计算机系统基础
Programming Assignment

PA 1 数据的表示、存取和运算

2017年9月8日

“一名优秀的厨师应当首先对各种食材了如指掌并掌握各种基本的烹饪技巧。”

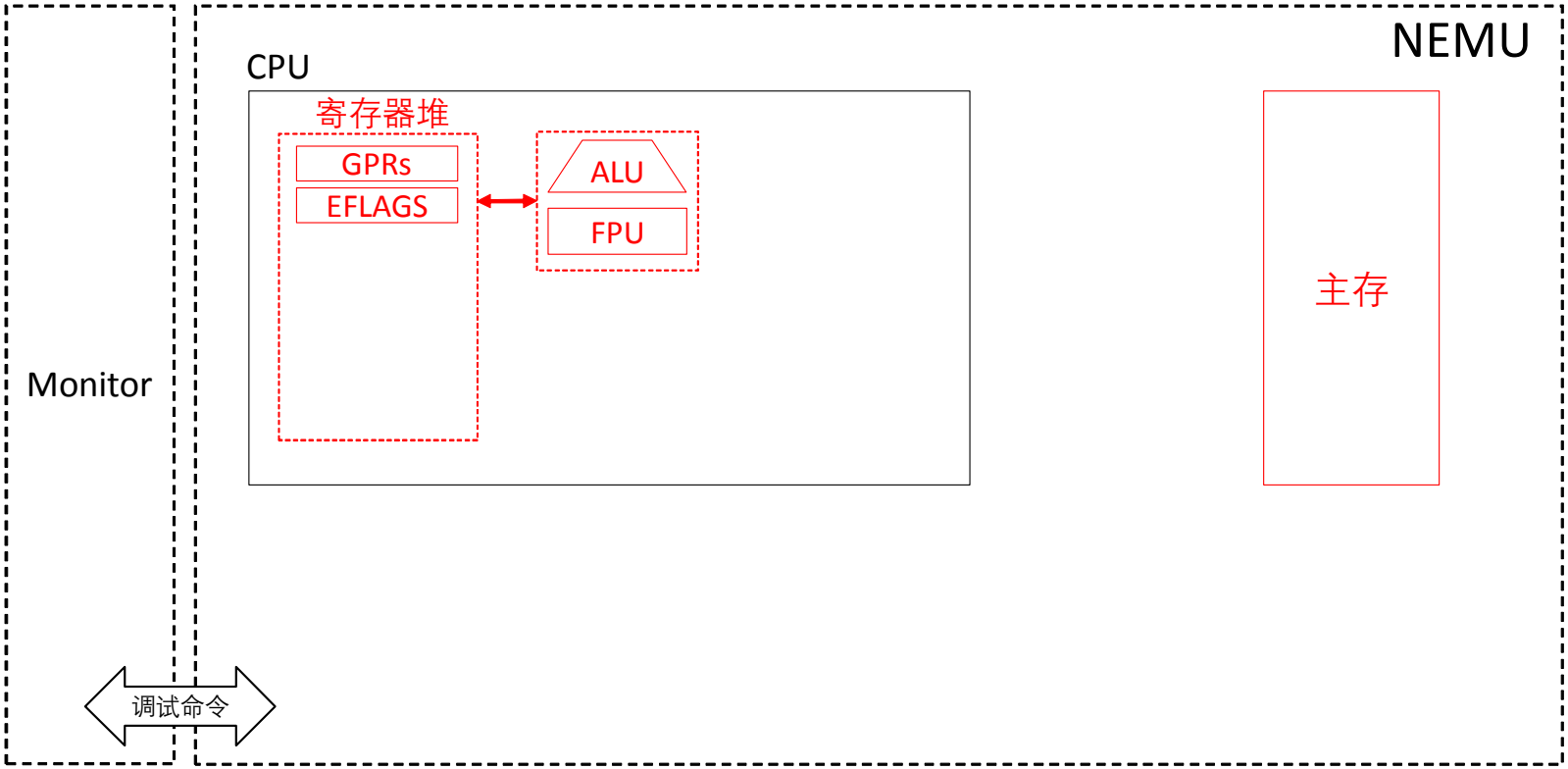
—— *L. Wang*

数据 = 食材

PA 1 – 目录

- PA 1-1 数据的类型和存取
- PA 1-2 整数的表示和运算
- PA 1-3 浮点数的表示和运算

PA 1 – 路线图



PA 1-1 数据的类型和存取

PA 1-1 数据的类型和存取

- 数据基本可以分为两个大类
  - 非数值型数据
    - 各类文字的编码
    - 指令的操作码
  - 数值型数据（重点关注）
    - 整数
    - 浮点数
  - 在计算机内，所有类型的数据（包括代码）都表现为01串

PA 1-1 数据的类型和存取

- NEMU模拟的是i386体系结构
  - 数据存储的最小单位是比特（bit）
  - 数据存储的基本单位是字节（byte）
    - 典型长度为：1字节、2字节、4字节
  - 内存中采用小端方式存储
    - 对于超过一个字节的数据
    - 低有效位的字节在前（低地址），高有效位的字节在后（高地址）

想表达的数字	内存中的数字
0x12345678	0x78 0x56 0x34 0x12

低地址　　高地址

PA 1-1 数据的类型和存取

“巧妇难为无米之炊，没有锅她也不行。”

