

# **ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x07**

**DE LA COD SURSĂ LA EXECUȚIE**

Cristian Rusu

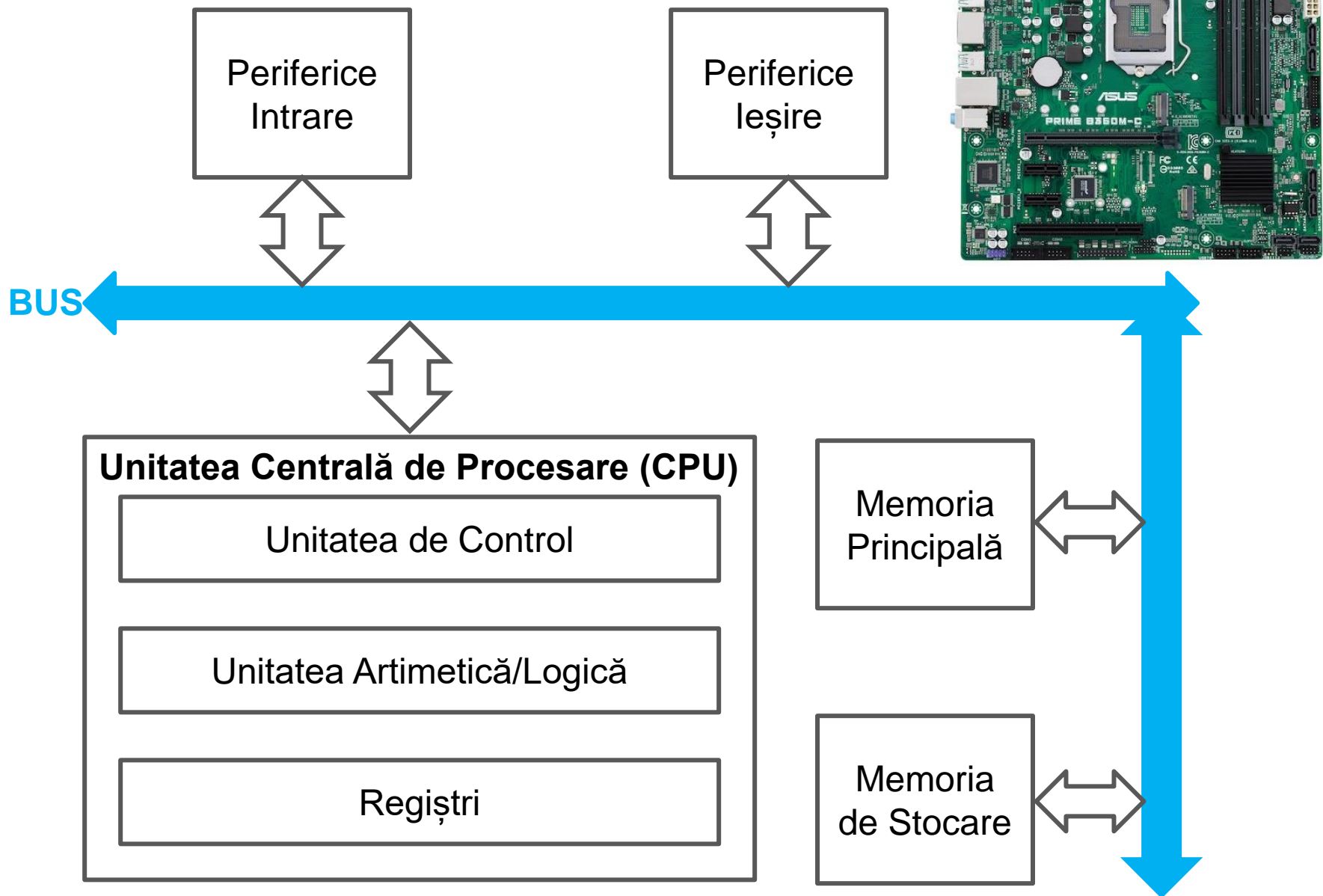
# DATA TRECUTĂ

- **arhitectura de bază a calculatoarelor**
- **una dintre concluzii: comunicarea, la fel de importantă ca și procesarea în sine**

