21.02.2023

ORGANIZAREA CALCULATOARELOR

Curs 1

Exam 67%

Proiect 33% < 50% laborator
50% proiect

La examen: probleme (open book) ♡
(max 5 pb)
2p / problemă

La laborator: 2 teste (pb. de antrenament)

La proiect:

LD → AC → OC → FIC

- performanța alg
- memorii

Computer science

Computer engineering

Software Engineering

Information Technology

$CPU_{time} = IC * CPI * Clock\ cycle\ time$

Power Wall

Networks-On-Chip

Federated Learning

- loi de la IO în BSO Consultații

Capitolul 1.

Booth's algorithm

$125_{[7]}$ → gunoi

0125_6 → gunoi

$0.12_{[7]}$ → gunoi

0.120_6

$Y \cdot X$

$X = x_{n-1} x_{n-2} \dots x_i \dots x_1 x_0 \rightarrow x_i \in B = \{0,1\}$

$X = -x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i$

$1010_{C_2} \rightarrow 2 - 8 = -6$

1001_{C_1}

1110_{SM}

-6_{10}

$(x_{i-1} - x_i) \cdot 2^i \cdot Y$

x_i	x_{i-1}	2^i 2^{i+1} OP
0	0	−
0	1	+Y
1	0	−Y
1	1	−

step 0: $(x_{-1} - x_0) \cdot 2^0 \cdot y +$

step 1: $(x_0 - x_1) \cdot 2^1 \cdot y +$

⋮

step i: $(x_{i-1} - x_i) \cdot 2^i \cdot y +$

⋮

step n-2: $(x_{n-3} - x_{n-2}) \cdot 2^{n-2} \cdot y$

$P = (-x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i) \times y$

$-x_i \cdot 2^i + x_i \cdot 2^{i+1} =$

$= (x_i \cdot 2^i)(-1+2)$

$= x_i \cdot 2^i$

Exemplu algoritmul lui Booth:

$X = -\frac{101}{128} = 1.1100101_{SM}$

$= 1.0011011$

$Y = -\frac{15}{32}$

$= 1.01111\,00_{SM}$

$= 1.1000100$

$101 - 64 = 37 - 32 = 5$

count	A	Q		M
000	00000000	10011011	0	11000100
+	00111100		-M	
	00111100			
	00011110	01001101	1	
001	00001111	00100110	1	
010	11000100		+M	
	11010011			
	11101001	10010011	0	
011 +	00111100		+M	
	00100101			
	00010010	11001001	1	

+M = 1100 0100

-M = 0011 1100

100		00001001	0110 0100	1
101 +		11000100		+M
		11001101		
		11100110	10110010	0
110		11110011	01011001	0
111 +		0011 1100		
		0010 1111	01011001	0

x_{i+1}	x_i	x_{i-1}	OP
0	0	0	-
0	0	1	+Y
0	1	0	+2Y
0	1	1	-2Y
1	0	0	-Y
1	0	1	-Y
1	1	0	-Y
1	1	1	-

$-Y \cdot 2^i + Y \cdot 2^{i+1} = +Y \cdot 2^i$

$+Y \cdot 2^{i+1} = 2Y \cdot 2^i$

28.02.2023

1. Computer Arithmetic

1.1. Speeding up multiplication Radix-4

x_i	x_{i-1}	OP
0	0	0
0	1	1
1	0	$\bar{1}$
1	1	0

$x = 10001101:0$

$x^* = \bar{1}001\,0\bar{1}1\bar{1}$

$x = 1111\,1100|0$

$x^* = 0000\,0\bar{1}00$

x_{i+1}	x_i	x_{i-1}	OP
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	$\bar{2}$
1	0	1	$\bar{1}$
1	1	0	$\bar{1}$
1	1	1	0

$x = -\frac{101}{128}$

$Y = -\frac{15}{32}$

$M = 11100\,0100_{C_2}$ $\quad 2M = 11000\,1000$

$-M = 00011\,1100_{C_2}$ $\quad -2M = 00111\,1000$

Count	A	Q	Q[-1]	M
00	0000 0000 0	10011011	0	11000100
+	0001 1110 0			
shiftez cu 2 poz ←	0001 1110 0			
	0000 0111 1	0010011 0	1	
01 +	0001 1110 0			
	0010 0101 1			
	0000 1001 0	1100100 1	1	
10 +	1100 0100 0			
	1100 1010			
	1111 0011 0	1011001 0	0	
11 +	0011 1100 0			
	0010 1111 0	1011001 0		

se transformă 1 în 0 la final

$x = 1001\,1011:0$ $\quad 1001\,1011:0$

$x^* = \bar{1}010\,\bar{1}10\bar{1}$ Booth codificat

$x^{**} = \bar{2}\ 2\ \bar{1}\ \bar{1}$

odul 2

1.2. Redundant digit set representation

Radix 2 : $\{0,1\} \rightarrow \{\bar{1},0,1\}$

Radix 4 : $\{0,1,2,3\} \rightarrow \{\bar{2},\bar{1},0,1,2\}$

$10\bar{1}1 = 1\cdot 2^0 + 0\cdot 2^2 + 1\cdot 2^3 = 7$

$0 \{ 0, 0$

$1 \{ 1, 0$

$\bar{1} \{ 0, 1$

$+3\ 2\ 1\ 0$

$102\bar{2}\bar{1} = 1\cdot 4^4 + 2\cdot 4^2 - 2\cdot 4^1 - 1\cdot 4^0$ – Radix 4

$= 279$

$0 \rightarrow \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix}$

$1 \rightarrow \begin{matrix} 0 & 1 \\ 0 & 0 \end{matrix}$

$2 \rightarrow \begin{matrix} 1 & 0 \\ 0 & 0 \end{matrix}$

$\bar{1} \rightarrow \begin{matrix} 0 & 0 \\ 0 & 1 \end{matrix}$

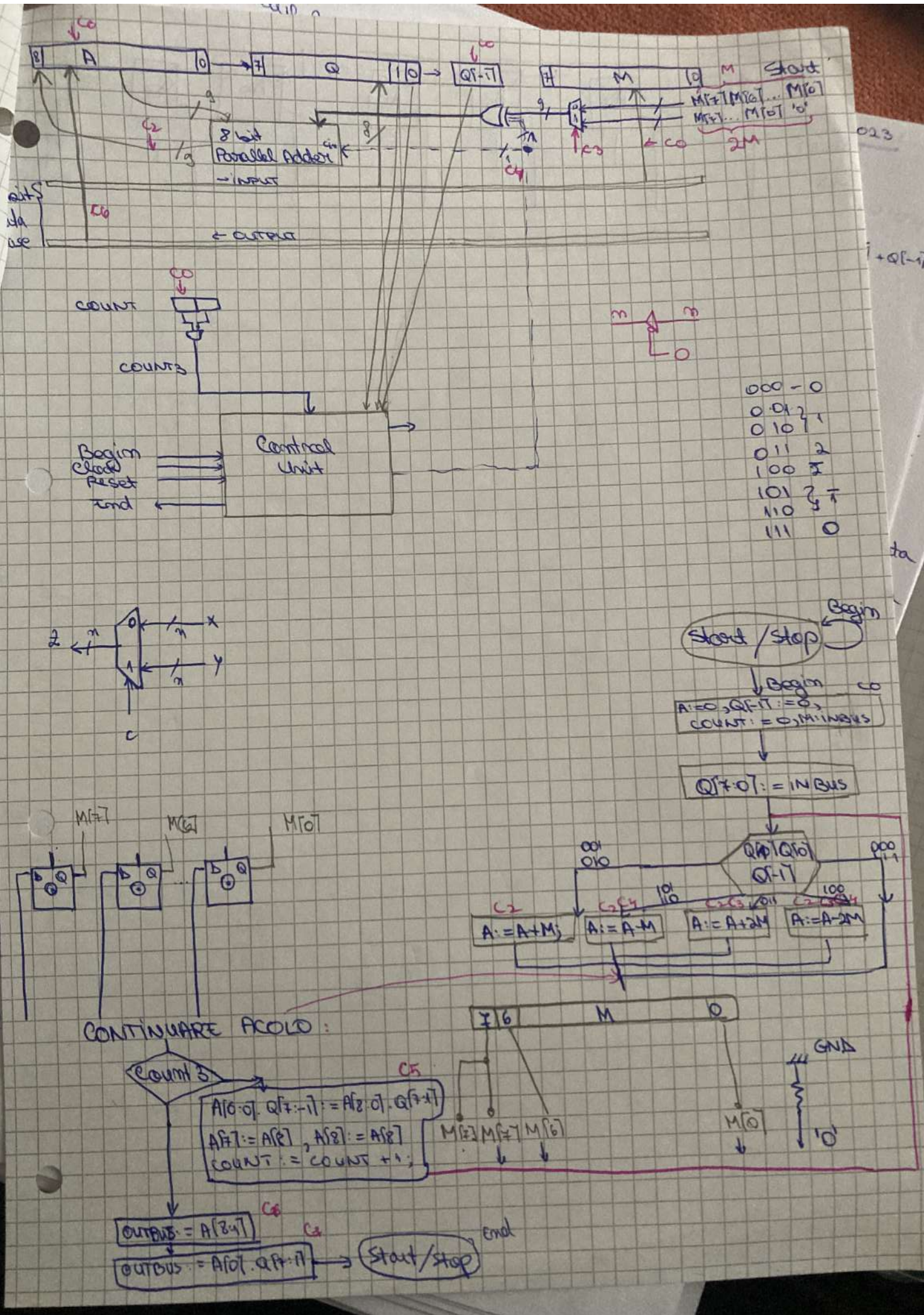
$\bar{2} \rightarrow \begin{matrix} 0 & 0 \\ 1 & 0 \end{matrix}$

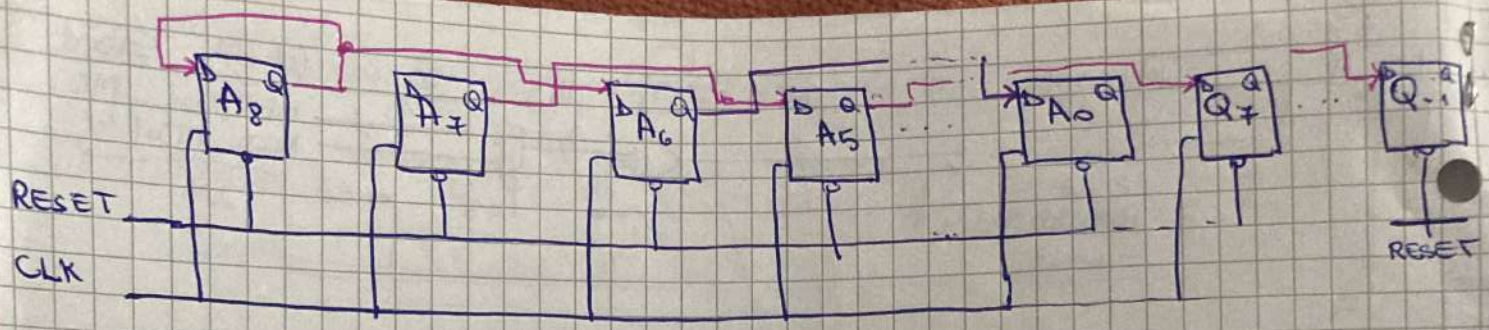
$1001 -$
0010
$\overline{0111}$

$01\ 00\ 10\ 00\ 00 -$
$00\ 00\ 00\ 10\ 01$
$\overline{01\ 00\ 01\ 01\ 11}$

$23 +$
256
$\overline{279}$

1.3. Hardware implementation of radix 4 – Booth


A
Q
Q[-1]
M
Start
8 bit Parallel Adder
INPUT
OUTPUS
COUNT
COUNT=3
Control Unit
Begin
Clock
Reset
End
000 - 0
001
010
011 2
100
101
110
111 0
Start / Stop
Begin
A:=0, Q[-1]:=0, COUNT:=0, M:=INBUS
Q[7:0]:=INBUS
A:=A+M
A:=A-M
A:=A+2M
A:=A-2M
CONTINUARE ACOLO:
COUNT=3
A[0].Q[7:-1]:=A[8:0].Q[7:1]
A[7]:=A[8], A[8]:=A[8]
COUNT:=COUNT+1;
OUTBUS:=A[8:1]
OUTBUS:=A[0].Q[7:1]
End
GND
'0'


A8
A7
A6
A5
A0
Q7
Q-1
RESET
CLK
RESET

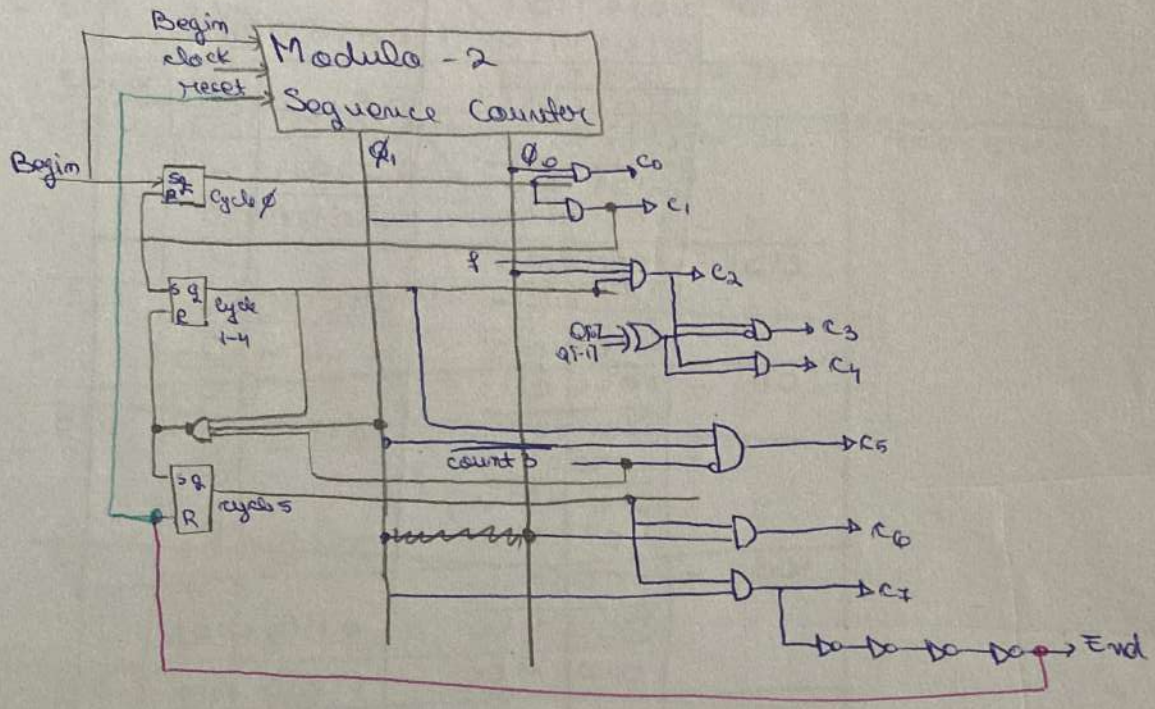
Curs 3

7.03.2023

Episodul 2

Sequence Counter

CU sythesis



(f)

Q_{i-1} \ $Q_i Q_{i-1}$	00	01	11	10
0	0	1	1	1
1	1	1	0	1

$f = Q_i\overline{Q_{i-1}} + Q_i\overline{Q_i} + Q_{i-1}Q_i$

1.3 Division

Dividend: 5771 | Divisor: 135 | Quotient: 42

5771 − 540 = 371; 371 − 270 = 101 (Reminder)

Binary

1010 0110 | 1001 | 10010 (18)

1010 − 1001 = 0001011; 1011 − 1001 = 00100 (4)

38 + 128 = 166

166 | 9 | 18

166 − 9 = 76; 76 − 72 = 4

Size of the quotient = size of the diversor = n bits

⇒ Dividunt size = 2n − 1 bits

1.3.1.

