Curs 13

Elemente de arhitectura procesoarelor

1. Instructiune. Ciclu masina. Stare.
2. Metode de control a transferului de date
3. Taxonomii ale arhitecturilor de prelucare
4. Paralelism - pipeline, superscalar
5. Legea lui Amdahl

with slides by S. Dandamudi, Peng-Sheng Chen, Kip Irvine, Robert Sedgwick and Kevin Wayne

1. Instructiune. Ciclu masina. Stare.

- Instructiunea este o operatie executata de procesor

  (ex. MOV AX,CX, ADD AX, DX, MUL CL,…)

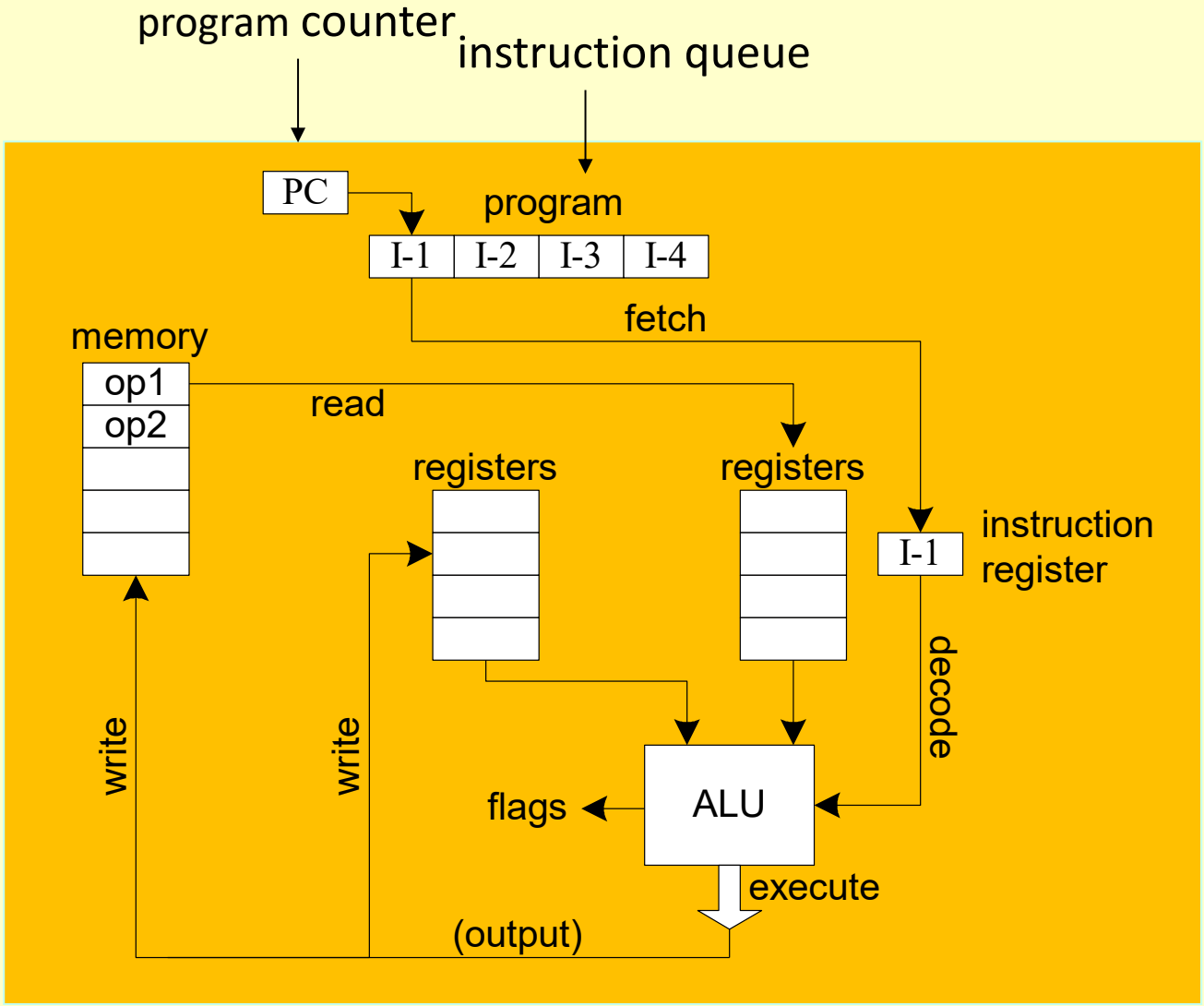
- Operatiile µP se executa sicronizat cu ceasul intern Φ

- Pentru executia unei instructiuni au loc operatii de RD, WR, interne = CICLII MASINA
- o instructiune contine mai multi ciclii masina (3,…n), iar fiecare ciclu masina dureaza mai multi tacti (STARI)

- *Tipuri de cicluri masina: fetch (extragere instr.), RD/WR memorie, RD/WR port, cerere-achitare bus, cerere-achitare intrerupere mascabila, cerere-achitare intrerupere nemascabila, ciclu de initializare (dupa RESET), ciclu de iesire din HALT)*

- orice instructiune este o combinatie de ciclii masina si incepe cu un ciclu FETCH – extragerea codului instructiunii din memorie

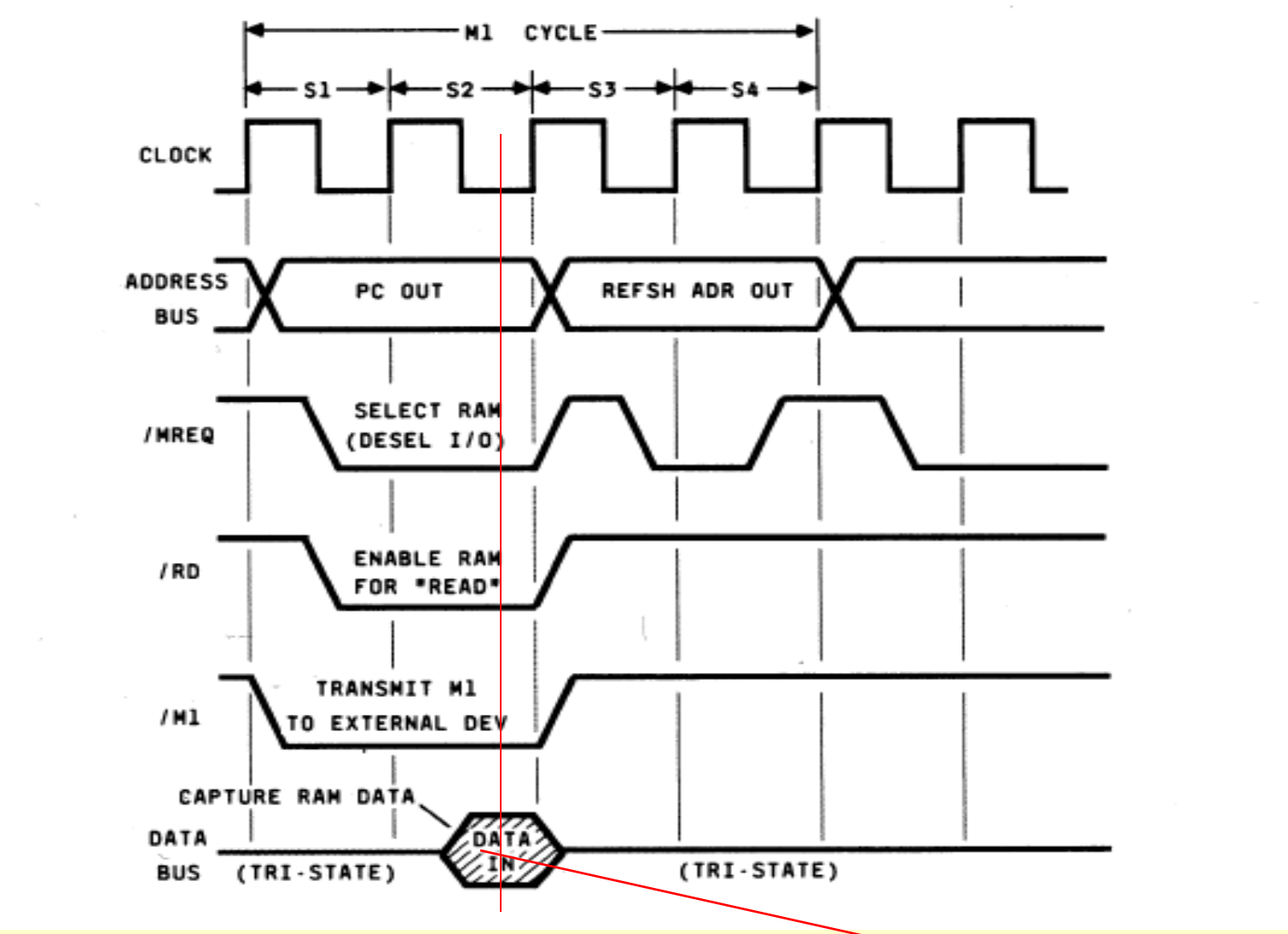
Ciclu executie instructiune



- Fetch
- Decode
- Fetch operands
- Execute
- Store output

3

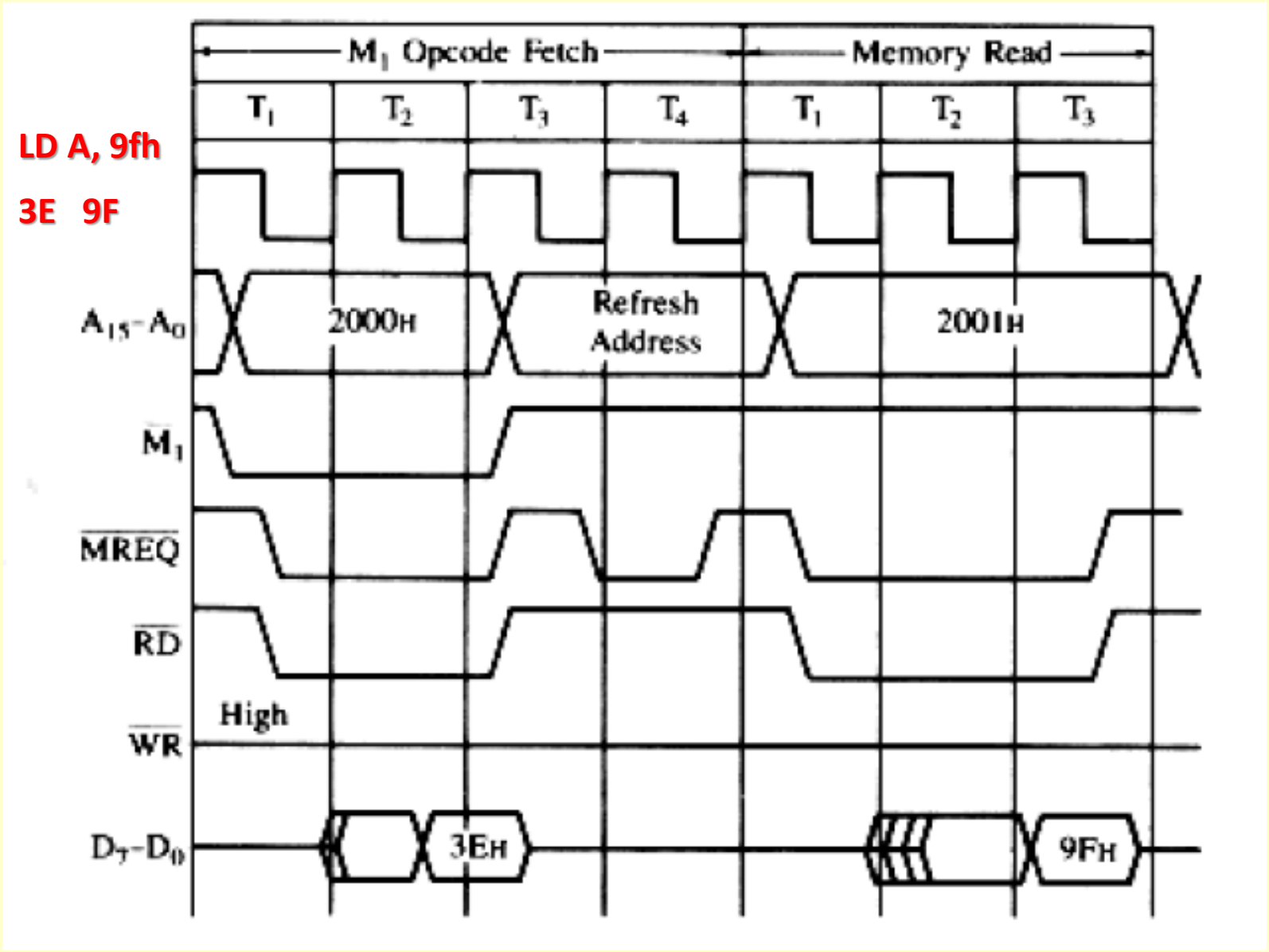
ciclu FETCH =M1 (Z80)