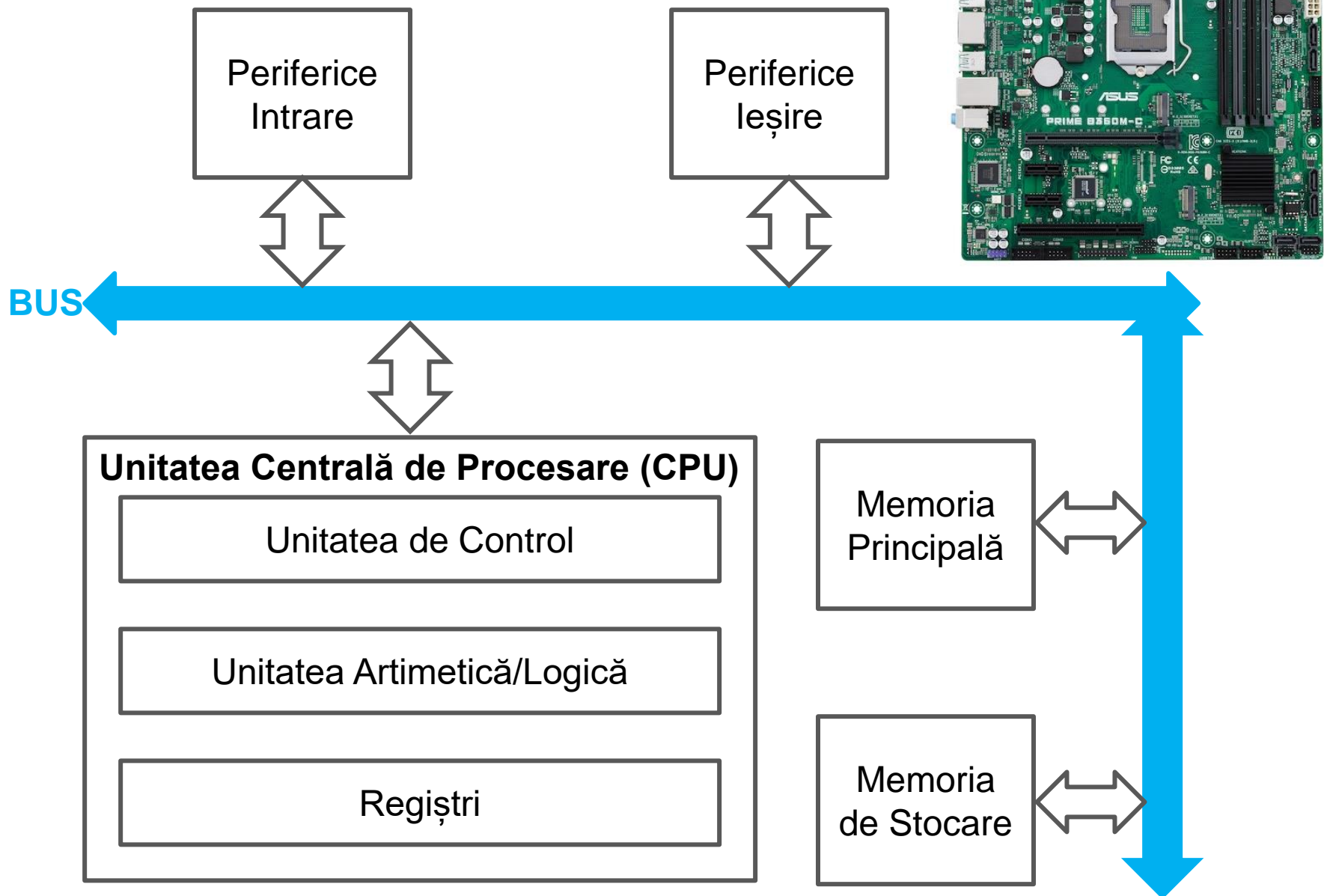
# CUPRINS

- **scurt review arhitectura de bază a calculatoarelor**
- **Instruction Set Architecture (ISA)**
- **de la cod sursă la cod mașină**
  - software cracking
  - executarea datelor

