1. push ebp **set up base pointer**
mov ebp, esp **remember the address of stack pointer to return to**
and esp, 0FFFFFFF0h **set up stack pointer**
sub esp, 20h **move stack pointer down to allow room for local vars**
call ___main
mov dword ptr [esp+1Ch], 3 **store 3 onto stack. basically make local var with value 3**
mov dword ptr [esp+18h], 5 **store 5 onto stack**
mov dword ptr [esp+14h], 0 **store 0 onto stack**
mov eax, [esp+1Ch] **move the value 3 into eax register**
imul eax, [esp+18h] **integer multiply eax(3) by 5 and store in eax**
mov edx, eax **move eax(15) into edx. both eax and edx now store 15**
mov eax, [esp+1Ch] **move 3 into eax**
mov ecx, eax **move 3 into ecx from eax. Now both eax and ecx have 3 stored inside**
shr ecx, 1Fh **shift right 31 bits, isolating the sign bit of ecx. ecx was positive so this is 0 now**
add eax, ecx **add 0 to eax and store in eax. eax still holds 3**
sar eax, 1 **shift the value in eax by one bit, essentially dividing by 2. This and the previous two lines amount to signed division by 2.**
sub edx, eax **Subtract 1 from edx(15) and store into edx. edx now holds 14.**
mov eax, edx **Move 14 into eax**
mov [esp+14h], eax **Store 14 into the variable at esp+14h**
mov eax, [esp+14h] **Get 14 from the memory variable and put it into eax**
mov [esp+4], eax **Move 14 into the esp + 4 memory address**
mov dword ptr [esp], offset aD ; "%d"
call _printf **Print the value 14 to the screen**
mov eax, 0 **0 out eax**
leave **done**
retn
_main endp

2. The functionality is as follows: **a**. The program declares 8 local variables with values 12, 15, 221, 3, 422, 54, 16, 67. It also declares two variables, one for the loop index(stored at [esp + 38h]) and a temp variable to store the max(explained afterwards. located at [esp + 3Ch]. This is shown in lines 40150E to 401556. **b**. Then, the program starts a loop and compares each of the local variables to the value stored in the max local variable. If the value of one of the local vars is greater than the value currently stored in[esp + 3Ch], that value is stored in [esp + 3Ch] and the loop iteration ends. This is shown in lines 40157F to 40157A(the left branch of

the IDA screenshot). **c**. Once the loop is over, the program prints the value stored at [esp + 3Ch]. This is the maximum value out of all 8 temporary variables. This is shown in the right branch of the IDA screenshot.

Essentially, this program finds the maximum value of all of the declared local variables. In my C implementation, I used an array rather than local variables to achieve the same functionality(the way this assembly is written tells me that this is some compiler optimization that turned a pointer into local vars using the stack).

3. Not really sure. I realize the multiplication by huge numbers and shifting is a compiler optimization for integer division by a constant. However, I couldn't figure out the whole thing.

4. This program first makes a buffer that contains the numbers 1 through 100 usin g a loop. This is shown in the left side of the IDA screenshot of this program(not inside the proc function). Once that buffer is made the proc function is called with parameters being the buffer, 7, and 100. Inside that function, the following algorithm is run:

1. Every 7th number is 0'd out. So the values 7, 14, 21, 28, 35 … 98 are 0'd out. This is shown in the furthest right branch of the IDA screenshot inside the proc function. Once we get to 98, the next number would be 105. However, once the value is greater than 100, the algorithm 0's out x - 100. Thus, in this case, the value 5 is 0'd out.
2. Now the algorithm 0's out on a +8 basis, so the value 13 is 0'd, and 21 should be zerod. However, due to the left branch of the screenshot in IDA, as 21 is already 0, the index actually moves up to the first non-0 value so the value 22 is 0'd. Then 30(22 + 8), 38(30 + 8), 46(38 + 8), … 95. Again, adding 8 would get us 103, but the algorithm subtracts 100 and zero's out 3.
3. Now the algorithm 0's out on a +9 basis and the process repeats with +10, +11, … until 99 indexes are 0'd out, leaving the number 50 which is printed out. Everytime a value is turned into 0, the outer loop iteration ends and a count is increased by 1.