Curs 8 - OC 12.04

Cap 3 Computer System Performance

3.1. Introduction

Computer Systems
1) Servers → one system, many users → Task Time, Throughput → Space → Memory, Circuit complexity
2) Desktops → Runtime → one sys, one user
3) Embedded Systems → IoT (internet of things), CPS (IoT + control)

deadline
dependability
Energy consuption

3.2 Parameters

- $Performance_x = \frac{1}{Exec.\ time_x}$

$$\frac{Performance\ A}{Performance\ B} = \frac{Execution\ time\ B}{Execution\ time\ A}$$

exemplu Computer X → 10 μs
Computer Y → 15 μs

$$\Rightarrow \frac{Performance\ X}{Perf.\ Y} = \frac{Exe.\ time\ Y}{Exe.\ time\ X} = \frac{15}{10} = 1,5$$

- CPU time

Performance A > Performance B (tranzitivitate)
Performance B > Performance C
$\Rightarrow$ Performance A > Performance C

program → Workload

Benchmark suite (cât mai variate) ← comp, simulator, ——
- kernels, toy programs, sythetic benchmark X
- real-world programs

SPEC ← www.spec.org - mașină referință

standard. performance. evaluation. corporation

1989, ---, 2017

n - number of programs in the benchmark suite

$$\text{Geometric mean} = \sqrt[n]{\prod_{i=1}^{n} \text{Sample } i}$$

Sample i: CPU time Y_i

performanța relativă față de mașină referință

$$\frac{\text{CPU time Reference } i}{\text{CPU time } x_i}$$

mașina referință

mașina referință folosită

$$\frac{\text{Performance } x}{\text{Performance } y} = \frac{\sqrt[n]{\prod_{i=1}^{n} \frac{\text{CPU time Reference } i}{\text{CPU time } x_i}}}{\sqrt[n]{\prod_{i=1}^{n} \frac{\text{CPU time } Y_i}{\text{CPU time } X_i}}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{CPU time } Y_i}{\text{CPU time } x_i}} =$$

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Performance } x_i}{\text{Performance } Y_i}}$$

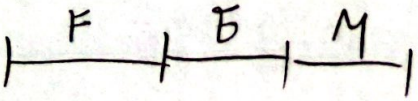
- Standard deviation $\sigma = \sqrt{\sum_{i=1}^{n} (\text{Sample } i - \text{gmean})^2}$ ← medie geometrică

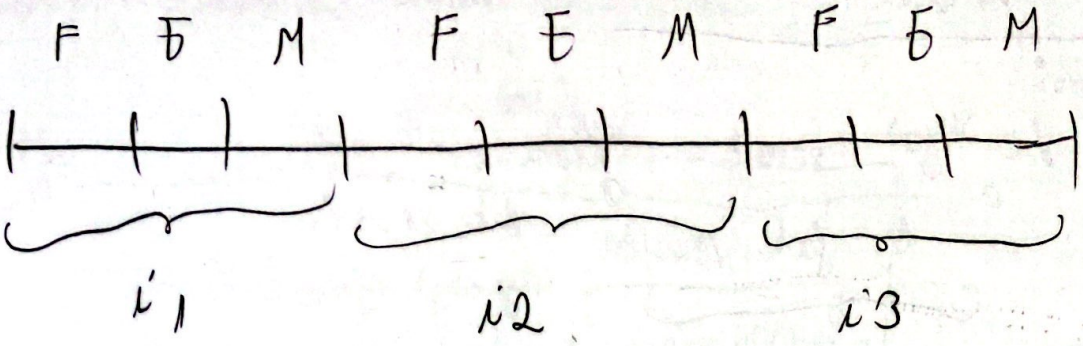
3.3 Quantitative principles in computer syst. design

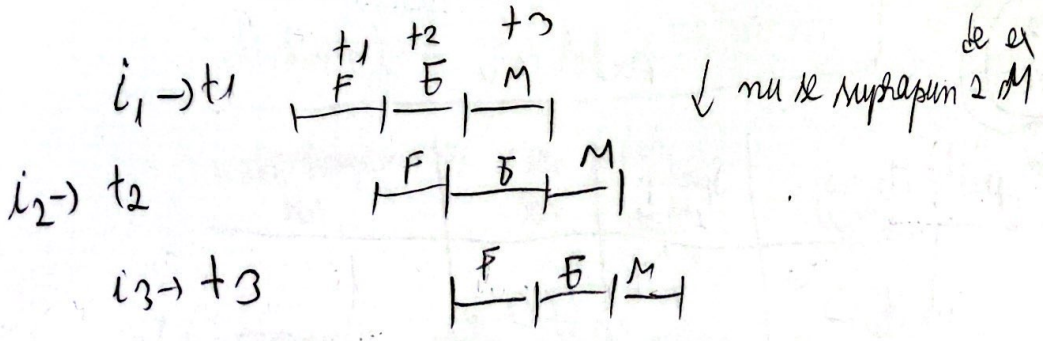
1) Use parallelism when possible

Intuition – level parallelism

Fetch, execute, memory





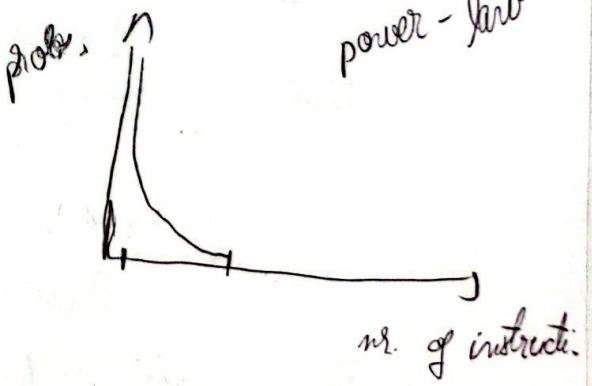


Boolean level : CLA

MPSoC
multi processor Systems - on - chips

(2) Locality principle

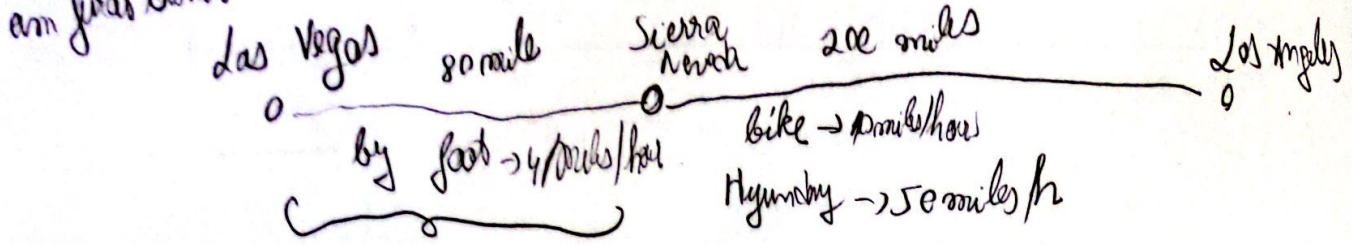
- spatial
- temporal



- The ~~proces~~ processor spends 90% of its time executing 10% of the instructions.
  - instr
  - data

3) Andahl 's law - makl the common case faster

am fost bani:



unitati masura > examen

Second part	Hours 2nd part	Speedup 2nd part	Hours tot	Speed up tot
Foot	50 hrs	1	70	1
Bike	20	2,5	40	1,75
Hyunday	4	12,5	24	2,92
Ferrari	1,6	50:1,6 = 31,25	21,6	70:21,6 = 5,24
Rocket car	1	50	21	70:21 = 3,33

$$\text{Overall speedup} = \frac{\text{Exec. time without the enhancement}}{\text{Exec. time with enhancement}}$$

$$= \frac{\text{Exec. time without enhancement}}{\text{Exe. without enh.}\left[1 - \text{Fraction enhanced}\right) + \frac{\text{Frac. enhanced}}{\text{Speedup}}\right]} = \frac{1}{\left(1 - \text{Fraction enhanced}\right) + \frac{\text{Fraction enhanced}}{\text{Speedup}}}$$

=) overall speedup =

fraction enhanced $= \frac{200}{250} = \frac{10}{14} = \frac{5}{7}$

(drum pattern optimisa)

(drum total)

$$\lim_{\text{speedup} \to \infty} (\text{Overall speedup}) = \frac{1}{1 - \text{Fraction enhanced}}$$

make the common case faster

3.4 The computer performance equation

CPU time - doesn't lie

CPU time = Clock cycles for a program X clock cycles time

$$\text{Clock frequency (rate)} = \frac{1}{\text{clock cycle time}}$$

- $10^3 \longrightarrow 2^{10}$ KiB ← binar!
- $10^6 \longrightarrow 2^{20}$ MiB
- $10^9 \longrightarrow 2^{30}$ GiB
- $10^{12} \longrightarrow 2^{40}$ TiB ← TiB
- $10^{15} \longrightarrow 2^{50}$ PiB
- $10^{18} \longrightarrow 2^{60}$ EiB
- $10^{21} \longrightarrow 2^{70}$ ZiB zetta
- $10^{24} \longrightarrow 2^{80}$ YiB yotta

$$CPU_{time} = C_{c/p} \times C_{ct}$$

(medie, separate pe tipuri)

$$= \text{Instruction count} \times \text{clock cycles per instruction} \times \text{clock cycles time}$$

(arhitectura) $= (IC \times CPI) \times \text{clock cycles time}$ (← tehnic, silicon)

$$CPU_{time} = \frac{\text{instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{clock cycles}} = \frac{\text{Time}}{\text{Program}}$$

$$CPI = \frac{\sum_{i=1}^{n} IC_i \times CPI_i}{I_C}$$

(nr instrucțiuni de tip i)

IC_i - instruction count for type i

CPI_i - clockcycles per instruction of type i

$$CPI = \sum_{i=1}^{n} \left(\frac{IC_i}{I_C}\right) \times CPI_i$$

exemplu

$FP_{instr} = 25\%$, $CPI_{FP} = 4{,}0$ c.c (clock cycles), $CPI_{others} = 1{,}33$ c.c,

$FP_{SQRT} = 2\%$, $CPI_{FPSQRT} = 20\%$ c.c.

variante: A. $CPI_{FPSQRT} = 2$ c.c. ; B. $CPI_{FP} = 2.5$ c.c.

$CPI_{original} = 0{,}25 * 4 + 0{,}75 * 1{,}33 \simeq 2$ c.c.

$CPI_A = 2$ c.c. $- (20-2) \cdot 0{,}02 = 1{,}64$ c.c.

✓ $CPI_B = 0{,}25 * 2{,}5 + 0{,}75 * 1{,}33 = 1{,}62$ c.c.

B e mai bun (make the common case faster)

Curs 9 - OC 19.04

mandi = 4 OC

open book exam (ne bătăiește vom noi)

Example 2

- Mașină A, CPU A, instrucțiuni comparație înainte salt condiționat

if a=b then
≡ ① ← limbaj de asamblare
else
≡ ②

branch not equal

```
CMP r1, r2
BNE address 1
```

≡ ①

branch necondiționat → B address 2

adress 1 ≡ ① ← linii cod
adress 2 ≡ ②

- Mașină B, are CPU_B, doar că
  în loc de
  CMP r1,r2
  BNE address1 , aici BNE r1,r2 adress 1

CPU_A = 20% cond branch instruction

CPI branch = 2 c.c.

CPI others = 1 c.c.

clock cycle $time_B$ = 1,25 × clock cycle $time_A$

Whitch CPU is better?

$CPU_{time\,A} = IC_A \times CPI_A \times$ Clock cycle time A

$CPU_{time\,B} = IC_A \times 0{,}8 \times 1{,}25 \times 1{,}25 \times$ Clock Cycle time A

$IC_B = 0{,}8 \times IC_A$

$CPI_B = 0{,}25 \times 2 + 0{,}75 \cdot 1 = 1{,}25$

20% ---- 8e).

x ---- 10%.

$x = \frac{20 \cdot 10}{80} = 25\%$

$CPU_{time\,A} = IC_A \times CPI_A \times$ Clock time A

$CPI_A = 0{,}2 \cdot 2\,cc + 0{,}8 \times 1{,}1\,cc = 1{,}2\,cc$

$CPU_{time\,A} = IC_A \times 1{,}2\,cc \times$ clock time A

$CPU_{time\,B} = IC_A \times 1{,}25 \times$ clock cycle time A

$CPU_{time\,A} < CPU_{time\,B} \Rightarrow$ Performance A > Performance B

Example 3

instruct type	Frequency	CPI
ALU	43%	1
LOAD	21%	2
STORE	12%	2
BRANCHE	24%	2

Old

, 25% ALU 1 operand from the memory

ex: c = a + b

LDR r1, var_a

LDR r2, [r1]

ADD r3, r1, r2

New

ALU register memory

ADD r3, r2, [var_a]

Nerv reg-mem ALU instr. → $CPI_{ALU} = 2.0$ c.c.

$CPI_{branches} = 3$ c.c.

Ce se schimbă? CPI_{nou}, iar clock cycle time nu e afectat

$$CPU_{time\ old} = IC_0 \times CPI_0 \times \text{Clock cycle time}_0 = IC_0 \times 1,57 \times \text{clock cycle old}$$

$CPI_0 = 0,43 \times 1 + 0,57 \times 2 = 1,57$ c.c.

$$CPU_{time\ new} = IC_n \times CPI_n \times \text{Clock cycle time}_0$$

$$IC_n = IC_0 (1 - 0,43 \times 0,25) = IC_0 \cdot 0,8925$$

$$CPI_n = \frac{(0,43 \times 0,25 \times 2) + (0,175 \times 0,43 \times 1) + (0,21 - 0,43 \cdot 0,25) \times 2 + 0,12 \times 2 + 0,24 \times 3}{1 - 0,43 \times 0,25}$$

(← e complicat)

$$CPI_n = \frac{1,6}{0,8925}$$

$$CPU_{time\ new} = IC_0 \times \frac{1,6}{1,57} \times \text{clock cycle time}_0$$

$CPU_{time\ old} < CPU_{time\ new}$ →)

Performance old > Performance new

3.5 Other metrics

MIPS
(million instruct per second)

MFLOPS
(million floating point operations per second)

$$MIPS = \frac{IC}{CPU_{time} \times 10^6} = \frac{IC}{IC \times CPI \times Clock\ cycle\ time \times 10^6} = \frac{Clock\ frequency}{CPI \times 10^6}$$

Exemplul 4 ← are ref la 3)

compiler nou - reducere cu 50% ALU

Clock cycle time = ~~20 ns~~ ns

fimp mediu instrucțiune

$$CPU_{time\ old} = IC_{old} \times 1,57 \times 20\ ns = IC_0 \times 31,4\ ns$$

$$MIPS_{old} = \frac{50 \cdot 10^6\ Hz}{1,57 \cdot 10^6} = 31,85\ MIPS$$

$$Clock\ frequency = \frac{1}{20 \cdot 10^{-9}\ s} = \frac{100}{2} \cdot 10^6 = 50\ MHz$$

$$CPU_{time\ new} = IC_n \times CPI_n \times 20\ ns \Rightarrow$$

$$IC_n = IC_0 (1 - 0,43 \times 0,5)$$

$$CPI_n = \frac{0,43 \cdot 0,5 \cdot 1 + 0,57 \cdot 2}{1 - 0,43 \cdot 0,5}$$

$$\Rightarrow CPU_{time\ new} = IC_0 (1 - 0,215) \times \frac{0,215 + 1,14}{1 - 0,215} \times 20\ ns$$

$$= 27,1\ ns \times IC_0$$

CPU time new < CPU time old

Performance new > Performance old

$$MIPS_{new} = \frac{50 * 10^6}{1,726 * 10^6} = \boxed{28,96}$$

performanta creste, MIPS scade

$$\text{Relative MIPS}_x = \frac{\text{Exec time reference}}{\text{Exec time } x} \times \text{MIPS}_{\text{reference}}$$

istoric → prima referinta

1 MIPS machine VAX 11/780

$$MFLOPS = \frac{IC\ FP}{\text{CPU time} \cdot 10^6}$$

normalised IC_{FP} — Livermore loops

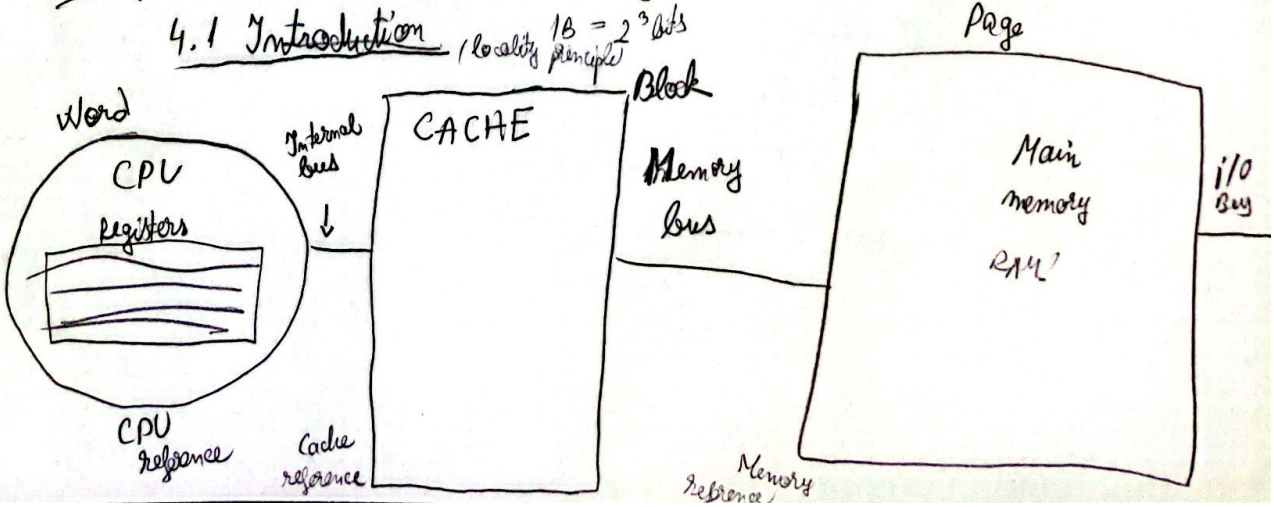
Instruct type	Unnormalised IC
+, -, x, CMP	1
div, sqrt	4
Exp, sin	8

