量 z）
                li      s1, 2           # x = 2;
                li      s2, 3           # y = 3;
                mv      a0, s1          # 实参 x->形参 x
                mv      a1, s2          # 实参 y->形参 y
                jal     ra, Mult
                mv      s3, a0          # z = Multiply (x, y);
                mv      a0, s2          # 实参 y->形参 x
                mv      a1, s3          # 实参 z->形参 y
                jal     ra, Mult
                mv      s1, a0          # x = Multiply (y, z);
                mv      a0, s1          # 实参 x->形参 y
                mv      a1, s3          实参 y->形参 x
                jal     ra, Mult
                mv      s2, a0          # y = Multiply (x, z);
#省略 printf 函数调用
                li      a0, 0           # return 0;
                lw      s3, 0(sp)       # 恢复 x19
                lw      s2, 4(sp)       # 恢复 x18
                lw      s1, 8(sp)       # 恢复 x9
```

```
                lw      fp, 12(sp)      # 恢复 x8
                lw      ra, 16(sp)      # 恢复 x1
                addi    sp, sp, 20      # 弹出 main 函数的栈帧
                ret
# Multiply 函数
    Mult:       addi    sp, sp, -8      # 为 Mult 函数分配栈帧
                sw      ra, 4(sp)       # x1（返回地址）
                sw      fp, 0(sp)       # x8（帧指针）
                addi    fp, sp, 4       # 调整帧指针
    # x*y
                addi    t4, x0, 0       #flag=0，计算 x*y
                addi    t1, a0, 0       # x
                addi    t2, a1, 0       # y
    Multiply:
                andi    t0, t0, 0       #x5，积
    loop:       beqz    t2, exit        #x6，乘数
                andi    t3, t2, 1
                beqz    t3, next
                add     t0, t0, t1      #x7，被乘数
    next:       srli    t2, t2, 1
                slli    t1, t1, 1
                j       loop
    exit:       addi    t5, x0, 1
                beq     t4, t5, end
    # x*y*z
                addi    t1, t0, 0       # x*y
                la      gp, z
                lw      t2,0(gp)        # z
                addi    t4,x0,1         # flag=1，计算 x*y*z
                j       Multiply
    end:        addi    a0, t0, 0       # return x*y*z;
                lw      fp, 0(sp)       #恢复 x8
                lw      ra, 4(sp)       #恢复 x1
                addi    sp, sp, 8       # 弹出 Mult 函数的栈帧
                ret
```

8.3 对于如下程序：

```
#include <stdio.h>

void Swap (int x, int y);

int main()
{
    int x = 1;
```

```
    int y = 2;
    printf ("x = %d, y = %d\n ", x, y);

    Swap (x, y);
    printf ("x = %d, y = %d\n ", x, y);
}

void Swap (int x, int y)
{
    int temp;

    temp = x;
    x = y;
    y = temp;
    printf ("x = %d, y = %d\n ", x, y);
}
```

1）程序的输出是什么？

2）请写出这段 C 程序的 RISC-V 汇编代码（忽略 printf 库函数的调用）。

提示：main 函数将局部变量 x 和 y 分配给寄存器 s1 和 s2；Swap 函数将形参 x 和 y 分配给参数寄存器 a0 和 a1，将局部变量 temp 分配给寄存器 s1。

答：

1）x = 1, y = 2
x = 2, y = 1
x = 1, y = 2

2）

```
main:       addi    sp, sp, -16     # 为 main 函数分配栈帧
            sw      ra, 12(sp)      # x1（返回地址）
            sw      fp, 8(sp)       # x8（帧指针）
            addi    fp, sp, 12      # 调整帧指针
            sw      s1, 4(sp)       # x9（局部变量 x）
            sw      s2, 0(sp)       # x18（局部变量 y）
            li      s1, 1           # x = 1;
            li      s2, 2           # y = 2;
#   省略 printf 函数调用
            mv      a0, s1          # 实参 x->形参 x
            mv      a1, s2          # 实参 y->形参 y
            jal     ra, Swap        # Swap (x, y);
#   省略 printf 函数调用
            li      a0, 0           # return 0;
            lw      s2, 0(sp)       # 恢复 x18
            lw      s1, 4(sp)       # 恢复 x9
            lw      fp, 8(sp)       # 恢复 x8
            lw      ra, 12(sp)      # 恢复 x1
            addi    sp, sp, 16      # 弹出 main 函数的栈帧
```

```
                ret
# Swap 函数
Swap:           addi    sp, sp, -12     # 为 Swap 函数分配栈帧
                sw      ra, 8(sp)       # x1（返回地址）
                sw      fp, 4(sp)       # x8（帧指针）
                addi    fp, sp, 8       # 调整帧指针
                sw      s1, 0(sp)       # x9（局部变量 temp）
                addi    s1, a0, 0       # temp=x;
                addi    a0, a1, 0       # x=y;
                addi    a1, s1, 0       # y=temp;
    #   省略 printf 函数调用
                lw      s1, 0(sp)       #恢复 x9
                lw      fp, 4(sp)       #恢复 x8
                lw      ra, 8(sp)       #恢复 x1
                addi    sp, sp, 12      # 弹出 Swap 函数的栈帧
                ret
```

8.4 对于如下程序：

```
#include <stdio.h>

int Func1 (int x, int y);
int Func2 (int x, int y);
int Func3 (int x, int y);

int main ()
{
    int x;

    x = Func1 (3, 10);
    printf ("%d\n", x);
}

int Func1 (int x, int y)
{
    int z;

    z = Func2 (x, y);
    return z;
}

int Func2 (int x, int y)
{
    int z;
```

```
    z = Func3 (y, x) * y;
    return z;
}

int Func3 (int x, int y)
{
    int z;

