INSTR. TESTATE

2008 0004
2009 0002
0109 5020
114B 0001
0109 5822
AC08 0008
0800 0008
014B 6022
8C0D 0008

0: addi $t0, $0, 4 // $t0 = 0+4 = 4

OP	RS	RD	IMM
8	0	8	4
001000	00000	01000	0000000000000100
2	0	0	8	0	0	0	4

4: addi $t1, $0, 2 // $t1 = 0+2 = 2

8	0	9	2
001000	00000	01001	0000000000000010
2	0	0	9	0	0	0	2

8: add $t2, $t0, $t1 // t2 = 4+2 = 6

OP	RS	RT	RD	Sh	func
0	8	9	10	0	32
000000	01000	01001	01010	00000	100000
0	1	0	9	5	0	2	0

C: XOR $t3, $t0, $t1 // t3 = 4^2 = 6

0	8	9	11	0	38
000000	01000	01001	01011	00000	100110

10: beq $t2, $t3, next // t2 = t3.

4	10	11	1
000100	01010	01011	0000000000000001
1	1	4	B	0	0	0	1

14: sub $t3, $t0, $t1 // not taken

01095822

18: next: sw $t0, 8($0) // scrie în word-ul de adr 2 val din t0=4

8+0

OP	RS	RT	IMM
43	0	8	8
101011	00000	01000	0000000000001000
A	C	0	8	0	0	0	8

1C: j target

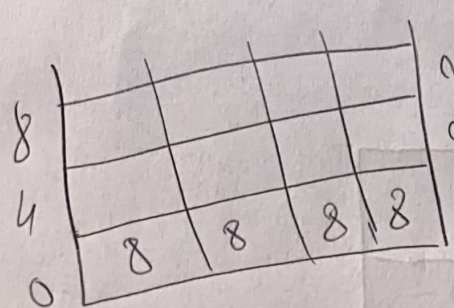
000010	00000000000000000000001001
0	8	0	0	0	0

20: sub $t4, $t2, $t3

014B6022

24: target: lw $t5, 8($0) // stocheaza în t5 val. din memorie de la adr 2 (adresa 8)

35	0	13	8
100011	00000	01101	0000000000001000
8	C	0	D	0	0	0	8



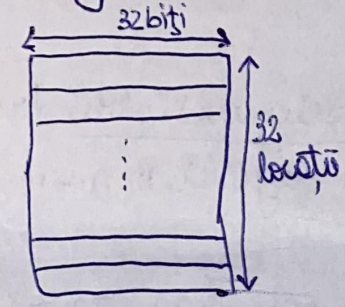
Instructions

R-type:
- Add (addition)
- Sub (substraction)
- Mult (multiplication)
- Div (division)
- And (AND)
- Or (OR)
- Nor (NOR)

I-type:
- Lw (load word)
- Sw (store word)
- Bne (branch not equal)
- Beq (branch equal)
- Addi (add immediate)
- Ori (or immediate)

J-type: J (jump)

Registers



- MIPS registers names are preceded by the $ sign.
- ex 1: the variables a, b, c are placed in $s0, $s1, $s2.

a = b + c ; (⟹) add $s0, $s1, $s2

SAVED REGISTERS

$s0 = a , $s1 = b , $s2 = c , $s3 = d

- ex 2: a = b + c - d ⟺

TEMPORARY REGISTERS

(⟹) sub $t0, $s2, $s3 # t = c - d

add $s0, $s1, $t0.

Table 6.1 : MIPS register set (p. 294 - carte) 317 pdf

- ~~the only~~ hold commonly used operands → faster acces. (faster to check in a smaller area)

Memory: • data can also be stored in memory.

• memory has many data location, but accessing it takes a longer amount of time.