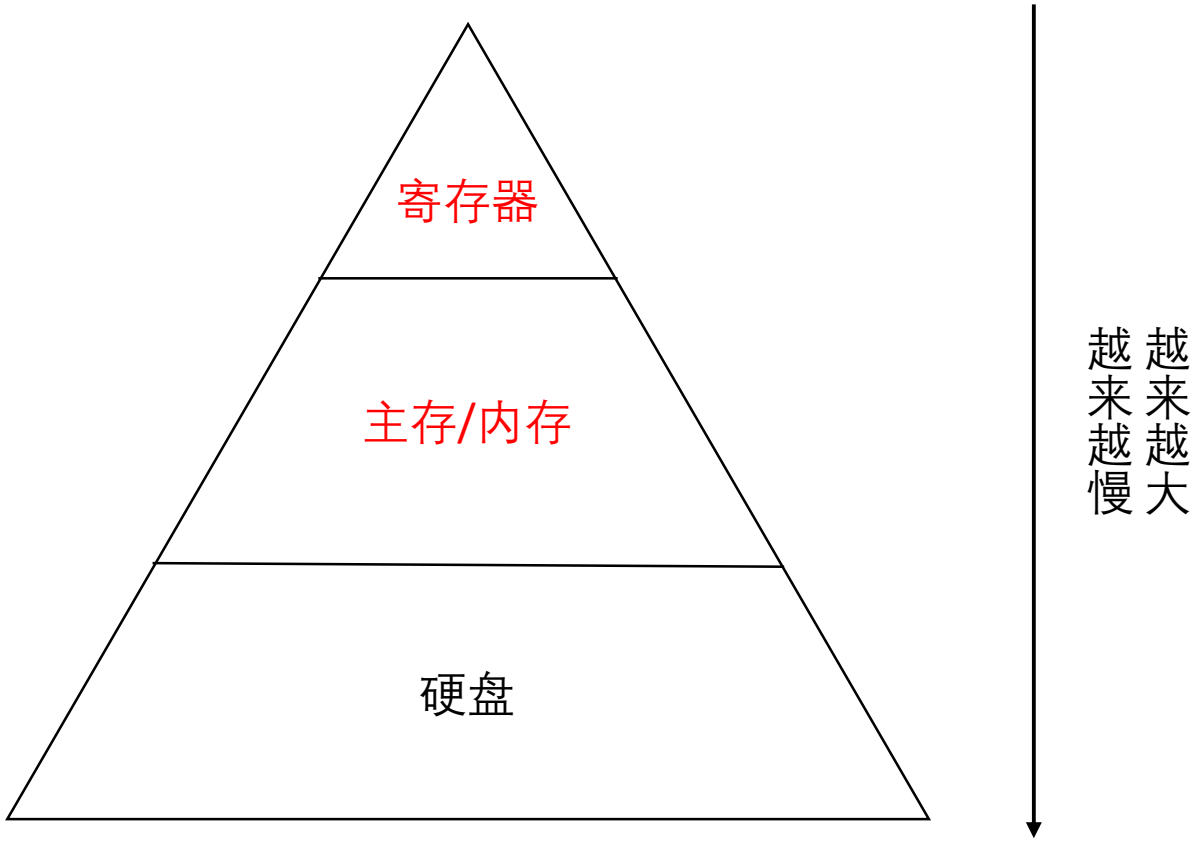
—— L. Wang

我们首先要为待处理的各类数据找到摆放它们的场所，在计算机中，这些场所就是各种类型的存储器

PA 1-1 数据的类型和存取

	计算机	餐厅	
解决计算问题的步骤	程序	菜谱	做菜的步骤
程序处理的对象	数据	食材	做菜加工的对象
执行程序的器件	CPU	大厨	执行菜谱的人
储存正在处理的数据	CPU内部的寄存器	灶台上的锅	放置正在加工的食材
储存马上要用的数据	主存	厨房里的冰箱	储存马上用的菜谱和食材
存储大量的数据	硬盘	仓库	啥都放这里

