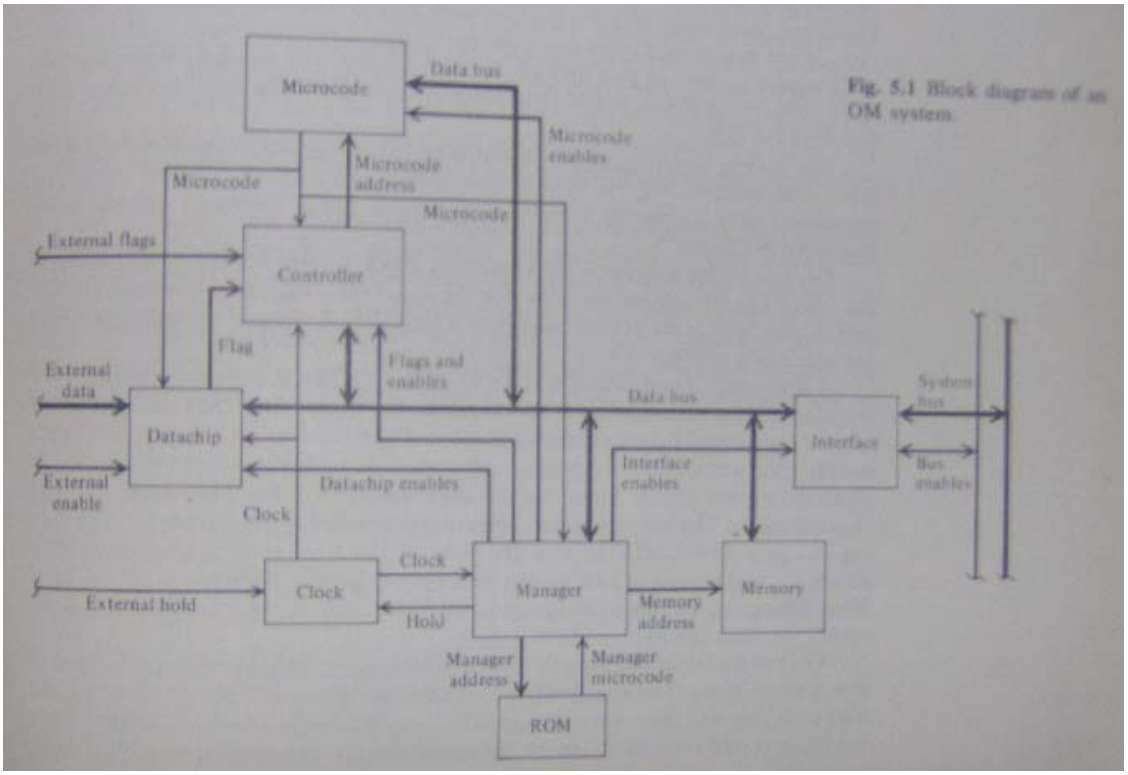


Fig.1 Schema bloc a microprocesorului

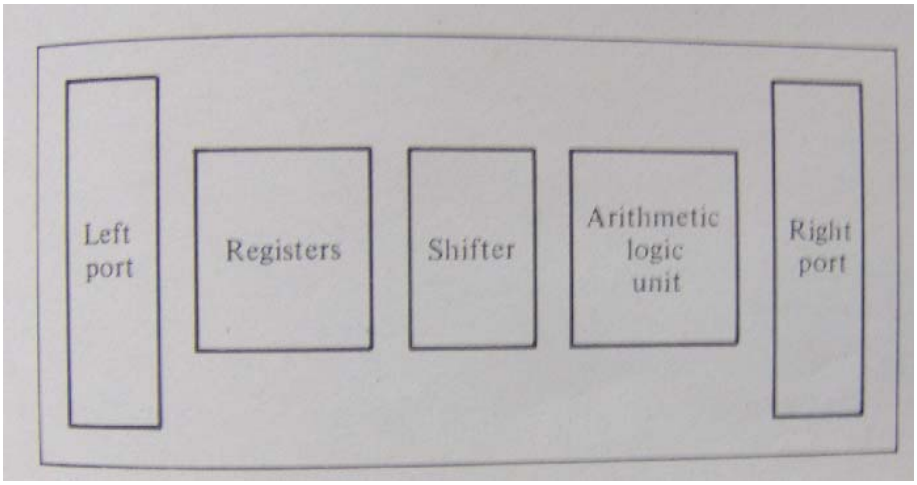


Fig. 2.Planul general al unitatii de executie

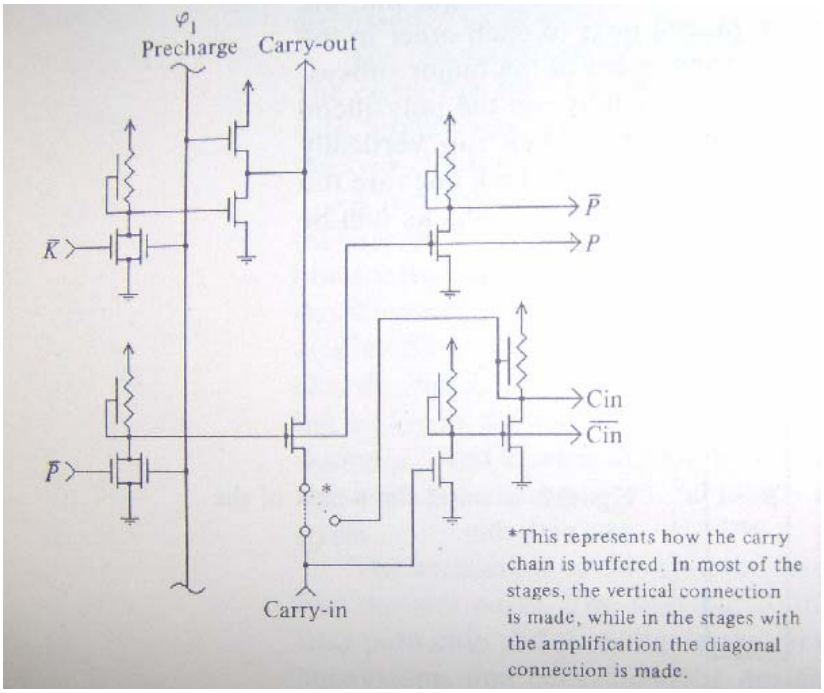


Fig.3. Lantul de transport de tip Manchester, al unitatii de executie

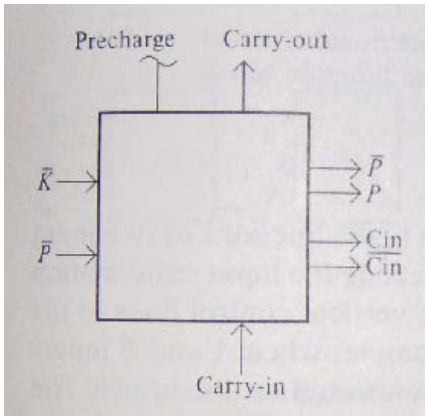


Fig.4. Reprezentarea abstracta a lantului de transport.

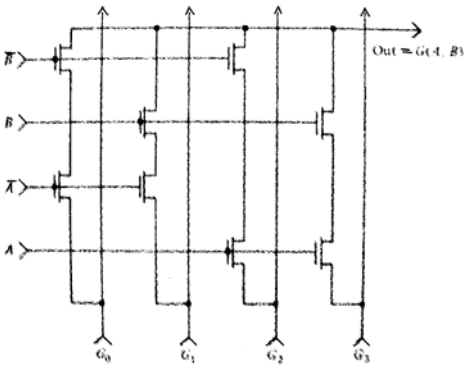


Fig. 5 Diagrama retelei logice programabile, la nivelul tranzistoarelor de trecere,