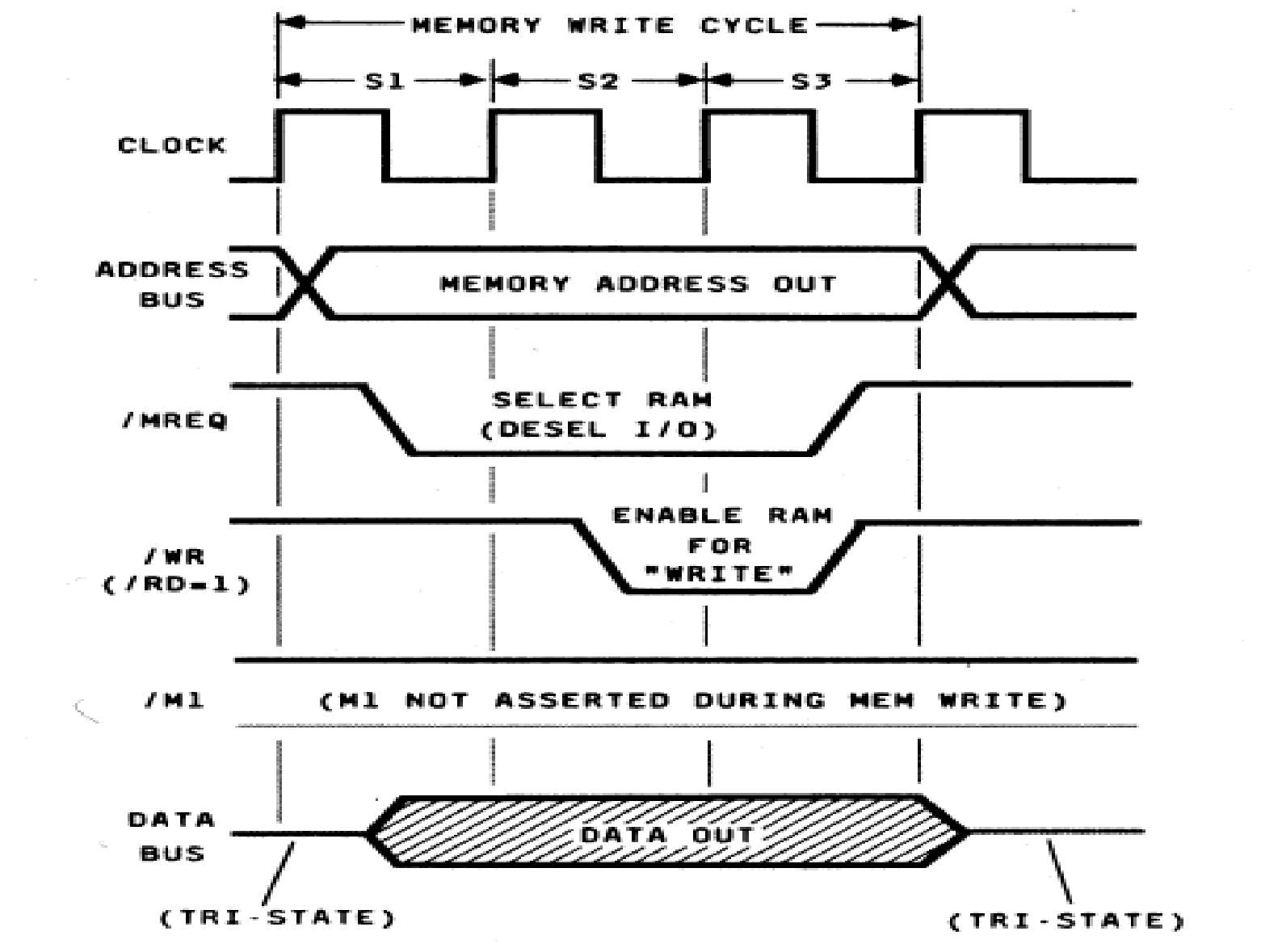


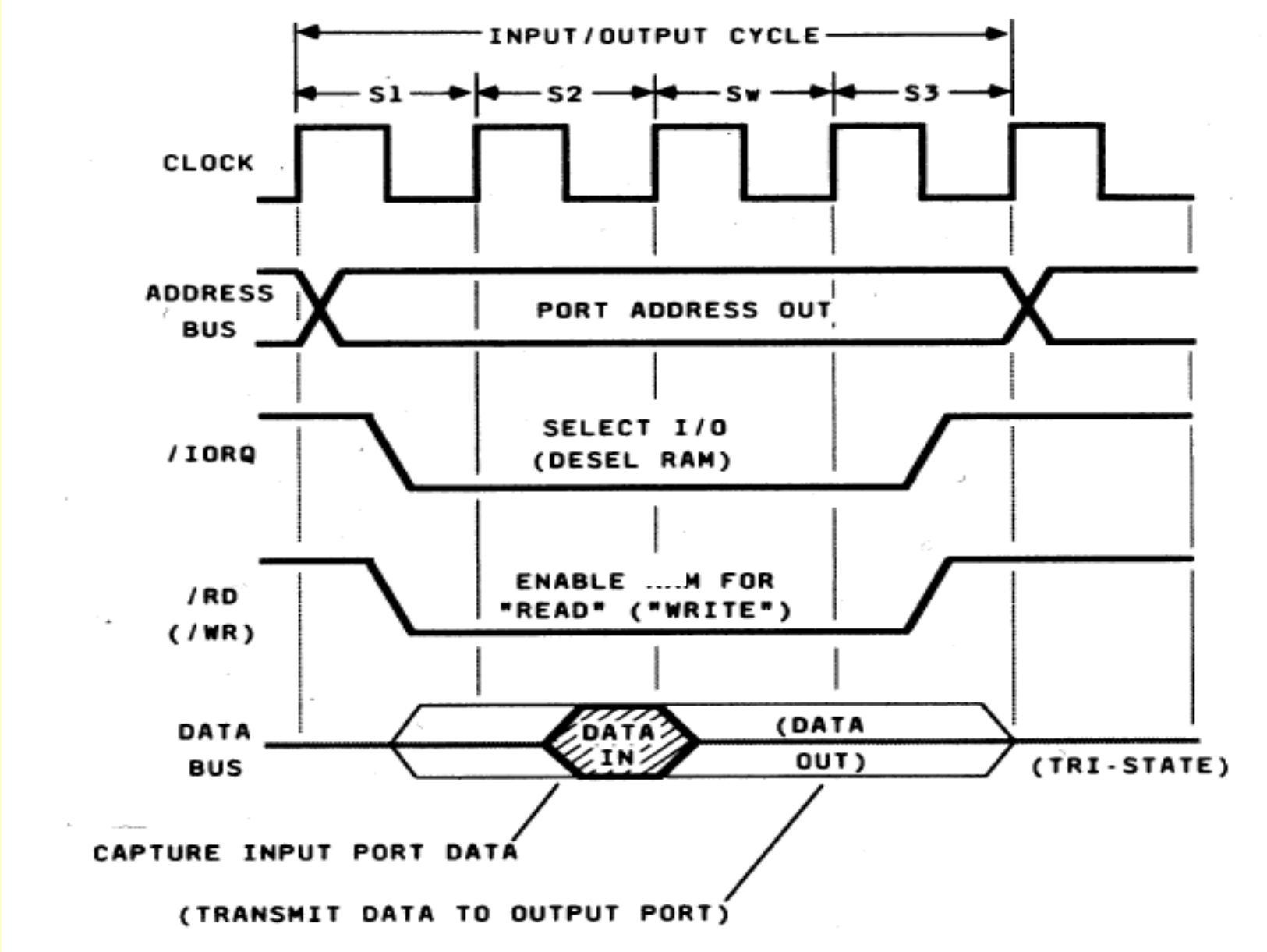
Zilog - Limbaj Asamblare LD A, 9Fh **=> Limbaj Masina** 3E 9F

LD A, 9fh

3E 9F


M1 Opcode Fetch
Memory Read
T1
T2
T3
T4
T1
T2
T3
A15-A0
2000H
Refresh Address
2001H
M1
MREQ
RD
High
WR
D7-D0
3EH
9FH


MEMORY WRITE CYCLE
S1
S2
S3
CLOCK
ADDRESS BUS
MEMORY ADDRESS OUT
/MREQ
SELECT RAM
(DESEL I/O)
/WR
(/RD=1)
ENABLE RAM
FOR
"WRITE"
/M1
(M1 NOT ASSERTED DURING MEM WRITE)
DATA BUS
DATA OUT
(TRI-STATE)
(TRI-STATE)


INPUT/OUTPUT CYCLE
S1
S2
Sw
S3
CLOCK
ADDRESS BUS
PORT ADDRESS OUT
/IORQ
SELECT I/O
(DESEL RAM)
/RD
(/WR)
ENABLE ....M FOR
"READ" ("WRITE")
DATA BUS
DATA IN
(DATA OUT)
(TRI-STATE)
CAPTURE INPUT PORT DATA
(TRANSMIT DATA TO OUTPUT PORT)

2. Metode de control a transferului de date

- Memoria pastreaza programe/date, dar are capacitate limitata

- recurgem la I/O (porturi) pentru a incarca si alte programe/date

- conexiunile pot fi seriale/paralele

- **Problema**

- sincronizarea µP cu perifericele – adaptarea vitezelor diferite de lucru

METODE de CONTROL UTILIZATE

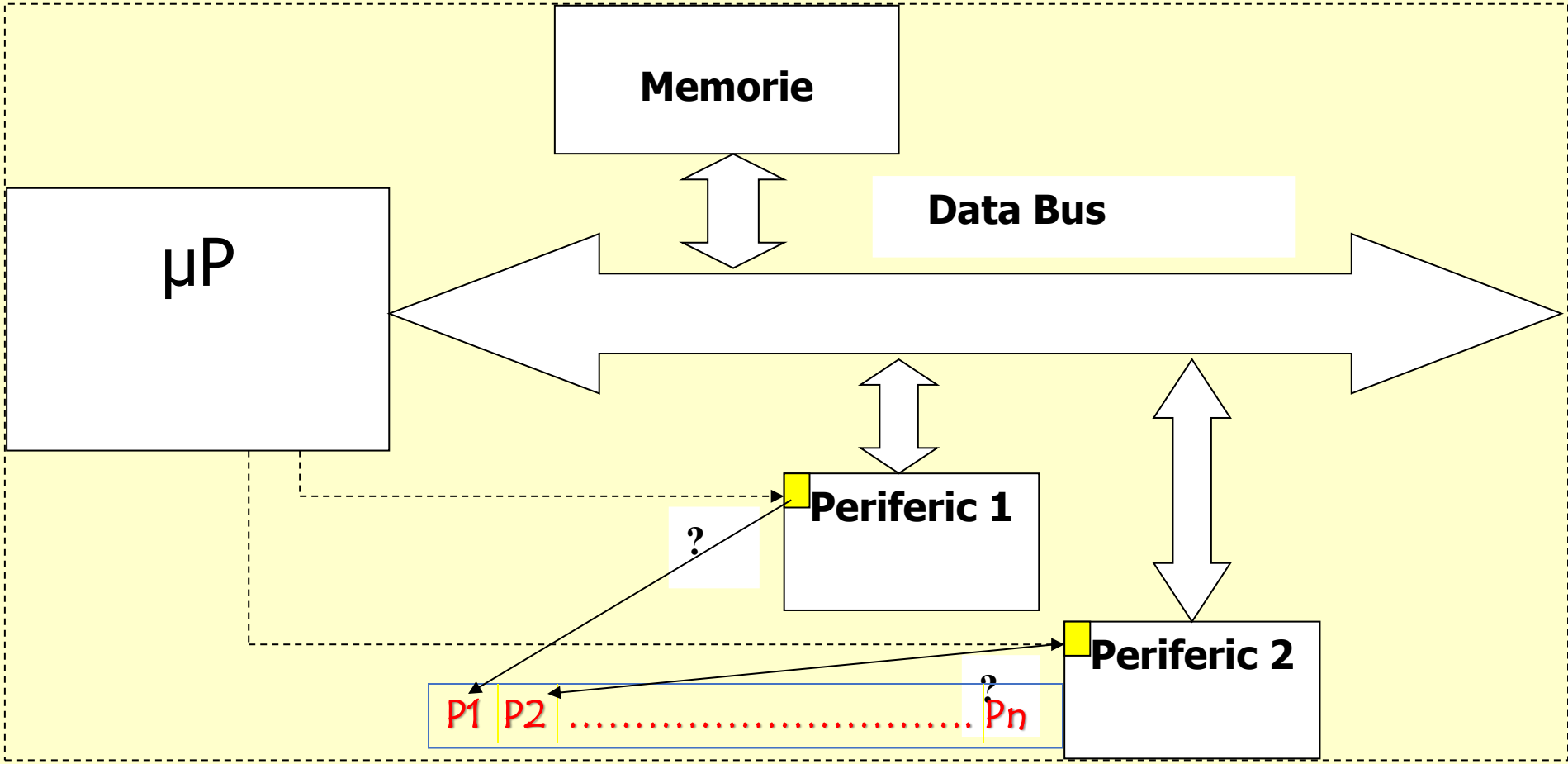
I. **POLLING** (interogare, program, I/O programate)