Capitolul 4 Memory hierarchy

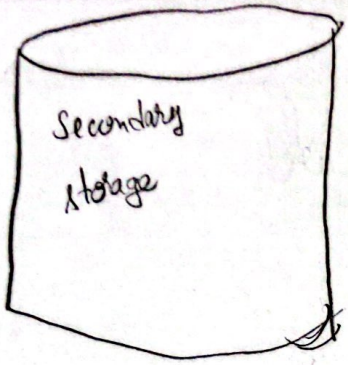
4.1 Introduction

locality principle

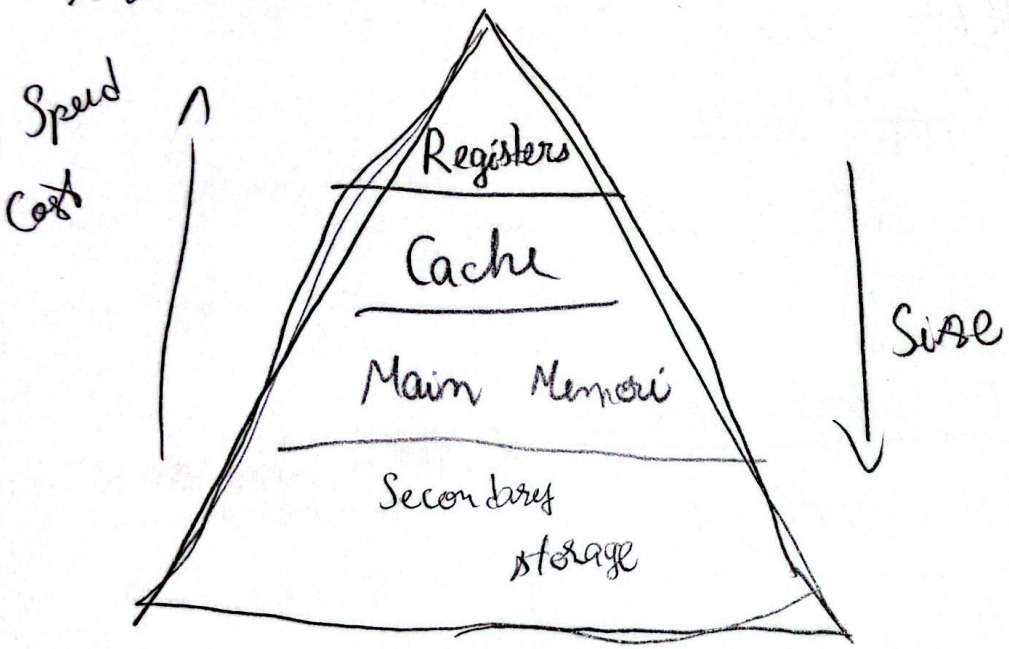
1B = 2^3 bits



I/O Bus



Rezumăm



Procesor manâncă instrucțiuni și date

Curs 10 - OC

4. Memory hierarchy

ascendere → ← evidențe unde sunt instrucțiuni

4.2 Cache mapping

4.2.1 Direct mapping

M_2 2^n blocks

M_1 2^m blocks

$m < n$

$M_2(0), M_2(1), \dots M_2(j) \dots M_2(2^n-1)$

$M_1(0), M_1(1), \dots M_1(i) \dots M_1(2^m-1)$

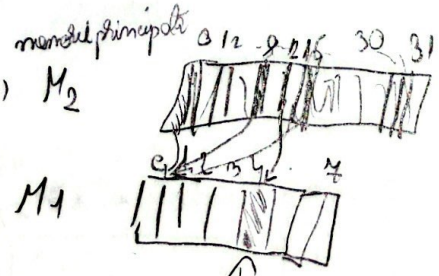
$i = j \bmod 2^m$

exemplu 1

$m = 3$

$n = 5$

=> M_2



la $M_1(0) \leftarrow 0, 8, 16, 28$

$M_1(1) \leftarrow 1, 9, 29$ ← modulo

aici $i = j \bmod 8$

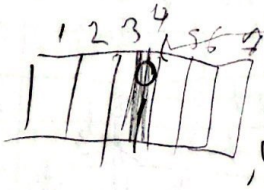
$M_1(4) = 12$

$12 : 8 = 1$ r 4

01100_2

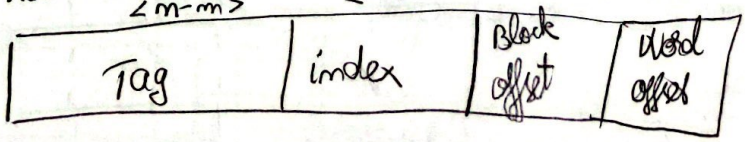
index

Tag

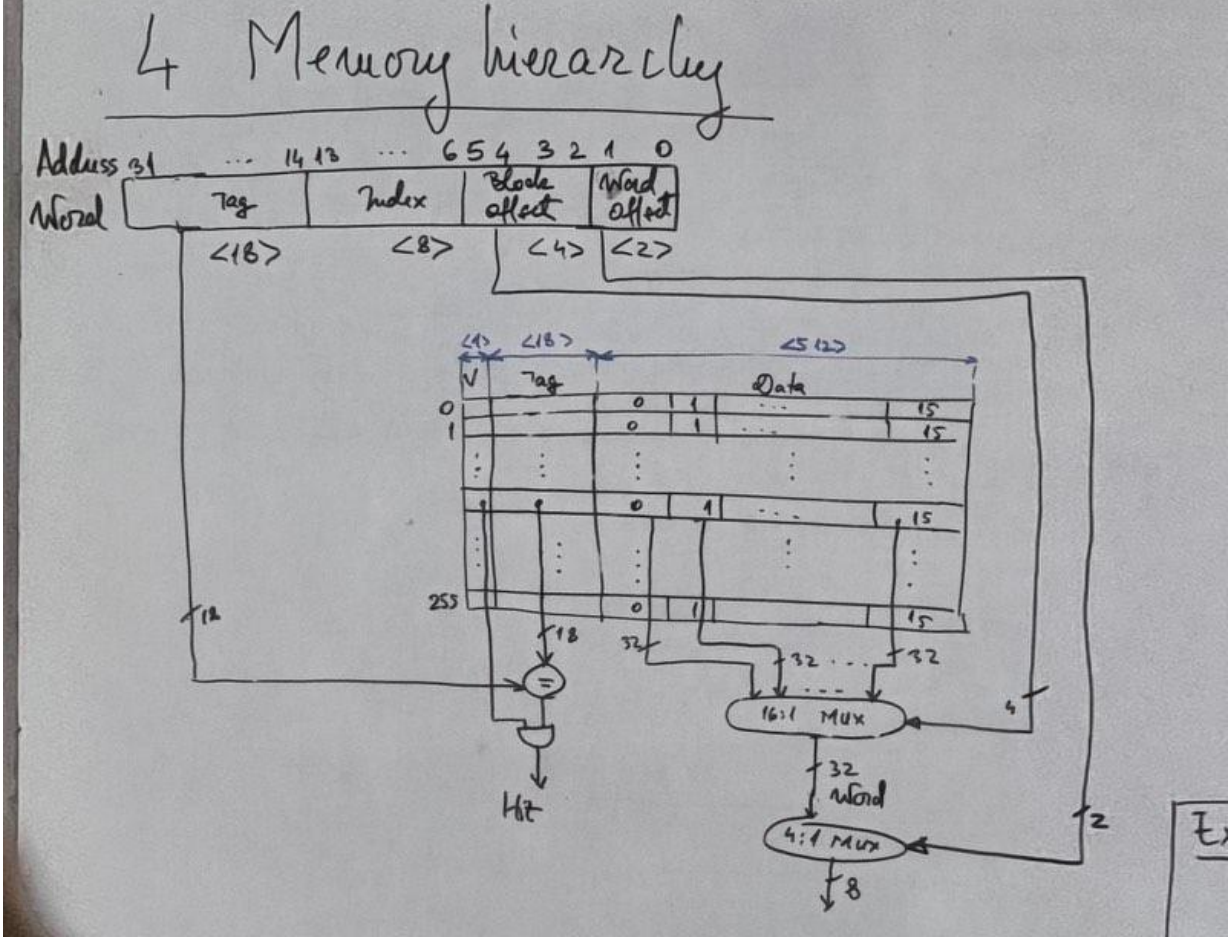


1 bloc are 2^p cuvinte

Address Word



4 Memory hierarchy



Example 2

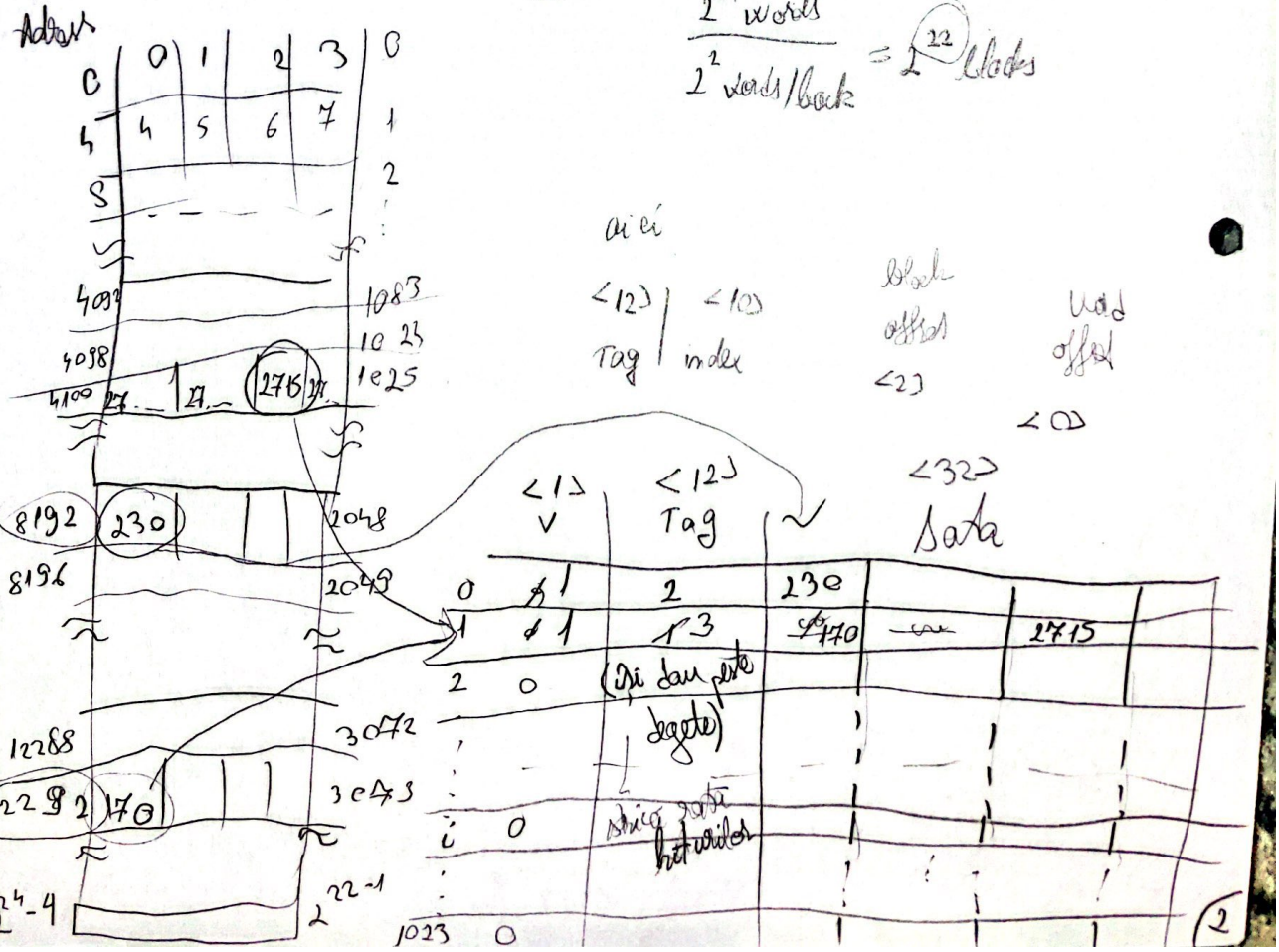
- Sist. memorie / memorie principala $MM = M_2 = 2^{24}$ cuvinte
- Un cuvant = 1 byte
  (1 word = 1 Byte)
- cache date size = 1 ki block
- mini program - 3 instructiuni, 3 instructiune

Adr	Code
8192	230
4102	2715
12 292	170

MM (poza)