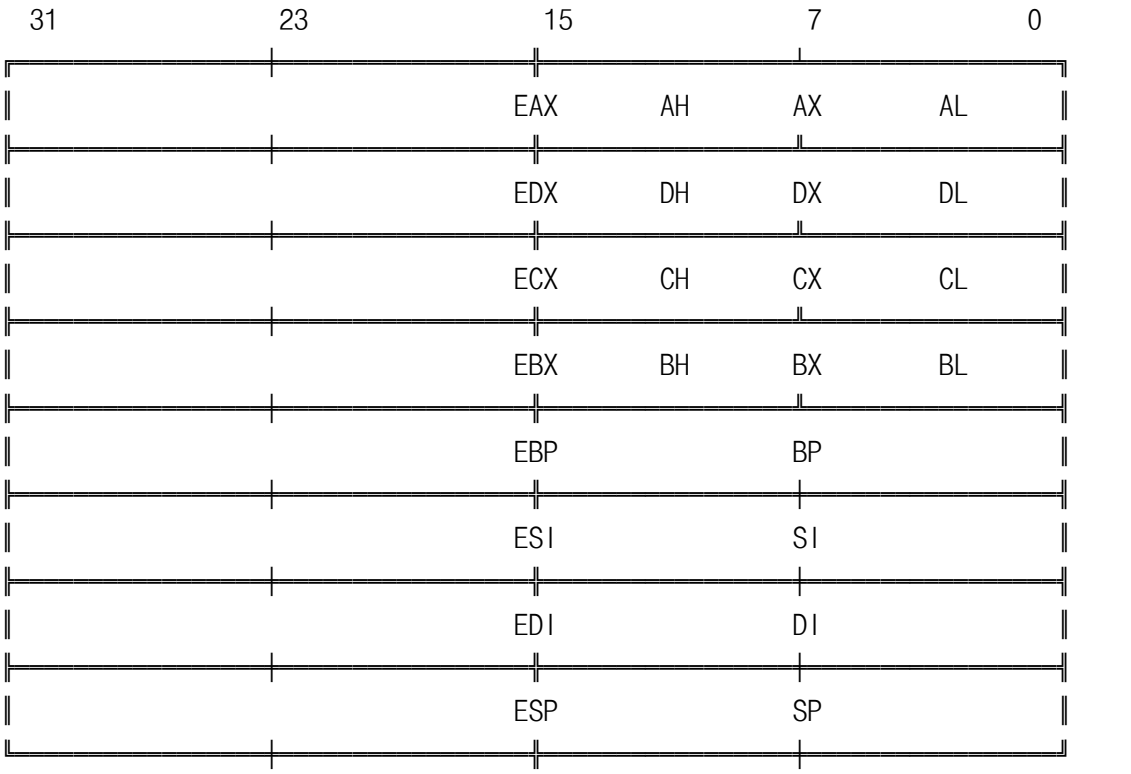
PA 1-1 数据的类型和存取



存储器的层次结构

PA 1-1 数据的类型和存取

- NEMU模拟的是i386体系结构
  - 8个32位通用寄存器



用C语言怎么模拟？

PA 1-1 数据的类型和存取

- NEMU模拟的是i386体系结构
  - 框架代码中nemu/include/cpu/reg.h

```
// define the structure of registers
typedef struct {

        // general purpose registers
        struct {
                struct {
                        struct {
                                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;
                };
        };

        // EFLAGS ···
} CPU_STATE;
```

要实现和前面图中一样的结构

PA 1-1 数据的类型和存取

- NEMU模拟的是i386体系结构
  - 框架代码中nemu/include/cpu/reg.h

```
// define the structure of registers
typedef struct {

        // general purpose registers
        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;
                };
        };

        // EFLAGS ···
} CPU_STATE;
```

教程中直接给了答案

PA 1-1 数据的类型和存取

在教程§1-1.3中提出的实验过程及要求：

1. 修改 CPU_STATUS 结构体中的通用寄存器结构体;
2. 使用 make 编译项目;
3. 在项目根目录通过./nemu/nemu 命令执行 nemu 并通过 reg_test()测试用例。

~PA 1-1顺利完成~

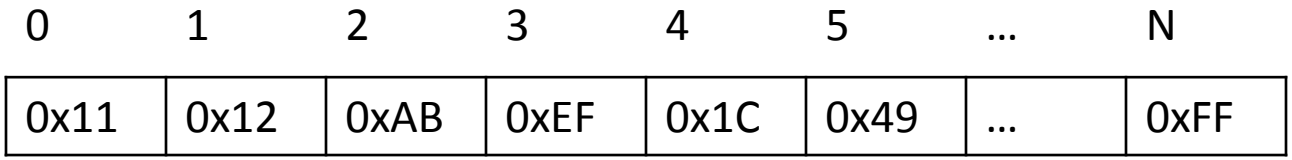
```
======== reg test ========
reg_test()      pass
```

在实验报告中,简要叙述:

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?

PA 1-1 数据的类型和存取

- 除了寄存器之外，还要熟悉主存的模拟方式及其接口
  - 以字节为基本编址单位
  - 通过地址直接访问某个字节



  - 怎么来模拟？数组！nemu/src/memory/memory.c中hw_mem[]
  - 模拟内存多大？nemu/include/memory/memory.h中MEM_SIZE_B
  - 模拟内的对外提供的读写接口？memory.h声明memory.c中实现