# ARHITECTURA DE BAZĂ

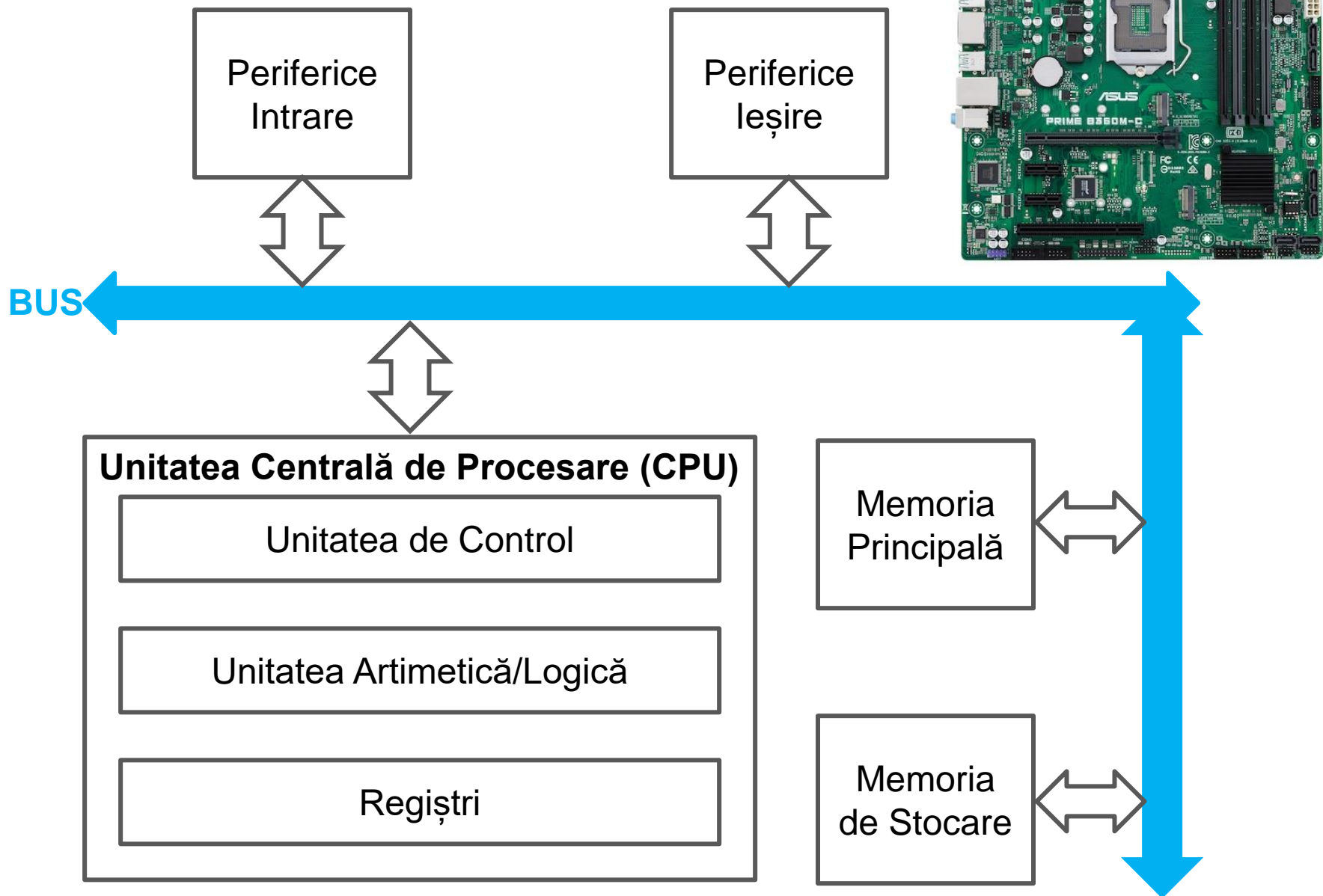


# ARHITECTURA DE BAZĂ



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

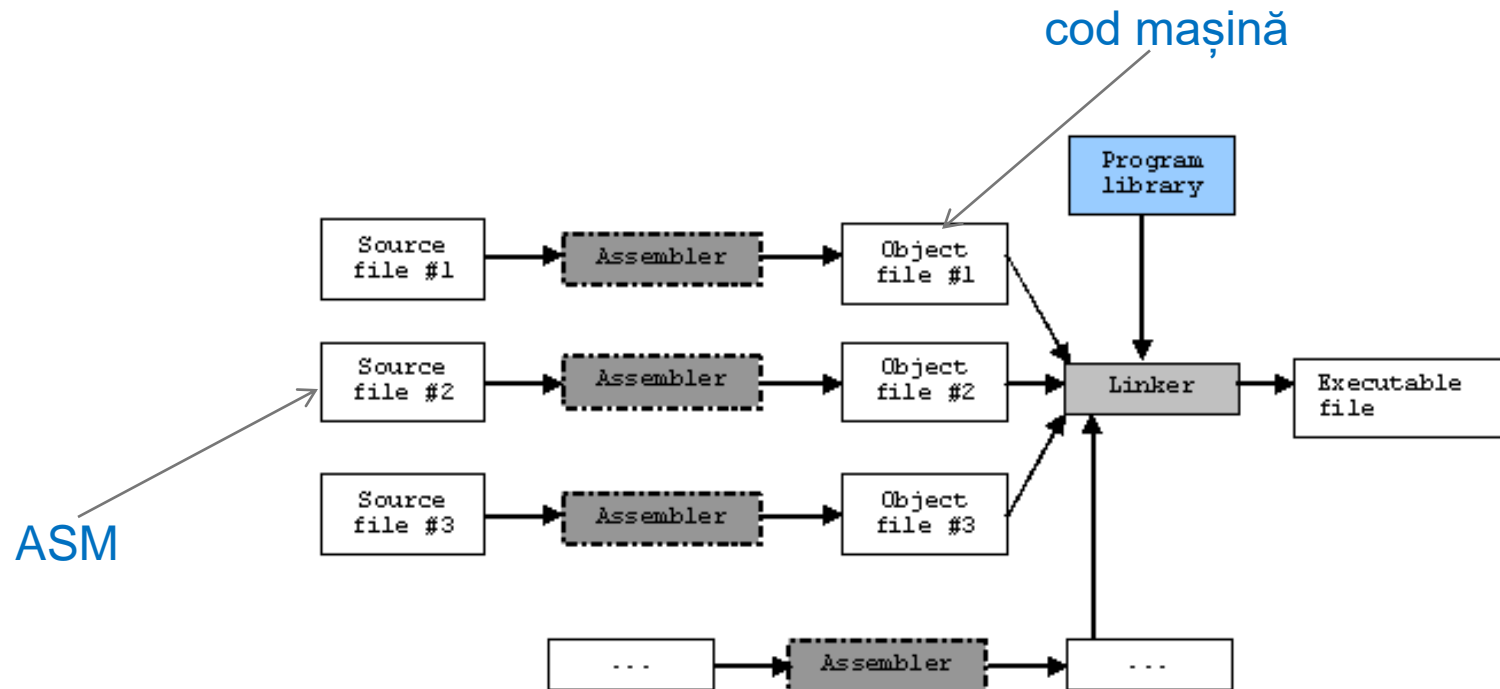
# ARHITECTURA DE BAZĂ



un astfel de sistem poate executa doar *cod mașină*

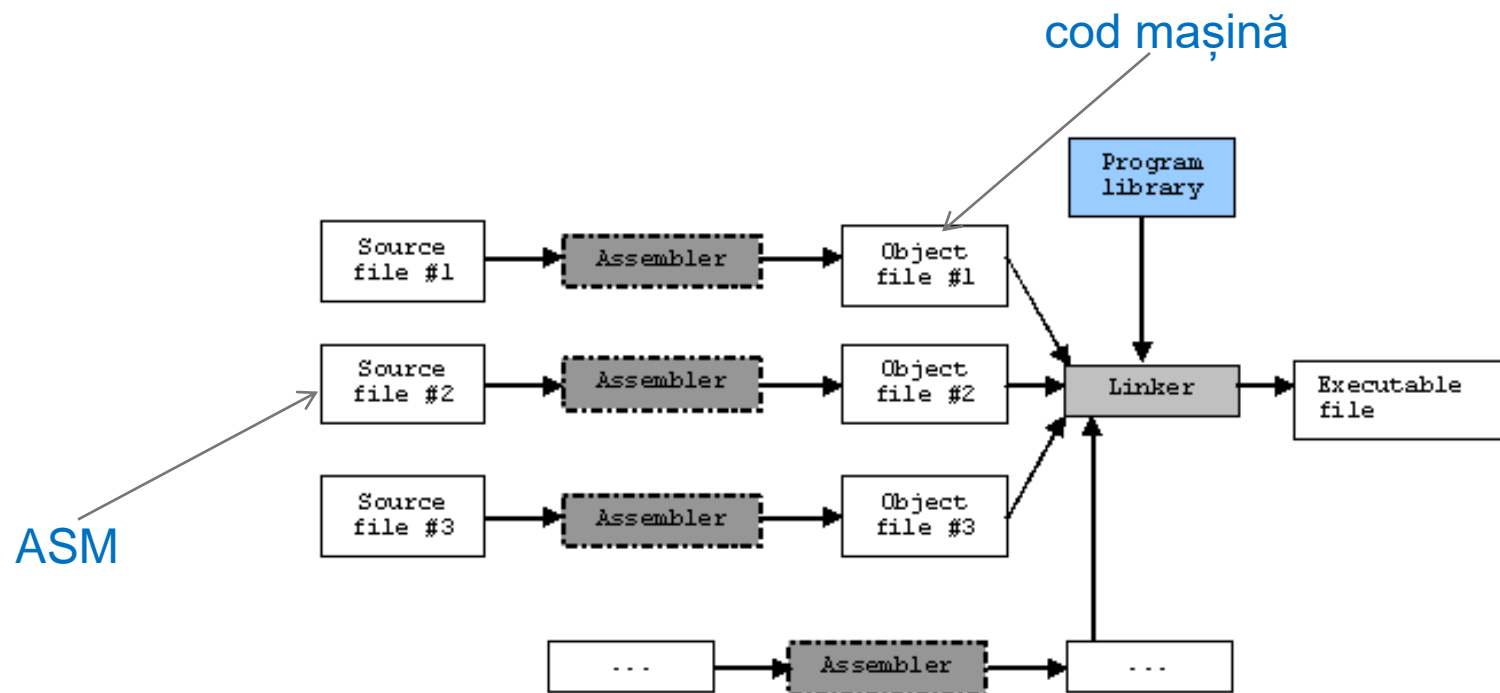
# DE LA COD SURSĂ LA EXECUȚIE

- **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



# DE LA COD SURSĂ LA EXECUȚIE

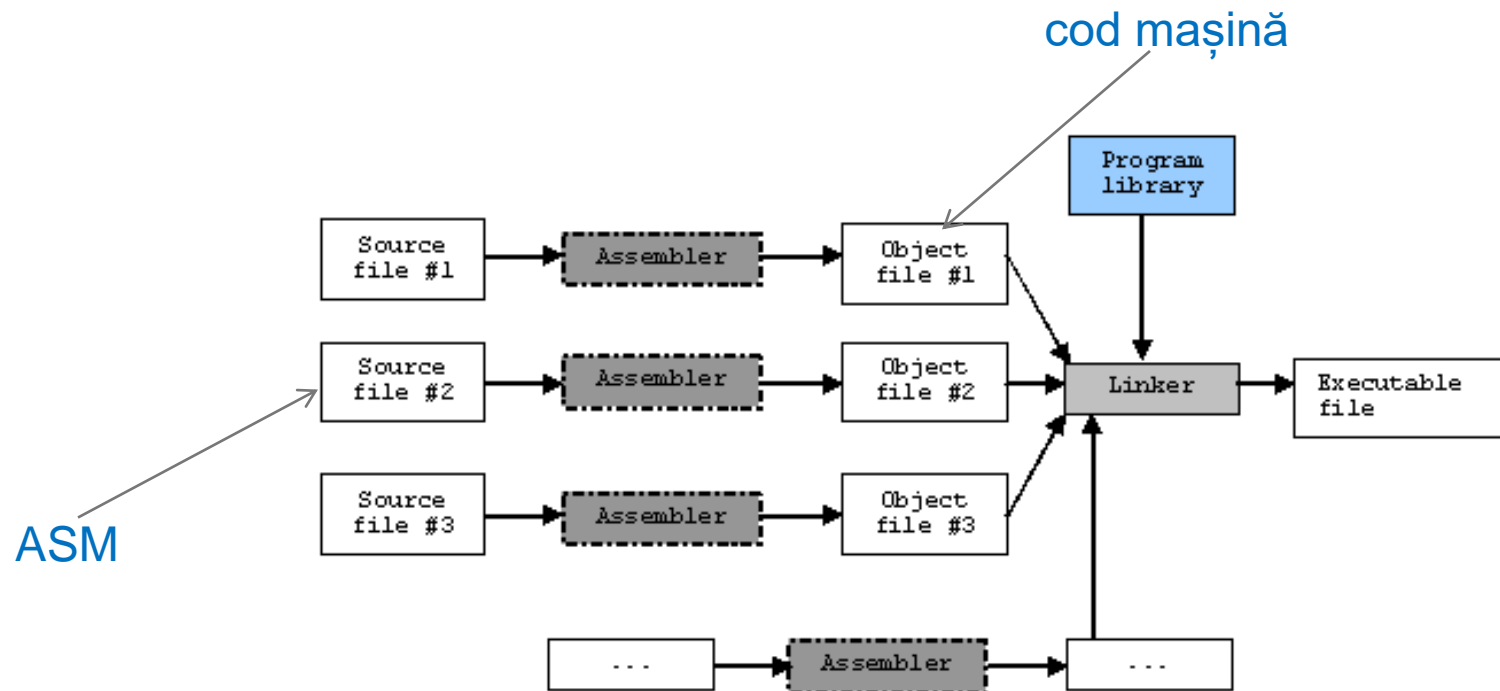
- **cod mașină (machine code)**
  - la laborator, primul vostru program ASM a fost:
    - `as --32 program_exit.asm -o program_exit.o`
    - `ld -m elf_i386 program_exit.o -o program_exit`
    - `./program_exit`