Block number

$\frac{2^{24} \text{ words}}{2^2 \text{ words/block}} = 2^{22}$ blocks



2

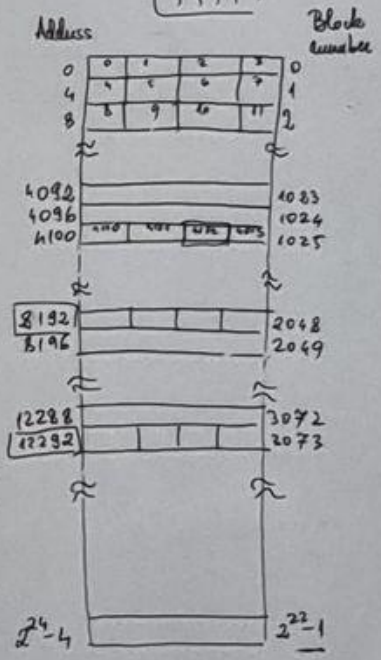
MM2 = 2^{24} words

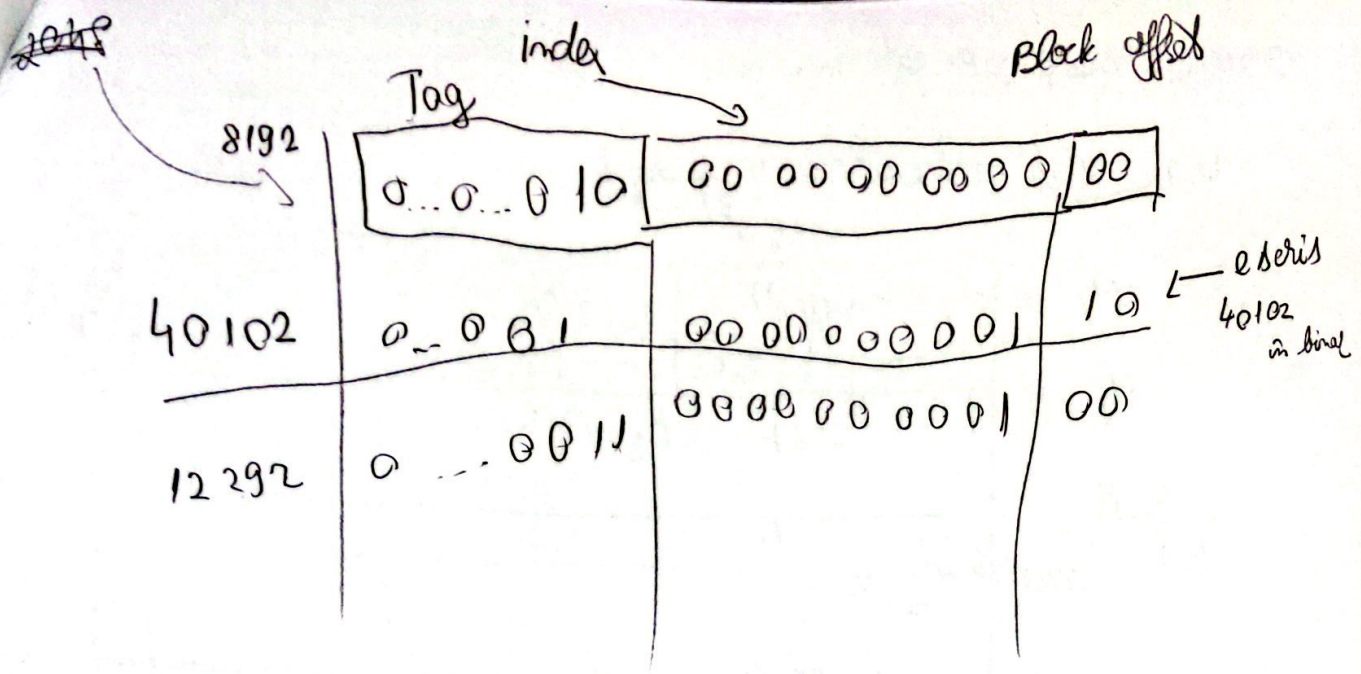
= 1 byte

ata size = 1Ki block

Code
230
2715
170

2048 +
1024
3072





- example3

Intensity's FastMath

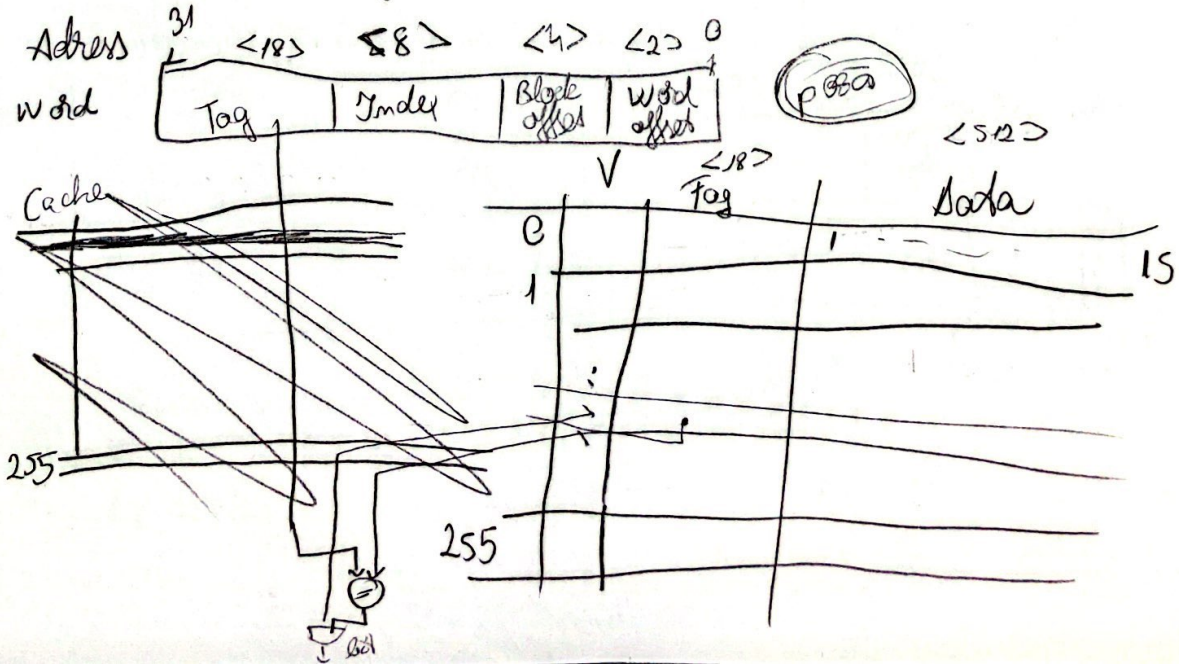
1 word = 4 bytes = 2^2

1 block = 16 words

$2^4 \cdot 2^2 - 2^3 \cdot 2^9$ bits/state

capacità memoria master: $S_{MM} = 4\ GiB = 2^2 \cdot 2^{30} B = 2^{32} B$

capacità: $S_{cache} = 2^8$ blocks



Vrem miss rate cat maimic

4.2.2 Set associative mapping

$M_1 : M_1(0) \dots M_1(i') \dots M_1(2^{m_1})$

$M_2 : M_2(0) \dots M_2(j) \dots M_2(2^{m_2})$

Set associativity is 2^s

$i' = j \bmod 2^{m-s}$

Exemplu 5

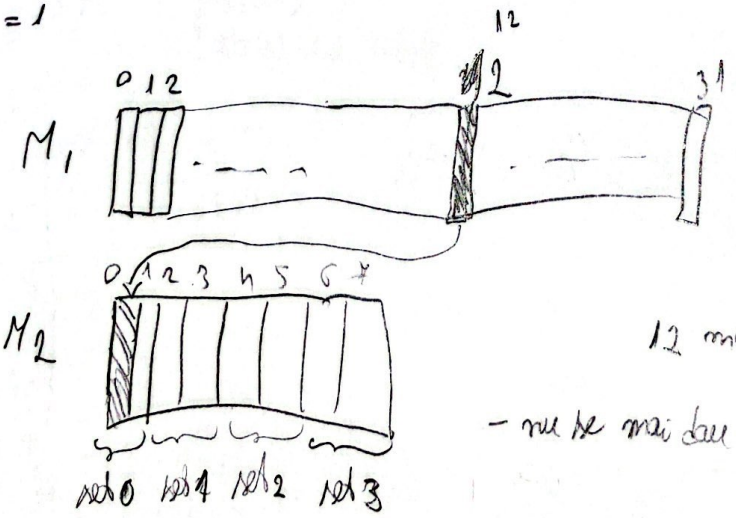
Set associativity is on $k = 2^{i=1}$ ways

$k = 2$

$m = 3$

$n = 5$

$s = 1$



$m - s = 3 - 1 = 2$

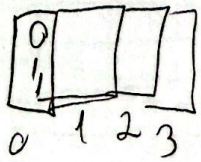
01100

Tag index

$12 \bmod 2^{3-1} = 12 \bmod 4 = 0$

- nu se mai dau afara in tag, corespunzare

Tag



Exemplul 6

	V <1>	Tag <13>	Data <32>			
0	1	4	230	—	—	whatever
	1	2	—	—	2715	—
	⋮					
	0					
	⋮					
511	0					

Bank 0

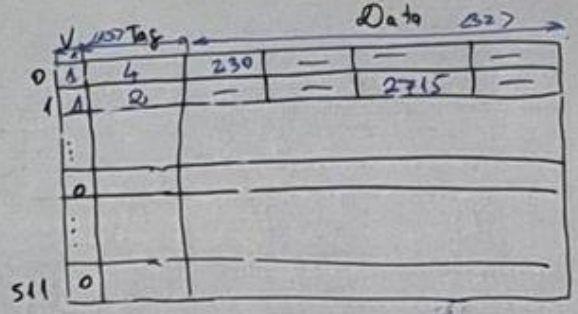
	V	Tag	Data			
0	0	0				
1	1	6	140	—	—	—
	⋮					
	0					
	⋮					
511	0					

Bank 1

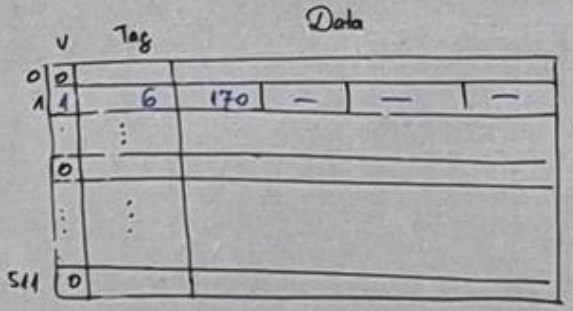
a

Adresa word	<11> Tag	<9> index	<2> word offset
8192	... 0 0 100	0000000001	00
4102	... 0 10	1	10
12292	... 0 110	1	00

Example 6



Bank 0



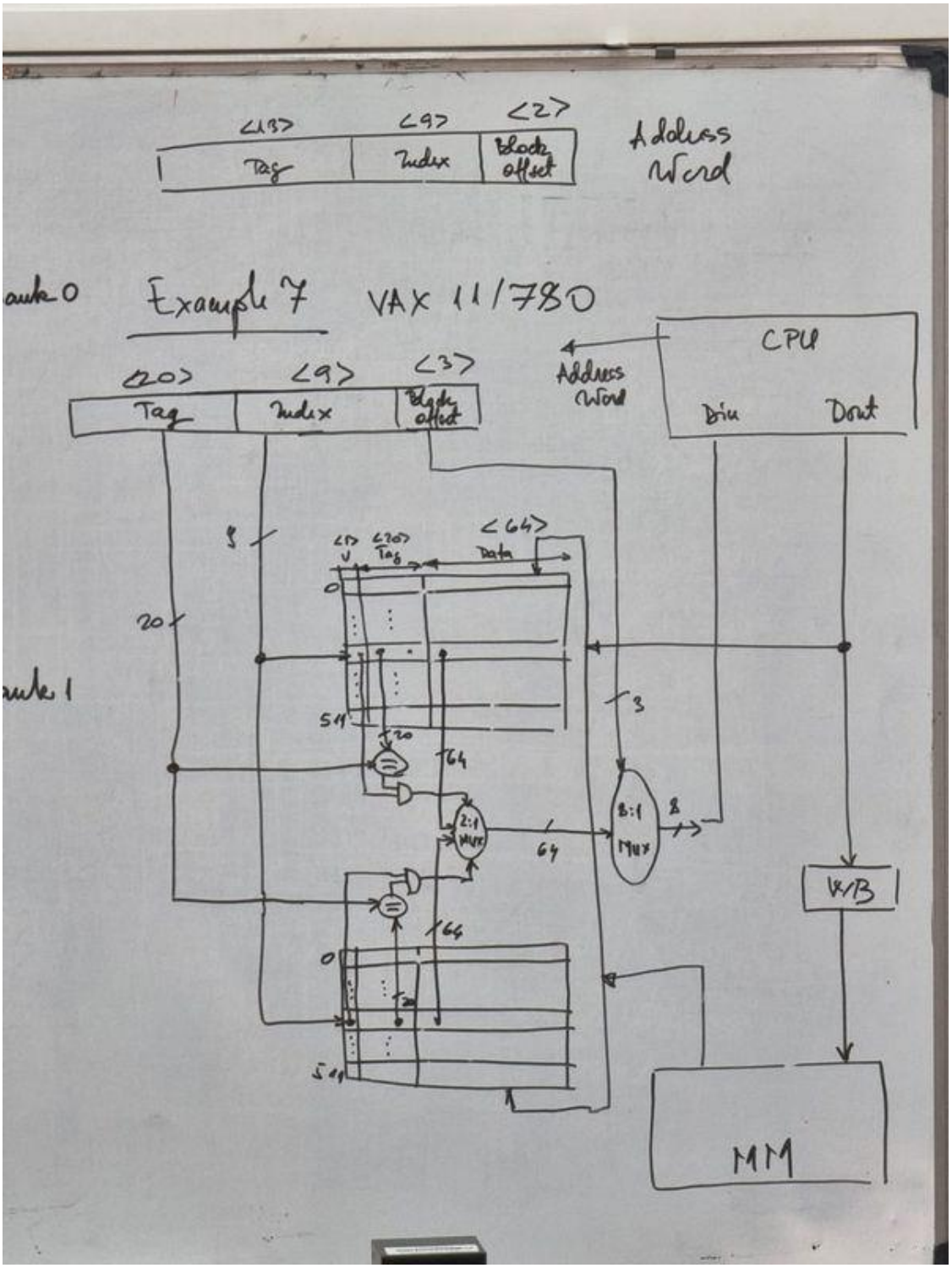
Bank 1

Exemplu

VAX 11/780

3.05, 13:45

4.2.3 Full asociative mapping


<13>
<9>
<2>
Tag
Index
Block offset
Address
Word
Example 7
VAX 11/780
<20>
<9>
<3>
Tag
Index
Block offset
Address
Word
CPU
Din
Dout
8
<1>
V
<20>
Tag
<64>
Data
0
20
511
20
64
2:1
MUX
64
3
8:1
MUX
8
W/B
64
0
511
MM

10.05

Curs 11 - OC

Fel problema ca la examen

4.2.2 Set-associative mapping

$4GiB = 2^2 \cdot 2^{30} B = 2^{32} B$

$MM_{size} = 4GiB$

byte adresabil

$1 word = 8B \quad 8 = 2^3$

4-way set associative

$1 Block = 16 words = 2^4 words$

a) formatul adresei

b) dimensiune totala cache

c) logic diagrame cache (desen din ala urat)

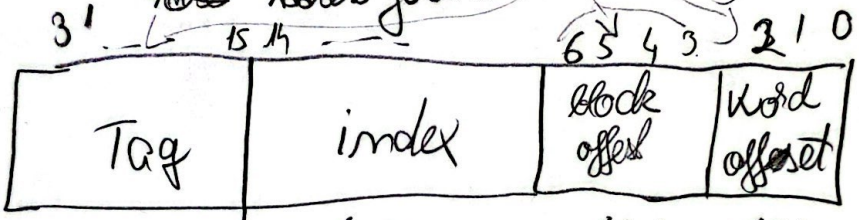
$128 KiB$ Data Cache

- Cache size $= 128 KiB = 2^7 \cdot 2^{10} B = 2^{17} B = \frac{2^{17}}{2^7} = 2^{10}$ blocks

în principiu o linie de cache

1 Block = ?

a) Address format



$\frac{2^{10} \text{ (cache blocks)}}{2^2 \leftarrow 1} = 2^8$

(4 way SA)

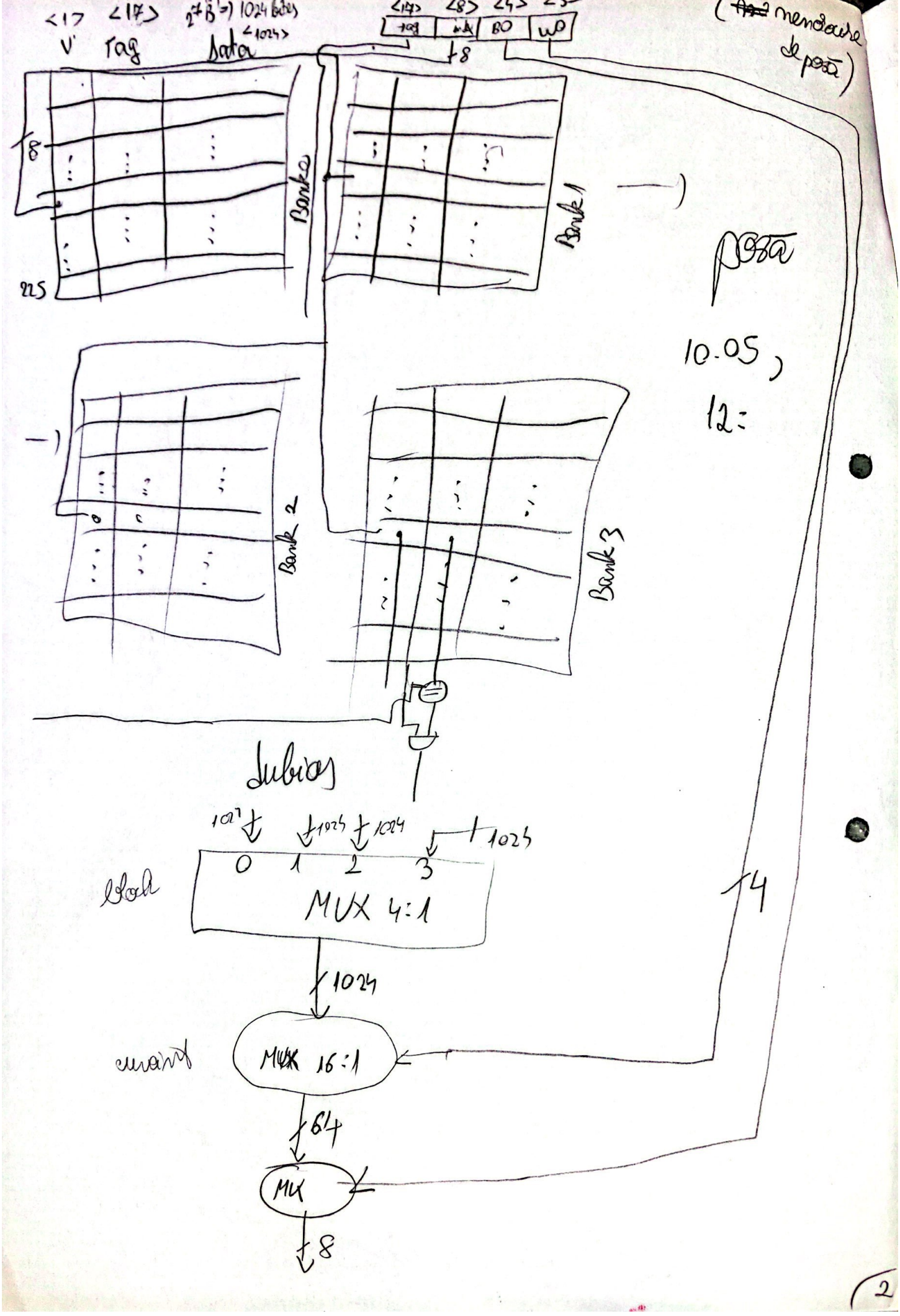
b) $\left[(1+17) \text{ biți} + 2^7 B\right] \cdot 2^{10} = \underbrace{18 \cdot 2^{10}}_{\text{biți}} + 128 KiB \approx$ aproximăm