II. **INTRERUPERI**

III. **DMA** (Direct Memory Access)- acces direct la memorie sau combinatii ale lor

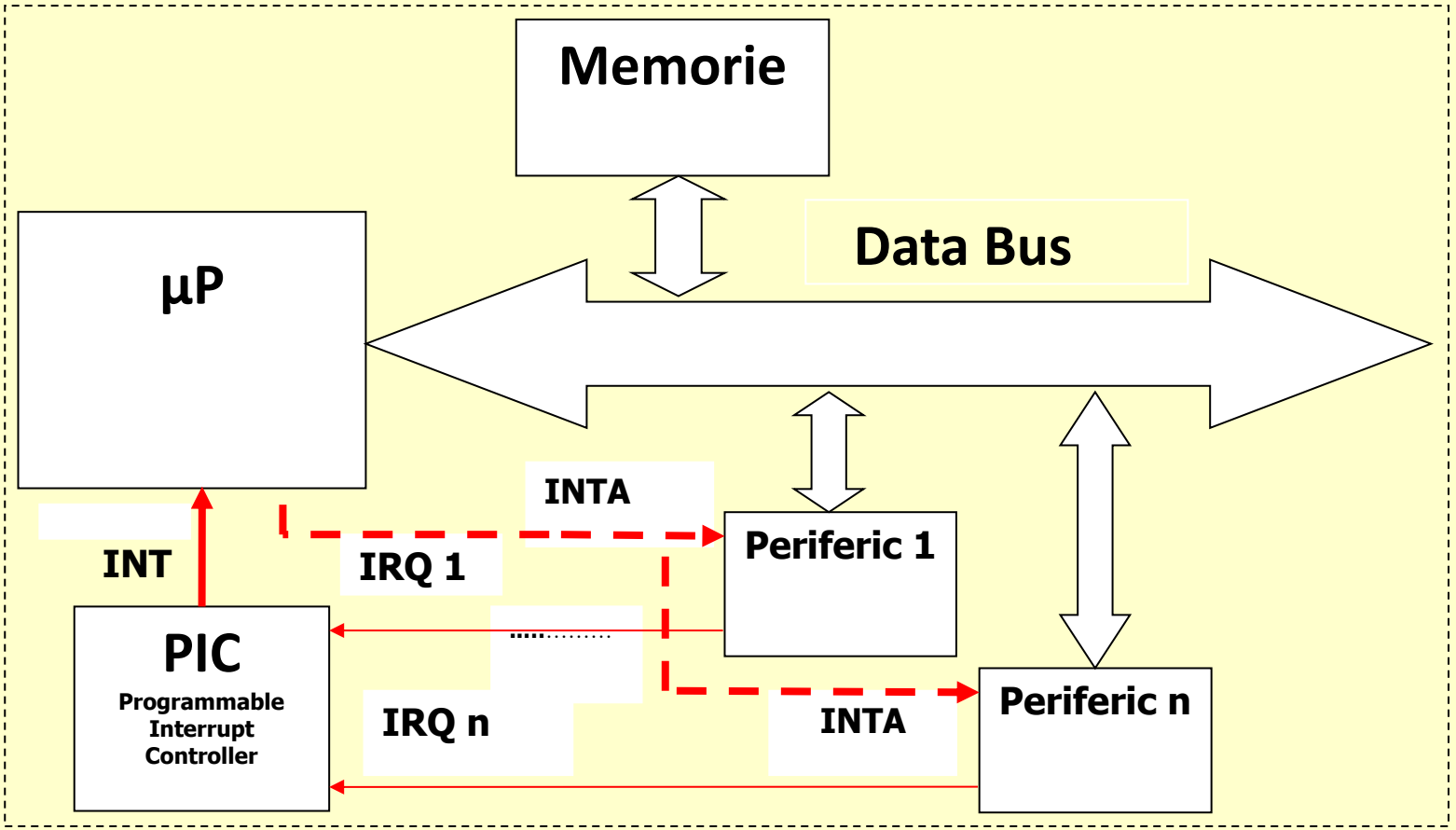
a. I/O programate (polling)

- simpla, sincrona, ieftina, nu necesita hard special
- Viteza de raspuns scazuta, ocupa procesorul 100%!!



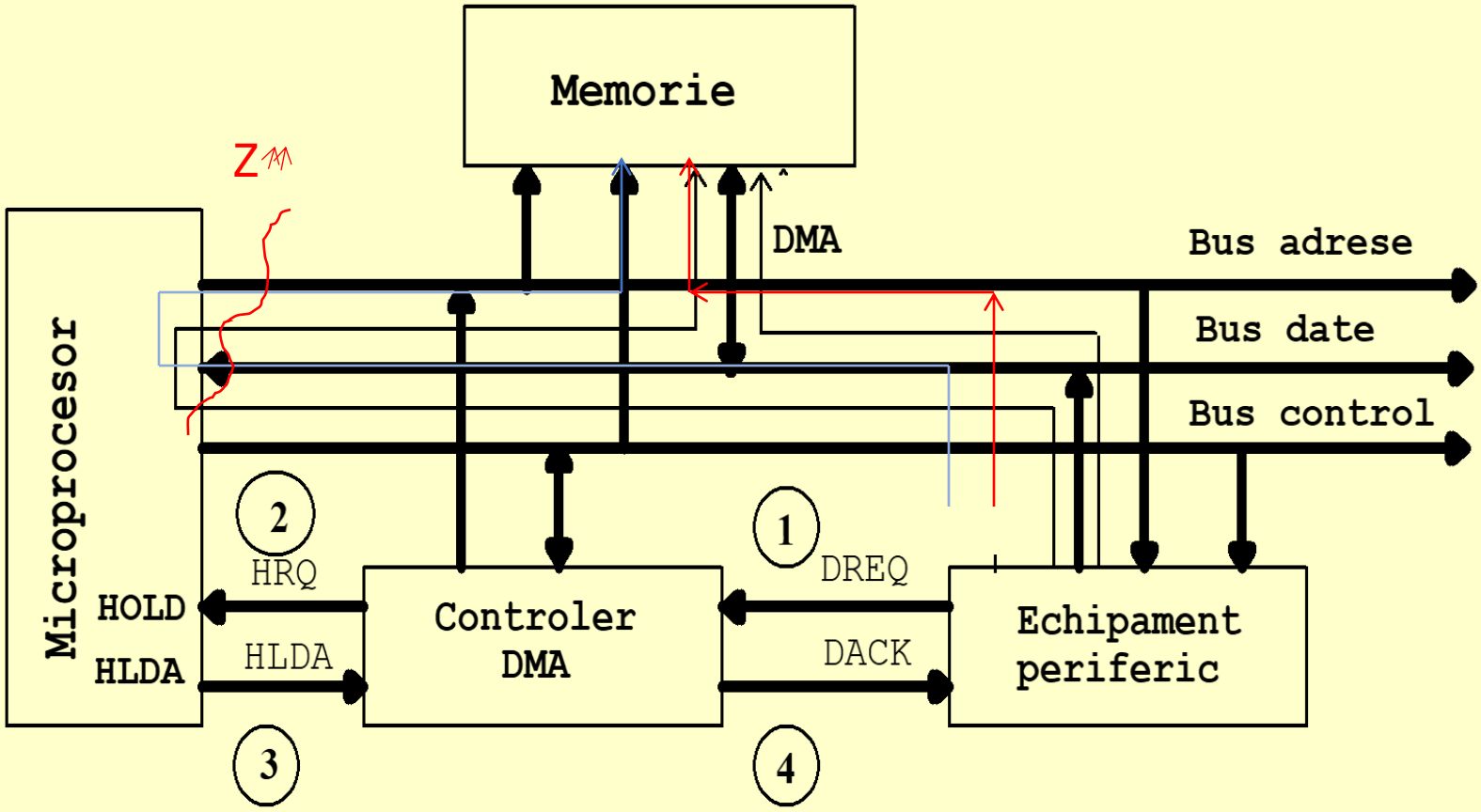
b. Intreruperi Hardware

- Metoda asincrona, are mai multi pasi/etape, prioritati



c. DMA (direct memory access) – acces direct la memorie

- Pentru periferice rapide : comprima intr-un ciclu MRD+I/O WR sau I/O RD + MWR



- Secventa de cod care transfera date de la un port de intrare in memorie

```
READ_BYTE:                          ;tacti
            IN    AL, DX            ;[13] citire port
            MOV[BX], AL             ;[2] scrie val. memorie
            INC  BX                 ;[2] BX=BX+1
            DEC CL                  ;[2] CL=CL-1
            JNZ  READ_BYTE          ;[10] sar daca CL nu e 0
```

- Aceasta secventa dureaza 29 ciclii de ceas, la 20MHz:

  fclk = 20MHz; Tclk = 1/fclk = 50ns;

  29 x 50ns = 1450ns = 1.45µs /byte

Rata de transfer poate ajunge la : 1/(1.45µs/B) = 670KB/s (lent)

- Prin transfer DMA se poate atinge ~ 10MB/s, la aceiasi frecventa de ceas

3. Taxonomii ale arhitecturilor de prelucare

- Flynn [1966]
- Feng [1972]
- Händler [1977]
- Moderne (Sima, Fountain & Kacsuk)

- Instruction-Level Parallelism (ILP) - Executarea operațiilor/instrucțiunilor independente dintr-un flux de instrucțiuni în paralel (pipelining, superscalar, VLIW)

- Thread-Level Parallelism (TLP)- Execută fluxuri de instrucțiuni independente, în paralel (multithreading, nuclee multiple)

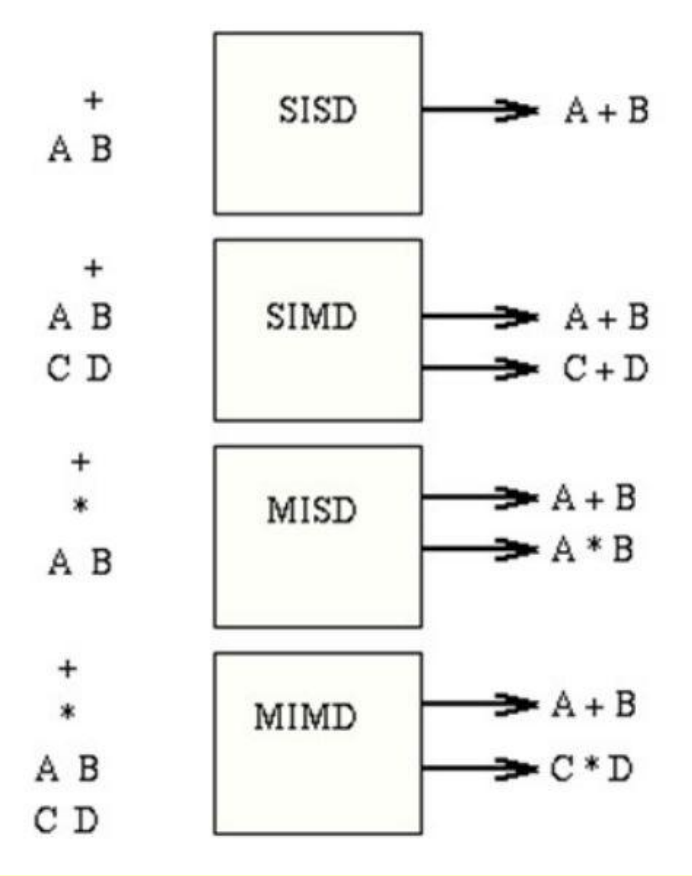
- Data-Level Parallelism (DLP)
  Executa mai multe operații de același tip în paralel (execuție vectorială/SIMD)

