6th laboratory

Rodrigo Castillo

September 14, 2020

1 operations :addition, substraction , increment, decrement and modulo

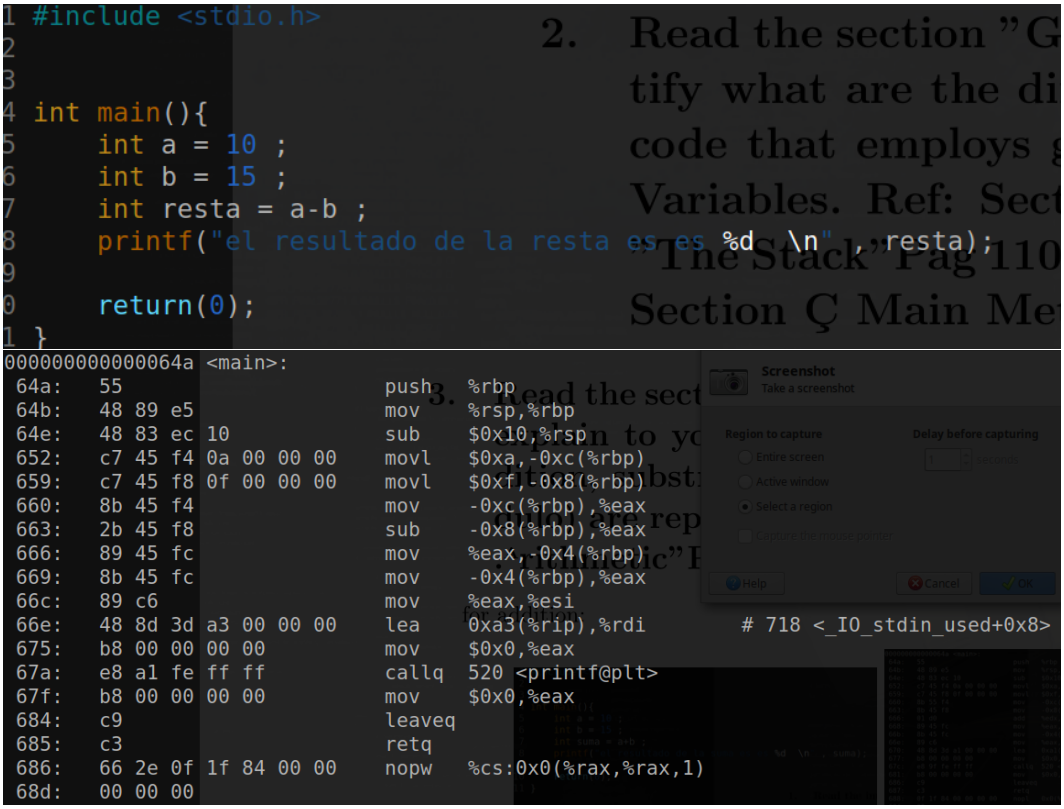
for addition: we can see that the addition operation is in line 666 , in operation *add* , as i defined two variables before, assembly code is locating hex values into the variables i defined and then, he es adding the values into the new variable

```c
#include <stdio.h>


int main(){
    int a = 10 ;
    int b = 15 ;
    int suma = a+b ;
    printf("el resultado de la suma es es %d  \n" , suma);

    return(0);
}
```

```
000000000000064a <main>:
 64a:   55                      push   %rbp
 64b:   48 89 e5                mov    %rsp,%rbp
 64e:   48 83 ec 10             sub    $0x10,%rsp
 652:   c7 45 f4 0a 00 00 00    movl   $0xa,-0xc(%rbp)
 659:   c7 45 f8 0f 00 00 00    movl   $0xf,-0x8(%rbp)
 660:   8b 55 f4                mov    -0xc(%rbp),%edx
 663:   8b 45 f8                mov    -0x8(%rbp),%eax
 666:   01 d0                   add    %edx,%eax
 668:   89 45 fc                mov    %eax,-0x4(%rbp)
 66b:   8b 45 fc                mov    -0x4(%rbp),%eax
 66e:   89 c6                   mov    %eax,%esi
 670:   48 8d 3d a1 00 00 00    lea    0xa1(%rip),%rdi        # 718 <_IO_stdin_used+0x8>
 677:   b8 00 00 00 00          mov    $0x0,%eax
 67c:   e8 9f fe ff ff          callq  520 <printf@plt>
 681:   b8 00 00 00 00          mov    $0x0,%eax
 686:   c9                      leaveq
 687:   c3                      retq
 688:   0f 1f 84 00 00 00 00    nopl   0x0(%rax,%rax,1)
 68f:   00
```