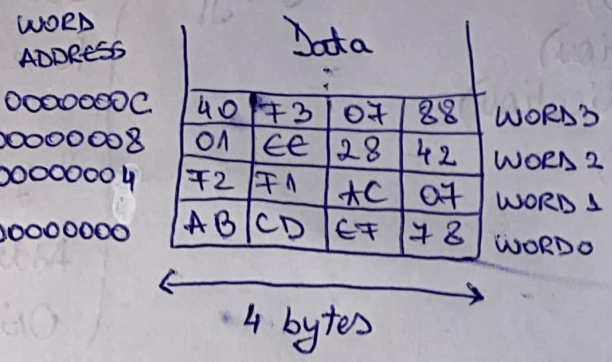
Word-addressable memory

word address	Data	
00000003	40730788	word 3
00000002	01EE2842	word 2
00000001	F271AC07	word 1
00000000	ABCDEF78	word 0

- each 32 bit data word has a unique 32 bit address.

(1)

Byte-addressable memory



The word address is 4 times the word number.

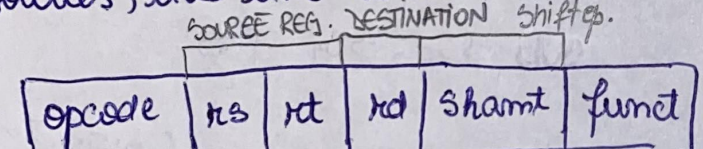
- the MIPS memory model, is byte-addressable, NOT word-addr.
- each data byte has a unique address.
- each word address is a multiple of 4.
- 4 memory locations store a single data from a register.

CONSTANTS / IMMEDIATES

↳ bc. their values are immediately available from the instr. and don't require a register or mem. access.

---

1. R-type (register type) use 3 registers as operands: two as sources, and one as destination.

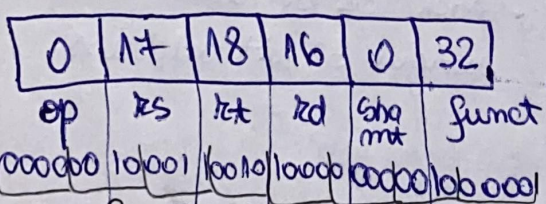


↳ det. which processor components will be util while performing the opperation

↳ which instr. is used.

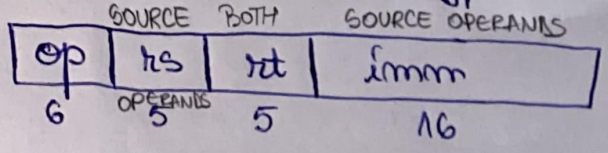
- all R-type instr. have opcode = 000000
- pg. 552 conta 575 Appendix pdf
- all R-type: shamt = 0

(ex) 1) add $S0, $S1, $S2



0x 0 2 3 2 8 0 2 0

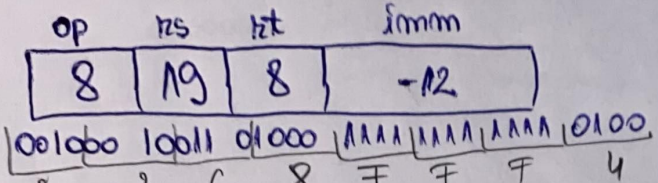
2. i-type (immediate type)



opcode : différit en fct de instr

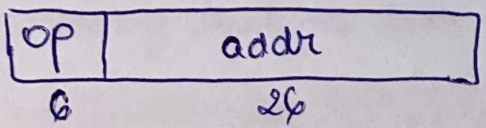
rt : → destination (for ex : addi, lw)
→ source (ex : sw)

(ex) 1) addi $t0, $s3, -12.



0X

3. J-type



---

lui : • load upper immediate instr.
• to assign 32-bit constants.
• loads a 16-bit immediate into the upper half of a register and sets the lower half to 0.

ori : • merges a 16-bit immediate into the lower half.

slt : set less than : [rs] < [rt] ? [rd] = 1 : [rd] = 0

beq : branch if equal : if ([rs] == [rt]) PC = BTA
↳ branch target address

sll : shift left logical : [rd] = [rt] << shamt
moves the bits left.