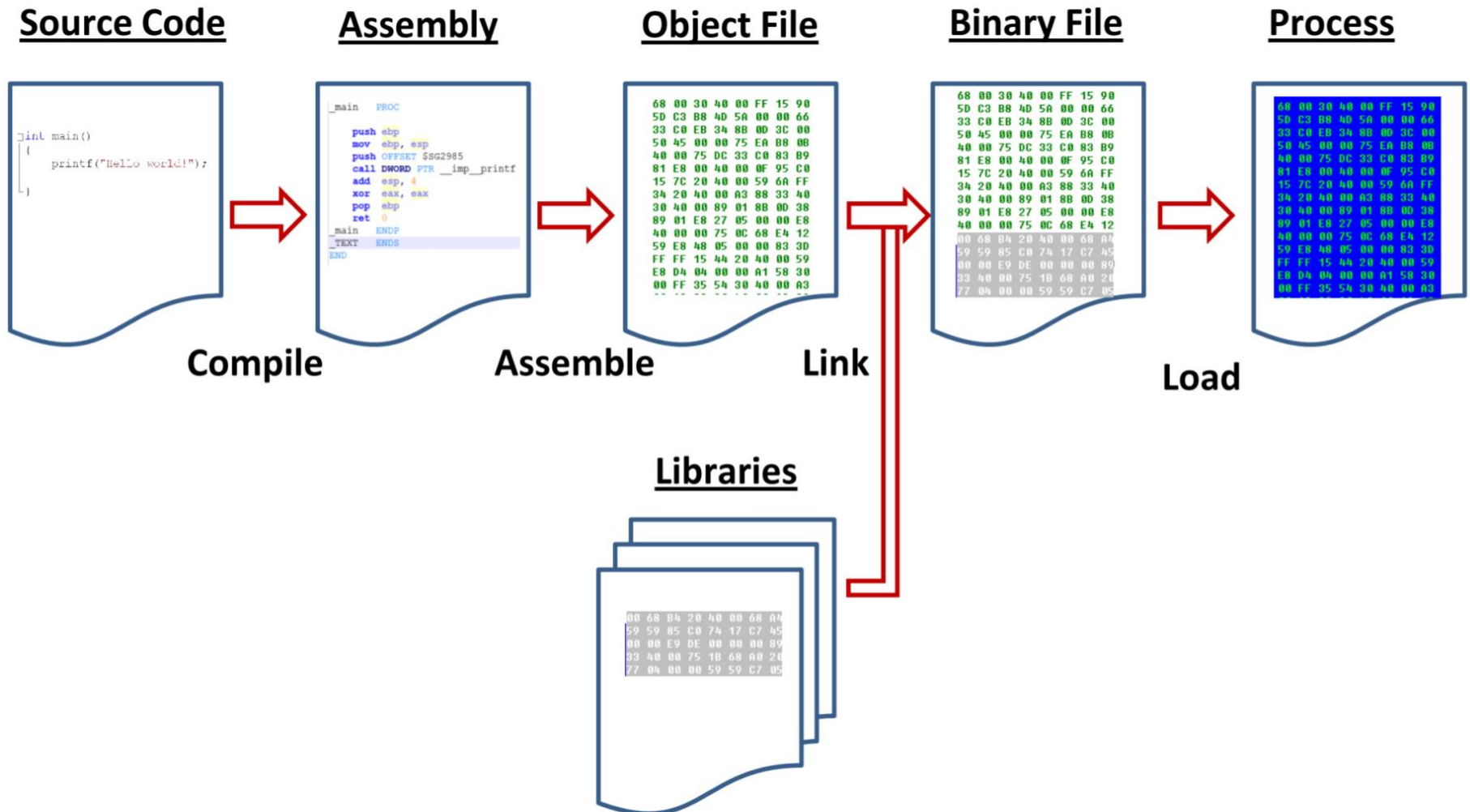
# DE LA COD SURSĂ LA EXECUȚIE

- **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`



# DE LA COD SURSĂ LA EXECUȚIE

- în general (nu doar pentru Assembly)



