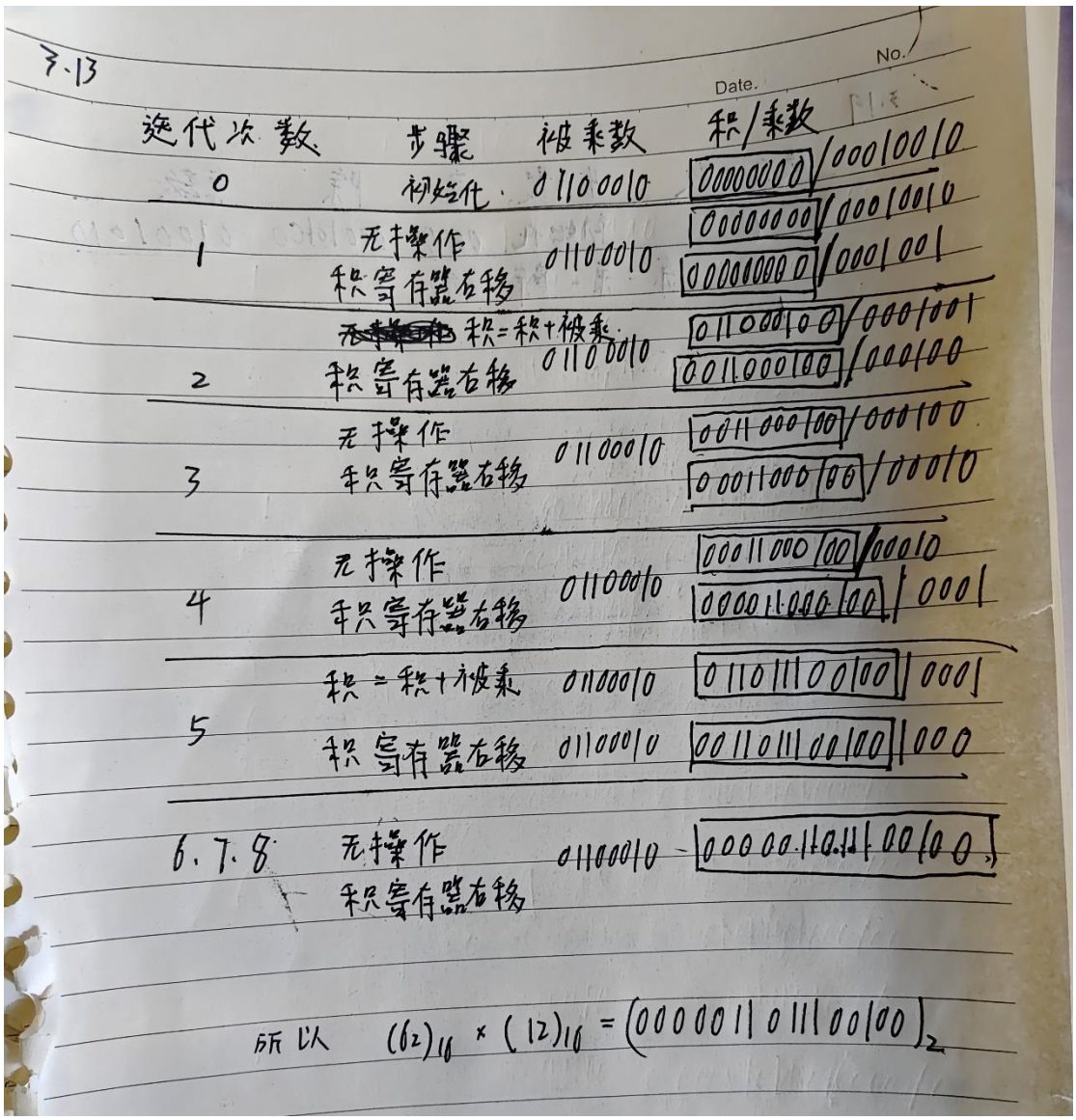