    z = x / y;
    return z;
}
```

1）程序的输出是什么？

2）请画出当程序从函数 Func3 返回之前的运行时栈，并标出各单元的内容。

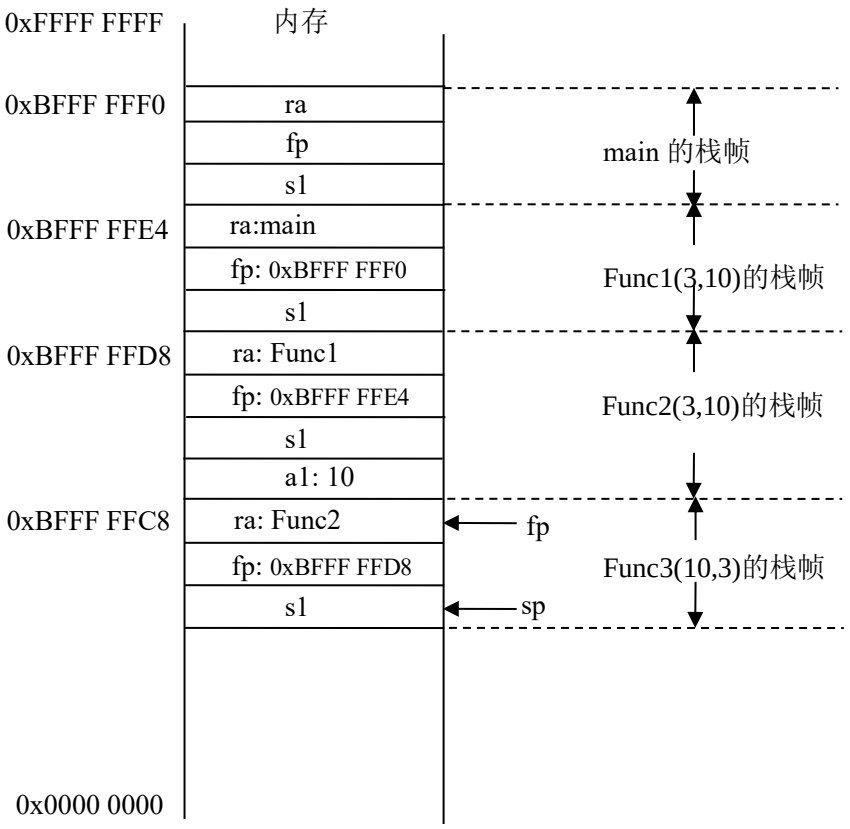
3）请写出这段 C 程序的 RISC-V 汇编代码（忽略 printf 库函数的调用）。

提示：main 函数将局部变量 x 分配给寄存器 s1；Func1，Func2 和 Func3 函数都是将形参 x 和 y 分配给参数寄存器 a0 和 a1，将局部变量 z 分配给寄存器 s1，将返回值分配给 a0。

答：

1）30

2）



3）

```
main:       addi    sp, sp, -12     # 为 main 函数分配栈帧
            sw      ra, 8(sp)       # x1（返回地址）
```

```
                sw      fp, 4(sp)       # x8（帧指针）
                addi    fp, sp, 8       #  调整帧指针
                sw      s1, 0(sp)       # x9（局部变量 x）
                li      a0, 3
                li      a1, 10
                jal     ra, Func1       # Func1 (3, 10);
                mv      s1, a0          # x = Func1 (3, 10);
        #   省略 printf 函数调用
                li      a0, 0           # return 0;
                lw      s1, 0(sp)       #  恢复 x9
                lw      fp, 4(sp)       #  恢复 x8
                lw      ra, 8(sp)       #  恢复 x1
                addi    sp, sp, 12      #  弹出 main 函数的栈帧
                ret
    # Func1 函数
    Func1:      addi    sp, sp, -12     #  为 Func1 函数分配栈帧
                sw      ra, 8(sp)       # x1（返回地址）
                sw      fp, 4(sp)       # x8（帧指针）
                addi    fp, sp, 8       #  调整帧指针
                sw      s1, 0(sp)       # x9（局部变量 z）
                mv      a0, a0
                mv      a1, a1
                jal     ra, Func2       # Func2 (x, y);
                mv      s1, a0          # z = Func2 (x, y);
                mv      a0, s1          # return z;
                lw      s1, 0(sp)       #恢复  x9
                lw      fp, 4(sp)       #恢复  x8
                lw      ra, 8(sp)       #恢复  x1
                addi    sp, sp, 12      #  弹出 Func1 函数的栈帧
                ret
# Func2 函数
    Func2:      addi    sp, sp, -16     #  为 Func3 函数分配栈帧
                sw      ra, 12(sp)      # x1（返回地址）
                sw      fp, 8(sp)       # x8（帧指针）
                addi    fp, sp, 12      #  调整帧指针
                sw      s1, 4(sp)       # x9（局部变量 z）
                sw      a1, 0(sp)       # x11(参数 y)
                mv      t0, a0
                mv      a0, a1
                mv      a1, t0
                jal     ra, Func3       # Func3 (y, x);
    # Func3 (y, x)*y
                mv      t1, a0
                lw      t2, 0(sp)       # y
```

```
    Multiply:
                    andi    t0, t0, 0       #x5，积
    loop:           beqz    t2, exit        #x6，乘数
                    andi    t3, t2, 1
                    beqz    t3, next
                    add     t0, t0, t1      #x7，被乘数
    next:           srli    t2, t2, 1
                    slli    t1, t1, 1
                    j       loop
    exit:           mv      s1, t0          # z = Func3 (y, x)*y;
                    mv      a0, s1          # return z;
                    lw      a1, 0(sp)       #恢复 x11
                    lw      s1, 4(sp)       #恢复 x9
                    lw      fp, 8(sp)       #恢复 x8
                    lw      ra, 12(sp)      #恢复 x1
                    addi    sp, sp, 16      # 弹出 Func3 函数的栈帧
                    ret
# Func3 函数
    Func3:          addi    sp, sp, -12     # 为 Func3 函数分配栈帧
                    sw      ra, 8(sp)       # x1（返回地址）
                    sw      fp, 4(sp)       # x8（帧指针）
                    addi    fp, sp, 8       # 调整帧指针
                    sw      s1, 0(sp)       # x9（局部变量 z）
                    mv      t5, x10
                    mv      t6, x11
        # 计算 t5/t6
        Divide:
                    andi    t0, t0, 0       # t0，商，初值为 0，即[31]位表示正数
                    addi    t1, x0, 0       # t1，余数，初值为 0，即[31]位表示正数
                    addi    t2, x0, 32      # 循环次数
                    lui     t3, 0x80000     # 掩码
        Dloop:      slli    t0, t0, 1       # 商的下一位，从[30]位开始
                    slli    t1, t1, 1       # 余数的下一位，从[30]位开始
                    and     t4, t5, t3      # 被除数的相应位是否为 1
                    beqz    t4, r0
                    ori     t1, t1, 1
        r0:         blt     t1, t6, Dnext   # 是否够减?
                    sub     t1, t1, t6      # 新的余数 <- 余数-除数
                    ori     t0, t0, 1       # 商的相应位上为 1
        Dnext:      slli    t5, t5, 1       # 被除数的下一位
                    addi    t2, t2, -1
                    beqz    t2, Dexit       # 做完 32 位
                    j       Dloop
        Dexit:      mv      s1, t0          # z = x/y;
```

```
mv      a0, s1          # return z;
lw      s1, 0(sp)       #恢复 x9
lw      fp, 4(sp)       #恢复 x8
lw      ra, 8(sp)       #恢复 x1
addi    sp, sp, 12      # 弹出 Func3 函数的栈帧
ret
```

8.5 对于如下计算第 *n* 个斐波纳契数的递归 C 函数：

