计算机系统基础
Programming Assignment

PA 1 数据的表示、存取和运算

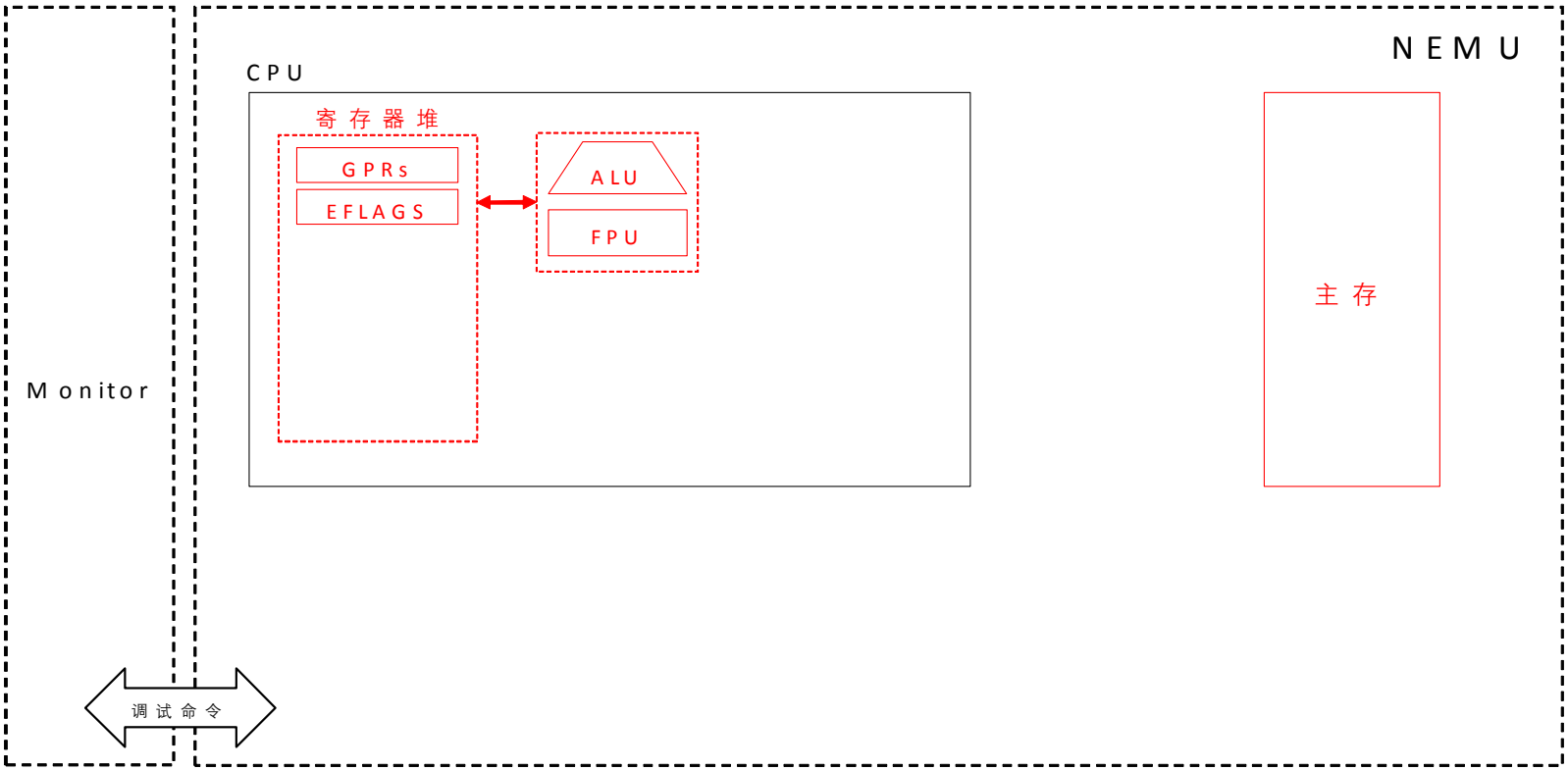
2019年9月20日

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

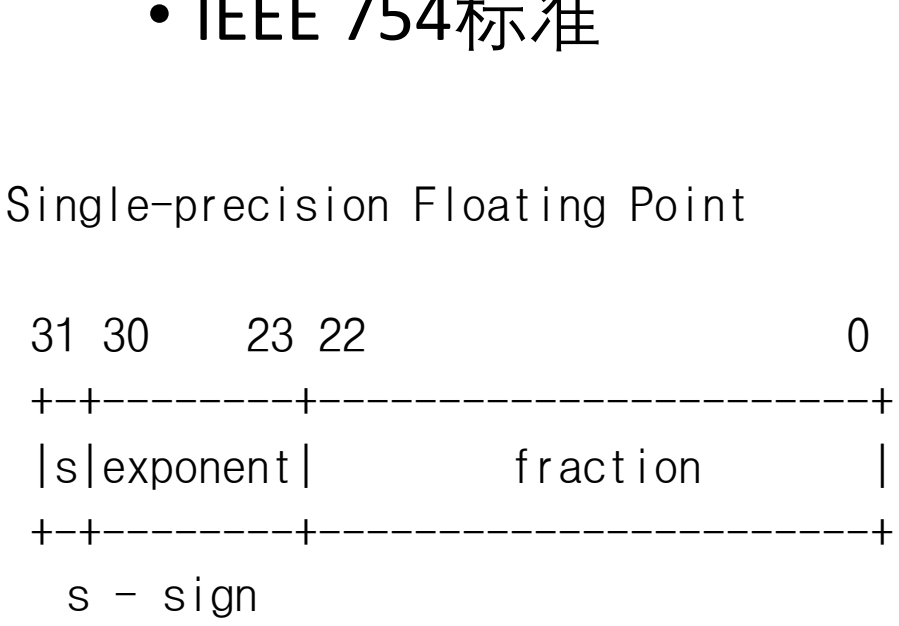
PA 1-3 浮点数的表示和运算

PA 1 – 路线图



PA 1-3 浮点数的表示和运算

- IEEE 754标准



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. 46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
- 实现的基本步骤，配合课本p.g. 72，§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

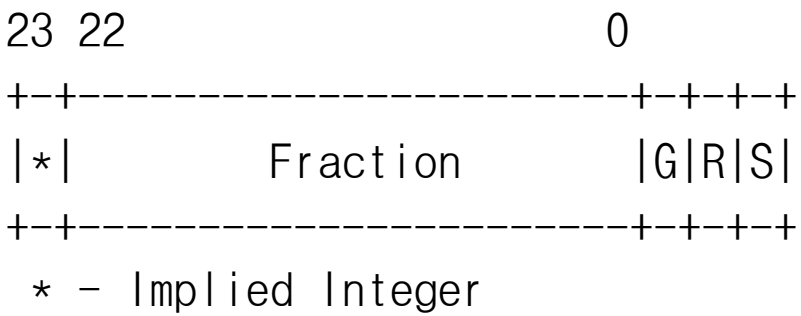
禁止采用利用FLOAT将uint32_t表示的浮点数转换成float再进行运算得到结果（及其类似）的实现方案，比如fpu.c中被注释掉的部分