Restoring division

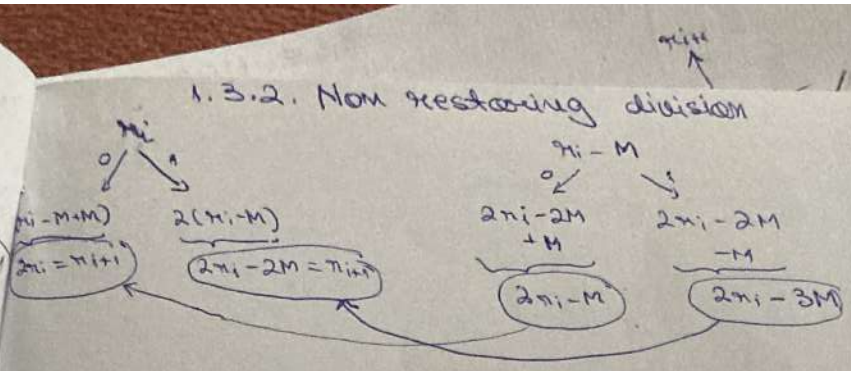
$1011010001011 = 5771_{10}$

$135_{10} = 10000111$

COUNT	A	Q	M
000	00101101	00010111	10000111
−	10000111		
	10100110	00010111	
+	10000111		
	00101101		
	01011010	00101110	
001	10000111		
−	10100111	00101110	
+	10000111		
	01011010		
	10111000	01011000	
010	10000111		
−	10101101	01011001	
	01011010		
	10111011	10110010	
011	10000111		
−	10100011	10110010	
+	10000111		
	01011010		
	10111001	01100100	
100	10000111		
−	10101110	01100101	
	01011100		
	11001010	11001010	
101	10000111		
−	10100101	11001010	
+	10000111		
	01011100		
	10111001	10010100	
110	10000111		
−	10110010	10010101	
	01110101		
	10100101	00101010	
111	10000111		
−	10101110	00101010	
+	10000111		
	01110101		
	101_{10}	42_{10}	

Exemplu: $n = 8 \Rightarrow$ 13 scriit. op.

1.3.2. Non restoring division



64 + 37 / 101

1.3.3 Division

COUNT	S	A	Q	M
000	0	0010 1101	0001 0111 | 0	1000 0111
	−	1000 0111		
	(1)	1010 0110		
	1			
001	+			
	(1)			
	1			
010	+ 0			
	(0)			
	0			
011	−			
	(1)			
	1			
100	+			
	(0)			
	0			
101	−			
	(1)			
	1			
110	+			
	(0)			
	0			
111	−			
	(1)			
	+			

SRT division

14.03.2023

Class 4

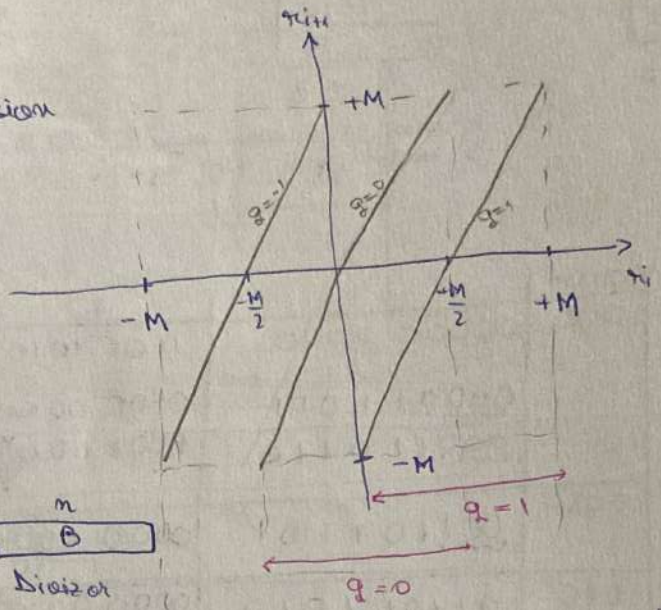
$r_{i+1} \leftarrow 2r_i - q_i M$ (M: divisor)

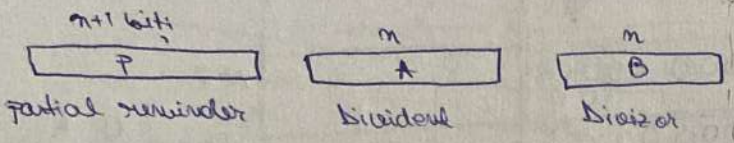
$|r_i| < M$

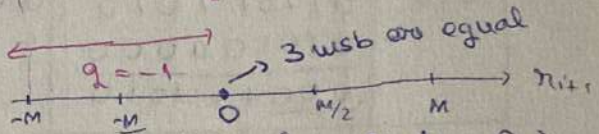
$\downarrow$

$|r_{i+1}| < M$

$q_i \in \{-1, 0, 1\}$







1. Shifting over zeros — Shift P.A.B k positions (k leading 0s)
2. For i=0 to n-1
   a) if 3 msb of P are equal
      $q_i = 0$
      Shift P.A one position left
   b) if 3 msb are not equal and P positive
      $q_i = 1$
      Shift P.A one position left
      $P \leftarrow P - B$
   c) if 3 msb are not equal and P negative
      $q_i = -1$
      Shift P.A. one position left
      $P \leftarrow P + B$
3. Correction
   if $P < 0$
      $P \leftarrow P + B$
      $Q \leftarrow Q - 1$
4. Shift P right k positions

213 | 5
20 | 42
= 13
10
= 3

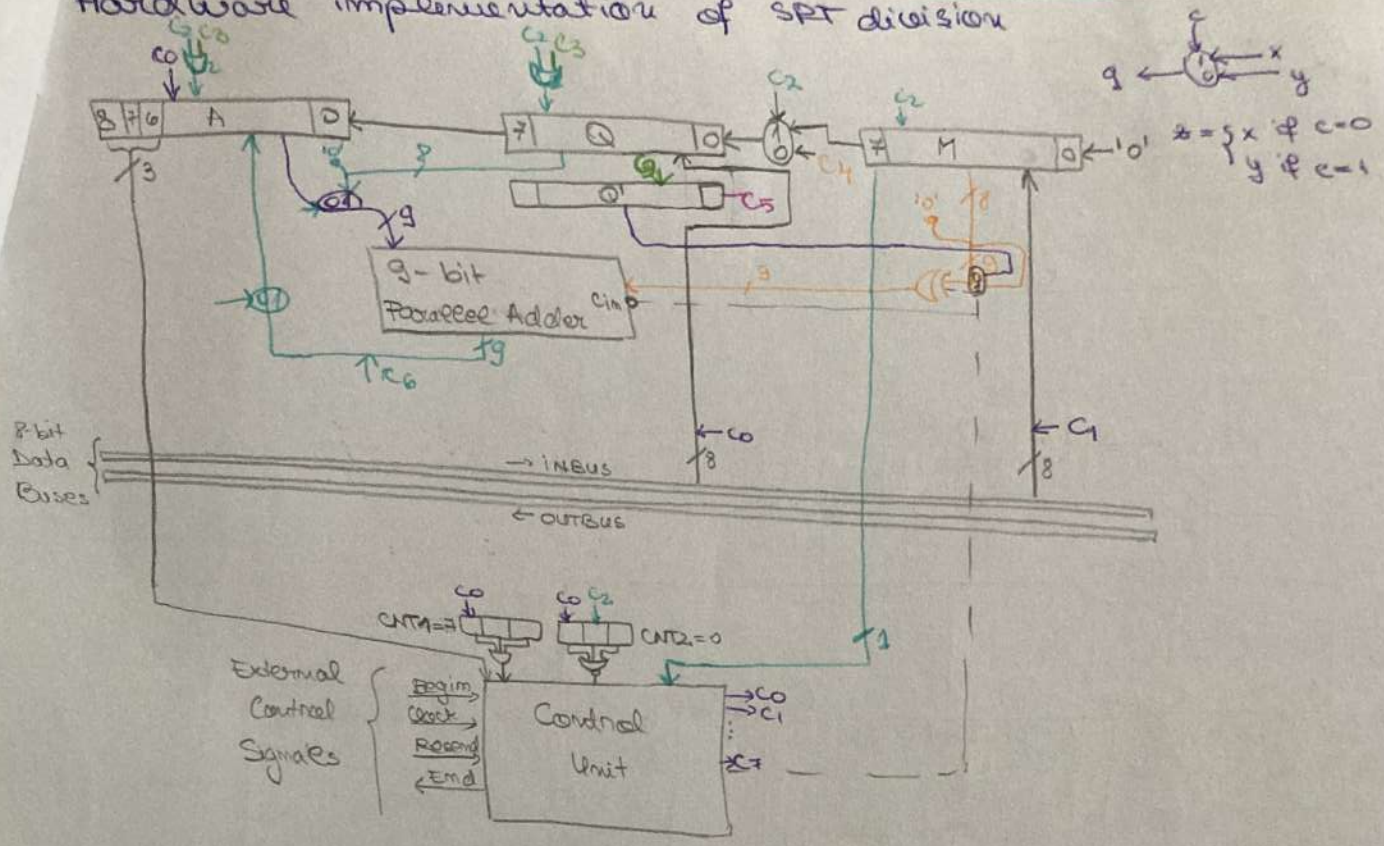
218 | 5 ← K
20 | 43
= 18
15
= 3

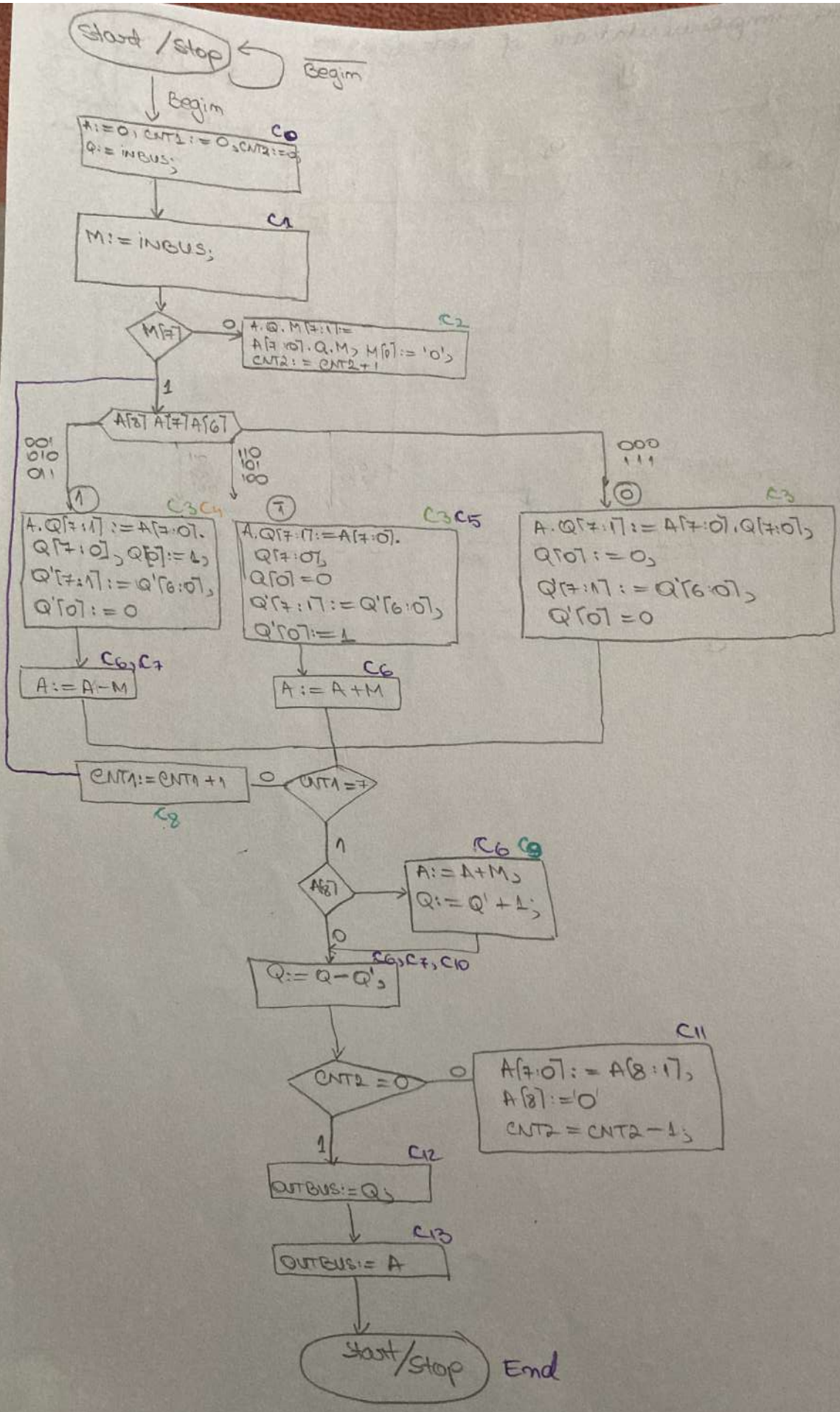
218_{10} = 1101 1010 = A
45_{10} = 0010 1011
5_{10} = 0000 0101 = B
3_{10} = 0000 0011

3 msb of P are equal ⇒ $q_i = 0$
3 msb of P NOT equal, P > 0 ⇒ $q_i = 1$, shift și P = P - B

COUNT	P	A	B	
000	00000 0000	1101 1010	0000 0101	K = 5
	00001 1011	0100 0000	1010 0000	<< 5
	00011 0110	1000 0000		<< 1
001	00110 1101	0000 0000		<< 1
010	01101 1010	0000 0001		0 → 0 A
	- 0101 0000 0	0000 0000		<< 1 0 A'
	00011 1010			1 → 1 A
				0 A'
				T → 0 A
				1 A'

Hardware implementation of SRT division





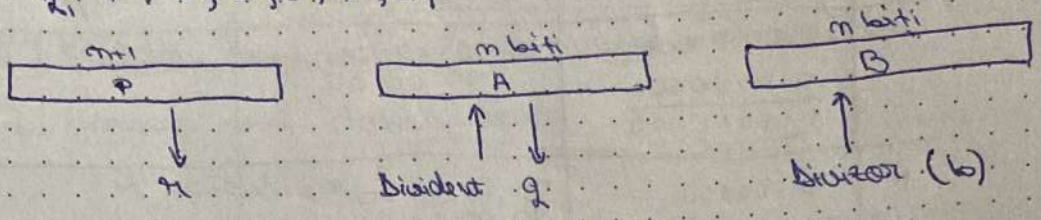
21.03.2023

Curs 5

1.4.5 Speeding up SRT Division

SRT Radix-4

$q_i \in \{\bar{2}, \bar{1}, 0, 1, 2\}$



$|r_i| < b \quad , \quad |r_{i+1}| < b$

$r_{i+1} \leftarrow 4r_i - q_i b$

$r_i = b$

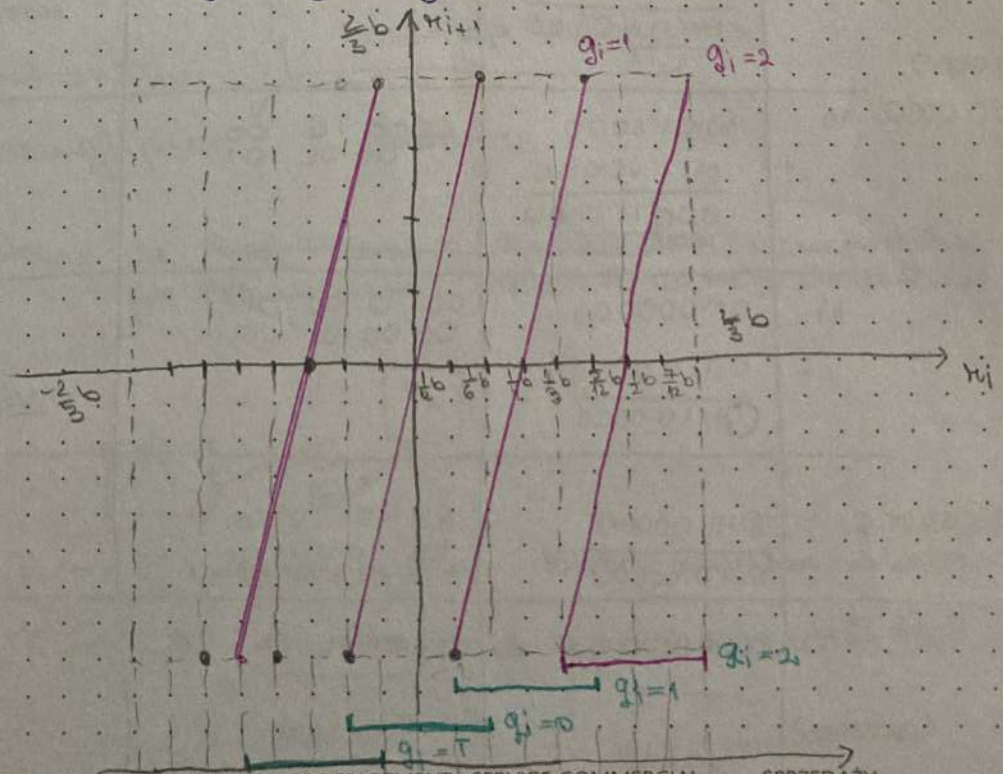
$r_{i+1} = 4b - q_i b$

Dacă am pune $|r_i| < \frac{2}{3} b$

$r_i = \frac{2}{3} b$

$r_{i+1} = \frac{8}{3} b - q_i b \qquad q_i = 2$

$\Rightarrow r_{i+1} = \frac{8}{3} b - \frac{6}{3} = \frac{2}{3} b$



HOMANIT GMBH & CO. KG
Bahnhofstraße 30
37412 Herzberg am Harz

VERTRIEB · SALES DEPARTMENT · SERVICE COMMERCIAL
Tel. +49 5521 84-0 · Fax +49 5521 84-269
info@homanit.de · www.homanit.de