```
int Fibonacci (int n)
{
    int sum;

    if (n == 0 || n==1)
        return 1;
    else {
        sum = (Fibonacci (n - 1) + Fibonacci (n - 2));
        return sum;
    }
}
```

1）请画出调用 Fibonacci(3)的运行时栈，并标出各单元的内容。

2）请写出 Fibonacci 函数的 RISC-V 汇编代码；

提示：Fibonacci 函数将形参 n 分配给参数寄存器 a0，将局部变量 sum 分配给寄存器 s1，将返回值分配给 a0；

3）将此递归函数转化为非递归函数，使用 for 循环实现；并给出非递归函数的 RISC-V 汇编代码；

4）对于非递归函数的计算，在不发生溢出的情况下，*n* 的取值最大可以是多少？

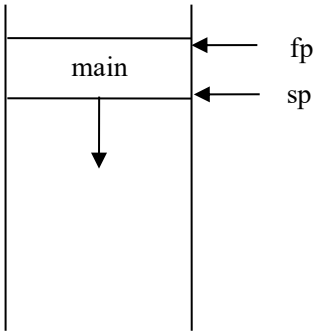
5）上机实践：对 Fibonacci 的循环结构版本与递归版本进行时间对比。

答：

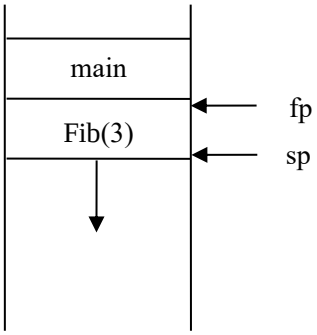
1）步骤 1~步骤 11，是 main 函数调用 Fib 函数的运行时栈情况

以步骤 4 为例，展示栈帧情况

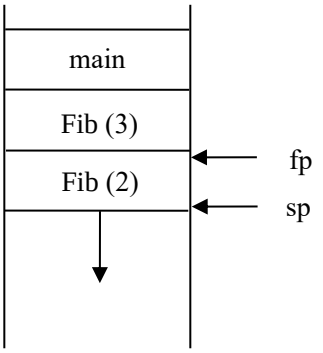
注：为了记录 Fib(n-1)的返回值，增加临时变量 temp，并为其分配 s2



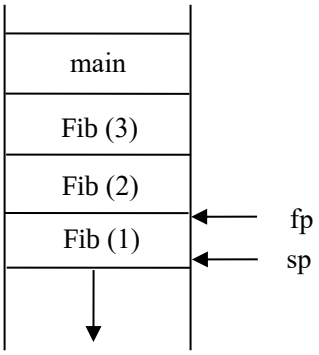
步骤 1：main 调用 Fib 前



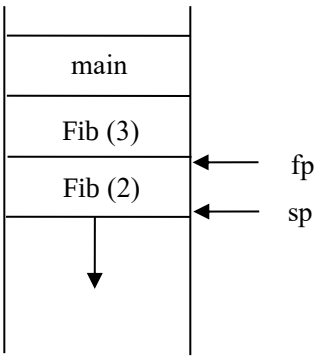
步骤 2：main 调用 Fib (3)



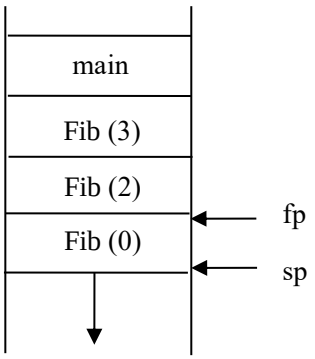
步骤 3：Fib (3)调用 Fib (2)



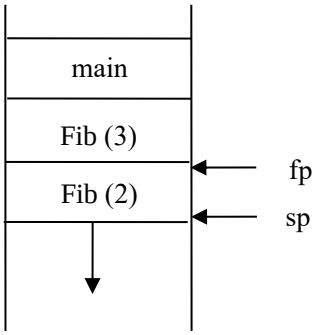
步骤 4：Fib (2)调用 Fib (1)



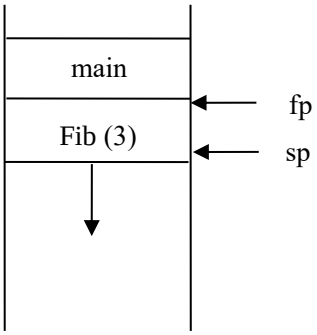
步骤 5：Fib (1)返回 Fib (2)



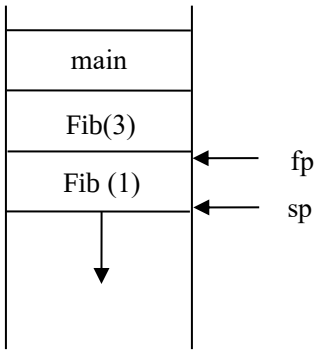
步骤 6：Fib (2)调用 Fib (0)



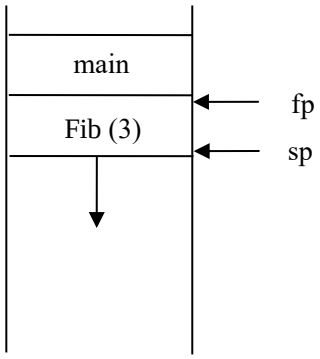
步骤 7：Fib (0)返回 Fib (2)



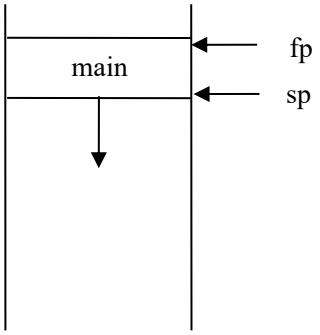
步骤 8：Fib (2)返回 Fib (3)



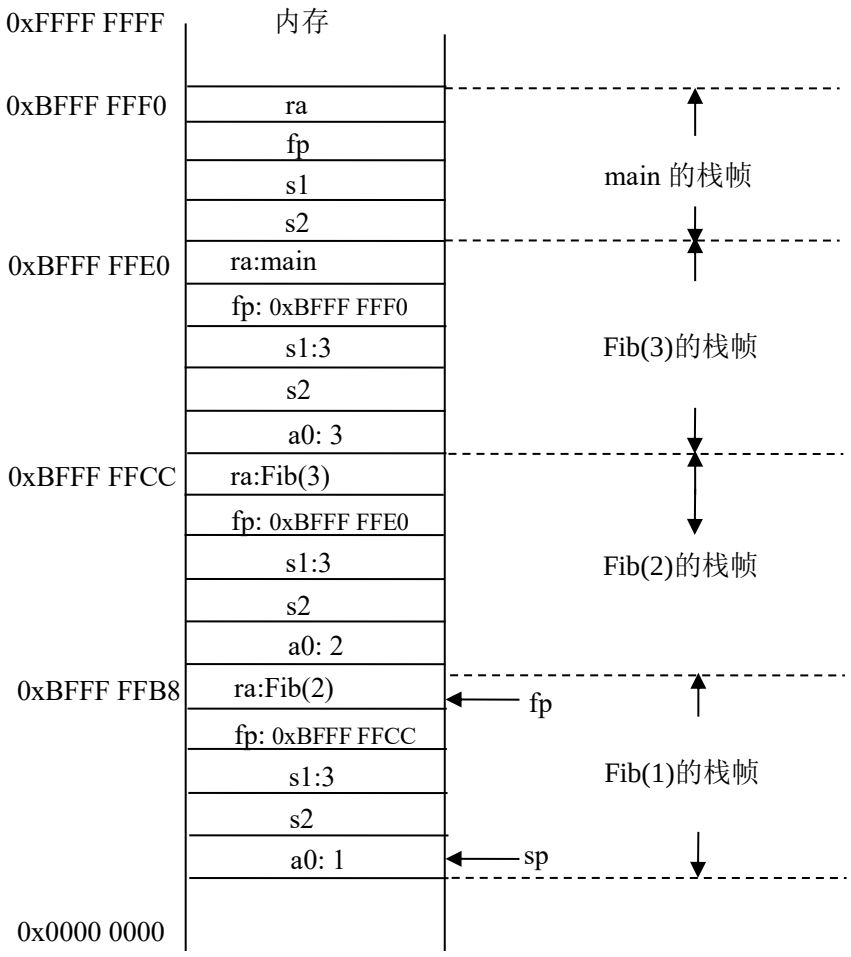
步骤 9：Fib (3)调用 Fib (1)



步骤 10：Fib (1)返回 Fib (3)



步骤 11：Fib (3)返回 main



2）