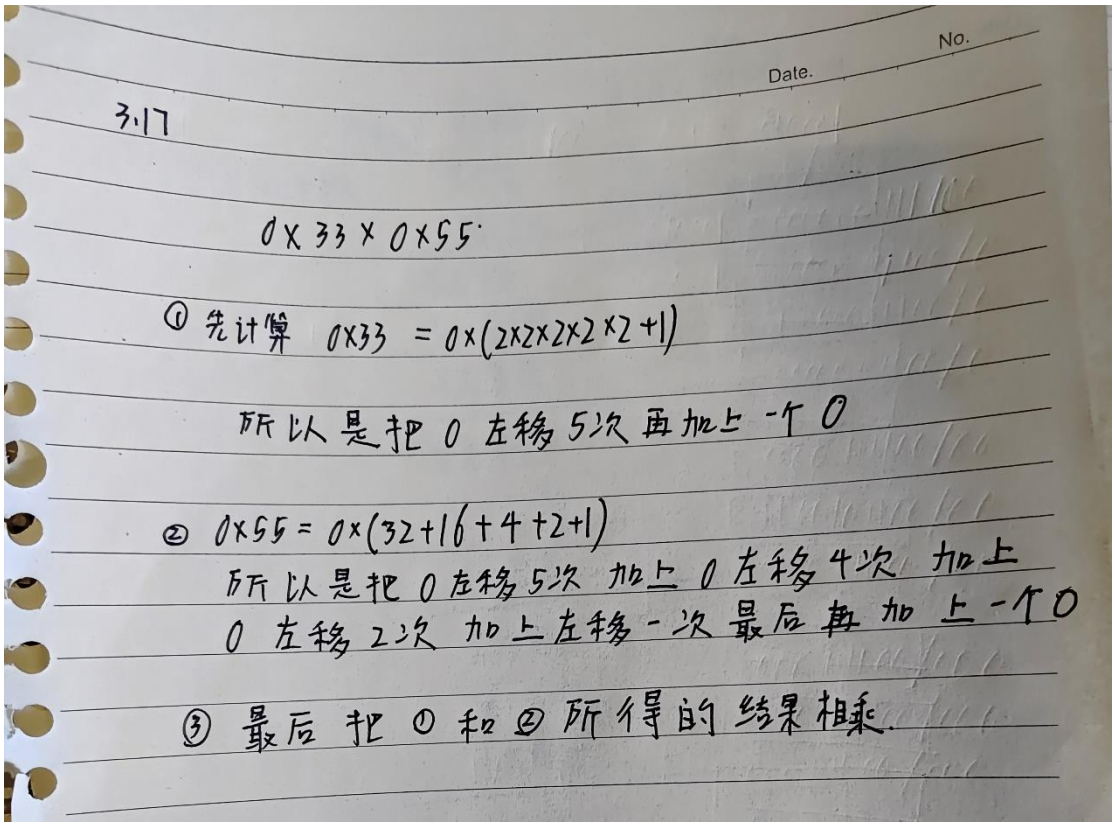