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？

- 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)

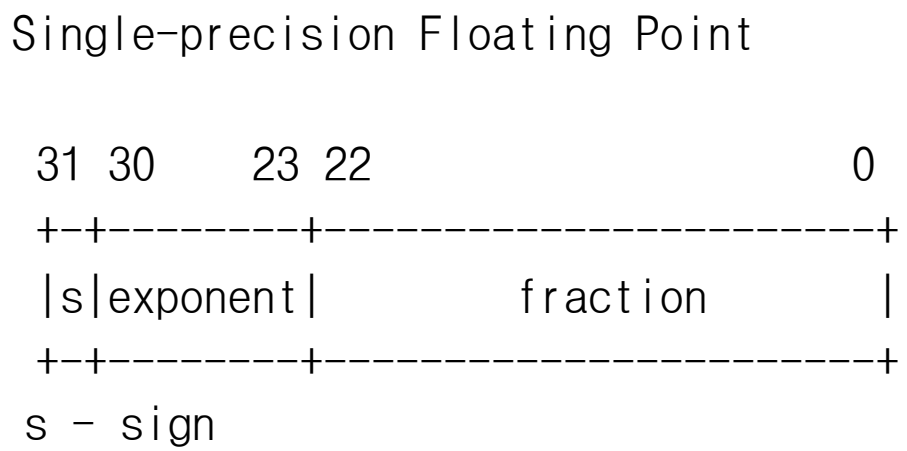
中间结果符号

中间结果阶码，带符号数以方便后续乘除法的处理

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
          - **需要将exp++，保证阶码真值为-126**
        - 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：以无符号数运算获取中间结果的解码和尾数
    - 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 --test-fpu xxx或make test_pa-1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_pa-1命令打包代码并上传，如出现问题，根据提示补交到cms备用窗口，谢谢！