```
Fib:        addi    sp, sp, -20       # 为 Fib 函数分配栈帧
            sw      ra, 16(sp)        # x1（返回地址）
            sw      fp, 12(sp)        # x8（帧指针）
            addi    fp, sp, 16        # 调整帧指针
            sw      s1, 8(sp)         # x9（局部变量 sum）
            sw      s2, 4(sp)         # temp, Fib(n-1)的返回值
            sw      a0, 0(sp)         # x10（参数）
            beqz    a0, exit1         # n==0?
            li      t0, 1
            beq     a0, t0, exit1     # n==1?
            addi    a0, a0, -1        # n-1
            jal     ra, Fib           # Fib(n-1)
            mv      s2, a0            # Fib(n-1)的返回值
            lw      a0, 0(sp)         # 恢复 x10（n）
            addi    a0, a0, -2        # n-2
            jal     ra, Fib           # Fib(n-2)
            add     s1, s2, a0        # sum = Fib(n-1) + Fib(n-2);
            mv      a0, s1            # return sum;
            j       exit2
exit1:      li      a0, 1             # return 1; //Fib(0),Fib(1)
exit2:      lw      s2, 4(sp)         #恢复 temp
            lw      s1, 8(sp)         #恢复 x9
            lw      fp, 12(sp)        #恢复 x8
            lw      ra, 16(sp)        #恢复 x1
            addi    sp, sp, 20        # 弹出 Fib 函数的栈帧
            ret
```

3）

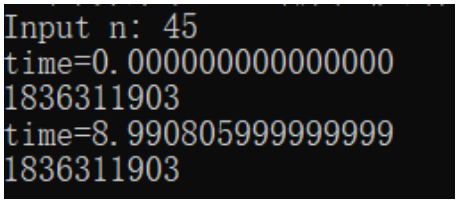
```
int i;
int f0 =1;
int f1 =1;
int sum;     //n>=2
for(i=2; i<=n; i++){
    sum = f0+ f1;
    f0 = f1;
    f1 = sum;
}
```

```
Fib:        addi    sp, sp, -24       # 为 Fib 函数分配栈帧
            sw      ra, 20(sp)        # x1（返回地址）
            sw      fp, 16(sp)        # x8（帧指针）
            addi    fp, sp, 20        # 调整帧指针
            sw      s1, 12(sp)        # x9（局部变量 i）
```

```
                sw      s2, 8(sp)       # x18（局部变量 f0）
                sw      s3, 4(sp)       # x19（局部变量 f1）
                sw      s4, 0(sp)       # x20（局部变量 sum）
                li      s2, 1           #f0=1;
                li      s3, 1           #f1=1;
                li      s1, 2           #i=2;
                addi    t0, a0, 1
loop:           bge     s1, t0, exit    # i<=n?
                add     s4, s2, s3      # sum=f0+f1;
                mv      s2, s3          # f0=f1;
                mv      s3, s4          # f1=sum;
                addi    s1, s1, 1       #i++
                j       loop

exit:           mv      a0, s4          # return sum;
                lw      s4, 0(sp)       #恢复  x20
                lw      s3, 4(sp)       #恢复  x19
                lw      s2, 8(sp)       #恢复  x18
                lw      s1, 12(sp)      #恢复  x9
                lw      fp, 16(sp)      #恢复  x8
                lw      ra, 20(sp)      #恢复  x1
                addi    sp, sp, 24      #  弹出 Fib 函数的栈帧
                ret
```

4）对于非递归函数的计算，在不发生溢出的情况下，*n* 的取值最大可以是 45，结果为 1836311903

5）见编程题——8-5.c，编译运行结果类似于下图：



9.1 对于如下程序：

```
#include <stdio.h>

int main() {
    int x = 1;
    int *ptr1;
    int **ptr2;

    ptr2 = &ptr1;
    *ptr2 = &x;
    **ptr2 = 2;

    x ++;
    (*ptr1) ++;
    (**ptr2) ++;

    printf ("%d \n", x);
}
```

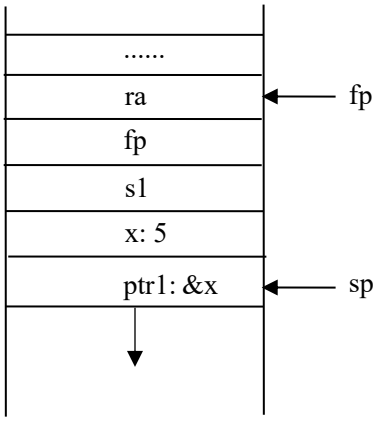
1）程序的输出是什么？提示：ptr2 是一个指向指针的指针。

2）请描述语句“(**ptr2) ++;”执行之后的运行时栈中的内容。提示：main 函数将局部变量 ptr2 分配给寄存器 s1，将局部变量 x 和 ptr1 分配到栈帧中。

答：

1）5

2）



9.2 将如下 C 函数翻译为 RISC-V 汇编代码。提示：Func 函数将形参 a 分配给参数寄存器 a0，将局部变量 sum 和 i 分配给寄存器 s1 和 s2，将返回值分配给 a0。