Figure 1: Code and disassembly for addition

for substraction: this is the same as the addition operation, but, the code is replacing addition operation with substraction operation in line 663 .



```
1 #include <stdio.h>
2
3
4 int main(){
5     int a = 10 ;
6     int b = 15 ;
7     int resta = a-b ;
8     printf("el resultado de la resta es es %d  \n" ,resta);
9
0     return(0);
1 }
```

```
000000000000064a <main>:
 64a:   55                      push   %rbp
 64b:   48 89 e5                mov    %rsp,%rbp
 64e:   48 83 ec 10             sub    $0x10,%rsp
 652:   c7 45 f4 0a 00 00 00    movl   $0xa,-0xc(%rbp)
 659:   c7 45 f8 0f 00 00 00    movl   $0xf,-0x8(%rbp)
 660:   8b 45 f4                mov    -0xc(%rbp),%eax
 663:   2b 45 f8                sub    -0x8(%rbp),%eax
 666:   89 45 fc                mov    %eax,-0x4(%rbp)
 669:   8b 45 fc                mov    -0x4(%rbp),%eax
 66c:   89 c6                   mov    %eax,%esi
 66e:   48 8d 3d a3 00 00 00    lea    0xa3(%rip),%rdi        # 718 <_IO_stdin_used+0x8>
 675:   b8 00 00 00 00          mov    $0x0,%eax
 67a:   e8 a1 fe ff ff          callq  520 <printf@plt>
 67f:   b8 00 00 00 00          mov    $0x0,%eax
 684:   c9                      leaveq
 685:   c3                      retq
 686:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
 68d:   00 00 00
```

Figure 2: Code and disassembly for substraction

for increment: first, in line 652 we can see that is adding the value $0xa$ into a register, this value, is 10 , then its adding $0x1$ into the same register, this means that is adding 1 and now the value of this register is 11 .

