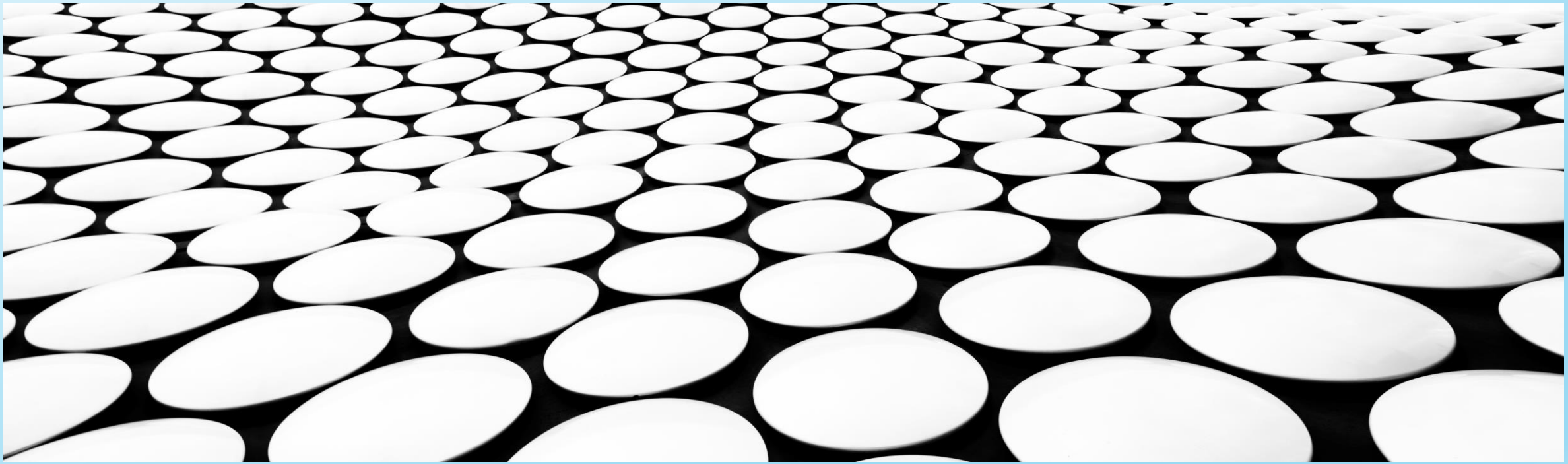
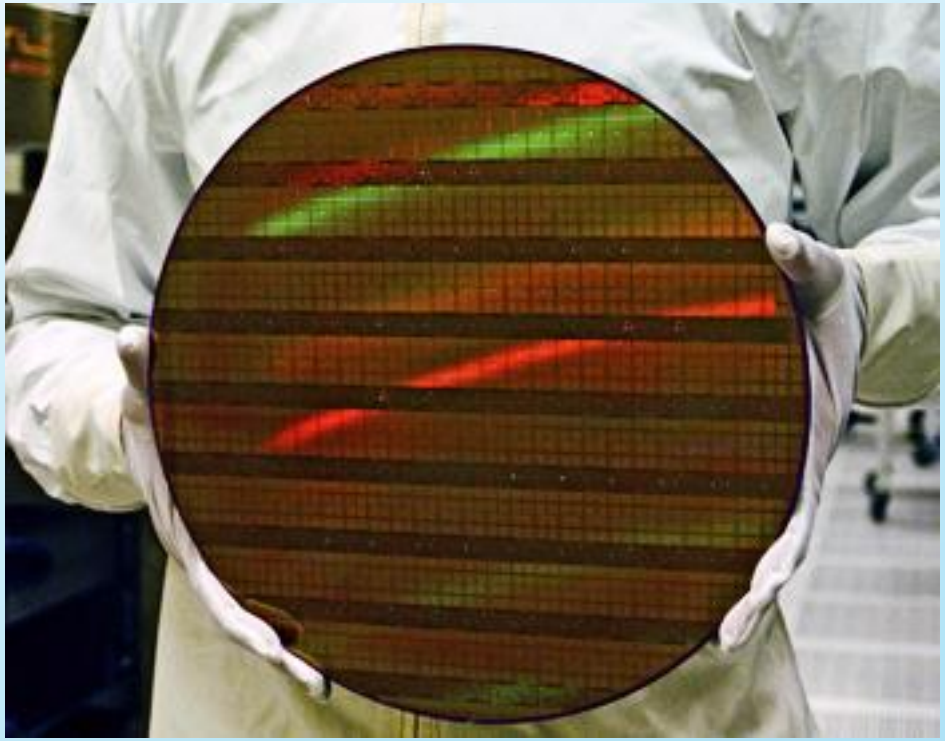
ARHITECTURA SISTEMELOR DE CALCUL

