1. **为什么%rbp一直要-24?**

这是指定栈中的内存地址，一般我们进入函数体之前需要开辟栈帧，将局部变量存放在其中，在代码中一般都是：

```
pushq   %rbp
movq    %rsp, %rbp
subq    %30, %rsp

// 上面这几行代码是向内存申请30个字节
movl    %eax, -24(%rbp)
// 这里是为了问题这样写
// 该行代码就是将eax寄存器里面的内容存到指定位置，该位置是哪里，是rbp - 24处的内存
```

比如说代码的Line154：

```
movq    %rdi,-24(%rbp)      ## move the pointer into memory,the size is 8 Bytes
```

这里就是rdi寄存器中传进来的变量存放在指定的栈帧的指定内存地址处，由于寄存器中存放的变量的大小是8字节，即4个字，所以要使用movq。

2. **Line 30-92：没看懂，为什么要这么写？这一大段每一个部分都是拿来干嘛的，整体的思路是什么样子的?**

为什么要这么做？？？？？

```
int find_p_bin_str(int p)
{
    int i;

    for (i = 0; i < 6; i++) {
        if (p >= p_bin[i].min && p <= p_bin[i].max) {
            return p_bin[i].bin_str;
        }
    }

    return 0;
}
```

该函数是源文件中的一个函数，客户要求将该文件中所有的函数全部转换为汇编格式的代码，所以就有了Line30-92。

3. **Line 157-158：功能有重合，为什么这样做?**

```
movzbl  BATT_STATUS_PORT(%rip),%eax     ## zero extension
movzbl  %al,%eax
```

该处代码就是为了将宏变量加载到寄存器中，这里一开始是先将一个字节的BATT_STATUS_PORT变量存放到寄存器eax中，并且将多余的位数补零，随后的操作的时候，这里是个人习惯所致，再将低8位取出并进行0扩展，双保险，确保eax寄存器中的确是我们想要的值。

4. Line 172：为何使用movzwl?

```
movzwl BATT_VOLTAGE_PORT(%rip),%eax
```

这里为什么要使用movzwl，你肯定是这样想的，为什么上面用的是movzbl但是这里使用的却是movzwl，这很简单，因为对应的变量的大小是不一样的，ATT格式的汇编，好处就是操作符会说明后面的操作数的位数。

```
// batt.h文件中
extern short BATT_VOLTAGE_PORT;

extern unsigned char BATT_STATUS_PORT;

extern int BATT_DISPLAY_PORT;
```

大小分别为2字节(movw)、1个字节(movb)、4个字节(movl).

这里是一个字的大小向4字的大小进行扩展，自然是使用movzwl,顺便提一下，z代表的而是0扩展。

5. Line 204：为什么要用movzwq?

```
typedef struct{
  short mlvolts;     // voltage read from port, units of 0.001 Volts (milli Volts)
  char  percent;     // percent full converted from voltage
  char  mode;        // 1 for percent, 2 for volts, set based on bit 4 of BATT_STATUS_PORT
} batt_t;

movq -24(%rbp),%rax
movzwq (%rax),%rax
```

batt指针指向的该结构体，因为此处是需要取出mlvolts变量的值，所以说我们先将指针指向的地址先取出来，存放到eax寄存器当中,接着，直接告诉编译器你从这个地址开始取出变量，该变量的大小是一个字，即mlvolts，并将其扩展成64位的大小，存放到rax寄存器当中。

6. Line 205：为啥用sub而不用mov?

这里执行的是`batt->mlvolts - 3000`,是一个减法，肯定要使用`sub`进行减法操作。

7. Line 251：为啥用Leave?

`leave`的作用就是回复栈帧，因为在前面我们已经申请了很多的内存了，然后再调用`ret`结束子函数并返回到主函数当中。

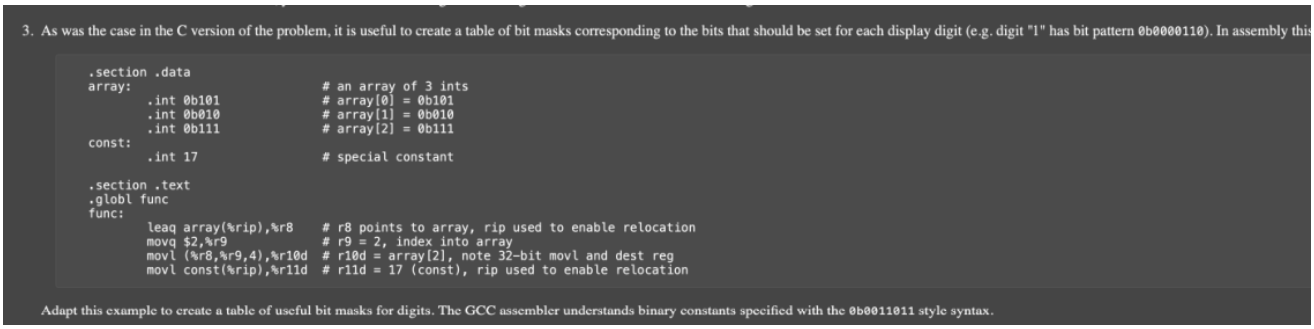
`leave`也可以使用以下语句进行代替：

```
movq    %rbp, %rsp
popq    %rbp
```

8. Line 269 – 277：这些变量都是干什么用的?

这些变量全部都是函数中的局部变量，这里画图是为了下面比较好的进行存取数据。

9. Line 450-451：r8，r9分别是啥，为何在这里要对他们进行清零?


3. As was the case in the C version of the problem, it is useful to create a table of bit masks corresponding to the bits that should be set for each display digit (e.g. digit "1" has bit pattern 0b0000110). In assembly this

```
.section .data
array:                  # an array of 3 ints
        .int 0b101      # array[0] = 0b101
        .int 0b010      # array[1] = 0b010
        .int 0b111      # array[2] = 0b111
const:
        .int 17         # special constant

.section .text
.globl func
func:
        leaq array(%rip),%r8    # r8 points to array, rip used to enable relocation
        movq $2,%r9             # r9 = 2, index into array
        movl (%r8,%r9,4),%r10d  # r10d = array[2], note 32-bit movl and dest reg
        movl const(%rip),%r11d  # r11d = 17 (const), rip used to enable relocation
```

Adapt this example to create a table of useful bit masks for digits. The GCC assembler understands binary constants specified with the 0b0011011 style syntax.

这是所给文档中提供参考的一种寻址方式，具体内容可以自己搜索相关知识查看。

这里清零是因为我们一般使用寄存器之前应该有一个清零操作，防止寄存器中的数据干扰我们的程序。