~PA 1-3顺利完成~

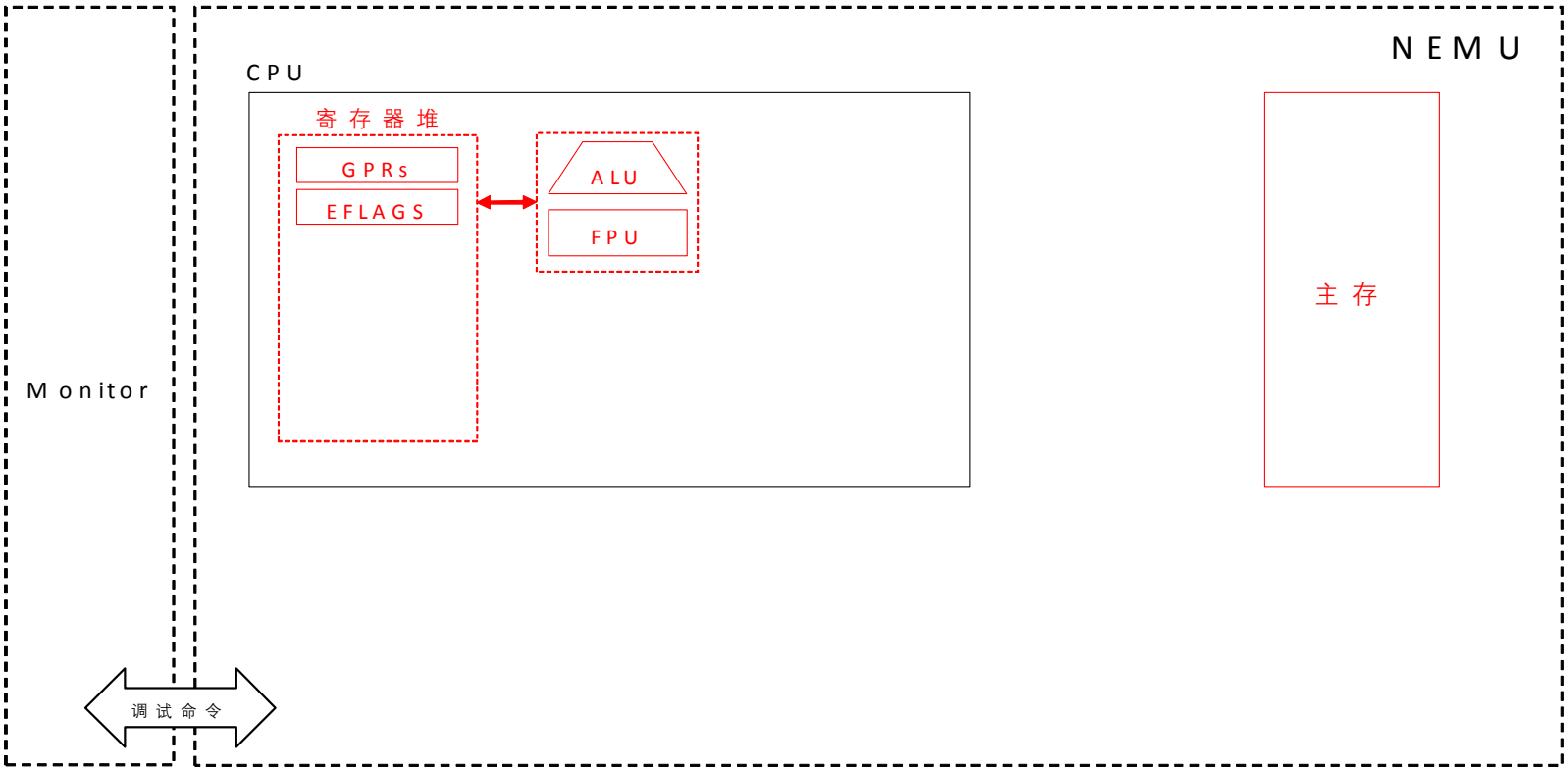
```
pu_test_add()  pass
fpu_test_sub()  pass
fpu_test_mul()  pass
fpu_test_div()  pass
```

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
  - 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

- PA 1不设置小阶段
- 整个PA 1（包括实验报告）的提交截止时间
  - 2019年9月28日（周六）24点
- 提交方式
  - make submit_pa-1
  - 如果出错，按照提示进行
  - 无论自动提交是否成功，都将make submit的压缩包上传到cms系统的备用提交窗口
  - 实验报告上传到cms系统的报告提交窗口

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