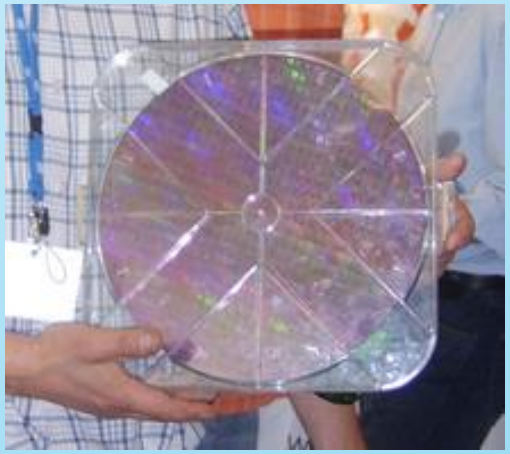
UB, FMI, CTI, ANUL III, 2022-2023

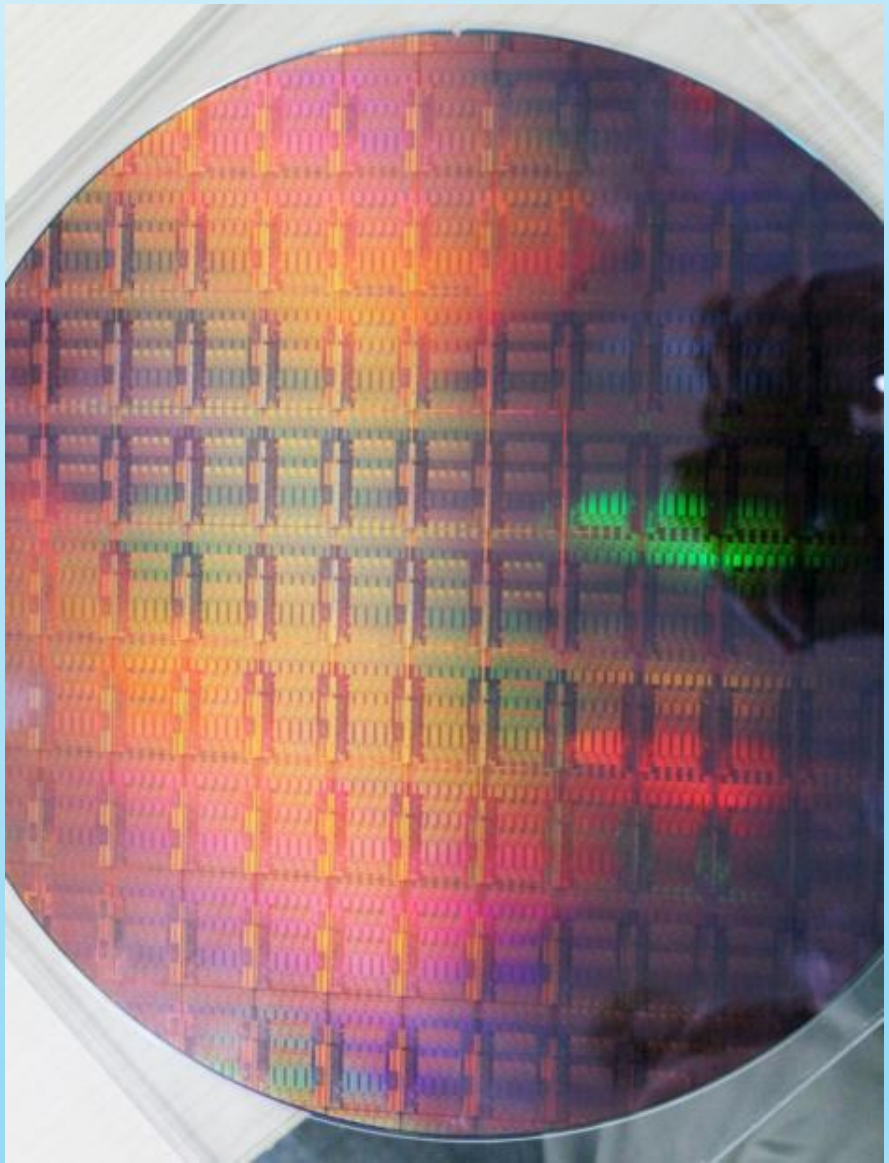


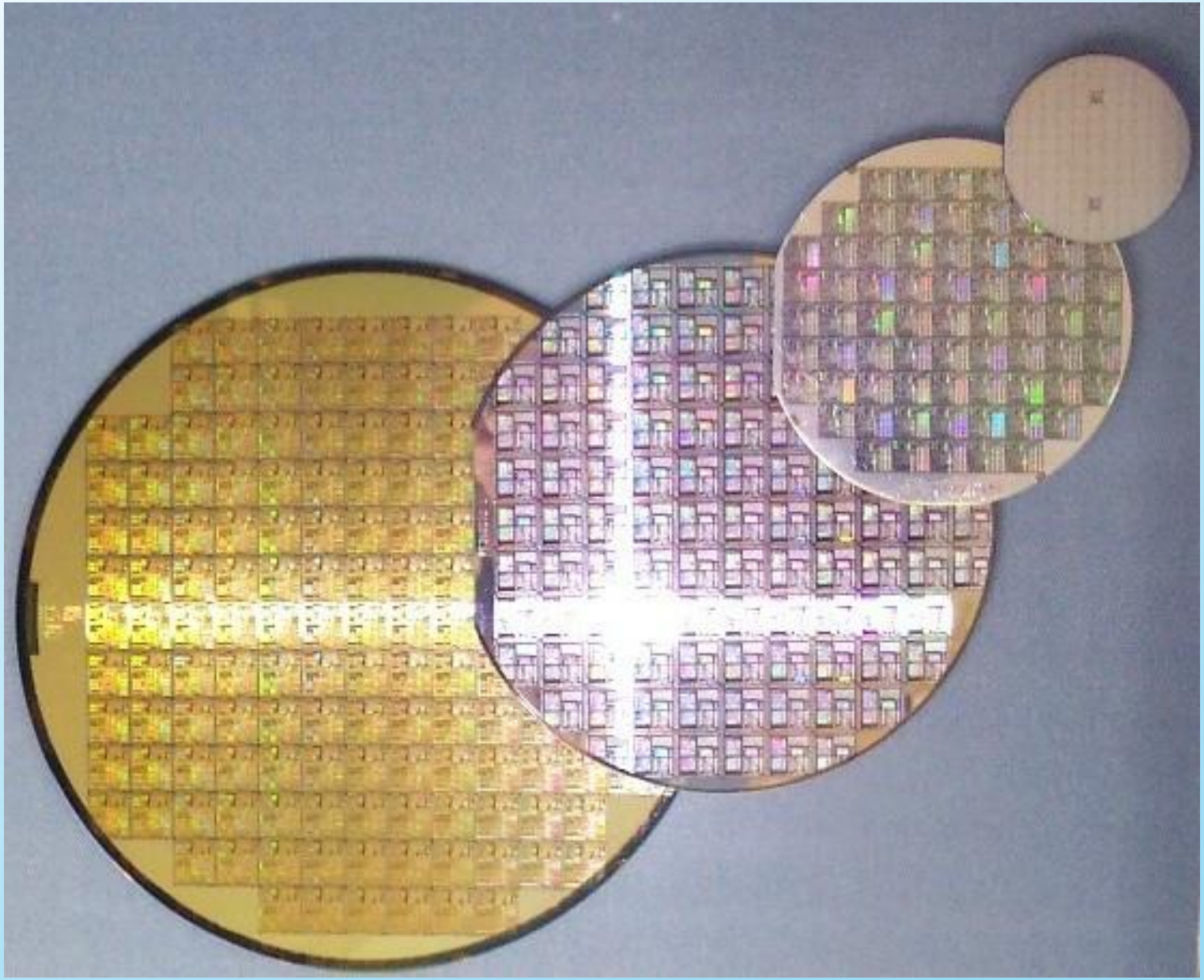
UNITATEA DE CONTROL

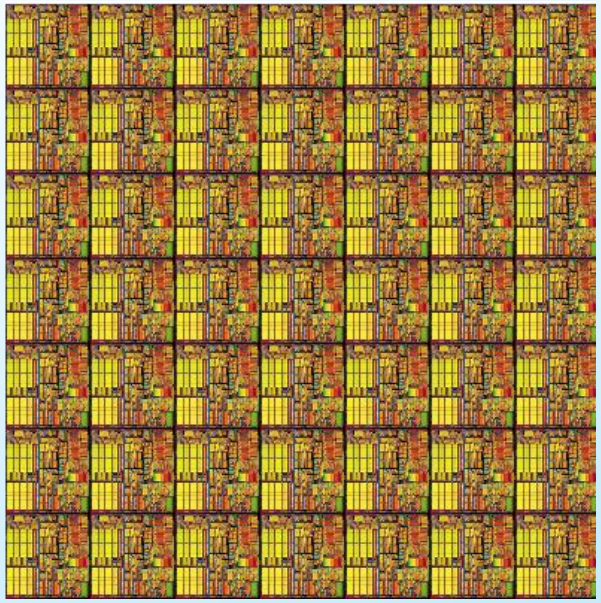
Cum se fabrica un microprocesor!

