Assembly language programming
By xorpd

BASIC ASSEMBLY

Introduction to branching

xorpd.net

Motivation

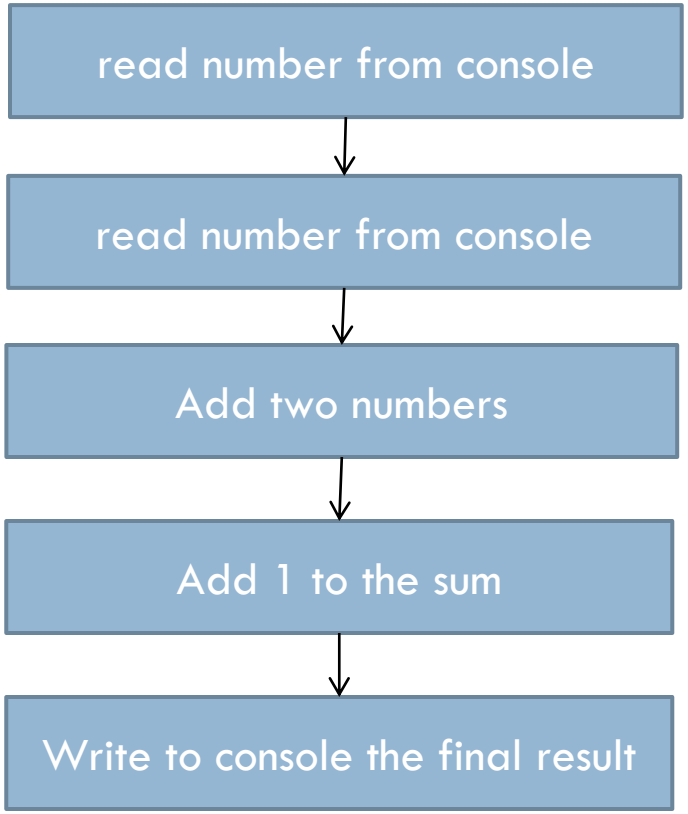
- So far we wrote some very simple programs.
  - We used our computer like a pocket calculator.
- We want to create more advanced programs.
  - Programs that run longer time.
  - Programs that take decisions.

Branching

- So far our programs ran linearly- from beginning to end.
  - No decision was made.
  - The time it took to execute the program was proportional to the amount of code we wrote.
- We would like to be able to do different things according to different results or values that we get.
  - Run a certain piece of code on some condition.
  - Run a certain piece of code many times.

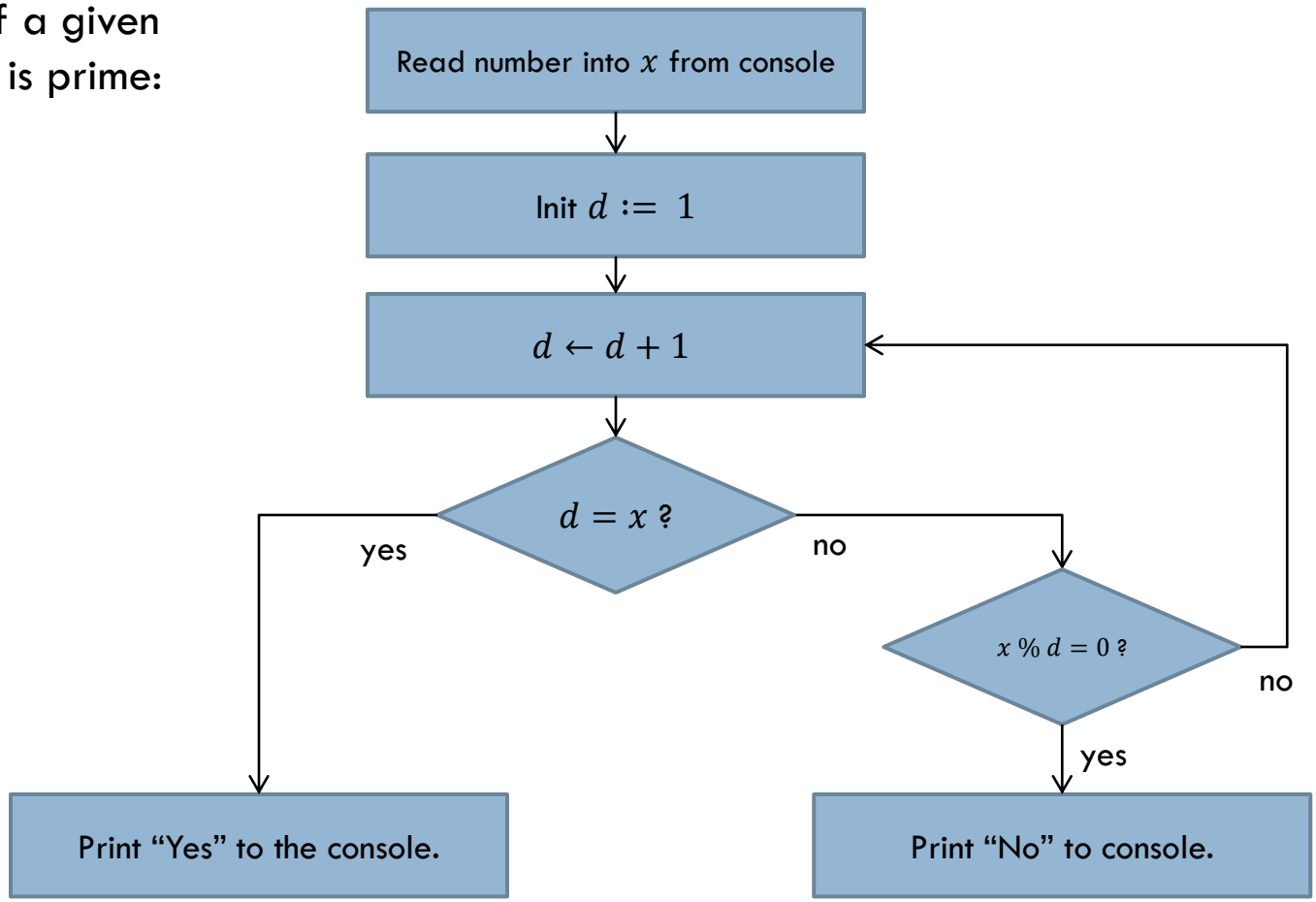
Linear program illustration

```
call    read_hex
mov     ecx,eax
call    read_hex
add     eax,ecx
inc     eax
call    print_eax
```