# DE LA COD SURSĂ LA EXECUȚIE

cod sursă: main.c

```
#include <stdio.h>

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

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 -S -o main.asm main.c

gcc -o main main.c

cod mașină, main (hexdump)

00000000	457f 464c 0102 0001 0000 0000 0000 0000
00000010	0003 003e 0001 0000 1060 0000 0000 0000
00000020	0040 0000 0000 0000 3978 0000 0000 0000
00000030	0000 0000 0040 0038 000d 0040 001f 001e
00000040	0006 0000 0004 0000 0040 0000 0000 0000
00000050	0040 0000 0000 0000 0040 0000 0000 0000
00000060	02d8 0000 0000 0000 02d8 0000 0000 0000

# DE LA COD SURSĂ LA EXECUȚIE

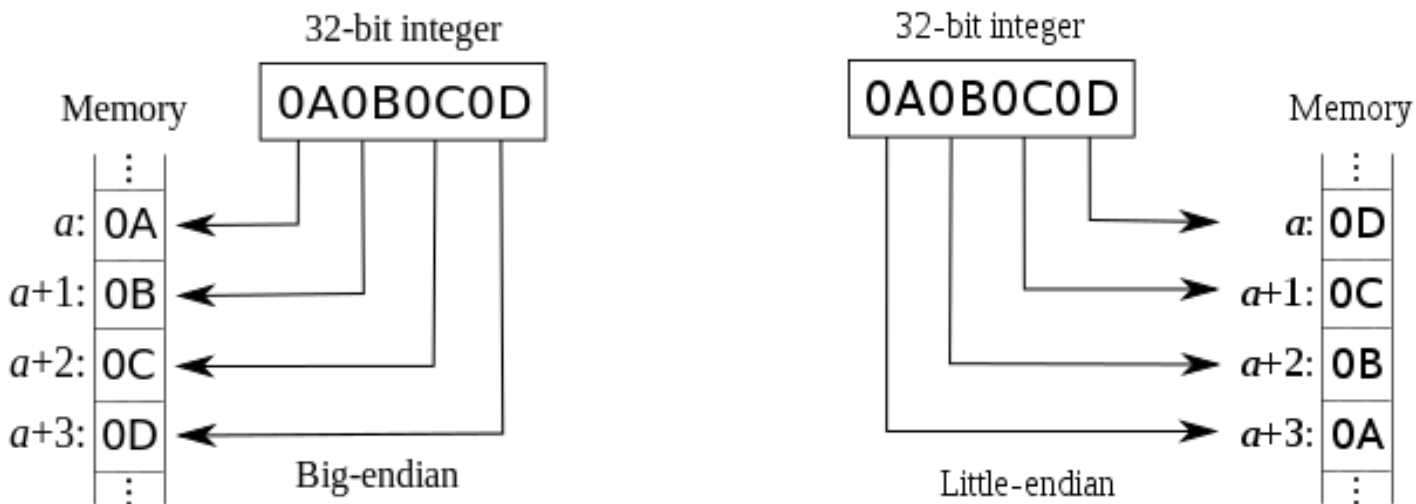
- objdump main