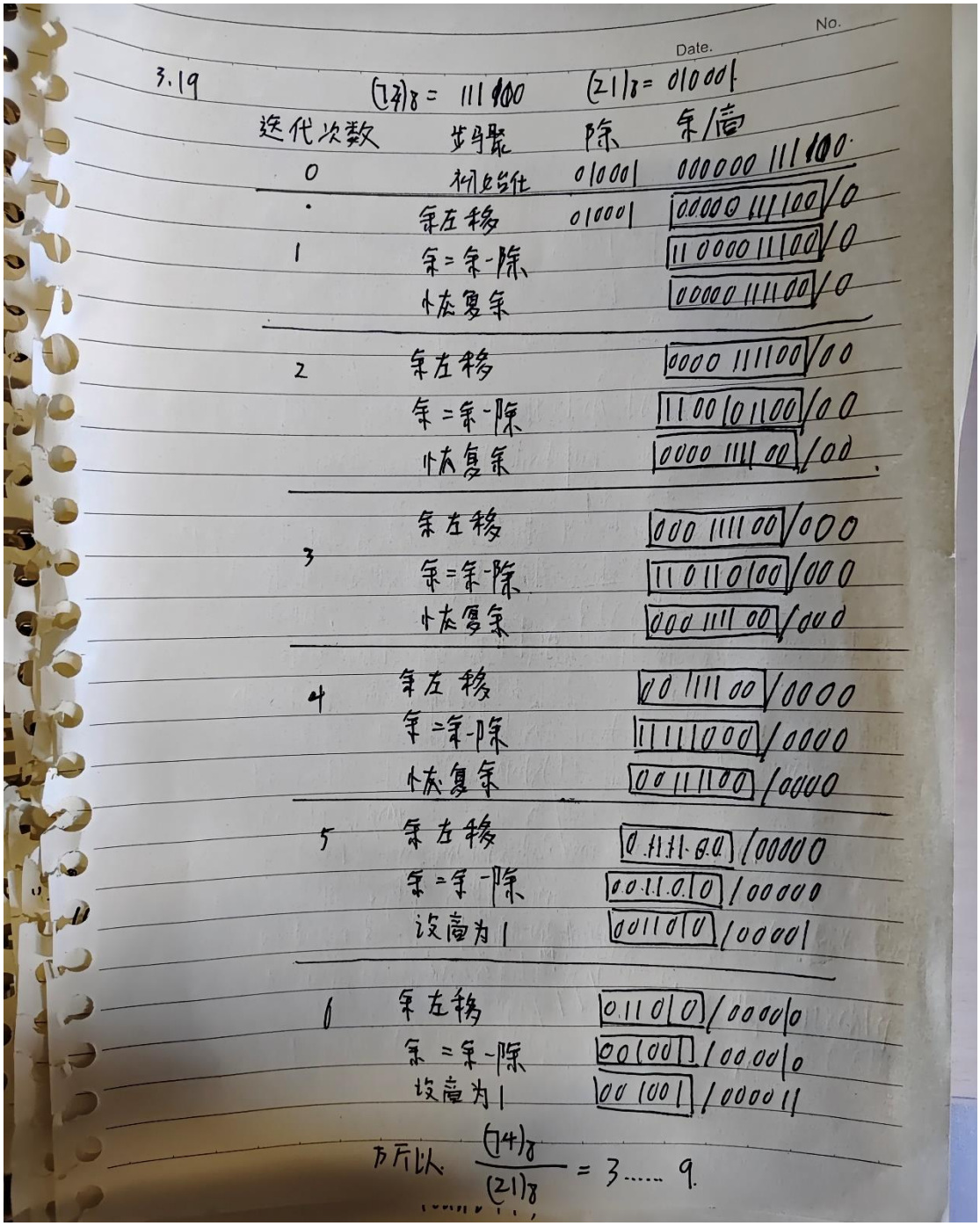
3. 13