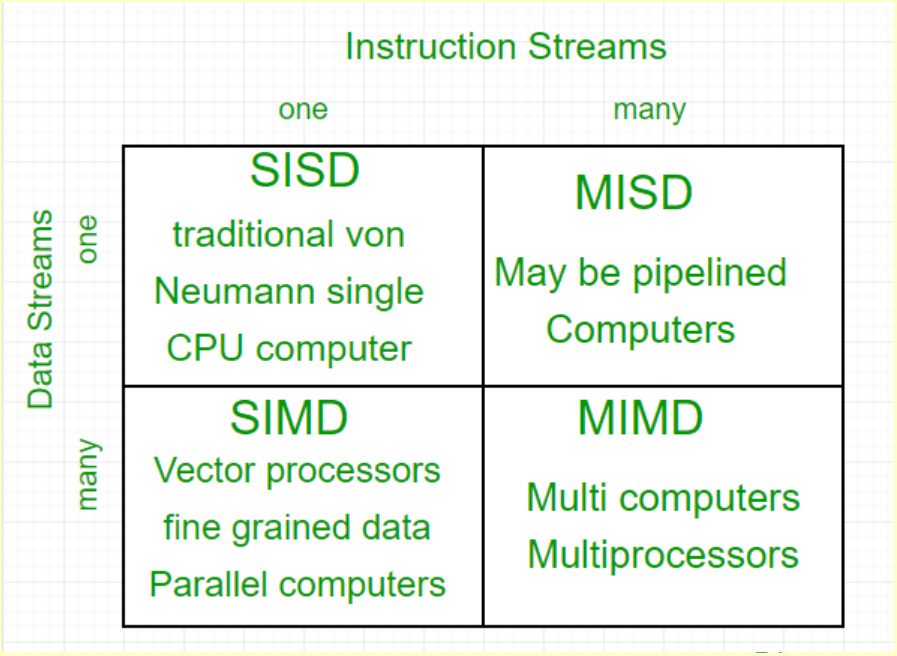
3.1. Clasificarea dupa Flynn

- Se bazeaza pe multiplicitatea *sirului de date sau instructiuni primite* de CPU pe parcursul executiei unui program

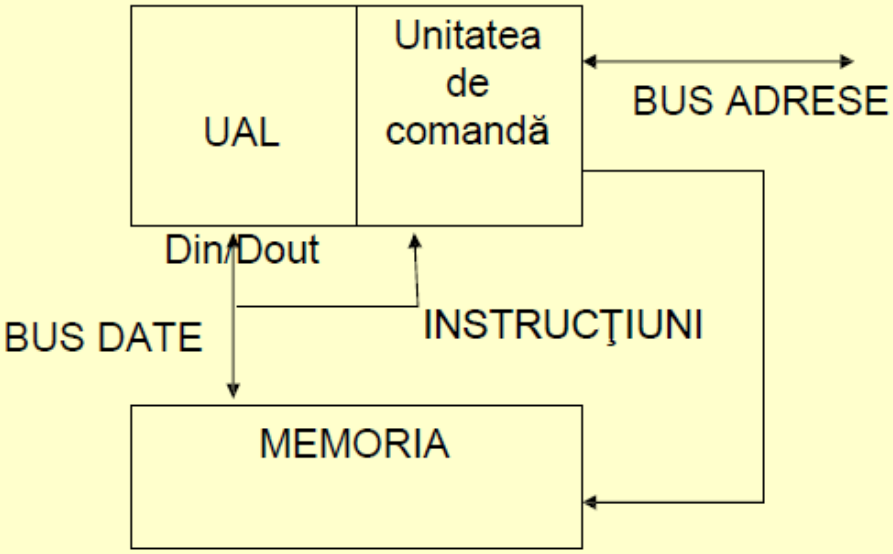
Tipuri de Arhitecturi

1) Single Instruction and Single Data stream (SISD)
2) Single Instruction and multiple Data stream (SIMD)
3) Multiple Instruction and Single Data stream (MISD)
4) Multiple Instruction and Multiple Data stream (MIMD)



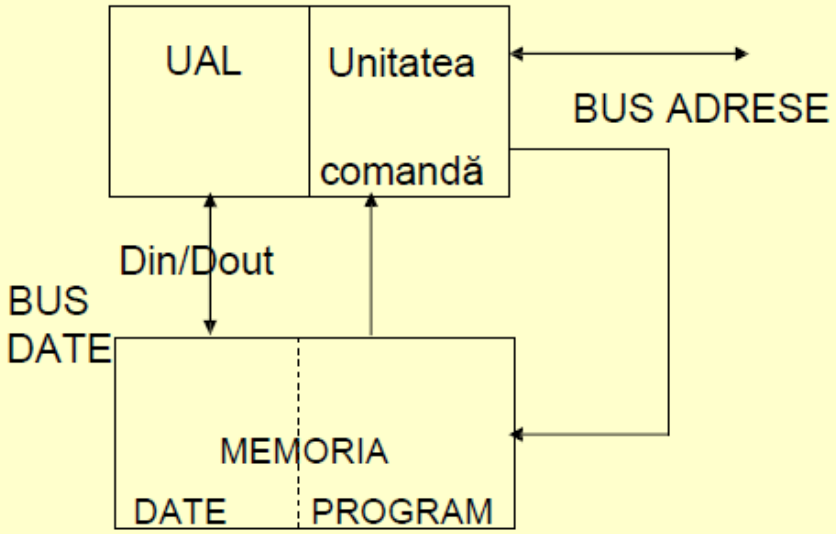


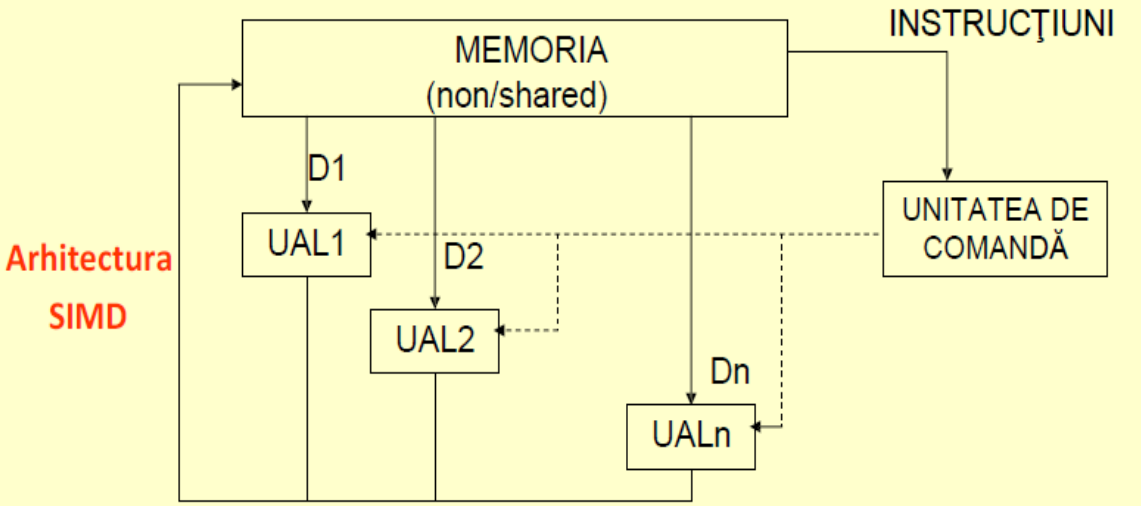
Clasificarea arhitecturilor de prelucrare - dupa Flynn

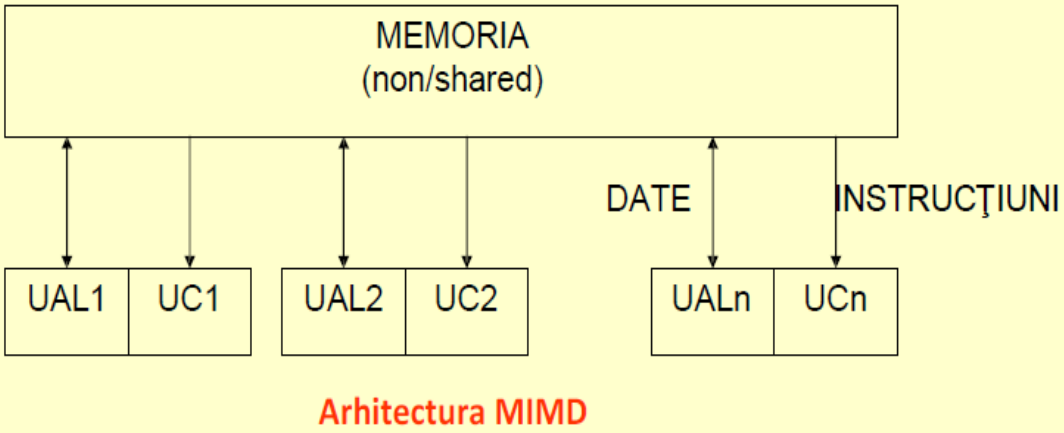


Arhitectura SISD (von Neumann)

Arhitectura HARVARD







Arhitectura MIMD

!!! TOP500 supercomputers sunt bazate pe arhitectura MIMD

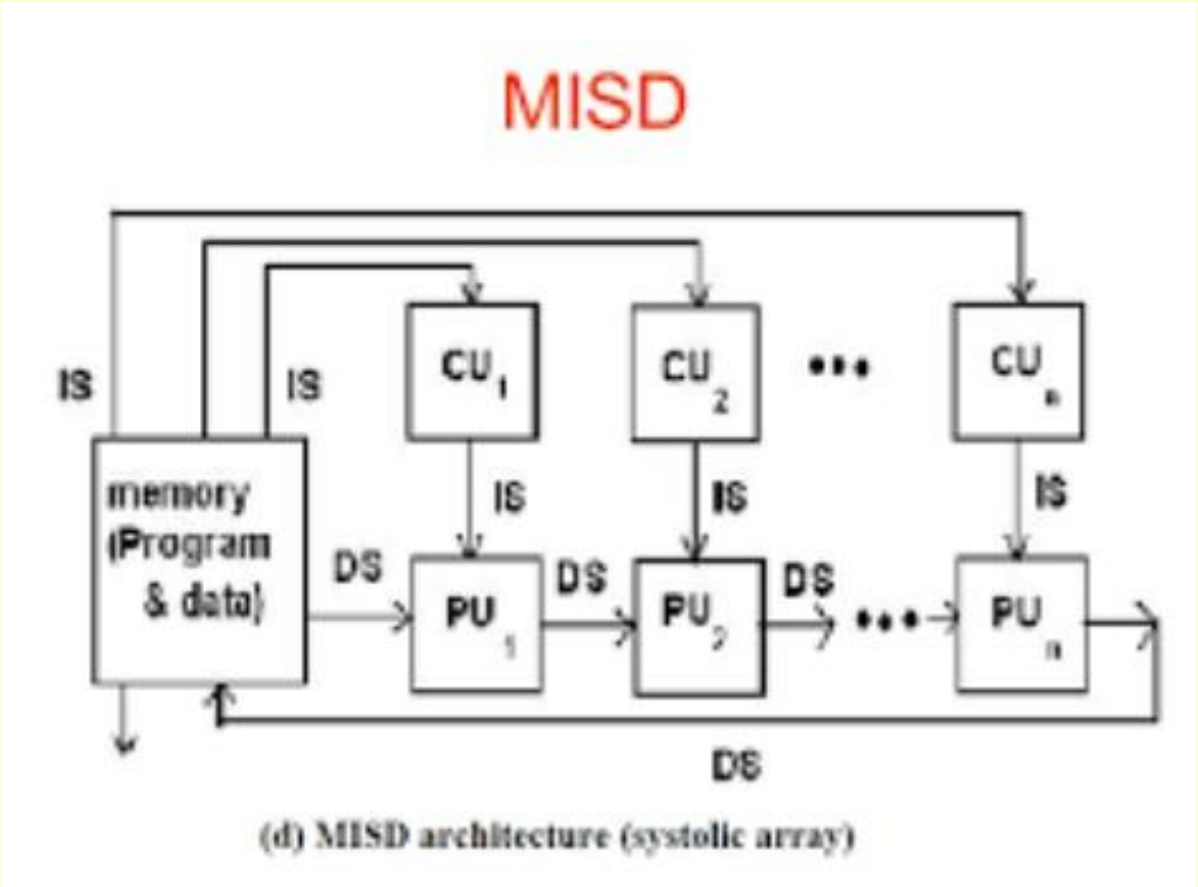
- **Single instruction, multiple thread (SIMT)** este un model de executie folosit in procesarea paralela unde SIMD este combinata cu multithreading.
- SIMT a fost introdusa de Nvidia