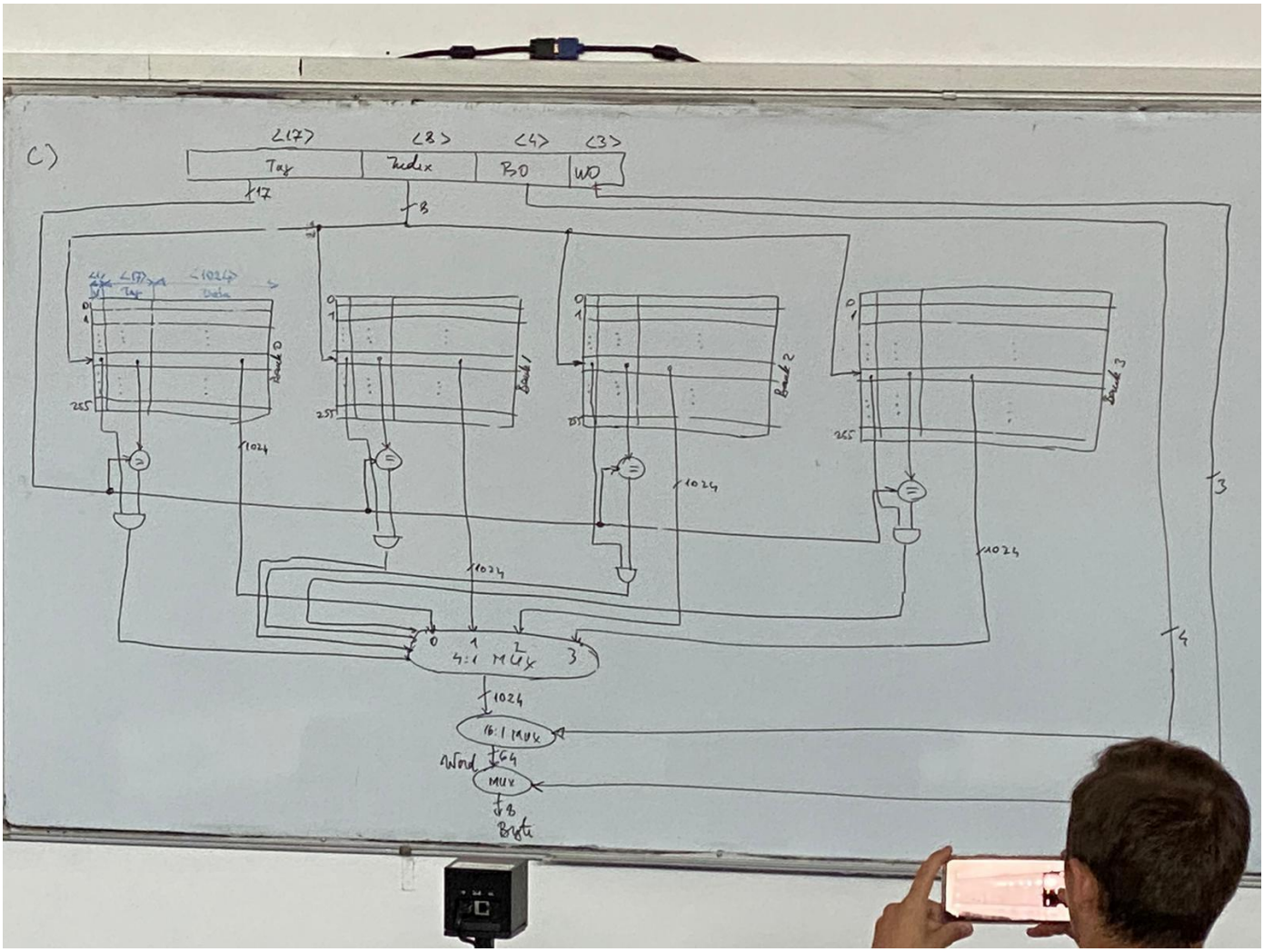
(câte are un Bloc) (câte blocuri am)

$\approx 2B \times 2^{10} + 128 KiB \approx 130 KiB$

c)

17	<8>	<4>	<3>
tag	index	block offset	000




c)
<17>
<8>
<4>
<3>
Tag
Index
BO
WO
17
8
Bank 0
Bank 1
Bank 2
Bank 3
255
1024
4:1 MUX
16:1 MUX
MUX
Byte

4.2.3 Full associative mapping

$k = 2^s$

$s = 0 \Rightarrow$ SM

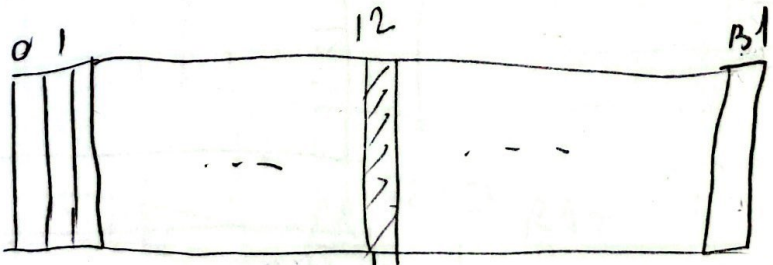
for $s = m$

2^m blocks in M1

Exemple 1

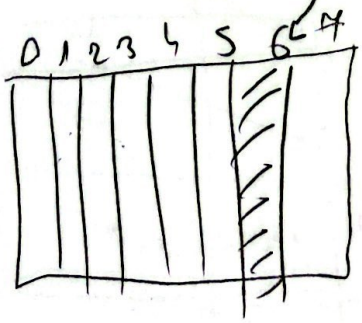
$m = 3$

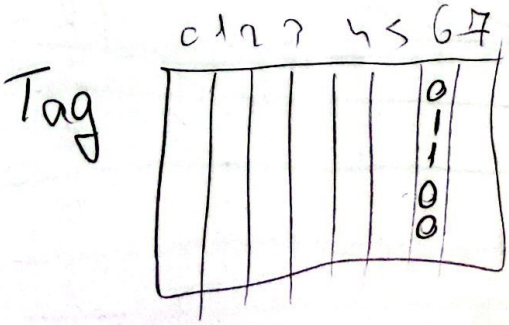
$n = 5$



01100 block ma

Tag





Example 2

$MM \rightarrow 2^{24} B = 16\ MiB$

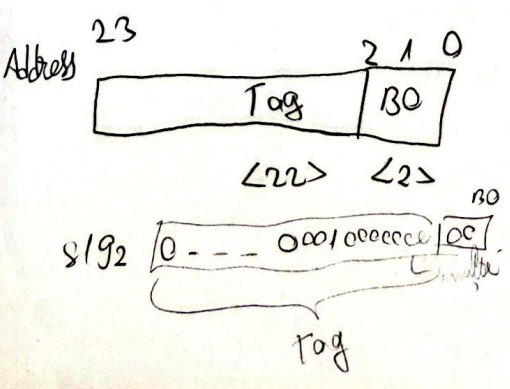
1 word = 1 B

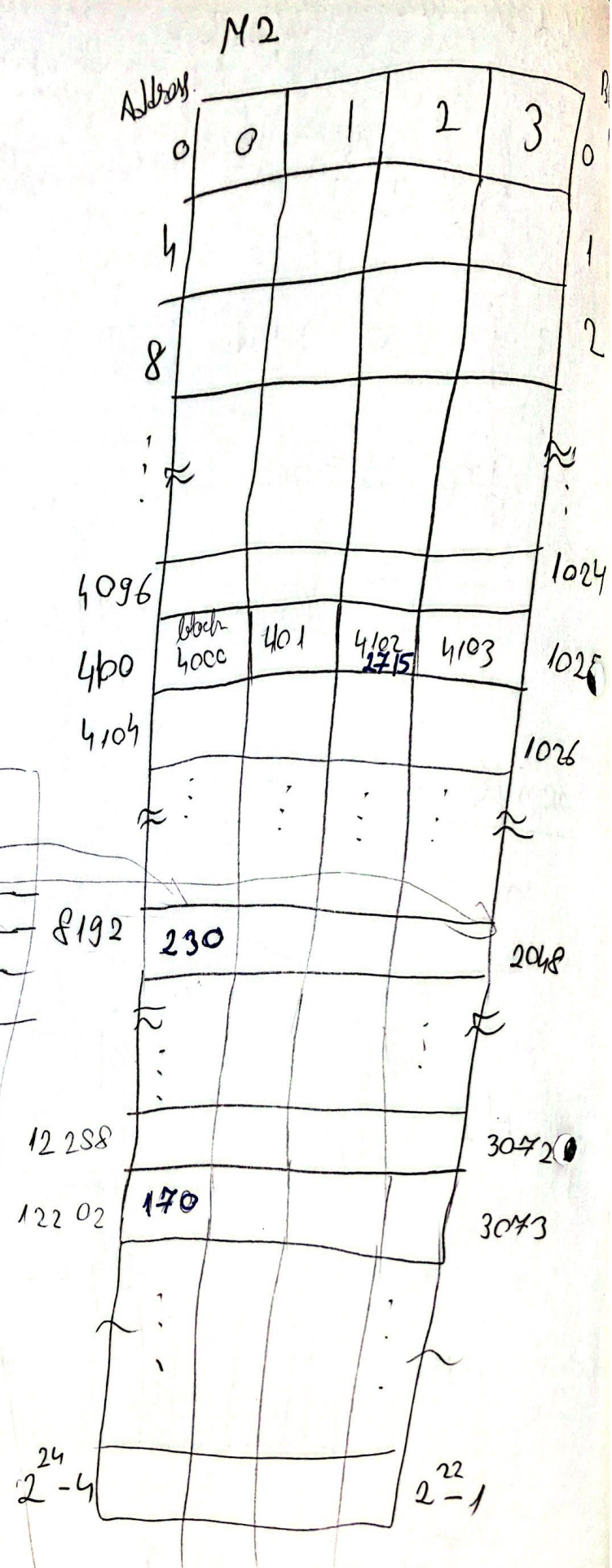
1 block = 4 words

1 Ki block cache data size

important

Address	Code
8192	230
4102	2715
12292	170





4 2

4.3 Replacement techniques

a) FIFO

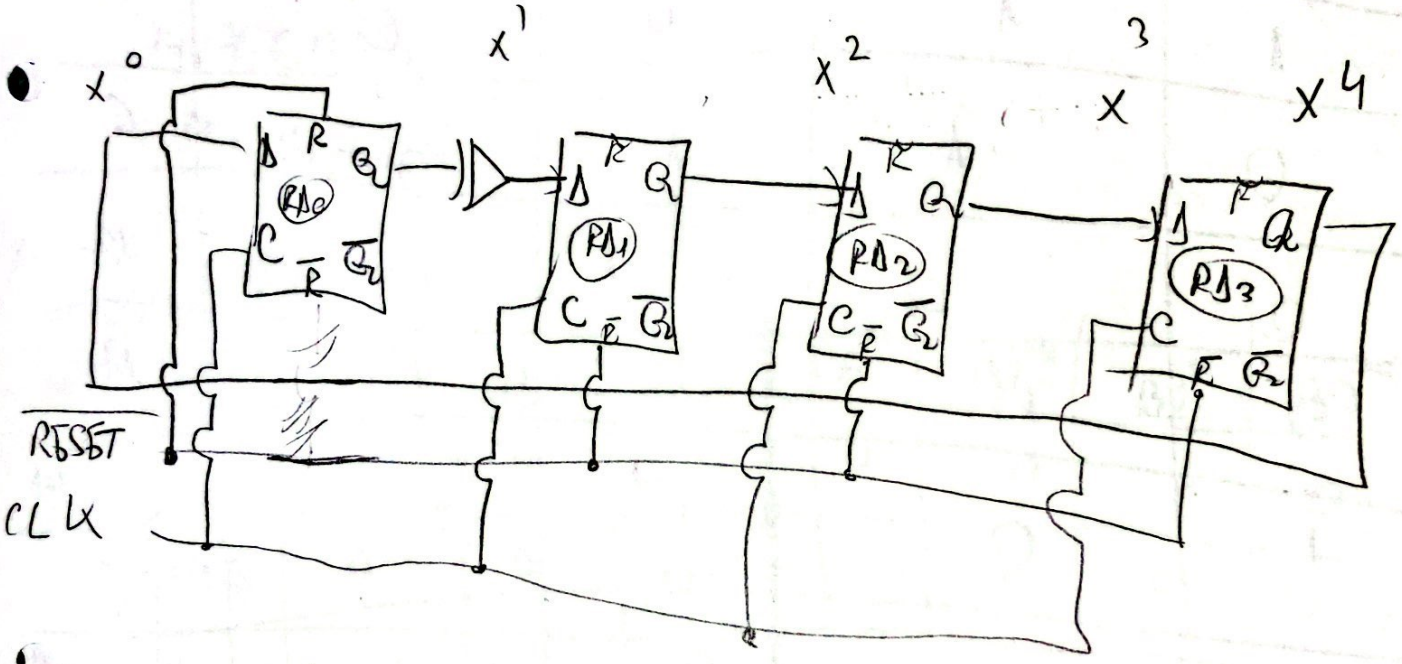
b) Random (for k small)

Galois ~~Galloa~~ ~~Godwelle~~

$$G(x) = x^4 + x + 1$$

32-
11
21

Linear Feedback Shift Register (LFSR)



RA0	RA1	RA2	RA3	
1	0	0	0	1
0	0	0	0	

R_{b0}	R_{b1}	R_{b2}	R_{b3}	k
1	0	0	0	1
0	0	1	0	2
0	0	1	0	4
0	0	0	1	8
1	1	0	0	3
0	1	1	0	6
0	0	1	1	12
1	1	0	1	11
1	0	1	0	5
0	1	0	1	10
1	1	1	0	7
0	1	1	1	14
1	1	1	1	15
1	0	1	1	13
1	0	0	1	9
1	0	0	0	1

c) Least recently used (la hit vârste mai mari rămân la fel (miss) ↳ când accesăm un block vârsta lui = const restul incrementăm) ?

Age registers ← modifici bițișori

doar cei valizi se incrementează

4 way SA

0, 1, 2, 3

14, 7, 6, 12, 10,

6, 7, 14, 7, 3, 14

	B0	B1	B2	B3	B0^α	B1^α	B2^α	B3^α
	0	0	0	0	0	0	0	0
14 M	14	0	0	0	(0)	0	0	0
7 M	14	7	0	0	1	(0) miss	0	0
6 M	14	7	6	0	2	1	(0)	0
12 M	14	7	6	12	3 (cel bătrân o dată)	2	1	(0)
10 M	10	7	6	12	0	3	2	1
6 H	10	7	(6)	12	1	3	(2)	1
7 H	10	(7)	6	12	2	0	1	3

copy	B0	B1	B2	B3	B0*	B1*	B2*	B3*
7H	10	7	6	12	2	0	1	(3)
14M	10	7	6	(14)	3	1	2	0
7H	10	(7)	6	14	3	0	2	1
3M	(3)	7	6	14	0	1	3	2
14H	3	7	6	(14)	1	2	3	0

4.4 Write policies

90% of all access are reads

pisica e și vie și moartă (nu poți avea 2 chestii în același timp)

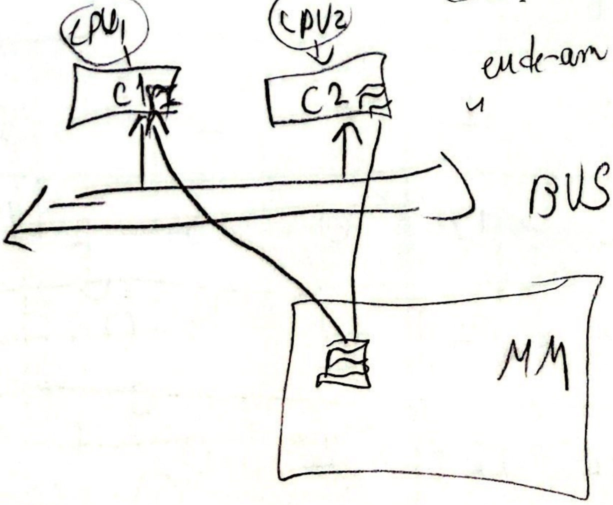
For writes techniques

a) Write Through (ciclări) — Cache coherency

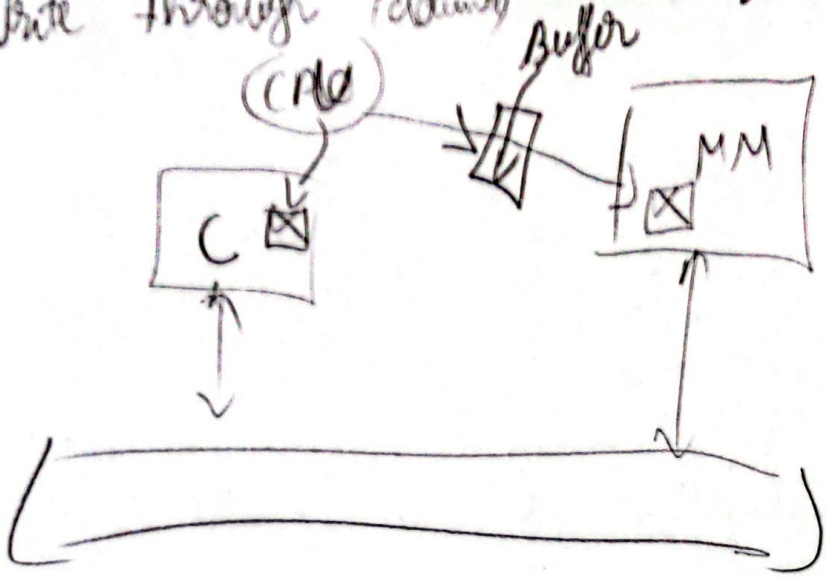
b) Write Back

„unde-am făcut-o eu te-apoi"

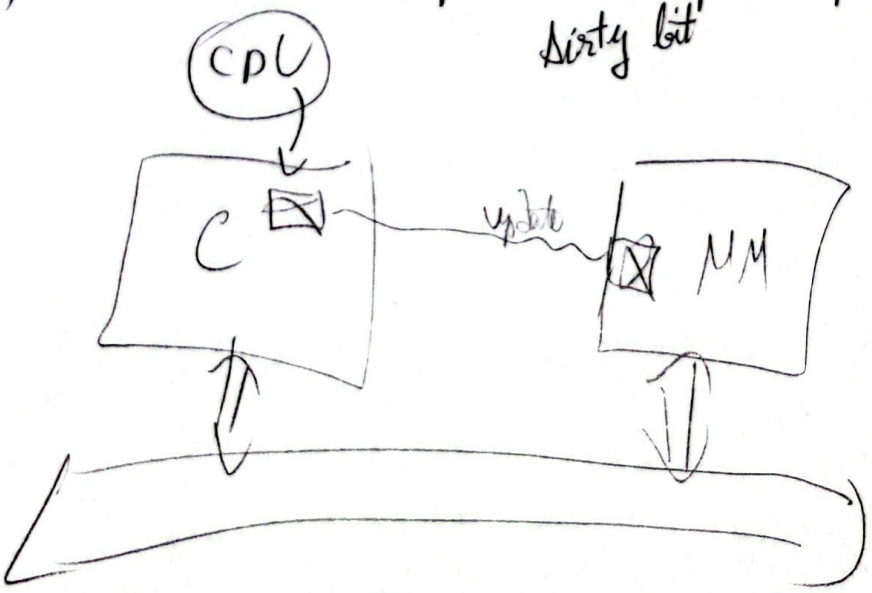
a)



a) Write through (oldies) (limitare)



- b) Write Back Update M1 upon replacement
  Dirty bit



- degeaba imi zici, daca nu faci exercitii cu mana ta ✓

In case of a Write Miss

α) Write allocate

β) Write No Allocate (nu mai aduce block in cache)

4.5 Memory hierarchy performance

AMAT = Average memory access time

$$AMAT = tac \cdot HitRate + (tac + MissPenalty)\,MissRate =$$

tac: time access (↑)

$(tac + MissPenalty)$: (timp)

$(MissPenalty + MissRate = 1)$

cât mai mic — cât mai mic

$$= tac + MissPenalty \cdot Miss\ Rate$$

Curs 12 - OC 14.05

4.5 Memory hierarchy performance

Average memory acces time = AMAT = tac + ---

1 clk cycle ↑ 1 clk cycle

AMAT = tac + Miss Rate x Miss Penalty (time)

CPU_{time} = IC x CPI x Clk cycle time

(real)

= IC x (CCPI + Memory stalls) Clk cycles time

Memory stalls = Memory accesses per instruction x Miss rate x Miss penalty (c-c)

(nu poate sa fie < 1)

Memory stalls = Memory accesses per instruct (Percentage read x Read Miss penalty + + Read miss penalty + Write miss penalty x Write miss Rate)

Exemplu 1

- cpi ideal = 8,5 cc
- memory access per instruction = 3
- Miss Rate = 11%
- Miss Penalty = 6 c.c.

tac = 1 cc. = 5 ns

AMAT?