~要熟悉这些接口~

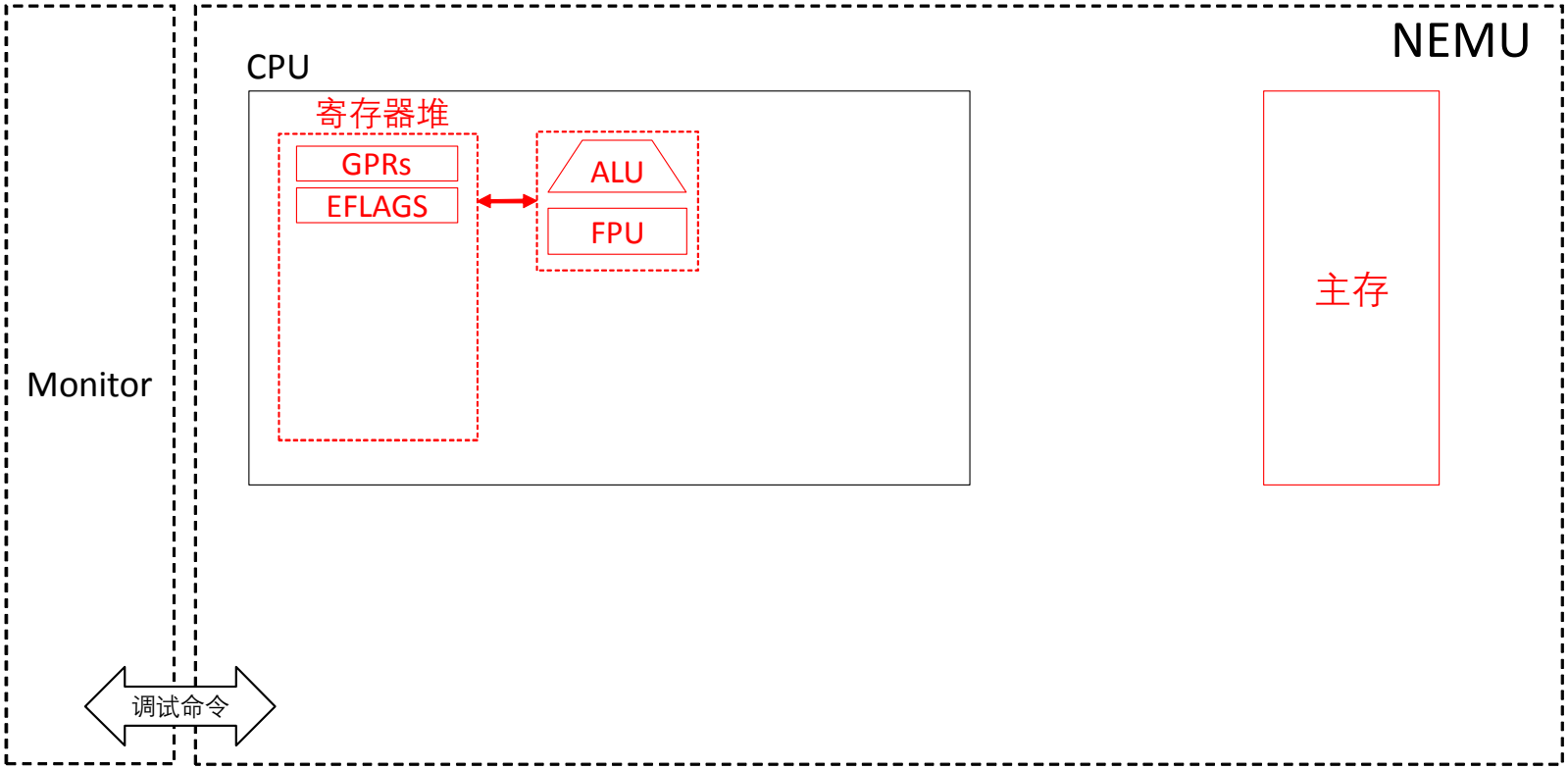
PA 1-2 整数的表示和运算

PA 1-2 整数的表示和运算

大厨灵巧的双手，拥有处理各类食材的扎实基本功。NEMU强大的CPU，则拥有处理各类数据（整数、浮点数）实现各种运算的功能。

我们要赋予计算机的CPU进行各种基本数值计算的能力，才好让它执行更为复杂的计算过程

PA 1 – 路线图



PA 1-2 整数的表示和运算

- 整数
  - 无符号整数
    - 32位整数：0x0 ~ 0xFFFFFFFF（32个1）
  - 带符号整数（归于某种无符号整数的表示方法）
    - 原码表示法：最高位为符号位
    - 补码表示法（普遍采用）
      - 各位取反末位加一
      - 用加法来实现减法
      - 可以试试X+(-X)等于多少，其中X为某一32位正整数，-X为其补码表示，运算结果截取低32位

PA 1-2 整数的表示和运算

- 在NEMU中模拟各类整数运算的部件
  - ALU – 算术逻辑单元
    - 能进行各类算术运算：加减乘除、移位
    - 能进行各种逻辑运算：与或非
    - 对应代码：nemu/src/cpu/alu.c
  - 得到运算结果的同时设置标志位
    - EFLAGS
      - 指示运算结果的标志位：CF, PF, AF, ZF, SF, OF, …
      - 指示机器状态的标志位：IF, …
      - 对应代码：nemu/include/cpu/reg.h

炒整数之菜

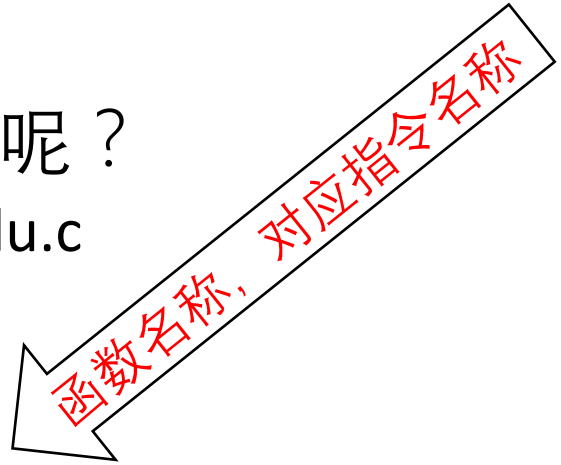
不模拟AF

目前只关心这个

油温几成热？菜
炒糊了没有？

PA 1-2 整数的表示和运算

- 怎么来完成模拟呢？
  - nemu/src/cpu/alu.c




```
uint32_t alu_add(uint32_t src, uint32_t dest) {
        printf("\e[0;31mPlease implement me at alu.c\e[0m\n");
        assert(0);
        return 0;
}
```

要替换成教程中说明的正确实现：返回dest + src的结果，并设置标志位

第一步：找到需要实现的函数
执行./nemu/nemu遇到错误提示

PA 1-2 整数的表示和运算

- 怎么来完成模拟呢？
  - nemu/src/cpu/alu.c
  - 找到i386手册Sec. 17.2.2.11
    - 有关ADD指令的描述 p.g. 261 of 421
    - 看Flags Affected: OF, SF, ZF, AF, CF, and PF as described in Appendix C
    - 找到Appendix C并仔细体会（AF不模拟）