- Subcategoriile SIMD din taxonomia lui Flynn includ ***Array Processing și SIMT***, care reprezintă evoluții ale modelului Single Instruction, Multiple Data pentru procesare masivă paralelă pe date uniforme.
- Array Processing folosește matrici de procesoare elementare sincronizate, în timp ce SIMT (Single Instruction, Multiple Threads) extinde conceptul la execuție în lock-step pe thread-uri cu memorie locală, fiind baza arhitecturilor GPU moderne precum NVIDIA CUDA.
- Array Processing în SIMD - Reprezintă procesoare în care o instrucțiune unică controlează un array bidimensional de Processing Elements (PE), fiecare cu memorie locală, executând operații simultan pe vectori mari.

Caracteristici cheie: Masking pentru activarea selectivă a PE-urilor; interconectare mesh sau hypercube; procesare în timp constant pentru vectori mari (ex. ILLIAC IV cu 64 PE).
Avantaje: Eficiență pe operații matriciale (matrix multiply, FFT); scalabil hardware-wise.
Ex.: Cray-1 vector processor, attached array processors pentru calcul numeric.

- SIMT (Single Instruction, Multiple Threads)

Extensie modernă SIMD, unde o instrucțiune se aplică unui grup de thread-uri în lock-step, dar fiecare thread are program counter și memorie privată, permițând divergență controlată prin predicate.
Caracteristici cheie: Execuție warp (32 thread-uri pe NVIDIA); reconvergenta la sfârșitul blocurilor condiționale; diferă de SIMD clasic prin suport pentru stack pointer individual pe thread.
Avantaje: Flexibilitate mai mare decât SIMD pur (gestionare date independente); optimizat pentru grafică și ML.
Ex.: GPU-uri CUDA, OpenCL kernels; ILLIAC IV precursor cu predicate masking.



Ex. Z = sin(x)+cos(x)+tan(x)

Arhitectura MISD

- ➔ Mai mult o configuratie teoretica decat una practica
- ➔ mașinile MISD pot fi aplicate la calculatoare de timp real tolerante la erori
- ➔ Foarte putine abordari, fara produse comerciale
- ➔ (Ex: C.mmp - Carnegie-Mellon University.)

Comparație Tehnică

Categorie	Paralelism Principal	Exemple Hardware	Aplicații Optime	Limitări
SISD	Instrucțiuni pipeline	Procesor scalar	Sarcini secvențiale	Fără paralelism nativ geeksforgeeks
SIMD	Date vectoriale	GPU, AVX	Imagine, ML	Date regulate wikipedia
MISD	Instrucțiuni redundante	Pipeline fault-tolerant	Securitate	Rar implementat doc.sling
MIMD	Task-uri independente	Multicore, clustere	General parallel	Overhead sincronizare abhik

Comparație Sintetică

Clasă	Avantaj Principal	Dezavantaj Principal	Aplicații Tipice
SISD	Simplitate, cost scăzut	Fără paralelism nativ	Software serial geeksforgeeks
SIMD	Eficiență date masive	Date neregulate	GPU, procesare media wikipedia
MISD	Toleranță erori	Implementări rare	Sisteme fault-tolerant doc.sling
MIMD	Flexibilitate scalabilă	Complexitate programare	Supercomputere, cloud abhik

- Clasele din taxonomia lui Flynn (SISD, SIMD, MISD, MIMD) prezintă avantaje și dezavantaje distincte, determinate de capacitatea de paralelism, scalabilitate și tipul de aplicații suportate.

Aceste caracteristici influențează alegerea arhitecturii în funcție de sarcinile specifice, de la procesare secvențială simplă la calcul de înaltă performanță.

SISD (Single Instruction, Single Data) - arhitectura clasică secvențială, bazată pe un singur flux de instrucțiuni și date.

•Avantaje: Simplitate de implementare și programare; cost redus; eficientă pentru sarcini seriale fără paralelism; predictibilă în execuție.

•Dezavantaje: performanță limitată pe date mari; nu exploatează paralelismul modern; viteză dependentă de transferul intern de date.

SIMD (Single Instruction, Multiple Data)- Ideal pentru operații uniforme pe vectori sau matrici, comun în GPU

•Avantaje: eficiență ridicată pe date paralele (imagine, ML); utilizare optimă a memoriei; scalabil cu numărul de elemente procesate simultan (ex. AVX, CUDA).

•Dezavantaje: Necesită date regulate și instrucțiuni identice; ineficient pe cod ramificat (branching); programare complexă prin vectorizare.

MISD (Multiple Instruction, Single Data) - Rară, orientată spre redundanță și pipeline-uri specializate.

•Avantaje: toleranță excelentă la erori prin procesare multiplă; fiabilitate în sisteme critice (fault-tolerance); utilă în aplicații pipeline (ex. procesare semnal).

•Dezavantaje: Puțin practică; overhead mare de sincronizare; rar implementată în hardware comercial; eficiență scăzută pe sarcini generale.

•MIMD (Multiple Instruction, Multiple Data) - Cea mai versatilă, dominantă în sisteme multicore și clustere.

•Avantaje: flexibilitate maximă pentru task-uri independente; scalabil la mii de procesoare; suportă shared/distributed memory; ideal pentru HPC și aplicații generale.

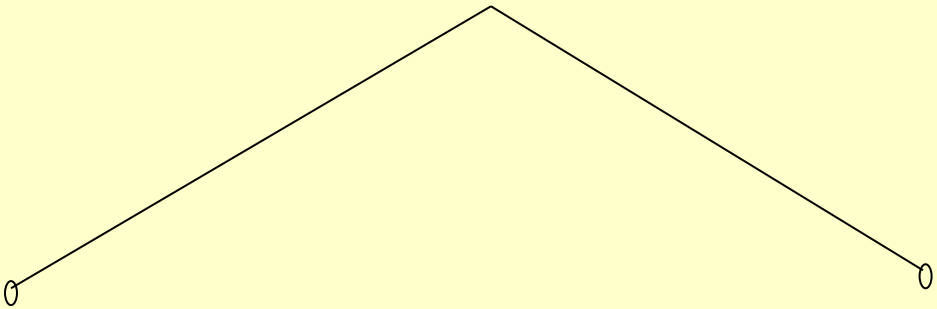
•Dezavantaje: programare dificilă (sincronizare, race conditions); overhead de comunicație în distributed memory; vulnerabil la bottleneck-uri de memorie.

Diferențe și Relații

- Array Processing este SIMD "pur" cu hardware dedicat (mai multe ALU sincronizate), în timp ce SIMT este SIMD "software-managed" pe multiprocesoare streaming, combinând elemente MIMD-like prin divergență.

Aspect	Array Processing (SIMD clasic)	SIMT (SIMD extins)
Unitate execuție	Processing Elements fixe	Thread-uri dinamice
Memorie	Locală shared	Locală per thread
Divergență	Masking strict	Predicate reconvergentă
Exemple	ILLIAC IV, vector processors	NVIDIA GPU warps wikipedia

3.2. Clasificari Moderne



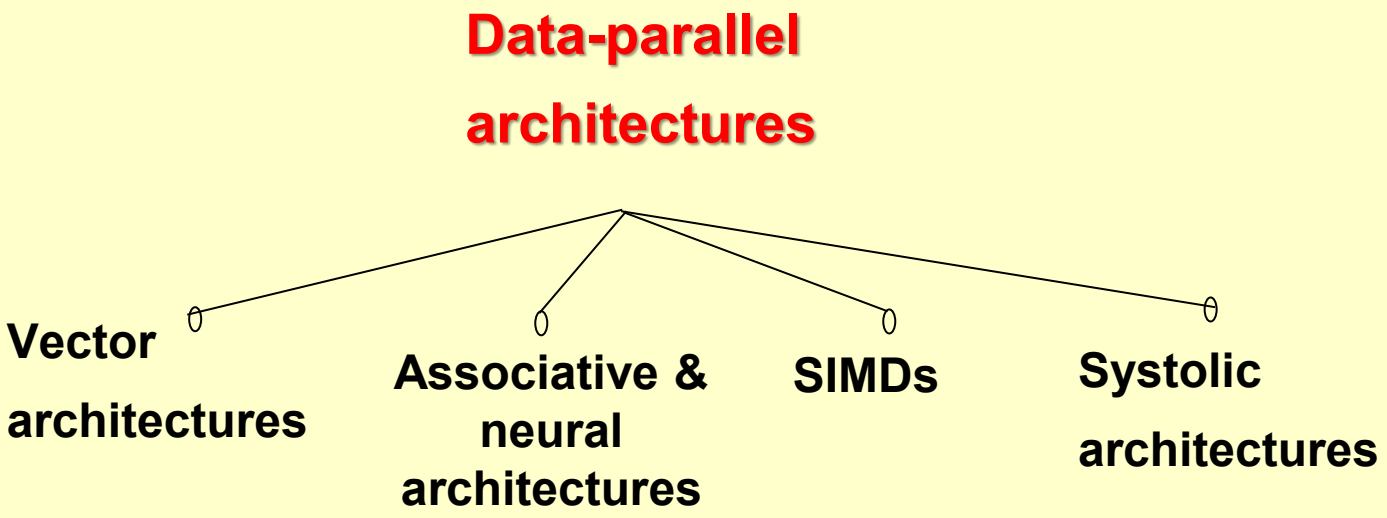
Arhitecturi de date paralele

(Data-parallel Architectures)

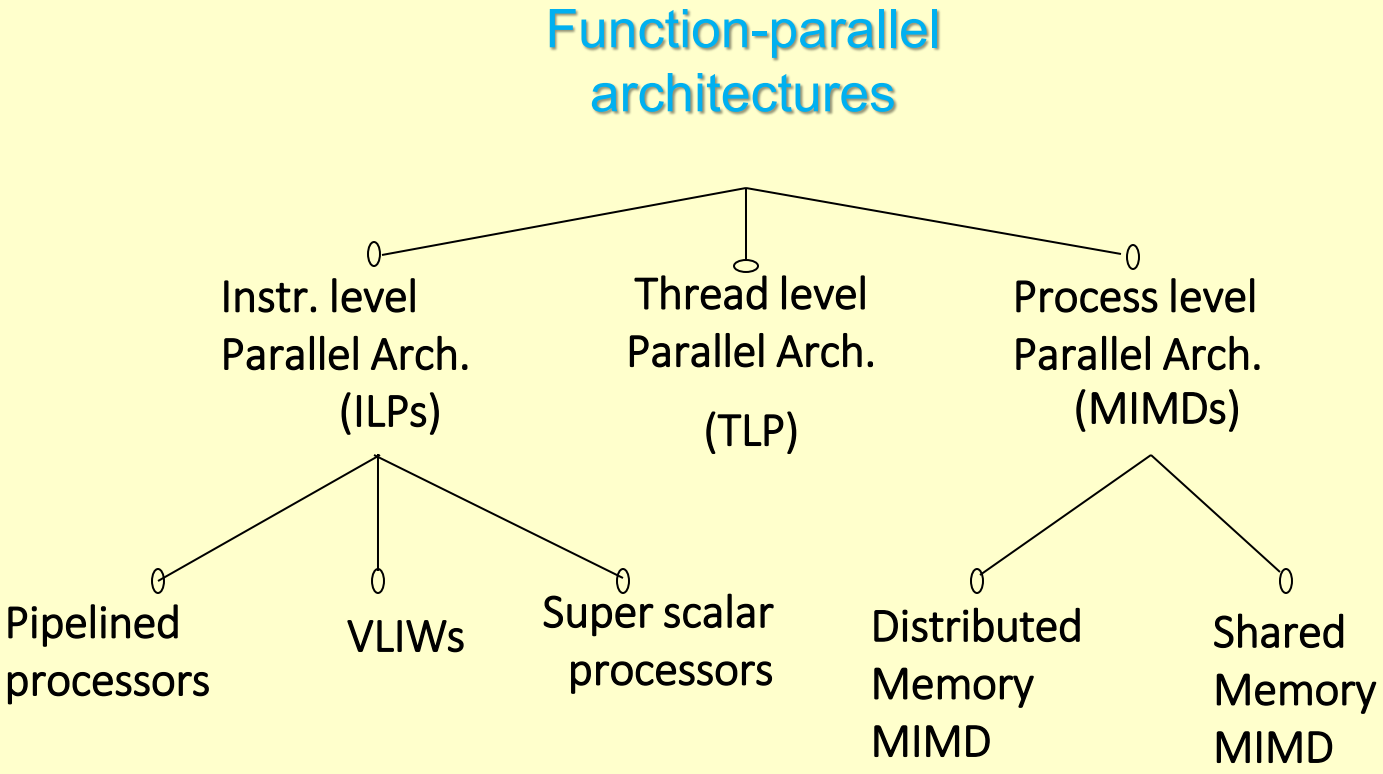
Arhitecturi Functionale paralele

(Function-parallel architectures)