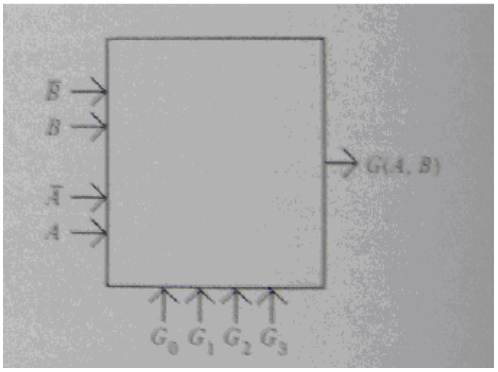


Fig.6. Reprezentarea functionala abstracta a retelei logice programabile.

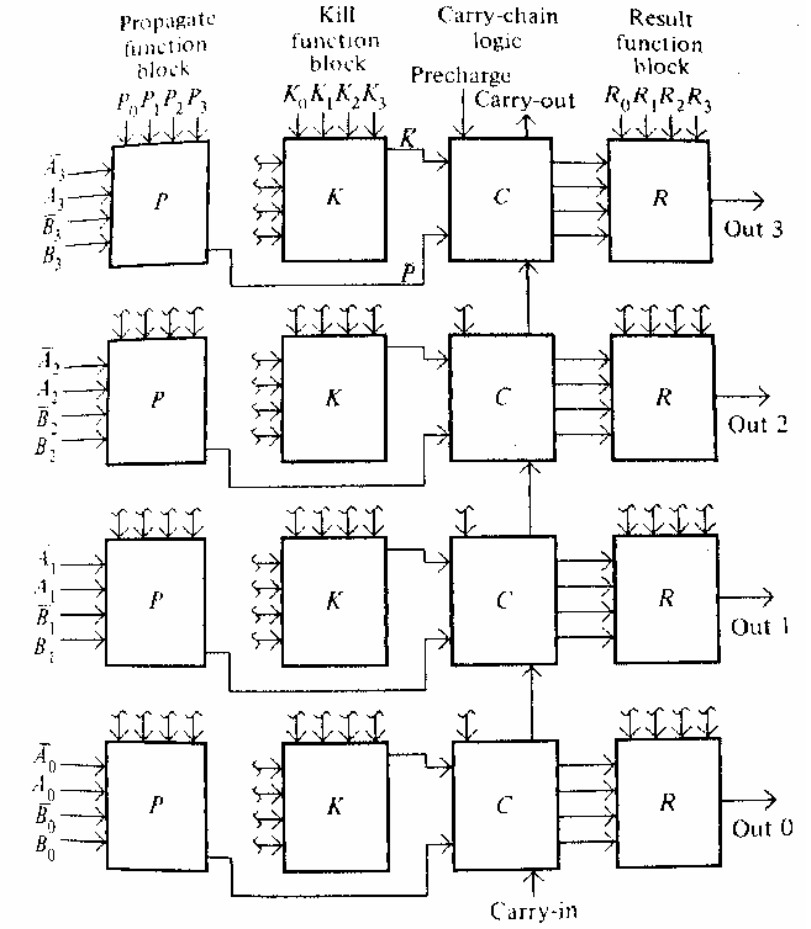


Fig.7. Schema bloc a unei UAL pe 4 biti

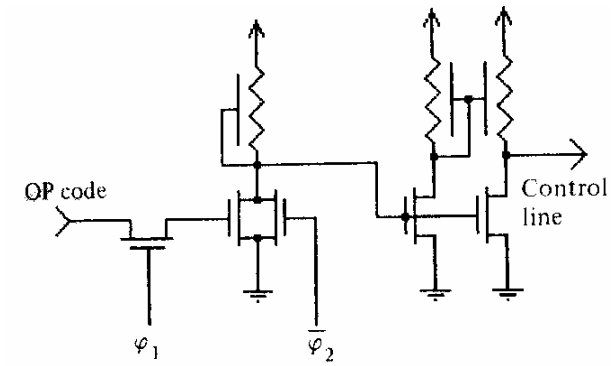


Fig.8. Circuit de comanda pentru UAL. Toate iesirile sunt la nivel ridicat pe durata lui φ2 negat; termenii selectati sunt pe nivel coborat pe durata lui φ2; codul de operatie este valid pe durata lui φ1.

