Вопрос к лабораторной работе:
на -O0 сказать, в какой строке происходит первое обращение в память (в 3 строке, для ARM несложно определить, что обращение в память происходит после вызова команд str или ldr, еще одним индикатором обращения в память являются квадратные [] скобки)

Some useful info:

The operation part of a floating-point instruction always starts with an F.

The bottom 64-bits of each of the Q registers can also be viewed as D0-D31, 32 64-bit wide registers for floating-point and NEON use.

X – 64-bits and

W – 32-bits of X

Branch	Interpretation	Normal uses
B	Unconditional	Always take this branch
BAL	Always	Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison gave lower
BCS	Carry set	Arithmetic operation gave carry-out
BHS	Higher or same	Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

The B instruction will branch. It jumps to another instruction, and there is no return expected. The Link Register (LR) is not touched.
The BL instruction will branch, but also link. LR will be loaded with the address of the instruction after BL in memory, not the instruction executed after BL. It will then be possible to return from the branch using LR.

```
Power:
        sub     sp, sp, #32      //sp = sp - 32
```

```
        str     d0, [sp, 8]      //store a register value into memeory, d0 is
stored sp+8
        str     d1, [sp]         //d1 is stored in sp
        fmov    d0, 1.0e+0       //d0 = 1.0
        str     d0, [sp, 24]     //d0 is stored in sp+24
        mov     w0, 1            //w0 = 1
        str     w0, [sp, 20]     // w0 is stored in sp+20
        b       .L2              //jump
.L3:
        ldr     d1, [sp, 24]     //d1 = sp + 24
        ldr     d0, [sp, 8]      //d0 = sp + 8
        fmul    d0, d1, d0       //d0 = d1 * d0
        str     d0, [sp, 24]     //sp + 24 = d0
        ldr     w0, [sp, 20]     //w0 = sp + 20
        add     w0, w0, 1        //w0 = w0 + 1
        str     w0, [sp, 20]     //sp + 20 = w0
.L2:
        ldr     w0, [sp, 20]    //w0 = sp + 20
        scvtf   d0, w0          //convert fixed-point w0 to floating-point format
and store in d0
        ldr     d1, [sp]        //d1 = sp
        fcmpe   d1, d0          //d1 – d0
        bge     .L3             //jump if greater or equal
        ldr     d0, [sp, 24]    //d0 = sp + 24
        add     sp, sp, 32      //sp = sp + 32
        ret
.LC0:
        .string "%f\n"
main:
        stp     x29, x30, [sp, -64]!
pre-index variant that modified the address before storing.
add sp, sp, -64
stp x29, x30, [sp] (sp = x29, sp + 8 = x30)

        mov     x29, sp              //x29 = sp
        str     d8, [sp, 16]         //sp + 16 = d8
        mov     x0, 211106232532992  //x0 = 211106232532992
        movk    x0, 0x4062, lsl 48   //moves an immediate value but leaves
the other bits of the register untouched (the K is for keep)
      в младшие 16 бит записываем число 211106232532992, потом начиная с 48 бита
записываем число 0x4062
        fmov    d0, x0               //d0 = x0
        str     d0, [sp, 40]         //sp + 40 = s0
        fmov    d0, 5.0e-1           //d0 = 0.5
        str     d0, [sp, 32]         //sp + 32 = d0
        str     xzr, [sp, 56]        //sp + 56 = xzr
        fmov    d0, 1.0e+0           //d0 = 1.0
        str     d0, [sp, 48]         //sp + 48 = d0
        b       .L6                  //безусловыний branch
.L7:
```

```
        ldr     d1, [sp, 48]            //d1 = sp + 48
        fmov    d0, 1.0e+0              //d0 = 1.0
        fadd    d0, d1, d0              //d0 = d1 + d0
        fmov    d1, d0                  //d1 = d0
        fmov    d0, -1.0e+0             //d0 = -1.0
        bl      Power
        fmov    d8, d0                  //d8 = d0
        ldr     d1, [sp, 48]            //d1 = sp + 48
        ldr     d0, [sp, 32]            //d0 = sp + 32
        bl      Power
        fmul    d1, d8, d0              //d1 = d8 * d0
        ldr     d0, [sp, 48]            //d0 = sp + 48
        fdiv    d0, d1, d0              //d0 = d1 / d0
        ldr     d1, [sp, 56]            //d1 = sp + 56
        fadd    d0, d1, d0              //d0 = d1 + d0
        str     d0, [sp, 56]            //sp + 56 = d0
        ldr     d1, [sp, 48]            //d1 = dp + 48
        fmov    d0, 1.0e+0              //d0 = 1.0
        fadd    d0, d1, d0              //d0 = d1 + d0
        str     d0, [sp, 48]            //sp + 48 = d0
.L6:
        ldr     d1, [sp, 48]            //d1 = dp + 48
        ldr     d0, [sp, 40]            //d0 = dp + 40
        fcmpe   d1, d0                  //flag d1 - d0
        bls     .L7                     //0 or negative (lower or same)
        ldr     d0, [sp, 56]            //d0 = sp + 56
        adrp    x0, .LC0                //
        add     x0, x0, :lo12:.LC0      //x0 = x0 + :lo12:.LC0
        bl      printf
        mov     w0, 0                   //w0 = 0
        ldr     d8, [sp, 16]            //d8 = sp + 16
        ldp     x29, x30, [sp], 64      //x29 = sp, s30 = sp+8, sp = sp + 64
        ret



Power:
        fmov    d2, 1.0e+0      //d2 = 1
        mov     w0, 1           //w0 = 1
        fcmpe   d1, d2          флаг по d1-d2
//floating-point compare. If E ispresent, an exception is raised if either
operand is any kind of NaN. The FCMP instruction subtracts the value in Fm from
the value in Fd and sets the VFP condition flags on the result. FCMP instructions
can produce Invalid Operation exceptions.
        bmi     .L1             result minus or negative // "jumps", to the address
specified if, and only if the negative flag is set. If the negative flag is clear
when the CPU encounters a BMI instruction, the CPU will continue at the instruction
following the BMI rather than taking the jump.
.L4:
```

```
        add     w0, w0, 1     //w0 = w0 + 1
        fmul    d2, d2, d0    //d2=d2*d0
```
FMUL and FNMUL operations can produce Invalid Operation, Overflow, Underflow, or Inexact exceptions.
```
        scvtf   d3, w0        //covert fixed-point w0 to floating-point format
```
and store in d3
```
        fcmpe   d3, d1        //flag d3 – d1
        bls     .L4           // (if (or (not cbit) zbit) (set pc soffset8))
                               типа lower or same
.L1:
        fmov    d0, d2        //d0 = d2
        ret
.LC0:
        .string "%f\n"
main:
        fmov    d5, 1.0e+0            //d5 = 1.0
        movi    d0, #0                //d0 = 0
```
if s (i) is specified:
    updates the N and Z flags according to the result