SPRZEDAŻY
Tel. +48 94 3100571
info@homanit.pl · www.homanit.pl

211 | 6
18 | 35
=31
30
=1

COUNT	P	A	B
00	00000 0000 0 0001 1010 0011 01, 001	1101 0011 0110 0000 1000 00 00 00	0000 0110 1100 0000 b=12
01	1101 00110 - 1100 00000 0001 00110	0000 00 10 00 00	
10	0100 11 000 - 0110000000 1110 11 000	0000 10 01 00 00 00	
11	1011 00000 + 0110000000 0001 00000	00 10 01 00 00 00 00 01	
CONVERSIE SI SHIFTARE	0000 0 0001 Rest	00 100011 Quotient	

shiftez dreapta

0 → 00
00

1 → 01
00

2 → 10
00

3 → 00
01

5 → 00
10

202 | 7
14 | 28
=62
56
=6

COUNT	P	A / A'	B
00	00000 0000 0 0001 1001 0 0110 0101	1100 1010 0100 0000 0000 0000 00	0000 0111 1110 0000 b=14
01	1100 1010 00 - 111 0000000 1110 1010 0	00 00 00 10 00 00	
10	1010 1 0000 + 0111 0000 00 0001 1000 0	0000 10 00 00 00 01	
11	0110000000 - 0111 0000 0 1110 0000	00 10 00 01 00 00 01 00	
CORECTIE +	0111 00000 0110 00000	00 10 00 01 00 00 01 01	
shift dr	000000110	00011100	
Conversie	Rest	Quotient	

B = 01110 0000
2B = 1110 0000

Cap. 2

Performanța sistemelor de calcul

→ 3 tipuri de sisteme de calcul

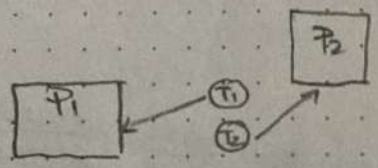
1. Desktop computers — 40 years of progress
2. Servers and datacenters
   - scientific
   - many users at the same time
3. Embeded computers

Performanța
- Time
- Space (Memory, circuits)
- Power consumption
- Dependability

Runtime
Execution Time } Desktop computers
Throughput — servers

Sisteme de Real Time - Embeded computers
- performanță: worst case: deadline

Exemplu:



① $P_1 \rightarrow P_1'$ better runtime ⇒ better runtime ⇒ better ~~throughput~~ throughput

② $P_1 \rightarrow P_1, P_2$ ⇒ better throughput ⇒ better runtime

HOMANIT GMBH & CO. KG
Bahnhofstraße 30
37412 Herzberg am Harz

VERTRIEB · SALES DEPARTMENT · SERVICE COMMERCIAL
Tel. +49 5521 84-0 · Fax +49 5521 84-269
info@homanit.de · www.homanit.de

SPRZEDAŻY
Tel. +48 94 3100571
info@homanit.pl · www.homanit.pl

$$\text{Performance}_x = \frac{1}{\text{Execution time}_x}$$

Exemplu :

Computer A rulează o program în 15 secunde

Computer B rulează the same program în 10 secunde

$$\frac{\text{Perf. B}}{\text{Perf. A}} = \frac{\text{Exec. time A}}{\text{Exec time B}} = \frac{15}{10} = 1{,}5$$

Benchmark suits

Curs 6

2.2. Performance metrics

$$\text{Performance}_x = \frac{1}{\text{Execution time}_x}$$

- Real programs (win)
- Not
  - toy programs
  - synthetic
  - Kernels

Dec	Binary	
10^3	2^{10}	1K
10^6	2^{20}	1 Mi
10^9	2^{30}	1 Gi
10^{12}	2^{40}	1 Ti
10^{15}	2^{50}	1 Pi peta
10^{18}	2^{60}	1 Ei exa
10^{21}	2^{70}	1 Zi zetta
10^{24}	2^{80}	1 Yi yotta

- transitivity $A > B$ & $B > C$

$\Rightarrow \boxed{A > C}$

Benchmark suites

SPEC : Standard Performance Evaluation Corporations

1989, 92, 95, 2000, 2006, 2017

- CPU SPEC
- SPEC graphics
- SPEC Java

2.2.1. Reporting performance results

SPEC Ratio

$$\frac{\text{SPEC Ratio A}}{\text{SPEC Ratio B}} = \frac{\frac{\text{Execution time reference}}{\text{Execution time A}}}{\frac{\text{Execution time reference}}{\text{Execution time B}}} =$$

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

HOMANIT GMBH & CO. KG
Bahnhofstraße 30
37412 Herzberg am Harz

VERTRIEB · SALES DEPARTMENT · SERVICE COMMERCIAL
Tel. +49 5521 84-0 · Fax +49 5521 84-269
info@homanit.de · www.homanit.de

SPRZEDAŻY
Tel. +48 94 3100571
info@homanit.pl · www.homanit.pl

Geometric mean $= \sqrt[n]{\prod_{i=1}^{n} \text{Sample } i}$ → SPEC ratio i ↓ program

1. The geometric means of ratios = the ratio of geometric means
2. The ratio of geometric means

$$\frac{\text{Geometric mean A}}{\text{Geometric mean B}} = \frac{\sqrt[n]{\prod_{i=1}^{n} \text{Spec Ratio A}_i}}{\sqrt[n]{\prod_{i=1}^{n} \text{SPEC ratio B}_i}} = \quad n = 29$$

$$= \sqrt{\prod_{i=1}^{n} \frac{\text{SPEC Ratio A}_i}{\text{SPEC ratio B}_i}} = \sqrt{\prod_{i=1}^{n} \frac{\frac{\text{Execution time ref.}}{\text{Execution time A}}}{\frac{\text{Execution time ref.}}{\text{Execution time B}}}}$$

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Execution time B}_i}{\text{Execution time A}_i}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Performance A}_i}{\text{Performance B}_i}}$$

$$\text{stdev} = \sqrt{\sum_{i=1} (\text{Sample}_i - \text{Mean})^2}$$

↑ SPEC ratio i

Geometric standard deviation

$$\text{Geometric mean} = \exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} \ln(\text{Sample}_i)\right)$$

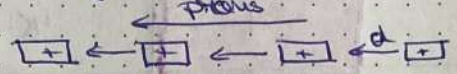
$$\text{gstdev} = \exp\left(\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} \left[\ln(\text{sample}_i) - \ln(\text{Geom. mean})\right]^2}\right)$$

2.3. Quantitative principles of computer design

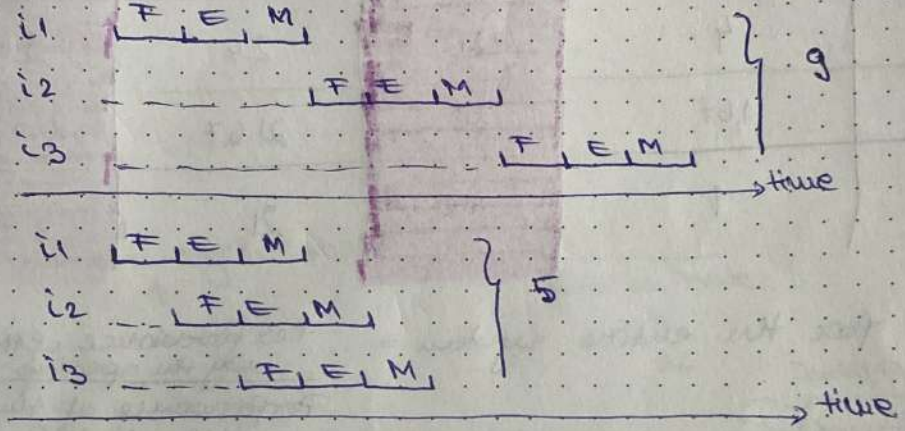
SLD - system level design

1. Take advantage of parallelism (when possible)

CLA



Instruction - Level parallelism



2. Principle of locality

Obs. Programs tend to reuse data and instructions they have used recently.

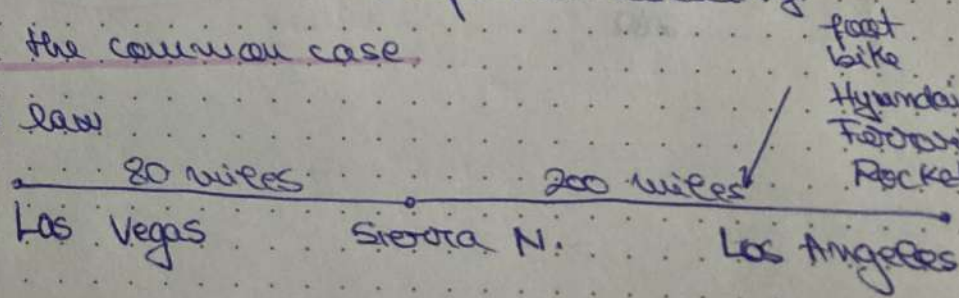
Rule of ...

A program spends 90% of the time on 10% of the code

- Mostly applied on instructions
- Also works on data
- 2 types of locality → temporal locality
  → spatial locality

3. Focus on the common case

Amdahl's law



Vehicle for the second part

Vehicle for the 2nd part	Hours for the 2nd part	Speed up in the desert	Hours for the entire trip	Speedup for the entire trip
Feet	50	1	40	1
Bike	20	2.5	70	1.75
Hyundai	4	12,5	24	2.9
Ferrari	1,67	30	21.67	3.23
Rocket car	1	50	21	3.3

Speedup for the entire system = $\frac{\text{Performance of the task using the speedup when possible}}{\text{Performance of the task without the speedup}}$

$= \frac{\text{Execution time of task without speedup}}{\text{Execution time of task using speedup when possible}} =$

$= \frac{\text{Execution time of task without speedup}}{\text{Execut. time of task without speedup}\left[(1-\text{Fraction enhanced}) + \frac{\text{Fraction enh.}}{\text{Speedup}}\right]}$

$= \frac{1}{(1-\text{Fraction enhanced}) + \frac{\text{Fraction enhanced}}{\text{speedup}}}$

$\frac{200}{280}$

$(1-\text{Fraction enhanced}) \rightarrow \frac{200}{280}$

$\frac{\text{Fraction enhanced}}{\text{speedup}}$ → Speedup in the desert

Exemplul 1

4.04.2023

HOMANIT

Limit of system enhancement

Speedup for the entire system =

$$= \frac{1}{1 - \text{Fraction enhanced}}$$

$\lim$ speedup $\to \infty$

2.4. Computer performance equation

Clock cycle time = $\frac{1}{\text{Clock rate}}$

↑ s

↑ Hz

Clock cycles for a program = Instruction count (IC) (Integer) × Clock cycles per Instruction (CPI)

CPU time = IC * CPI · Clock cycle time

$$\text{CPU time} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Clock cycles}}{\text{Instructions}} \cdot \frac{\text{Seconds}}{\text{Clock cycles}} = \frac{\text{seconds}}{\text{program}}$$

Improvement = reduce IC

RISC processors

CISC

HOMANIT GMBH & CO. KG
Bahnhofstraße 30
37412 Herzberg am Harz

VERTRIEB · SALES DEPARTMENT · SERVICE COMMERCIAL
Tel. +49 5521 84-0 · Fax +49 5521 84-269
info@homanit.de · www.homanit.de

SPRZEDAŻY
Tel. +48 94 3100571
info@homanit.pl · www.homanit.pl

4.04.2023

Curs 7.

= continuare =

Exemplul 1

- Frequency of FP operations = 25 %
- CPI_{FP} = 4 clock cycles
- CPI of other instructions = 1,33 clock cycles
- Frequency of FPSQRT = 2 %
- CPI_{FPSQRT} = 20 clock cycles

Design optimization alternative

a) decrease CPI_{FPSQRT} to 2 clock cycles

b) decrease CPI of all FP to 2.5 clock cycles

CPU time reference = $IC_x \cdot CPI_x \cdot$ clock cycle time x

$CPI_x = 0,25 \cdot 4 + 0,75 \cdot 1,33 = 2$ c.c

Alternative A

CPU time A = $IC_x \cdot CPI_A \cdot$ clock cycle time A

$CPI_A = 2 - 18 \cdot 0,02 = \boxed{1,64 \text{ c.c}}$

Amdahl's Law

Alternative B

CPU time B = $IC_x \cdot CPI_B \cdot$ clock cycle time x

$CPI_B = 0,25 \cdot 2,5 + 0,75 \cdot 1,33 = \boxed{1,625 \text{ cc}}$ ✓

Curs nr 8

Exemplul 2

CPU_A RISC

20 % au branch instruction
=> 2 % compare instruction

$CPI_{branch} = 2$ cc

$CPI_{all\ other} = 1$ cc

$a \to r_1$
$b \to r_2$

```
CMP r1,r2
BNE address
...} 1
B address#2
address# ...} 4
address2 ...
```

if a=b then
~~jump adress~~ } 1
else
- - - -
~~address~~ - - - } 2
- - - -

CPU_B CISC

BNE r_1, r_2, address

Clock cycle time is extended with 25 %

$CPU\ time_A = IC_A \cdot CPI_A \cdot$ Clock cycle time A = $IC_A \cdot 1{,}2 \cdot$ Clock cycle time A

se degradează clock-ul => crește timpul

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

$CPI_B = IC_B \cdot CPI_B \cdot$ Clock cycle time B

$IC_B = (100 - 20)\ \%\ IC_A$

Clock cycle time B = 1,25 · Clock cycle time A

20 ... 100
x ... 80

$x = \frac{80 \cdot 20}{100} = 16$

20 ... 80
x ... 100

$x = \frac{20 \cdot 100}{80}$ $\quad x = 25\ \%$

4

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

$CPU\ time_B = IC_A \cdot 0{,}8 \cdot 1{,}25 \cdot 1{,}25 \cdot$ Clock cycle time A
$= IC_A \cdot 1{,}25 \cdot$ Clock cycle time A

Concluzie: Sometimes less is more ♡!

Example 3

Load - store machine

```
LDR  r3, [r4]
ADD  r2, r2, r3
STR  r2, [r5]
```

Register memory

```
ADD  r2, r3, [r4]
```

instruction

Instruction	CPI	Frequency
ALU	1	43 %
Load	2	21 %
Store	2	12 %
Branch	2	24 %

25 % of ALU instructions have an operand taken from memory and placed in a register

New machine

- a new type of ALU instructions
- ALU CPI = 2 cc
- Branch CPI = 3 cc

Rezolvare:

$CPU\ time_{old} = IC_{old} \cdot CPI_{old} \cdot Clock\ cycle\ time_{old}$

$= IC_{old} \cdot 1,57 \times Clock\ cycle\ time_{old}$ ✓

$CPI_{old} = 1 \cdot 0,43 + 2 \cdot 0,57 = 1,57$

~~CPU time new =~~

$IC_{new} = IC_{old}\ (1 - 0,25 \cdot 0,43)$

$Clock\ cycle\ time_{old} = Clock\ cycle\ time_{new}$

~~$(0,43 \cdot 0,75) \cdot 1 + (0,43 \cdot 0,25) \cdot 2 + (0,21 - 0,43 \times 0,25) \cdot 2$~~

~~$+ 0,12 \times 2 + 0,24 \cdot 3$~~

~~CPI new =~~

$$CPI_{new} = \frac{(0,43 \cdot 0,75) \cdot 1 + (0,43 \cdot 0,25) \cdot 2 + (0,21 - 0,43 \cdot 0,25) \cdot 2 + 0,12 \cdot 2 + 0,24 \cdot 3}{1 - 0,43 \cdot 0,25} \quad (*)$$

$$CPU\ time_{new} = IC_{old} \cdot (1 - 0,25 \cdot 0,43) \cdot \frac{(*)}{(1 - 0,43 \cdot 0,25)} \cdot Clock\ cycle\ time_{old}$$

$= IC_{old} \times 1,908 \times Clock\ cycle\ time_{old}$ ✗

2.5. Other metrics

MIPS = Million Instructions Per Second

$$MIPS = \frac{IC}{CPU_{time} \cdot 10^6} = \frac{IC}{IC \cdot CPI \cdot \text{clock cycle time} \cdot 10^6} =$$

$$= \frac{\text{Clock rate}}{CPI \cdot 10^6}$$

Problems with MIPS

1. Cannot compare 2 components with distinct instruction sets
2. MIPS varies for the same computer for different programs
3. Sometimes MIPS is misleading.