Arhitecturi de Date Paralele



Arhitecturi Functionale Paralele

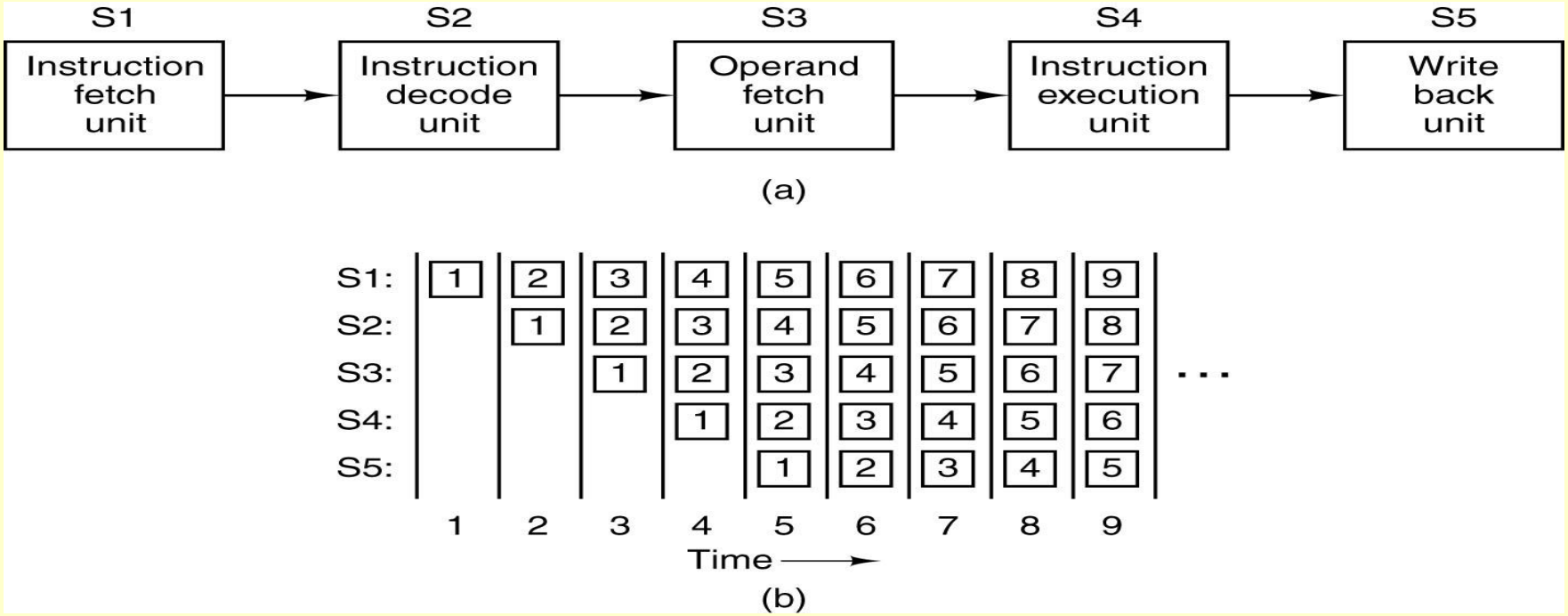


4. Paralelism - pipeline, superscalar

- Paralelism la nivel de instructiune (ILP)
  - Pipeline (o linie de asamblare): Tehnica prin care executia mai multor instructiuni este suprapusa (faze diferite)
  - Arhitecturi Superscalare:
    - Se folosesc linii de asamblare (pipeline) paralele

- Paralelism la nivel procesor/procesare :
  - Arii de calcul
  - Multiprocesoare
  - Multicomputere

a. Paralelism la nivel instructiune - Pipeline



- Pipeline de 5 nivele (exemplu: MIPS, Pentium)
- Fiecare instructiune este “sparta” in mai multe etape (faze)
- Fazele instructiunilor se pot executa concurrent/paralel, deoarece exista resurse (hard) separat pentru fiecare faza (etapa)

  => la fiecare pas se folosesc unitati functionale diferite
- Proiectare Sincrona => etapa cea mai lenta este dominanta

- ***Nota*:** Timpul de executie pentru o singura instructiune nu este imbunatatit!!!

Se imbunatateste rata de executie a instructiunilor

Cycles \ Stages	S1	S2	S3	S4	S5	S6
1	I-1					
2		I-1				
3			I-1			
4				I-1		
5					I-1	
6						I-1
7	I-2					
8		I-2				
9			I-2			
10				I-2		
11					I-2	
12						I-2

Exemplu de procesor non-pipeline.
Multi cicli pierduti!!

- Mai eficienta utilizare a ciclilor.
  O rata mai mare de executie a instructiunilor:
  (80486 incepe folosirea pipeline)

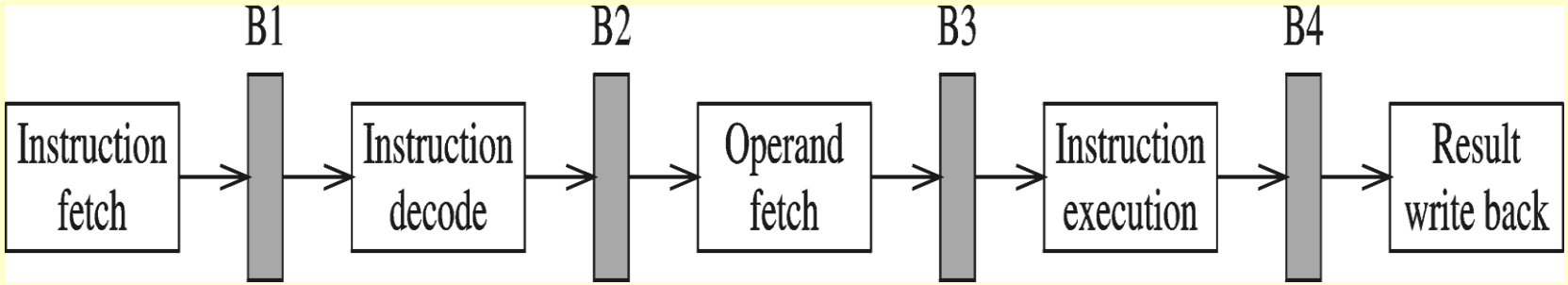
Ptr. *k* nivele si *n* instructiuni, nr. necesar de ciclii:

$k + (n - 1)$ fata de k*n

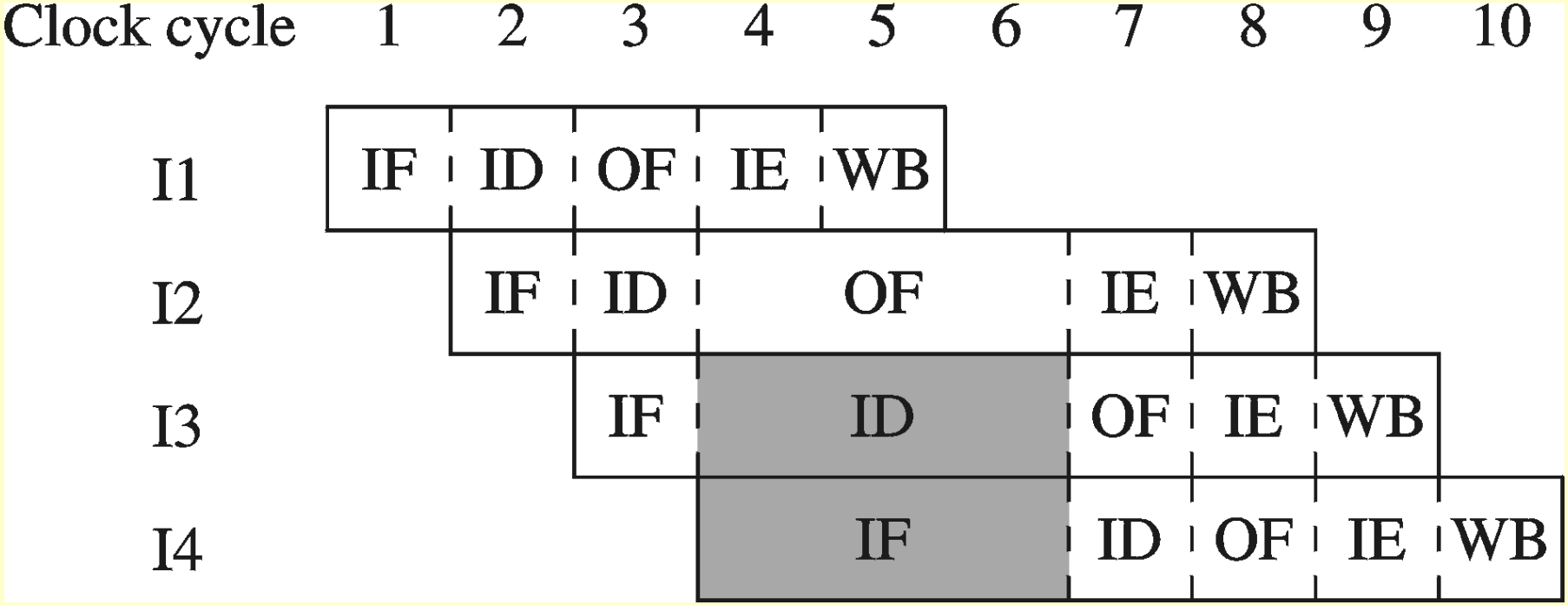
Cycles \ Stages	S1	S2	S3	S4	S5	S6
1	I-1					
2	I-2	I-1				
3		I-2	I-1			
4			I-2	I-1		
5				I-2	I-1	
6					I-2	I-1
7						I-2

Executia in Pipeline

- Pipeline-ul necesita buffere
  - Fiecare buffer pastreaza o singura valoare
  - Scenariul ideal : sarcini egale la fiecare nivel
    - Uneori nu e posibil
    - Cel mai lent nivel determina rata fluxului in intregul pipeline



- Câteva motive pentru faze de lucru (durate) inegale
  - o *faza complexa* nu poate fi subdivizata în mod convenabil
  - o operație are *durata variabilă* de timp pentru a o executa, de ex. preluarea unui operand depinde de locul unde se află :registre, m. cache, memoria DRAM
  - complexitatea operatiei depinde de *tipul de operație*
    ADD: poate lua un singur ciclu (4)
    MUL: poate dura mai multe cicli (144)



- Ciclii pierduti in pipeline
- Cand un nivel necesita doi sau mai multi tacti, ciclii sunt “risipiti”