第二步：掏出i386手册

PA 1-2 整数的表示和运算

- 怎么来完成模拟呢？
  - nemu/src/cpu/alu.c

```
uint32_t alu_add(uint32_t src, uint32_t dest) {
        uint32_t res = 0;
        res = dest + src;

        set_CF_add(res, src);
        set_PF(res);
        // set_AF();
        set_ZF(res);
        set_SF(res);
        set_OF_add(res, src, dest);

        return res;
}
```

cpu是个全局变量：nemu/src/cpu/cpu.c

```
void set_CF_add(uint32_t result, uint32_t src) {
        cpu.eflags.CF = result < src;
}

void set_PF(uint32_t result) {
        // set according to the low-order 8-bits of the result
}

void set_ZF(uint32_t result) {
        cpu.eflags.ZF = (result == 0);
}

void set_SF(uint32_t result) {
        cpu.eflags.SF = sign(result);
}

void set_OF_add(uint32_t result, uint32_t src, uint32_t dest) {
        if(sign(src) == sign(dest)) {
                if(sign(src) != sign(result))
                        cpu.eflags.OF = 1;
                else
                        cpu.eflags.OF = 0;
        } else {
                cpu.eflags.OF = 0;
        }
}
```

第三步：按照手册规定的操作进行实现

PA 1-2 整数的表示和运算

- 怎么来完成模拟呢？
  - nemu/src/cpu/alu.c
  - 实现完，保存alu.c
  - 执行make编译
  - 执行./nemu/nemu
  - 测试用例位于nemu/src/cpu/test/alu_test.c

```
======== alu test ========
alu_test_add()  pass
```

第四步：执行测试用例

PA 1-2 整数的表示和运算

~PA 1-2顺利完成~

- 实验的要求
  - 把教程中提到的所有运算函数都实现，通过测试用例
  - 参见教程§1-2.3 实验过程及要求

```
======== alu test ========
alu_test_add()  pass
alu_test_adc()  pass
alu_test_sub()  pass
alu_test_sbb()  pass
alu_test_and()  pass
alu_test_or()   pass
alu_test_xor()  pass
alu_test_shl()  pass
alu_test_shr()  pass
alu_test_sal()  pass
alu_test_sar()  pass
alu_test_mul()  pass
alu_test_div()  pass
alu_test_imul() pass
alu_test_idiv() pass
```

注意：移位操作不测试OF位
imul操作所有标志位都不测试

注意：禁止采用测试用例里面使用内联汇编进行实现的方法，但是可以学习这种交叉验证的思想

第四步：执行测试用例

PA 1-2 整数的表示和运算

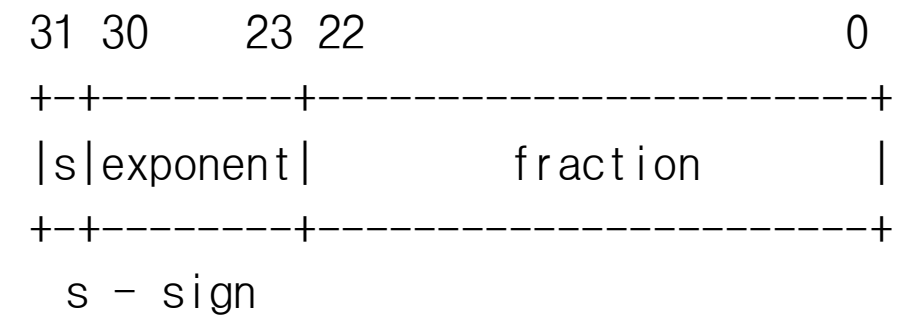
- 实现ALU的目的
  - 复习课本第二章内容，准备迎接小测验
  - 在alu.c中实现的这些函数，到了PA 2实现对应指令的时候，就可以直接调用了

PA 1-3 浮点数的表示和运算

PA 1-3 浮点数的表示和运算

- IEEE 754标准

Single-precision Floating Point



32位单精度浮点数的表示方法

0 表示正数 +
1 表示负数 -

假设是规格化数

$= s\ 1.fraction * 2^{(exponent - 127)}$

阶码exponent –偏置常数127构成指数部分，注意非规格化数exponent为0，但指数为-126

尾数fraction部分加上隐藏位构成significand（非规格化数隐藏位是0）

PA 1-3 浮点数的表示和运算

- IEEE 754标准

For his fundamental contributions to numerical analysis. One of the foremost experts on floating-point computations. Kahan has dedicated himself to "making the world safe for numerical computations"!

William Kahan （1933 - ?）
ACM Turing Award, 1989

对于IEEE 754标准在数值计算方面的深层次讨论已经超出了我目前的能力范围，在此只能对其表面上所展现的运算规则进行探索。在大师面前我永远只是个小学生，加油！

PA 1-3 浮点数的表示和运算

- IEEE 754标准
  - nemu/include/cpu/reg_fpu.h中的FLOAT结构体
  - 对于各种类型浮点数的解释，课本p.g. 45表2.2
    - 零
    - 无穷大
    - NaN
    - 规格化数
    - 非规格化数

PA 1-3 浮点数的表示和运算

- NEMU中模拟浮点数的算术运算的部件
  - FPU - 浮点运算单元
  - 实现浮点数运算：加减乘除
  - 相关代码：nemu/src/cpu/fpu.c
  - 需要实现internal_float_xxx()函数

炒浮点数之菜

- 实现的基本步骤，配合课本p.g. 69，§2.7.7的内容
  - 处理各类边界条件：零、NaN、无穷大
  - 提取符号、阶码和尾数
  - 以无符号整数运算分别得到结果的符号、阶码、尾数
    - 加减法：对阶 -> 尾数加减
    - 乘除法：阶码加减 -> 尾数乘除
  - 尾数规格化和舍入