```
int Func (int *a) {
    int sum = 0;
    int i;
    for (i=0; i<5; i++)
        sum = sum + a[i];
    return sum;
}
```

```
Func:       addi    sp, sp, -8      # 分配栈帧
            sw      s1, 4(sp)       # s1（寄存器的保存）
            sw      s2, 0(sp)       # s2（寄存器的保存）
            li      s1, 0           #sum=0;
            li      s2,0            #i=0;
            li      t0,6
loop:       bge     s2,t0,exit      #i<5?
            slli    t1,s2,2
            add     t2,a0,t1
            lw      t3,0(t2)        #a[i]
            add     s1,s1,t3        #sum = sum + a[i];
            addi    s2,s2,1         #i++
            j       loop
exit:       mv      a0, s1          #return sum;
            lw      s2, 0(sp)       # 恢复寄存器 s2
            lw      s1, 4(sp)       # 恢复寄存器 s1
            addi    sp, sp, 8
            ret
```

9.3 将如下 C 函数翻译为 RISC-V 汇编代码。提示：StringLength 函数将形参 string 分配给参数寄存器 a0，将局部变量 index 分配给寄存器 s1，将返回值分配给 a0。

```
int StringLength (char string[]) {
    int index = 0;

    while (string[index] != '\0')
        index = index + 1;

    return index;
}
```

```
StringLength:   addi    sp, sp, -4      # 分配栈帧
                sw      s1, 0(sp)       # s1（寄存器的保存）
                li      s1, 0           #index=0;
loop:           add     t0,a0,s1
                lb      t1,0(t0)        #string[index]
                beqz    t1,exit         #string[index]!='\0'?
                addi    s1,s1,1         #index++;
                j       loop
exit:           mv      a0, s1          #return index;
                lw      s1, 0(sp)       # 恢复寄存器 s1
                addi    sp, sp, 4
                ret
```

9.4 如下 C 函数将一个字符串中的小写字母转换为大写字母，代码中存在 bug，请找出并修复。

```
char *ToUpper (char *inchar) {
        char str[10];
        int i = 0;

        while ( inchar[i] != '\0' ) {
            if ('a' <= inchar[i] && inchar[i] <= 'z')
                str[i] = inchar[i] - ('a' - 'A');
            else
                str[i] = inchar[i];
            i++;
        }
        return str;
}
```

答：ToUpper 函数中返回值是一个局部变量的指针，离开该函数后，不可再对其进行操作；根据题意可知，返回值应该仍然是参数 inchar。

解决方案——将 char str[10];改为 char *str=inchar;即可。

9.5 请将图 9.9 的冒泡排序程序中的 main 函数翻译为 RISC-V 汇编代码。提示：main 函数将局部变量 index 分配给寄存器 s1，将数组 numbers 分配到栈帧中；假设字符串"%d"和"%d\n"位于静态数据区，"%d"的首地址为 LC0，"%d\n"的首地址位于 LC1。

```
int main()
{
    int index;
    int numbers [10];

    for (index = 0; index < 10; index++)
    {
        scanf ("%d", &numbers[index]);
    }

    BubbleSort (numbers, 10);

    for (index = 0; index < 10; index++)
        printf ("%d\n", numbers[index]);
}
```

```
                .data
                .align      2
LC0:            .string     "%d"
LC1:            .string     "%d\n"
                .text
                .align      2
```

```
                .globl      main
main:           addi        sp, sp, -48         # 分配栈帧
                sw          ra, 44(sp)          # 保存返回地址
                sw          fp, 40(sp)          # 保存帧指针
                addi        fp, sp, 44          # 调整帧指针
                li          s1,0                #index=0;
                li          t0,10
loop1:          bge         s1,t0,next          #index<10?
                slli        t1,s1,2
                add         t1, t1,sp           #&numbers[index]
                la          a0, LC0             # 传递参数
                mv          a1,t1               # 传递参数
                call        scanf               # scanf("%d",&numbers[index]);
                addi        s1,s1,1             #index++
                j           loop1
next:           mv          a0,sp               # 传递参数
                li          a1,10               # 传递参数
                call        BubbleSort          #BubbleSort (numbers, 10);
                li          s1,0                #index=0;
                li          t0,10
loop2:          bge         s1,t0,next2         #index<10?
                slli        t1,s1,2
                add         t1, t1,sp           #&numbers[index]
                lw          a1,0(t1)            # numbers[index] ，传递参数
                la          a0,LC1              # 传递参数
                call        printf              # printf("%d\n",numbers[index]);
                addi        s1,s1,1             #index++
                j           loop2
next2:          li          a0, 0               # return 0;
                lw          ra, 44(sp)          # 恢复寄存器
                lw          fp, 40(sp)          # 恢复寄存器
                addi        sp, sp, 48          # 弹出栈帧
                ret                             # 返回
```

9.6 对于如下程序：

```
#include <stdio.h>

void P ( int * );

int a = 3;
int b = 4;

int main ( )
{
```

```
        int* p = &a;
        P ( p );
        printf ( "%d\n", *p );
}

void P ( int* p )
{
        p = &b;
        *p = 5;
}
```

1）程序的输出是什么？

2）请写出这段 C 程序的 RISC-V 汇编代码。

提示：a 和 b 是全局变量，main 函数将局部变量 p 分配给寄存器 s1；P 函数将形参 p 分配给参数寄存器 a0。

答：

1）3

2）

```
                .data
                .align      2
a:              .word       3
b:              .word       4
LC0:            .string     "%d\n"

                .text
                .align      2
                .globl      main
main:           addi        sp, sp, -12         #  分配栈帧
                sw          ra, 8(sp)           #  保存返回地址
                sw          fp, 4(sp)           #  保存帧指针
                addi        fp, sp, 8           #  调整帧指针
                la          s1, a               # int* p = &a;
                mv          a0, s1              #  传递参数 p
                call        P                   #P ( p );
                lw          a1,0(s1)            # *p，传递参数
                la          a0,LC0              #  传递参数
                call        printf              # printf("%d\n",*p);
                li          a0, 0               # return 0;
                lw          ra, 8(sp)           #  恢复寄存器
                lw          fp, 4(sp)           #  恢复寄存器
                addi        sp, sp, 12          #  弹出栈帧
                ret                             #  返回
# P 函数比较简单，栈帧为空
P:              la          a0,b                #p = &b;
                li          t0,5
```

