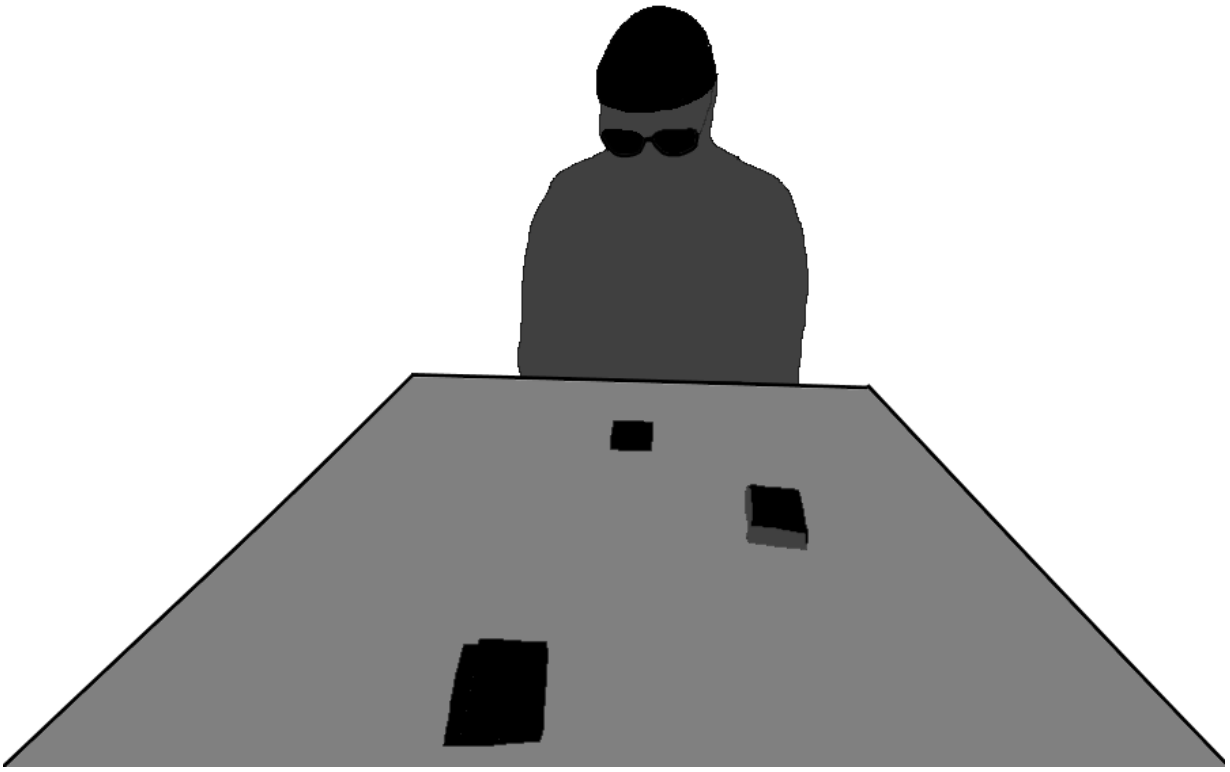
Proiect în echipă: Aparat Blackjack



Brănoiu Cătălin
Sima Andreea
Ștefan Rareș
Ulmeanu Vlad-Adrian
grupa 315CA
Asistent: Teodor Popescu

Cuprins

Tema proiectului

Temă

Tema aleasă prezintă realizarea unui aparat de Blackjack, capabil de a simula unele dintre activităţile jocului real echivalent. Partea jocului ce se doreşte imitată în cadrul proiectului presupune existenţa unui jucător şi a unui pachet de cărţi, ale căror valori sunt cuprinse între 3 şi 10 inclusiv.

Jucătorul depune o sumă de bani, apoi trage câte cărţi doreşte, una câte una, putându-se opri după orice tragere, dacă suma valorilor cărţilor pe care le-a tras până atunci este cel mult egală cu 21. Jocul se termină automat dacă scorul jucătorului a depăşit 21.

Dacă după terminarea jocului scorul jucătorului este cel mult 21, acesta este plătit cu o sumă care variază în funcţie de mai multe praguri: scor cel puţin 20, scor cel puţin 16, sau scor cel puţin 12. Dacă scorul a depăşit 21, sau scorul este mai mic strict decât 12, se consideră că jucătorul a pierdut şi nu primeşte nicio recompensa monetară.