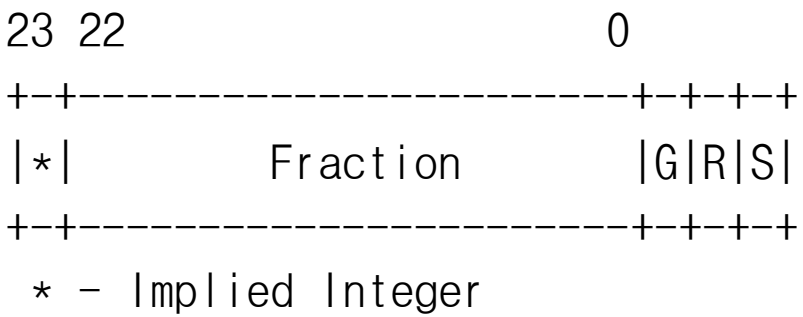
禁止采用利用FLOAT将uint32_t表示的浮点数转换成float再进行运算得到结果（及其类似）的实现方案，比如fpu.c中被注释掉的部分

PA 1-3 浮点数的表示和运算

- 浮点数的加减法（以internal_float_add()为例）
  - nemu/src/cpu/fpu.c
  - 第一步：处理各类边界条件：零、NaN、无穷大
    - 框架代码已经做好了
  - 第二步：提取符号、阶码和尾数
    - 框架代码也做好了，关键是尾数是否要加上隐藏位

PA 1-3 浮点数的表示和运算

- 浮点数的加减法（以internal_float_add()为例）
  - nemu/src/cpu/fpu.c
  - 第三步：对阶
    - 小阶向大阶看齐，中间运算结果的阶码是大的那个
    - 阶小的那个数的尾数向右移shift位
      - shift = 大的阶码 – 小的阶码。计算时有些细节问题？看教程
      - 将尾数进行右移，这时候需要谈到保护位(guard, G)、舍入位(round, R)、和粘位(sticky, S)
      - 右移的时候尾数的低位进入GRS bits，注意sticky位的处理规则



框架代码中的实现技巧：将运算过程中间结果的尾数（不计隐藏位）扩展为26位，即，在提取尾数后，将尾数左移3位，最低3位表示GRS bits

PA 1-3 浮点数的表示和运算

- 浮点数的加减法（以internal_float_add()为例）
  - nemu/src/cpu/fpu.c
  - 第四步：以无符号数加法计算尾数中间结果
    - 假设两个参与运算的尾数已经扩展了GRS bits
    - 根据浮点数符号位得到尾数的补码表示
    - 进行无符号数加法得到尾数的中间结果，根据尾数中间结果的符号确定结果浮点数的符号，并将尾数转回原码表示
      - GRS bits自然参与运算
      - 得到的中间结果仍假设为具有26位尾数（不计隐藏位）

```
23 22                      0
+-+----------------------+-+-+-+
|*|       Fraction       |G|R|S|
+-+----------------------+-+-+-+
 * - Implied Integer
```

若中间结果隐藏位后面恰好为26位，自然天下太平。但是！若两个数的阶码恰好相等，这时候尾数加法的结果可能是：

```
   1xxxxx  (26个x)
+  1yyyyy  (26个y)
= 10zzzzz  (26个z)
```

隐藏位后面有27位了，怎么办？

PA 1-3 浮点数的表示和运算

- 浮点数的加减法（以internal_float_add()为例）
  - nemu/src/cpu/fpu.c
  - 第五步：尾数规格化

若中间结果隐藏位后面恰好为26位，自然天下太平。但是！若两个数的阶码恰好相等，这时候尾数加法的结果可能是：

```
   1xxxxx   （26个x）
+  1yyyyy   （26个y）
= 10zzzzz   （26个z）
```

隐藏位后面有27位了，怎么办？

中间结果符号

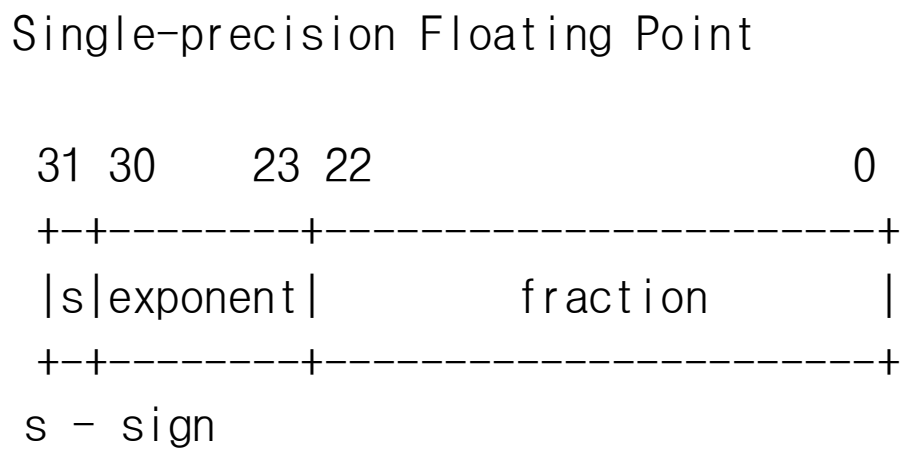
中间结果阶码，带符号数以方便后续乘除法的处理

- 将尾数右移1位，同时阶码加1
- uint32_t internal_normalize(uint32_t sign, int32_t exp, uint64_t sig_grs)

中间结果尾数（低3位是GRS bits）约定尾数不计隐藏位的长度为26位（传入时可能不满足），64位以方便后续乘除法的处理

PA 1-3 浮点数的表示和运算

- 浮点数的加减法（以internal_float_add()为例）
  - nemu/src/cpu/fpu.c
  - 第五步：尾数规格化
    - uint32_t internal_normalize(uint32_t sign, int32_t exp, uint64_t sig_grs)
    - 函数功能：尾数规格化，顺便把舍入一起做了，返回一个符合IEEE 754标准的浮点数（放在一个32位无符号整型数中）



