ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x07

DE LA COD SURSĂ LA EXECUȚIE

Cristian Rusu

DATA TRECUTĂ

- **arhitectura de bază a calculatoarelor**

- **Instruction Set Architecture (ISA)**

CUPRINS

- **scurt review arhitectura de bază a calculatoarelor**

- **de la cod sursă la cod mașină**
  - software cracking
  - executarea datelor

ARHITECTURA DE BAZĂ

Periferice Intrare

Periferice Ieșire

BUS

Unitatea Centrală de Procesare (CPU)

Unitatea de Control

Unitatea Artimetică/Logică

Regiștri

Memoria Principală

Memoria de Stocare

ARHITECTURA DE BAZĂ

Periferice Intrare

Periferice Ieșire

BUS

Unitatea Centrală de Procesare (CPU)

Unitatea de Control

Unitatea Artimetică/Logică

Regiștri

Memoria Principală

Memoria de Stocare

comunicarea cu perifericele se face de obicei prin buffer-e în memorie

ARHITECTURA DE BAZĂ

Periferice Intrare

Periferice Ieșire

BUS

Unitatea Centrală de Procesare (CPU)

Unitatea de Control

Unitatea Artimetică/Logică

Regiștri

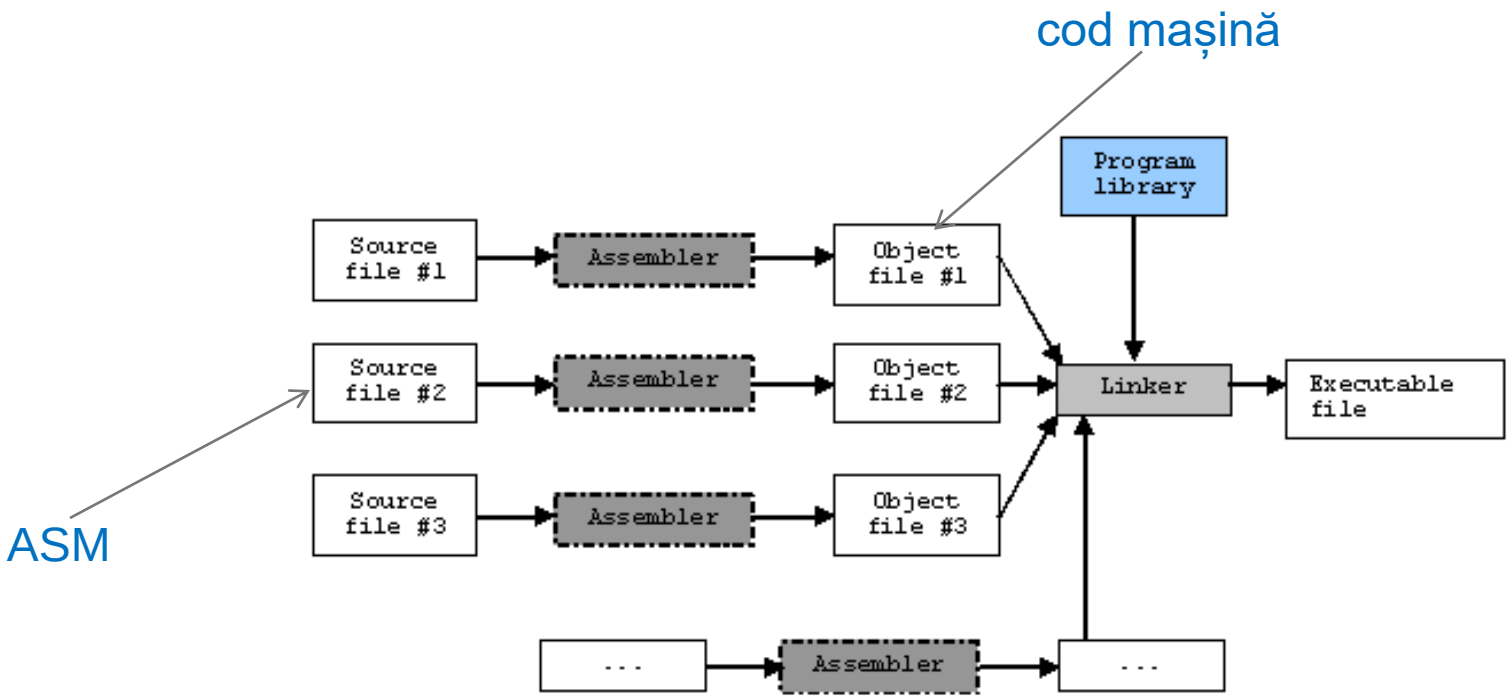
Memoria Principală

Memoria de Stocare

un astfel de sistem poate executa doar *cod mașină*

DE LA COD SURSĂ LA EXECUȚIE

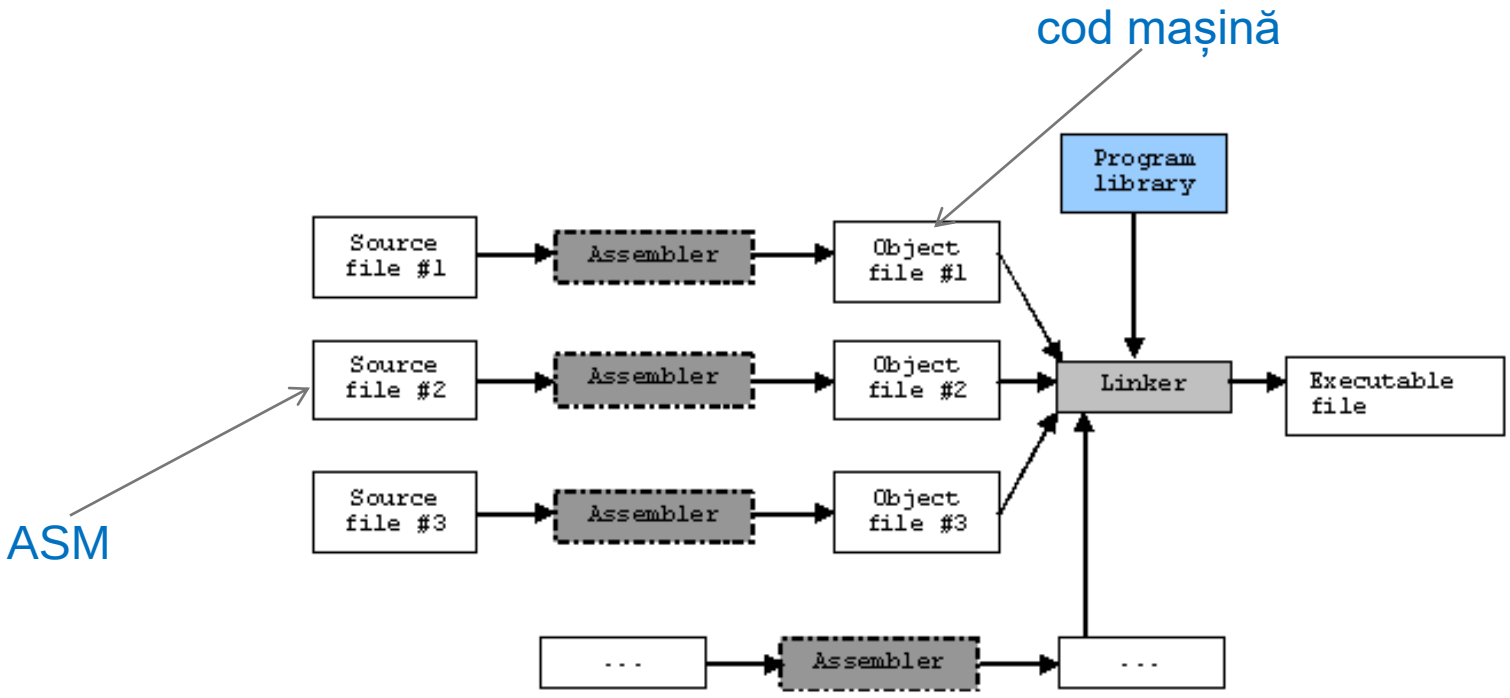
- **cod mașină (machine code)**
  - instrucțiuni binare executate direct de CPU
  - CPU poate executa doar cod mașină (orice altceva e tradus în CM)
  - cum obține cod mașină?
    - din cod sursă
    - codul sursă este generic
    - codul mașină e specific pentru Assembler, CPU, OS



