Basic Assembly

Introduction to subroutines

Assembly language programming
By xorpd

xorpd.net

Objectives

- We will discuss the advantages of breaking our code into smaller pieces.

- We will show a first attempt of breaking our code into smaller pieces.

Managing large code

- As our code becomes longer, it could become harder to manage.
- We want our code to have the following properties:
  - Easy to develop.
    - Particularly – easy to develop in a group of people.
  - Easy to understand.
    - Some pieces of code are so large that no one human being can perceive the full complexity of the system.
  - Easy to read (Even for other people!)
  - Easy to maintain.
    - Changes could be done easily.
    - Bugs or mistakes could be traced easily.

Managing large code (Cont.)

- A common solution would be to **break our code into smaller pieces**.

- Each piece of code will have a specific purpose.
  - We call each piece a **function**, or a **subroutine**.

- While reading the code, instead of thinking about instructions, we could think about functions:
  - We abstract away the internal details of a function.
  - We care about the "what" instead of "how".

- We could always dive into the details of the "how", if we want to.

Managing large code (Cont.)

- You have already seen and used some functions:
  - read_hex
  - print_eax
  - read_line
- Note that you managed to use those functions without knowing how they work internally.

First attempt – Simple jumps

- Example: We want to find all the prime numbers between 1 and n.
  - A prime number is a number that is only divisible by 1 and itself. (Examples: 2,3,5,7,11,13, …)
- We will write a function **is_prime** that given a number in eax, returns:
  - $eax = 0$ if the number is not prime.
  - $eax = 1$ if the number is prime.
- Finally we will invoke the is_prime function inside a loop, for every number between 1 and n.

First attempt – Simple jumps (Cont.)

- Example for is_prime function.

```
is_prime:
    mov     esi,eax
    mov     ecx,1
    xor     edi,edi
    inc     edi
check_divisor:
    inc     ecx
    cmp     ecx,esi
    jae     end_divisors_loop
    mov     eax,esi
    cdq
    div     ecx
    test    edx,edx
    jnz     not_divisor
    xor     edi,edi
not_divisor:
    jmp     check_divisor
end_divisors_loop:
    mov     eax,edi
```

First attempt – Simple jumps (Cont.)

- Example for is_prime function.
    - We don’t care about the details at the moment.

```
is_prime:
    ; Check if eax is prime.
    ; If it is, return eax = 1.
    ; Else return eax = 0
    ...
```

- How can we use this function?

First attempt – Simple jumps (Cont.)

- Simple example of using is_prime:

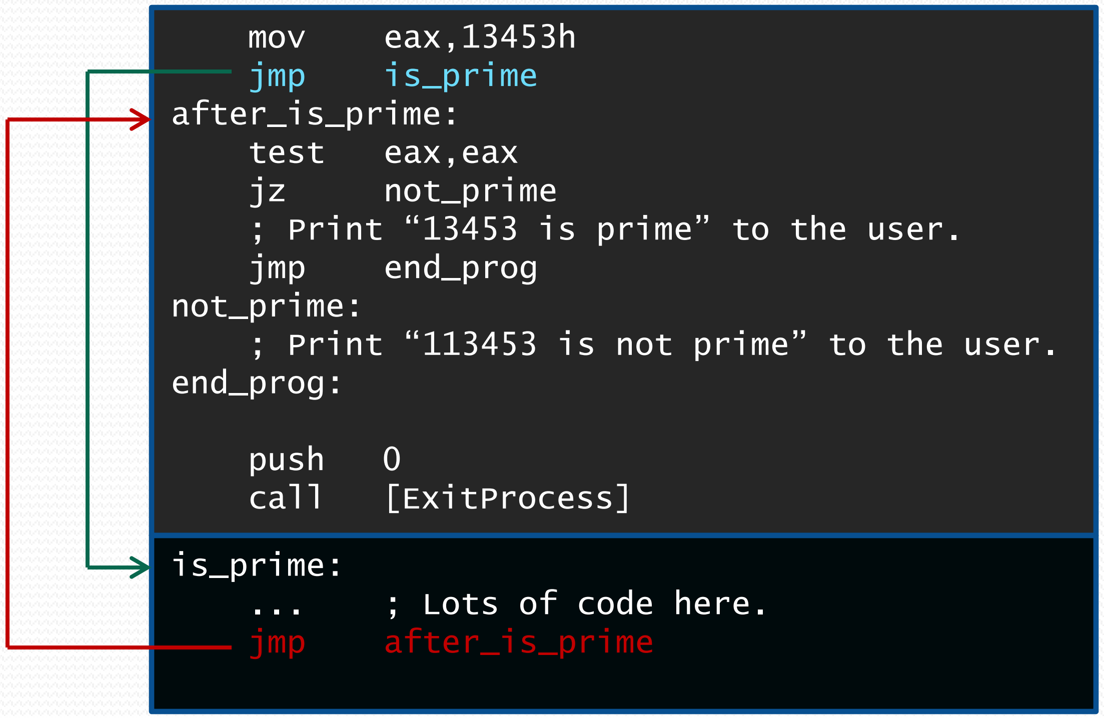
```
    mov     eax,13453h
    jmp     is_prime
after_is_prime:
    test    eax,eax
    jz      not_prime
    ; Print “13453 is prime” to the user.
    jmp     end_prog
not_prime:
    ; Print “113453 is not prime” to the user.
end_prog:

    push    0
    call    [ExitProcess]
```

```
is_prime:
    ...     ; Lots of code here.
    jmp     after_is_prime
```