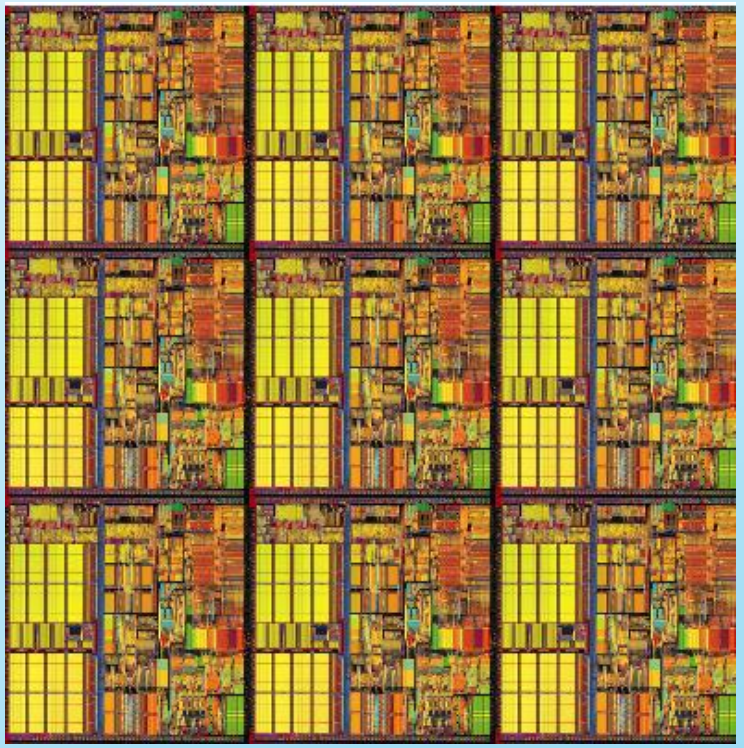


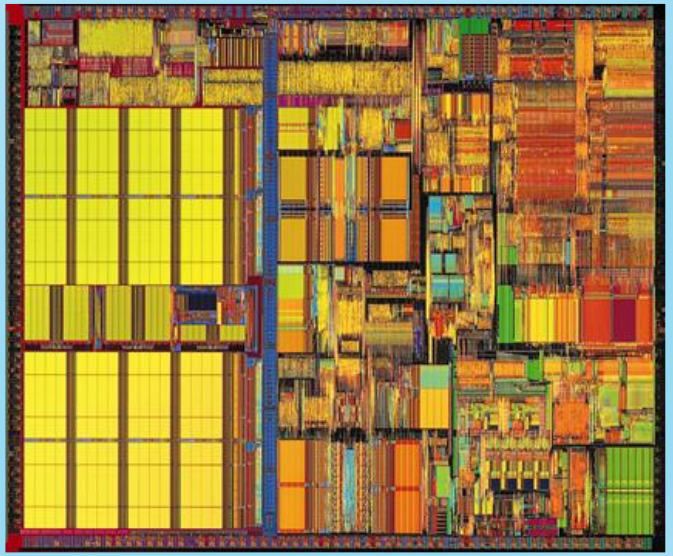




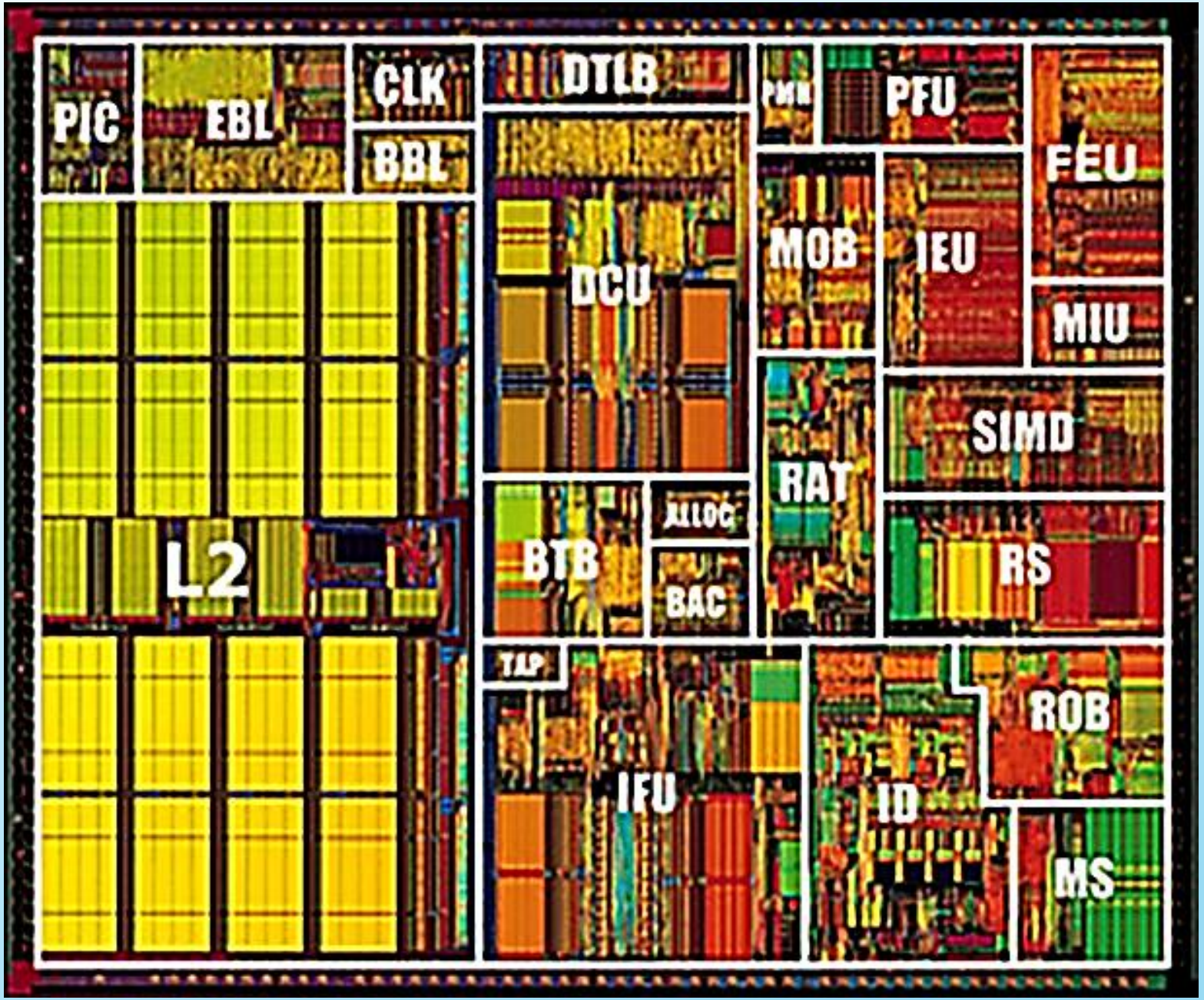








Procesor Pentium III



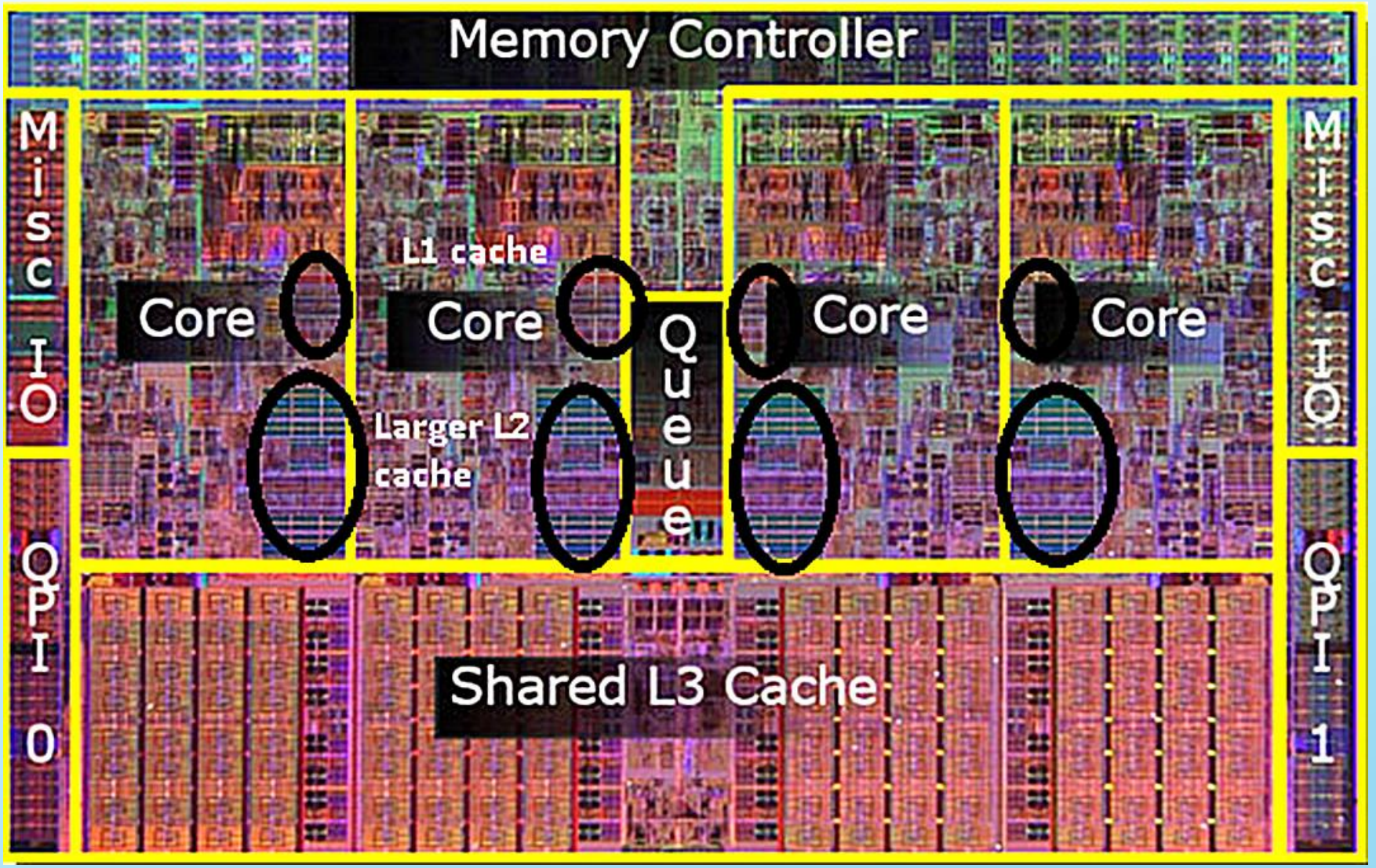
- 0.18 micron,
6-layer metal CMOS process technology
- 28.1 **M** transistors
- 3-way superscalar out-of-order execution micro-architecture
- 256K Level 2 Cache
- 133 MHz IO bus

FEU - Floating point Execution Unit.