https://en.wikipedia.org/wiki/C_standard_library

DE LA COD SURSĂ LA EXECUȚIE

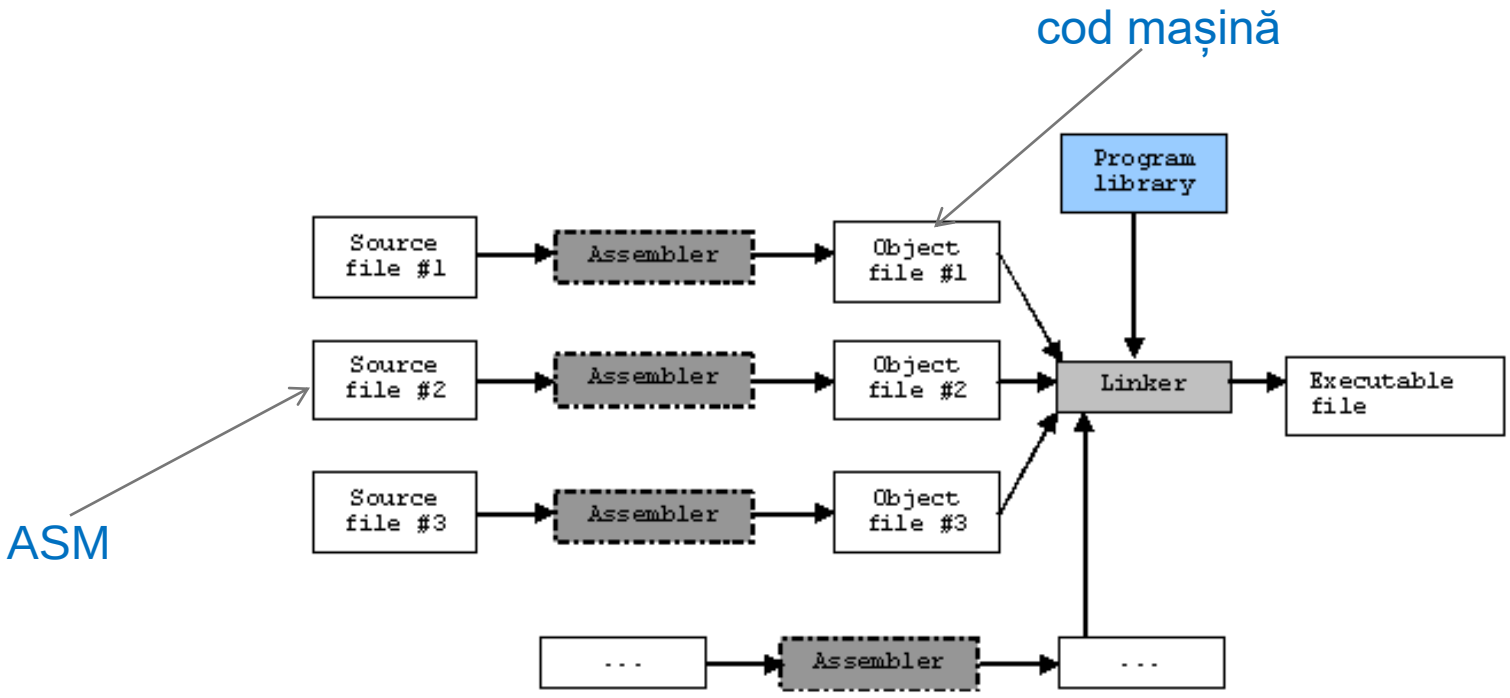
- **cod mașină (machine code)**
  - la laborator, primul vostru program ASM a fost:
    - as --32 program_exit.asm -o program_exit.o
    - ld -m elf_i386 program_exit.o -o program_exit
    - ./program_exit



https://www.tenouk.com/ModuleW.html

DE LA COD SURSĂ LA EXECUȚIE

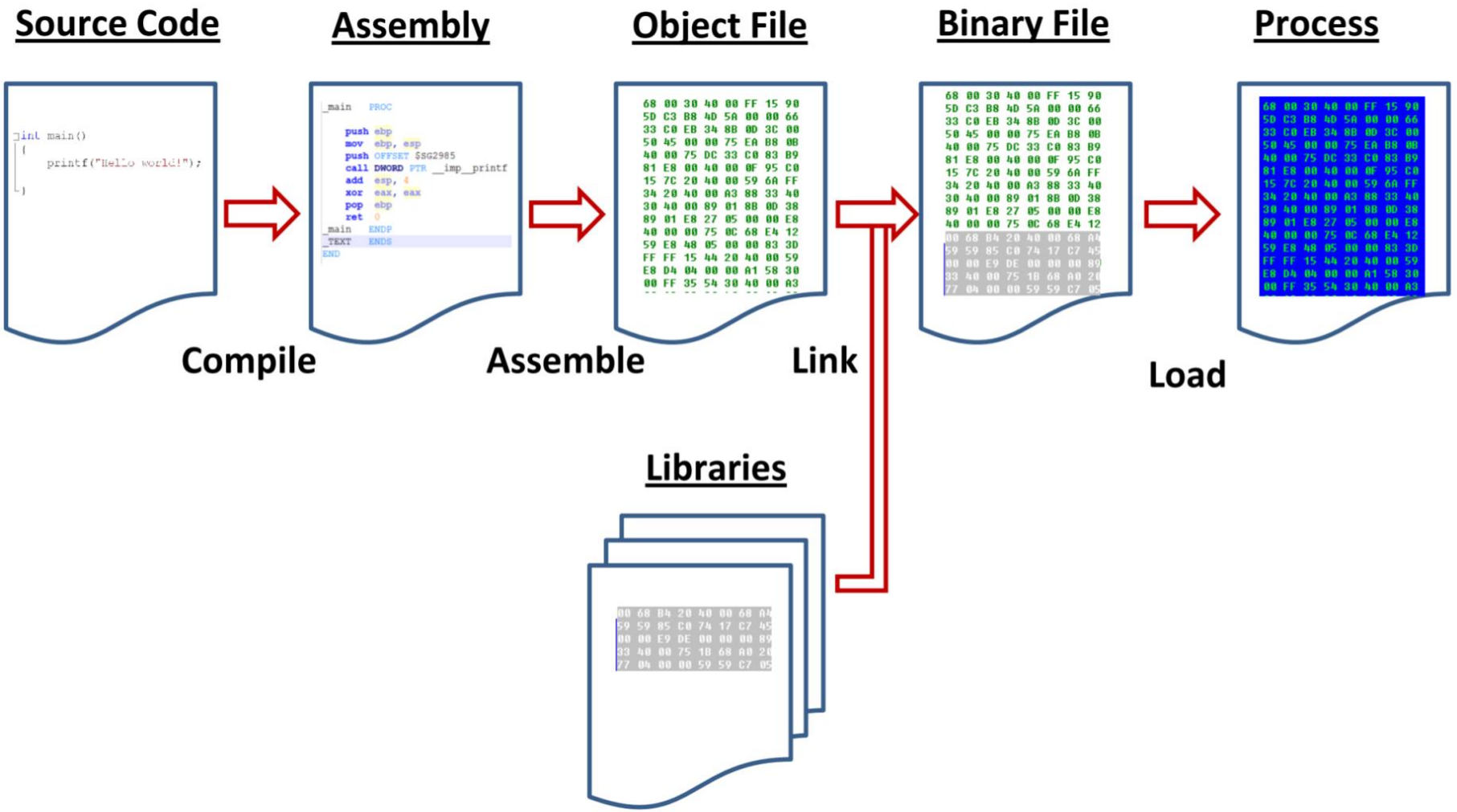
- **cod mașină (machine code)**
  - la laborator, pentru programele ASM unde ați folosit scanf/printf:
    - as --32 matrix.asm -o matrix.o
    - gcc –m32 matrix.o -o matrix
    - ./matrix



https://www.tenouk.com/ModuleW.html

DE LA COD SURSĂ LA EXECUȚIE

- în general (nu doar pentru Assembly)



https://slideplayer.com/slide/4695781/

DE LA COD SURSĂ LA EXECUȚIE

cod sursă: main.c

```
#include <stdio.h>

int main()
{
        printf("hello\n");
        return 42;
}
```

gcc -S -o main.asm main.c

cod sursă, assembly main.s

```
.LC0:
        .string "hello"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        leaq    .LC0(%rip), %rdi
        call    puts@PLT
        movl    $42, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

gcc -o main main.c

cod mașină, main (hexdump)