```
00000000000001149 <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

```
00000000000001149 <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

# ARHITECTURA SETULUI DE INSTRUCȚIUNI

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

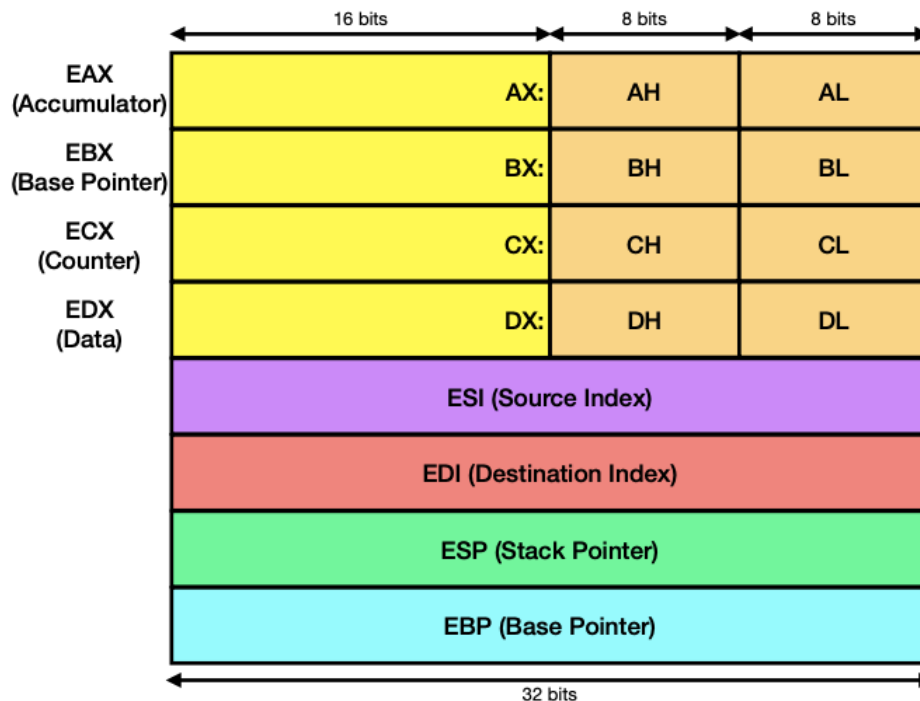
# ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Architecture (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**

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

Instruction Mnemonic	Condition (Flag States)	Description
<b>Unsigned Conditional Jumps</b>		
J A/J NBE	(CF or ZF) = 0	Above/not below or equal
J AE/J NB	CF = 0	Above or equal/not below
J B/J NAE	CF = 1	Below/not above or equal
J BE/J NA	(CF or ZF) = 1	Below or equal/not above
J C	CF = 1	Carry
J E/J Z	ZF = 1	Equal/zero
J NC	CF = 0	Not carry
J NE/J NZ	ZF = 0	Not equal/not zero
J NP/J PO	PF = 0	Not parity/parity odd
J P/J PE	PF = 1	Parity/parity even
J CXZ	CX = 0	Register CX is zero
J ECXZ	ECX = 0	Register ECX is zero
<b>Signed Conditional Jumps</b>		
J G/J NLE	((SF xor OF) or ZF) = 0	Greater/not less or equal
J GE/J NL	(SF xor OF) = 0	Greater or equal/not less
J L/J NGE	(SF xor OF) = 1	Less/not greater or equal
J LE/J NG	((SF xor OF) or ZF) = 1	Less or equal/not greater
J NO	OF = 0	Not overflow
J NS	SF = 0	Not sign (non-negative)
J O	OF = 1	Overflow
J S	SF = 1	Sign (negative)

**FLAGS**

**Instruction Pointer (IP):** următoarea instrucțiune care trebuie executată

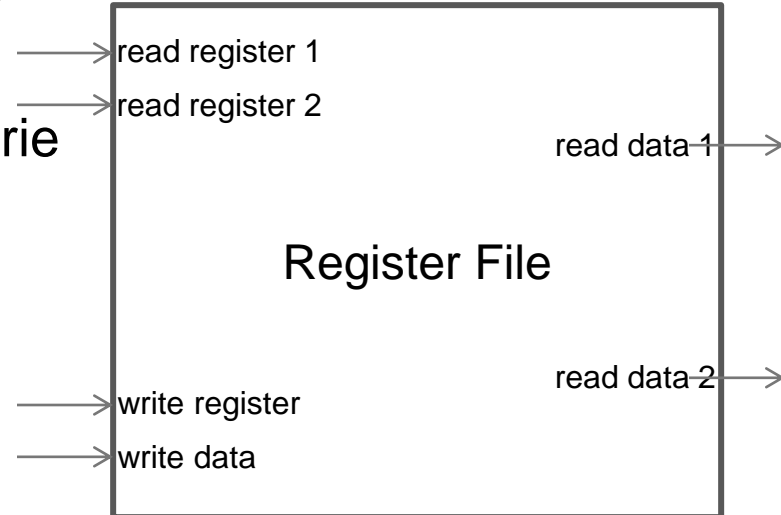
**Stack Pointer (ESP):** adresa stivei

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



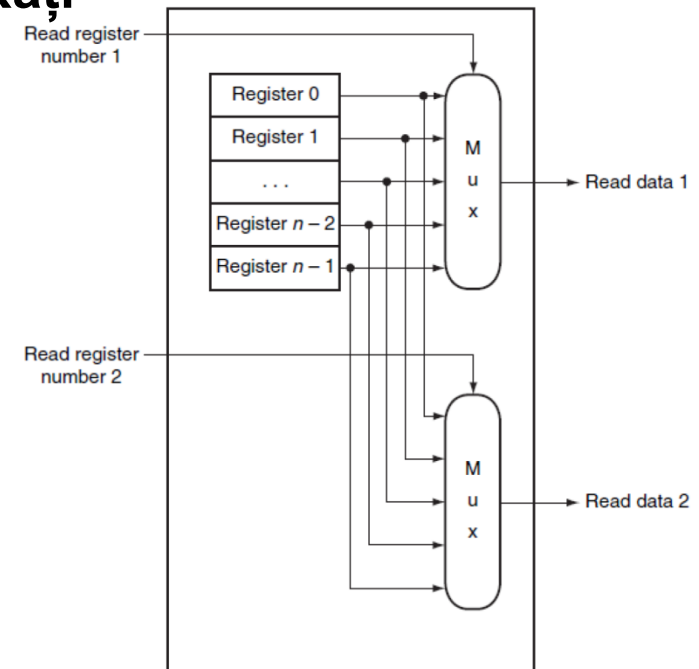
# ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Architecture (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



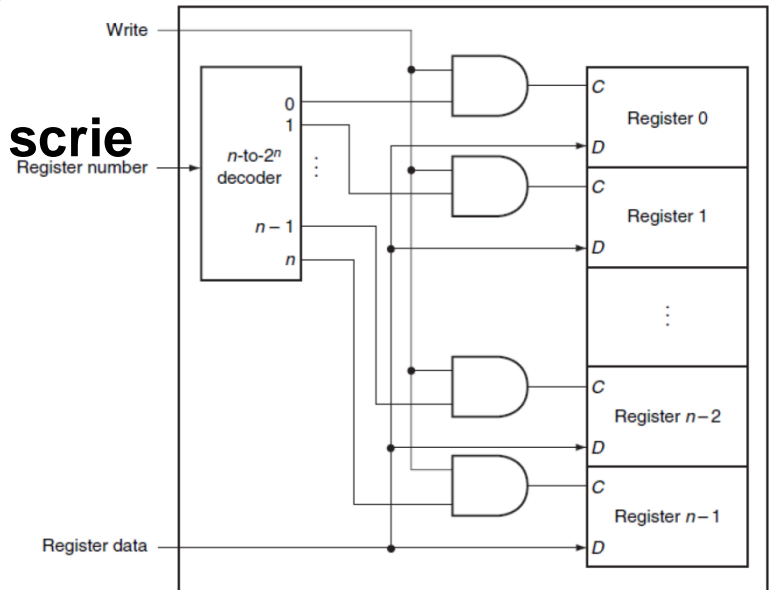
# ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Architecture (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 Architecture (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: indexul 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 Architecture (ISA)**
  - structura sintactică și semantică a limbajului Assembly
    - regiștri
    - **instrucțiuni**
    - tipuri de date
    - metode de adresare a memoriei
  - **<opcode> <listă operanzi>**
    - add op1, op2 ( $op2 \leftarrow op2 + op1$ )
  - **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 Architecture (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

\* pot fi mici variații în funcție de definiții, windows vs linux etc.

# ARHITECTURA SETULUI DE INSTRUCȚIUNI

- **Instruction Set Architecture (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`
    - $\text{Base} + \text{Index} * \text{Scale} + \text{Displacement}$
    - îl aveți explicat detaliat în suportul de laborator

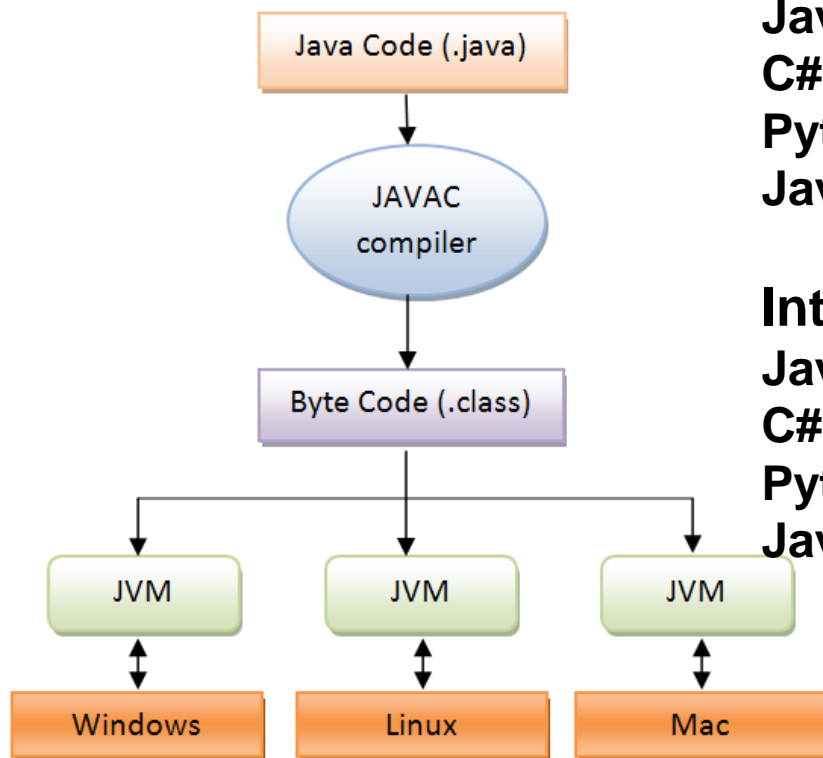
# ARHITECTURA SETULUI DE INSTRUCȚIUNI

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

# 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:** .py, python byte-code (fișiere .pyc)

**Javascript:** .js

## Interpreter:

**Java:** Java VM

**C#:** Common Language Runtime (CLR) în .NET

**Python:** python Virtual Machine

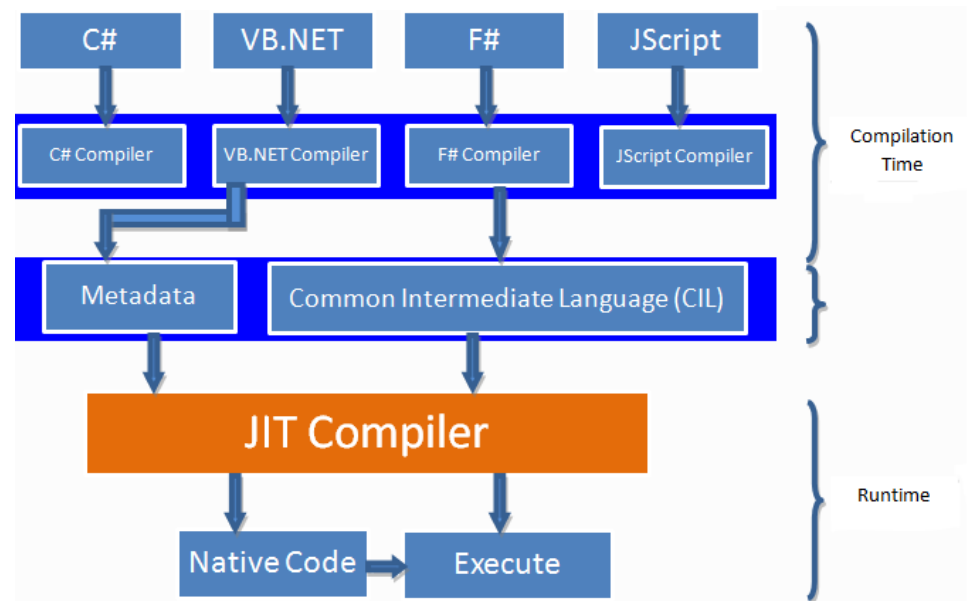
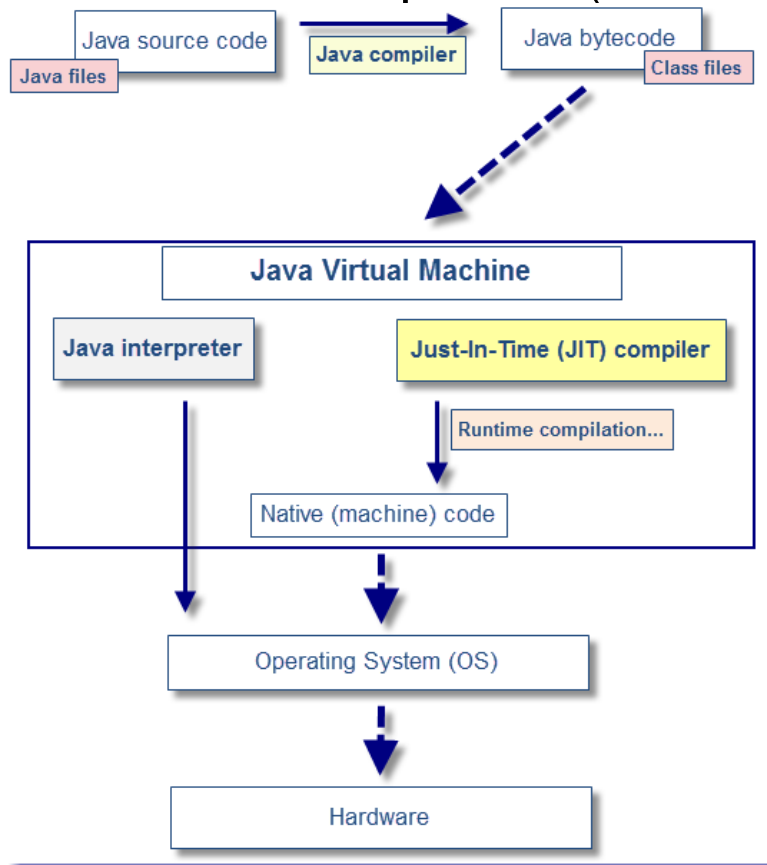
**Javascript:** V8 sau Spider Monkey



# 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