CPU_{time}?

Degradare performanta cache

perfect = Miss rate = 0

AMAT : 5 ms !

perfect

$AMAT_{real} = 5\,ms + 0.11 \times 6\ c.c. \times 5\ ms$

$= 8,3\ ms$

$CPU_{time\ ideal} = IC \times \underbrace{8,5}_{\text{average time/per instr}} \times 5\,ms$

$CPU_{time\ real} = IC \times CPI_{real} \times 5\,ms$

$CPI_{real} = 8,5 + 3 \times 0,11 \times 6 \simeq 10,5$ cc

$\frac{CPI\ real}{CPI\ ideal} = \frac{10,5}{8,5}$ raport degradare performanta

Exemplu 2

64 KiB 2 way SA

cc time = 20 ns — clk frecuency

Mem accesses per ~~inst~~ inst = 3

CPI ideal = 1,5 cc

Miss penalty = 200 ns

degradation with 5%.

DM Cache?

Miss rate $_{DM}$ = 3,9% (direct mapping)

Miss rate $_{SA}$ = 3%

$AMAT_{1w} = 20\,ns + 0{,}039 \times 200 = 27{,}8\,ns$

↓ frecventa ⇒ ↑ timp

clk cycle time $_{1w} = 20\,ns$

clk cycle time $_{2w} = 20\,ns + 1{,}085 = 21{,}7\,ns$

$AMAT_{2w} = 21{,}7\,ns + 0{,}03 \times 200\,ns = 27{,}7\,ns$ is better

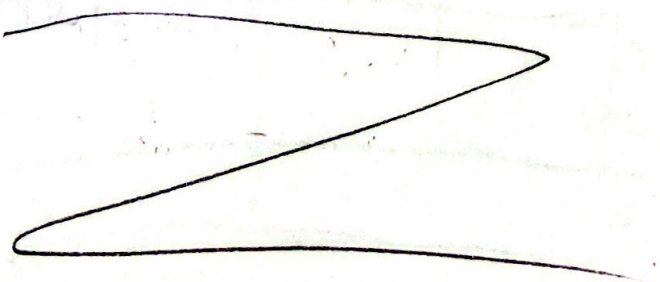
$CPU_{time\ 1w} = IC \times (CPI_{ideal} \times clk\ cycle\ time + Mem\ accesses/instruct \times Miss\ Rate \times \underbrace{Miss\ penalty \times clk\ cycle\ time}_{Miss\ penalty\ time})$

$CPU_{time\ 1w} = IC \times (1{,}5 \times 20ns + 1{,}3 \times 0{,}039 \times 200\,ns)$

$= IC \times 40{,}14\,ns$ is better

$CPU_{time\ 2w} = IC \times (1{,}5 + 21{,}7\,ns + 1{,}3 \times 0{,}03 \times 200\,ns)$

$= IC \times 40{,}35\,ns$



4.6 The 3 sources of cache misses

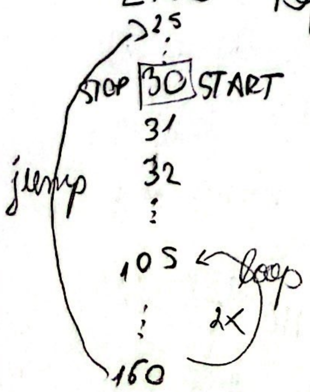
(the 3 C's)

la bucheș, cache gol, deci miss — !balanța ↙

- Cs:
  - compulsory (cold-start misses) — soluție: bigger block sizes
  - conflict – many ~~ ~~ blocks mapped at the same index — soluție: higher SA
  - capacity – cache prea mic — iei un cache mai mare

Exemplu 3

Un cache
- 4 way SA
- 2^{3} 8 sets, 4 words / block
- Addressable unit word
- MM size 2^{16} words
- cache is initially empty
- tac = 30 ns = 1 clk cycle
- Miss penalty = 480 ns
- LRU Replacement



a) Memory address format

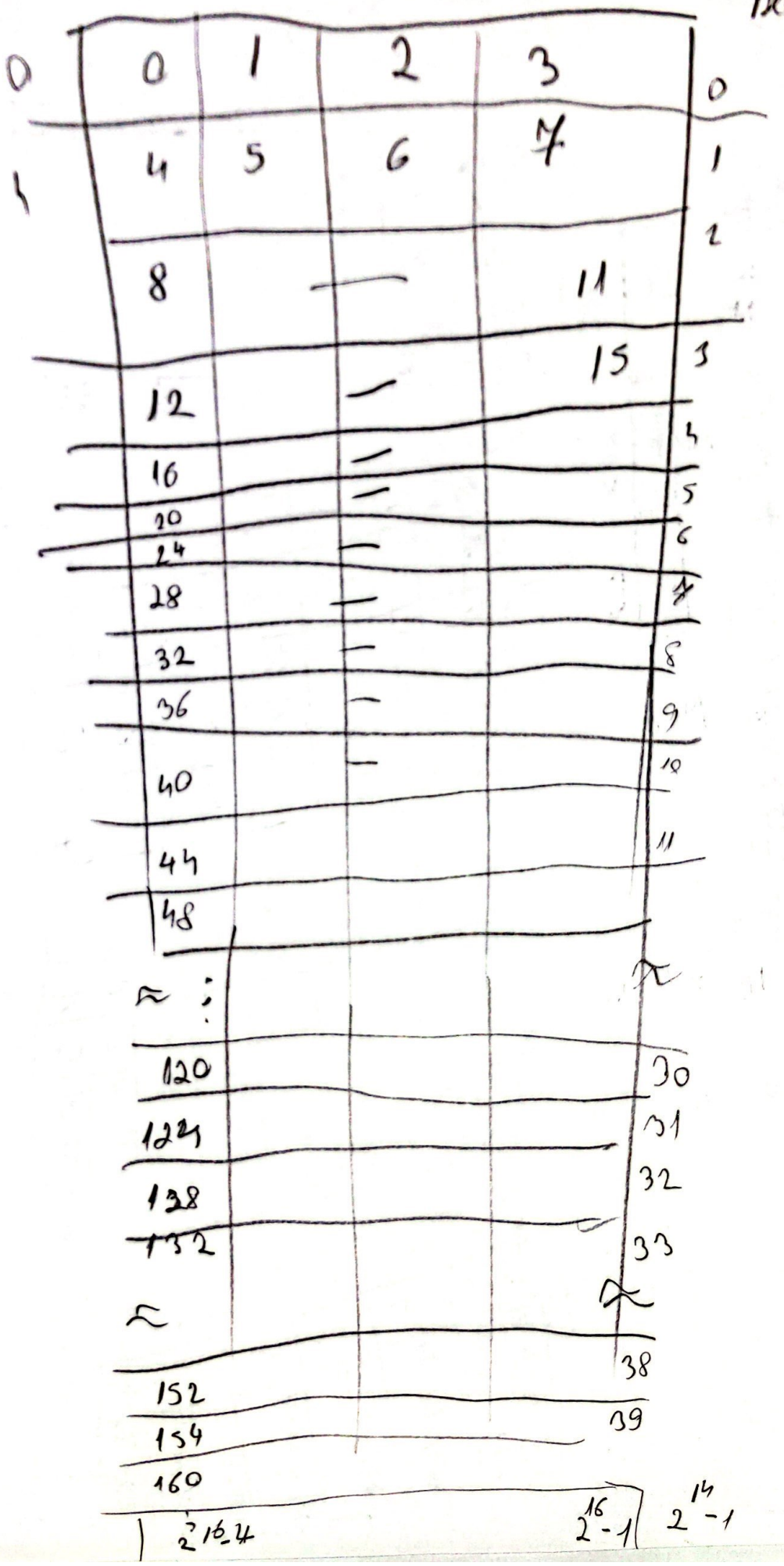
a.1) Total cache capacity if 1 word = 32 bits

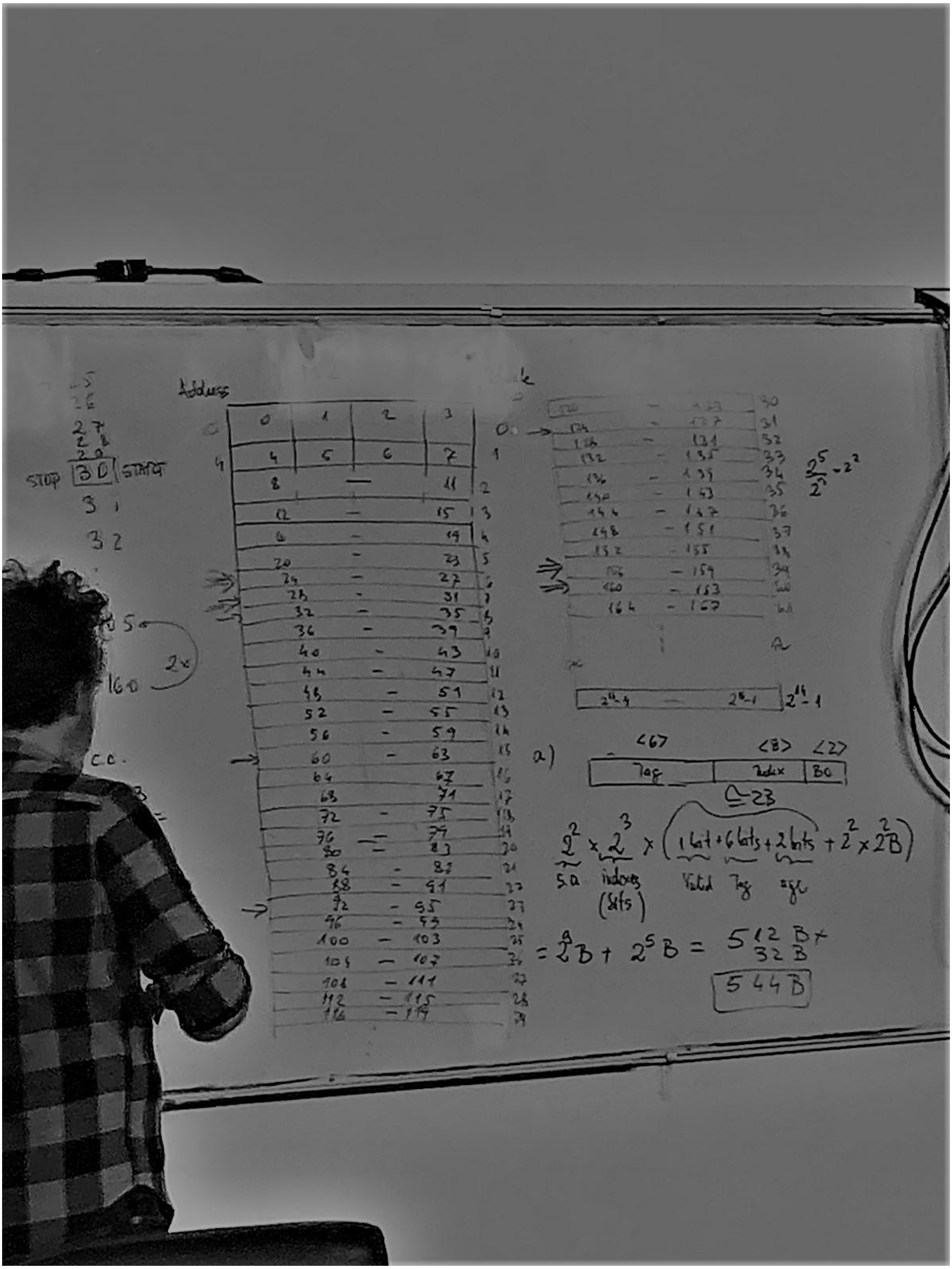
b) Hit ratio, AMAT for the workload adrese începute

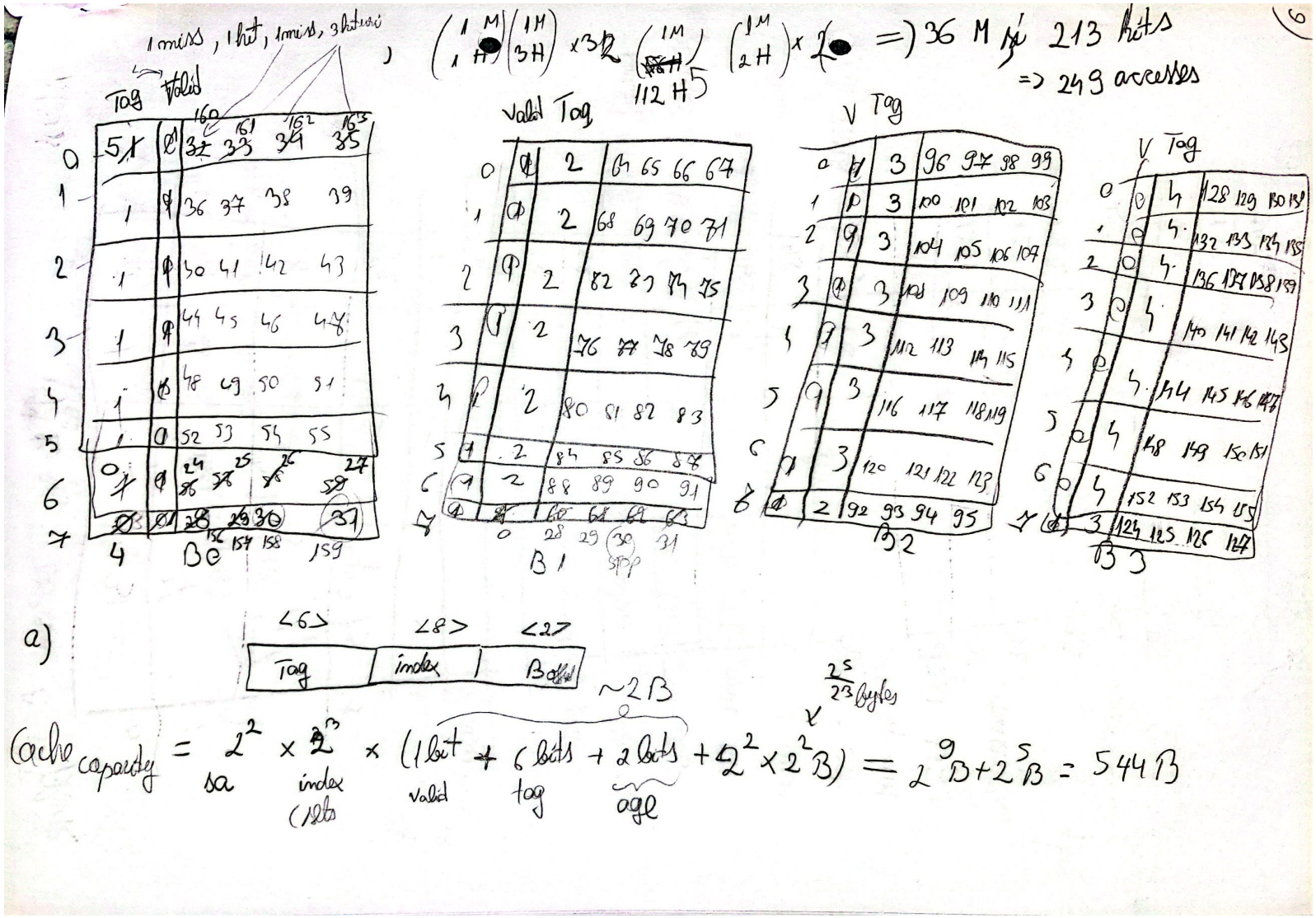
Address

M2

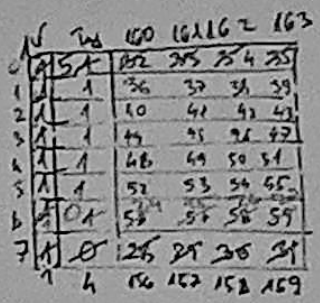
Block



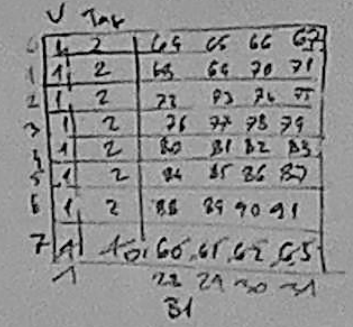

Address
STOP
30
START
Tag
Index
BO
Valid
Tag
544 B



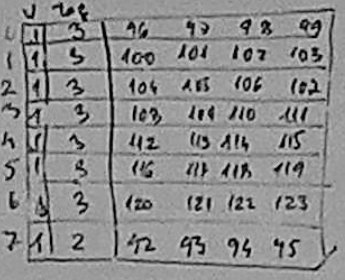
4.5 Memory hierarchy performance



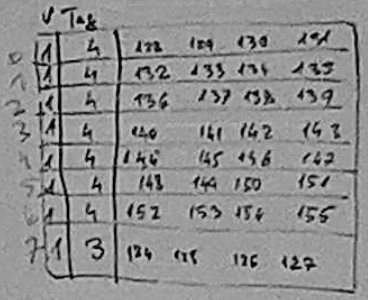
B0



B1



B2



B3

$$\binom{1M}{1H}\binom{1M}{3H}\times 32\ \binom{1M}{\ }\binom{\ }{112H}\binom{1M}{2H}\times 2$$

36 misses
213 hits
249 accesses

101 → 112 → 213

$$\text{Hit Ratio} = \frac{213}{249} = 85.5\%$$

$$\text{Miss Rate} = 14.5\%$$

$$\text{AMAT} = 30\text{ns} + 0.145 \times 30\text{ns} \times$$

99.6 ns

Hit ratio $= \frac{213}{249} = 85,5\%$

Miss Ratio $= 14,5\%$

Miss penalty (data prob) $= 480\,ns = \left[\frac{480\,ns}{30\,ns}\right] = 16\,cc$

$AMAT = 30\,ns + 0,145 * 30\,ns \times 16\,cc$

$= 99,6\,ns$

Switch 2 probe exemple

24.05

Curs 13 - oe

Examen 5 probleme

4.6

Miss Rate = 14,5%.