First attempt – Simple jumps (Cont.)

- Simple example of using is_prime:



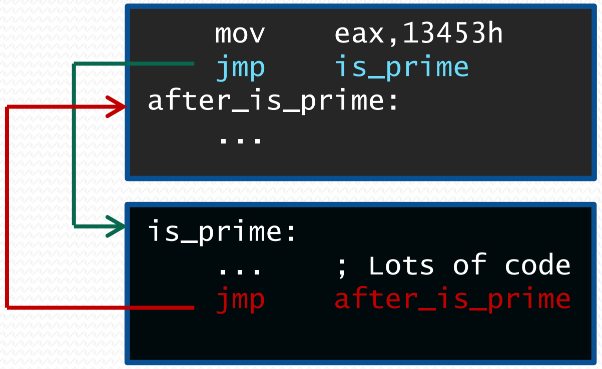
```
    mov     eax,13453h
    jmp     is_prime
after_is_prime:
    test    eax,eax
    jz      not_prime
    ; Print “13453 is prime” to the user.
    jmp     end_prog
not_prime:
    ; Print “113453 is not prime” to the user.
end_prog:

    push    0
    call    [ExitProcess]
```

```
is_prime:
    ...     ; Lots of code here.
    jmp     after_is_prime
```

First attempt – Simple jumps (Cont.)

- Pros:
  - **Abstraction**: The author/reader of the top code doesn't have to know much about primes.
  - **Readability**: The code on the top is easier to read.
  - **Easy to change**: If we wanted a different primality test, we could just change the code on the bottom.



```
    mov     eax,13453h
    jmp     is_prime
after_is_prime:
    ...
```

```
is_prime:
    ...     ; Lots of code
    jmp     after_is_prime
```

- Cons of the simple jumps method:
  - We can only invoke is_prime from one place.
    - is_prime doesn't know where it was invoked from.
  - We have to add the after_is_prime label.
  - is_prime changes the contents of the registers.

First attempt – Simple jumps (Cont.)

- Our first attempt was rather crude.
  - However, we have already achieved some advantages using it.
- We want a better solution for function mechanism.
- Wanted features:
  - Being able to call our functions from anywhere.
  - A function won't change the registers.
    - Or at least won't change most of the registers.

Summary

- Breaking our code into smaller functions makes it easier to understand and maintain.

- We have seen one attempt of breaking our code into smaller functions.
  - It was only partially successful.

- In the next lessons we are going to see a better functions mechanisms.

Further code reading

- Read is_prime.asm.
  - It finds all the prime numbers between 1 and n.
- Understand how it works.