Assembly x86

Task

We covered the fundamentals of the Assembly language. Given the Assembly code for the x86 CPU below, identify the purpose of each instruction by **providing a description for each line of code**. Remember that numbers in the format **0xYY** are hexadecimal numbers. To convert them to decimal numbers, you can use an online converter or the calculator on your computer (for programmers).

```
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>
```

Solution

Assembly is a **low-level programming language** that provides a way to write instructions directly for a computer's CPU using **mnemonic codes that correspond closely to machine code instructions.**
It allows for precise control over hardware and is specific to a computer architecture.
Programmers use assembly language for performance-critical tasks and system programming where fine-grained hardware manipulation is required.
In malware analysis, assembly language is used to **understand and reverse-engineer malicious software**.

Since malware often avoids detection by using obfuscation techniques, analyzing its assembly code helps analysts see the exact instructions executed by the CPU.
This allows them to identify the malware's **behavior, functionality, and any potential vulnerabilities it exploits**.

Let’s analyze in detail the code provided.

```
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>
```

1. `0x00001141 <+8>: mov EAX,0x20`
   - **Description:** Load the hexadecimal value `0x20` (32 in decimal) into the `EAX` register.
2. `0x00001148 <+15>: mov EDX,0x38`
   - **Description:** Load the hexadecimal value `0x38` (56 in decimal) into the `EDX` register.
3. `0x00001155 <+28>: add EAX,EDX`
   - **Description:** Add the value in the `EDX` register (56) to the value in the `EAX` register (32), resulting in `EAX` holding the value 88.
4. `0x00001157 <+30>: mov EBP, EAX`
   - **Description:** Copy the value in the `EAX` register (88) into the `EBP` register.
5. `0x0000115a <+33>: cmp EBP,0xa`
   - **Description:** Compare the value in the `EBP` register (88) with the hexadecimal value `0xa` (10 in decimal).

6. `0x0000115e <+37>: jge 0x1176 <main+61>`

   - **Description:** Jump to the instruction at address `0x1176` if the value in `EBP` (88) is greater than or equal to 10. This is a conditional jump.

7. `0x0000116a <+49>: mov eax,0x0`

   - **Description:** Load the value `0x0` (0 in decimal) into the `EAX` register. (This instruction is executed only if the previous jump is not taken, i.e., if `EBP` was less than 10.)

8. `0x0000116f <+54>: call 0x1030 <printf@plt>`

   - **Description:** Call the function `printf`. The address `0x1030` is the location of the `printf` function in the Procedure Linkage Table (PLT), which is used for dynamic linking of functions at runtime.

The code compares the sum of 32 and 56 with 10 and, if the sum is greater than or equal to 10, it jumps to another part of the program. Otherwise, it sets `EAX` to 0 and calls the `printf` function.

Federico Biggi