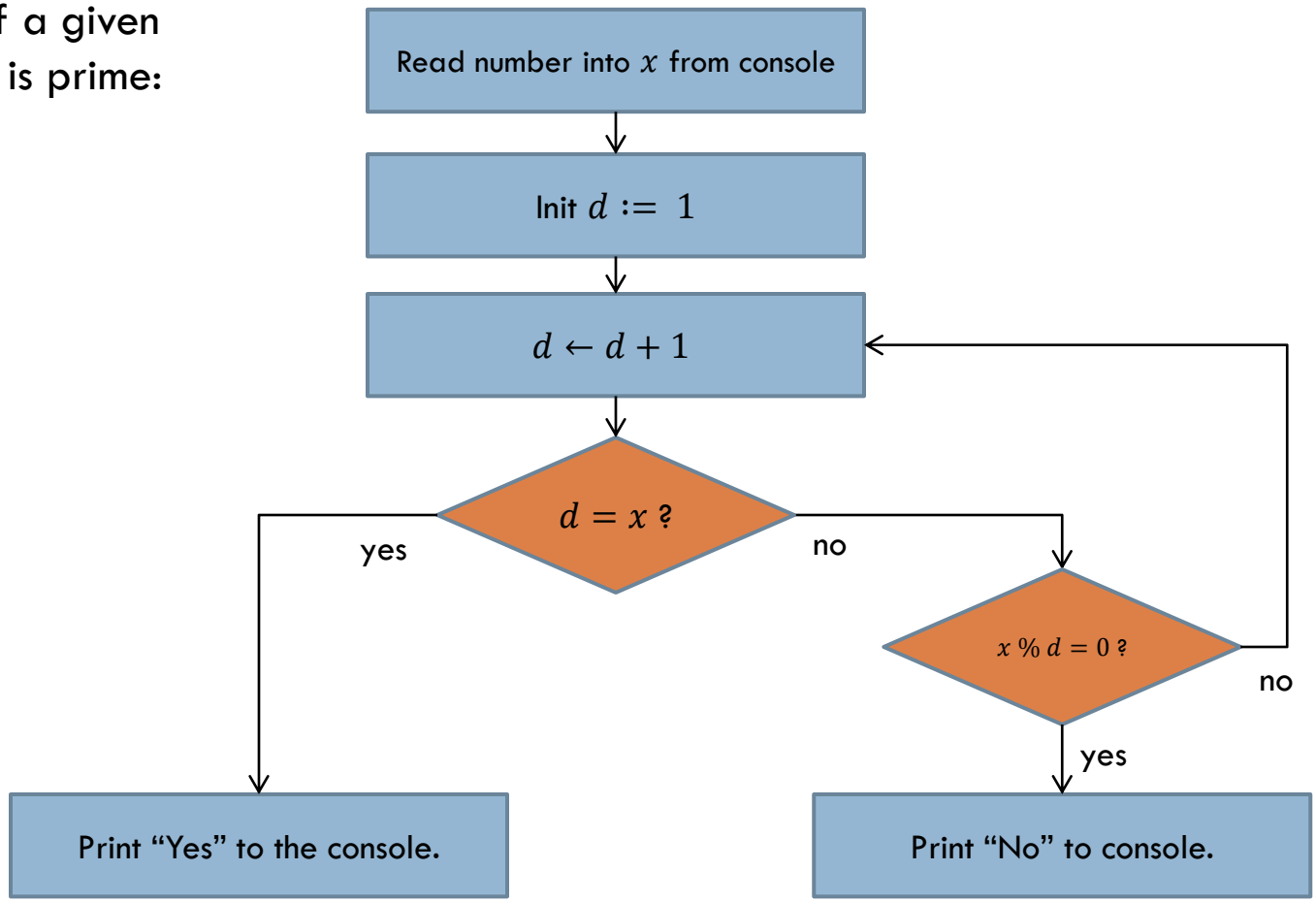
Branching illustration

Check if a given number is prime:



Branching illustration

Check if a given number is prime:



The EIP register

- Extended instruction pointer.
- 32 bits size.
  - 64 bits size in long-mode.
- Contains the address of the current instruction.
  - Points to the current instruction.
- If we want to execute code from a different location, we should change EIP.
- In 32 bit protected mode, EIP could not be changed directly.
  - `mov eip,eax` is not valid.

Unconditional jump

- The JMP instruction allows to set the value of eip.
  - JMP dest
  - Actually "jumps" to a different location in the program, to execute different code.
- Examples:
  - jmp ecx
    - Changes eip to the contents of ecx. The execution will continue from the address ecx. ($eip \leftarrow ecx$).
  - jmp 777d1044h
    - Changes eip to the value 0x777d1044. The program will continue execution on that address. ($eip \leftarrow 0x777d1044$).

Labels

- When writing our programs, we usually can't predict their loading location in memory.
- Labels are a way of referring to a location in our program, without knowing the exact address of that location at runtime.
- Example:

```
        mov ecx,0
my_label:
        inc ecx
        jmp my_label
```

JMP (Example)

```
        mov ecx,0
my_label:
        inc ecx
        jmp my_label
```

JMP (Example)

```
004f1000        mov ecx,0
            my_label:
004f1005        inc ecx
004f1006        jmp my_label
```

JMP (Example)

```
004f1000        mov ecx,0

004f1005        inc ecx
004f1006        jmp 004f1005
```

JMP (Example)

```
004f1000    mov ecx,0

004f1005    inc ecx
004f1006    jmp 004f1005
```

ecx	eip
????????	004f1000

JMP (Example)

```
004f1000        mov ecx,0

004f1005   ⟹    inc ecx
004f1006        jmp 004f1005
```

ecx	eip
00000000	004f1005

JMP (Example)

```
