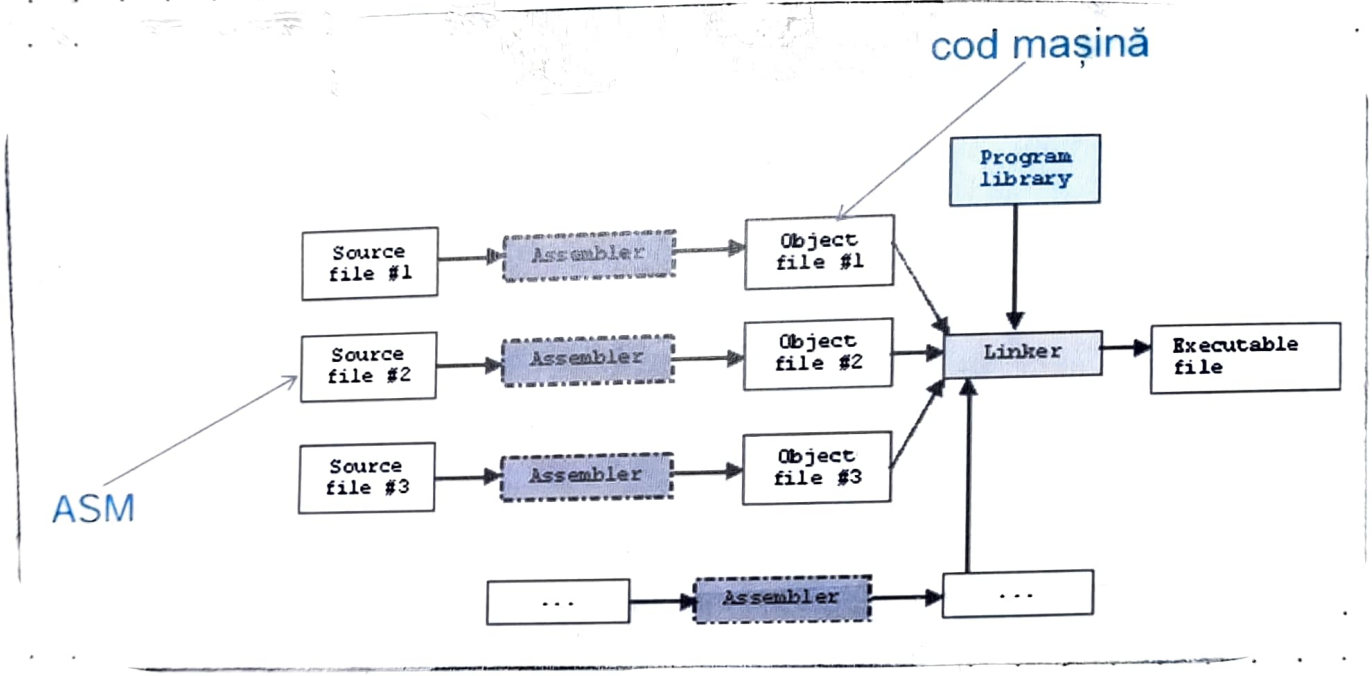
CURS 8

DE LA COD SURSĂ LA EXECUȚIE

- **Cod mașină**
  - instrucțiuni binare executate de CPU
  - se obține din cod sursă
  - e specific pt Assembler, CPU, OS
  - CPU poate executa doar cod mașină, altfel e transf. în CM



Source Code (C, C_{++}) → (compile) Assembly → (Assemble) Object File → (Link) → Binary File → (Load) Process

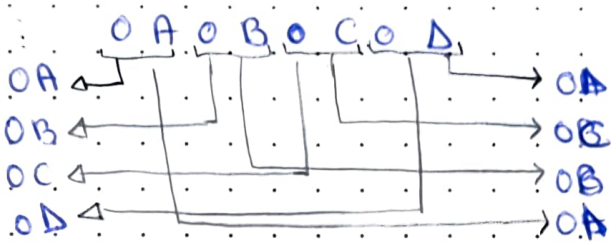
Libraries → Link

- **Objdump main**
  - instrucțiunile sunt codate cu aceeași lungime

- Big-endian →

&

Little-endian →




```
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
    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
    116e:       66 90                   xchg   %ax,%ax
```

