**1. Problem 2.87**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Description | Hex | M | E | V | D |
| -0 | 8000 | 0 | 0 | -0 | -0.0 |
| Smallest value > 2 |  |  |  |  |  |
| 512 | 6000 | 0 | 24 | 512 | 512.0 |
| Largest denormalized |  |  |  |  |  |
| -∞ | FC00 | 0 | 31 | -∞ | -∞ |
| Number with hex representation 3BB0 | 3BB0 |  | -1 |  | 0.9609375 |

**2. Problem 2.93**

float\_bits absVal = (~(1 << 31) & f);

float\_bits NaN = !((absVal >> 23) ^ 0xFF) && ((0xFF << 23) ^ absVal);

if(NaN) return f;

return absVal;

**3. Problem 3.58**

int decode2(long x, long y, long z)

{

y -= z;

x = x \* y;

long a = y;

a = a << 63;

a = a >> 63;

a = a ^ x;

return a;

}

subq %rdx, %rsi

movq %rsi, %rax

salq $63, %rax

sarq $63, %rax

imulq %rdi, %rsi

xorq %rsi, %rax

ret

**4. Problem 3.59**

store\_prod:

movq %rdx, %rax // moves y into rax

cqto // rax = yl rdx = yh

movq %rsi, %rcx // rcx = x

sarq $63, %rcx // rcx = all 0 if x is positive rcx = all FF if x is negative

imulq %rax, %rcx // rcx = yl \* rcx

imulq %rsi, %rdx // rdx = xl \* yh

addq %rdx, %rcx // rcx = (yl \* xsign) + (xl \* yh)

mulq %rsi // rax = lower 64 bit of unsigned multiply of x and 7; rdx = upper 64 bit

addq %rcx, %rdx // rdx = (yl \* xsign) + (xl \* yh) + (x \* y)h

movq %rax, (%rdi) //moves the lower 64 bit into dest

movq %rdx, 8(%rdi) //moves the upper 64 bit into dest

ret

The lower 64 bit can be calculated easily with an imulq. The upper 64 bit can be obtained by multiplying yl by all 0s if x is positive or all 1s if x is negative and then adding (xl \* yh) and the upper 64 bit of x unsigned multiply y.

**5. Problem 3.60**

A. x = rdi; n = esi; result = rax; mask = rdx;

B. result = 0; mask = 1;

C. >= 0

D . mask gets bit shifted left by n.

E. result gets updated by orq %r8, %rax. result is equal to result | (mask & x)

F.

int loop(long x, int n)

{

long result = 0;

long mask;

for (mask = 1; mask >= 0; mask = mask << n) {

result |= mask & x;

}

return result;

}

loop:

.LFB0:

.cfi\_startproc

movl %esi, %ecx

movl $1, %edx

movl $0, %eax

.L3:

movq %rdx, %r8

andq %rdi, %r8

orq %r8, %rax

salq %cl, %rdx

testq %rdx, %rdx

jns .L3

rep ret

.cfi\_endproc

gcc -S -O1 -m64 3\_60.c

This ends up basically the same as the assembly code in the textbook except the order rdx and rdi are anded are flipped, which doesn't matter. Also the textbook assembly code has an extra section that is not created in my code, but it does the same thing.