```
sw          t0,0(a0)
ret
```

10.1 对于如下 switch 语句：

```
int result = 0;
switch (x) {
case 1:
    result += 1;
    break;
case 2:
    result += 2;
case 3:
    result += 3;
    break;
case 4:
case 5:
    result += 5;
    break;
default:
    result = 0;
}
```

1）请使用 GCC 编译器，查看该 switch 语句的汇编代码；

2）将此 switch 语句转化为级联的 if-else 语句，查看级联的 if-else 语句的汇编代码；

3）将级联的 if-else 语句与 switch 语句的汇编代码进行对比：当 x 的值分别为 0、1、2、3、4、5 和 6 时，执行的指令数目。

答：

1）见编程题——10-1-1.s

2）见编程题——10-1-2.s

3）从 main 开始，到 L3（不包括 printf 调用）：

x	switch	if
0	15	18
1	16	11
2	17	14
3	16	14
4	16	17
5	16	19
6	10	18

10.2 对于如下程序：

```
#include <stdio.h>

void Swap (int x, int y);

int main()
{
    int x = 1;
    int y = 2;
```

```
        printf ("x = %d, y = %d\n ", x, y);

        Swap (x, y);
        printf ("x = %d, y = %d\n ", x, y);
}

void Swap (int x, int y)
{
        int temp;

        temp = x;
        x = y;
        y = temp;
        printf ("x = %d, y = %d\n ", x, y);
}
```

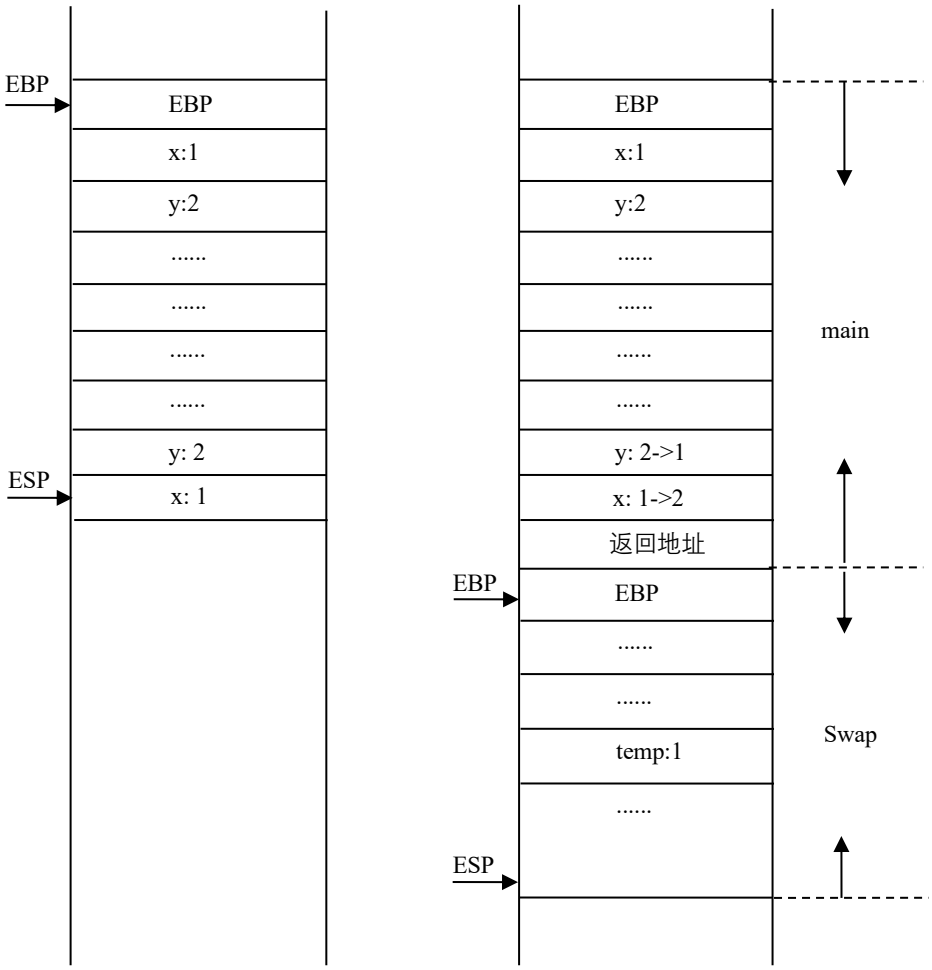
1）程序的输出是什么？

2）请使用 GCC 编译器，生成 X86 汇编文件，并查看该汇编文件，画出调用 Swap 函数前后的运行时栈的情况，并标出相关单元的内容。

答：

1）x = 1, y = 2
x = 2, y = 1
x = 1, y = 2

2）见编程题——10-2.s



(a) 调用 Swap 函数前　　(b) 进入 Swap 函数

10.3 对于如下计算第 *n* 个斐波纳契数的递归 C 函数:

```
int Fibonacci (int n)
{
    int sum;