Ex:

Incep cu LSB

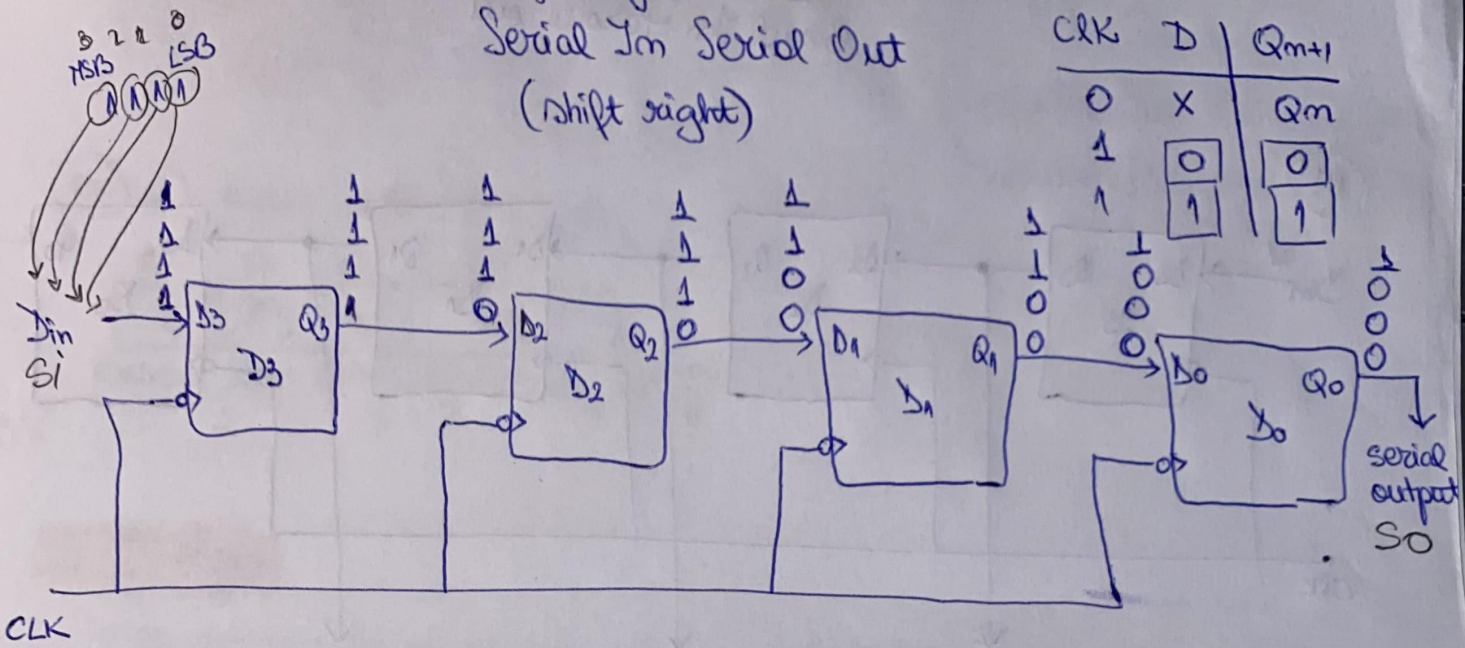
Shift Register (SISO)

Serial In Serial Out

(shift right)