```
0000000 457f 464c 0102 0001 0000 0000 0000 0000
0000010 0003 003e 0001 0000 1060 0000 0000 0000
0000020 0040 0000 0000 0000 3978 0000 0000 0000
0000030 0000 0000 0040 0038 000d 0040 001f 001e
0000040 0006 0000 0004 0000 0040 0000 0000 0000
0000050 0040 0000 0000 0000 0040 0000 0000 0000
0000060 02d8 0000 0000 0000 02d8 0000 0000 0000
```

.

DE LA COD SURSĂ LA EXECUȚIE

- **objdump main**

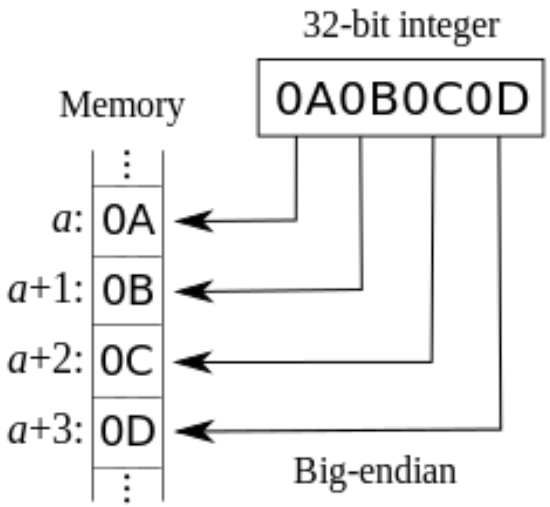
```
0000000000001149 <main>:
    1149:       f3 0f 1e fa             endbr64
    114d:       55                      push   %rbp
    114e:       48 89 e5                mov    %rsp,%rbp
    1151:       48 8d 3d ac 0e 00 00    lea    0xeac(%rip),%rdi        # 2004 <_IO_stdin_used+0x4>
    1158:       e8 f3 fe ff ff          callq  1050 <puts@plt>
    115d:       b8 2a 00 00 00          mov    $0x2a,%eax
    1162:       5d                      pop    %rbp
    1163:       c3                      retq
    1164:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
    116b:       00 00 00
    116e:       66 90                   xchg   %ax,%ax
```

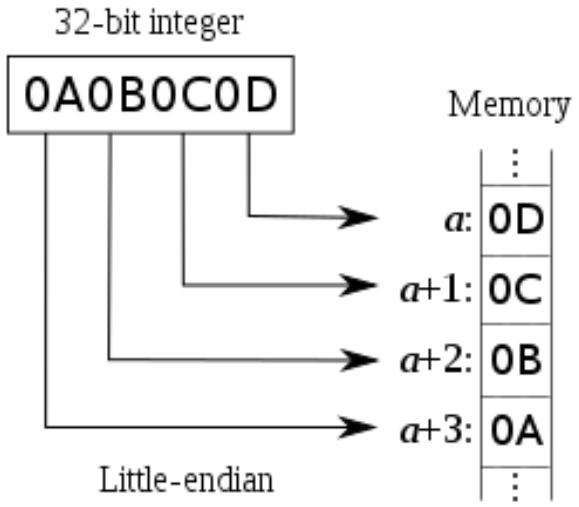
.

DE LA COD SURSĂ LA EXECUȚIE

- **objdump main**