    can update the C flag during the calculation of Operand2

    does not affect the V flag

```
        stp     x29, x30, [sp, -16]! //sp = sp – 16, sp = x29, s + 8 = x30
        mov     w1, 150              //w1 = 150
        fmov    d7, d5               //d7 = d5
        fmov    d6, 5.0e-1           //d6 = 0.5
        mov     x29, sp              //x29 = sp
.L16:
        fmov    d4, d5               //d4 = d5
        mov     w0, 1                //w0 = 1
        fadd    d5, d5, d7           //d5 = d5 + d7
        fmov    d1, 1.0e+0           //d1 = 1
        fcmpe   d4, #0.0             //d4 - 0;
        bmi     .L11
.L13:
        add     w0, w0, 1
        fneg    d1, d1
        scvtf   d2, w0
        fcmpe   d5, d2
        bge     .L13
        fcmpe   d4, d7
        mov     w0, 1
        fmov    d2, 1.0e+0
        bmi     .L11
.L15:
        add     w0, w0, 1
        fmul    d2, d2, d6
        scvtf   d3, w0
```

```
        fcmpe   d3, d4
        bls     .L15
        fmul    d1, d1, d2
.L11:
        fdiv    d1, d1, d4
        subs    w1, w1, #1
        fadd    d0, d0, d1
        bne     .L16
        adrp    x0, .LC0
        add     x0, x0, :lo12:.LC0
        bl      printf
        mov     w0, 0
        ldp     x29, x30, [sp], 16
        ret
```