3.13
No.
Date.
迭代次数 步骤 被乘数 积/乘数
0 初始化 01100010 00000000 0001 0010
1 无操作 01100010 00000000 0001 0010
积寄存器右移 00000000 0 0001 001
2 积=积+被乘 01100010 01100010 0 0001 001
积寄存器右移 00110001 00 0001 00
3 无操作 01100010 00110001 00 0001 00
积寄存器右移 00011000 100 0001 0
4 无操作 01100010 00011000 100 0001 0
积寄存器右移 00001100 0100 0001
5 积=积+被乘 01100010 01101110 0100 0001
积寄存器右移 00110111 00100 0000
6.7.8 无操作 01100010 00000110 1100 1000
积寄存器右移
所以 (62)10 × (12)10 = (0000 0111 0010 0000)2

3.17


Date.
No.
3.17
0x33x0x55.
① 先计算 0x33 = 0x(2x2x2x2x2+1)
所以是把 0 左移 5次 再加上一个 0
② 0x55 = 0x(32+16+4+2+1)
所以是把 0 左移 5次 加上 0 左移 4次 加上
0 左移 2次 加上左移一次 最后再加上一个0
③ 最后把 ① 和 ② 所得的结果相乘.

3.19



3.19

$(74)_8 = 111100$　　$(21)_8 = 010001$

迭代次数	步骤	除	余/商
0	初始条件	010001	000000 111100
1	余左移	010001	000001 111000
	余=余-除		110000 111000
	恢复余		000001 111000
2	余左移		000011 110000
	余=余-除		111001 110000
	恢复余		000011 110000
3	余左移		000111 100000
	余=余-除		111010 100000
	恢复余		000111 100000
4	余左移		001111 000000
	余=余-除		111110 000000
	恢复余		001111 000000
5	余左移		011110 000000
	余=余-除		001101 000000
	设商为1		001101 000001
1	余左移		011010 000010
	余=余-除		001001 000010
	设商为1		001001 000011

所以 $\frac{(74)_8}{(21)_8} = 3 \cdots\cdots 9$

3.28

3.28

先把 -1.5625×10^{-1} 转换成二进制

$\because 1.5625\times10^{-1}=0.5625=(0.0010)_2$

所以 $-1.5625\times10^{-1}=-(0.0010)_2=(-0.10\times2^{-2})_2$

所以尾数为 $-.10$　　指数为 -2

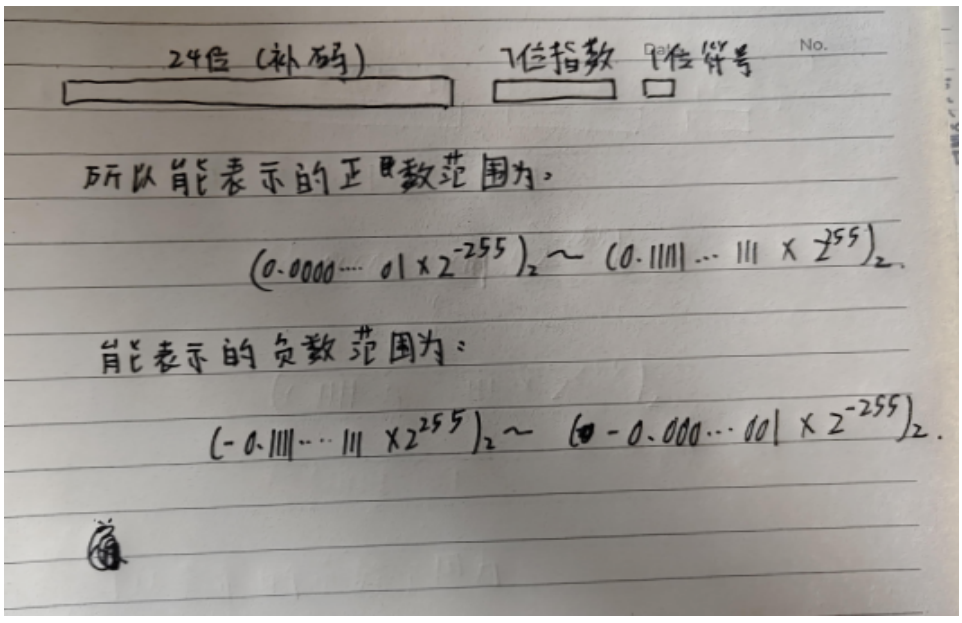
尾数转换为补码的话是：1011 0000 0000 …… 0000

（1位符位　数值位）

因为指数是 -2，所以最右边是1，前面七位是 0000010

二进制的表示为：

1011 0000 0000 0000 0000 0000 0000 0101



所以能表示的正数范围为：

$(0.0000\cdots01\times2^{-255})_2\sim(0.1111\cdots111\times2^{255})_2$

能表示的负数范围为：

$(-0.1111\cdots111\times2^{255})_2\sim(-0.000\cdots001\times2^{-255})_2$

3.39

Date. No.