(cod mașină)

→ adresa de memorie

Cod sursă

Assembly

```
#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   0x5a0 <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)
```

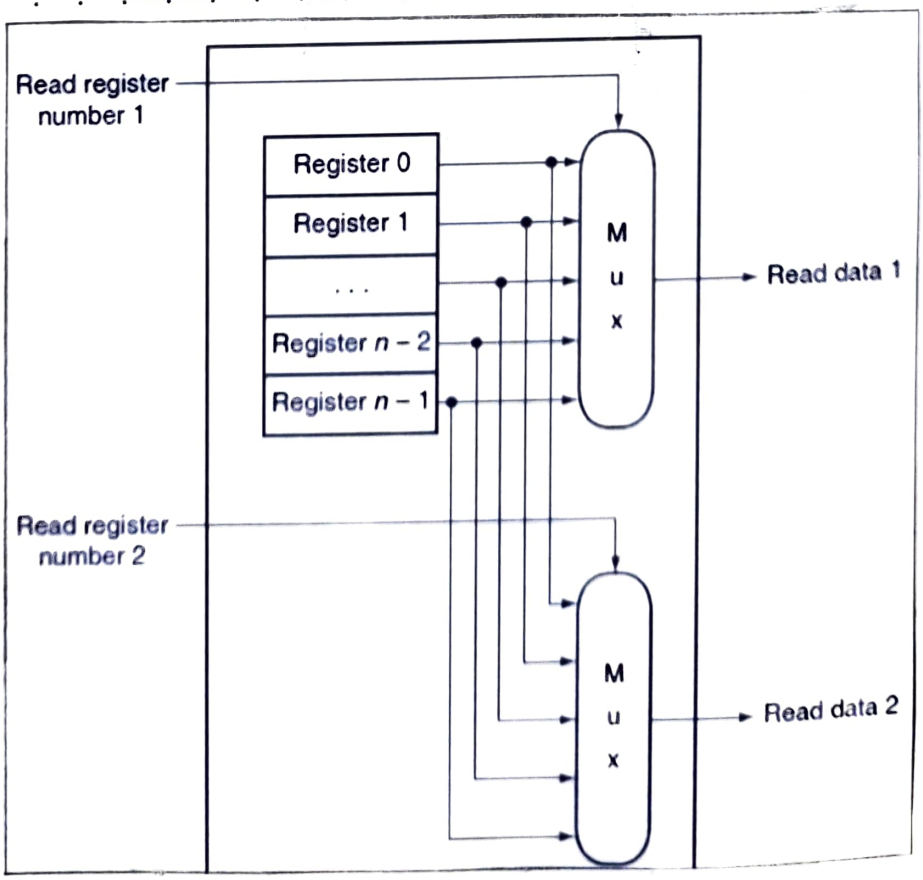
- Instruction Set Architecture (ISA)
  - registrii
  - instrucțiuni & tipuri de date
  - metode de adresare a memoriei