```
#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

```
(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) █
```

verifică dacă ceva este egal cu 2

call la strcmp  
apoi jne

din nou call la puts  
avem asta în cod?

# UN EXEMPLU

- gdb checklicense

```
#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>: ine     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



# UN EXEMPLU

- objdump -d checklicense

```
00000000000000740 <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  %eax,%rsp
7c0: c3              retq
7c1: 66 2e 0f 1f 84 00 00 nopw    %cs:0x0(%rax,%rax,1)
7c8: 00 00 00          nopl    0x0(%rax,%rax,1)
7cb: 0f 1f 44 00 00    nopl    0x0(%rax,%rax,1)
```

- **hexeditor checklicense**

```

00000790 75 0E 48 8D 3D E2 00 00 00 E8 32 FE FF FF EB 1A .....2.....u.H.=.....$.H.
000007A0 48 8D 3D E4 00 00 00 E8 24 FE FF FF EB 0C 48 8D H.=.....$.H.
000007B0 3D E4 00 00 00 E8 16 FE FF FF B8 00 00 00 00 C9 =.....D..
000007C0 C3 66 2E 0F 1F 84 00 00 00 00 00 0F 1F 44 00 00 .f.....D..
000007D0 41 57 41 56 41 89 FF 41 55 41 54 4C 8D 25 F6 05 AWAVA..AUATL.%..
000007E0 20 00 55 48 8D 2D F6 05 20 00 53 49 89 F6 49 89 .UH.-... .SI..I.
000007F0 D5 4C 29 E5 48 83 EC 08 48 C1 FD 03 E8 9F FD FF .L).H...H.....
00000800 FF 48 85 ED 74 20 31 DB 0F 1F 84 00 00 00 00 00 .H..t 1.....
00000810 4C 89 EA 4C 89 F6 44 89 FF 41 FF 14 DC 48 83 C3 L..L..D..A...H..
00000820 01 48 39 DD 75 EA 48 83 C4 08 5B 5D 41 5C 41 5D .H9.u.H...[JA\A]
00000830 41 5E 41 5F C3 90 66 2E 0F 1F 84 00 00 00 00 00 A^A...f.....
00000840 F3 C3 00 00 48 83 EC 08 48 83 C4 08 C3 00 00 00 ....H...H.....
00000850 01 00 02 00 43 68 65 63 6B 69 6E 67 20 74 68 65 ....Checking the
00000860 20 6C 69 63 65 6E 73 65 3A 20 25 73 20 2E 2E 2E license: %s ...
00000870 0A 00 41 42 43 44 45 46 47 48 00 41 63 63 65 73 ..ABCDEFGH.Acces
00000880 73 20 67 72 61 6E 74 65 64 21 00 41 63 63 65 73 s granted!.Acces
00000890 73 20 64 65 6E 69 65 64 00 55 73 61 67 65 3A 20 s denied.Usage:
000008A0 3C 6B 65 79 3E 00 00 00 01 1B 03 3B 3C 00 00 00 <key>.....;<...
000008B0 06 00 00 00 18 FD FF FF 88 00 00 00 58 FD FF FF .....X...
000008C0 B0 00 00 00 68 FD FF FF 58 00 00 00 98 FE FF FF ....h...X.....
000008D0 C8 00 00 00 28 FF FF FF E8 00 00 00 98 FF FF FF .....(.....
000008E0 30 01 00 00 00 00 00 00 14 00 00 00 00 00 00 00 0. ....
000008F0 01 7A 52 00 01 78 10 01 1B 0C 07 08 90 01 07 10 .zR..x.....
00000900 14 00 00 00 1C 00 00 00 08 FD FF FF 2B 00 00 00 .....+...
00000910 00 00 00 00 00 00 00 00 14 00 00 00 00 00 00 00 .....
00000920 01 7A 52 00 01 78 10 01 1B 0C 07 08 90 01 00 00 .zR..x.....
00000930 24 00 00 00 1C 00 00 00 88 FC FF FF 40 00 00 00 $. ....@...
00000940 00 0E 10 46 0E 18 4A 0F 0B 77 08 80 00 3F 1A 3B ...F..J..w...?..;
00000950 2A 33 24 22 00 00 00 00 14 00 00 00 44 00 00 00 *3$"...D...
00000960 A0 FC FF FF 08 00 00 00 00 00 00 00 00 00 00 00 .....
00000970 1C 00 00 00 5C 00 00 00 C8 FD FF FF 81 00 00 00 ...\. ....
00000980 00 41 0E 10 86 02 43 0D 06 02 7C 0C 07 08 00 00 .A...C...|....
00000990 44 00 00 00 7C 00 00 00 38 FE FF FF 65 00 00 00 D...|...8...e...
000009A0 00 42 0E 10 8F 02 42 0E 18 8E 03 45 0E 20 8D 04 .B...B...E...
000009B0 42 0E 28 8C 05 48 0E 30 86 06 48 0E 38 83 07 4D B.(.H.0..H.8..M
000009C0 0E 40 72 0E 38 41 0E 30 41 0E 28 42 0E 20 42 0E .@r.8A.0A.(B. B.
000009D0 18 42 0E 10 42 0E 08 00 14 00 00 00 C4 00 00 00 .B..B.....
000009E0 60 FE FF FF 02 00 00 00 00 00 00 00 00 00 00 00 \.....
000009F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000A00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000A10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000A20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000A30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000A40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
^G Help ^C Exit (No Save) ^T goTo Offset ^X Exit and Save ^W Search ^U Undo ^L Redraw ^E Text

```

schimbăm JNE?  
care este noul OPCODE  
pentru noua instrucțiune?

- **hexeditor checklicense**