AMAT = 20 ns

$$CPU_{time} = IC \times (CPI_{ideal} + \underbrace{\frac{Mem\ acc}{instr} \times Miss\ Rate}_{Miss/instruction} \times Miss\ Penalty) \times \text{schimb (clk cycle time)}$$

Example 1

instruction miss rate — data miss rate

Cache A – 1M — IMR = 2% — DMR = 1,7%.

Cache B – 4 way SA — IMR = 1,6% — DMR = 1,4%.

- load store

20% of all instr sunt load store

Clock cycle time B degradat by 10%.

CPI ideal = 3 c.c

Miss Penalty = 200 ns

Clk cycle time A = 20 ns

$$Miss\ Penalty_A = \left\lceil \frac{200\ ns}{20\ ns} \right\rceil = 10$$

cel mai mic intreg ≥ ca ↙ a,a

Clk cycle time = 1,1 Clk cycle time A

$$CPU_{time\ A} = IC \times (CPI + Misses/instr_A \times Miss\ penalty_A) \times Clk\ cycle\ time_A$$

~~$= IC \times 20 ns$~~

Clk cycle time B = 22 n

$$Miss\ Penalty\ B = \left\lceil \frac{200}{22} \right\rceil = 10cc$$

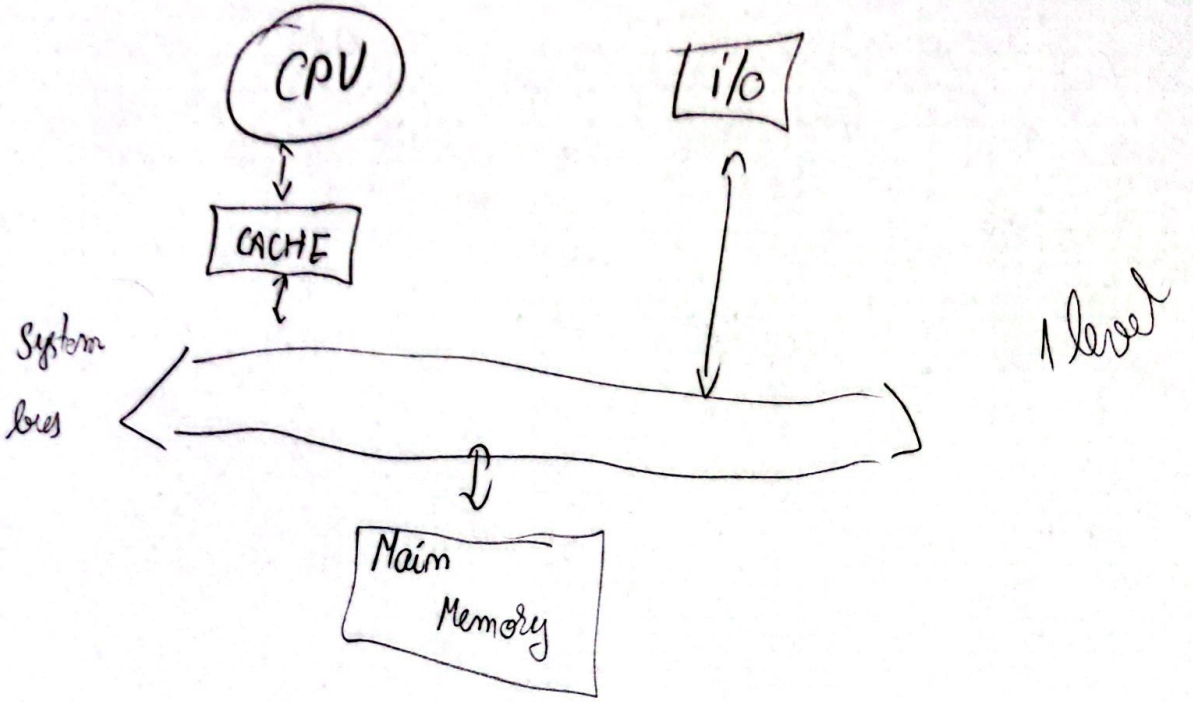
$$CPU_{time\ A} = IC \times 20\ ns\ (3 + 10 \times 0,0234) = IC \times 64,68\ ns$$

$$Miss/instr_A = IMR_A \times 1 + 0,2 \times DMR = 0,02 + 0,2 \times 0,017 = 0,0234$$

$$CPU_{time\ B} = IC \times 22\ ns \times (3 + 0,0188 \times 10) = 70,136\ ns \cdot IC$$

$$Misses/instr_B = IMR_B + 0,2 \times DMR_B = 0,016 + 0,2 \times 0,014$$

4.7 Buss usage as performance metric



Example 2

$BUS_{bandwidth} = 10^9$ words/second (viteză, presiune)

Hit rate = 90% ⇒ Miss Rate = 10%.

1 block = 4 words

BUS width = 2 words (câte cuv transferate pe clock)

Processor records references to the cache 10^8 words/sec

30% of references are writes (words/writes)

At any moment 35% of all cache blocks are dirty

Write allocate policy for miss

a) WB , b) WT

~~BUS~~ BUS is occupied in case of a

a) 10^8 words/sec × 10% × (93 × Write Miss Penalty + 7 × Read Miss Penalty)

Miss Rate

Read miss penalty = $93\% \times \frac{4 \text{ words}}{2 \text{ words}} + 1 \times \frac{4 \text{ words}}{2 \text{ words}} = 2{,}7$ BUS Reads

Dirty Taste

= Write miss penalty

$\frac{4 \text{ words}}{2 \text{ words}}$ BUS Writes — UPDATE

$\frac{4 \text{ words}}{2 \text{ words}}$ BUS Reads — ALLOCATE

WB - CPU scrie doar în cache

a)

$$BUS\ used_{WB} = 10^8\ words/sec \times 0,27 = 2,7 \cdot 10^7\ words/sec$$

$$\%\ BUSS\ Used_{WB} = \frac{2,7 \cdot 10^7\ wds}{10^9\ wds} = 2,7\%$$

WT - CPU scrie și în cache și în main memory

b) $BUS\ used_{WT} = 10^8\ words/sec \times 0,1 \times (0,3 \times Write\ Miss\ Penalty_{WT} + 0,7$ Read Miss Penalty$_{WT}$) $+ 10^8\ word/sec \times 0,9 \times 0,3 \times$ Write Hit Penalty

~~$= 10^8\ wod/s \times 0,9 \times 0,3 \times$ write Hit penalty~~

Write Miss Penalty $WT = \overbrace{\frac{4\ words}{2\ words}}^{\text{ALLOCATE}} + 1\ Bus\ write = 3\ Buss\ Accesses$

2 Bus Reads

Read Miss Penalty $WT = \frac{4\ words}{2\ words} = 2\ Bus\ Accesses$

2 Bus Reads

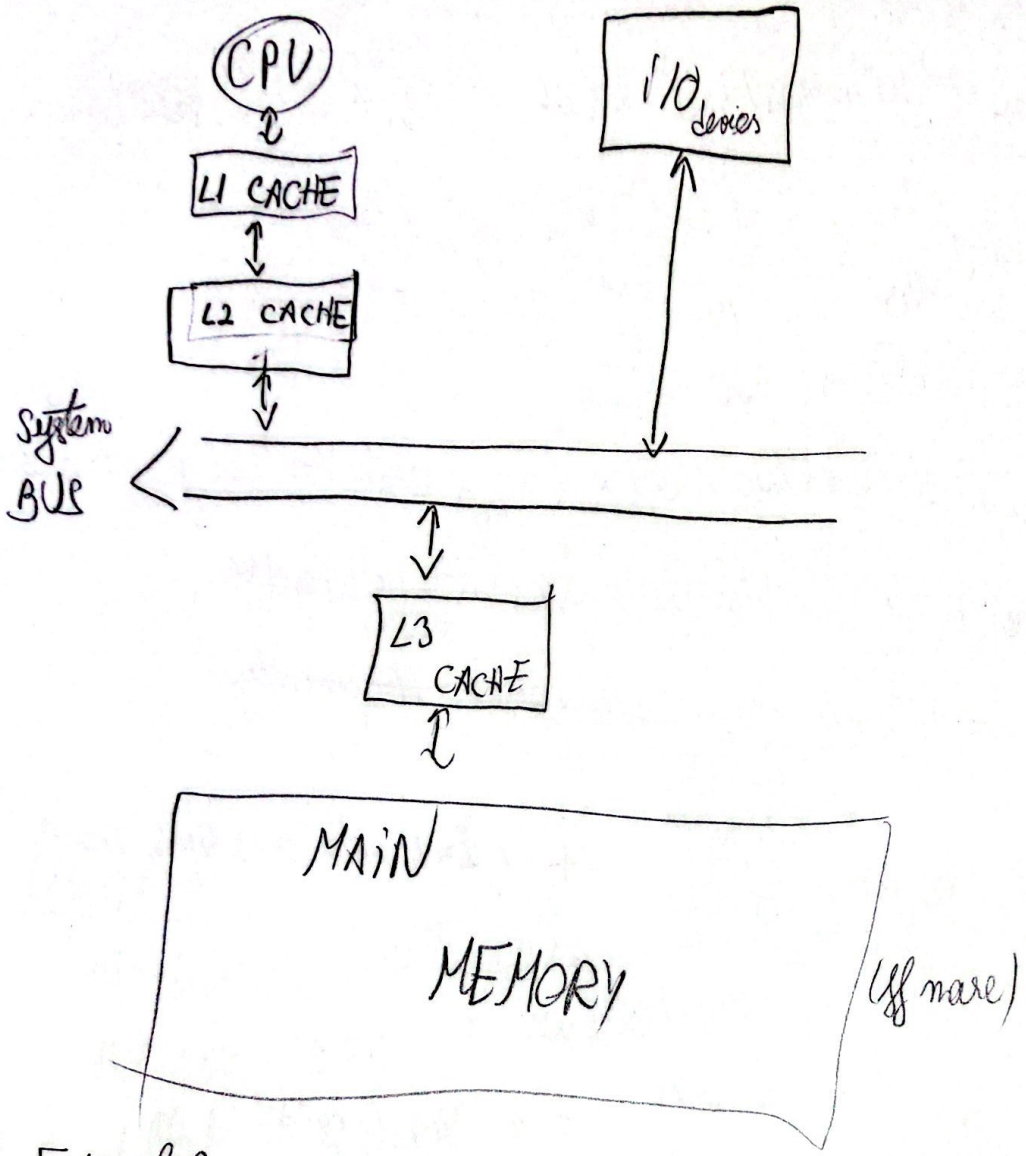
Write Hit Penalty = 1 BUS Write = 1 BUS accesses

$BUS\ Used_{WT} = 10^8\ W/s \times 0,1\ (0,3 \times 3 + 0,7 \times 2) + 10^8 \times 0,9 \times 0,3 \times 1 =$

$= 5 \cdot 10^7\ words/sec$

$\%\ BUS\ Used_{WT} = \frac{5 \cdot 10^7\ W/s}{10^9\ W/s} = 5\%$

4.8 Cache performance optimisation – multiple levels



Example 3

2 level cache system

CPI ideal = 10 cc

CLK Rate = 4 GHz CCT = $\frac{1}{4 \cdot 10^9 Hz}$ = 0,25 nS

MM acces time = 100 ns (including all miss handling)

Miss Rate per instruction = 2% for level 1 (primary cache)

Miss Rate × Mem acc pe inst

- 2nd level cache Tacc = 5 ns for each hit or miss

Miss Rate per instruction = 95%

CPU time = IC × (CPI, Miss rate instr$_{2l}$ × Miss Penalty$_{L2}$ + Miss Rate per instr$_{L2}$ × Miss Penalty MM) × Clk cycle time

$$\text{Miss Penalty}_{MM} = \left\lceil \frac{100ns}{0,25ns} \right\rceil = 400cc$$

$$\text{Miss Penalty}_{L2} = \left\lceil \frac{5ns}{0,25} \right\rceil = 20cc$$

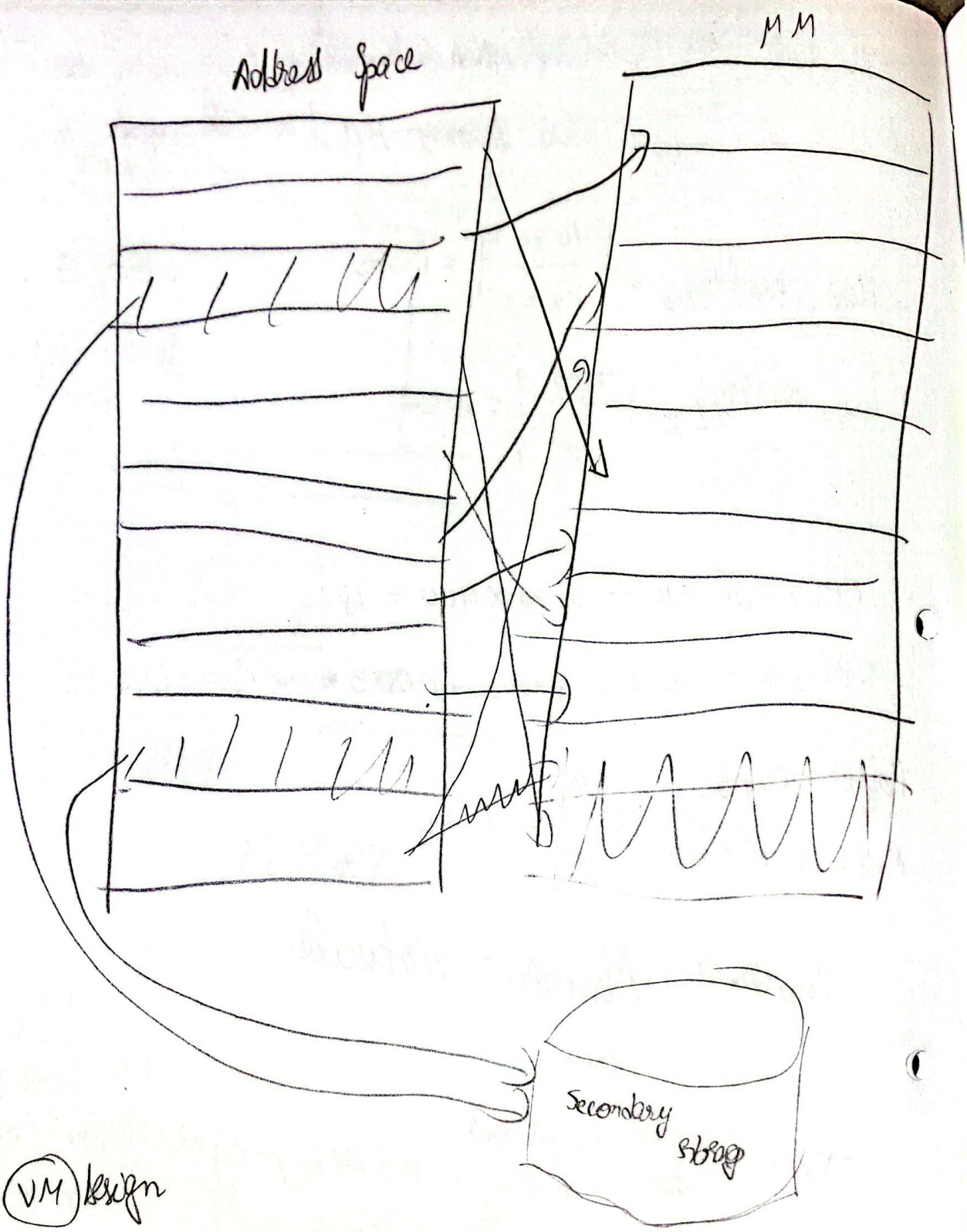
- CPI, level = 10 + 0,02 × 400 = 18 cc

CPI 2 level = 10 + 0,02 × 20cc + 0,005 * 400cc = 12,4 cc

$$\frac{\text{Performance 2 level}}{\text{Performance 1 level}} = \frac{18}{12,4} = 1,45$$