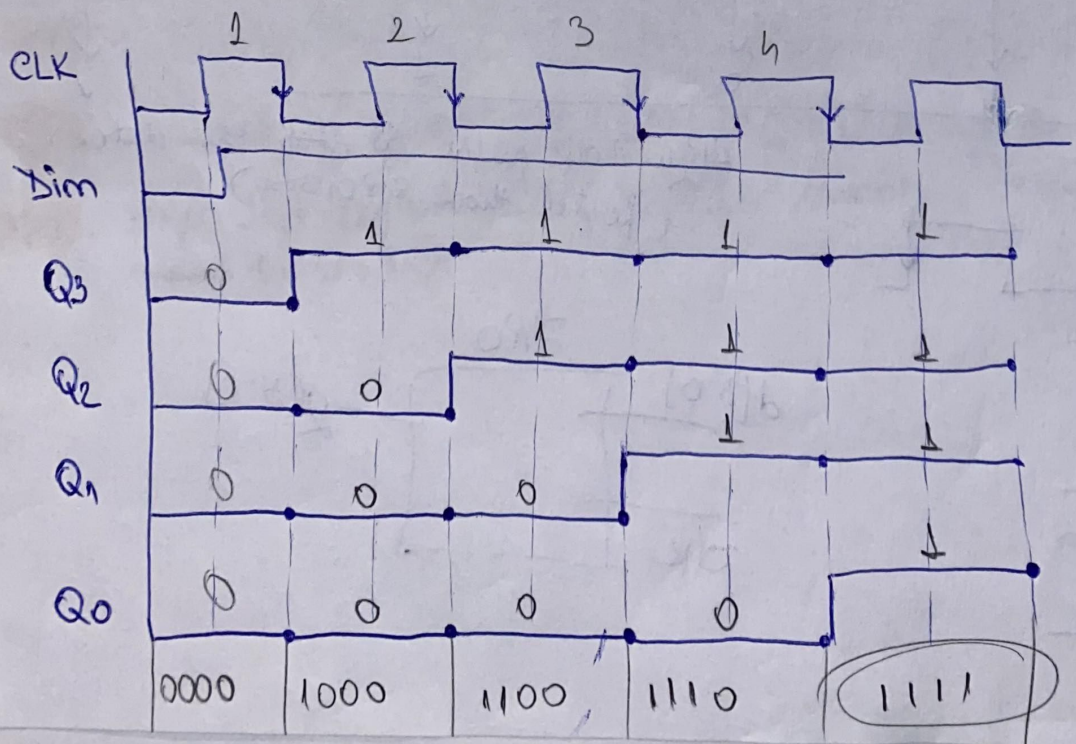
D:

CLK	D	Q_{n+1}
0	X	Q_n
1	0	0
1	1	1



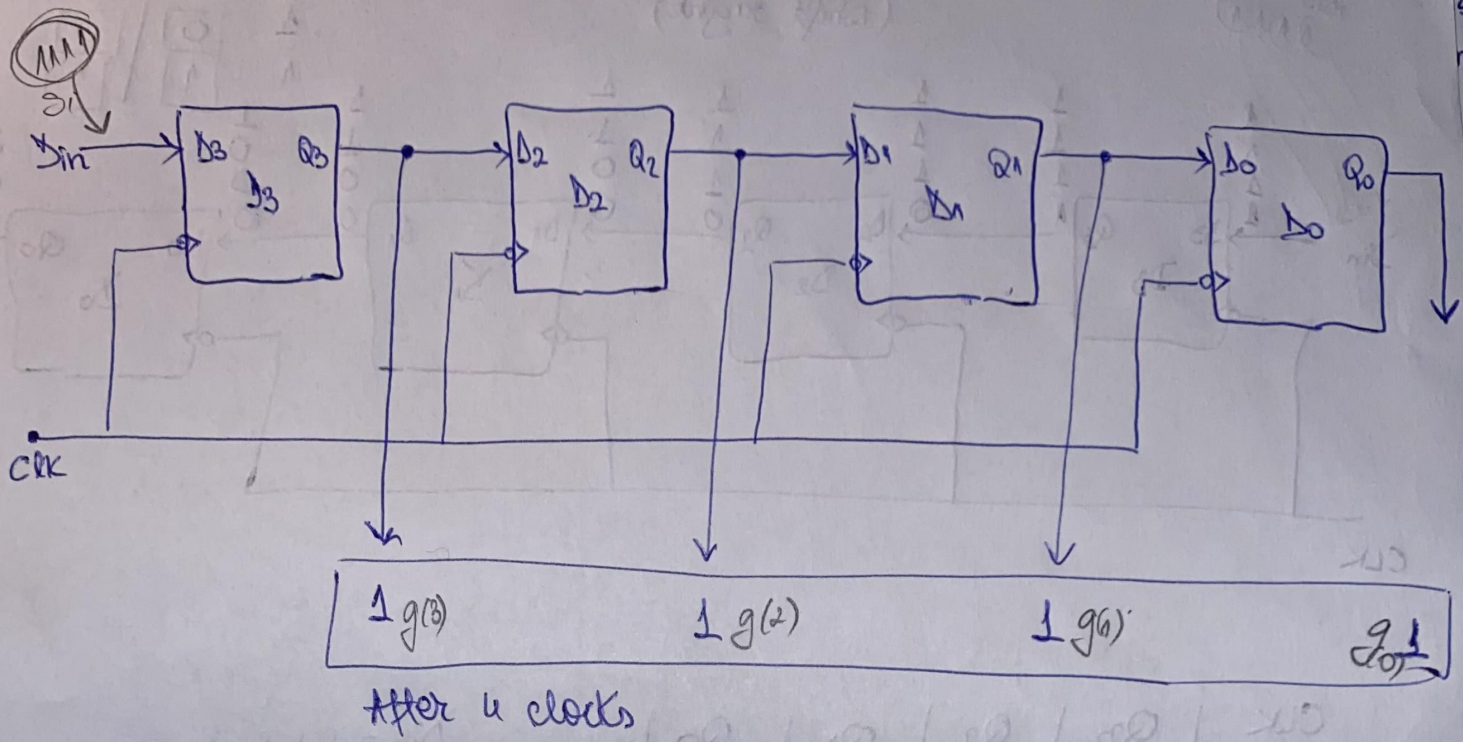
CLK	$q[3]$ Q_3	$q[2]$ Q_2	$q[1]$ Q_1	$q[0]$ Q_0
0 Initially	0	0	0	0
1 ↓	1	0	0	0
2 ↓	1	1	0	0
3 ↓	1	1	1	0
4 ↓	1	1	1	1