004f1000        mov ecx,0

004f1005        inc ecx
004f1006        jmp 004f1005
```

ecx	eip
00000001	004f1006

JMP (Example)

```
004f1000        mov ecx,0

004f1005   ⟹    inc ecx
004f1006        jmp 004f1005
```

ecx	eip
00000001	004f1005

JMP (Example)

```
004f1000        mov ecx,0

004f1005        inc ecx
004f1006        jmp 004f1005
```

ecx	eip
00000002	004f1006

JMP (Example)

```
004f1000        mov ecx,0

004f1005   ⟹    inc ecx
004f1006        jmp 004f1005
```

ecx	eip
00000002	004f1005

JMP (Example)

```
004f1000        mov ecx,0

004f1005        inc ecx
004f1006        jmp 004f1005
```

ecx	eip
00000003	004f1006

JMP (Example)

```
004f1000        mov ecx,0

004f1005        inc ecx
004f1006        jmp 004f1005
```

ecx	eip
00000003	004f1006

Inifinite loop!

JMP (Cont.)

- It is the job of the assembler to translate labels into actual addresses.
- JMP looks like a simple instruction, however it has few different encoding versions:
  - Absolute jump – Jump to a specified location in memory.
  - Relative jump – Jump to a location which is X bytes from this location.
- The assembler will pick the suitable version for you.
  - So don't worry about it at the moment.

JMP (Cont.)

- Jump allows to change eip unconditionally.
- We would like to change eip conditionally:
  - Based on some previous values that we have obtained.

- How could we do that?
  - We will learn in the following lectures how to branch our code according to the result of the last calculation.

Summary

- So far we created only linear programs.
  - We used our computer like a pocket calculator, which doesn't really give us much power as programmers.
- The JMP instructions allows us to branch unconditionally.
  - We created a simple loop to demonstrate that.
- We will later learn how to branch conditionally –
  - Branch according to the result of the last calculation.