PA 1-3 浮点数的表示和运算

- 浮点数的加减法（以internal_float_add()为例）
  - nemu/src/cpu/fpu.c
  - 第五步：尾数规格化
    - uint32_t internal_normalize(uint32_t sign, int32_t exp, uint64_t sig_grs)
    - 分情形讨论
      - 符号位不去管它，规格化和舍入都不会改变符号
      - 首先，对于加减法，中间结果的exp必然>=0
      - 第二，根据sig_grs的情况，要对其进行必要的左规和右规操作
        - Case 1: exp > 0，且，sig_grs隐藏位后面超过了26位
          - 条件：sig_grs >> (23 + 3) > 1 且 exp > 0
          - 操作：将尾数右移1位，exp++，直至sig_grs >> (23 + 3) == 1
            - 注意sticky bit的操作
          - 例外：exp加过了头（>= 0xFF了），阶码上溢
        - Case 2: exp > 0，且，sig_grs隐藏位后面不足26位（例如：1.xxx + (–1.0)）
          - 条件： sig_grs >> (23 + 3) == 0 且 exp > 0
          - 操作：尾数左移1位，exp--，直至sig_grs >> (23 + 3) == 1
          - 例外：exp减过了头（==0了），得到了非规格化数
            - 注意为了配合非规格化数的阶码为0表示2^{-126}，需要额外将尾数右移一次，注意sticky bit的操作
        - Case 3：exp == 0，且， sig_grs >> (23 + 3) == 1
          - 额外将尾数右移一次，注意sticky bit的操作
        - 其它情形：无需进行规格化（有哪些情形？）

理解教程中的伪代码

PA 1-3 浮点数的表示和运算

- 浮点数的加减法（以internal_float_add()为例）
  - nemu/src/cpu/fpu.c
  - 第六步：舍入
    - uint32_t internal_normalize(uint32_t sign, int32_t exp, uint64_t sig_grs)
    - 如果没有产生溢出，根据GRS bits的取值情况进行舍入
      - 就近舍入到偶数
      - 舍入若产生尾数加1，有可能出现破坏规格化的情况
        - 此时需要进行额外的一次右规并判断阶码上溢的情况
  - 第七步：得到返回结果
    - 简单问题，框架代码已经给出

实现完浮点数加法浮点数减法自然同时实现

PA 1-3 浮点数的表示和运算

- 浮点数的乘除法
  - nemu/src/cpu/fpu.c
  - 第一步：处理各类边界条件：零、NaN、无穷大
  - 第二步：提取符号、阶码和尾数（尾数采用64位表示比较方便）
  - 第三步：以无符号数加法计算尾数中间结果
    - 符号位的处理：简单
    - 阶码的处理：
      - 乘法：阶码相加，扣除多加的偏置常数
      - 除法：阶码相减，加上多减掉的偏置常数（可能减出负数来，所以exp中间结果用带符号数比较方便）
    - 尾数的处理（与加减法不同，先不预留出给GRS bits的3位）
      - 乘法：尾数做无符号整数乘法，中间结果不计隐藏位有46位尾数
      - 除法：被除数左移23位，除以除数，中间结果不计隐藏位有23位尾数
      - 我们约定包含GRS bits的尾数中间结果不计隐藏位是26位，因此需要将尾数因整数乘除法运算而多出来的尾数归到阶码上去

PA 1-3 浮点数的表示和运算

- 浮点数的乘除法
  - nemu/src/cpu/fpu.c
  - 第三步：以无符号数加法计算尾数中间结果
    - 尾数的处理（与加减法不同，先不预留出给GRS bits的3位）
      - 乘法：尾数做无符号整数乘法，中间结果不计隐藏位有46位尾数
      - 除法：被除数左移23位，除以除数，中间结果不计隐藏位有23位尾数
      - 我们约定包含GRS bits的尾数中间结果不计隐藏位是26位，因此需要将尾数因整数乘除法运算而多出来的尾数位数归到阶码上去
        - 乘法：阶码额外减去 46 – 26 = 20位
        - 除法：如果按照上述处理，则无需调整。但是，为了保证精度，框架代码做了额外的移位处理（左移加右移共移了shift位），因此，阶码额外减去shift – 26位

1xxxx * 1yyyy = 1zzzz

23个x，23个y，46个z

左移至高位无0

⟸

00001xxxx / 1yyy000

⟹

右移至低位无0

PA 1-3 浮点数的表示和运算

- 浮点数的乘除法
  - nemu/src/cpu/fpu.c
  - 第四步：尾数规格化，在加减法基础上的额外情形
    - uint32_t internal_normalize(uint32_t sign, int32_t exp, uint64_t sig_grs)
    - 额外情形：exp < 0
      - 操作：和sig_grs >> (23 + 3) > 1的情形一样，需要右规，直至
        - 得到非规格化数：exp == 0 且 sig_grs >> (23 + 3) <= 1 且 sig_grs > 0 （舍入之后仍大于0）
          - 在while循环外多右移一次配合非规格化数阶码的约定
        - 或，得到规格化数：exp > 0 且 sig_grs >> (23 + 3) == 1
      - 例外：
        - 已经无法右规了sig_grs <= 4 （舍入后就是0了），exp仍然小于0，产生阶码下溢
        - 其它情况已在做加减法时列举了
  - 第五步：舍入
  - 第六步：得到返回结果