Registrii

- flags

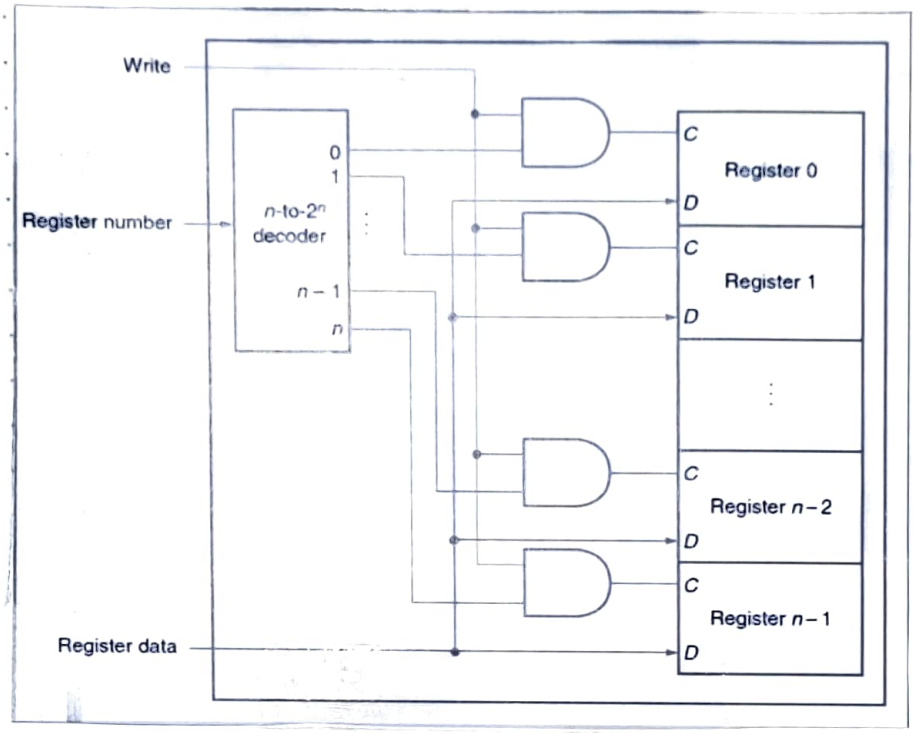
Instruction Mnemonic	Condition (Flag States)	Description
Unsigned Conditional Jumps		
JA/JNBE	(CF or ZF) = 0	Above/not below or equal
JAE/JNB	CF = 0	Above or equal/not below
JB/JNAE	CF = 1	Below/not above or equal
JBE/JNA	(CF or ZF) = 1	Below or equal/not above
JC	CF = 1	Carry
JE/JZ	ZF = 1	Equal/zero
JNC	CF = 0	Not carry
JNE/JNZ	ZF = 0	Not equal/not zero
JNP/JPO	PF = 0	Not parity/parity odd
JP/JPE	PF = 1	Parity/parity even
JCXZ	CX = 0	Register CX is zero
JECXZ	ECX = 0	Register ECX is zero
Signed Conditional Jumps		
JG/JNLE	((SF xor OF) or ZF) = 0	Greater/not less or equal
JGE/JNL	(SF xor OF) = 0	Greater or equal/not less
JL/JNGE	(SF xor OF) = 1	Less/not greater or equal
JLE/JNG	((SF xor OF) or ZF) = 1	Less or equal/not greater
JNO	OF = 0	Not overflow
JNS	SF = 0	Not sign (non-negative)
JO	OF = 1	Overflow
JS	SF = 1	Sign (negative)

- grupati si inexati
  - → read register 1/2: index de citire
  - → read data 1/2: datele citite



→ write register: indexul în care se scrie

→ write data: datele care se scriu



- Metode de adresare a memoriei
  - adresare imediată
    - imediat: mov $172, %rdi
    - cu registru: mov %rcx, %rdi
    - cu memorie: mov 0x172, %rdi
  - adresare indirectă
    - prin registru: mov (%rax), %rdi
    - indexat: mov 172(%rax), %rdi
    - bazat pe IP: mov 172(%rip), %rdi
  - cazul cel mai general: mov 172(%rdi, %rdx, 8), %rax
    - base + Index * scale + displacement
- excepție de la regulă
  - bytecode (cod interpretat): instrucțiunile sunt executate de un interpretor care apoi le trimite la CPU
  - totul e lent pt că mai este un pas de procesare

- JIT compilation (Just-in-Time) ajută

Interpreted code:

JAVA: java byte-code

C#: Common Intermediate Language (CIL)

Python: .py > python byte-code

Javascript: .js

Interpreter

Java: Java VM

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

Python: python VM

Javascript: V8 sau Spider Monkey