汇编语言第8次作业

找一段包含嵌入式汇编的C代码或函数，将其转换为汇编代码，并给出注释和分析报告。
可以从linux/include/asm-i386/string.h选取
函数中把static inline去掉，gcc -m32 -O1 -S xxx.c
可以从ffmpeg、x264等代码中选取
也可以从其他开源代码中选取，如github中

选取代码

所选取的代码是`linux/arch/x86/include/asm /checksum_32.h`中的`ip_fast_csum`函数，函数源代码如下：

```
static inline __sum16 ip_fast_csum(const void *iph, unsigned int ihl)
{
    unsigned int sum;

    asm("  movl (%1), %0\n"
        "  subl $4, %2\n"
        "  jbe 2f\n"
        "  addl 4(%1), %0\n"
        "  adcl 8(%1), %0\n"
        "  adcl 12(%1), %0\n"
        "1: adcl 16(%1), %0\n"
        "  lea 4(%1), %1\n"
        "  decl %2\n"
        "  jne  1b\n"
        "  adcl $0, %0\n"
        "  movl %0, %2\n"
        "  shrl $16, %0\n"
        "  addw %w2, %w0\n"
        "  adcl $0, %0\n"
        "  notl %0\n"
        "2:"
    /* Since the input registers which are loaded with iph and ihl
       are modified, we must also specify them as outputs, or gcc
       will assume they contain their original values. */
        : "=r" (sum), "=r" (iph), "=r" (ihl)
        : "1" (iph), "2" (ihl)
        : "memory");
    return (__force __sum16)sum;
}
```

汇编代码

将`static inline`删去后使用命令`gcc -m32 -O1 -S ip_fast_csum.c`生成的汇编代码。
在实际生成中，还添加了

```
#include <stdint.h>
typedef uint16_t __u16;
typedef __u16 __sum16;
```

保证__sum16的定义存在并和源代码一致
此外，还将

```
return (__force __sum16)sum;
```

改为了

```
return (__sum16)sum;
```

因为__force是Linux内核中的宏定义，不适用于普通的C代码。

生成的汇编代码如下：

```
    .file   "ip_fast_csum.c"
    .text
    .globl  ip_fast_csum
    .type   ip_fast_csum, @function
ip_fast_csum:
.LFB0:
    .cfi_startproc
    movl    4(%esp), %edx
    movl    8(%esp), %ecx
#APP
# 10 "ip_fast_csum.c" 1
      movl (%edx), %eax
  subl $4, %ecx
  jbe 2f
  addl 4(%edx), %eax
  adcl 8(%edx), %eax
  adcl 12(%edx), %eax
1: adcl 16(%edx), %eax
  lea 4(%edx), %edx
  decl %ecx
  jne   1b
  adcl $0, %eax
  movl %eax, %ecx
  shrl $16, %eax
  addw %cx, %ax
  adcl $0, %eax
  notl %eax
2:
# 0 "" 2
```

```
#NO_APP
    ret
    .cfi_endproc
.LFE0:
    .size   ip_fast_csum, .-ip_fast_csum
    .ident  "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
    .section    .note.GNU-stack,"",@progbits
```

解释说明

函数ip_fast_csum的输入是IP报头指针iph和IP报头长度ihl，输出是一个16位的校验和，在程序中对应的是sum变量。 下面对嵌入的汇编代码进行解释：

约束：

```
        : "=r" (sum), "=r" (iph), "=r" (ihl)
        : "1" (iph), "2" (ihl)
        : "memory");
```

- "memory" 表示内存被该嵌入式汇编代码读取或写入
  - 这种约束的作用是告诉编译器，在执行这段汇编代码之前和之后，必须考虑内存状态的变化，不能对内存访问进行优化
- 输入参数iph和ihl被修改，因此在输出部分也列出
  - 使用1和2，表示前后的寄存器是同一个

代码逐行解释

调用函数时

```
    movl    4(%esp), %edx
    movl    8(%esp), %ecx
```

- iph存在esp+4的位置，将其移入%edx中
- ihl存在esp+8的位置，将其移入%ecx中

然后执行：

```
    movl (%edx), %eax
```

- 将iph指向的地址处的第一个32位数据加载到sum中

```
    subl $4, %ecx
```

- ihl减去 4，因为已经加载了一个32位数据

```
    jbe 2f
```

- 检查剩余长度是否小于等于零，如果 `ihl` 小于等于零，则跳转到标签 `2` 结束计算
    - 标签中的`f`表示`forward`，表示向前跳转

```
    addl 4(%edx), %eax
    adcl 8(%edx), %eax
    adcl 12(%edx), %eax
```

- 累加`iph`指向的地址处接下来的三个32位数据到`sum`中
    - 使用`addl`和`adcl`指令，`adcl`指令会将前一次操作的进位加到结果中。

```
1: adcl 16(%edx), %eax
    lea 4(%edx), %edx
    decl %ecx
    jne 1b
```

- 循环处理剩余的数据
    - 使用标签`1`作为循环的起始。
    - `adcl 16(%1), %0`：将下一个32位数据累加到`sum`中。
    - `lea 4(%1), %1`：将`iph`指针向前移动4字节。
    - `decl %2`：将`ihl`减1。
    - `jne 1b`：如果`ihl`不为零，跳转到标签`1`继续循环。
        - 标签中的`b`表示`backward`，表示向后跳转

```
    adcl $0, %eax
```

- 将最终的进位加到`sum`中

```
    movl %eax, %ecx
    shrl $16, %eax
    addw %cx, %ax
    adcl $0, %eax
```

- 将高16位和低16位相加
    - `movl %0, %2`：将`sum`复制到`ihl`。
    - `shrl $16, %0`：将`sum`右移16位，得到高16位。
    - `addw %w2, %w0`：将低16位和高16位相加。
    - `adcl $0, %0`：如果有进位，则再加上进位。

```

```

```
    notl %eax
```

- 对 `sum` 取反，得到最终的校验和。

```
2:
```

- 标签 `2` 用于跳转出计算

```
    ret
```

- 退出函数，返回结果