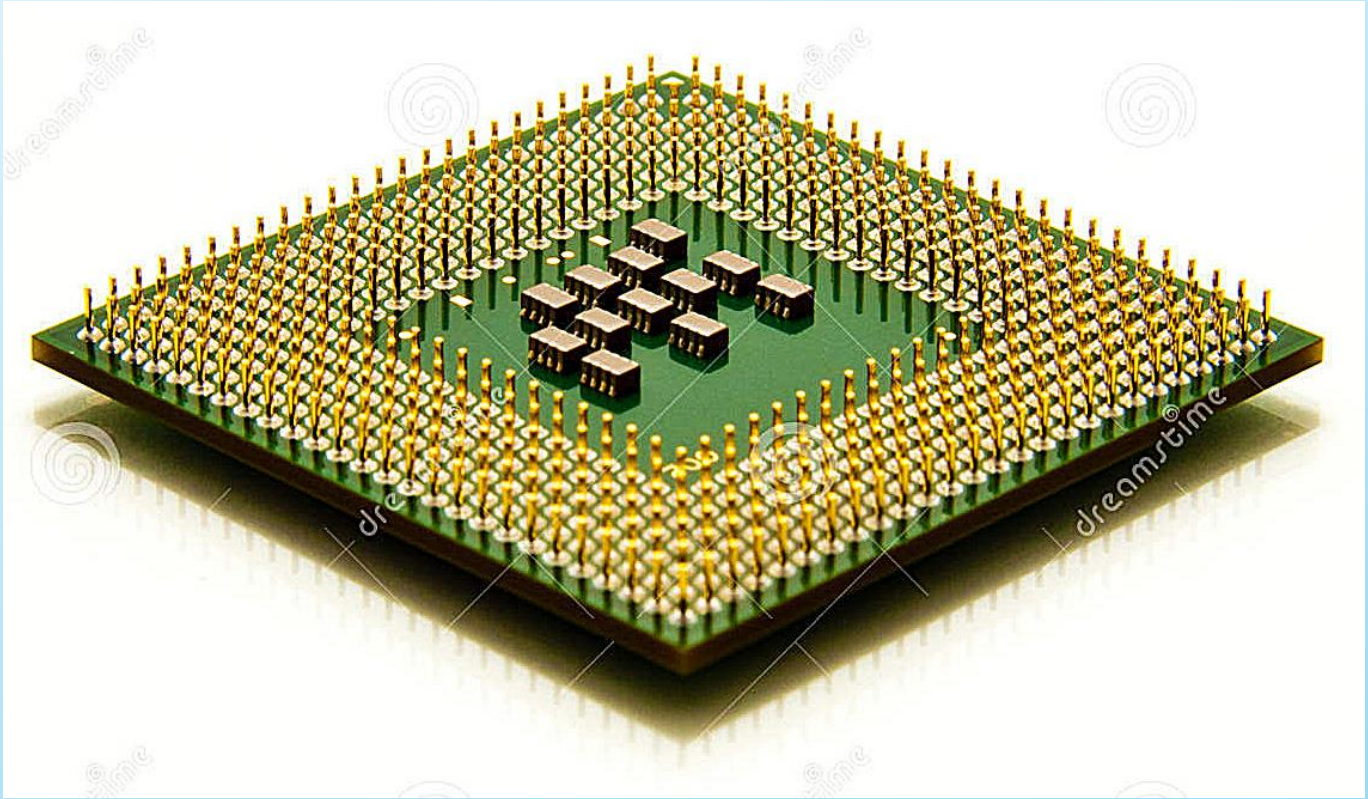
IEU - Integer Execution Unit.

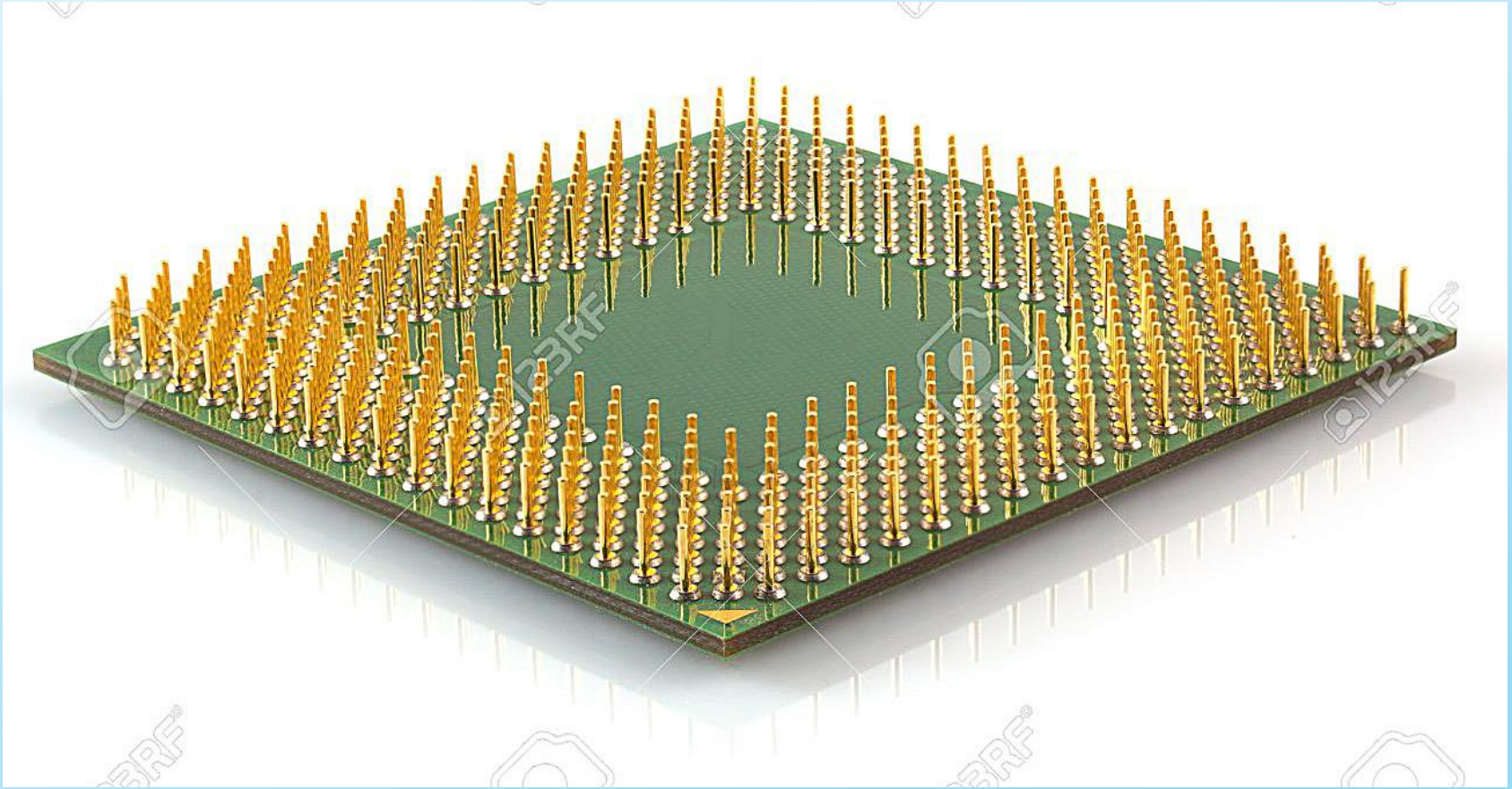
ID - Instruction Decoder.