Din = 1 1 1 1



SIPO

Serial In Parallel out

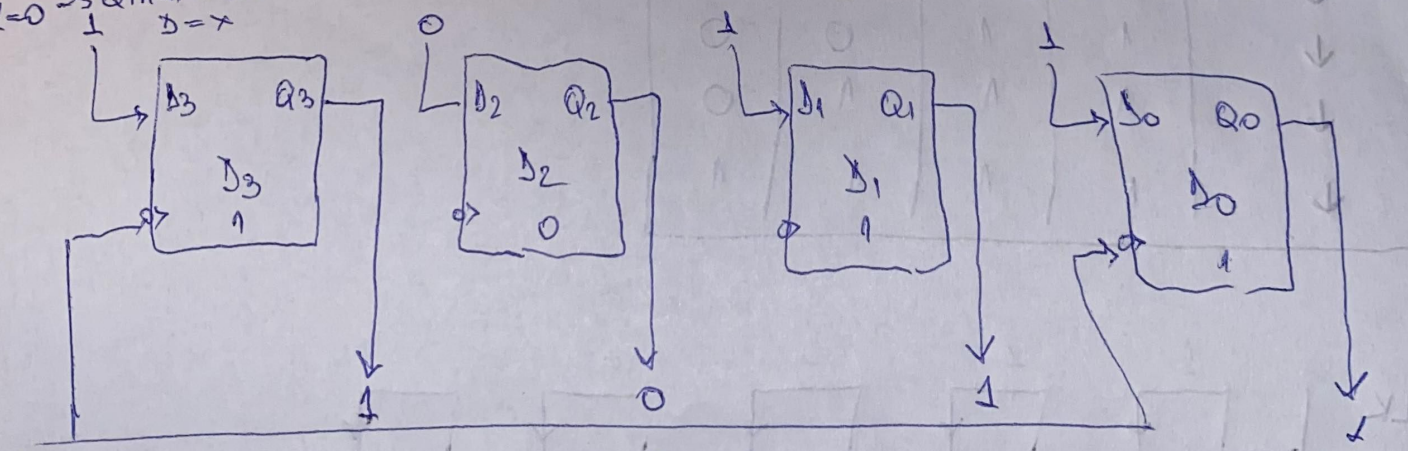


After 4 clocks

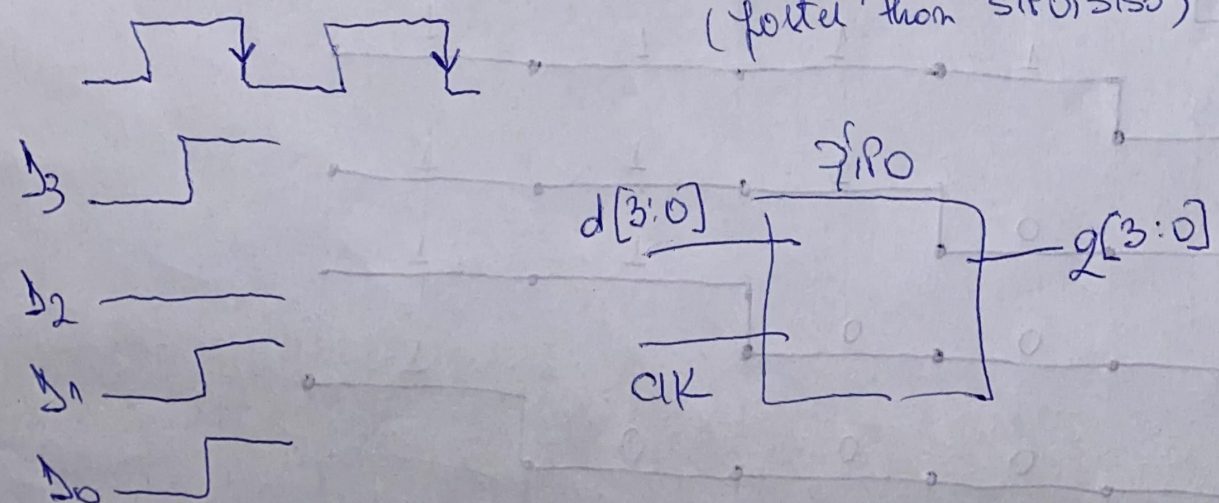