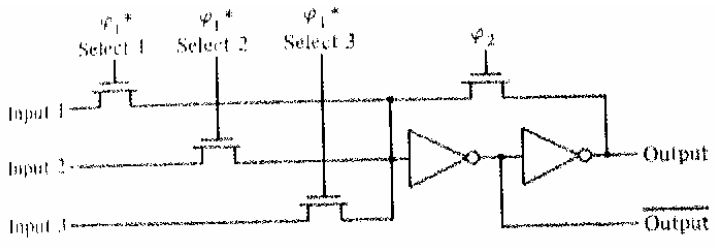


Fig.9. Registru de intrare UAL si multiplexor.

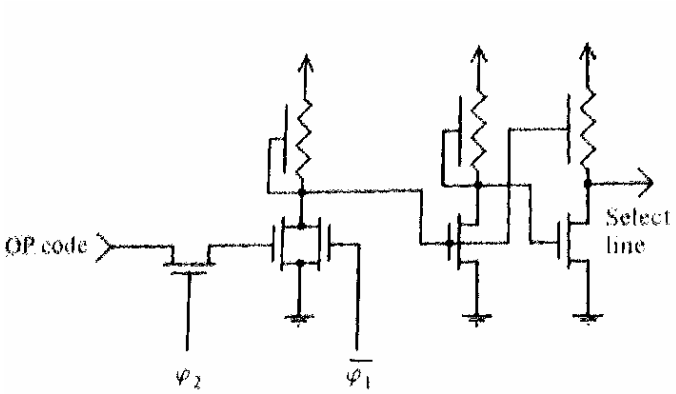


Fig.10. Circuit de comanda pentru selectie . Toate iesirile sunt la nivel coborat pe durata lui φ1negat; termenii selectati sunt pe nivel ridicat pe durata lui φ1; codul de operatie este valid pe durata lui φ2.

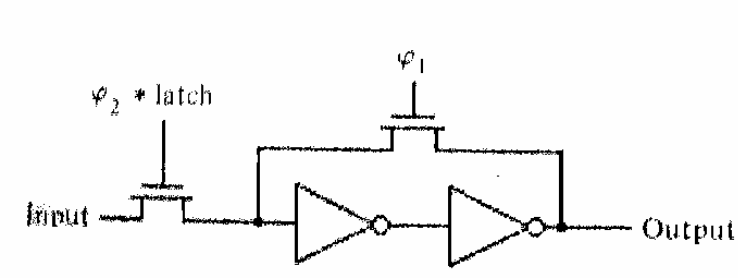


Fig.11. Registru de iesire.

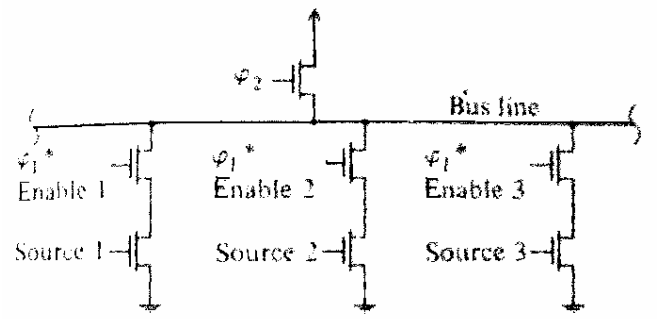


Fig.12. Circuit de magistrala preincarcat

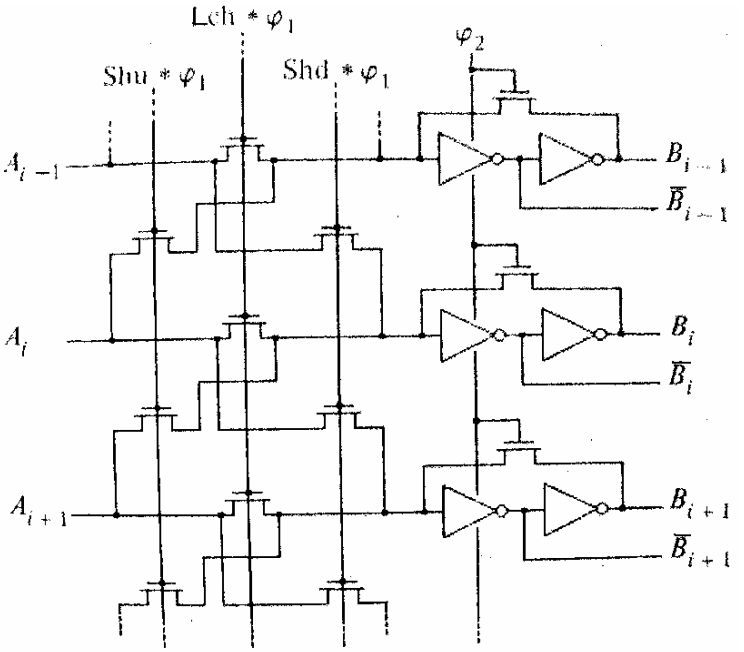


Fig. 13. Circuit simplu de deplasare cu un bit spre stanga.

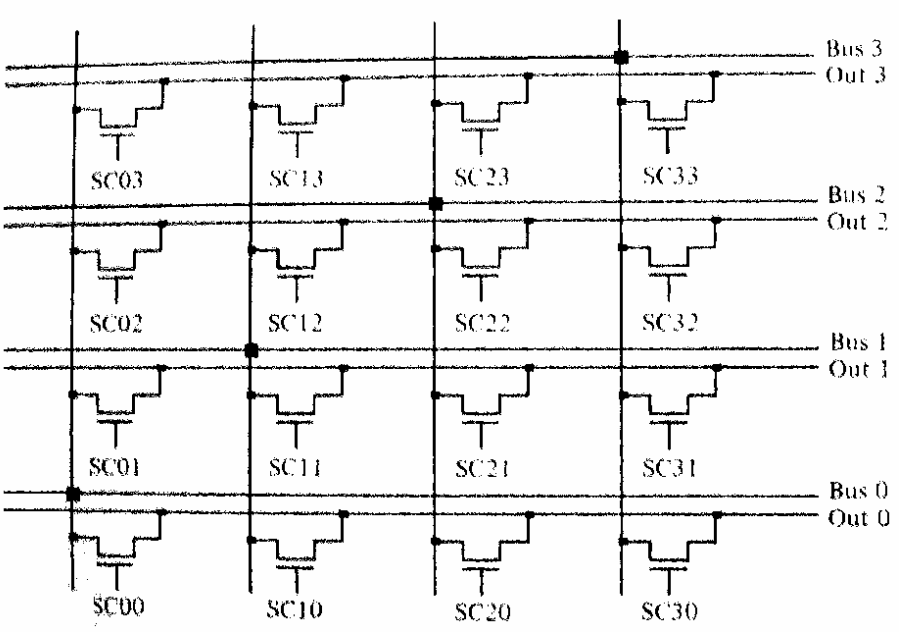


Fig. 14. Comutator crossbar 4 x 4.