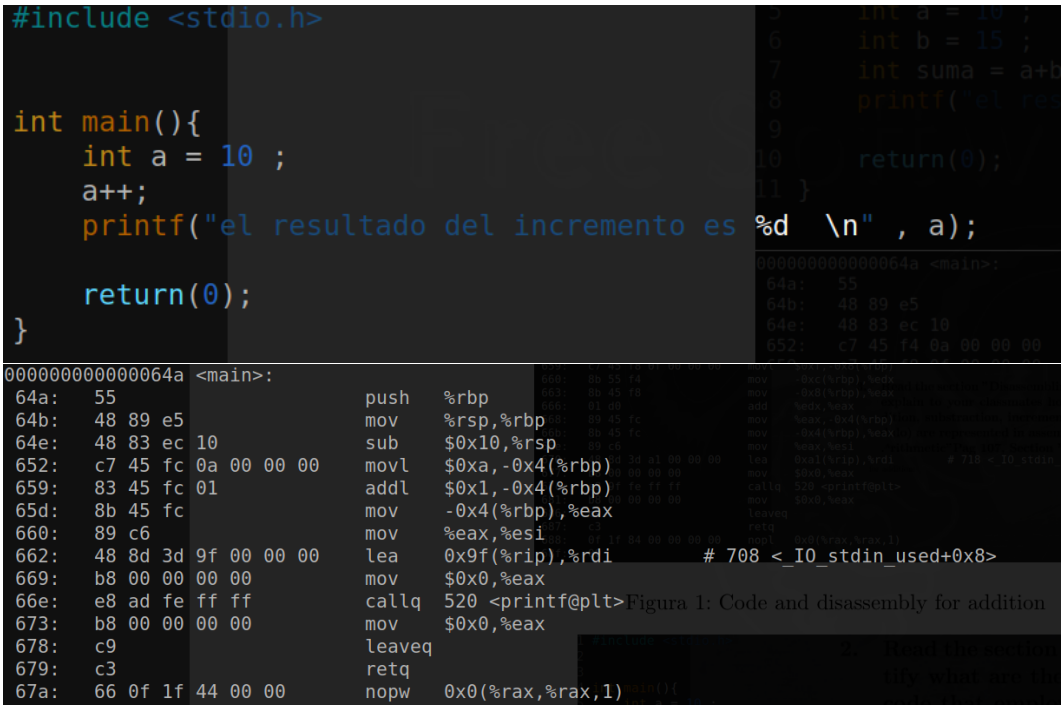
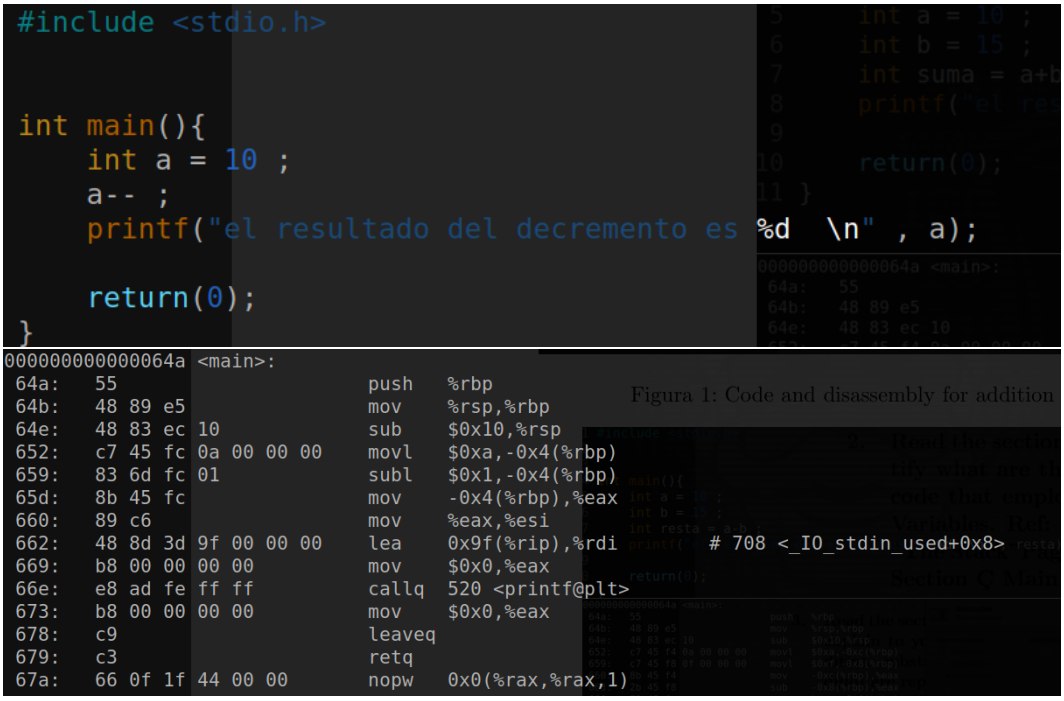


Figura 1: Code and disassembly for addition

Figure 3: Code and dissasembly for increment

for decrement: is exactly the same as increment, but now, he es substracting instead of adding $0x1$ value into the register.



```c
#include <stdio.h>


int main(){
    int a = 10 ;
    a-- ;
    printf("el resultado del decremento es %d  \n" , a);

    return(0);
}
```

```
000000000000064a <main>:
 64a:   55                      push   %rbp
 64b:   48 89 e5                mov    %rsp,%rbp
 64e:   48 83 ec 10             sub    $0x10,%rsp
 652:   c7 45 fc 0a 00 00 00    movl   $0xa,-0x4(%rbp)
 659:   83 6d fc 01             subl   $0x1,-0x4(%rbp)
 65d:   8b 45 fc                mov    -0x4(%rbp),%eax
 660:   89 c6                   mov    %eax,%esi
 662:   48 8d 3d 9f 00 00 00    lea    0x9f(%rip),%rdi        # 708 <_IO_stdin_used+0x8>
 669:   b8 00 00 00 00          mov    $0x0,%eax
 66e:   e8 ad fe ff ff          callq  520 <printf@plt>
 673:   b8 00 00 00 00          mov    $0x0,%eax
 678:   c9                      leaveq
 679:   c3                      retq
 67a:   66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
```