6.41

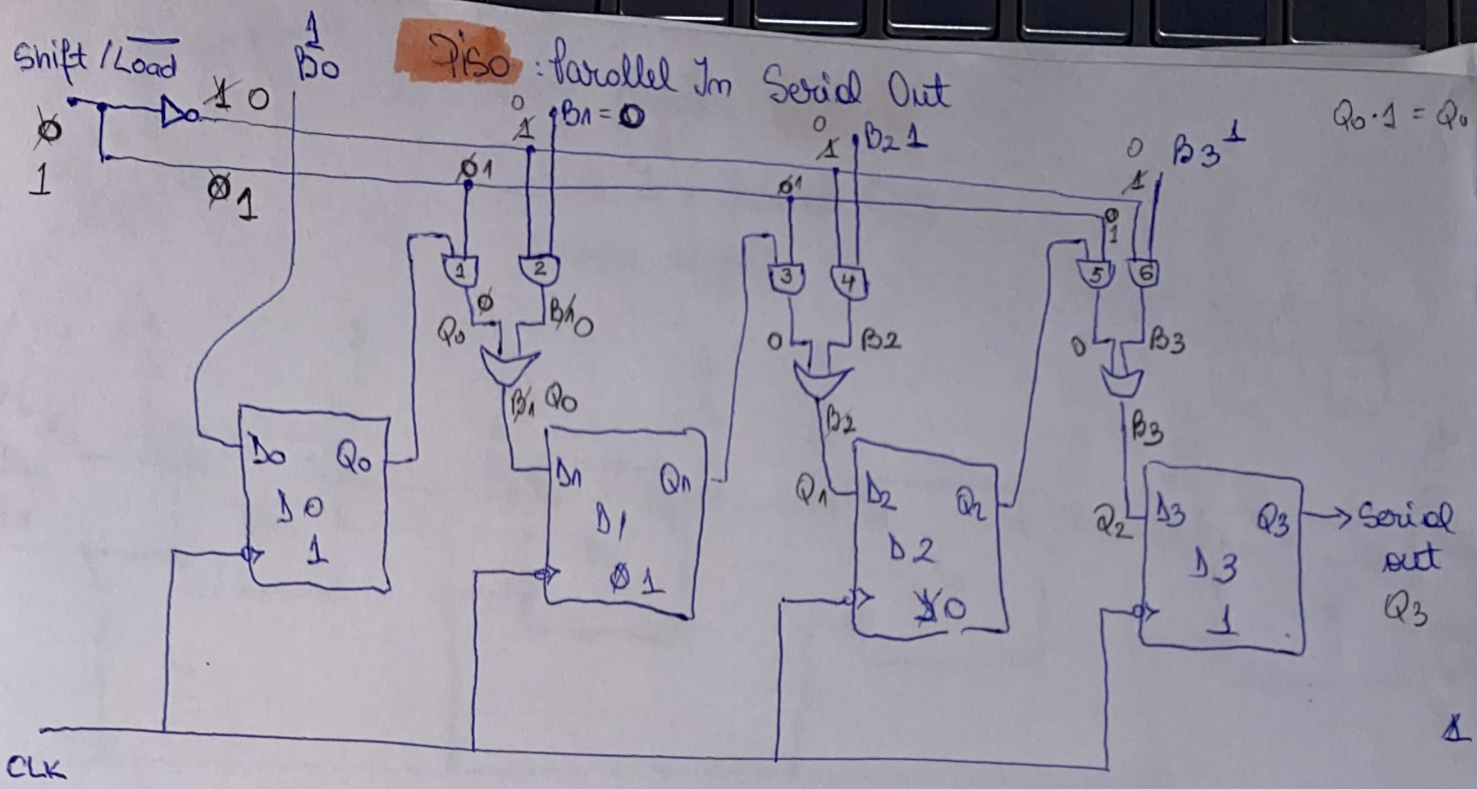
PIPO - Parallel In Parallel out (Storage register / Buffer reg)