```

00000790 75 0E 48 8D 3D E2 00 00 00 E8 32 FE FF FF EB 1A u.H.=.....2....
000007A0 48 8D 3D E4 00 00 00 E8 24 FE FF FF EB 0C 48 8D H.=.....$.H..
000007B0 3D E4 00 00 00 E8 16 FE FF FF B8 00 00 00 00 C9 =.....
000007C0 C3 66 2E 0F 1F 84 00 00 00 00 00 0F 1F 44 00 00 .f.....D..
000007D0 41 57 41 56 41 89 FF 41 55 41 54 4C 8D 25 F6 05 AWAVA..AUATL.%..
000007E0 20 00 55 48 8D 2D F6 05 20 00 53 49 89 F6 49 89 .UH.-.. .SI..I.
000007F0 D5 4C 29 E5 48 83 EC 08 48 C1 FD 03 E8 9F FD FF .L).H...H.....
00000800 FF 48 85 ED 74 20 31 DB 0F 1F 84 00 00 00 00 00 .H..t 1.....
00000810 4C 89 EA 4C 89 F6 44 89 FF 41 FF 14 DC 48 83 C3 L..L..D...A...H..
00000820 01 48 39 DD 75 EA 48 83 C4 08 5B 5D 41 5C 41 5D .H9.u.H...[A\A]
00000830 41 5E 41 5F C3 90 66 2E 0F 1F 84 00 00 00 00 00 A^A...f.....
00000840 F3 C3 00 00 48 83 EC 08 48 83 C4 08 C3 00 00 00 ....H...H.....
00000850 01 00 02 00 43 68 65 63 6B 69 6E 67 20 74 68 65 ....Checking the
00000860 20 6C 69 63 65 6E 73 65 3A 20 25 73 20 2E 2E 2E license: %s .
00000870 0A 00 41 42 43 44 45 46 47 48 00 41 63 63 65 73 ..ABCDEFGH.Acces
00000880 73 20 67 72 61 6E 74 65 64 21 00 41 63 63 65 73 s granted!.Acces
00000890 73 20 64 65 6E 69 65 64 00 55 73 61 67 65 3A 20 s denied.Usage:
000008A0 3C 6B 65 79 3E 00 00 00 01 1B 03 3B 3C 00 00 00 <key>.....;<...
000008B0 06 00 00 00 18 FD FF FF 88 00 00 00 58 FD FF FF .....X...
000008C0 B0 00 00 00 68 FD FF FF 58 00 00 00 98 FE FF FF ....h...X.....
000008D0 C8 00 00 00 28 FF FF FF E8 00 00 00 98 FF FF FF .....(.....
000008E0 30 01 00 00 00 00 00 00 14 00 00 00 00 00 00 00 0.
000008F0 01 7A 52 00 01 78 10 01 1B 0C 07 08 90 01 07 10 .zR..x.....
00000900 14 00 00 00 1C 00 00 00 08 FD FF FF 2B 00 00 00 .....+.....
00000910 00 00 00 00 00 00 00 00 14 00 00 00 00 00 00 00 .....
00000920 01 7A 52 00 01 78 10 01 1B 0C 07 08 90 01 00 00 .zR..x.....
00000930 24 00 00 00 1C 00 00 00 88 FC FF FF 40 00 00 00 $......@...
00000940 00 0E 10 46 0E 18 4A 0F 0B 77 08 80 00 3F 1A 3B ...F...J...w...?.;
00000950 2A 33 24 22 00 00 00 00 14 00 00 00 44 00 00 00 *3$".....D...
00000960 A0 FC FF FF 08 00 00 00 00 00 00 00 00 00 00 00 .....
00000970 1C 00 00 00 5C 00 00 00 C8 FD FF FF 81 00 00 00 ....\.....
00000980 00 41 0E 10 86 02 43 0D 06 02 7C 0C 07 08 00 00 .A...C...|. ....
00000990 44 00 00 00 7C 00 00 00 38 FE FF FF 65 00 00 00 D...|. ...8...e...
000009A0 00 42 0E 10 8F 02 42 0E 18 8E 03 45 0E 20 8D 04 .B...B...E...
000009B0 42 0E 28 8C 05 48 0E 30 86 06 48 0E 38 83 07 4D B.(..H..0..H.8..M
000009C0 0E 40 72 0E 38 41 0E 30 41 0E 28 42 0E 20 42 0E .@r.8A.0A.(B. B.
000009D0 18 42 0E 10 42 0E 08 00 14 00 00 00 C4 00 00 00 .B..B.....
000009E0 60 FE FF FF 02 00 00 00 00 00 00 00 00 00 00 00 `.....
000009F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000A00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000A10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000A20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000A30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000A40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

schimbați voi și alte lucruri:

- parola
- textul afișat



# 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)	<a href="#">Torrent</a>	2020.4	4.1G	50492d761e400c2b5e22c8f253dd6f75c27e4bc84e33c2eff272476a0588fb02
Kali Linux 64-Bit (Live)	<a href="#">Torrent</a>	2020.4	3.3G	4d764a2ba67f41495c17247184d24b7f9ac9a7c57415bbbed663402aec78952b

# EXECUȚIA DATELOR

- fie următorul program foarte simplu (shellcode.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 întâmplă 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