Stages

exe

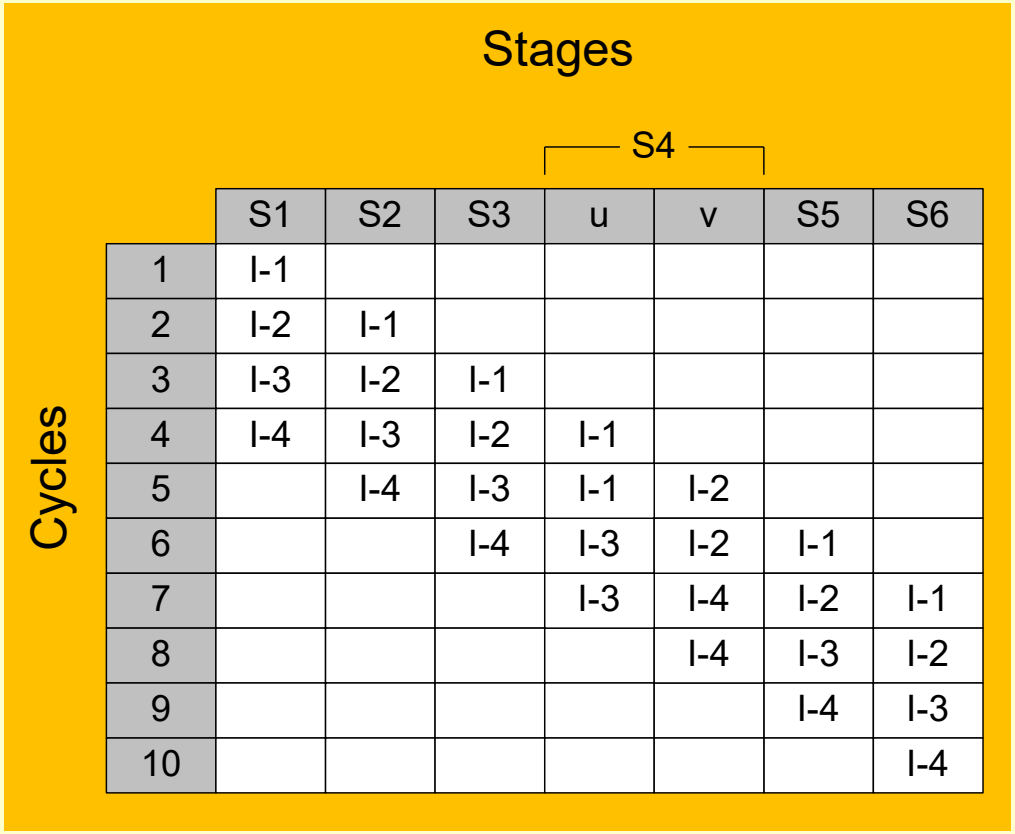
Cycles	S1	S2	S3	S4	S5	S6
1	I-1					
2	I-2	I-1				
3	I-3	I-2	I-1			
4		I-3	I-2	I-1		
5			I-3	I-1		
6				I-2	I-1	
7				I-2		I-1
8				I-3	I-2	
9				I-3		I-2
10					I-3	
11						I-3

Ptr. *k* nivele si *n* instructiuni, nr. de ciclii necesari este:
$k + (2n - 1)$ in loc de k+ (n-1)

- Performante:
  - Viteza sau banda de exec. instructiunilor (MIPS=Milioane Instructiuni/Sec.)
  - Latenta/ timp Executie –>nr. de tacti pe care le ia unei instr. ptr. a furniza date ptr. alta instructiune
- Mecanism complet hardware (pipeline)
- Toate masinile moderne folosesc tehnica pipeline
  - A fost tehnica cheie a anilor ’80 pentru imbunatatirea performantelor
  - Din anii ’90 evolutia a fost spre pipeline-uri multiple

b. Arhitecturi Superscalare

- Un procesor superscalar are pipeline-uri multiple de executie
- In fig. faza S4 are 2 pipeline-uri (u si v).



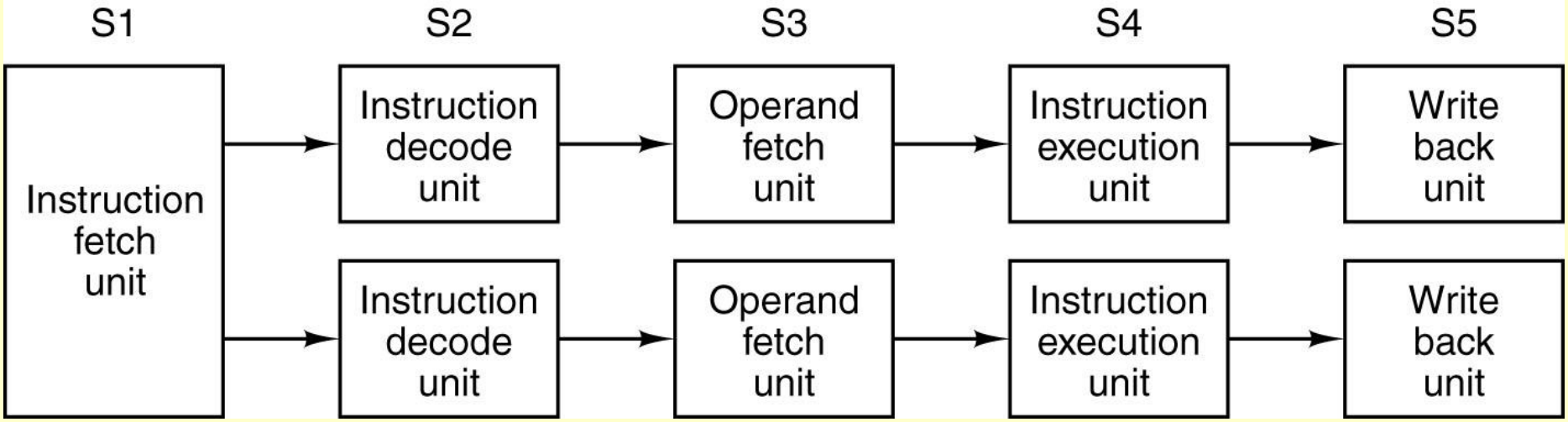
	S1	S2	S3	u	v	S5	S6
1	I-1						
2	I-2	I-1					
3	I-3	I-2	I-1				
4	I-4	I-3	I-2	I-1			
5		I-4	I-3	I-1	I-2		
6			I-4	I-3	I-2	I-1	
7				I-3	I-4	I-2	I-1
8					I-4	I-3	I-2
9						I-4	I-3
10							I-4

-Ptr. *k* stari si *n* instructiuni, numarul necesar de ciclii este : $k + n$

Pentium: 2 pipeline

Pentium Pro: 3 pipeline

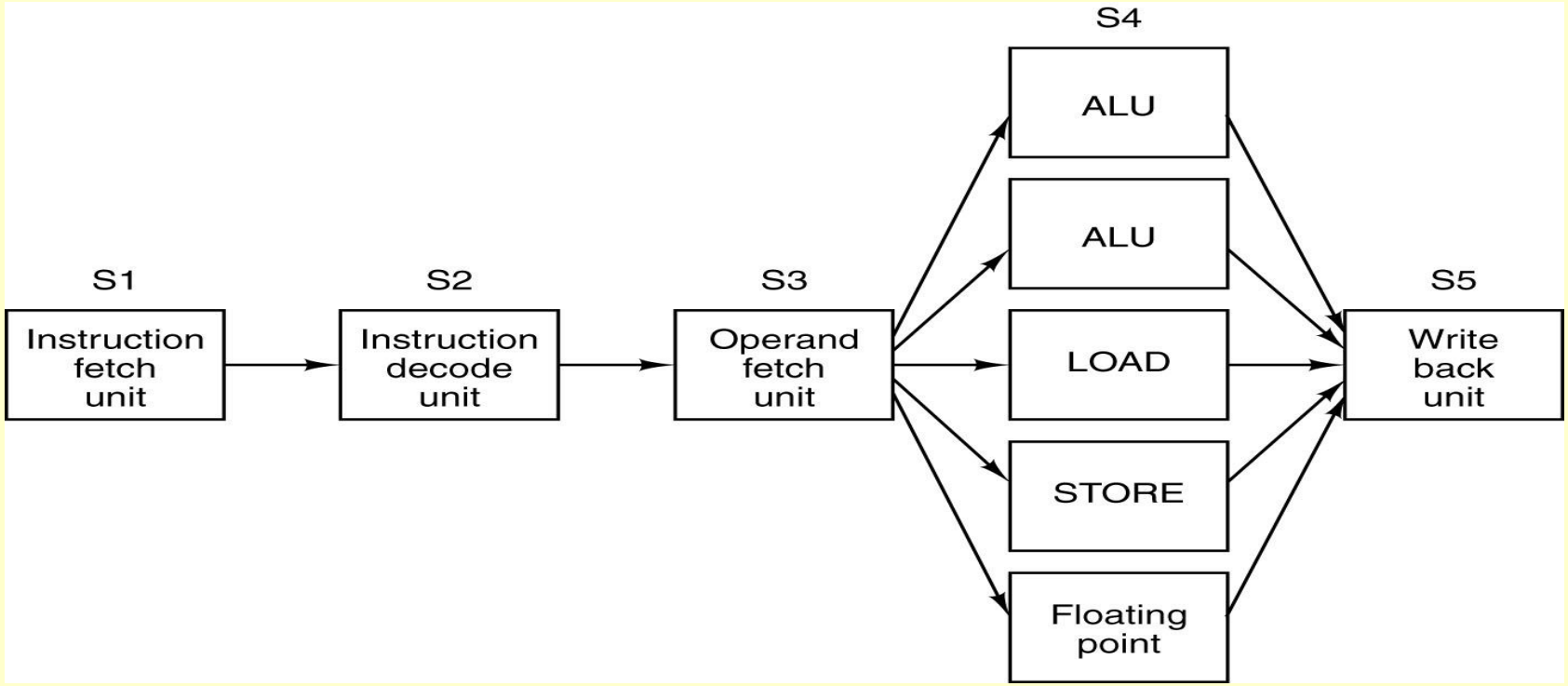
32



- Pipeline-uri duale cu 5 nivele cu unitate de FETCH comuna:
  - fiecare pipeline are hardware propriu pentru fiecare nivel, furnizand decodare si executie duplicate;
  - Atentie la instrucţiunile dependente sau incompatibile!

→Acest lucru poate fi realizat de către ***compilator*** sau sunt controlate în timpul rulării de ***hardware suplimentar.***

• De asemenea, poate fi un pipeline cu unitati functionale multiple numai la unul sau mai multe nivele ale pipeline-ului



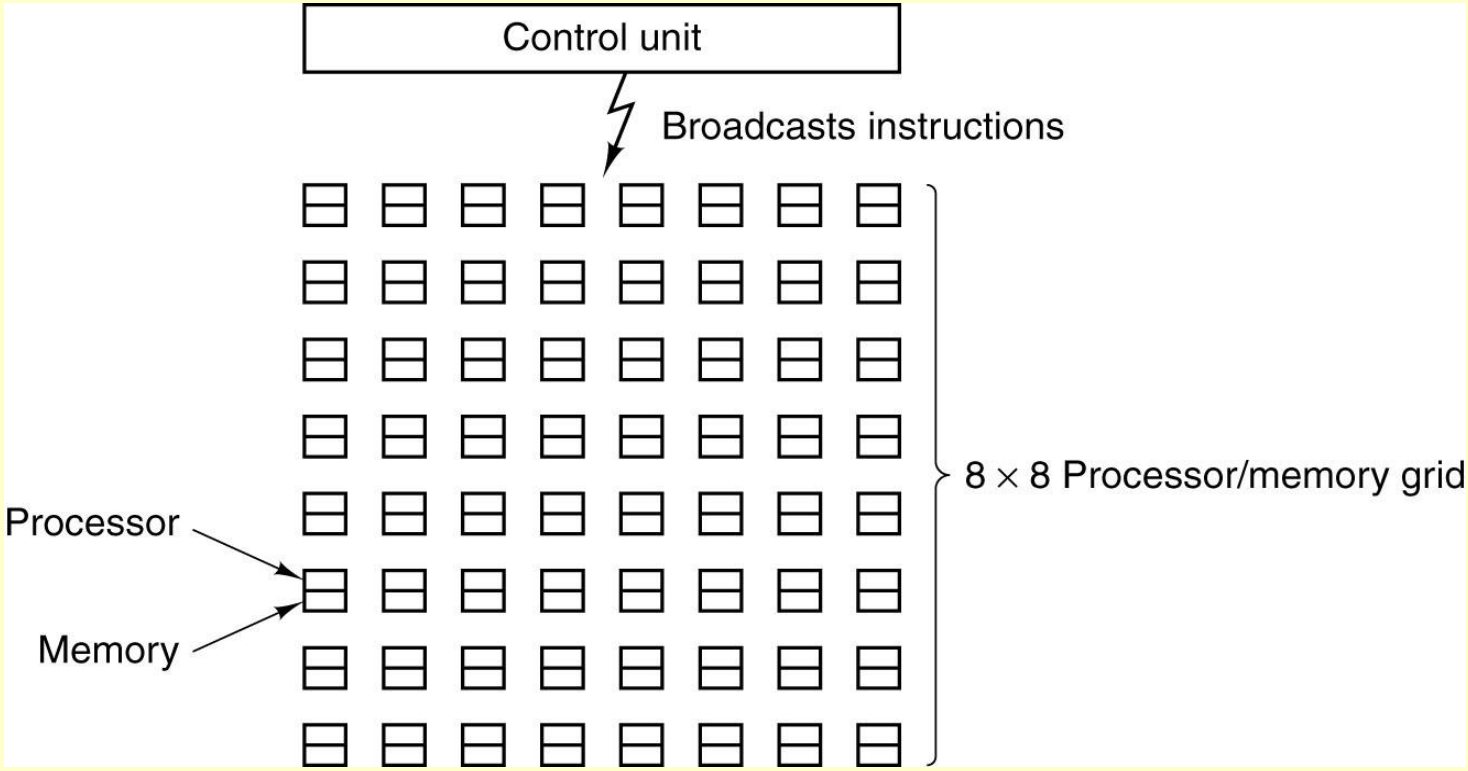
Procesor superscalar cu 5 unitati functionale in faza S4