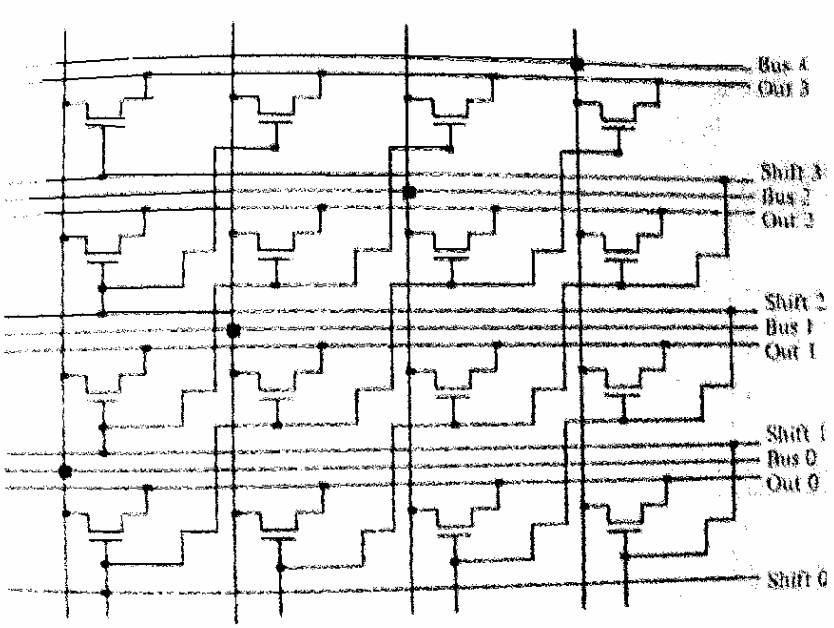


Fig. 15. Circuit de deplasare circulara (barrel shifter) 4 x 4.

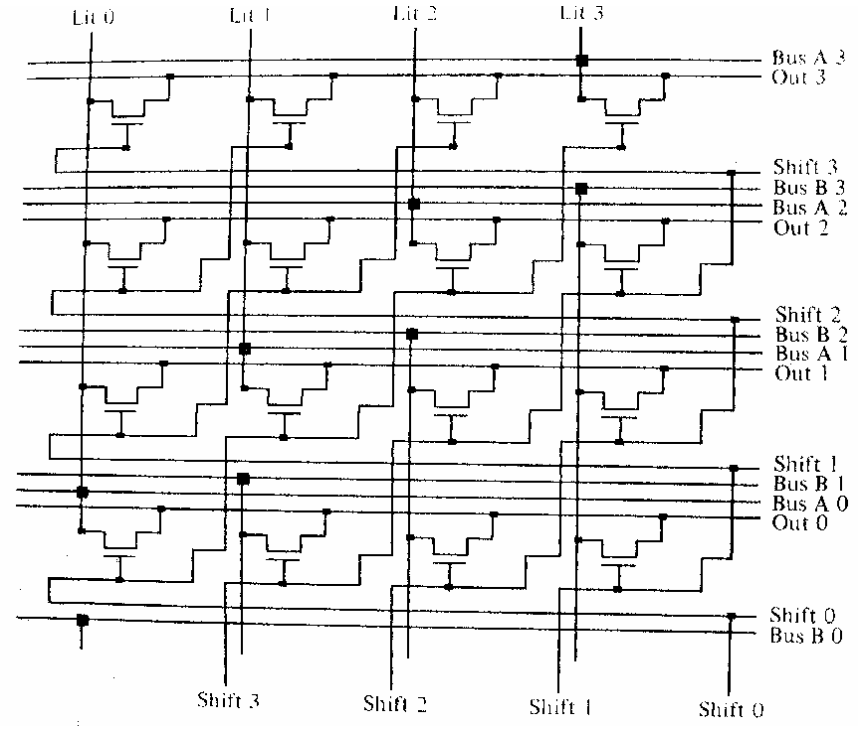


Fig. 16. Circuit de deplasare 4 x 4, cu trasee verticale sectionate si doua magistrale de date.

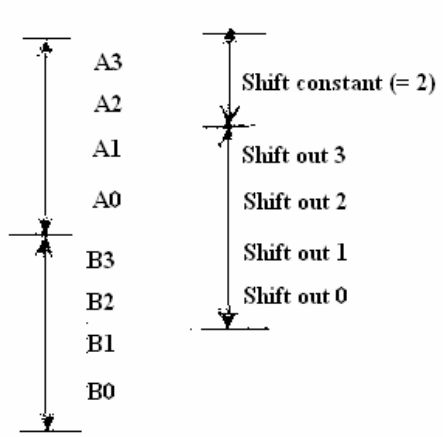


Fig . 17. Reprezentarea conceptuala a operarii circuitului de deplasare.

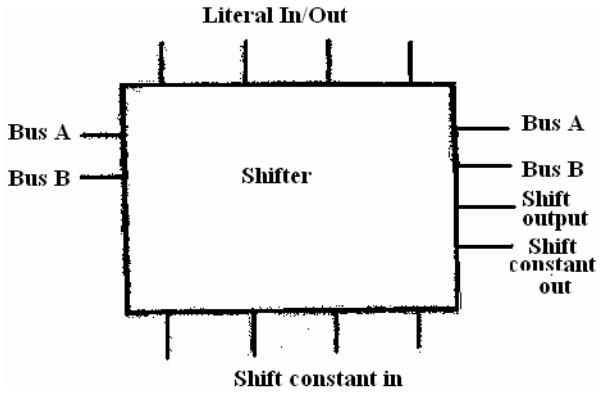


Fig. 18. Diagarma bloc a circuitului de deplasare.