```
0000000000001149 <main>:
    1149:       f3 0f 1e fa             endbr64
    114d:       55                      push   %rbp
    114e:       48 89 e5                mov    %rsp,%rbp
    1151:       48 8d 3d ac 0e 00 00    lea    0xeac(%rip),%rdi        # 2004 <_IO_stdin_used+0x4>
    1158:       e8 f3 fe ff ff          callq  1050 <puts@plt>
    115d:       b8 2a 00 00 00          mov    $0x2a,%eax
    1162:       5d                      pop    %rbp
    1163:       c3                      retq
    1164:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
    116b:       00 00 00
    116e:       66 90                   xchg   %ax,%ax
```





de asemenea, observați că instrucțiunile nu sunt codate cu aceeași lungime

https://en.wikipedia.org/wiki/Endianness

ARHITECTURA SETULUI DE INSTRUCȚIUNI

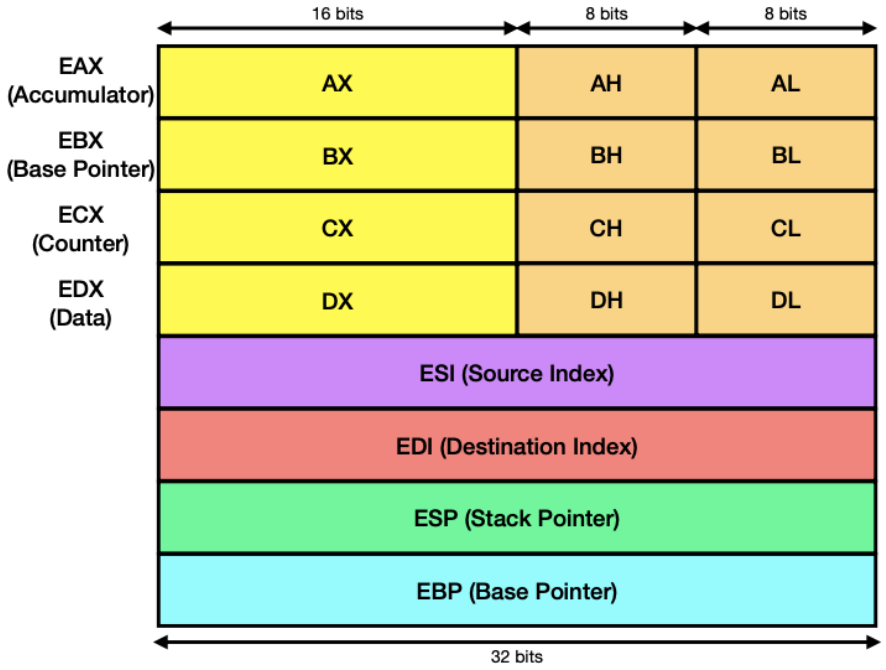
- **Instruction Set Arhitecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - regiștri
    - instrucțiuni
    - tipuri de date
    - metode de adresare a memoriei

.

ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Arhitecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - **regiștri**
    - instrucțiuni
    - tipuri de date
    - metode de adresare a memoriei

4 bits = 1 nibble
8 bits = 1 byte
16 bits = 1 word
32 bits = 1 dword
64 bits = 1 qword



FLAGS

Instruction Pointer (IP): următoarea instrucțiune care trebuie executată

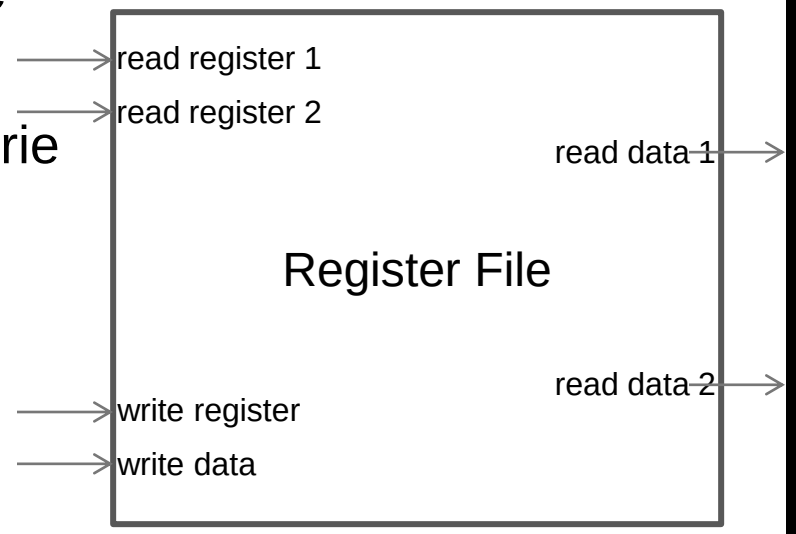
Stack Pointer (ESP): adresa stivei

YMM (pentru AVX) / **XMM** (pentru SSE): regiștrii pentru operații pe vectori

ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Arhitecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - **regiștri**
    - instrucțiuni
    - tipuri de date
    - metode de adresare a memoriei

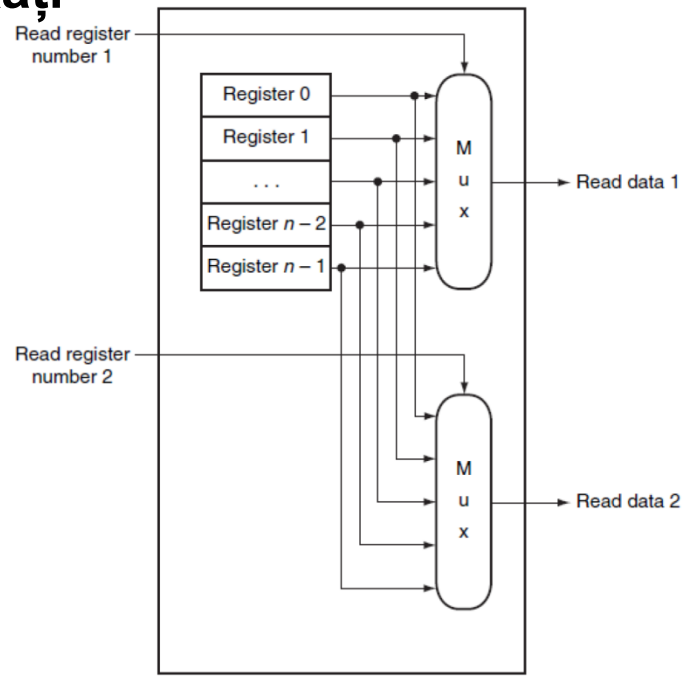
- **în general, regiștrii sunt grupați și indexați**
  - read register 1 / 2: indecșii de citire
  - read data 1 / 2: datele citite
  - write register: indexul în care se scrie
  - write data: datele care se scriu



ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Arhitecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - **regiștri**
    - instrucțiuni
    - tipuri de date
    - metode de adresare a memoriei

- **în general, regiștrii sunt grupați și indexați**
  - **read register 1 / 2: indecșii de citire**
  - **read data 1 / 2: datele citite**
  - write register: indexul în care se scrie
  - write data: datele care se scriu

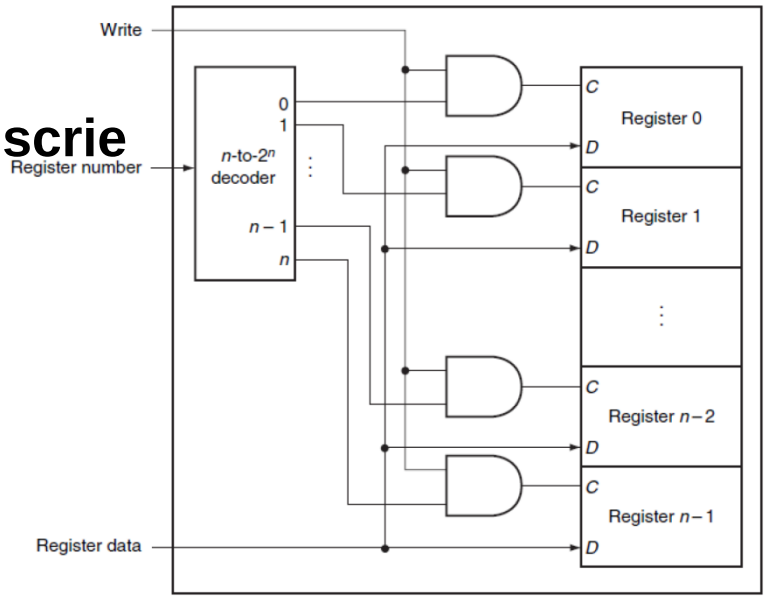


PH book, Apendix B.8

ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Arhitecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - **regiștri**
    - instrucțiuni
    - tipuri de date
    - metode de adresare a memoriei

- **în general, regiștrii sunt grupați și indexați**
  - read register 1 / 2: indecșii de citire
  - read data 1 / 2: datele citite
  - **write register: indexul în care se scrie**
  - **write data: datele care se scriu**