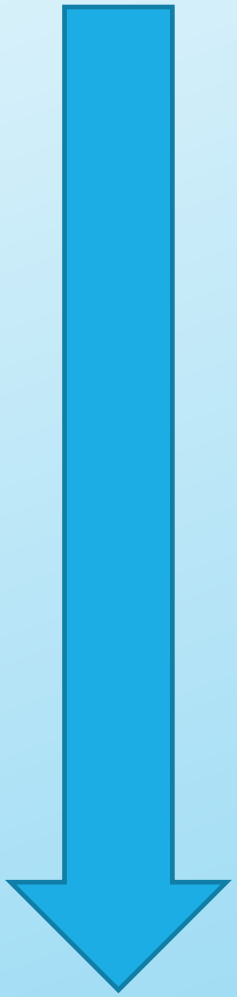
- **IFU** - Instruction Fetch Unit. Instruction fetch logic and a 16K Byte 4-way set-associative level one instruction cache resides in this block. Instruction data from the IFU is then forwarded to the ID.
- **ID** - Instruction Decoder. This unit is capable of decoding up to 3 instructions per cycle.
- **IEU** - Integer Execution Unit. This is responsible for ALU functionality of scalar integer instructions. Address calculations for memory referencing instructions are also performed here along with target address calculations for jump related instructions.
- **FEU** - Floating point Execution Unit. This performs floating point related calculations for both existing scalar instructions along with support for some of the SIMD-FP instructions.
- **DCU** - Data Cache Unit. Contains the non-blocking 16K Byte 4-way set-associative level one data cache along with associated fill and write back buffering.
- **PIC** - Programmable Interrupt Controller. Local interrupt controller logic for multi-processor interrupt distribution and boot-up communication.
- **CLK** - Clocking. Contains phase lock loop and other clocking control circuitry.
- **L2** - Level 2 Cache. 256K unified level two cache.


Memory Controller
Misc IO
Misc IO
L1 cache
Core
Core
Queue
Core
Core
Larger L2 cache
QPI 0
QPI 1
Shared L3 Cache

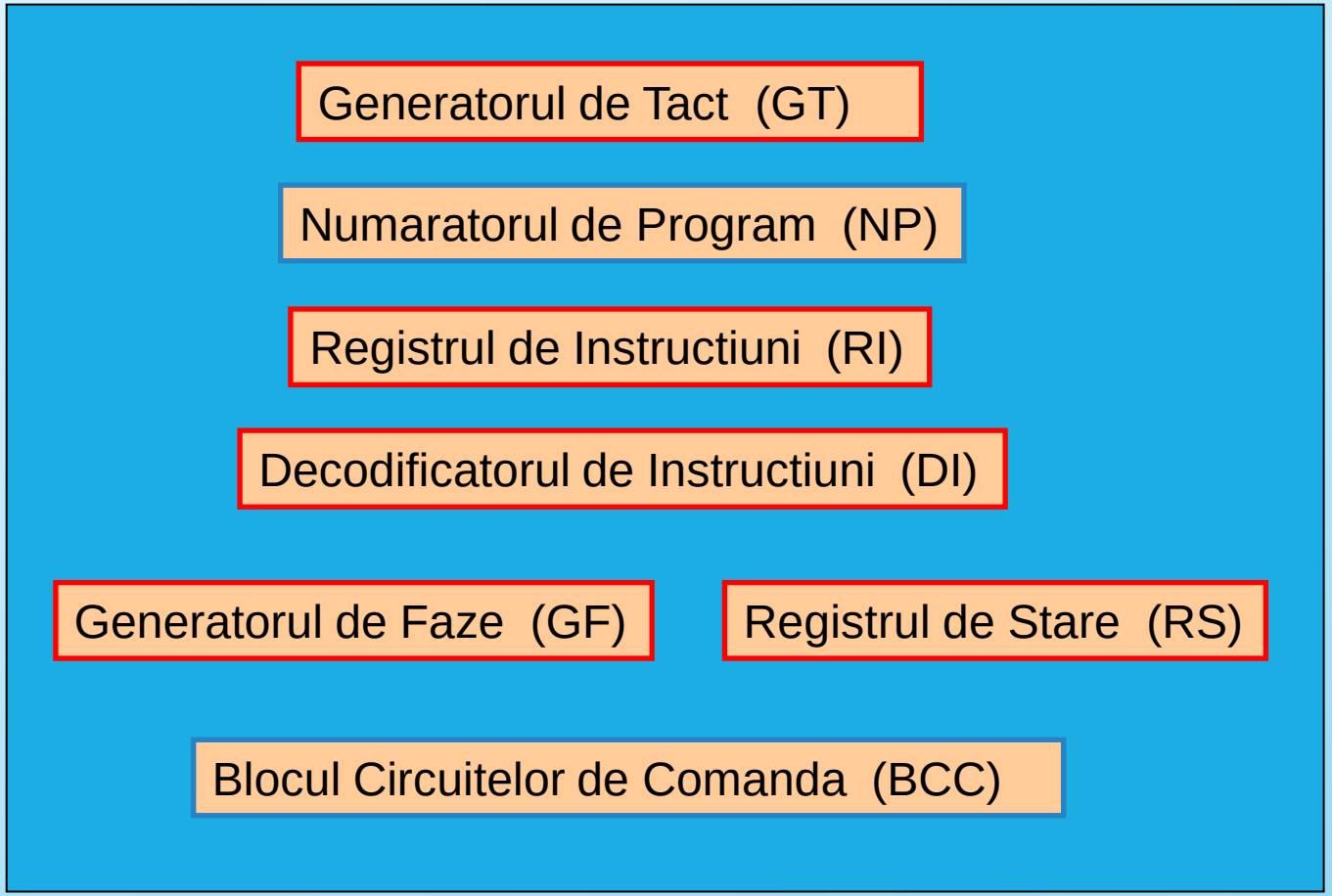

intel
Core™ i7


dreamstime
dreamstime





UNITATEA DE CONTROL



UNITATEA DE CONTROL

- Executa pe rand instructiunile din program

- Este formata din:
  - Generatorul de Tact (GT)
  - Numaratorul de Program (NP)
  - Registrul de Instructiuni (RI)
  - Decodificatorul de Instructiuni (DI)
  - Generatorul de Faze (GF)
  - Registrul de Stare (RS)
  - Blocul Circuitelor de Comanda (BCC)

Cadenta schimbarilor de stare

pentru toate circuitele secventiale din structura calculatorului

este data de catre GENERATORUL DE TACT [GT]

NP, RI, DI, GF

- NUMARATORUL DE PROGRAM [**NP**] pastreaza si ofera adresa, din memoria principala, a instructiunii in curs de executie.
- Codul instructiunii este citit din memoria principala si inscris in REGISTRUL DE INSTRUCTIUNI [**RI**].
- GENERATORUL DE FAZE [**GF**] construieste succesiunea de faze necesare pentru executia instructiunii din RI. Este, de regula, parte a decodificatorului de instructiuni [DI]

FAZE, RS

- Generarea fazei urmatoare este determinata de 3 elemente:
  - Tipul instructiunii
  - Faza curenta
  - Continutul registrului de stare.