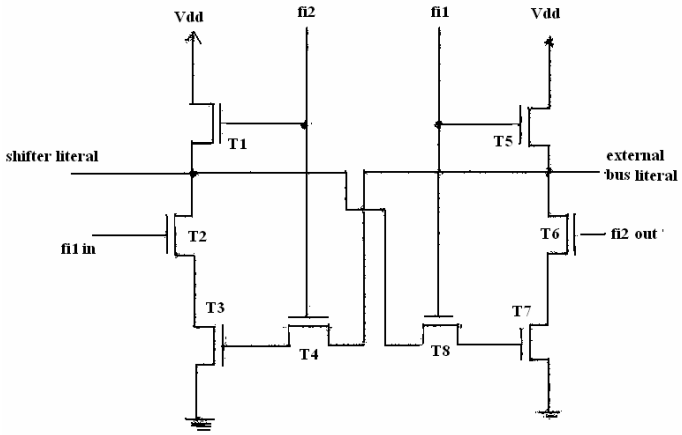


Fig. 19. Interfata Literal.

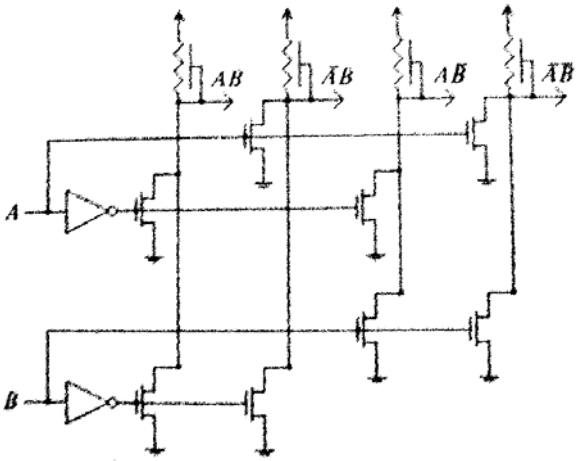


Fig. 20. Decodificator bazat pe NOR

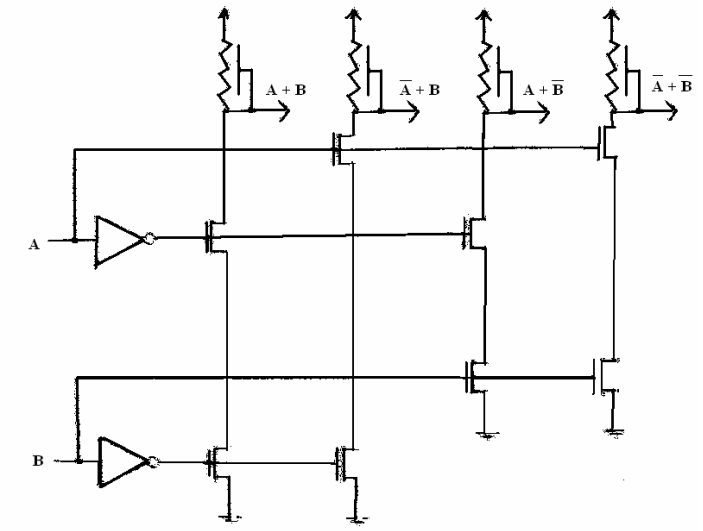


Fig. 21. Decodificator bazat pe NAND

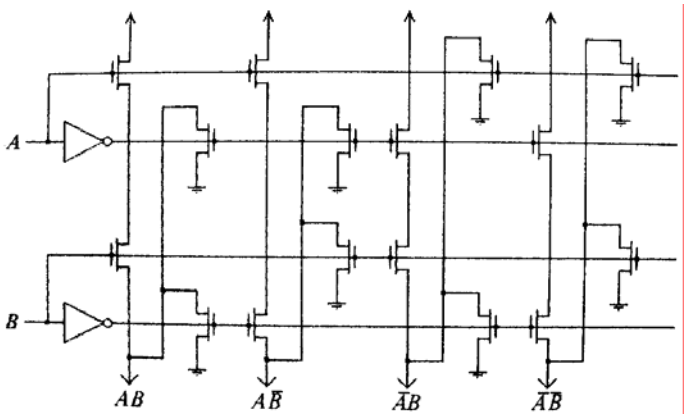


Fig. 22 Decodificator complementar.

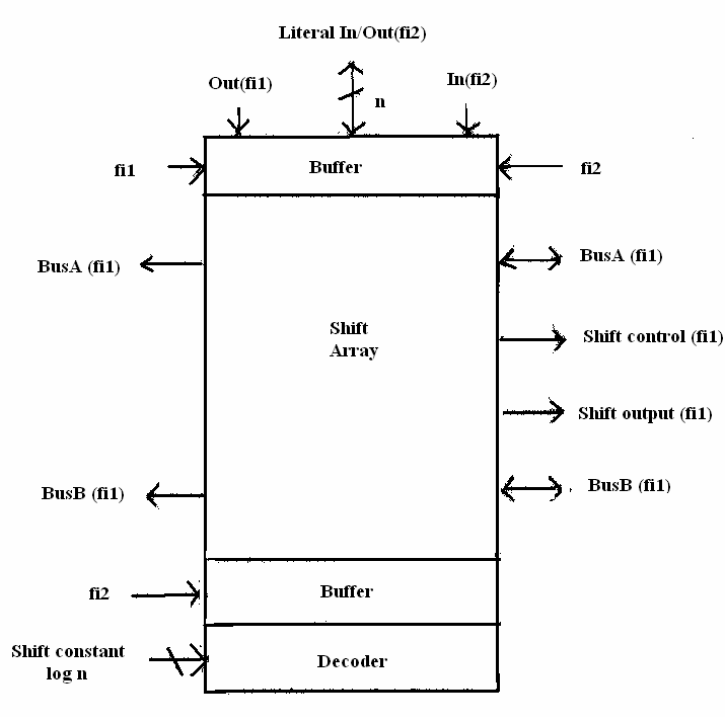


Fig. 23. Circuit de deplasare complet sincronizat.