PH book, Apendix B.8

ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Arhitecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - regiștri
    - **instrucțiuni**
    - tipuri de date
    - metode de adresare a memoriei
  - <opcode> <listă operanzi>
    - add op1, op2 (op2 ← op2 + op1)
    - **Categorii de instrucțiuni**
      - **transferul datelor**: mov, cmov, movq, movs, movz, push, pop
      - **aritmetică și logică**: add, sub, mul, imul, div, idiv, sal, sar, shl, shr, and, or, not, xor, test, cmp
      - **controlul programului**: call, ret, j*

.

ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Arhitecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - regiștri
    - instrucțiuni
    - **tipuri de date**
    - metode de adresare a memoriei

C declaration	Intel data type	GAS suffix	x86-64 Size (Bytes)
`char`	Byte	`b`	1
`short`	Word	`w`	2
`int`	Double word	`l`	4
`unsigned`	Double word	`l`	4
`long int`	Quad word	`q`	8
`unsigned long`	Quad word	`q`	8
`char *`	Quad word	`q`	8
`float`	Single precision	`s`	4
`double`	Double precision	`d`	8
`long double`	Extended precision	`t`	16

https://slideplayer.com/slide/9507029/

ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Arhitecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - regiștri
    - instrucțiuni
    - tipuri de date
    - **metode de adresare a memoriei**

  - **adresare imediată**:
    - imediat: mov $172, %rdi
    - cu registru: mov %rcx, %rdi
    - cu memorie: mov 0x172, %rdi

  - **adresare indirectă**
    - indirect prin registru: mov (%rax), %rdi
    - indirect indexat: mov 172(%rax), %rdi
    - indirect bazat pe IP: mov 172(%rip), %rdi

  - **cazul cel mai general**: mov 172(%rdi, %rdx, 8), %rax
    - Base + Index*Scale + Displacement
    - îl aveți explicat detaliat în suportul de laborator

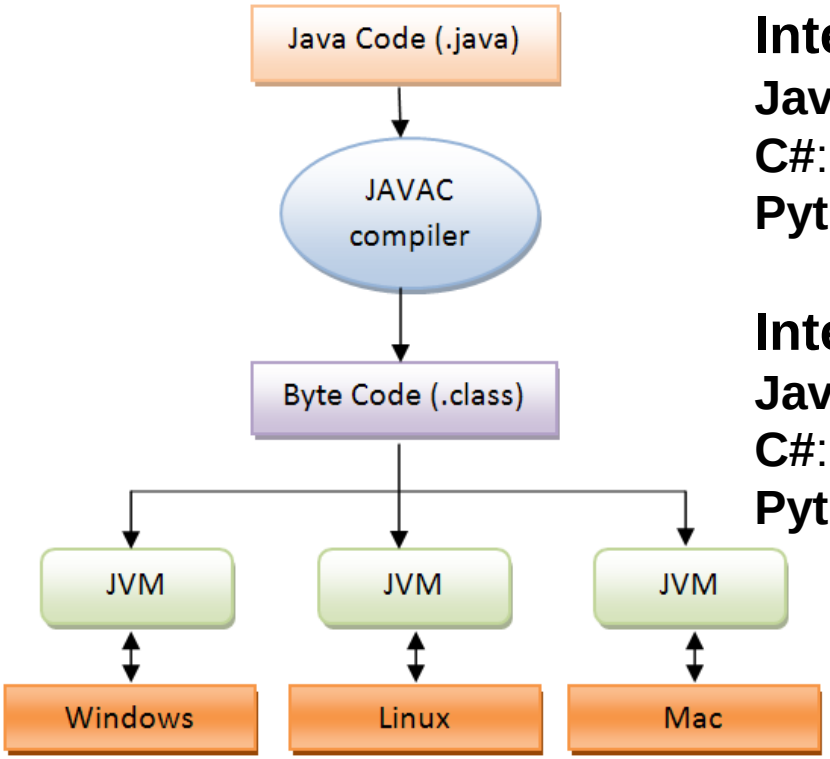
ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **câteva exemple în Assembly**
  - mov
  - push
  - call
  - cmp
  - add
  - pop
  - lea
  - test
  - je
  - xor
  - jmp
  - jne
  - ret
  - inc/sub

https://en.wikipedia.org/wiki/X86_instruction_listings

DE LA COD SURSĂ LA EXECUȚIE

- **excepție de la regulă**
  - bytecode (cod interpretat): instrucțiunile sunt executate de un interpretor care apoi le trimite la CPU

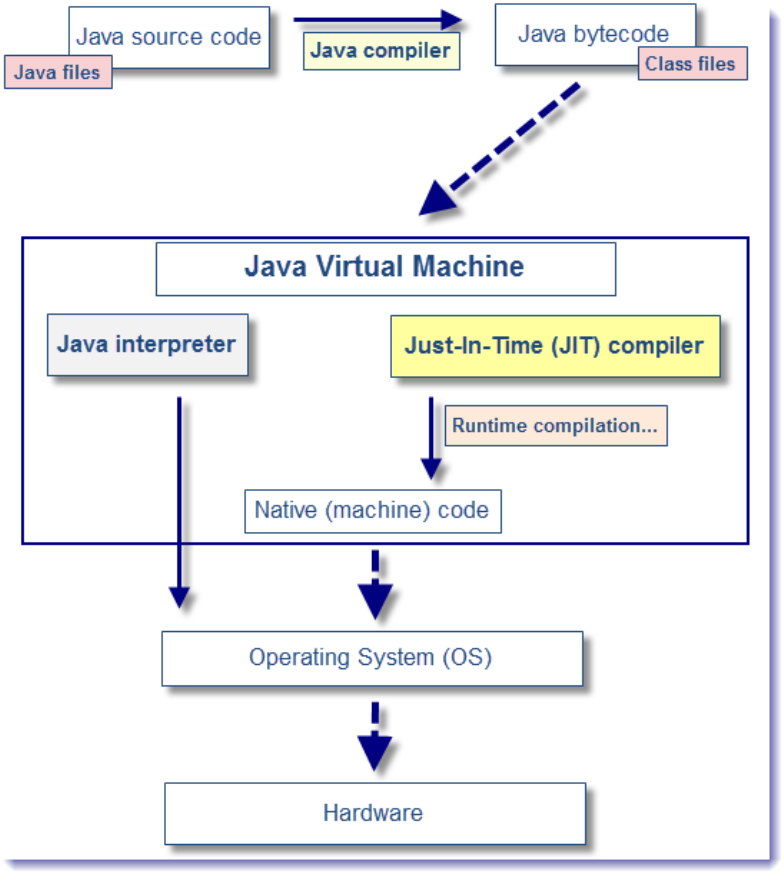


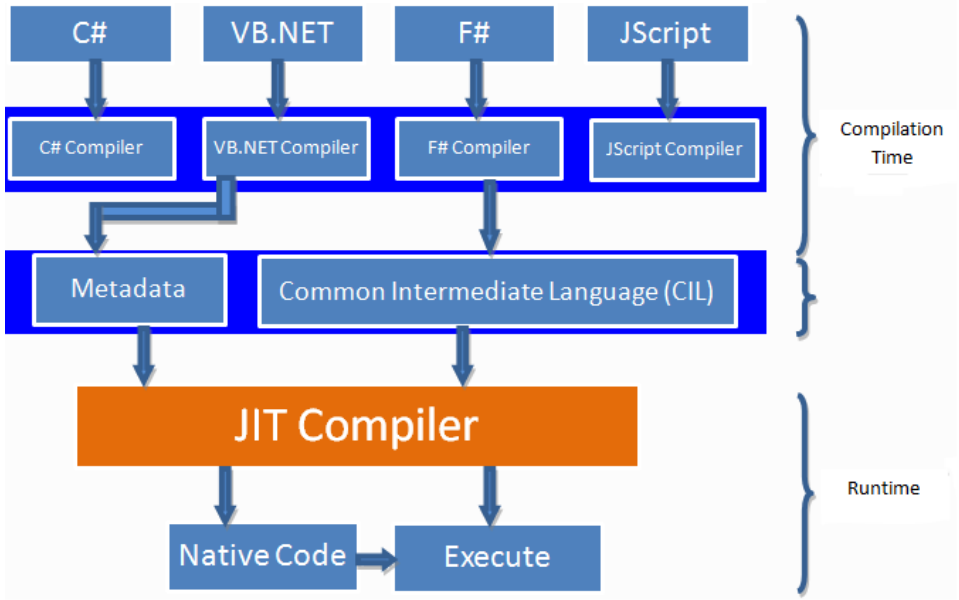
Interpreted code:
Java: java byte-code
C#: Common Intermediate Language (CIL)
Python: python byte-code (fișiere .pyc)

Interpreter:
Java: Java VM
C#: Common Language Runtime (CLR) în .NET
Python: python Virtual Machine

DE LA COD SURSĂ LA EXECUȚIE

- **excepție de la regulă**
  - bytecode (cod interpretat): instrucțiunile sunt executate de un interpretor care apoi le trimite la CPU
  - totul e lent pentru că mai este un pas de procesare
  - JIT compilation (Just-In-Time compilation) ajută





UN EXEMPLU

- următorul program simplu verifică o cheie de licență