Exemplul 4 The same L/S machine from Exemplul 3

Instructions	Frequency	CPI
ALU	43%	1
Load	21%	2
Store	12%	2
Branch	24%	2

Clock cycle time = 20 ns

ALU ops reduced with 50% by a compiler optimization

$$\text{Frequency} = \frac{1}{20\,ns} = \frac{1}{20 \cdot 10^{-9}\,s} = 50\ MHz$$

$$CPU_{time\ unoptimized} = IC_u \times CPI_u \times 20ns = IC_u \times 1{,}57 \times 20ns$$

$$= \boxed{IC_u \times 31{,}4\ ns}$$

$$MIPS_u = \frac{50 \cdot 10^6\ Hz}{1{,}57 \times 10^6} = \frac{50}{1{,}57} = \boxed{31{,}85}$$

18.04.2023

Cursul 8

Exemple

Instruction type	Frequency	CPI
ALU	43%	1
LOAD	21%	2
STORE	12%	2
BRANCH	24%	2

Clock cycle time = 20ns

Clock frequency = $\frac{1}{20ns} = 50Hz$

CPU time original = $IC \cdot CPI_{old} \cdot 20ns = IC_0 \cdot 1,57 \cdot 20 = \boxed{3,14\, IC_0}$

$CPI_{old} = 0,43 \cdot 1 + 0,57 \cdot 2 = 1,57$

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

$IC_{new} = IC_0 \left(1 - \frac{0,43}{2}\right) \Rightarrow CPI_{new} = \frac{\frac{0,43}{2} \cdot 1 + 0,57 \cdot 2}{\left(1 - \frac{0,43}{2}\right)} = 1,73$

$CPU_{new} = IC_0 \left(1 - \frac{0,44}{2}\right) \frac{\frac{0,43}{2} \cdot 1 + 0,57 \cdot 2}{\left(1 - \frac{0,43}{2}\right)} \cdot 20ns = \boxed{IC_0 \cdot 2,71\, ns}$

(Mai bună)

$MIPS_{new} = \frac{50 \cdot 10^6}{1,73 \cdot 10^6} = 28,9$

Relative MIPS = $\frac{\text{Execution time reference machine}}{\text{Execution time x}} \cdot MIPS_{\text{ref. machine}}$

$MFLOPS = \frac{IC_{fp}}{CPU_{time} \cdot 10^6}$

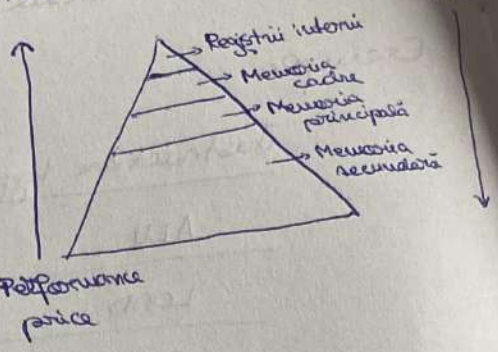
Livermore Loops

Floop	Norm Floop
+/-/.	1
Sqrt	2
Exp Sin	8

Curs 9

3. Memory hierarchy

Capitolul III : Ierarhia de memorie

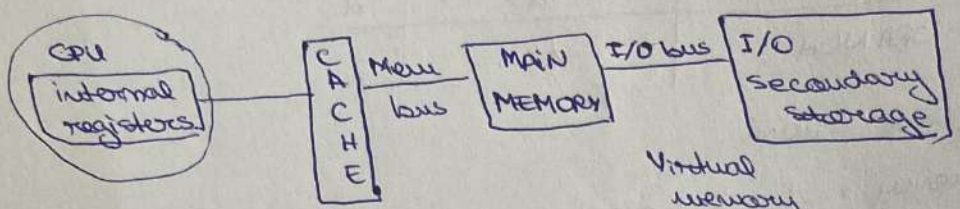


3.1. Introduction

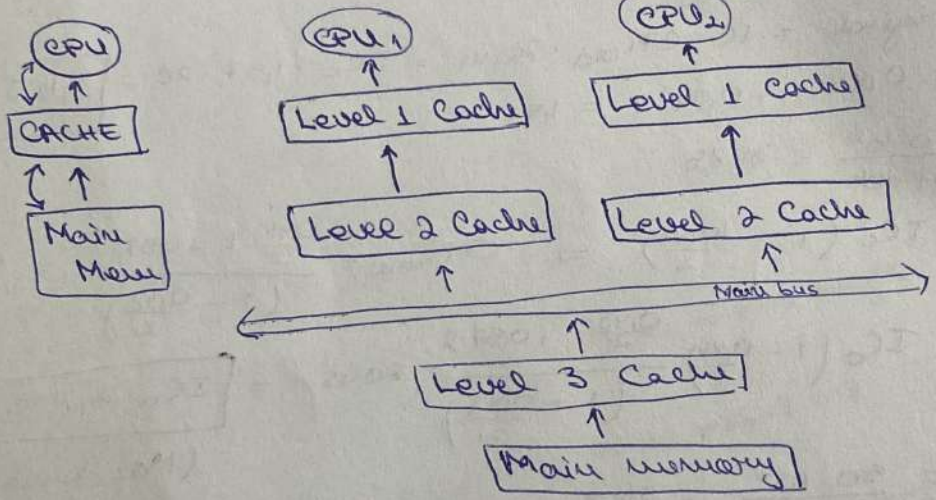
3.2. Cache mapping

Locality Principle

Make the common case faster



Virtual memory



Taxonomy & basic notions

Addressable unit → generally 1B = 1 Byte

1 word = 2^n bytes

1 block = 2^p words

1 cache hit

1 cache miss → Miss Rate ≅ 10%.

Cache coherency

Cache - unified

- split → Data
  → Instruction

3.2.1. Direct mapping

2^m blocks in M_2 (Main Memory)

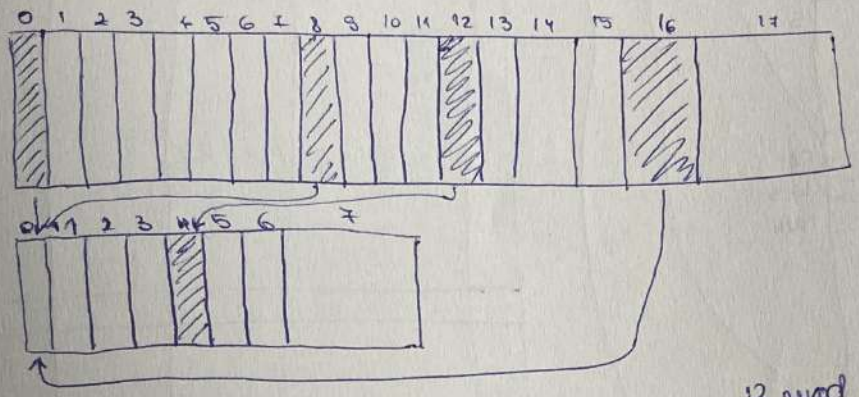
2^n blocks in M_1 (Cache)

$m < n$

j indexes the blocks in the main memory

i indexes the blocks in the cache M_1

$i = j$

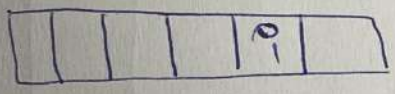


12 mod 8 = 4

01 100

Tag Index

Tag



Adress word:

Tag	Index	word offset	Byte offset
<m-w>	<w>	<p>	<r>

Exemplul 1:

Memoria principala: 2^{24} octeti → nr. of bits in the adress word

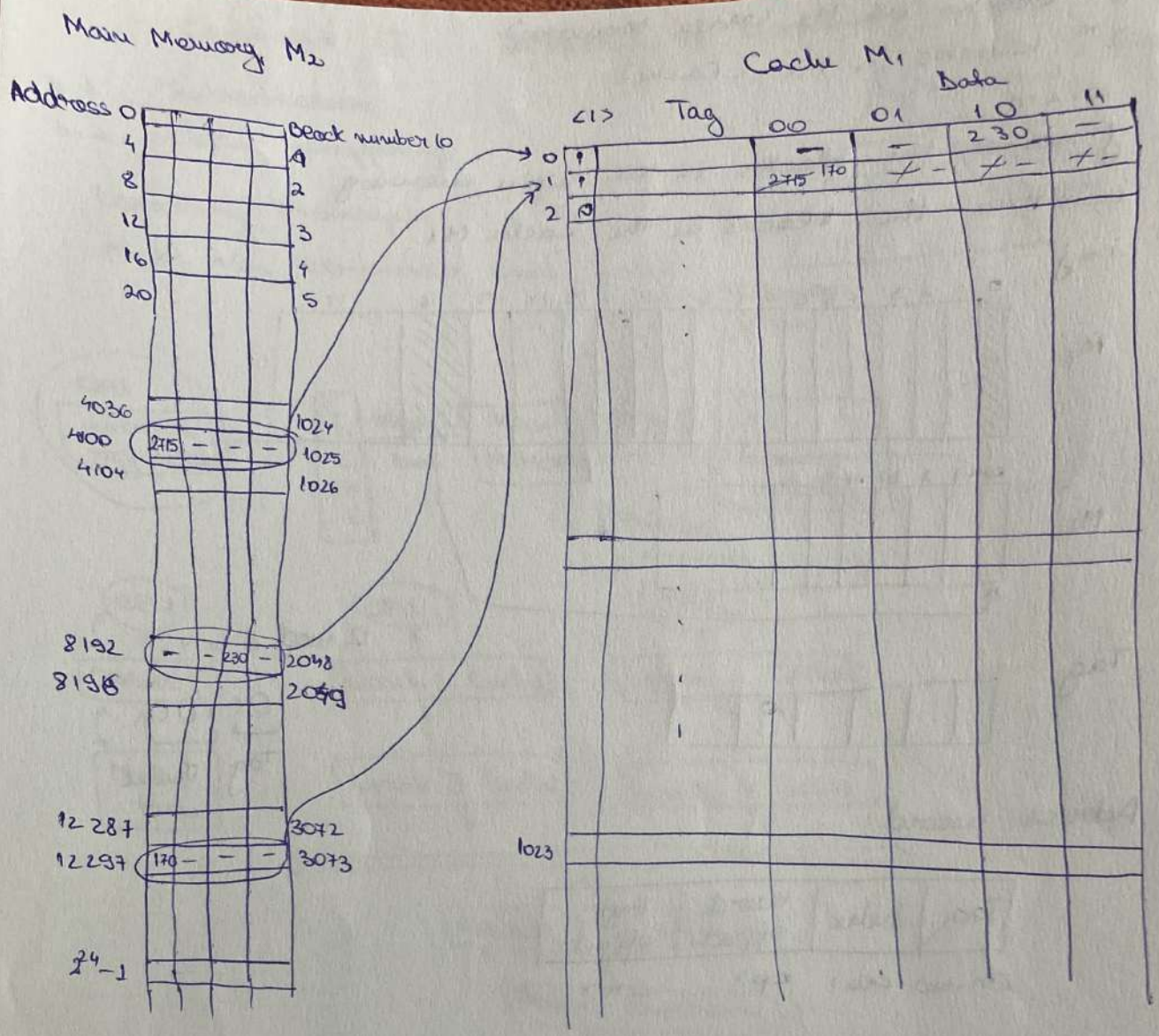
1 cuvânt = 1 byte

1 block = 4 words

Cach data dimension = 1 KBlock

Program:

Address	Code
8194	230
4100	2715
12292	170



Address word

Tag	Index	W0
<12>	<10>	<2>

	Tag	Index	W0
8196	000000000010	0000000000	10
4100	000000000001	0000000001	00
12292	000000000011	0000000001	00

Curs 9

3. Memory hierarchy

3.1. Introduction

3.2. Mapping

3.2.1. Direct mapping

Example: Intrinsity FastMATH
- MIPS architecture
- 32 bit words
- split (16 KiB)
  - 1 block = 16 words
  - byte addresses

Address word

byte offset length

1 word = 32 bits = 4 bytes = 2^2 B

1 block = 16 words = 2^4 words

Size (M_2) = $= 2^4 \cdot 2^2 B = 2^6 B$

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

$= 4 GB$

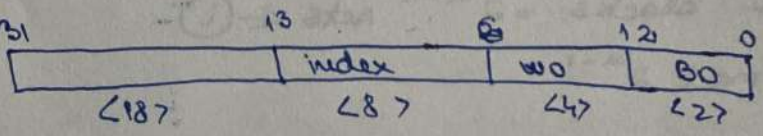
$= \frac{2^{32} B}{2^6 B} = 2^{26}$ blocks

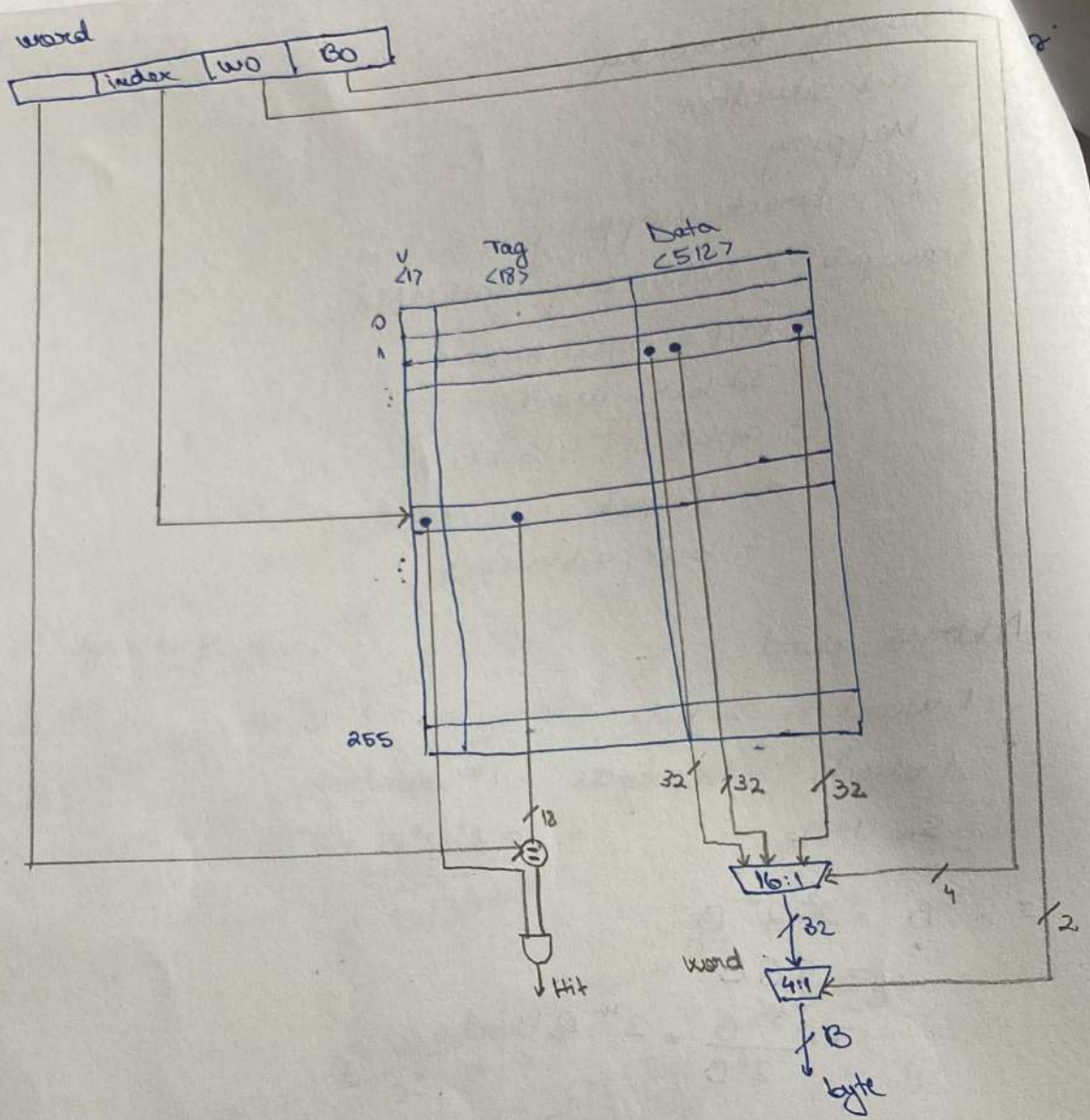
Size (M_1) = 16 KiB = 2^{14} B

$\frac{2^{14}}{2^6} = 2^8$ index blocks length

Tag length = M − N = 26 − 8 = 18

Address word





3.2.2. Set associative mapping

2^s - blocks at the same cache index

$M_2 \rightarrow 2^n$ blocks $\leftarrow (j)$

$M_1 \rightarrow 2^m$ blocks $= 2^{m-s}$ sets $\leftarrow (i)$

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

Exemplu:

$m = 5$

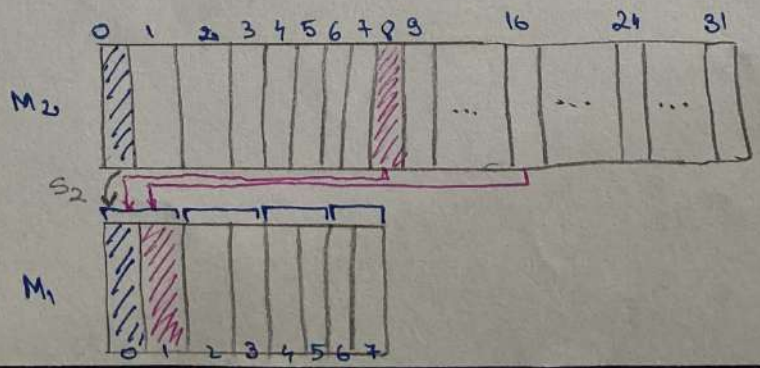
$m = 3$

$s = 1$

$0 \bmod 2^2 = 0$

$8 \bmod 4 = 0$

$16 \bmod 4 = 0$



00000
  └ index

2. 01000
   └ tag └ index

3. 10000
   └ tag └ index

4. 01011
   └ tag └ index

Example:

Address	Code
8194	230
4100	2715
12292	170

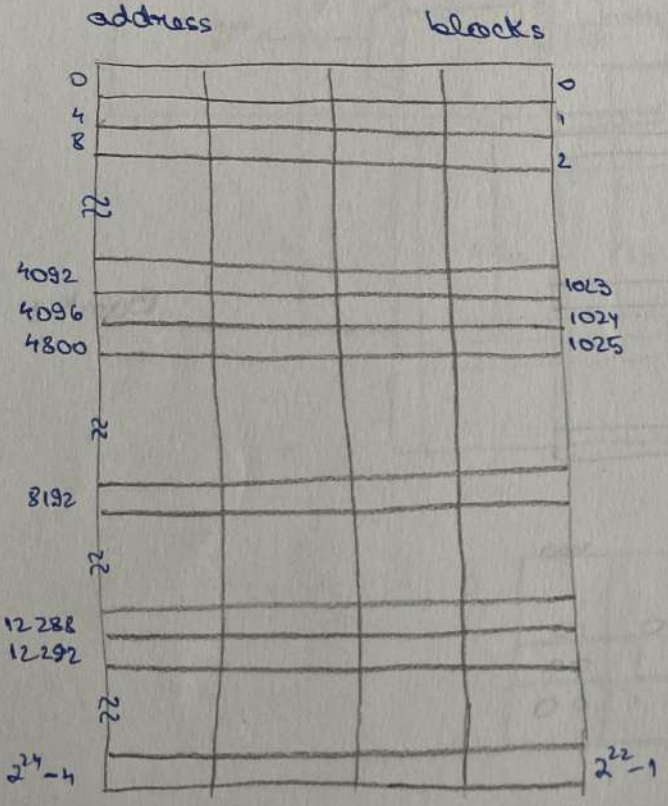
$MM = 2^{24}$ words

1 word = 1 B

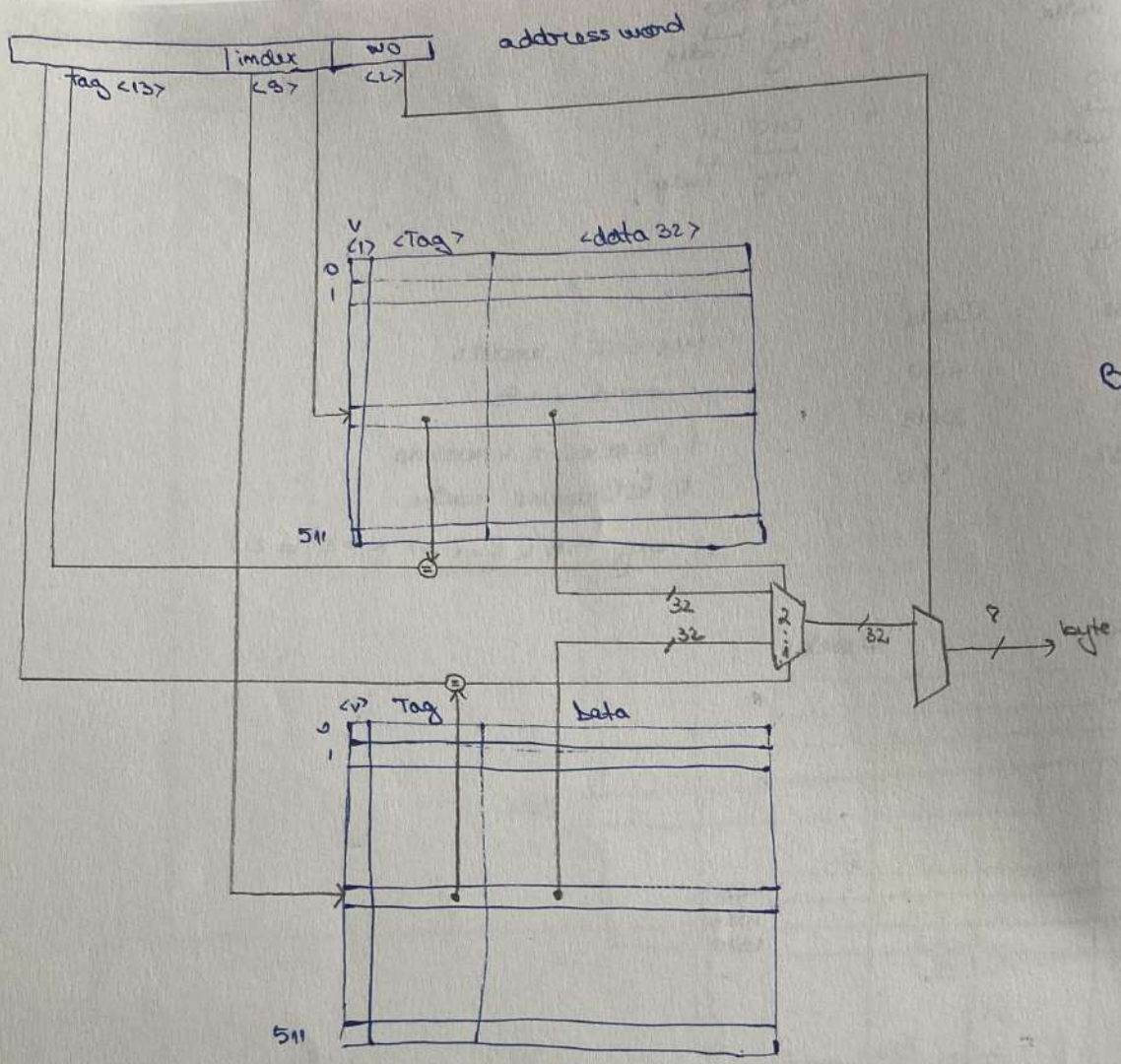
1 blocks = 4 words

1 KiBlocks cache

2-way set ($s=1 \Rightarrow k=2^1=2$)



Curs S10



Bank 0

Bank 1

Tag	Index	WO
8192 00 00 00 00 00 00 100	00 00 00 00 00 00	10
4100 0 0 00 00 00 00 00 0 1 0	0 0 0 0 0 0 0 0 0 1	00
0 0 0 0 0 0 0 0 0 0 0 1 1 0	0 0 0 0 0 0 0 0 1	00

Example:

1 MIPS machine VAX 11780

38 bit word address

cache 2-way SA

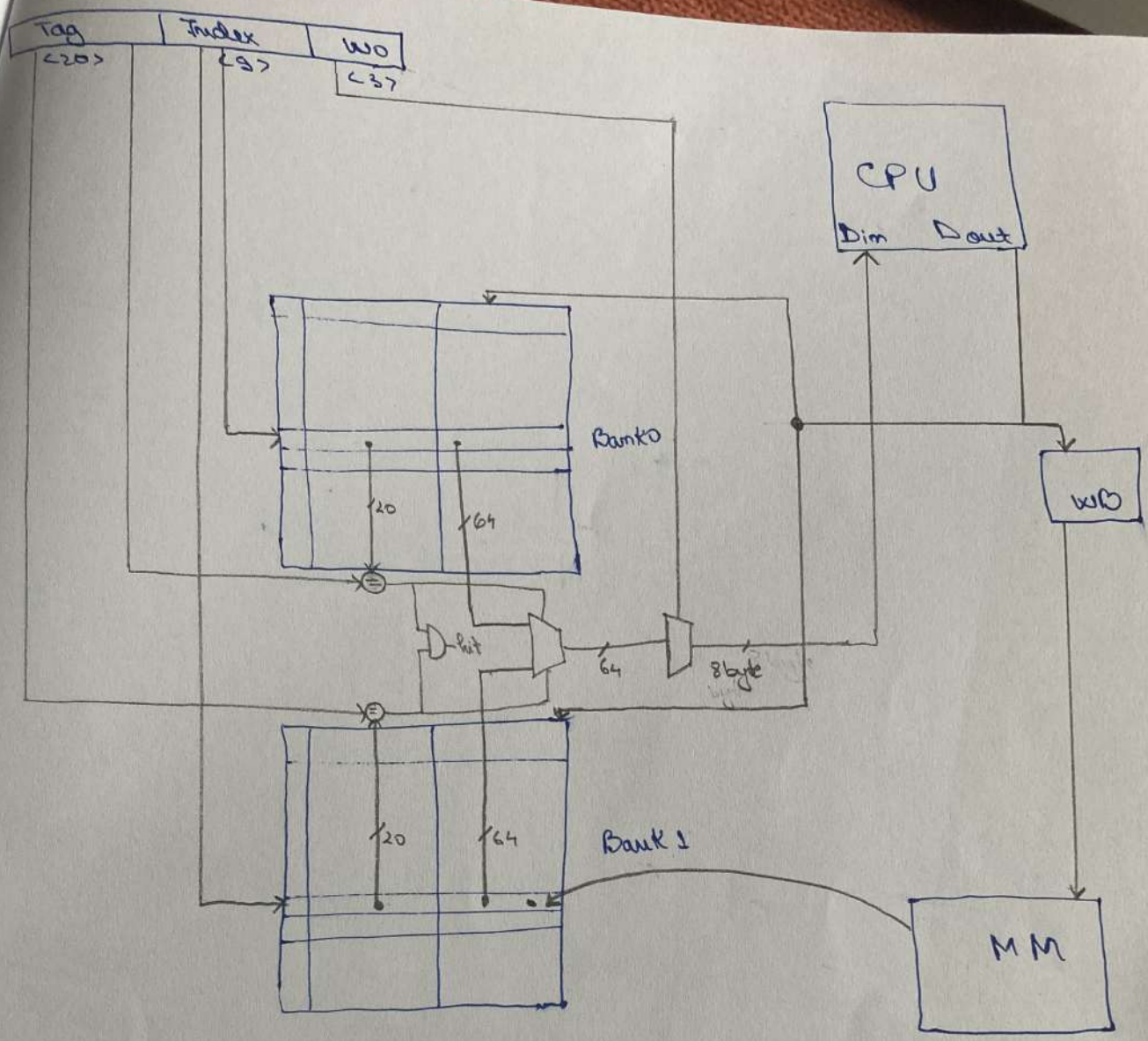
1 word = 1 byte = 2^0 B

1 block = 8 words = 2^3 words

cache size = 1 KiB

MM size = 2^{32} B = 4 GiB

cache size (data) = 2^{10} B locks = $\frac{2^{10}}{2^3} = 2^9$ sets


Tag
<20>
Index
<9>
WO
<3>
CPU
Din
Dout
Bank0
20
64
WO
bit
64
8byte
20
64
Bank 1
MM

Curs SIO 2.05.2023

Exemple 1

Size (M2) = 4 GiB – memoria principala
Byte addresses
1 word = 4B = 2^{2} B → BO (byte offset)
1 block = 8 words = 2^{3} words → WO (word offset)
4 way SA cache
cache data size (M_1) = 16 KiB

a) Address word format = ?
b) Total cache size = ?
c) Cache logic diagram

Size M_2 = 4 GiB = $2^2 \cdot 2^{30}$ B = 2^{32} B → Address word

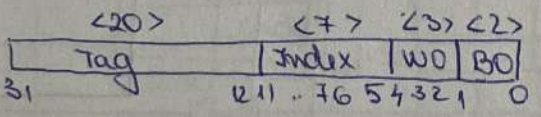
Data size M_1 = 16 KiB = $2^4 \cdot 2^{10}$ B = 2^{14} B

1 block = 2^3 words = $2^3 \cdot 2^2$ B = 2^5 ~~bytes/block~~ B = $2^5 \cdot 2^3$ b = 2^8 b

$$M_1 = 2^{14} \text{ B} = \frac{2^{14} \text{ B/cache}}{2^5 \text{ B/block}} = 2^9 \text{ block/cache}$$

$$\text{Bank size} = \frac{2^9 \text{ block/cache}}{2^2 \text{ n.a}} = 2^7 \text{ block/bank} \rightarrow \text{Index}$$

a)



Address word

b) $\underbrace{2^2}_{\text{n.a}} \cdot (1\text{b} + 7\text{b}$

$\underbrace{2^2}_{\text{n.a}} \cdot \underbrace{2^7}_{\text{cache lines/bank}} \cdot (\underbrace{1\text{b}}_{\text{valid}} + \underbrace{20\text{b}}_{\text{Tag}}$ (in plus) $+ \underbrace{2^5 \text{ B}}_{\text{data}}) = 30 \cdot 2^9 + 16 \text{ KiB}$

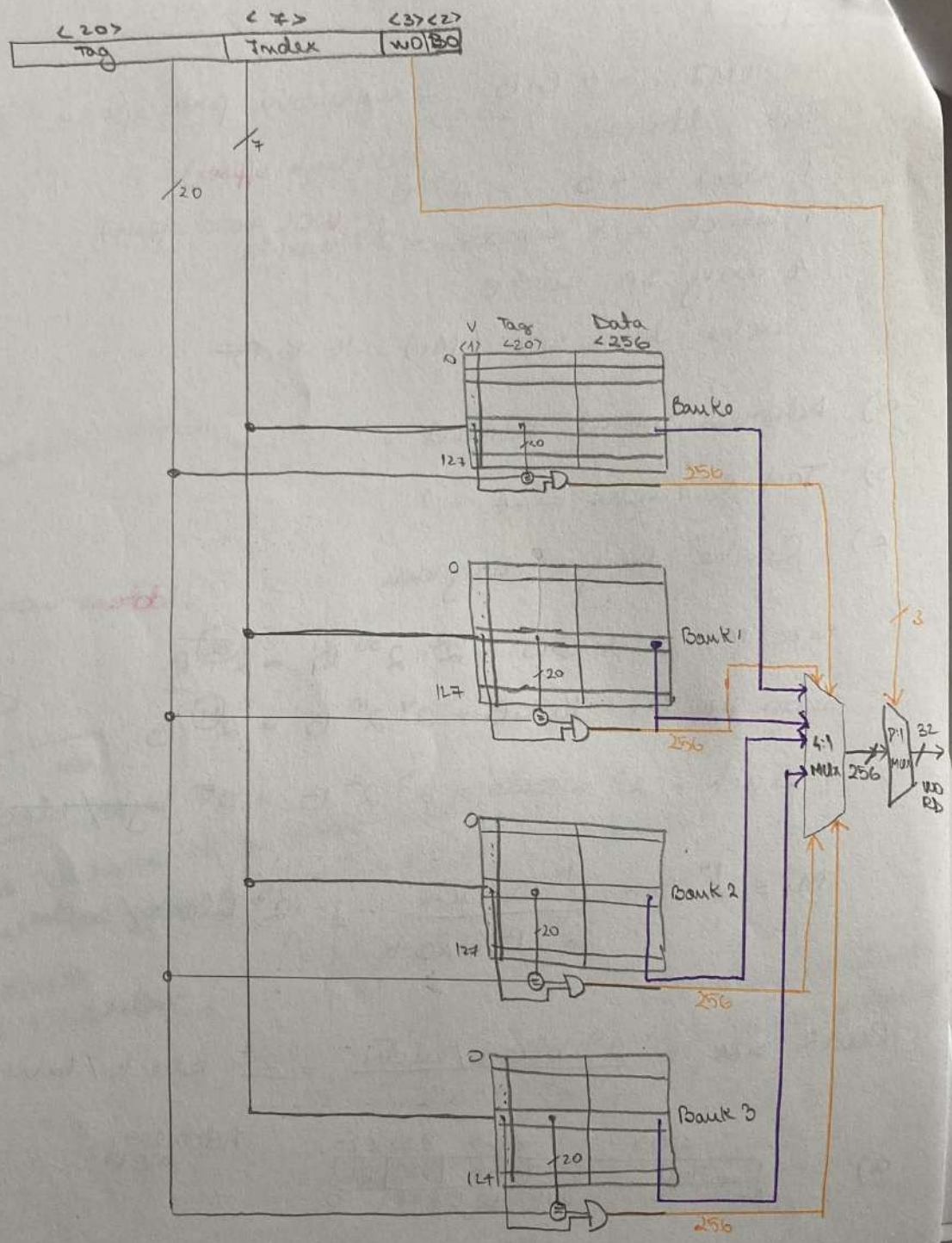
$\cong 18$ KiB

Write policies

Cach coherency (cache coherency)

sta

c)