3.39

先计算 $1.166015625\times10^{0}\times1.9760\times10^{4}$

因为 $(1.166015625)_{10}=1.1010101010\times2^{0}$

$(1.9760\times10^{4})_{10}=1.0011010011\times2^{14}$

$1.1010101010\times2^{0}\times1.0011010011\times2^{14}$

$$
\begin{array}{l}
1.1010101010 \\
\times\ 1.0011010011 \\
\hline
1101010101 0 \\
0000000000 \\
0000000000 \\
1101010101 0 \\
0000000000 \\
1101010101 0 \\
1101010101 0 \\
0000000000 \\
0101010101 0 \\
\hline
10.0000001001100001111
\end{array}
$$

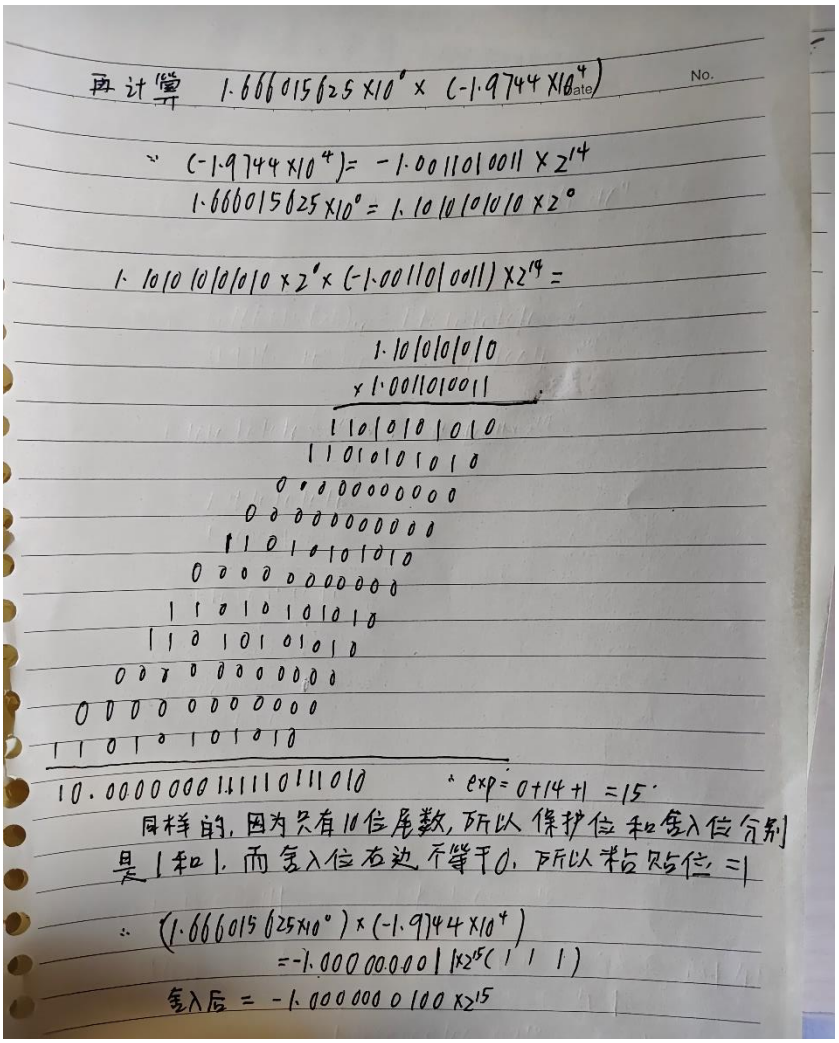
$\therefore exp=1+0+14=15$

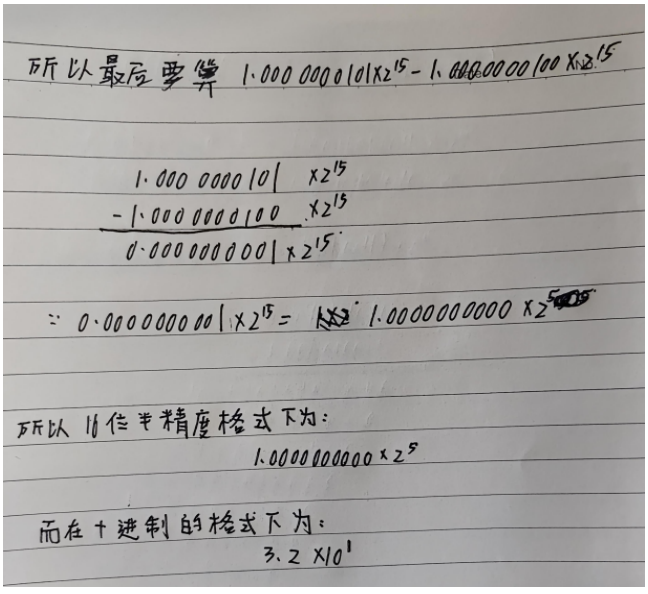
因为尾数只有10位，所以保护位和舍入位分别为1和1

且为舍入位右边不等于0，所以粘贴位为1.

最后运算为 1.0000000100×2^{15} (111)

舍入后 $=1.0000000101\times2^{15}$


再计算 1.6660156025×10^0 × (-1.9744×10^4)
No.
∵ (-1.9744×10^4) = -1.0011010011 × 2^14
1.6660156025×10^0 = 1.1010101010 × 2^0
1.1010101010 × 2^0 × (-1.0011010011) × 2^14 =
1.1010101010
× 1.0011010011
1101010101 0
11010101010
0.0000000000
00000000000
110101010100
0000000000000
1101010101000
0000000000000
0000000000000
1101010101000
10.0000000111111011010
* exp = 0+14+1 = 15.
同样的，因为只有10位尾数，所以保护位和舍入位分别是1和1，而舍入位右边不等于0，所以粘贴位=1