```c
#include <string.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
        if(argc==2) {
                printf("Checking License: %s\n", argv[1]);
                if(strcmp(argv[1], "AAAA-Z10N-42-OK")==0) {
                        printf("Access Granted!\n");
                } else {
                        printf("WRONG!\n");
                }
        } else {
                printf("Usage: <key>\n");
        }
        return 0;
}
```

UN EXEMPLU

- **gdb checklicense**

verifică dacă ceva este egal cu 2

call la strcmp
apoi jne

din nou call la puts
avem asta în cod?

```
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000000740 <+0>:     push   rbp
   0x0000000000000741 <+1>:     mov    rbp,rsp
   0x0000000000000744 <+4>:     sub    rsp,0x10
   0x0000000000000748 <+8>:     mov    DWORD PTR [rbp-0x4],edi
   0x000000000000074b <+11>:    mov    QWORD PTR [rbp-0x10],rsi
   0x000000000000074f <+15>:    cmp    DWORD PTR [rbp-0x4],0x2
   0x0000000000000753 <+19>:    jne    0x7ae <main+110>
   0x0000000000000755 <+21>:    mov    rax,QWORD PTR [rbp-0x10]
   0x0000000000000759 <+25>:    add    rax,0x8
   0x000000000000075d <+29>:    mov    rax,QWORD PTR [rax]
   0x0000000000000760 <+32>:    mov    rsi,rax
   0x0000000000000763 <+35>:    lea    rdi,[rip+0xea]        # 0x854
   0x000000000000076a <+42>:    mov    eax,0x0
   0x000000000000076f <+47>:    call   0x5e0 <printf@plt>
   0x0000000000000774 <+52>:    mov    rax,QWORD PTR [rbp-0x10]
   0x0000000000000778 <+56>:    add    rax,0x8
   0x000000000000077c <+60>:    mov    rax,QWORD PTR [rax]
   0x000000000000077f <+63>:    lea    rsi,[rip+0xec]        # 0x872
   0x0000000000000786 <+70>:    mov    rdi,rax
   0x0000000000000789 <+73>:    call   0x5f0 <strcmp@plt>
   0x000000000000078e <+78>:    test   eax,eax
   0x0000000000000790 <+80>:    jne    0x7a0 <main+96>
   0x0000000000000792 <+82>:    lea    rdi,[rip+0xe2]        # 0x87b
   0x0000000000000799 <+89>:    call   0x5d0 <puts@plt>
   0x000000000000079e <+94>:    jmp    0x7ba <main+122>
   0x00000000000007a0 <+96>:    lea    rdi,[rip+0xe4]        # 0x88b
   0x00000000000007a7 <+103>:   call   0x5d0 <puts@plt>
   0x00000000000007ac <+108>:   jmp    0x7ba <main+122>
   0x00000000000007ae <+110>:   lea    rdi,[rip+0xe4]        # 0x899
   0x00000000000007b5 <+117>:   call   0x5d0 <puts@plt>
   0x00000000000007ba <+122>:   mov    eax,0x0
   0x00000000000007bf <+127>:   leave
   0x00000000000007c0 <+128>:   ret
End of assembler dump.
(gdb)
```

UN EXEMPLU

- **gdb checklicense**

```c
#include <string.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
        if(argc==2) {
                printf("Checking License: %s\n", argv[1]);
                if(strcmp(argv[1], "AAAA-Z10N-42-OK")==0) {
                        printf("Access Granted!\n");
                } else {
                        printf("WRONG!\n");
                }
        } else {
                printf("Usage: <key>\n");
        }
        return 0;
}
```

```
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000000740 <+0>:     push   rbp
   0x0000000000000741 <+1>:     mov    rbp,rsp
   0x0000000000000744 <+4>:     sub    rsp,0x10
   0x0000000000000748 <+8>:     mov    DWORD PTR [rbp-0x4],edi
   0x000000000000074b <+11>:    mov    QWORD PTR [rbp-0x10],rsi
   0x000000000000074f <+15>:    cmp    DWORD PTR [rbp-0x4],0x2
   0x0000000000000753 <+19>:    jne    0x7ae <main+110>
   0x0000000000000755 <+21>:    mov    rax,QWORD PTR [rbp-0x10]
   0x0000000000000759 <+25>:    add    rax,0x8
   0x000000000000075d <+29>:    mov    rax,QWORD PTR [rax]
   0x0000000000000760 <+32>:    mov    rsi,rax
   0x0000000000000763 <+35>:    lea    rdi,[rip+0xea]        # 0x854
   0x000000000000076a <+42>:    mov    eax,0x0
   0x000000000000076f <+47>:    call   0x5e0 <printf@plt>
   0x0000000000000774 <+52>:    mov    rax,QWORD PTR [rbp-0x10]
   0x0000000000000778 <+56>:    add    rax,0x8
   0x000000000000077c <+60>:    mov    rax,QWORD PTR [rax]
   0x000000000000077f <+63>:    lea    rsi,[rip+0xec]        # 0x872
   0x0000000000000786 <+70>:    mov    rdi,rax
   0x0000000000000789 <+73>:    call   0x5f0 <strcmp@plt>
   0x000000000000078e <+78>:    test   eax,eax
   0x0000000000000790 <+80>:    jne    0x7a0 <main+96>
   0x0000000000000792 <+82>:    lea    rdi,[rip+0xe2]        # 0x87b
   0x0000000000000799 <+89>:    call   0x5d0 <puts@plt>
   0x000000000000079e <+94>:    jmp    0x7ba <main+122>
   0x00000000000007a0 <+96>:    lea    rdi,[rip+0xe4]        # 0x88b
   0x00000000000007a7 <+103>:   call   0x5d0 <puts@plt>
   0x00000000000007ac <+108>:   jmp    0x7ba <main+122>
   0x00000000000007ae <+110>:   lea    rdi,[rip+0xe4]        # 0x899
   0x00000000000007b5 <+117>:   call   0x5d0 <puts@plt>
   0x00000000000007ba <+122>:   mov    eax,0x0
   0x00000000000007bf <+127>:   leave
   0x00000000000007c0 <+128>:   ret
End of assembler dump.
(gdb)
```

UN EXEMPLU

- **informațiile executabilului**
  - file checklicense
- **hex viewer**
  - hexdump –C checklicense
- **hex editor**
  - hexeditor checklicense
- **scoate toate string-urile din fișier**
  - strings checklicense
- **dump al obiectelor din fișier**
  - objdump –x checklicense
- **analiză binară avansată**
  - radare2 (r2)
  - ghidra

https://www.nsa.gov/resources/everyone/ghidra/

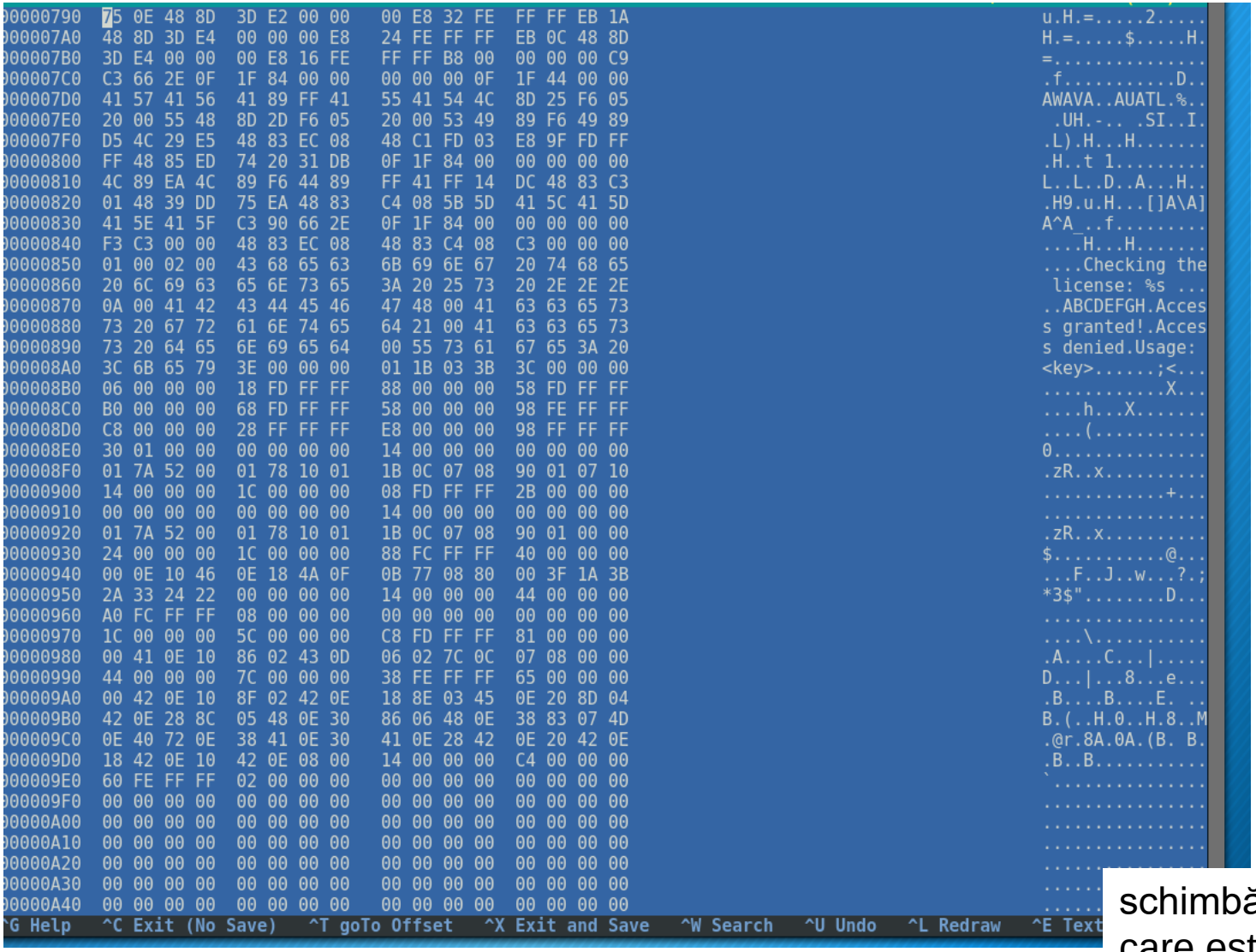
UN EXEMPLU

- **objdump –d checklicense**