PA 1-3 浮点数的表示和运算

§1-3.3 实验过程及要求

1.实现nemu/src/cpu/fpu.c中的各个整数运算函数；

2.将internal_normalize()函数补完；

3.使用make命令编译项目；

4.使用./nemu/nemu命令执行NEMU并通过各个浮点数运算测试用例。

在实验报告中，回答以下问题：

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。

请通过make submit命令打包代码并上传cms，同时打包实验报告和调查问卷，谢谢！

~PA 1-3顺利完成~

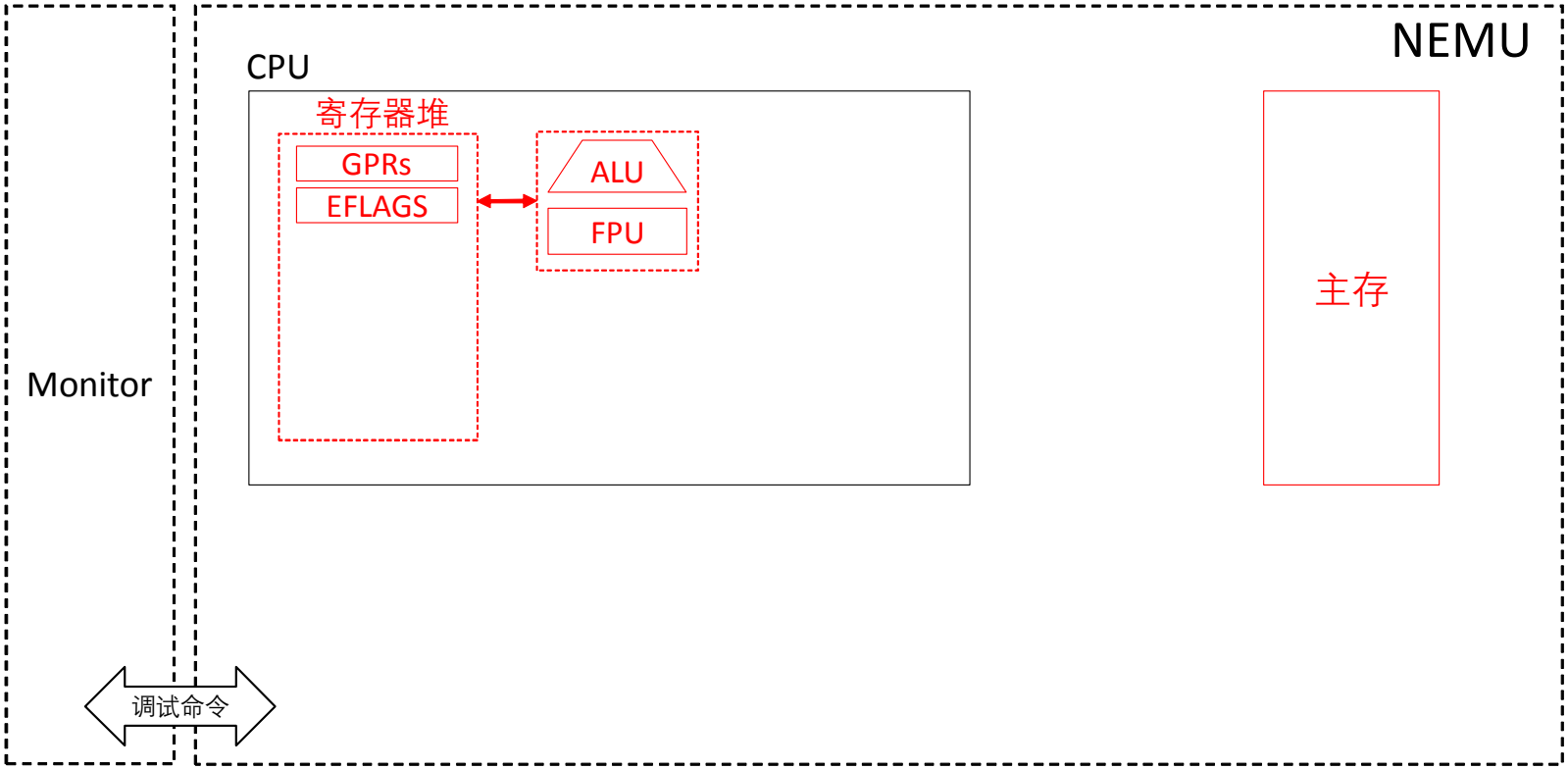
```
======== fpu test ========
fpu_test_add()  pass
fpu_test_sub()  pass
fpu_test_mul()  pass
fpu_test_div()  pass
```

测试用例位于nemu/src/cpu/test/fpu_test.c

PA 1-3 浮点数的表示和运算

- 实现FPU的目的
  - 巩固理论知识，备战小测验
  - 在fpu.c中实现的这些internal函数，用于实现fpu.c中的浮点数运算功能，进而实现x87浮点数指令，框架代码已经给出了这些指令的实现，位于nemu/src/cpu/instr/x87.c
- 有关x87指令的详细说明，i386手册上没有
  - 在古代对于没有FPU的CPU，是采用软件浮点数（直接编译出基于整数运算的浮点数处理过程）来实现的
  - 在某些嵌入式系统上，使用binary scaling这样的定点小数表示法
  - x87指令参见：http://www.felixcloutier.com/x86/

PA 1 数据的表示、存取和运算



PA 1 数据的表示、存取和运算

新东方初级班毕业

我了解到了厨房里面有锅和冰箱。同时学会了整数和浮点数两种食材的处理方法！

PA 1 数据的表示、存取和运算

- PA 1不设置小阶段
- 整个PA 1的提交截止时间
  - 2017年9月20日（周三）24点

PA 1到此结束

祝大家学习快乐，身心健康！

欢迎大家踊跃参加问卷调查

（量表一和二今天就要做一次，量表一、二、三到PA 1截止时再做一次）