∴ (1.6660156025×10^0) × (-1.9744×10^4)
= -1.0000000011 × 2^15 (1 1 1)
舍入后 = -1.0000000100 × 2^15


所以最后要算 1.0000000101×2^15 - 1.0000000100×2^15
1.0000000101 × 2^15
- 1.0000000100 × 2^15
0.0000000001 × 2^15
∵ 0.0000000001 × 2^15 = 1.0000000000 × 2^5
所以16位半精度格式下为：
1.0000000000 × 2^5
而在十进制的格式下为：
3.2 × 10^1

3.47

3.47.

因为最大有128位的寄存器，而每一个数组元素的大小为16位，所以我们可以同时进行8个16位数的运算，在本题中我们可以同时进行8个乘法运算。

假设：

该指令集中有可以执行八路乘法的指令，也有用8个16位数装填128位寄存器的指令。

代码：

```
F[127:0] = {0, 0, 0, 0, f[3], f[2], f[1], f[0]}
// 把 f[3:0] 装填进 F[127:0]

A[127:0] = {sig_in[7], sig_in[6] ..... sig_in[0]}
// 把 sig_in[7:0] 装填进 A[127:0]
// 因为我们每次循环可以对八个信号处理
// 所以需要 128/8 = 16 次循环就可

for(int i=0; i<16; i++){
    B[127:0] = {sig_in[i*8+7] ..... sig_in[i*8]}
    // 需要 B 来更新 A
    int count = 0;
    for(int j=0; j<8; j++){
        C[127:0] = A*f;
        // A和对应位置上的数相乘
        D[15:0] = C[15:0] + C[31:16] + C[47:32] + C[63:48]
```

```
// C的低四个16位相加保存起来 D[15:0]。
sig_out = D[15:0]; // 输出就为 D[15:0]
count++;
B = {sig_in[i*8+7+count] ..... sig_in[i*8+count]};
// 更新 B
A = A>>16; // 把A右移16位
A = B or A; // A和B做或运算，
};//内循环结束
};//外循环结束
```