```
0000000000000740 <main>:
 740:   55                      push   %rbp
 741:   48 89 e5                mov    %rsp,%rbp
 744:   48 83 ec 10             sub    $0x10,%rsp
 748:   89 7d fc                mov    %edi,-0x4(%rbp)
 74b:   48 89 75 f0             mov    %rsi,-0x10(%rbp)
 74f:   83 7d fc 02             cmpl   $0x2,-0x4(%rbp)
 753:   75 59                   jne    7ae <main+0x6e>
 755:   48 8b 45 f0             mov    -0x10(%rbp),%rax
 759:   48 83 c0 08             add    $0x8,%rax
 75d:   48 8b 00                mov    (%rax),%rax
 760:   48 89 c6                mov    %rax,%rsi
 763:   48 8d 3d ea 00 00 00    lea    0xea(%rip),%rdi        # 854 <_IO_stdin_used+0x4>
 76a:   b8 00 00 00 00          mov    $0x0,%eax
 76f:   e8 6c fe ff ff          callq  5e0 <printf@plt>
 774:   48 8b 45 f0             mov    -0x10(%rbp),%rax
 778:   48 83 c0 08             add    $0x8,%rax
 77c:   48 8b 00                mov    (%rax),%rax
 77f:   48 8d 35 ec 00 00 00    lea    0xec(%rip),%rsi        # 872 <_IO_stdin_used+0x22>
 786:   48 89 c7                mov    %rax,%rdi
 789:   e8 62 fe ff ff          callq  5f0 <strcmp@plt>
 78e:   85 c0                   test   %eax,%eax
 790:   75 0e                   jne    7a0 <main+0x60>
 792:   48 8d 3d e2 00 00 00    lea    0xe2(%rip),%rdi        # 87b <_IO_stdin_used+0x2b>
 799:   e8 32 fe ff ff          callq  5d0 <puts@plt>
 79e:   eb 1a                   jmp    7ba <main+0x7a>
 7a0:   48 8d 3d e4 00 00 00    lea    0xe4(%rip),%rdi        # 88b <_IO_stdin_used+0x3b>
 7a7:   e8 24 fe ff ff          callq  5d0 <puts@plt>
 7ac:   eb 0c                   jmp    7ba <main+0x7a>
 7ae:   48 8d 3d e4 00 00 00    lea    0xe4(%rip),%rdi        # 899 <_IO_stdin_used+0x49>
 7b5:   e8 16 fe ff ff          callq  5d0 <puts@plt>
 7ba:   b8 00 00 00 00          mov    $0x0,%eax
 7bf:   c9                      leaveq
 7c0:   c3                      retq
 7c1:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
 7c8:   00 00 00
 7cb:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
```

UN EXEMPLU

- **hexeditor checklicense**

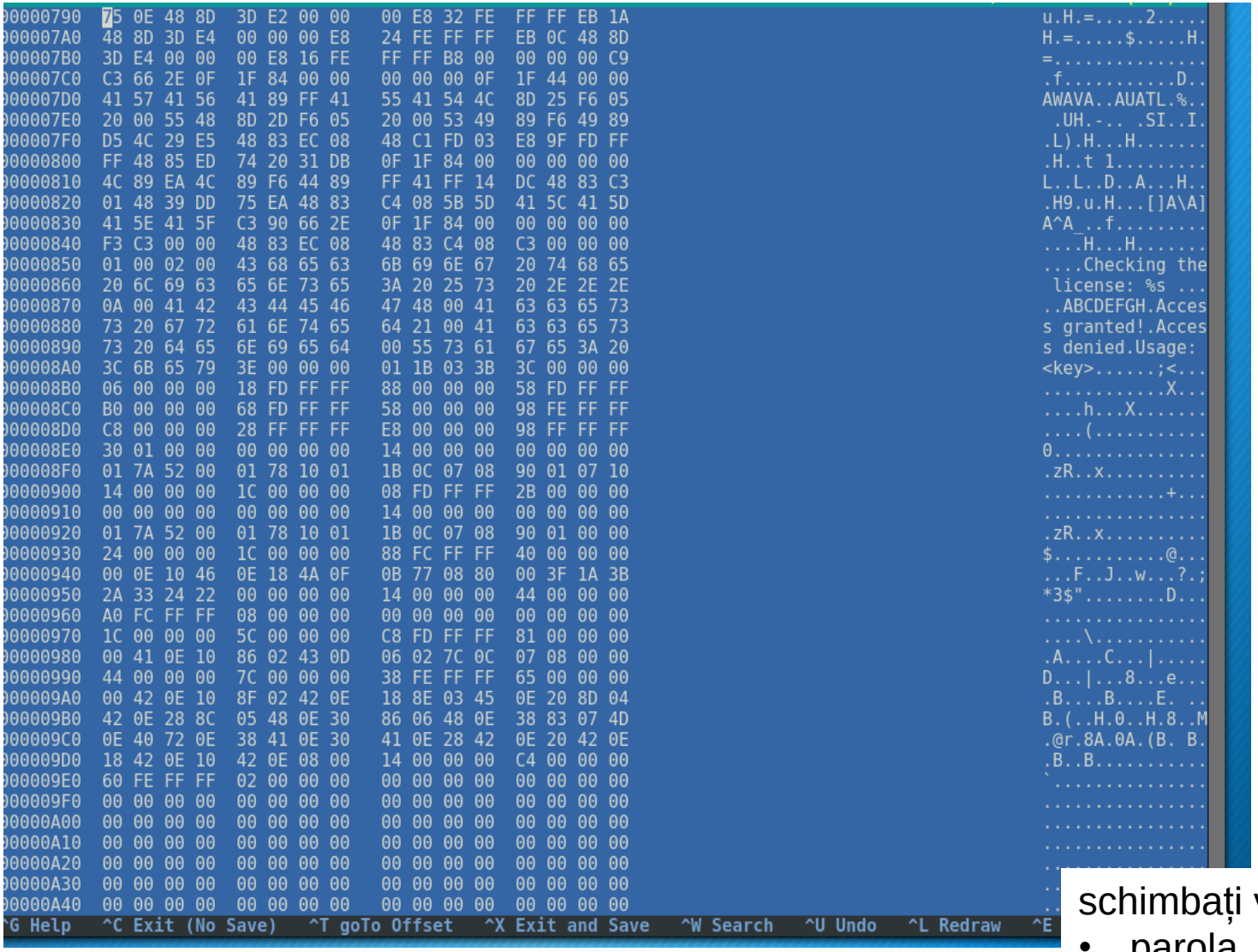


schimbăm JNE?
care este noul OPCODE
pentru noua instrucțiune?

Intel 64 and IA-32 Architectures Software Developer Manuals,
http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

UN EXEMPLU

- **hexeditor checklicense**



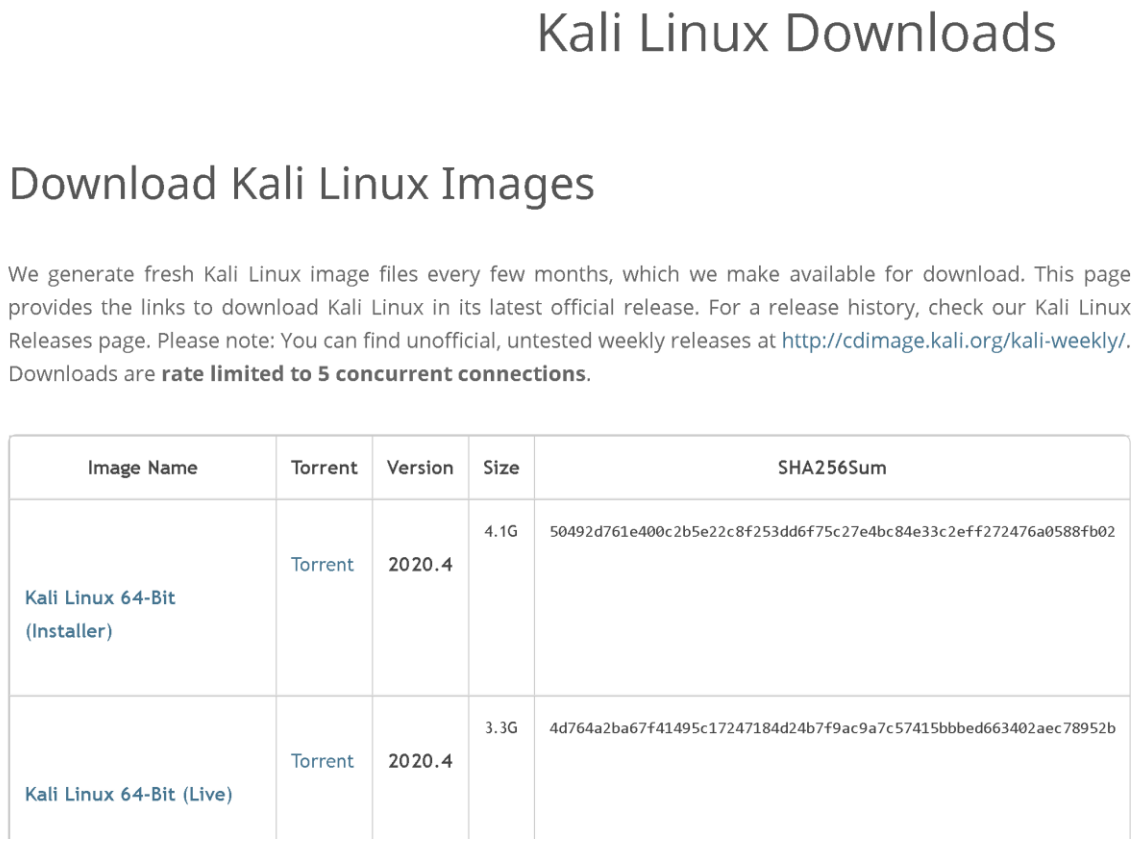
schimbați voi și alte lucruri:

- parola
- textul afișat

Intel 64 and IA-32 Architectures Software Developer Manuals,
http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

UN EXEMPLU

- **ce am făcut?**
  - am modificat, permanent, fișierul binar
  - cum ne putem da seama că un fișier a fost modificat?



Kali Linux Downloads

Download Kali Linux Images

We generate fresh Kali Linux image files every few months, which we make available for download. This page provides the links to download Kali Linux in its latest official release. For a release history, check our Kali Linux Releases page. Please note: You can find unofficial, untested weekly releases at http://cdimage.kali.org/kali-weekly/. Downloads are **rate limited to 5 concurrent connections**.

Image Name	Torrent	Version	Size	SHA256Sum
Kali Linux 64-Bit (Installer)	Torrent	2020.4	4.1G	50492d761e400c2b5e22c8f253dd6f75c27e4bc84e33c2eff272476a0588fb02
Kali Linux 64-Bit (Live)	Torrent	2020.4	3.3G	4d764a2ba67f41495c17247184d24b7f9ac9a7c57415bbbed663402aec78952b

https://www.kali.org/downloads/

EXECUȚIA DATELOR

- **fie următorul program foarte simplu (shellcode.c)**