• Definitia actuala a **arhitecturii superscalare** : un procesor care ***poate*** executa ***mai multe instrucţiuni*** pe un ciclu de ceas

c. Paralelism la nivel Procesor

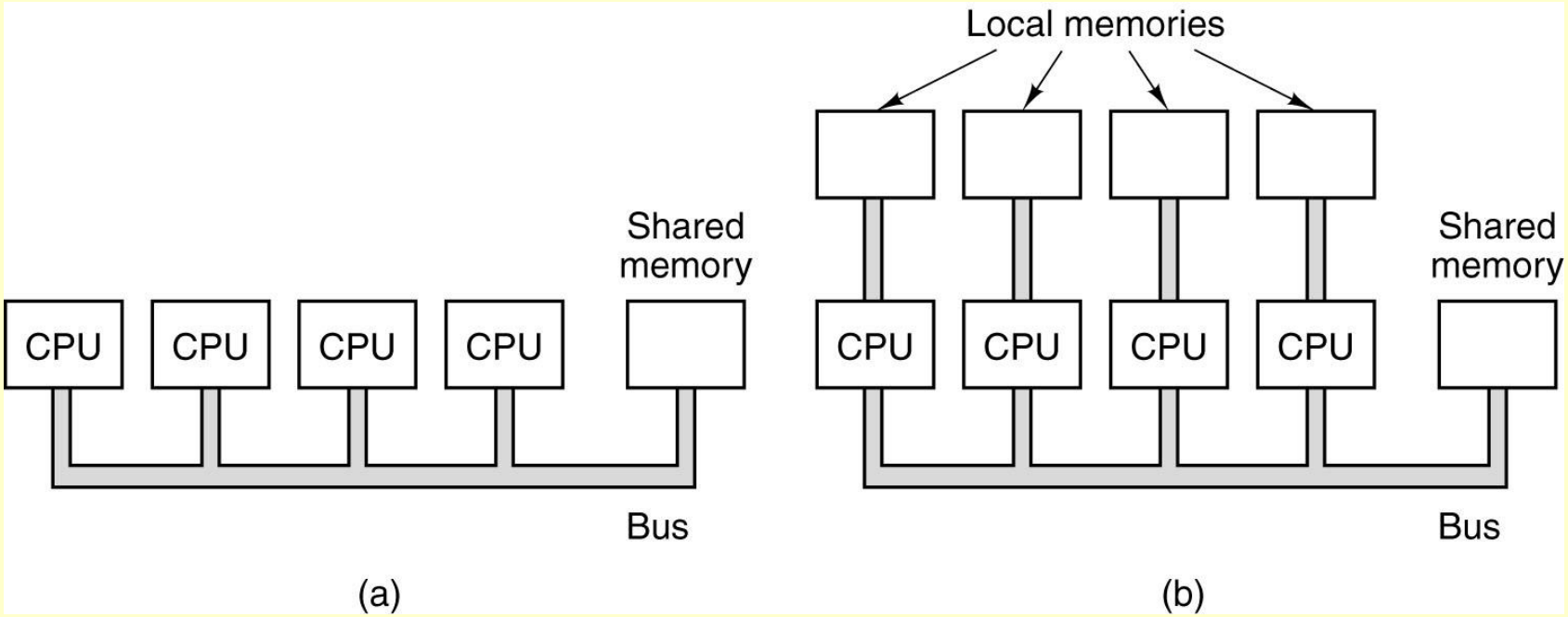
- Se refera la un calculator tip multiprocesor

  SIMD Processor: Single Instruction Multiple Data



Arie de procesare de tip ILLIAC IV

Paralelism la nivel Procesor



- Sistem Multiprocesor :

  a) *Multiprocesor cu un singur bus* : UCP multiple, independente partajaza o memorie comuna si alte resurse

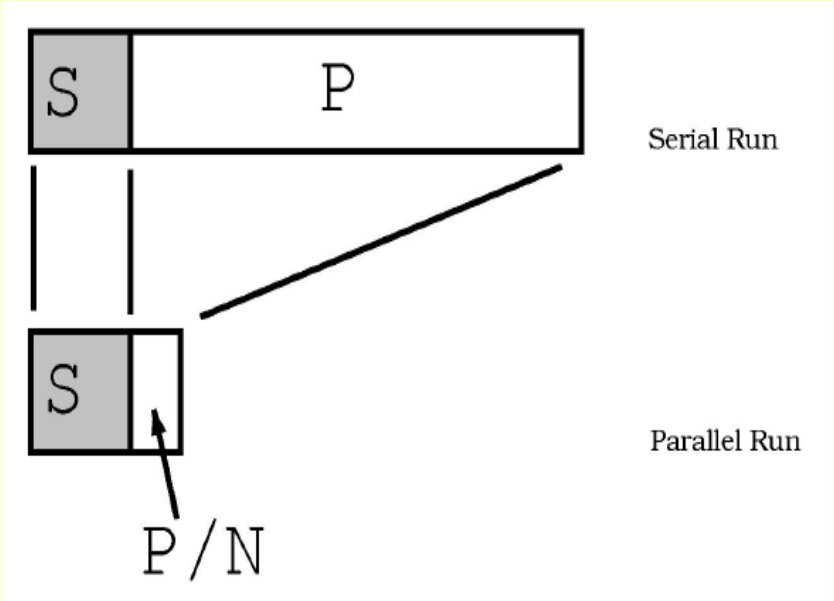
  b) *Un multi-computer cu memorii locale*:

  - Încerca să reducă numărul de conflicte - procesoarele comunica prin intermediul schimbului de mesaje

5. Legea lui Amdahl (1967 – IBM)

- **Accelerarea prelucrarii prin folosirea de unitati de procesare paralele**

$$Speedup \leq \frac{S+P}{S+P/N}$$



- S (%) - procentul de instructiuni secventiale prin natura lor
- P=1-S(%) - procentajul de instructiuni paralelizabile
- N este numarul de procesoare folosite in calculul paralel

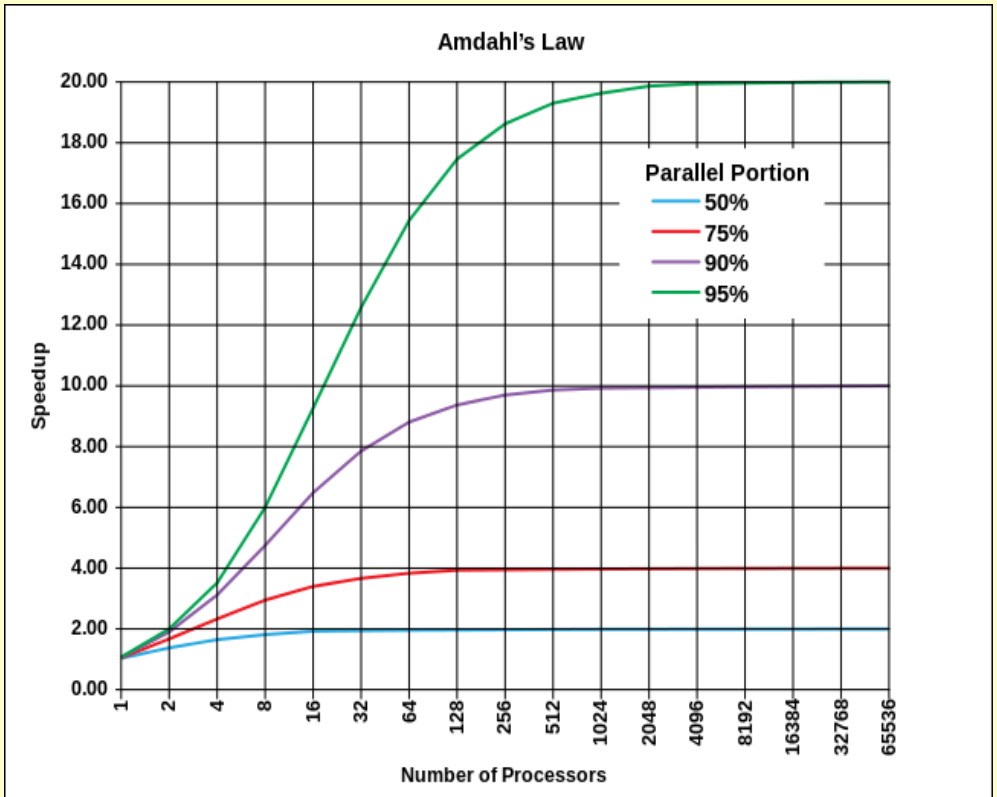
$$Speedup \le \frac{S+1-S}{S+\frac{1-S}{N}} \qquad Speedup \le \frac{1}{S+\frac{1-S}{N}}$$

Ex. N=10, 10% secvential, 90% paralel → S_p=5.26

- Dacă considerăm că problema e infinit paralelizabilă (S=0), se obtine următoarea limită superioară pentru accelerarea procesarii:

$$Speedup \le \frac{1}{S+\frac{1-S}{N}} = \frac{1}{0+\frac{1-0}{N}}$$

$$Speedup \le N$$



Exemple:

p=50%, n->∞ => S=2

p=75%, n->∞ => S=4

p=95%, n->∞ => S=20

Legea lui Lee = Legea Amdahl generalizata

q_k - procentul din program ce poate fi executat cu k procesoare
t_1 - timpul de rulare secvenţiala a programului

$$t_1 = \sum_{k=1}^{p} q_k t_1 \qquad t_p = \sum_{k=1}^{p} \frac{q_k t_1}{k}$$

$$q_k = \frac{1}{p}$$

$$S_p = \frac{t_1}{t_p} = \frac{\sum_{k=1}^{p} q_k t_1}{\sum_{k=1}^{p} \frac{q_k t_1}{k}}$$

$$S_p = \frac{1}{\frac{1}{p}\sum_{k=1}^{p} \frac{1}{k}} = \frac{p}{\sum_{k=1}^{p} \frac{1}{k}}$$

$$S_p = \frac{\sum_{k=1}^{p} q_k}{\sum_{k=1}^{p} \frac{q_k}{k}} = \frac{1}{\sum_{k=1}^{p} \frac{q_k}{k}}$$

$$S_p \le \frac{p}{\log_2(p)}$$

Teme de studiat.

https://serverguy.com/comparison/cpu-vs-gpu-vs-tpu/
https://www.linkedin.com/pulse/cpu-vs-gpu-tpu-when-use-your-machine-learning-models-bhavesh-kapil/
https://www.backblaze.com/blog/ai-101-gpu-vs-tpu-vs-npu/
How do Graphics Cards Work? Exploring GPU Architecture