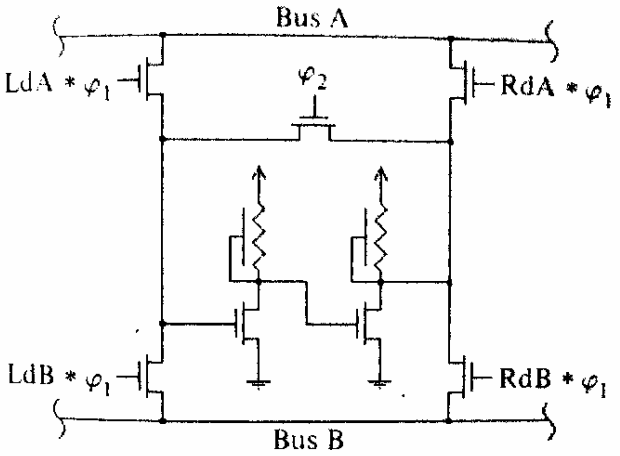


Fig. 24 Celula de registru biport.

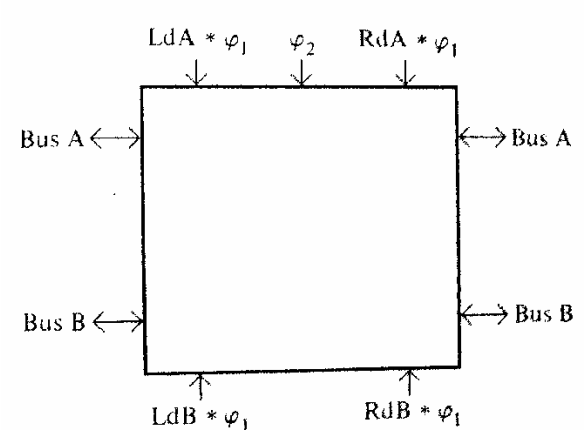


Fig. 25. Diagrama bloc a unei celule de registru biport.

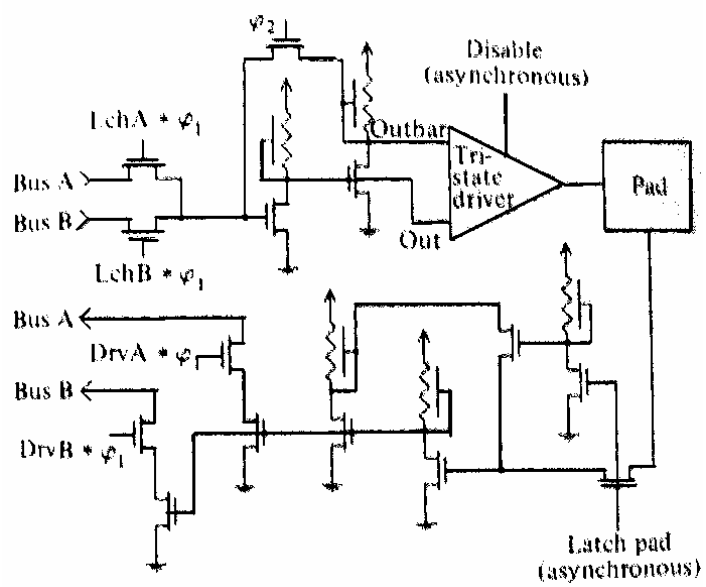


Fig. 26. Circuitul de I/E conectat la un plot bidirectional TS.

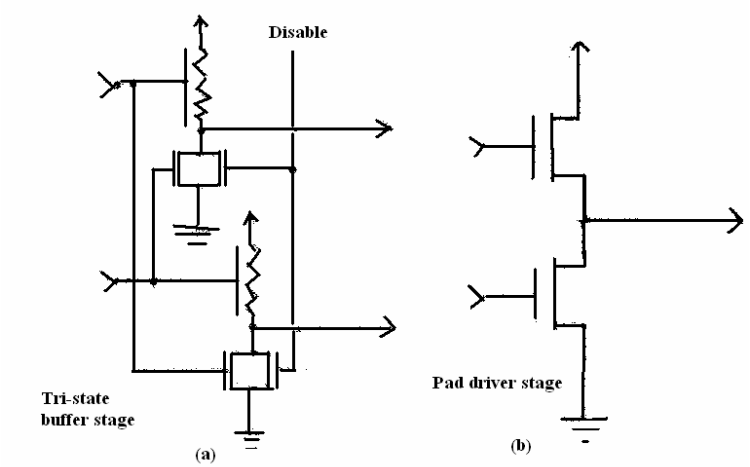


Fig. 28 Etaj tampon TS (a) si Circuit de comanda a plotului (b).

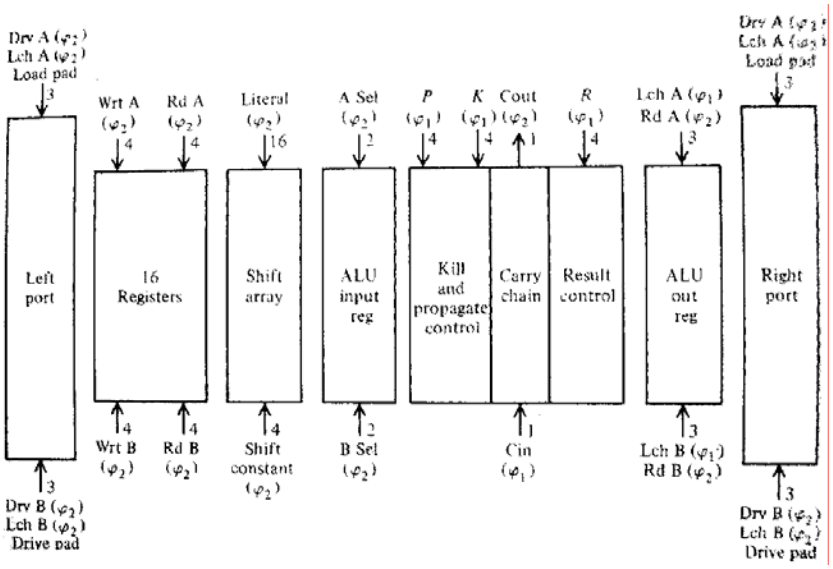


Fig. 29. Diagrama bloc a UAL, cu mentionarea traseelor de comanda.