09.05.2023

2.3 Full associative mapping

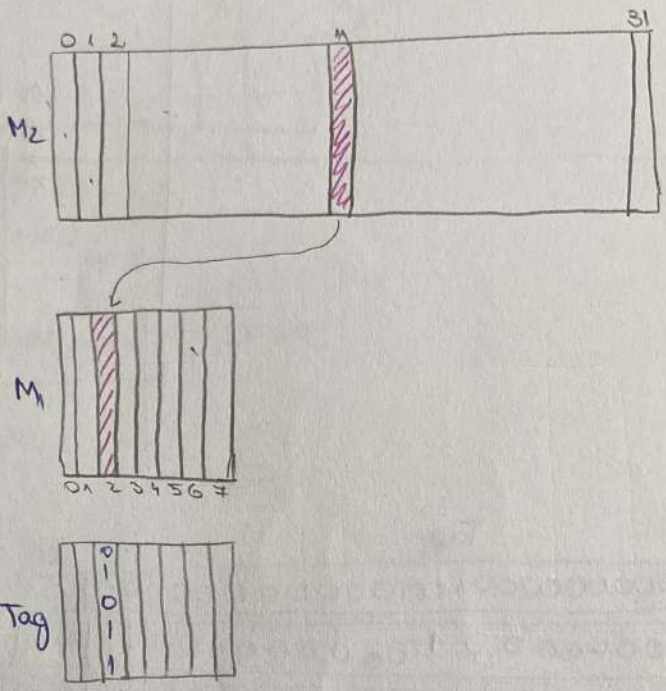
$K = 2^n$ → if $s=0$ ⇒ ΔM
→ if $s=M$ ⇒ ΔA

Exemple 2:

Size $(M_2) = 2^M$ blocks
Size $(M_1) = 2^m$ blocks

$n=5$
$m=3$

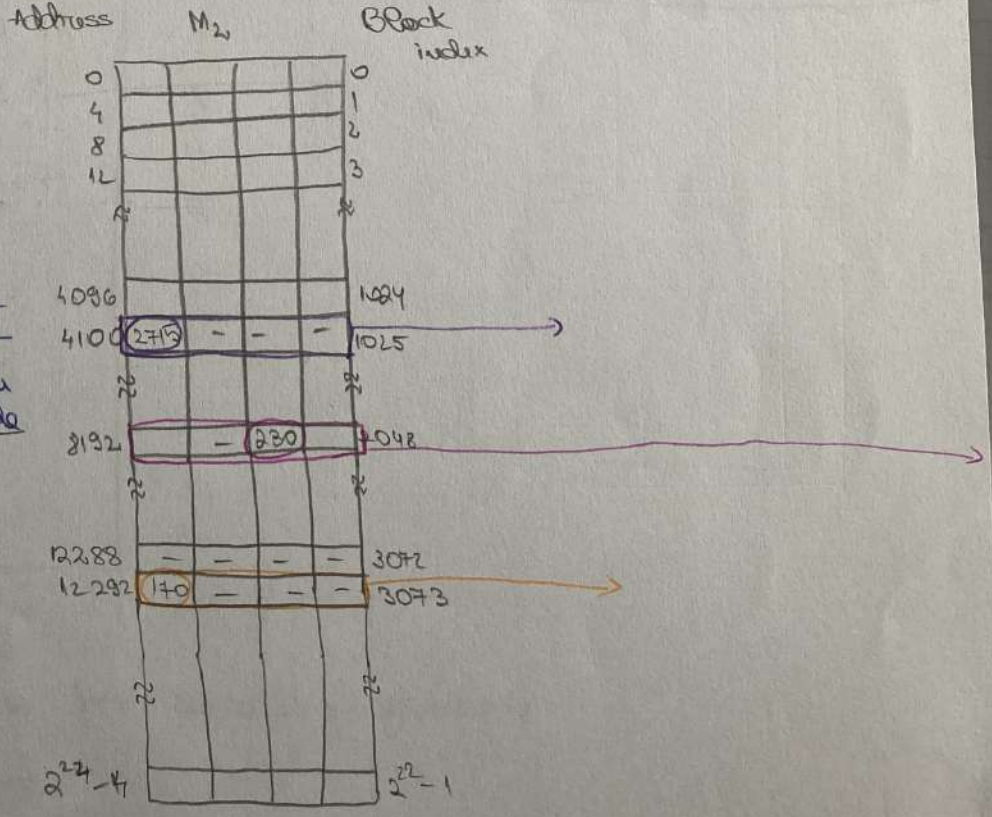
$11 = \underbrace{01011}_{\text{Tag}}$



Exemple 3:

2^{24} B → M_2
1 word → 1 B
1 block = 4 words
M_1 Size = 1 KiBlock

Address	Instruction code
8194	230
4100	2715
12292	170



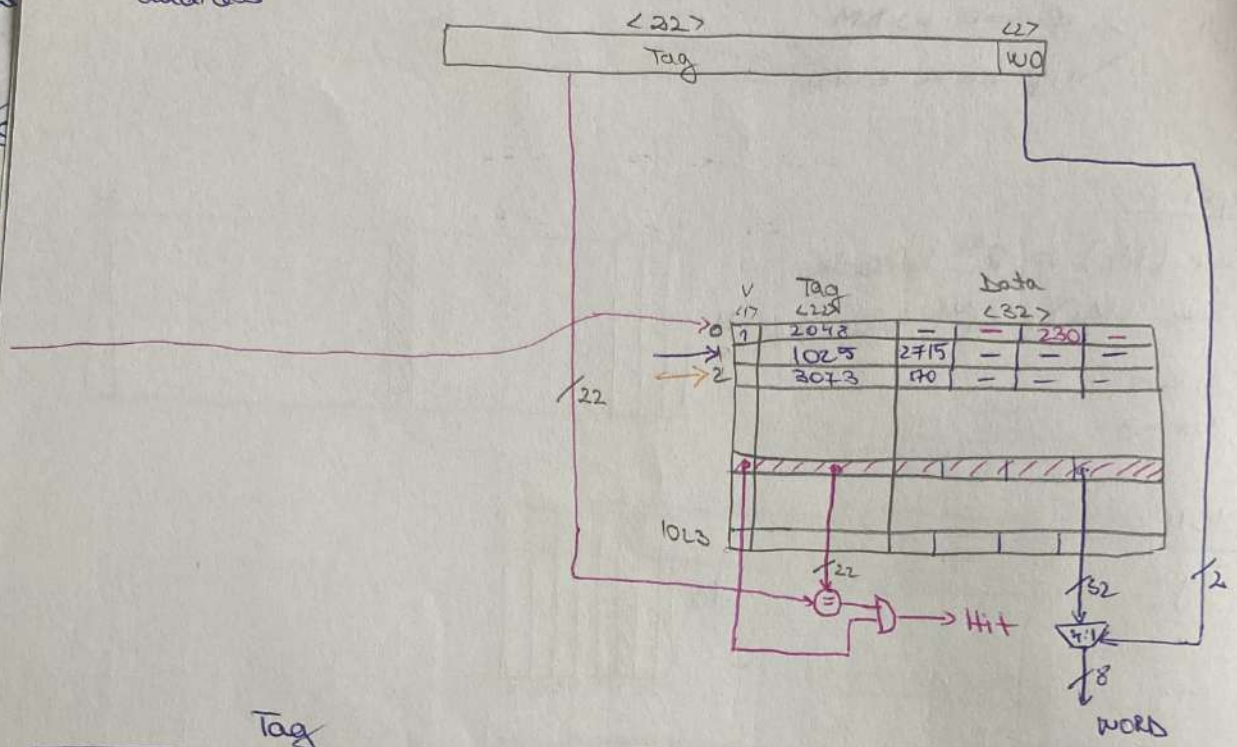
write policies

Cache coherency (cache coherency)

stat

writes

Word address



Tag

WO

00000000001000000000000000 10 8194

00000000002100000000001 00 4100

00000000011000000000000 00

09.05.2023

3. Replacement methods

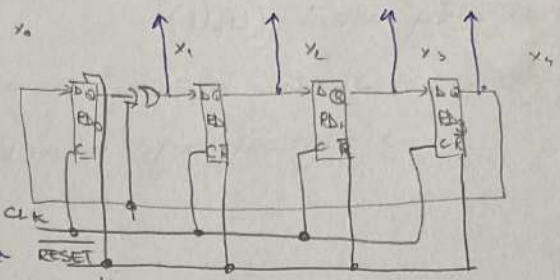
What happens if the cache is full?

SA & FA

3 Politics

1. Random

Pseudo-random



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

Period $= 2^{\text{Deg}(G(x))} - 1$

Linear feedback shift Register : LFSR

RD_0	RD_1	RD_2	RD_3	Dec
1	0	0	0	1
0	1	0	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
				10
				7
				14
				15
				13
				9
				1

Not compliant with the locality principle

May be used for low s

2. FIFO · Not compliant with the locality principle

3. Least recently used (LRU)

Age registers → each block in a set has an age
→ highest age block is replaced

Example 4

4 - ways SA cache

	Code	Banks				Age			
		0	1	2	3	0*	1*	2*	3*
M	14	0	0	0	0	0	0	0	0
M	7	14	0	0	0	0	0	0	0
M	6	14	7	0	0	1	0	0	0
M	12	(14)	7	6	0	2	1	0	0
M	10	(14)	7	6	12	3	2	1	0
H	6	10	7	6	12	0	3	2	1
H	7	10	7	(6)	12	1	3	0	2
M	14	10	7	6	(12)	2	0	1	(3)
H	7	10	7	6	14	3	1	2	0
M	3	10	7	6	14	3	0	2	1
	14	3	7	6	14	0	1	3	2
		3	7	6	(14)	1	2	3	0

09.05.2023

Curs 11

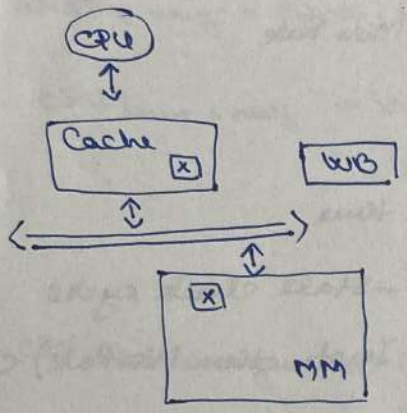
3.4. Write policies

Code coherency (cache coherency)

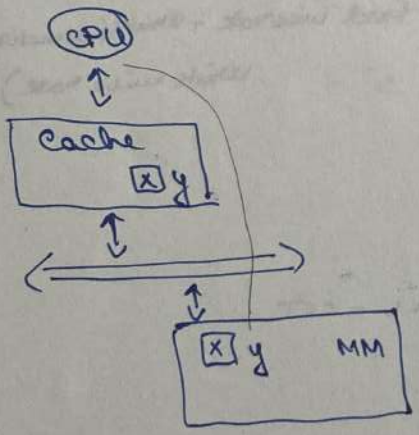
Statistically $\cong$ 10% are writes

a) write through

- write buffer (dimension optimization)



b) write back - update upon replacement



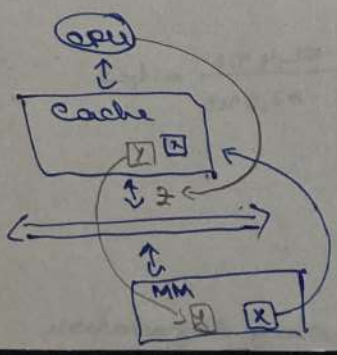
What happens in case of a write miss?

[What happens for a read miss? - always allocate]

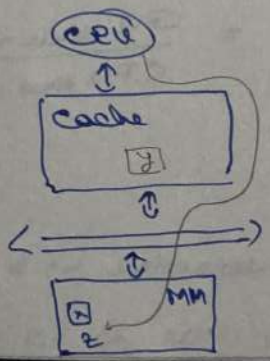
a) write allocate

b) write no allocate

a)



b)



b + α - WB + W allocate

$\alpha + \beta$ WT + W no allocate

Curs 12

3.5. Cache memory performance

a) Hit rate ↔ Miss rate ; MR + HR = 1

The 3 main causes of misses

- compulsory (cold start)
  - bigger block size
- conflict
  - higher SA, FA ; 32 way / 64 way ; SA ≅ FA
- capacity

b) Average memory access time (AMAT)

AMAT = t_{ac} · Hit Rate + (t_{ac} + Miss penalty) Miss Rate

= t_{ac} + Miss Rate · Miss Penalty

↙ cache access time (1cc - 2cc)

c) CPU time = Clock cycles · Clock cycle time

Clock cycles = Execution clock cycles + stall clock cycles

= IC · CPI + ic · Memory access Instruction · Miss Rate) · CCT

CPU time = IC · CPI ideal + Memory access per instruction · Miss Rate · Miss Penalty) · CCT

Stall clock cycles = (~~Reads/prog~~ Reads per instruction · Read miss rate + Writes/instruction · Write miss rate) · Miss penalty

Exemple 1:

CPI ideal : 8,5 cc Miss penalty : 6 cc

Memory access per instruction : 3 t_{ac} = 5 ns = CCT

a) AMAT = ?

Miss penalty = 6 cc = 6 · 5 ns = 30 ns

AMAT = 5 ns + 0,11 · 30 ns = 8,3 ns

b) $CPU_{time\ ideal}$ = IC · 8,5 · 5 ns = IC · 42,5 ns

$CPU_{time\ real}$ = IC · $\underbrace{(8,5 + 3 \cdot 0,11 \cdot 6)}_{cc}$ · 5 ns = IC · 52,4 ns

$$\frac{\text{Performance ideal}}{\text{Performance real}} = \frac{\text{CPU time real}}{\text{CPU time ideal}} = \frac{IC \cdot 52,4\ ns}{IC \cdot 42,5\ ns} = 1,23$$

Exemple 2:

Cache : 64 KiB

CCT = 20 ns

Memory access / interaction = 1.3

CPI ideal = 1,5 cc

SA ⇒ additional MUX ⇒ 8,5 % clock frequency degradation

Übung 12

(1)

Which is better?

1 way SA or 2 way SA

	1 way SA	2 way SA
MR	3,9 %	3 %
MR	200 ns	200 ns

a) $AMAT_{1\,way} = 20\,ns + 0{,}039 \cdot 200\,ns = 27{,}8\,ns$

$AMAT_{1\,way} = 20\,ns \cdot 1{,}085 + 0{,}03 \cdot 200 = 27{,}7\,ns$ ✓

b) $CPU_{time\ 1\,way} = ic \cdot (1{,}5 + 1{,}3 \cdot 0{,}039 \cdot \text{Miss Penalty}^{s}) \cdot CCT$

$= ic \cdot (1{,}5 \cdot 20\,ns + 1{,}3 \cdot 0{,}039 \cdot \underbrace{\text{Miss Penalty}^{+}}_{20\,ns})$

$= ic\,(30\,ns + 10{,}14\,ns) = ic \cdot 40{,}14\,ns$ ✓

$CPU\,time\ 2\,way = ic\,(1{,}5 + 1{,}3 \cdot 0{,}03 \cdot \text{Miss Penalty}^{c}) \cdot CCT$

$= ic\,[(1{,}5 \cdot 20 \cdot 10 \cdot 85)\,ns + 1{,}3 \cdot 0{,}03 \cdot 200\,ns]$

$= (32{,}55\,ns + 7{,}8)\,ic$

$= ic\ 40{,}35\,ns$

Curs 12 16.05.2023

Write back
- Update upon replacement
  - 1 dirty bit

Miss - allocation

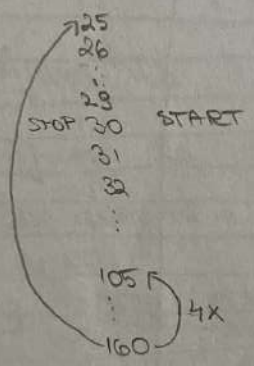
Exemple de probleme (EXAMEN)

1.

