ARMAssemblyInt:FloatingPoint:B

```
01     .arch armv6
02     .section      .rodata
03     .align 2
04 .LC0:
05     .ascii "%lf%lf%lf\000"
06     .align 2
07 .LC1:
08     .ascii "Root is %lf\012\000"
09     .text
10     .align 2
11     .global       main
12     .type  main, %function
13 main:
14     @ args = 0, pretend = 0, frame = 32
15     @ frame_needed = 1, uses_anonymous_args = 0
16     stmfd  sp!, {fp, lr}
17     add    fp, sp, #4
18     sub    sp, sp, #32
19     sub    r1, fp, #20 @ a at [fp, #-20]
20     sub    r2, fp, #28 @ b at [fp, #-28]
21     sub    r3, fp, #36 @ c at [fp, #-36]
22     ldr    r0, L3+8
23     bl     __isoc99_scanf
24     fldd   d6, [fp, #-28] @ d6 = b
25     fldd   d7, [fp, #-28] @ d7 = b
26     fmuld  d6, d6, d7     @ d6 = b*b
27     fldd   d7, [fp, #-20] @ d7 = a
28     fldd   d5, .L3        @ d5 = 4.0
29     fmuld  d5, d7, d5     @ d5 = a*4.0
30     fldd   d7, [fp, #-36] @ d7 = c
31     fmuld  d7, d5, d7     @ d7 = 4.0*a*c
32     fsubd  d7, d6, d7     @ d7 = b*b - 4.0*a*c
33     fcpyd  d0, d7         @ d0 = d7, to pass to sqrt
34     bl     sqrt           @ run sqrt
35     fcpyd  d6, d0         @ d6 = sqrt(b*b - 4.0*a*c)
36     fldd   d7, [fp, #-28] @ d7 = b
37     fsubd  d6, d6, d7     @ d6 = sqrt(b*b - 4*a*c) - b
38     fldd   d7, [fp, #-20] @ d7 = a
39     faddd  d7, d7, d7     @ d7 = 2*a
40     fdivd  d7, d6, d7     @ d7 = sqrt(b*b - 4*a*c) - b/(2*a)
41     fstd   d7, [fp, #-12] @ x = d7
42     ldr    r0, .L3+12         @ "Root is %lf\n" as param 1
43     ldrd   r2, [fp, #-12] @ x as param 2 in r2/3
44     bl     printf
45     mov    r3, #0
46     mov    r0, r3
47     sub    sp, fp, #4
48     @ sp needed
49     ldmfd  sp!, {fp, pc}
50 .L4:
51     .align 3                  double value:
52 .L3:                          0x4010000000000000
53     .word  0
54     .word  1074790400 @ 0x40100000
55     .word  .LC0
56     .word  .LC1
```

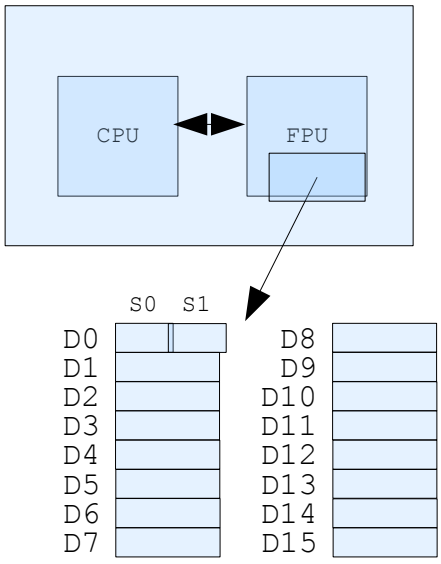
```
int main() {
   double a, b, c, x;

   scanf("%lf%lf%lf", &a, &b, &c);

   x = (-b + sqrt(b*b - 4*a*c)) / (2*a);

   printf("Root is %lf\n", x);

   return 0;
}
```




```
00001000110010101110101011111111

big-endian
00001000  11001010  11101011  01111111

  100       101       102       103

little-endian
01111111  11101011 11001010  00001000

  100       101       102       103
```

```
0 10000000001 0000000000000000000000000000000000000000000000000000
+     2^2       1.0
```