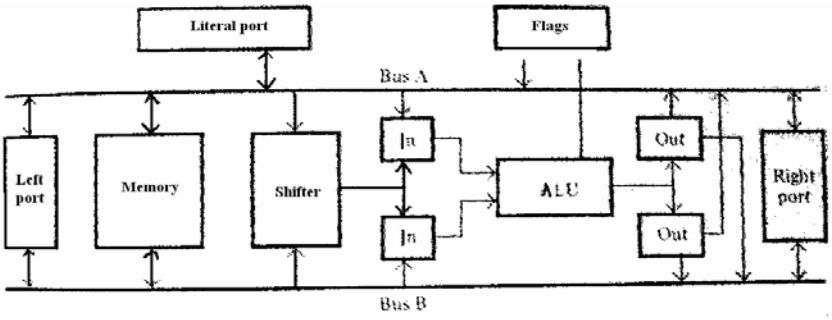


Fig. 30. Diagrama bloc a UAL.

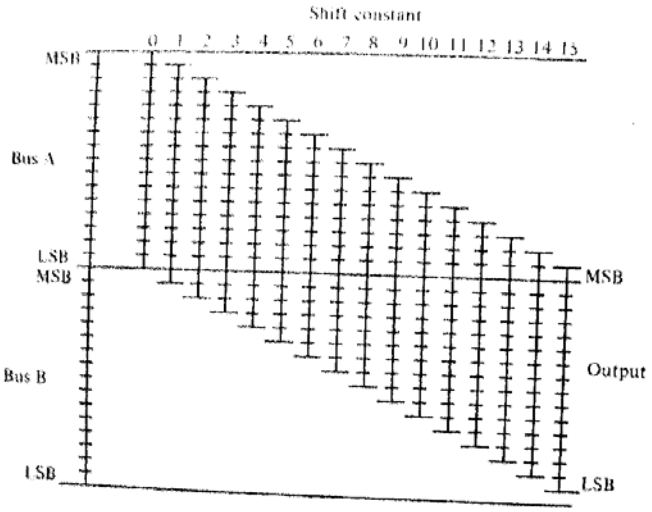


Fig. 31. Operarea circuitului de deplasare.

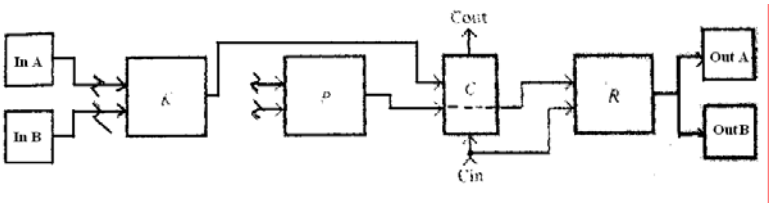


Fig. 32. Diagrama bloc a unui bit al UAL.

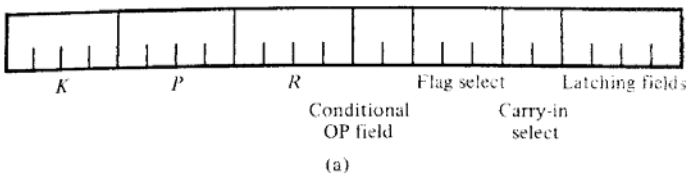


Fig. 33 (a). Faza 2 COP (intra in φ1)

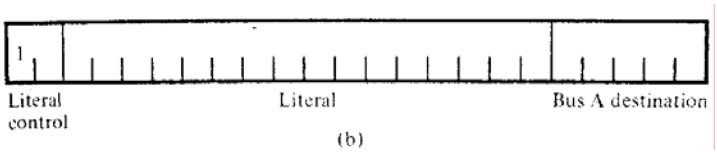


Fig. 33 (b). Faza 1 transfer COP de la Literal (intra in φ2)

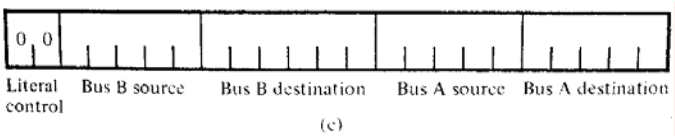


Fig. 33 (c). Faza 1 COP normal (intra in φ2)

Tab. 1 Transferuri pe magistrale

	Bus A Source		Bus A Destination
0*nnnn*	Register *n*	0*nnnn*	Register *n*
10000	Right port pins	10000	Left port, drive now
10001	Right port latch	10001	Left port, drive ϕ_2
10010	Left port pins	1001*x*	Left port, no drive
10011	Left port latch	10100	Right port, drive now
10100	ALU output latch *A*	10101	Right port, drive ϕ_2
10101	ALU output latch *B*	1011*x*	Right port, no drive
10110	Flag register	11000	ALU input latch *A*
		11001	ALU input latch *A* gets shift out
		11010	ALU input latch *A* gets shift control
		11011	Flag register

Tab. 1 Transferuri pe magistrale

Bus B Source		Bus B Destination	
0*nnnn*	Register *n*	00*nnnn*	Register *n*
10000	Right port pins	010000	Left port, drive now
10001	Right port latch	010001	Left port, drive φ_2
10010	Left port pins	01001*x*	Left port, no drive
10011	Left port latch	010100	Right port, drive now
10100	ALU output latch *A*	010101	Right port, drive φ_2
10101	ALU output latch *B*	01011*x*	Right port, no drive
		0110*xx*	ALU input latch *B*
		10*nnnn*	ALU input latch *B* gets shift output, shift constant = *n*
		11*nnnn*	ALU input latch *B* gets shift control, shift constant = *n*

c

Tab2. Codificarea operatiilor in UAL, la nivelul microinstructiunilor.

