Please comment any errors, thanks :D

a)

Local variables can either be allocated on the stack or they can be stored in registers. If allocating on the stack, we subtract the space that we need from the stack pointer after the base pointer has been set up. This way we can refer back to our local variables using an offset from the base pointer. If storing in registers, we need to be careful that our variables are not overwritten accidentally.

b)

Binary IEEE;

1100 0001 0001 0001 0000 0000 0000 0000

Sign bit: 1 (negative)

Exponent: 130 (excess 127) => 3

Significand: 1.001 0001 0000 0000 0000 0000

8 + 1 + 1/16 = 9.0625

So **decimal value is: -9.0625**

-9 as binary is -1001, 0.0625 as binary = 0.0001

so **binary value is -1001.0001**

When they say ‘binary’ value, do they want the Two’s Complement? *I think they just mean the binary equivalent of the decimal value.*

c)

-666.75

Sign bit: 1

666: 1010011010

0.75: 11

666.75: 1010011010.11

Exponent: 9 => 136 (excess 127)

which is 1000 1000 in binary

1100 0100 0010 0110 1011 0000 0000 0000

C 4 2 6 B 0 0 0

d)

; int rex (int a, b, c)

rex:

push rbp ; Save base pointer

mov rbp, rsp ; Set up new base ptr

; we don't need to save rax as this is used for the return value

push rbx ; variable e

push rcx ; temporary

mov [rbp+16],3 ; a = 3

mov [rbp+24], 2 ; b = 2

mov rax, [rbp+24] ; d = b

sub rax, [rbp+32] ; d = b - c

imul rax, 2 ; d = 2 \* ( b \- c )

cmp rax, 0 ; d == 0

je endwhile ; skip loop

while:

cmp rax, 0 ; d > 0

jle else ; go to else if d <= 0

dec rax ; d = d - 1

jmp endif ; go to end of if

else:

inc rax ; d = d + 1

endif:

; push args for call

push rax ; save d

mov rcx, [rbp+32] ; arg c

push rcx

mov rcx, [rbp+16] ; arg a

push rcx

push rax ; arg d

call rex ; rex(d, a, c)

add rsp, 24 ; remove args

mov rbx, rax ; e = return value

pop rax ; d = previous value

; finish while loop

cmp rax, 0 ; d != 0

jne while ; next iteration

endwhile:

pop rcx ; restore rcx

pop rbx ; restore rbx

; we don't restore rax as this is our return value

pop rbp ; restore rbp

ret ; jump to caller