```c
#include         <stdio.h>
#include         <stdlib.h>
#include         <unistd.h>
#include         <string.h>
#include         <errno.h>

int main()
{
  int e;
  char *argv[] = { "/bin/ls", "-l", NULL };

  e = execve("/bin/ls", argv, NULL);
  if (e == -1)
      fprintf(stderr, "Error: %s\n", strerror(errno));
  return 0;
}
```

EXECUȚIA DATELOR

- **același program în Assembly**

```
.text
.globl _start

_start:
        xor %eax,%eax
        push %eax
        push $0x68732f2f
        push $0x6e69622f
        mov %esp,%ebx
        push %eax
        push %ebx
        mov %esp,%ecx
        mov $0xb,%al
        int $0x80

        movl $1, %eax
        movl $0, %ebx
        int $0x80
```

```
root@kali:~# objdump -d shellcode

shellcode:     file format elf32-i386


Disassembly of section .text:

08048054 <_start>:
 8048054:       31 c0                   xor    %eax,%eax
 8048056:       50                      push   %eax
 8048057:       68 2f 2f 73 68          push   $0x68732f2f
 804805c:       68 2f 62 69 6e          push   $0x6e69622f
 8048061:       89 e3                   mov    %esp,%ebx
 8048063:       50                      push   %eax
 8048064:       53                      push   %ebx
 8048065:       89 e1                   mov    %esp,%ecx
 8048067:       b0 0b                   mov    $0xb,%al
 8048069:       cd 80                   int    $0x80
 804806b:       b8 01 00 00 00          mov    $0x1,%eax
 8048070:       bb 00 00 00 00          mov    $0x0,%ebx
 8048075:       cd 80                   int    $0x80
```

EXECUȚIA DATELOR

- **un program echivalent**

```
#include <stdio.h>
#include <string.h>

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
        fprintf(stdout,"Length: %d\n",strlen(shellcode));
        (*(void(*)()) shellcode)();
        return 0;
}
```

ce se întamplă aici?
afișați shellcode-ul pe ecran

EXECUȚIA DATELOR

- **un program echivalent**

```
#include <stdio.h>
#include <string.h>

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
        fprintf(stdout,"Length: %d\n",strlen(shellcode));
        (*(void(*)()) shellcode)();
        return 0;
}
```

aceste programe nu mai pot rula pe sisteme de operare moderne

- Data Execution Prevention (DEP) e activ

CE AM FĂCUT ASTĂZI

- **Instruction Set Architecture (ISA)**
- **de la cod sursă la cod mașină**
- **un exemplu simplu de *software cracking* și *shellcode execution***

.

DATA VIITOARE ...

- **acoperim concepte mai complexe care aduc performanță sporită**
  - pipelining
  - branch prediction
  - out of order execution
- **sisteme multi-procesor**
- **performanța calculatoarelor**

LECTURĂ SUPLIMENTARĂ

- **PH book**
  - 1.3 Below Your Program
  - 1.4 Under the Covers
  - 2.15 Advanced Material: Compiling C and Interpreting Java

.