Cache (instruction) 4-way SA (8 sets) !!!
- 1 block = 4 words
- 2^{16} words → M_2
- Addressable unit → word
- Cache : initially empty
- tac = 30 ns = Clock cycle time
- Miss Penalty = 480 ns (c.c)
- LRU replacement

a) Memory address format =)
Total cache capacity , 1 word = 32 B

b) Hit ratio and AMAT,
- execută un program dat ca adresă a secvenței:



Tag

29 = 000|111|01

Tag = N_M mod nr. de seturi
=

Curs 13

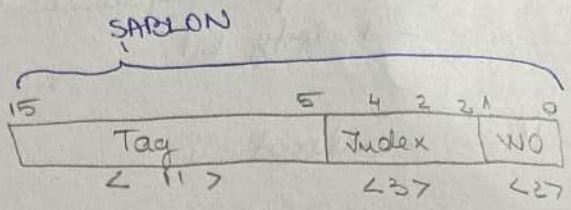
Rezolvare:

a) nu are byte offset (BO)

dar 1 block = 4 words = 2^{2} → WO

8 seturi = 2^{3} → Index

2^{16} → Address words → M_2



S.A : $4 = 2^2$

Total cache = S.A · nr. seturi / bank · [(1 + 11) + 2^2 B]

(1 + 11): 1 = valid, 11 = Tag

Blocks/cache

1 word = 32 biți = 4B = 2^2 B

↓ instrucțiune

Total cache = 4 · 8 · (12 + 4)

= ~~4 · 8 · 16~~ 4 · 8 · 16

= $2^2 \cdot 2^3 \cdot 2^4$

= ~~192~~ B

b) M_2

Address	Block number
0-3	0
4-7	1
8-11	2
12-15	3
16-19	4
20-23	5
24-27	6
→ 28-31	(7)
→ 32-35	8
36-39	9
40-43	10
44-47	11
48-51	12
52-55	13
56-59	16
60-63	15
64-67	16
68-71	17
72-75	18
76-79	19
80-83	20
84-87	21
88-91	22
92-95	23
96-99	24
100-103	25
104-107	26
108-111	27
112-115	28
116-119	29
120-123	30
124-127	31
128-131	32
132-135	33
136-139	34
140-143	35
144-147	36
148-151	37
152-155	38
156-159	39

a) WT

b) WB

23.05.2021

$10^8 \times 0.1$ ...

Tag	
15	32 – 35
1	36 – 39
1	40 – 43
1	44 – 47
1	48 – 51
1	52 – 55
10	56 – 59 24-27
Ø	28 – 31

4

B0

156 – 159

V	Tag	
0	1	2	64 – 67
1	1	2	68 – 71
2	1	2	72 – 75
3	1	2	76 – 79
4	1	2	80 – 83
5	1	2	84 – 87
6	0	2	88 – 91
7	1	1	60 – 63

B1

160 – 163	40

V	Tag	
0	1	3	96 – 99
1	1	3	100 – 103
2	1	3	104 – 107
3	1	3	108 – 111
4	1	3	112 – 115
5	1	3	116 – 119
6	1	3	120 – 123
7	1	2	92 – 95

B2

V	Tag	
0	1	4	128 – 131
1	1	4	132 – 135
2	1	4	136 – 139
3	1	4	140 – 143
4	1	4	144 – 147
5	1	4	148 – 151
6	1	4	152 – 155
7	1	3	124 – 127

B3

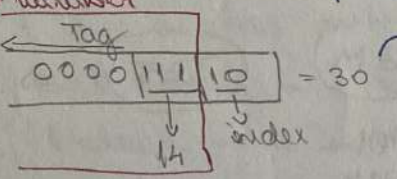
$\binom{1M}{1H}\binom{1M}{3H} \times 31$

Prima instr. la adresa 30 => 1M (1 miss)

Alocă blocul, blocul se află în tabel 28 – 31

Tag: câtul împărțirii lui Block number la

Block number



→ ȘABLONU DE LA ÎNCEPUT

Nu cred că o doua ? Ca nu =:CC

Urm. block: 32 – 35 → 8 modulo 8

8 => Tag = 1

9 modulo 8 = 1

Ajung la 23 pe ↓ 23 modulo 8 = 7 => mă duc la indexu 7

Ajung la 156 în blocul 39, 39 modulo 8 = 7

Dar cache-ul e plin, la indexul 7 dar LRU deci replasez indexu 7 din B0

156 (M) 157, 158, 159 (H) => $\binom{1M}{3H} \times 1$

La 160 (1M) în blocul 40, 40 modulo 8 = 5 deci repl. indexu 5 din B0

23.05.2023

Au executat tot de la 30 la 160
=> sare înapoi la 105 care e în cash
105 - 160 cash-ul are instrucțiunile
=> 56 * 4 = (224 H, 0 M)
0 face de 4 ori (bucla)
atâtea valori sunt deja în tabel de la 160 la 105

Deci au executat și bucla mică
sare înapoi la 25
25 lipsește => (1M)
replasez pe ăla mai vechi
~~24,25,26~~ - sunt => (2H)
~~25~~,26,27
28 lipsește (1M)
29,30 -> (2H)

=> (1M, 1H)(1M, 3H) x 32 (1M)(224 H)(1M, 2H)(1M, 2H)

deci avem 131 + 224 + 6 = 230 + 131 = 361 accese

(36 M) (325 H) => 361

101 + 224 = 325

Hit ratio = $\frac{325}{361} = 0,9002 \simeq 90\%$

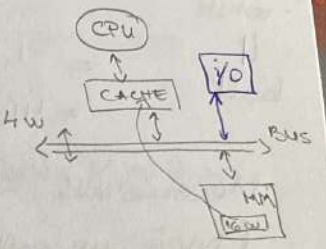
Curs 13

Continuare: calcul AMAT

AMAT = 30 ns + 0,9 · 480 ns

Exemplul 2.

Sist. de calcul load store cu un cache următoarele valori empirice atunci când rulăm ceva:



10% Data Miss Rate

8% Instruction Miss Rate

20% Load Store instructions

15% of the blocks are dirty at any time

CPI = 2 cc dacă cash-ul este perfect

CPI ideal = 2 cc

IC = 10 000

Clock frequency = 4,5 GHz

Miss Penalty este determinată de faptul că un acces la bus revendică 10 cc

1 Bus access = 10 cc

Bus width = 4 words

1 block = 16 words

Politică de tip Write Back + Write Allocate (entire block)

a) AMAT = ?

b) CPU time = ?

$$AMAT = t_{ac} + MissRate \cdot MissPenalty^t$$

$$\text{Clock cycle time} = \frac{1}{\underbrace{4,5 \cdot 10^9}_{\text{Clock freq.}}} = \frac{1}{4,5 \cdot 10^9} \text{ s} = \frac{1}{4,5} \cdot 10^{-9} = 0,22 \text{ ns}$$

$$MissPenalty^t = \text{Clock cycle time} \cdot MissPenalty^c$$

$MissPenalty^c$

16 words allocation $\Rightarrow \frac{16}{4} = 4$ Bus accesses

(4 ← Bus width)

10 cc $\Rightarrow$ 4 · 10 cc pt. o alocare

Curs 13

Read Miss Penalty = alocare + update

40 cc < allocation penalty
update penalty

Read Miss Penalty = ~~Allocation Penalty~~ + ~~procentul de dirty~~ · Update Penalty
0,15

write
||
back = 1,15 · 40
= 46 cc

Mecanismul este același la Writing Miss Penalty
(Dacă nu aveam Write Back nu erau egale)

Write Miss Penalty = 1,15 · 40
= 46 cc

=> ~~Read~~ Miss Penalty^c = 46 ~~46~~ cc
~~46 · 22 cc~~

=> Miss Penalty^t = 46 · 0,22
= 10,21 ns

AMAT = 0,22 ns + ...

10 % Data Miss Rate - numai din când în când (la load store)
8 % Instruction Miss Rate - de fiecare dată

Miss Rate = 0,08 + 0,2 · 0,1 = 0,1
↓ L/S instruction ↘ Instr. Miss Rate

AMAT = 0,22 ns + ~~0~~ · 10,12 ns = ~~12,34~~ ns = 0,22 + 0,0833 · 10,12
= 1,06

Miss Rate = Misses Per Instruction / Mem. access per instruction = 0,1 / 1 + 0,2
↓ pt. citire instrucțiune ↘ pt acces la memorie pt. date

ures 13

a) WT
b) WB

23.05.2023

CPU time = IC (CPI ideal + Memory accessed per instn. × Miss Rate × Miss Penalty) · Clock cycles time

= 10000 (2 + Mem. acc. per instn. · Miss Rate · 46) · 0,22 ns

Mem. acc per instn · Miss rate ⇒ nr toate accesele la mem.

Nr Miss
⇒ Mem. acc. per instn · Miss Rate = Misses per instn.

=

= 1 × Instruct. Miss Rate + L/S Instruction · Data Miss Rate

= 1 · IMR + 0,2 · DMR = 0,1

CPU time = 10 000 (2 + 4,6) · 0,22

= 6,6 · 0,22 · 10 000

= 66 · 22 · 10

= 14520 ns

= 14,52 µs

Exemplul 3

a) % BUS Used

Bus Bandwidth = 10^9 words/sec
Bus width = 2 words
Hit rate = 90 %
1 block = 4 words
In cazul unui Miss e alocat tot blocul
Block allocation
CPU reference frequency = 10^2 words /sec
30 % writes
35 % of blocks are dirty
write allocate for a write miss

a) ~~cache~~ cache-ul este write through

b) cache-ul ~~este~~ write back

10^8 words/sec · $\underbrace{0,1}_{\text{Miss Rate}}$ · ~~Read~~ Miss Penalty^b

Miss Penalty^b = 0,7 Read miss penalty^b + 0,3 Write miss penalty^b

Read miss penalty^b $= \underbrace{0,35 \cdot \frac{4 \text{ words}}{2 \text{ words}}}_{\text{update writes}} + \underbrace{1 \cdot \frac{4 \text{ words}}{2 \text{ words}}}_{\text{Reads}} = 1,35 \cdot 2 = 2,7$

= write miss penalty

23.05.2023

Curs 13

a) WT
b) WB

$10^8 \times 0,1 \times (\%\ Reads \times Read\ Miss\ Penalty + \%\ Writes \times Write\ Miss\ Penalty^b)$

$Read\ Miss\ Penalty^b_{WB} = Write\ Miss\ Penalty^b_{WB}$

$= 0,35 \times \underbrace{\frac{4\ words}{2\ words}}_{BUS\ Writes} + \underbrace{\frac{4\ words}{2\ words}}_{BUS\ Reads}$

$= 1,35 \times 2$ BUS Accesses

$= 2,7$

$10^8 \times 0,1 \times 2,7 = 2,7 \times 10^7$

$= 2,7\ \%$

a) $10^8 \times 0,1 \times \underbrace{(\%\ Reads \times Read\ Miss\ Penalty^b_{WT} + \%\ Writes \times Write\ Miss\ Penalty^b_{WT})}_{2,3}$

$+ 10^8 \cdot 0,3 \times \underbrace{0,9}_{Hit} \times Write\ Hit\ Penalty$

$Read\ Miss\ Penalty^b_{WT} = \underbrace{\frac{4\ words}{2\ words}}_{Allocate} = 2\ BUS\ reads$

$Write\ Miss\ Penalty^b_{WT} = \underbrace{\frac{4\ words}{2\ words}}_{\substack{Allocate \\ BUS\ Reads}} + 1\ BUS\ writes = 3\ BUS\ Acc.$

$Write\ Hit\ Penalty^b_{WT} = 1\ Bus\ write = 1\ BUS\ Access$

$0,7 \cdot 2 + 0,3 \times 3 = 2,3$

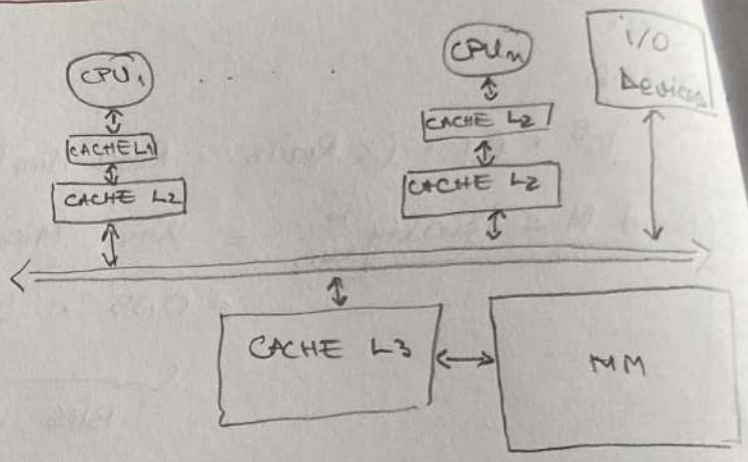
$\Rightarrow 10^7 \cdot 2,3 + 10^7 \cdot 2,7 = 5 \times 10^7$

$\%\ BUS\ Used_{WT} = \frac{5 \cdot 10^7}{10^9} = \frac{5}{100} = 5\%$

Curs 14

3.6 Reducing miss penalties and increasing cache performance

- victim caches
- multiple cache levels



Exemplul 1

CPI ideal = 1 cc

Clock rate = 4 GHz $\Rightarrow$ Clock cycle time = $\frac{1}{4 \cdot 10^9}$ s = 0,25 ns

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

Miss rate per instruction = 2% (at the primary cache)

L_2 cache = 5 ns access time

New miss rate = 0,5 %

Miss Penalty$_{MM}$ = 400 ns

CPU time 1-level = IC × (1 + 0,02 × 400) = IC × 9cc × 0,25 ns

Mem. access per instruction × Miss Rate = Misses per instruction

CPU time 2-level = IC × (1 + 0,02 × 20 + 0,05 × 400) × 0,25 = 3,4

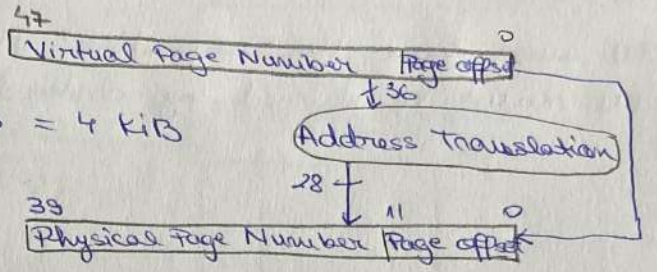
$$\frac{\text{Performance 2-level}}{\text{Performance 1-level}} = \frac{\text{CPU time 1-level}}{\text{CPU time 2-level}} = \frac{9cc}{3,4} = \underbrace{2,65}_{\text{Speed up-ul}}$$

adus de al doilea lvl.