D=0 → Qn+1 = 0
D = 1 → Qn+1 = 1
CLK=0 → Qn+1 = Qn



Requ. 1 clk pulse to store the data.
(faster than SIPO, SISO)

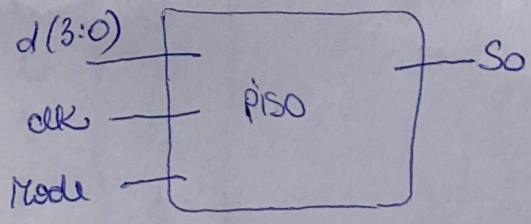




1st mode: Load Mode : 0
2nd mode: Shift Mode : 1 (Q1 was the output for D0
Now is the input for D1)

1 1 1 1
1 0 1 1
1 0 1 1

if mode = 1 parallel load
mode = 0 serial shift



Godbolt
Godbold

Karnaugh Maps

$\overline{A \oplus B} = (A \cdot B) + (\bar{A} \cdot \bar{B}) = A \odot B$

$A \oplus B = \bar{A} \cdot B + A \cdot \bar{B}$

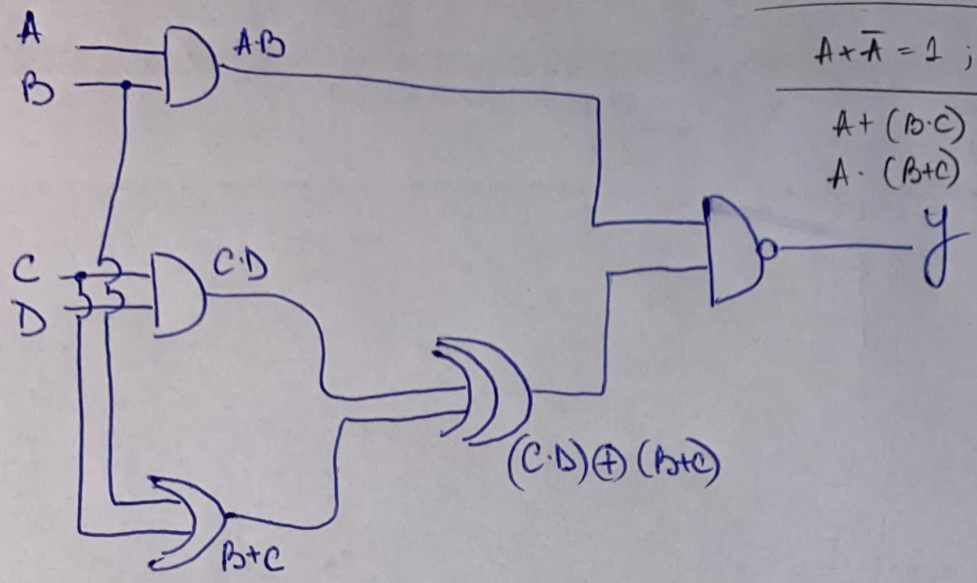
$A + \bar{A} = 1 \; ; \; A \cdot \bar{A} = 0 \; ; \; A + A = A \; ; \; A \cdot A = A$