    if (n == 0 || n==1)
        return 1;
    else {
        sum = (Fibonacci (n - 1) + Fibonacci (n - 2));
        return sum;
    }
}
```

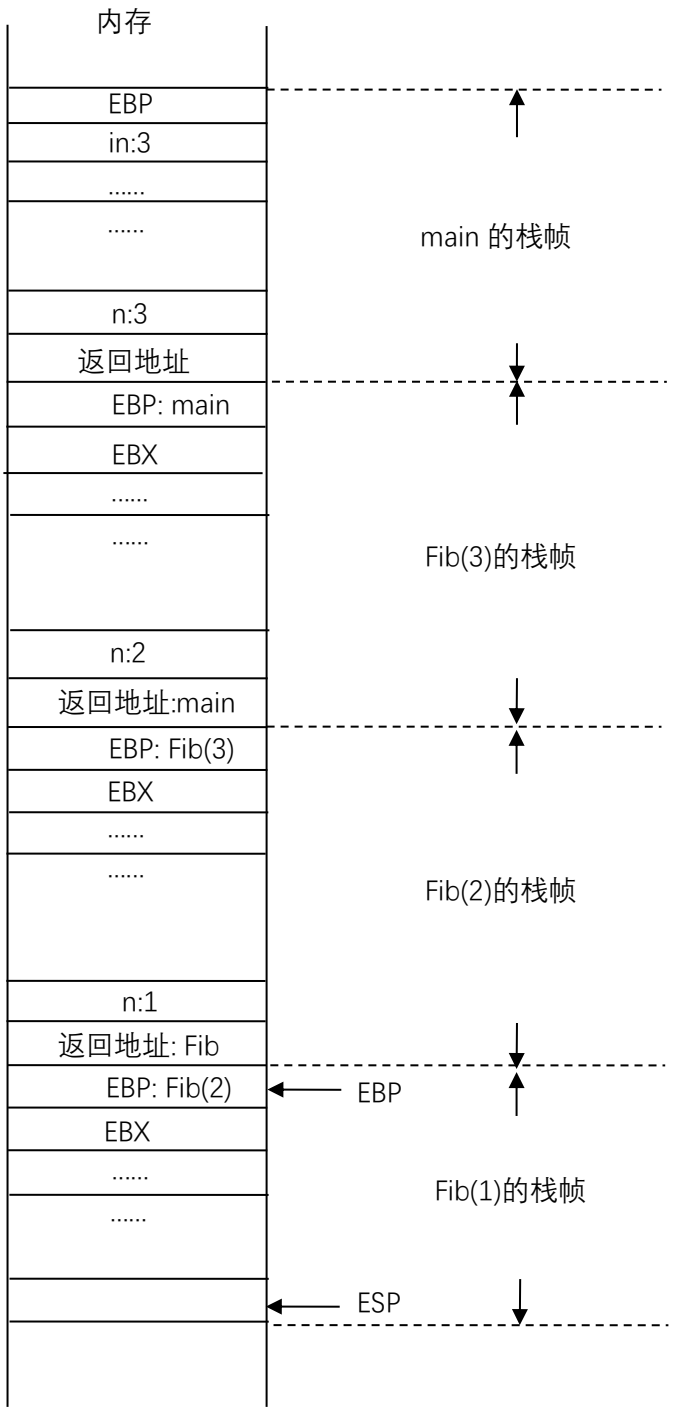
1）请使用 GCC 编译器，生成 X86 汇编文件，并查看该汇编文件，画出调用 Fibonacci(3)的运行时栈情况，并标出相关单元的内容。

2）将此递归函数转化为非递归函数，使用 for 循环实现；再次查看新的汇编文件，画出调用 Fibonacci 函数前后的运行时栈情况，并标出相关单元的内容。

答:

1）见编程题——10-3-1.s

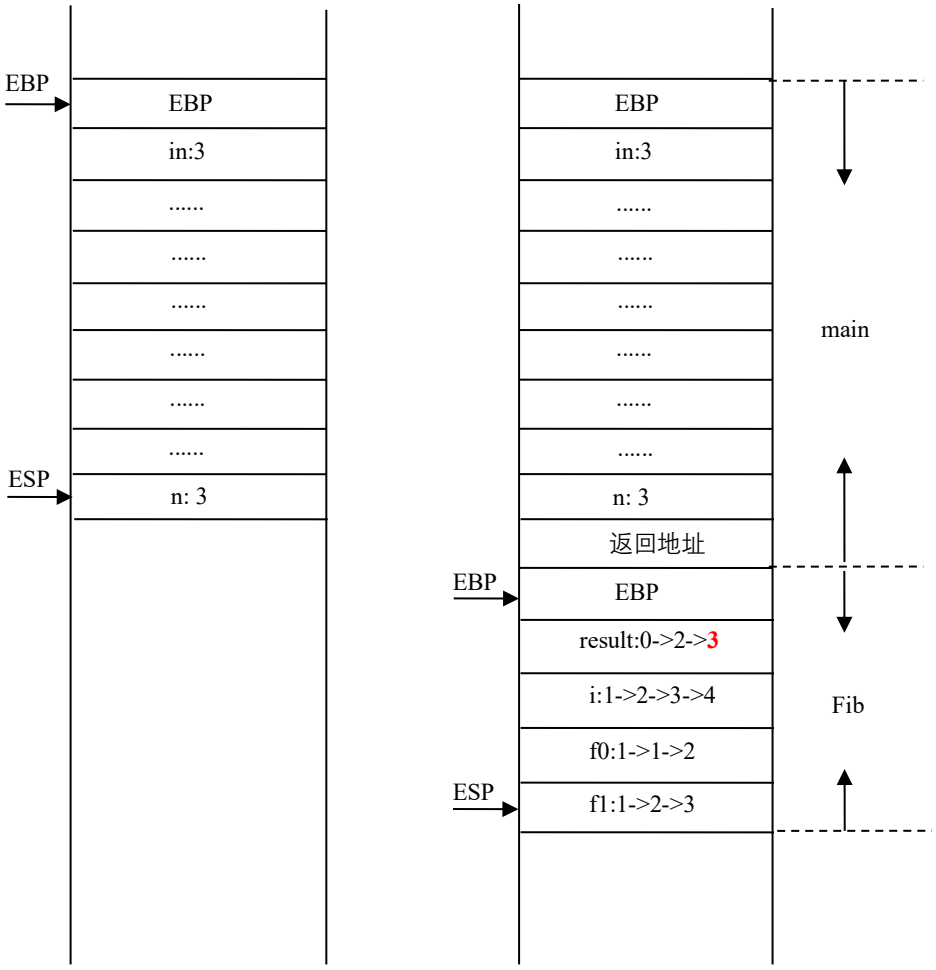
调用 Fibonacci(3)的运行时栈情况与习题 8.5 类似，给出步骤 4 的栈帧情况



2）见编程题——10-3-2.s

```
int Fibo1 (int n)
{
  int result=0;
  int i=1;
```

```
  int f0=1;
  int f1=1;