Capitolul 5. Memorii virtuale

- secondary storage
- Virtual machines, processes
  - active → according to the locality principle
  - each with its own memory space
- Blocuri — Paged (egale)
  - Segmented (inegale (segmente din pagini))



VM design

dictated by the huge miss penalties

Milions of cc

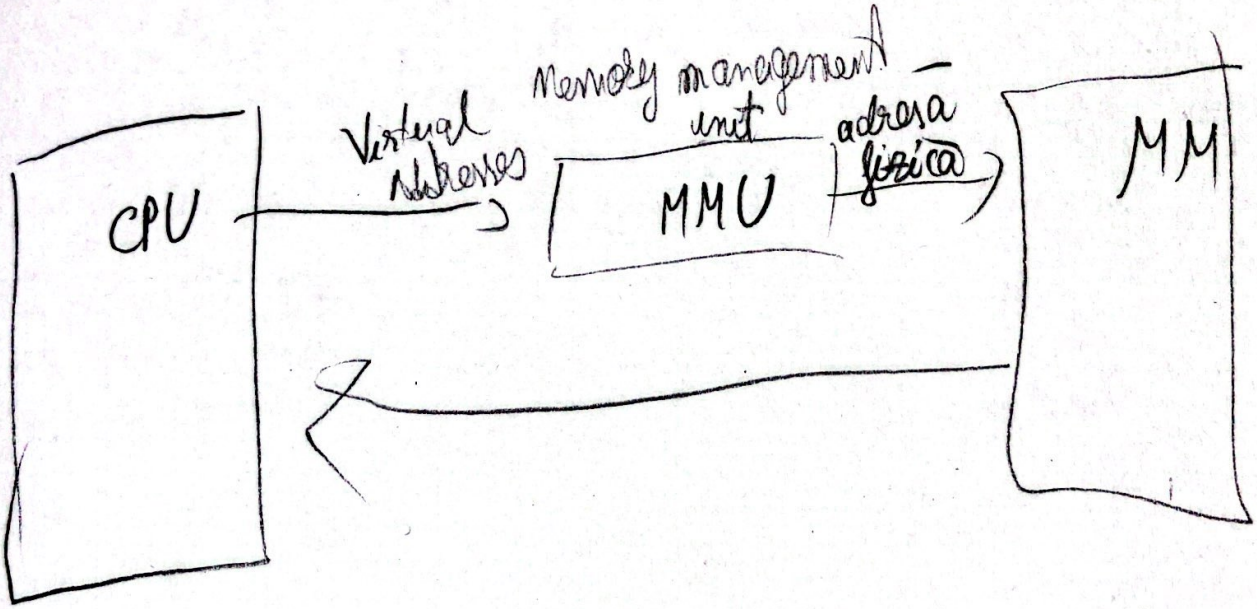
MM is 100 000 time faster than the secondary storage

- large pages ARM VE pagina 4 KiB → 64 KiB

Servers 32 KiB - 64 KiB

Embedded → 1 KiB

- full associative mapping
- write back



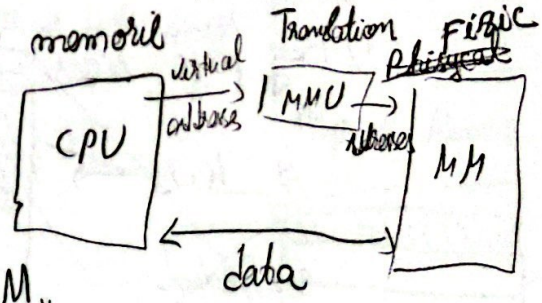
Curs 14 - OC 31.05

5. Virtual machines

5.1 introduction

procese active - dispun de CPU

programe - își protejează spațiul de memorie

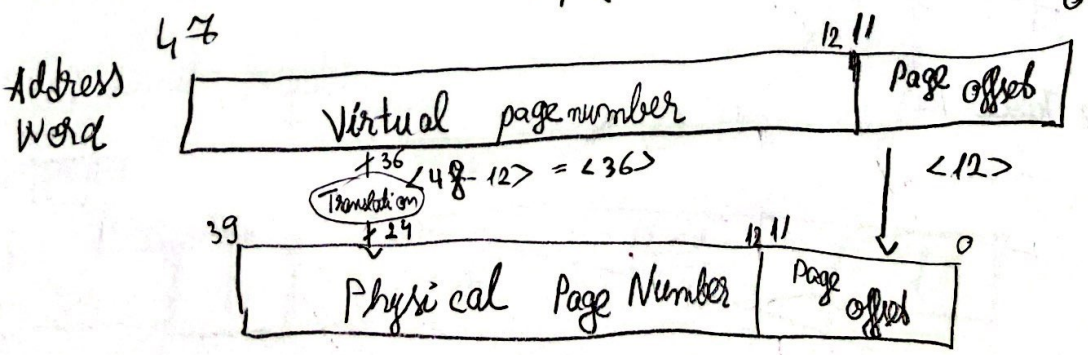


- locality principle

Cache	VM (virtual memory)	ARM
block	pages / (segments)	granule
miss	page fault	MMU exception

exemple ARM V8

- address Word = 64 biți (16 biți unused)
- byte addresses → 4 KiB pages = 2^{12} B
- ~~4 KiB pages = 2^{12} B~~ → page table entries → 64 biți



Virtual memory size = 2^{48} = 256 TiB

Physical memory size = 2^{40} = 1 TiB

Design choices

(page fault rate)

PROBLEMA Huge miss penalty → millions of clock cycles

Soluzioni

1) Big page sizes → a amortiza

4 KiB → 1 KiB — 32 KiB, 64 KiB

embeded — desktop, Servers

2) Replacement policy in software

3) Full associativity → implica implementare bazata pe page tables

4) Write Back with Write allocate

~~5.2) Virtual~~

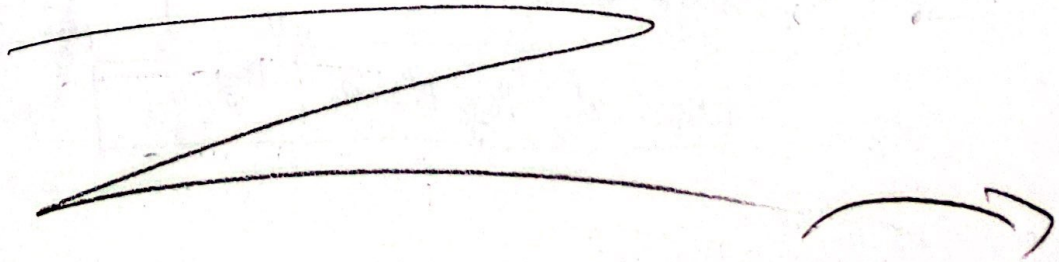
Cap 5.2) Virtual memory mapping

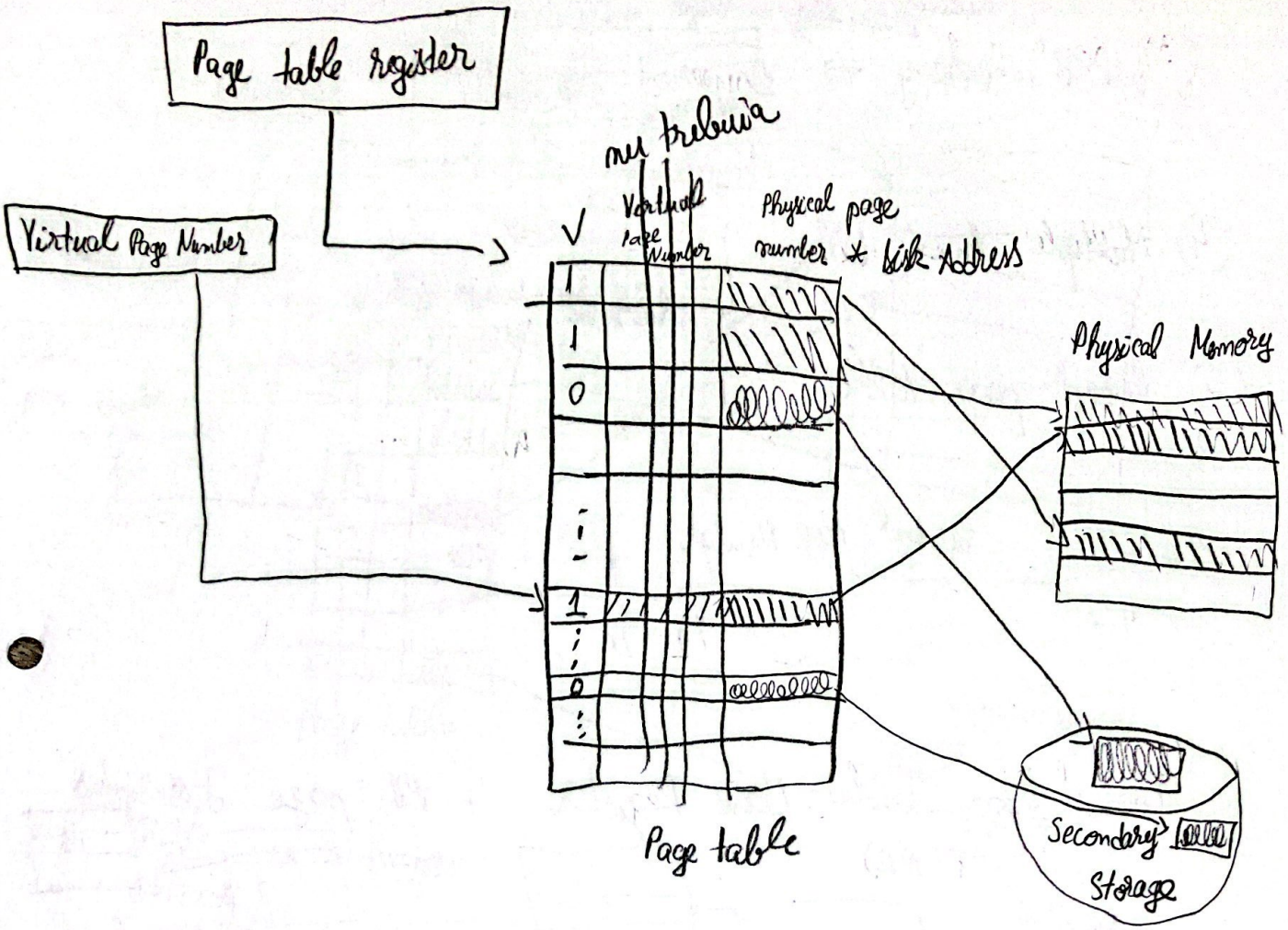
valid — 1 memorie ~~and~~ fizica

0 secondary storage

Register for Page Tables

Page Table Re



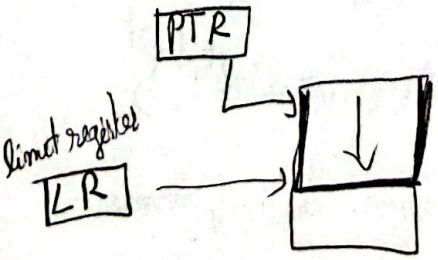


Page Table size

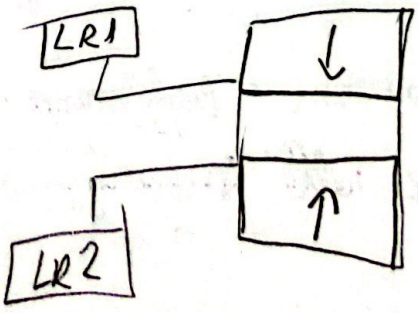
- 2^{36} page table entries $\times 2^3 B = 2^{39} B = 95\ TiB$ >> trebuie să facem ceva

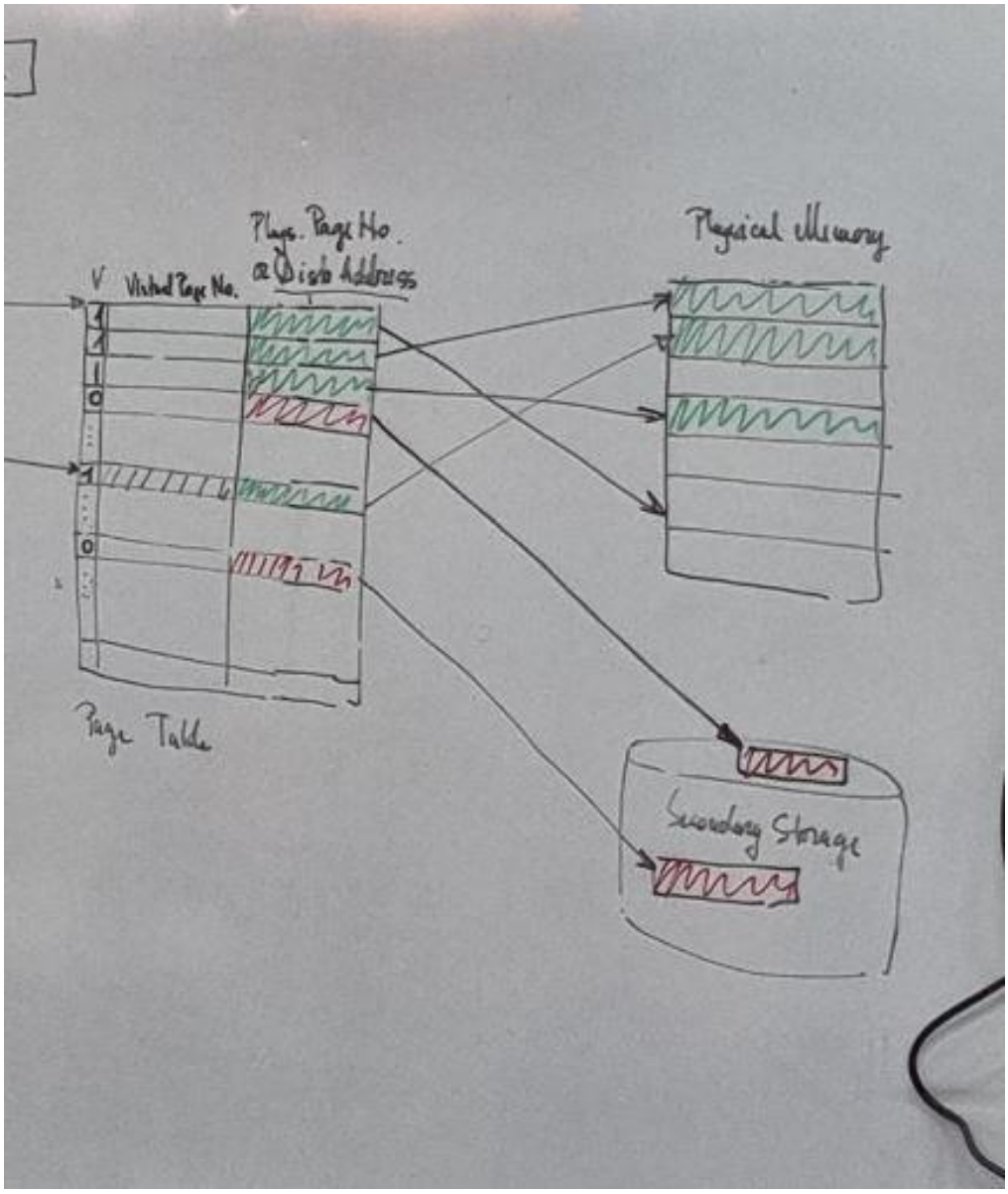
5-2-1) Page table size reduction

① Limit register



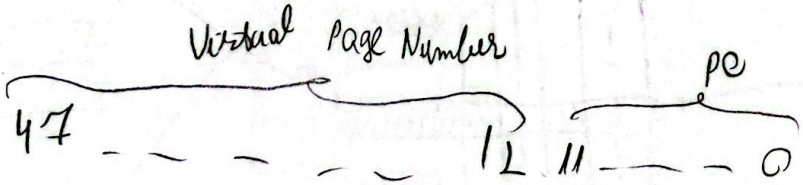
② Heap stack



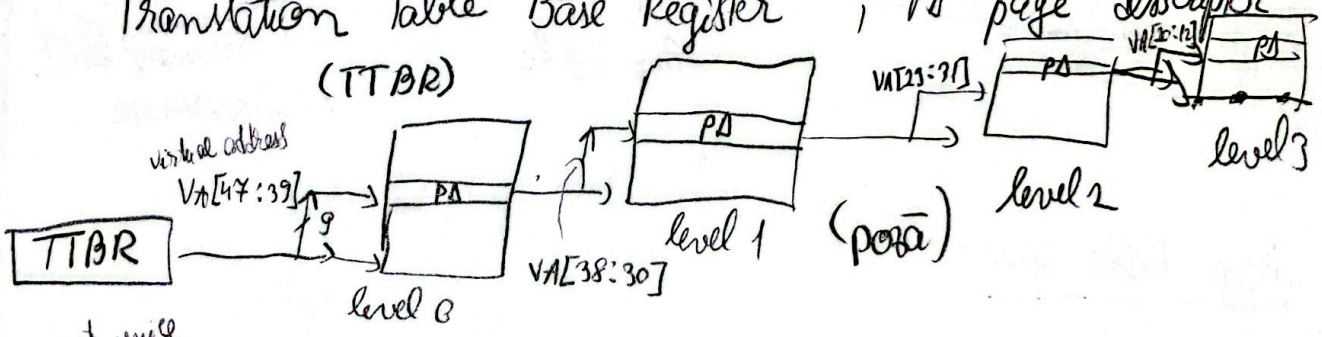

Phys. Page No.
or Disk Address
Physical Memory
V
Virtual Page No.
1
1
1
0
1
0
Page Table
Secondary Storage

