Pa 1.1.1

寄存器组织形式：

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;

通过Union的巧妙运用，来实现对32位寄存器的不同表示方式cpu.eax 与 cpu.gpr[0].\_32 等价

此外，利用memory 大数组来模拟实现主存

Pa 1.2 整数运算

CF : 针对无符号数的运算 carry.

特别的，在移位运算中，它表示最后一位被移出的位。

PF: 低八位的奇偶位

OF：针对有符号数的 overflow.

ZF ： 零符号

SF: 符号位（根据data\_size来得知）

下面给出各个运算过程中的flag计算：

|  |  |  |
| --- | --- | --- |
| Operator | CF（特别关注无符号数的范围即可） | OF |
| Add dest src | Result<dest | 若操作数同号，且结果与dest不同号则置1 |
| Sub dest src | Result>dest | 若操作数异号，且结果与dest不同号则置1 |
| Mul | 高位不为0 则置1 | 高位不为0 则置1 |
| iMul | ———— | —————— |
| Div | ———— | —————— |
| iDiv | ———— | —————— |
|  |  |  |

此外：

掩码应用： （0xFFFFFFFF>>(32-data\_size)）& src;

取高位数： src>>(data\_size)<<data\_size;

Pa 1.3 浮点数运算

这一部分细节性内容较多，代码写起来也比较琐碎

1. Add
2. 获取有效位significand :规格化 f.sig | 0x800000

非规格化 f.sig

1. 左移3位，空出相应的GRS

确定阶数时，必须考虑到非规格化数的exp大小应该为1

1. 确定小阶需要右移的shift =(b.exp==0?1: b.exp) – (a.exp==0? 1: a.exp)

（对阶）右移的同时 sticky bit 一直被放置于最低位

1. 尾数加减： 先将尾数变为补码表示，再根据结果判断 Sign

再把尾数变回原码

if (sign(sig\_res))

{

f.sign = 1;

sig\_res \*= -1;

}

1. 规格化与舍入
2. Multiple
3. Division