  for(i=2;i<=n;i++){
        result=f0+f1;
        f0=f1;
        f1=result;
  }
  return result;
}
```



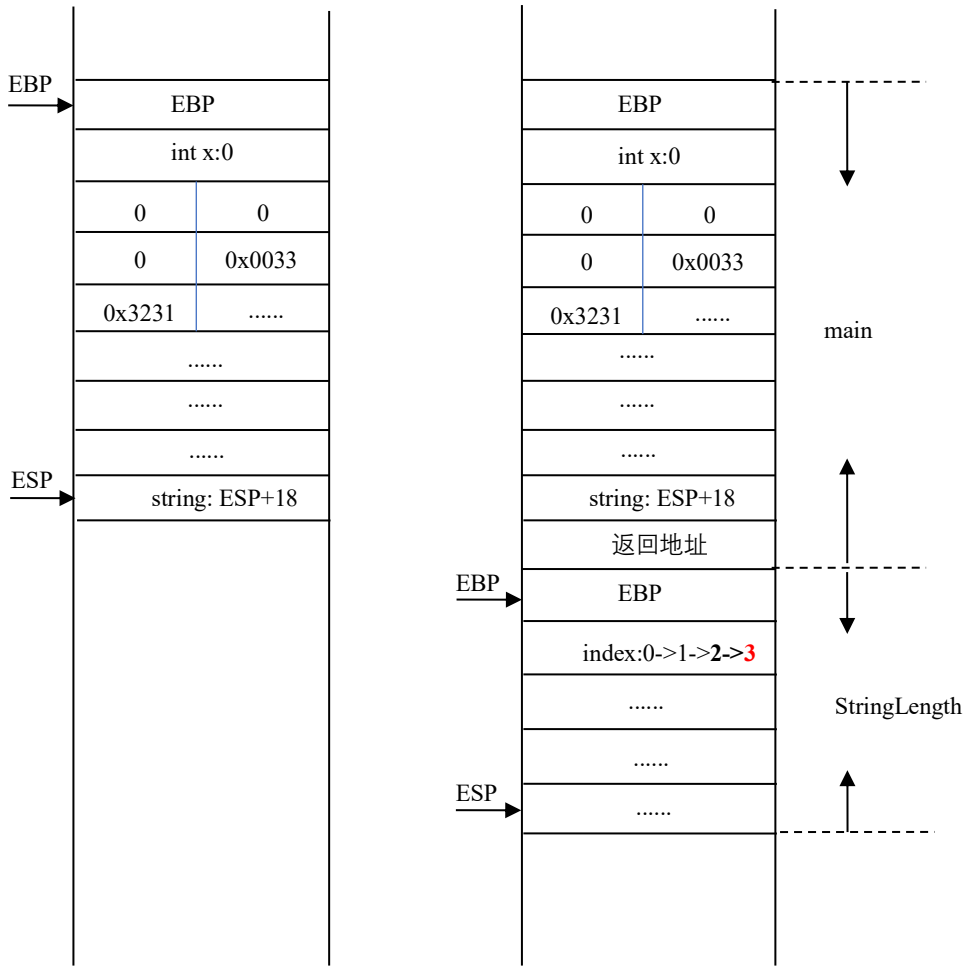
(a) 调用 Fib 函数前　　(b) 进入 Fib 函数

10.4 请使用 GCC 编译器，查看 StringLength 函数的 X86 汇编代码，画出调用 StringLength 函数前后的运行时栈情况，并标出相关单元的内容。提示：char 类型仅占一个字节。

```
int StringLength (char string[]) {
    int index = 0;

    while (string[index] != '\0')
```

```
        index = index + 1;

    return index;
}
```

答：见编程题——10-4.s
以 char string[10]="abc"为例



(a) 调用 StringLength 函数前　　(b) 进入 StringLength 函数

10.5 对于如下程序:

```
#include <stdio.h>

void P ( int * );

int a = 3;
int b = 4;

int main ( )
```

```
{
        int* p = &a;
        P ( p );
        printf ( "%d\n", *p );
}

void P ( int* p )
{
        p = &b;
        *p = 5;
}
```

1）程序的输出是什么?

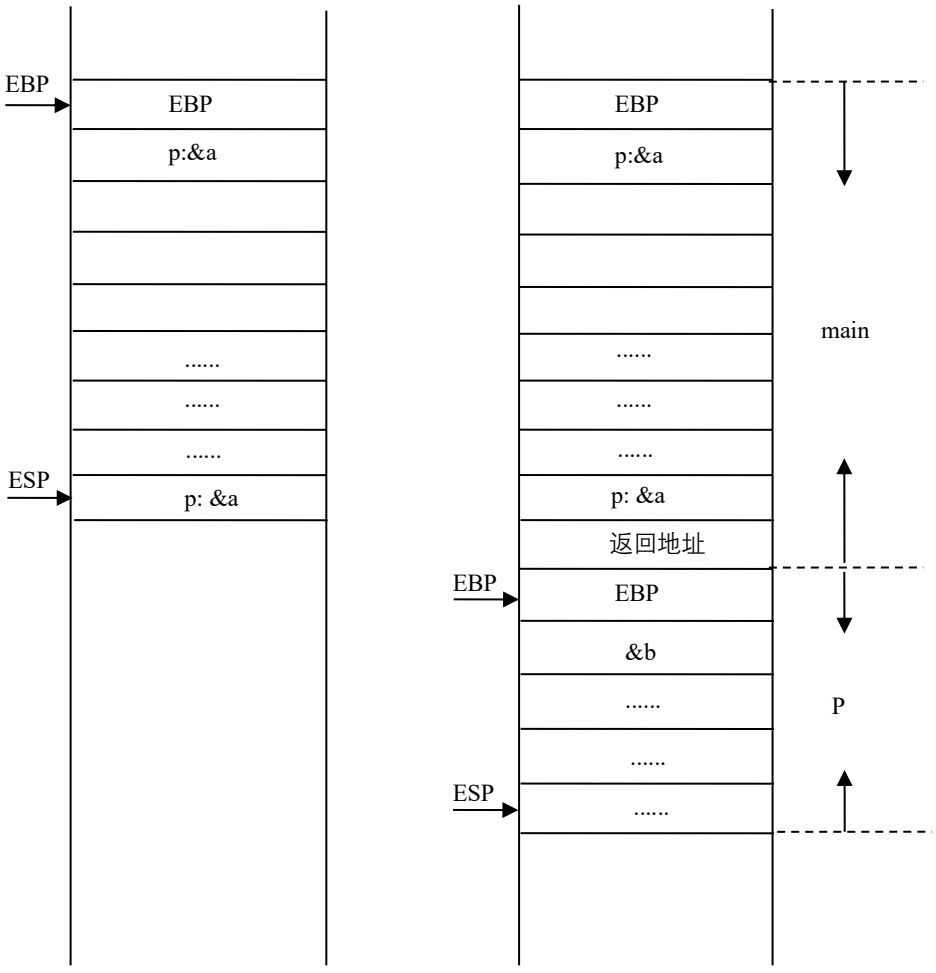
2）请使用 GCC 编译器，生成 X86 汇编文件，并查看该汇编文件，画出调用 P 函数前后的运行时栈和静态数据区的情况，并标出相关单元的内容。

提示：a 和 b 是全局变量，位于静态数据区（.data)。

答:

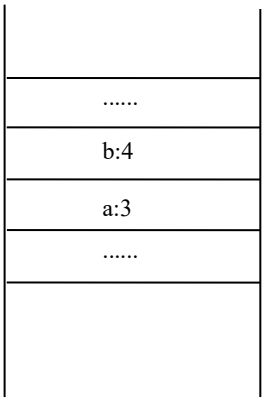
1）3

2）见编程题——10-5.s

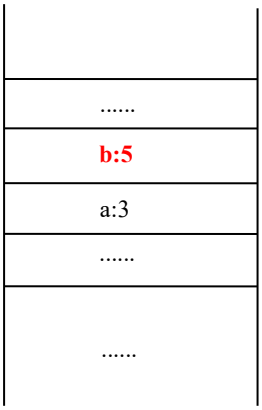


(a) 调用 P 函数前　　(b) 进入 P 函数

静态数据区：



(c) 调用 P 函数前



(d) 进入 P 函数