Basic Assembler

Code Flow:

When analyzing a piece of code, you should realize that the processor is dumb and it just follows one line after the other... It does anything the code tells it to do.

To do this, the processor has an instruction pointer. With it, it points to the instruction it wants to execute.

The processor also executes instructions that change the flow of the code.

When trying to understand a piece of code, try to be the processor....
Compare registers, compare variables, execute jumps, and execute calls. This is the best way to understand a piece of code.

Registers:

Registers are variables the processor uses. The most used ones are : eax, ebx, ecx and edx. You will also see edi, esi, esp, ebp.

These registers which start with an e, are all 32 bits registers. The 16 bit equivalents of these registers you will also see here and there. These are actually the same registers, but the variable size is only 16 bits. These registers are: ax, bx, cx, dx, di, si, sp, bp.

While we are at it, there are also 8 bit values of some of the registers.
You might see the following 8 bit registers: al, ah, bl, bh, cl, ch, dl, dh.
Here the l stands for the lower and h for the higher 8 bits of a 16 bit register.

Flags:

Flags are boolean variables the processor uses for itself to check what the result of a logical instruction was. The most important flag is the **Zero Flag**. Its value can be zero or non-zero.

Frequently seen Assembler instructions:

cmp xxx, yyy	This instruction compares value xxx with value yyy. if they are equal, the zero flag is set.
test xxx, xxx	Compares if value xxx is equal to zero. If it is, the zero flag is set.
jmp <address>	Unconditional jump. Jumps always. Not important for crackers
jz <address>	jump if Zero flag is set. Same as **je**
jnz <address>	jump if Zero flag is Not set. same as **jne**
jxx <address>	jump if <condition>greater, less, greater or equal, less or equal.
call <address>	The Processor saves the address of the next instruction, then it jumps to address <address>. Executes all lines until it counters a ret instruction. After the **ret** instruction it returns to the line of next instruction These are subroutines / procedures / functions.
push <variable>	This moves a value into the memory the processor uses. This is usually done before a call is executed. The parameters of the

function are 'pushed'. The values are moved into memory so that the subroutine can access them easily.