- In REGISTRUL DE STARE [RS] sunt pastrate informatii despre modul de efectuare a operatiilor comandate de BCC, sau sunt pastrate informatii despre continutul registrilor generali

BCC
(Blocul Circuitelor de Comanda)

- Toate operatiunile elementare care se realizeaza pe parcursul executiei unei instructiuni sunt comandate de catre semnalele generate de BCC (microoperatiuni).

  - Operatiunile elementare se numesc **microoperatiuni**
  - Semnalele care comanda microoperatiunile se numesc: **microcomenzi**
  - Microcomenzile nu sunt accesibile programatorului

- Microcomenzile sunt trimise elementelor de executie din structura calculatorului:
  - registre, UAL, memorie, porturi, etc

  care realizeaza operatiuni elementare cum ar fi:
  - inscriere, validare iesire, stergere, deplasare stânga, etc.

- Executia unei instructiuni se desfasoara sub forma unei succesiuni de microoperatii.

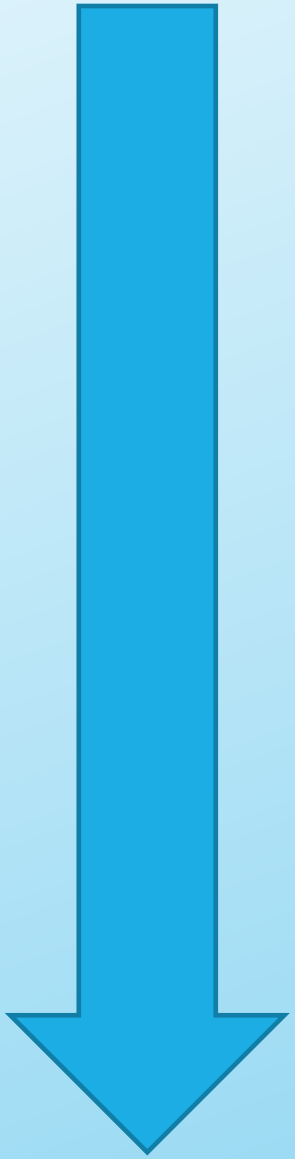
- Toate microoperatiile care se executa in acelasi timp definesc o situatie in executia instructiunii numita **faza.**

Executia instructiunilor de catre UC
(ciclul extragere-decodificare-executie)

Pentru fiecare instructiune UC executa urmatoarea secventa de pasi:

1. extrage din memorie de la adresa specificata in **NP** o instructiune si o depoziteaza in **RI**
2. determina tipul instructiunii din **RI**
3. determina adresele operanzilor daca instructiunea foloseste date din memorie
4. incarca datele, de la adresele specificate (daca exista) in **registri generali**
5. executa instructiunea
6. inscrie rezultatul executiei in locul indicat de instructiune
7. modifica numaratorul de program (**NP**) pentru a indica urmatoarea instructiune
8. sare la pasul 1.

- Multimea de instructiuni disponibile unui programator pentru un anumit nivel de masina (virtuala) se numeste **set de instructiuni** al acelei masini. Setul de instructiuni difera de la o masina la alta, de la un nivel la altul.
- Masinile cu un set redus de instructiuni sunt numite calculatoare **RISC** (**R**educed **I**nstruction **S**et **C**omputer)
- Setul de instructiuni contine urmatoarele **tipuri** de instructiuni:
  - Aritmetice
  - Logice
  - Salt
  - Apel
  - Deplasare (registru-registru, memorie-registru, memorie-memorie)
  - Oprire



ISA (Instruction Set Architecture)
Arhitectura Setului de Instructiuni

ISA contine toate operatiile si componentele calculatorului care pot fi “vizibile” programatorului.

Arhitectura include:

- Organizarea memoriei
  - Spatiul de adrese (cate locatii pot fi adresate)
  - Adresabilitate (cati biti pe locatie)
- Setul de registri
  - Cat de multi, ce dimensiuni, mod de utilizare
- Setul de instructiuni
  - Opcodes (**operation code**)
  - Tipuri de date
  - Moduri de adresare

Tipuri de ISA (Instruction Set Architecture)

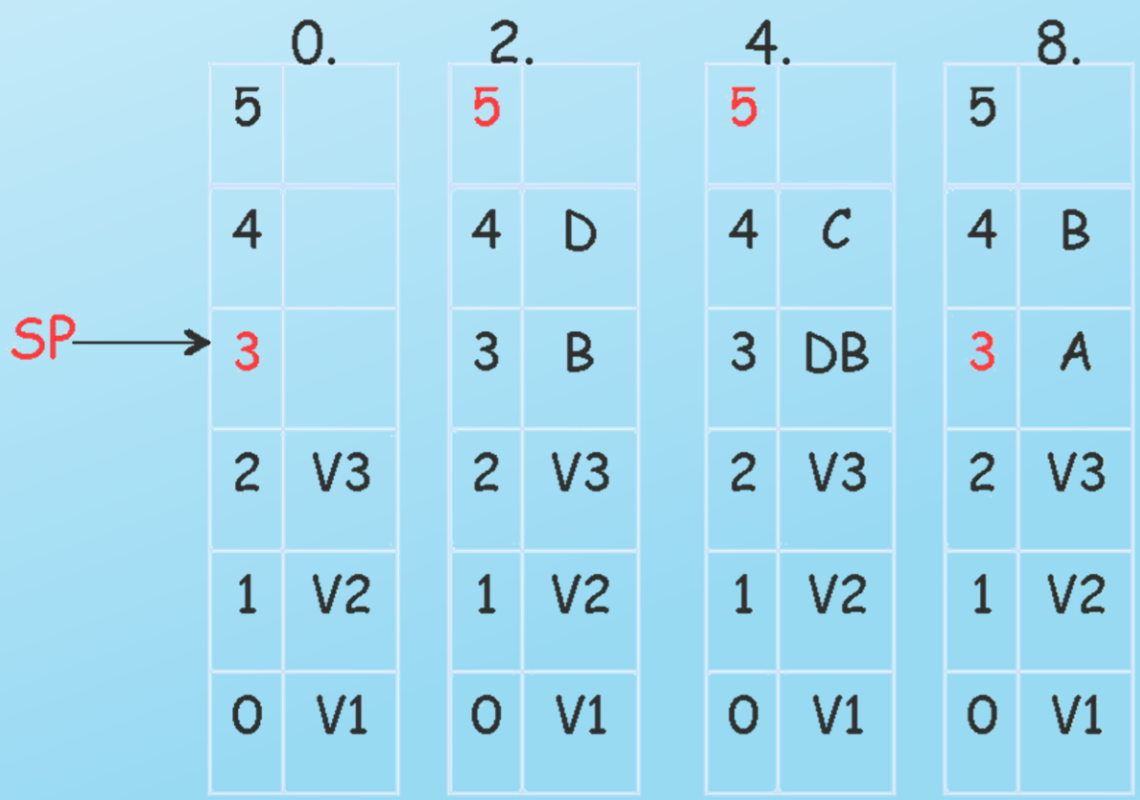
- Arhitectura cu **stiva** (stack)
  - Fara operanzi expliciti
- Arhitectura cu **acumulator**
  - Un operand explicit
- Arhitecturi cu **registri** de uz general
  - 3 operanzi expliciti
  - 3 subtipuri
    - Memorie-memorie
    - Registru-memorie
    - Registru-registru