Figura 1: Code and disassembly for addition

Figure 4: Code and disassembly for decrement opperation

for modulo: is actually the same as addition and substraction, but the opperation *idivl* stands for division, the result of this divion is going to store the residue in the designer register.

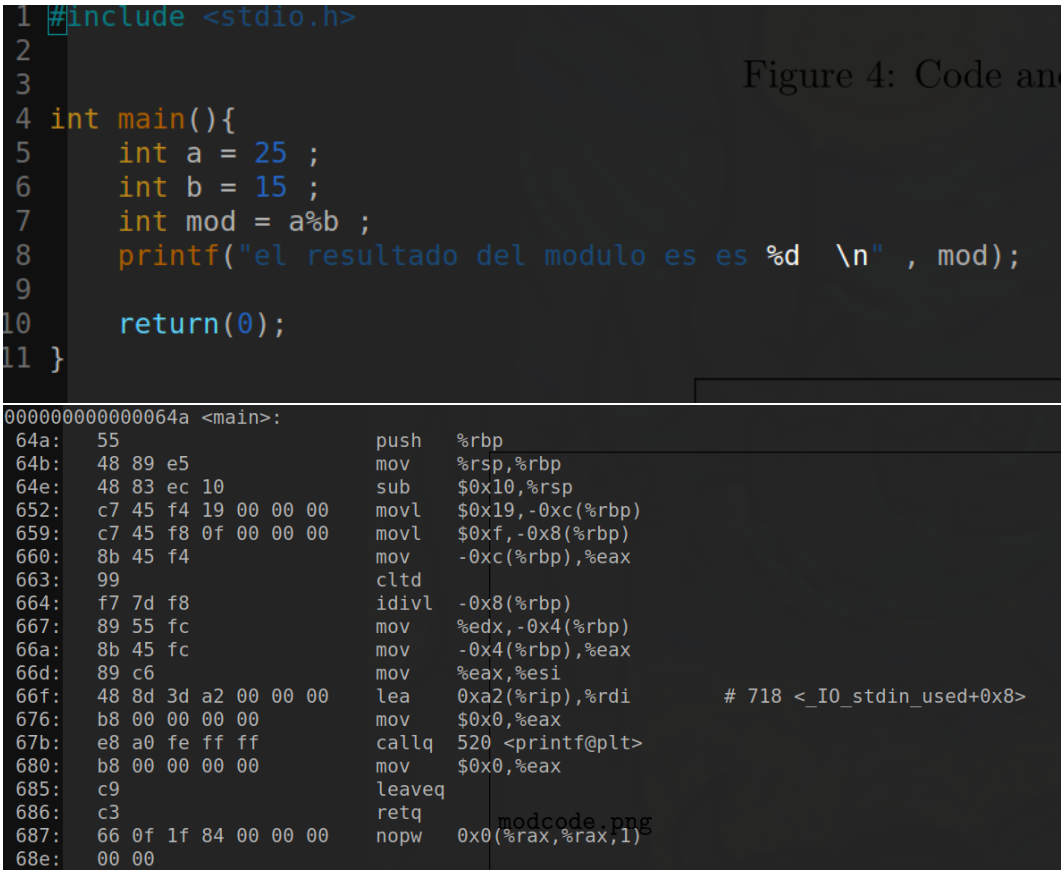


Figure 5: Code for modulo

2 Q4: Read the section "Recognizing if Statements" and explain to your classmates how to recognize an if/else structure in assembly code. Ref: Section "Conditionals" Pag 113, Section "Branching" Pag 113