	K	*P*	*R*	Cin	Cond	
$A + B$	1	6	6	0	0	Add
$A + B +$ Cin	1	6	6	1	0	Add with carry
$A - B$	2	9	6	2	0	Subtract
$B - A$	4	9	6	2	0	Subtract reverse
$A - B -$ Cin	2	9	6	1	0	Subtract with borrow
$B - A -$ Cin	4	9	6	1	0	Subtract reverse with borrow
$-A$	12	3	6	2	0	Negative *A*
$-B$	10	5	6	2	0	Negative *B*
$A + 1$	3	12	6	2	0	Increment *A*
$B + 1$	5	10	6	2	0	Increment *B*
$A - 1$	12	3	9	2	0	Decrement *A*
$B - 1$	10	5	9	2	0	Decrement *B*
$A \wedge B$	0	8	12	0	0	Logical AND
$A \vee B$	0	14	12	0	0	Logical OR
$A \oplus B$	0	6	12	0	0	Logical EXOR
$\neg A$	0	3	12	0	0	Not *A*
$\neg B$	0	5	12	0	0	Not *B*
A	0	12	12	0	0	*A*
B	0	10	12	0	0	*B*
Mul	1	14	14	0	1	Multiply step
Div	3	15	15	0	2	Divide step
A/O	0	14	12	0	3	Conditional AND/OR
Mask	10	5	8	2	0	Generate mask
SHL A	3	0	10	0	0	Shift A left
Zero	0	0	0	0	0	Zero

Tabela 3. Selectarea lui Cin

00	0
01	Flag bit
10	1
11	Flag bit complemented

Tab.4. Selectie OP Conditionala

Select	Flag bit	K	P	R	
0	x	----	----	----	Unconditional
1	0	---0	--0-	--0-	Multiply step
	1	----	0---	0---	
2	0	0--0	-00-	-00-	Divide step
	1	-00-	0--0	0--0	
3	0	----	----	----	AND/OR
	1	----	-00-	----	

Tab. 5. Selectarea noului bit indicator

Select	New flag bit
0	Old flag big
1	Carry-out
2	MSB
3	Zero
4	Less than
5	Less than or equal
6	Higher (in absolute value)
7	Overflow

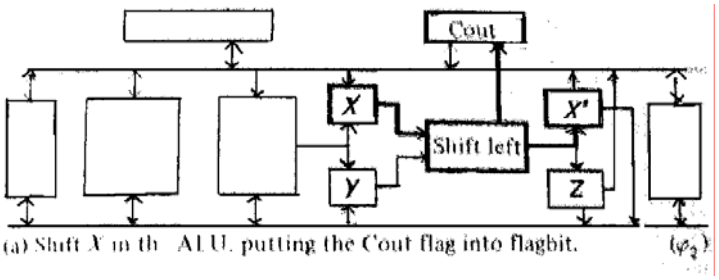
Tab. 6. Indicatorii de conditii.

Bit	Flag
0	Not changed
1	Not changed
2	Not changed
3	Not changed
4	Not changed
5	Previous value of flag bit
6	¬ Carry into MSB stage
7	Less than or equal
8	¬ Higher (in absolute value)
9	¬ Less than
10	LSB
11	¬ Zero
12	MSB
13	Overflow
14	¬ Carry-out
15	Current flag bit

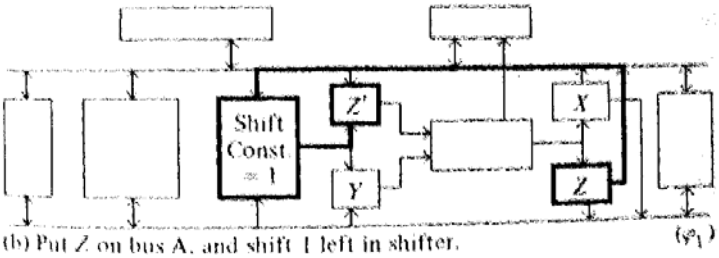
Tab. 7. Campul de memorare(latching)

Latching field	Register loaded
1*xxx*	Flag register loaded with current ALU flags
*x*1*xx*	ALU output latch *A* loaded with the ALU output
*xx*1*x*	ALU output latch *B* loaded with the ALU output
*xxx*1	The literal field during the next φ_2 is loaded with the contents of bus A during the last φ_2
0000	None of these registers are affected

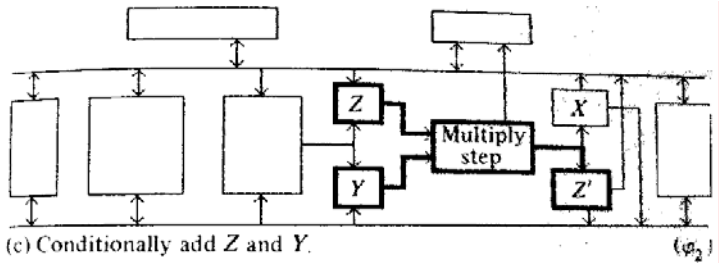
Exemplu de microprogramare a inmultirii Z ←X xY, pe 16 biti.



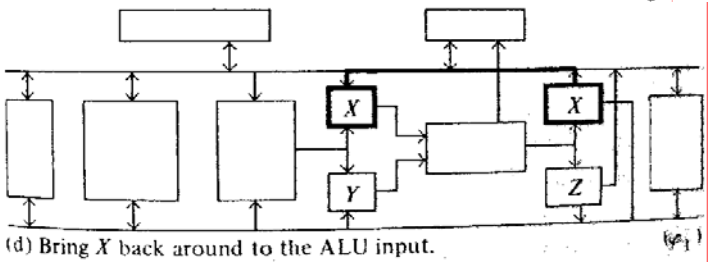
(a) Shift X in th ALU, putting the Cout flag into flagbit. (φ_2)



(b) Put Z on bus A, and shift 1 left in shifter. (φ_1)



(c) Conditionally add Z and Y. (φ_2)



(d) Bring X back around to the ALU input. (φ_1)

φ_2: ALU.Out.A←ALU(Shift A left)←ALU.In.A;
Latch Flags;
φ_1: ALU.In.A←Shift.out, Bus.A←ALU.Out.B;
R[1]←Bus.B←R[0]; This gives a shift constant of 1.
φ_2: ALU.Out.B←ALU(Multiply Step); *conditionally add.*
Flag←Cout;
φ_1: ALU.In.A←Bus.A←ALU.Out.A