Arhitectura cu stiva

- 0 operanzi expliciti
- Toti operanzii sunt in stiva

A = B * (C + D * B)

1. Push B
2. Push D
3. *
4. Push C
5. +
6. Push B
7. *
8. Pop A



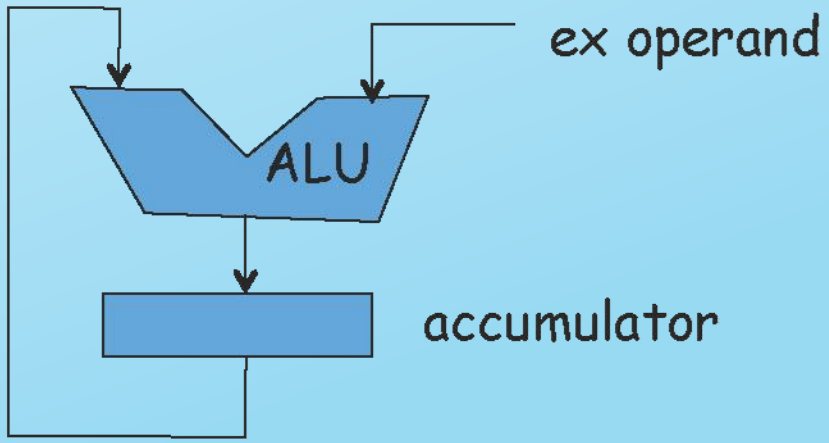
Arhitectura cu acumulator

- Un operand explicit
- Ceilalti doi operanzi sunt in acumulator (o sursa si o destinatie)

O memorie de tip acumulator este o memorie aditiva: retine suma dintre valoarea nou introdusa si valoarea initiala.

A = B * (C + D * B)

1. Load B
2. Mult D
3. Add C
4. Mult B
5. Store A



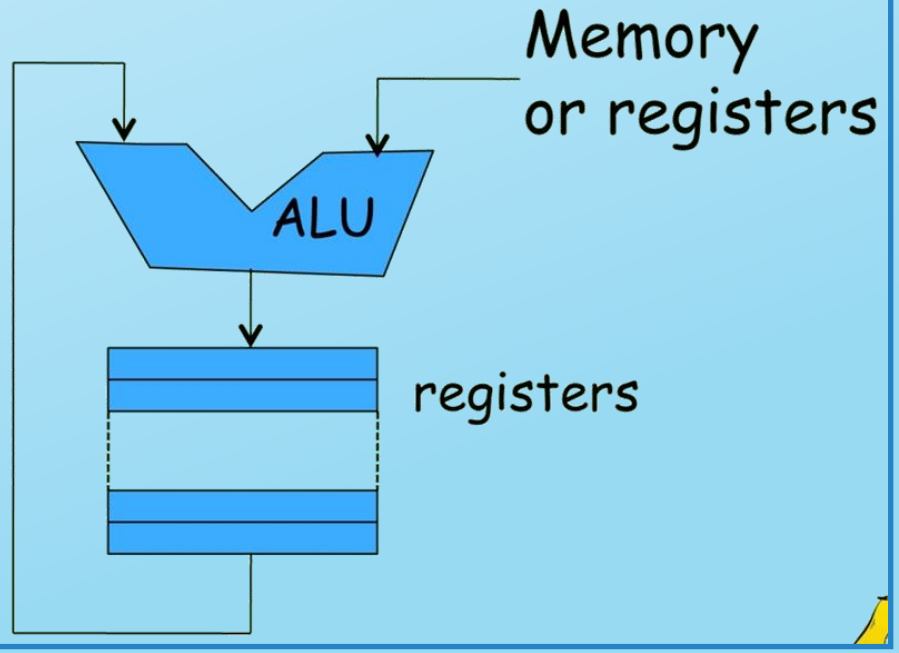
Arhitecturi cu **registri** de uz general

Arhitectura registru-memorie

- Doi operanzi: un registru (pentru o sursa si pentru destinatie) si o locatie de memorie

A = B * (C + D * B)

1. Load R0, B
2. Mult R0, D
3. Add R0, C
4. Mult R0, B
5. Store R0, A

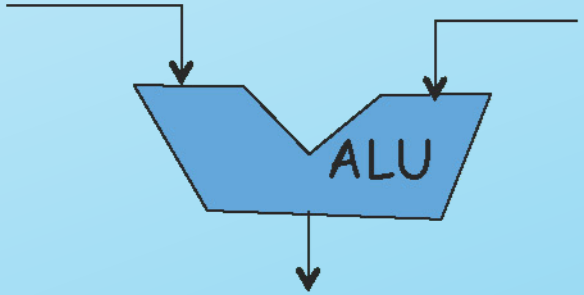


Arhitectura memorie-memorie

- Toti operanzii pot fi localizati: fie in memorie fie intr-un registru

A = B * (C + D * B)

1. Mult R0, D, B
2. Add R0, R0, C
3. Mult A, B, R0



Arhitectura registru-registru (load/store)

- Cei trei operanzi trebuie sa fie in registri separati

A = B * (C + D * B)

1. Load R0, B
2. Load R1, D
3. Mult R2, R0, R1
4. Load R3, C
5. Add R3, R3, R2
6. Mult R4, R3, R0
7. Store R4, A