S10-L3

by Otman Hmich & Michael Andreoli

Explain the following code line by line:

```
0x00001141 <+8>:    mov   EAX,0x20
0x00001148 <+15>:   mov   EDX,0x38
0x00001155 <+28>:   add   EAX,EDX
0x00001157 <+30>:   mov   EBP, EAX
0x0000115a <+33>:   cmp   EBP,0xa
0x0000115e <+37>:   jge    0x1176 <main+61>
0x0000116a <+49>:   mov   eax,0x0
0x0000116f <+54>:   call   0x1030 <printf@plt>
```

Assembly language, is a low-level programming language closely related to a computer's architecture.

Low-Level: It is called "low-level" because it operates directly on the processor's instructions, using a language very close to machine code (which the CPU can execute directly).

Simple Instructions: Each instruction in assembly represents a very simple and specific operation, such as moving data between registers, performing calculations, or controlling the program's execution flow.

Registers and Memory: Operations work with registers (small memory units within the CPU) and memory addresses. For example, you can load a value into a register, perform a calculation with it, and then save the result in another register or in memory.

Hardware-Linked: Each type of processor (x86, ARM, etc.) has its own set of assembly instructions, which means that assembly code written for one type of processor will not work on another type without modifications.

Main Use: It is mainly used when precise control over hardware is needed, such as in operating systems, drivers, firmware, and some high-performance

applications.

X86 REGISTERS:

EAX: accumulator register, used for arithmetic and logical operations.

EBX: base register, often used for data pointers.

ECX: counter register, used in loop operations.

EDX: data register, often used in input/output operations and multiplications.

0x00001141 <+8>: mov EAX,0x20

0x00001141 <+8> represents the memory address where 15 bytes later the operation is located. We move the hexadecimal value 0x20 (32 in decimal) into the EAX register.

0x00001148 <+15>: mov EDX,0x38

0x00001148 <+15> represents the memory address where 15 bytes later the operation is located. We move the hexadecimal value 0x38 (56 in decimal) into the EDX register.

0x00001155 <+28>: add EAX,EDX

0x00001155 <+28> represents the memory address where 15 bytes later the operation is located.

Add the contents of the EDX register (56) to that of EAX (32), so the content of EAX is now 88.

0x00001157 <+30>: mov EBP,EAX

0x00001157 <+30> represents the memory address where 15 bytes later the operation is located.

Move the content of the EAX register (88) into the EBP register.

0x0000115a <+33>: cmp EBP,0xA

0x0000115a <+33> represents the memory address where 15 bytes later the operation is located.

Compare the value 0xA (10) to the EBP register (88).

The ZF and CF flags are both set to 0 since the destination (EBP) is greater than the source (0xA).

0x0000115e <+37>: jge 0x1176 <main+61>

0x0000115e <+37> represents the memory address where 15 bytes later the operation is located. We jump to the memory location 0x1176 because in the comparison (cmp), the destination is greater than the source.

x0000116a <+49>: mov EAX,0x0

0x0000116a <+49> represents the memory address where 15 bytes later the operation is located.

Override the value 0x0 (0) to the EAX register.

0x0000116f <+54>: call 0x1030 <printf@plt>

0x0000116f <+54> represents the memory address where 15 bytes later the operation is located.

Using the call operand, we call the printf function located at the memory address 0x1030.