ARM V8 Address word → 64 bit

MS 16 bits are unused

Virtual Address



page size $= 2^{12}$ B $= 4$ KiB

2^{48} B $= 256$ TiB Virtual Mem space size

2^{40} $= 1$ TiB Physical Mem Size

Design choices → huge miss penalty

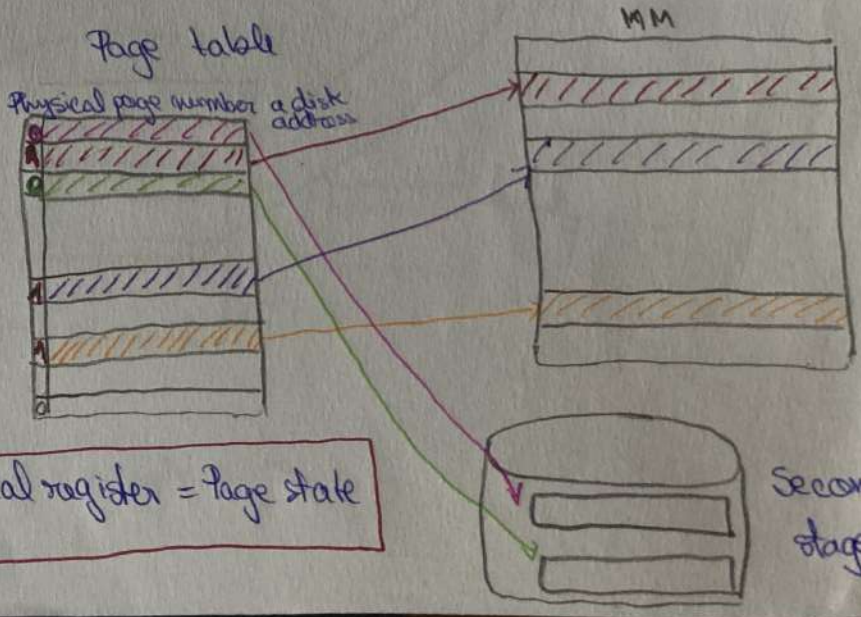
1. Large pages : 1 KiB - 64 KiB
   Embedded : 1 KiB - 4 KiB
   Desktop, Servers : 32 KiB - 64 KiB
2. Full associative mapping
   - implemented through pagetables

   WB is the only choice

Page table register

Virtual address

VPN	PO



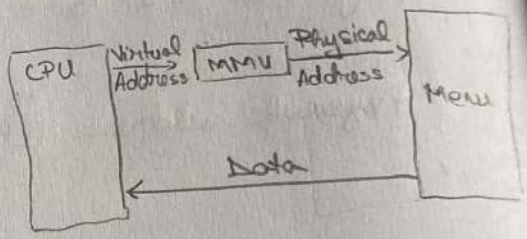
Page Table + PC + Internal register = Page state

Curs 14

4. Virtual memory

4.1. Motivation

1. CPU must have the "impretion" of ∞ size memory
2. CPU must be shared by distinct processes/virtual mach

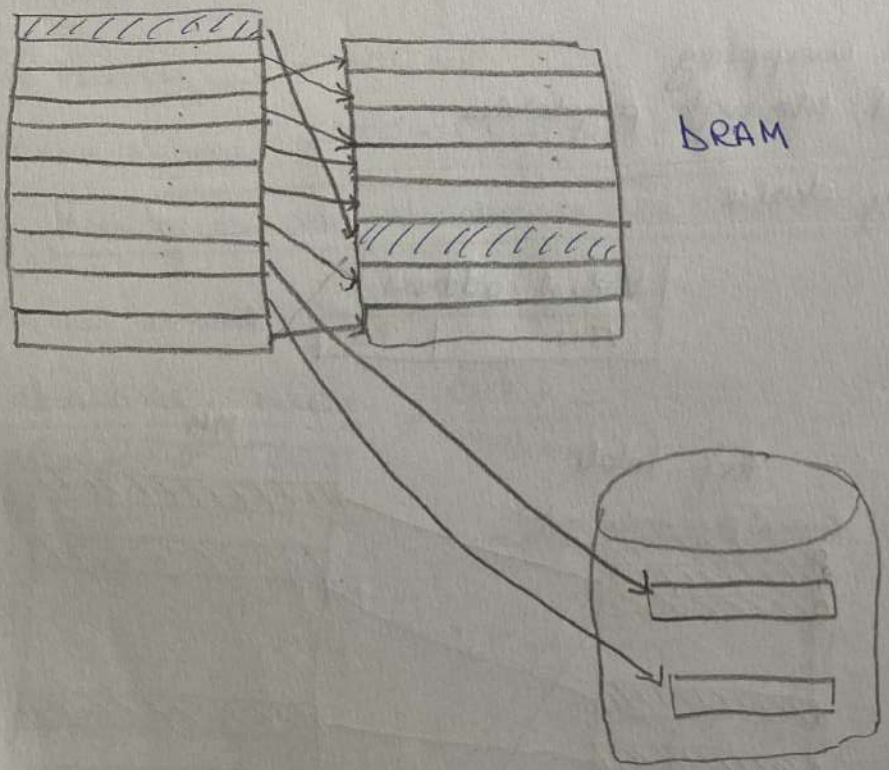


Virtual address —Translation→ Physical address

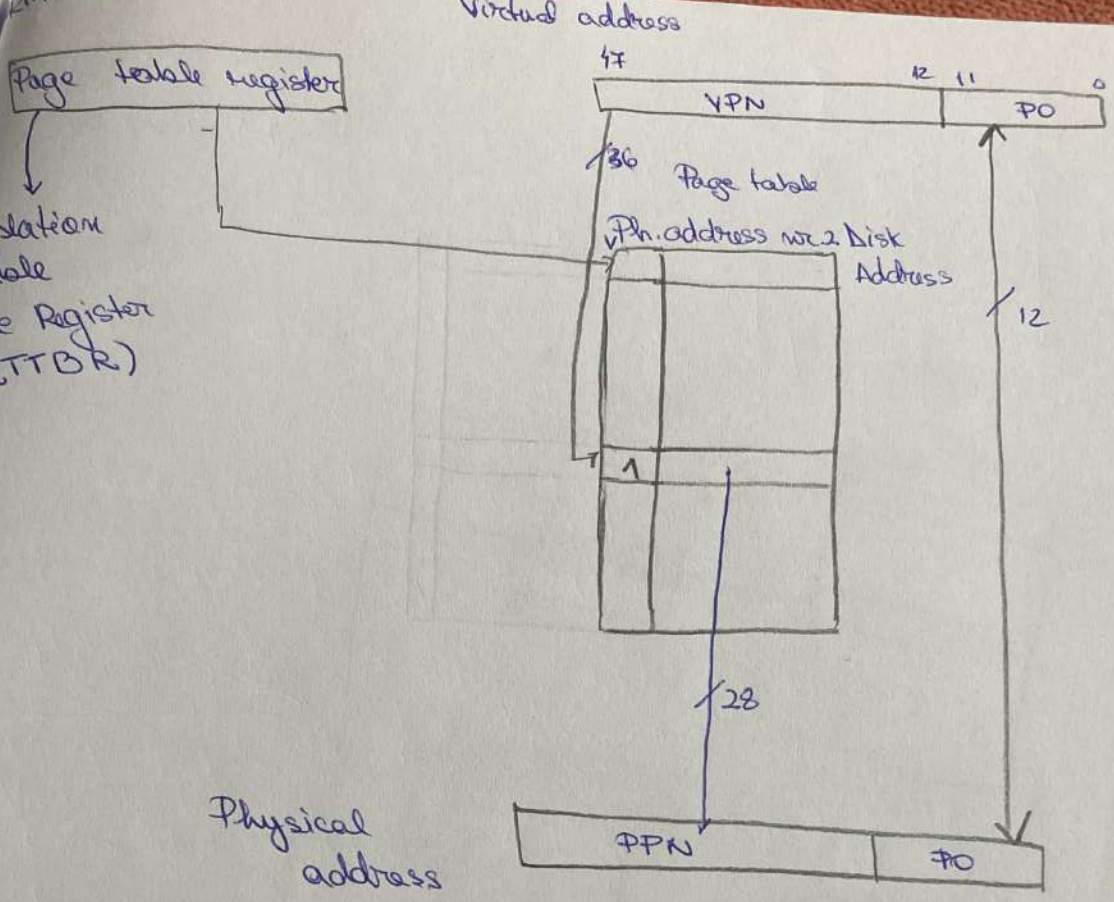
Locality principle

Blocks → Segments - dimensiune variabilă
→ Pages - dimensiune fixă

Virtual Memory Physical Memory



ARM V8

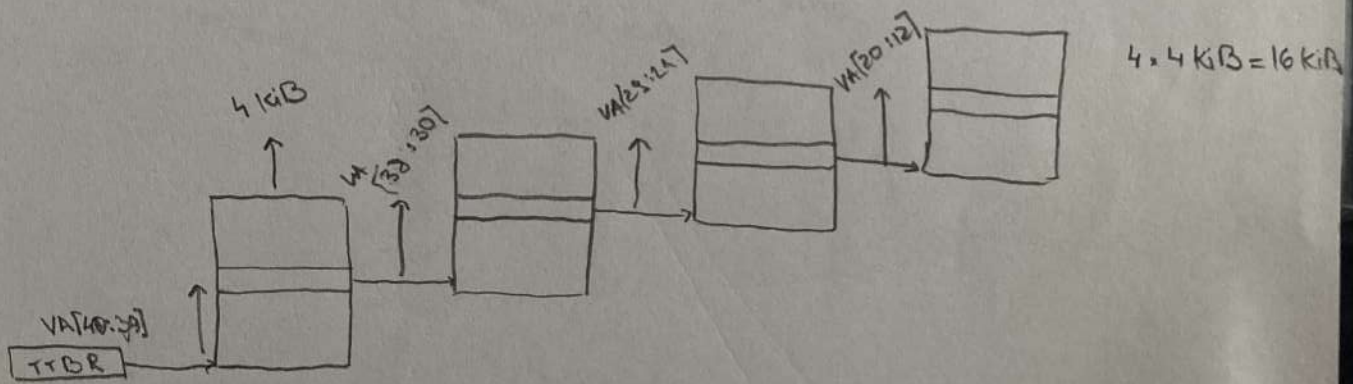


4.2 VM performance optimization

ARM V8 page table size $= 2^{36} \times 2^2 B = 2^{38} B = 0{,}25\ TiB$

⇒ Page table size reduction

- paged tables
- hierarchical page tables $2^2 \cdot 2^{10}$

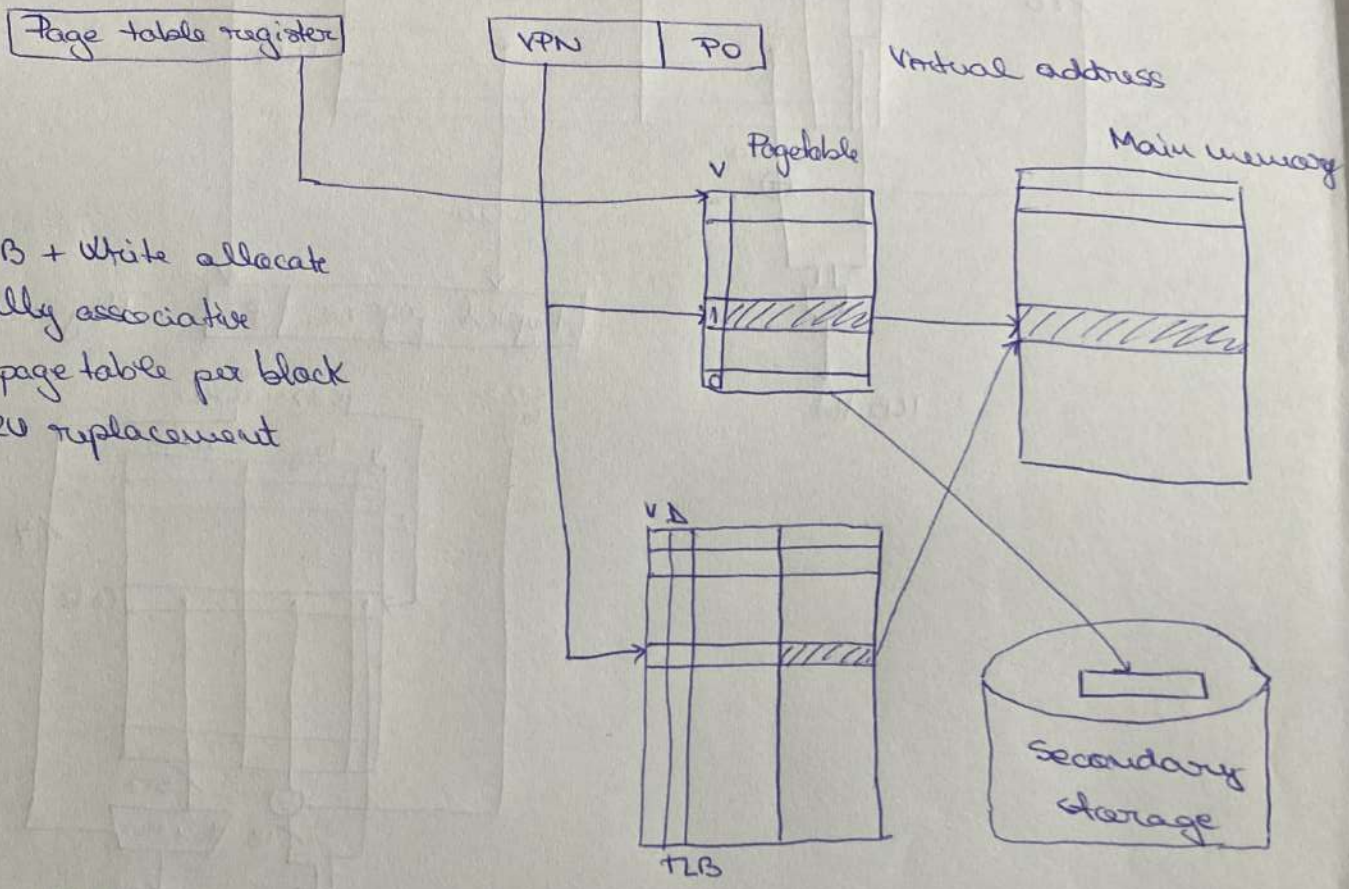


② Access time → Cache for addresses

Curs 14

4.2 VM optimization

Caching the page table → Translation Lookaside Buffer (TLB)



WB + Write allocate
fully associative
1 page table per block
LRU replacement

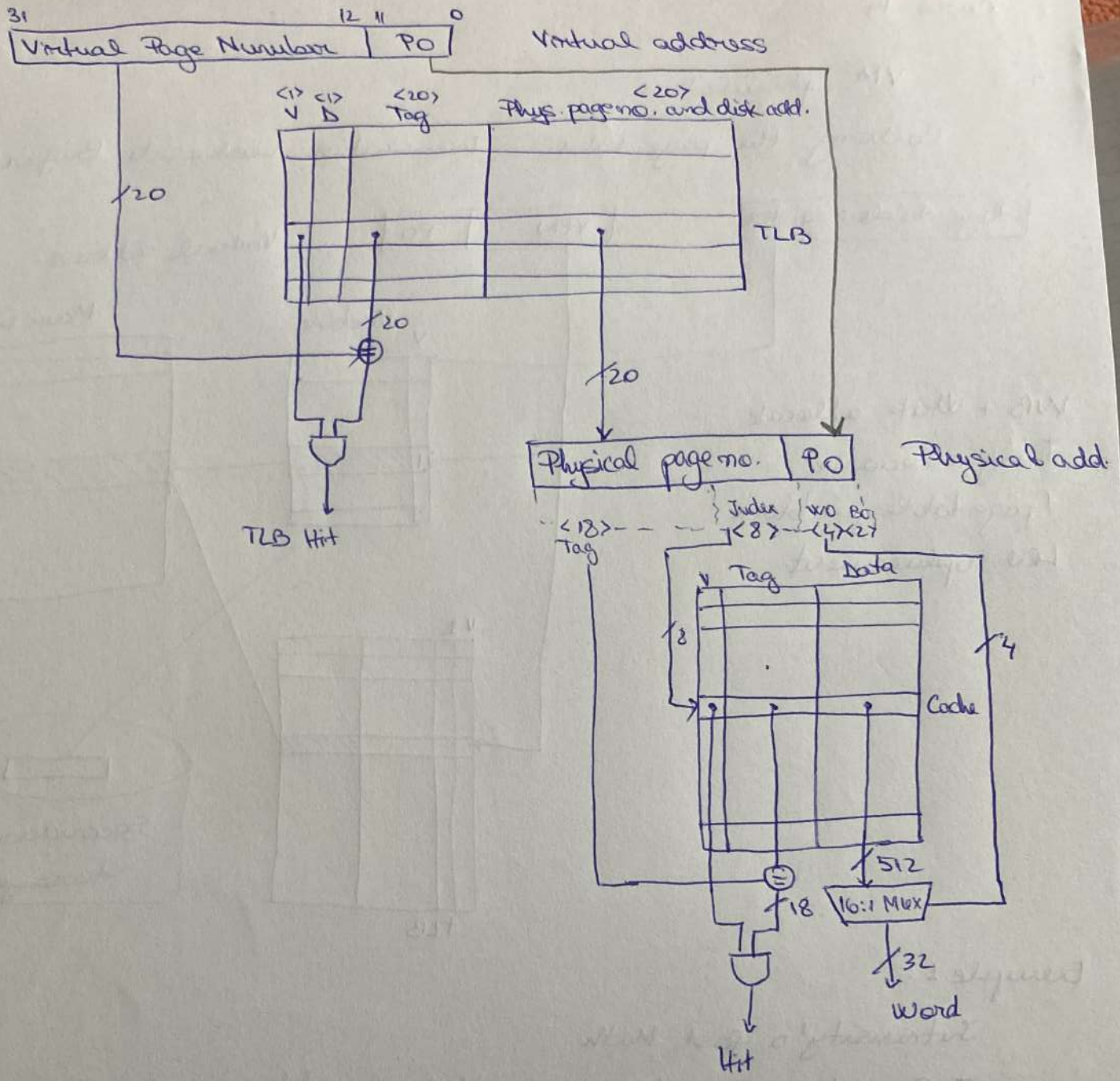
Example 1:

Intrinsity's fast Math

TLB 16-512 entries
block 1-2 page table entries
hit time = 1 cc
miss penalty = 10n-100n cc
Miss rate 0,1-1%

FA

Virtual address - 32 bits
Physical address - 32 bits
Page size = 4 KiB


31
12 11
0
Virtual Page Number
PO
Virtual address
<1> V
<1> D
<20> Tag
<20> Phys. page no. and disk add.
TLB
20
20
20
TLB Hit
Physical page no.
PO
Physical add.
Index
wo Bo
<13> Tag
<8>
<4><2>
V
Tag
Data
8
4
Cache
512
18
16:1 Mux
32
Word
Hit