Pa\_1-1

C语言中的struct和union关键字都是什么含义，寄存器结构体的参考实现为什么把部分struct改成了union？

①struct和union关键字的含义：

struct: 结构体. 结构体内的每个数据成员都有各自独立的存储空间. 结构体的大小与各数据成员的大小有关.

union: 联合体. 联合体只分配和最大数据成员所需存储空间相同的空间, 所有的数据成员都共享这一空间, 修改某一成员的同时可能会影响到其他成员. 联合体的使用会受到系统大端、小端存储方式的影响.

②参考实现中的修改的对应解释：

寄存器的参考实现如下：

typedef struct {

union {

union {

union {

uint32\_t \_32;

uint16\_t \_16;

uint8\_t \_8[2];

};

uint32\_t val;

} gpr[8];

struct { // do not change the order of the registers

uint32\_t eax, ecx, edx, ebx, esp, ebp, esi, edi;

};

};

} CPU\_STATE;

|  |  |
| --- | --- |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |

最内层的union (line 4 ~ 8)实现的是单个32位寄存器，并定义其低16位(line 6)，低8位、高8位(line 7)，如下所示：

31 23 15 7 0

╔═════════════════╪═════════════════╬═════════════════╧═════════════════╗

║ \_32 \_8[1] \_16 \_8[0] ║

╠═════════════════╪═════════════════╬═════════════════╩═════════════════╣

中间的union(line 3 ~ 10)实现的是eax, ecx, edx, ebx, esp, ebp, esi, edi这8个32位寄存器组，line9处的val变量和内层的union声明的变量共享同一空间，相当于line5处的\_32变量.

外层的union(line 2 ~ 14)将中间层union实现的8个寄存器和eax, ecx, edx, ebx, esp, ebp, esi, edi一一对应，struct(line 11)中的各个32位变量和前面实现的各个32位寄存器共享内存空间.

最内层的

Pa\_1-3

为浮点数加法和乘法各找两个例子：

1. 对应输入是规格化或非规格化数，而输出产生了阶码上溢结果为正（负）无穷的情况；
2. 对应输入是规格化或非规格化数，而输出产生了阶码下溢结果为正（负）零的情况。是否都能找到？若找不到，说出理由。
3. 浮点数加法：

正无穷：

0x7f7fffff + 0x7f000001 = 0x7f800000

负无穷：

0xff7fffff + 0xff000001 = 0xff800000

浮点数乘法:

正无穷：

0x7f000000 \* 0x40000000 = 0x7f800000

负无穷：

0xff000000 \* 0x40000000 = 0xff800000

1. 浮点数加法：

正零：

0x00000001 + 0x80000001 = 0x00000000

负零：

找不到合适的例子. 在fpu.c文件的uint32\_t internal\_float\_add(uint32\_t b, uint32\_t a)函数中，浮点数的阶码相加得到的结果对符号位的影响的相关代码如下所示：

sig\_res = sig\_a + sig\_b;

//如果结果为负数的处理：

if (sign(sig\_res))

{

f.sign = 1;

sig\_res \*= -1;

}

else

{

f.sign = 0;

}

要得到负零，首先就要求尾数相加得0.而代码对尾数相加的实现中，是通过无符号数（补码）相加的方式得到结果，0的表示方式是唯一的（即正零），故无法得到符号位是1的负零.

浮点数乘法：

正零：

0x00000001 \* 0x00000001 = 0x00000000

负零：

0x80000001 \* 0x00000001 = 0x80000000