$A + (B \cdot C) = (A + B) \cdot (A + C)$

$A \cdot (B + C) = A \cdot B + A \cdot C$

$A + 1 = 1 \; ;$

$A \cdot 0 = 0 \; ;$



$\Rightarrow y = \overline{[(C \cdot D) \oplus (B + C)] \cdot (A \cdot B)} =$

$= \overline{(C \cdot D) \oplus (B + C)} + \overline{(A \cdot B)} =$

$= [(C \cdot D) \cdot (B + C)] + [\overline{(C \cdot D)} \cdot \overline{(B + C)}] + \bar{A} + \bar{B} =$

$= (C \cdot D \cdot B + C \cdot D \cdot C) + (\bar{B} \cdot \bar{C} \cdot \bar{C} + \bar{B} \cdot \bar{C} \cdot \bar{D}) + \bar{A} + \bar{B} =$

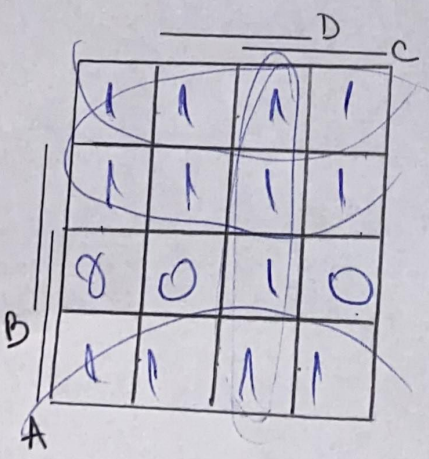
$= B \cdot C \cdot D + C \cdot D + \bar{B} \cdot \bar{C} + \bar{B} \cdot \bar{C} \cdot \bar{D} + \bar{A} + \bar{B} \Rightarrow \overline{(A \cdot B)}$

$= C \cdot D \underbrace{(B + 1)}_{1} + \bar{B} \cdot \bar{C} \underbrace{(1 + \bar{D})}_{1} + \bar{A} + \bar{B} =$

$= C \cdot D + \bar{B} \cdot \bar{C} + \bar{A} + \bar{B} =$

$= C \cdot D + \bar{B} (\bar{C} + 1) + \bar{A} =$

$= C \cdot D + \bar{A} + \bar{B} \cdot$



A	B	C	D	y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

$$y = \bar{A} + \bar{B} + C \cdot D$$

Temă: Adunare a doi vectori

```
int i;
int sum[5], a[5], b[5];

for (i=0; i<5; i++) {
        sum[i] = a[i] + b[i]; }
```

$s0 = sum base address
$s1 = a base address
$s2 = b base address
$s3 = i

```
add $t5, $t3, $t4   # $t5 = a[i] + b[i]
sw $t2, 0($s0)      # sum[i] = a[i] + b[i]
addi $s3, $s3, 1    # i = i + 1
j loop              # repeat
done:
```

initialization code

```
lui $s0, 0x1234     # $s0 = 0x12340000
ori $s0, $s0, 0x5000  # $s0 = 0x12345000   ) adresa lui sum
lui $s1, 0x2A01     # $s1 = 0x2A010000
ori $s1, $s1, 0x7000  # $s1 = 0x2A017000   ) adresa lui a
lui $s2, 0x37C2     # $s2 = 0x37C20000
ori $s2, $s2, 0x7000  # $s2 = 0x37C27000   ) adresa lui b
addi $s3, $0        # i = 0
addi $t0, $0, 5     # $t0 = 5
```

```
loop:
slt $t1, $s3, $t0   # i < 5 ?
beq $t1, $0, done   # if not then done
sll $t1, $s3, 2     # $t0 = i*4 (byte offset)
add $t1, $t1, $s0   # address of sum[i].
lw $t2, 0($t1)      # $t2 = sum[i]
add $t1, $t1, $s1   # address of a[i]
lw $t3, 0($t1)      # $t3 = a[i]
add $t1, $t1, $s2   # address of b[i]
lw $t4, 0($t1)      # $t4 = b[i]
```