③ Page Hashing → inverted page table

④ Multiple-level tables
⑤ Paged page table
} ARM V8



Translation Table Base Register (TTBR), PA = page descriptor



memorie virtuala

level 0 VA[47:39], level 1 VA[38:30], level 2 VA[29:21], level 3 VA[20:12]

o tabela de pagini, intra pe pagina

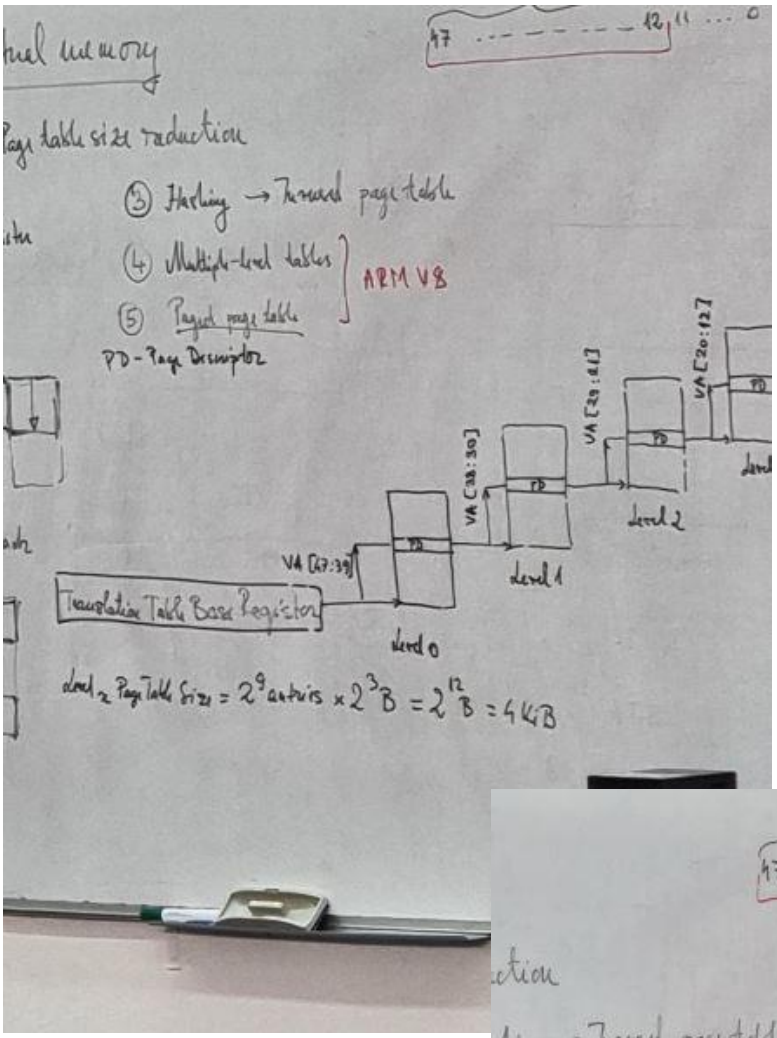
level 2 Page Table Size = 2^9 entries $\times 2^3 B = 2^{12} B = 4KiB$

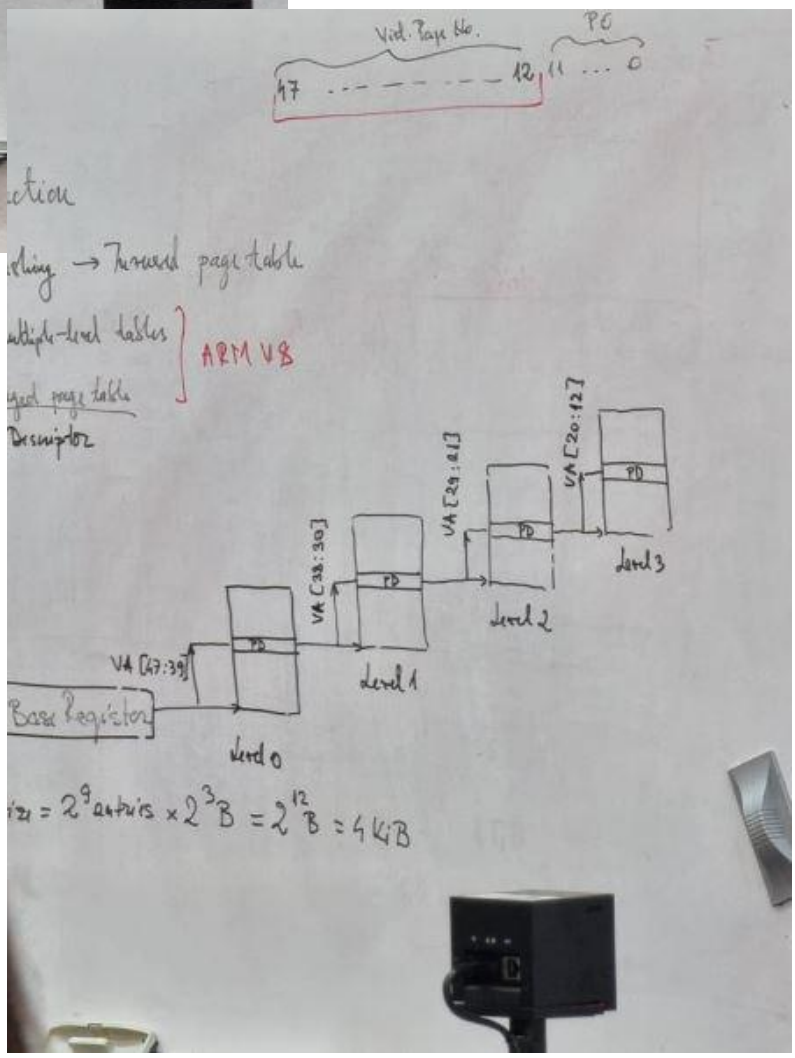
5.2.2 Translation Lookaside Buffer (TLB)

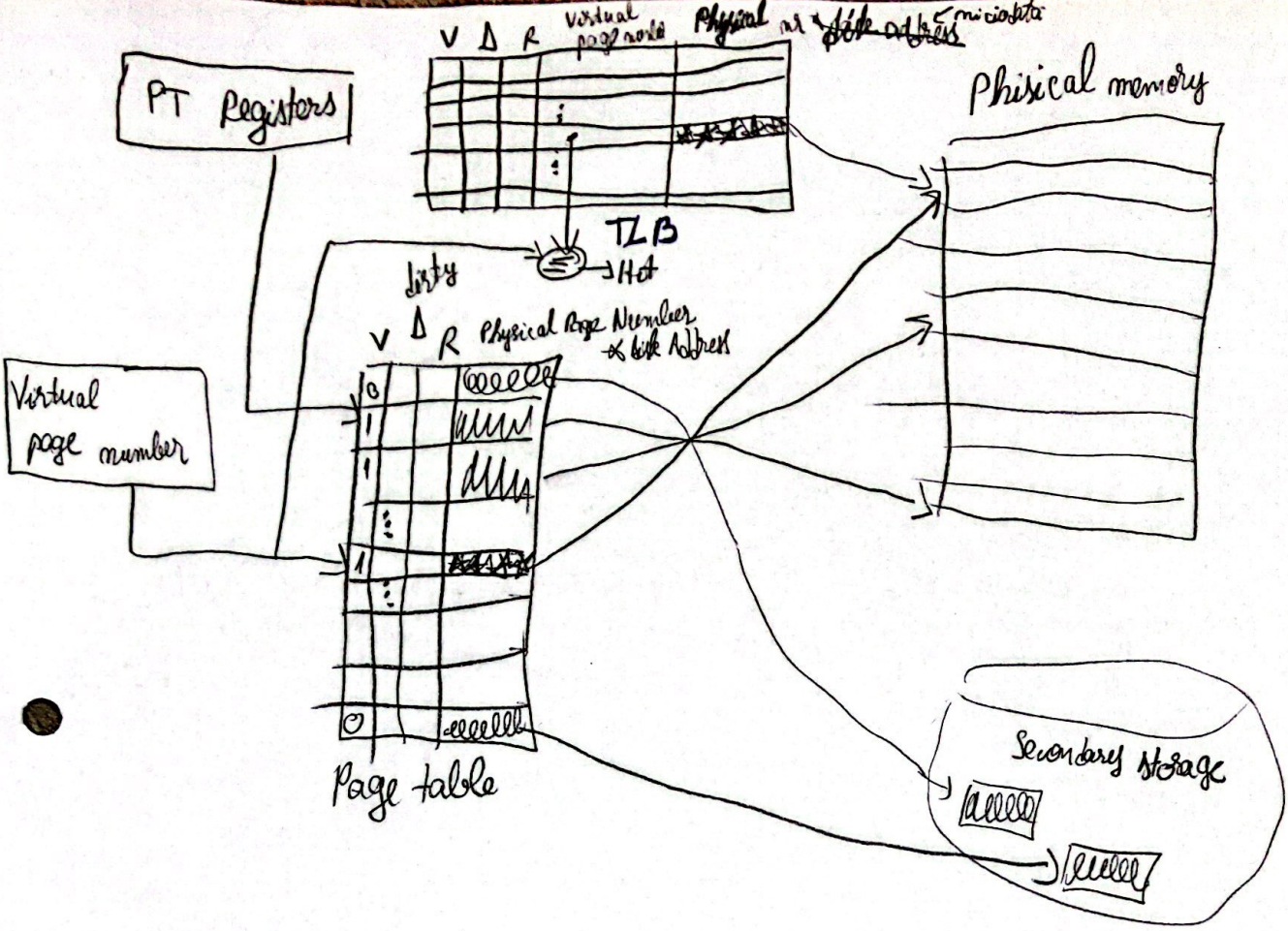
Page table mecanism → Performance overhead

1 acces mem = 2 de spt

⇒ Page Table Caching


47 ... 12 11 ... 0
Page table size reduction
③ Hashing → Inverted page table
④ Multiple-level tables
⑤ Paged page table
ARM V8
PD - Page Descriptor
VA [47:39]
VA [38:30]
VA [29:21]
VA [20:12]
PD
level 0
level 1
level 2
Translation Table Base Register
level 2 Page Table Size = 2^9 entries × 2^3 B = 2^12 B = 4 KiB


Virt. Page No.
PO
47 ... 12 11 ... 0
ARM V8
VA [47:39]
VA [38:30]
VA [29:21]
VA [20:12]
PD
level 0
level 1
level 2
level 3
Base Register
= 2^9 entries × 2^3 B = 2^12 B = 4 KiB



exemple 2

Ynfinsdty's Fast Math

Page - 4 KiB

4 GiB Physical and Virtual Mem Size

TLB acc time = 1 clk cycle

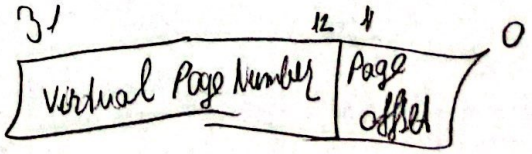
Miss Penalty TLB = 10 - 100 of clock cycles

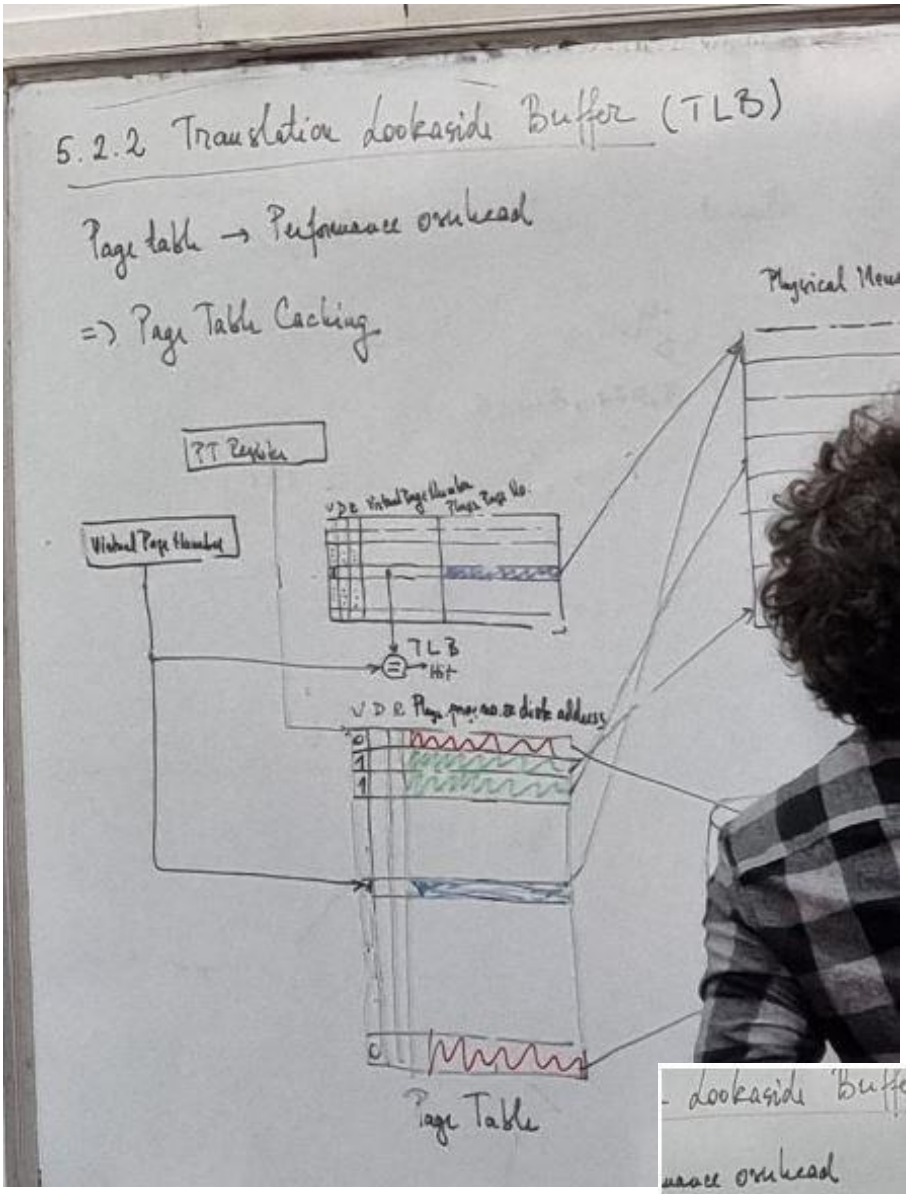
TLB size → 16 - 256 entries

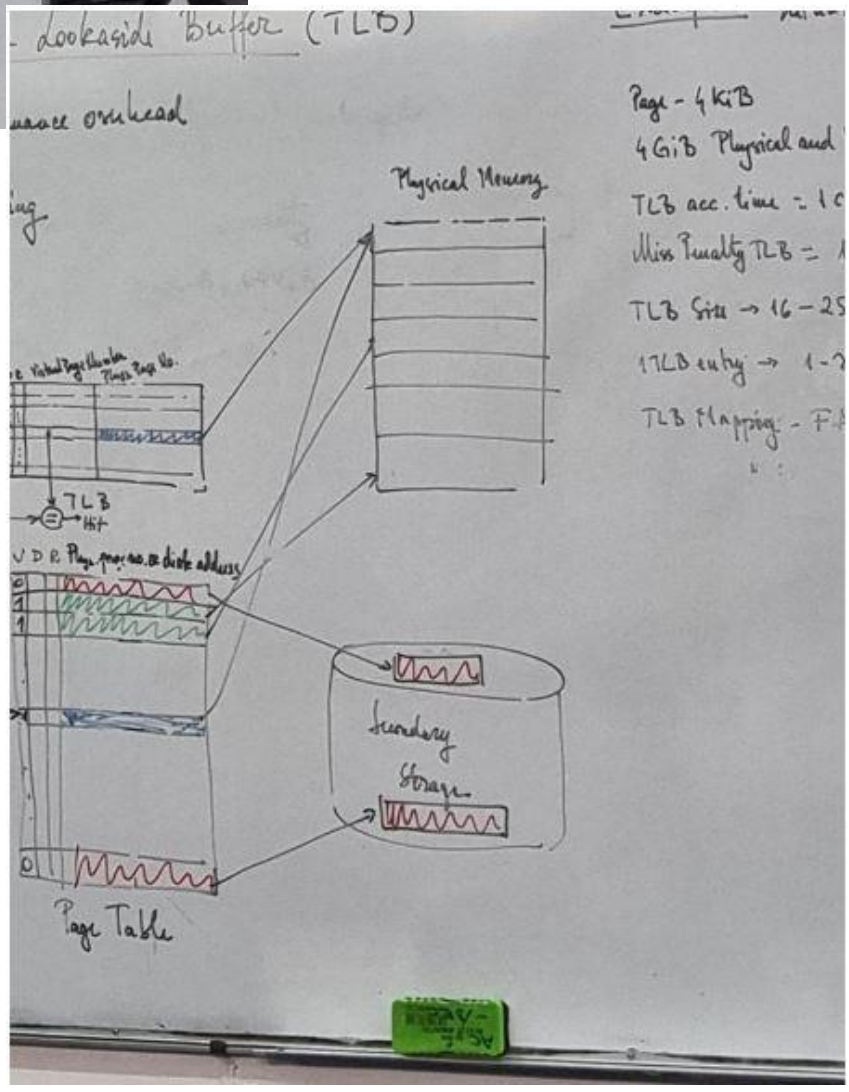
1 TLB entry → 1 - 2 pag table entries

TLB Mapping - FA

~~Word~~ Virtual Address




5.2.2 Translation Lookaside Buffer (TLB)
Page table → Performance overhead
=> Page Table Caching
Physical Memory
PT Register
Virtual Page Number
TLB
Hit
Page Table


Lookaside Buffer (TLB)
Physical Memory
Page - 4 KiB
TLB acc. time = 1
TLB Size → 16 – 25
TLB Mapping - F
Secondary Storage
Page Table