Detalii de implementare

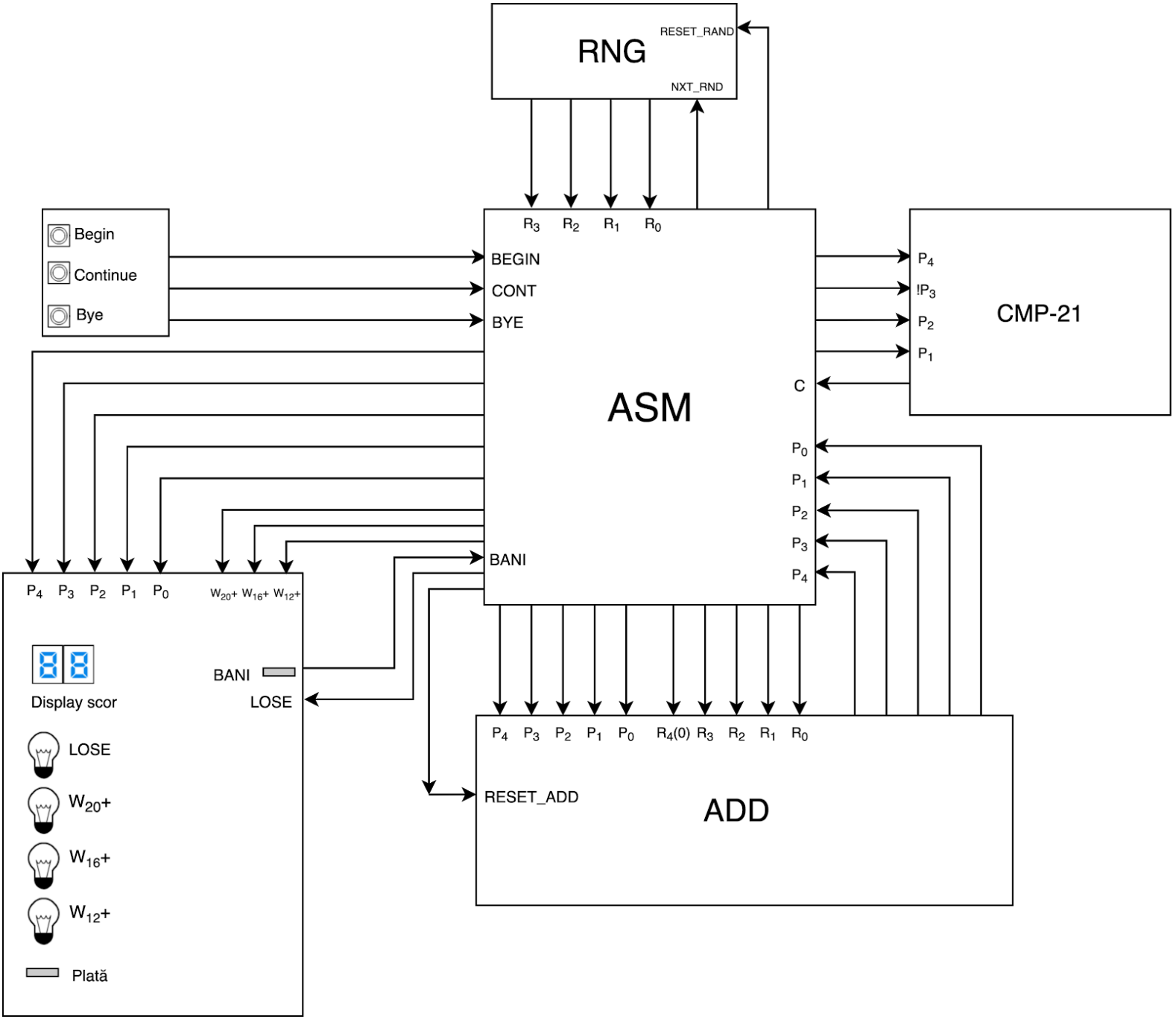
Aparatul va fi implementat pe o unitate de comandă microprogramată cu microinstrucţiuni cu format variabil.

Pentru proiectarea cablajului am ales următoarele componente digitale:

- 3 memorii **CAT22C10 64x4**
- 2 numărătoare **74163**
- 1 multiplexor 16:1 **74150**
- 2 IC cu porţi AND **7408**
- 1 IC cu porţi NOR **7402**

Mod de implementare

Realizare schemă bloc



Explicarea intrărilor din schema bloc

- **BEGIN -** jucătorul trebuie să apese butonul corespunzător de pe tastatură pentru a genera un semnal care permite algoritmului să treacă în următoarea fază; aici, introducerea banilor.
- **BANI -** pentru a începe jocul (după ce a apăsat pe **BEGIN**), jucătorul trebuie să introducă de fiecare dată o anumită cantitate de unităţi monetare. **BANI** este 1 doar imediat după ce a fost introdusă unitatea monetară, permiţând algoritmului să continue.
- $\mathbf{P_4, P_3, P_2, P_1, P_0}$ **-** "punctajul" jucătorului, reprezentat în biţi. Cel mai semnificativ bit este $\mathbf{P_4}$, iar cel mai puţin semnificativ este $\mathbf{P_0}$. Acesta este la început 0 şi nu va depăşi în niciun caz 31, oricum s-ar efectua adunarea (descrisă mai jos). $\mathbf{P_4, P_3, P_2, P_1, P_0}$ sunt reţinute fiecare într-un CBB-D.
- $\mathbf{R_3, R_2, R_1, R_0}$ **-** valoarea celei mai noi cărţi extrase, oferită de un generator de numere aleatoare (RNG). Generatorul din acest proiect are perioada 16 şi generează numere "aleatoare" din intervalul [3, 10]. $\mathbf{R_3, R_2, R_1, R_0}$ sunt reţinute fiecare într-un CBB-D.
- **CONT -** dacă jucătorul vrea să mai primească o carte, trebuie să genereze semnalul corespunzător prin apăsarea butonului de pe tastatura. În urma generării semnalului, submodulul RNG este rugat să dea un număr nou (să actualizeze valorile $\mathbf{R_3, R_2, R_1, R_0}$), se aşteaptă să se treacă un ciclu al ceasului (astfel încât valorile $\mathbf{R_3, R_2, R_1, R_0}$ să fie scrise în CBB-urile corespunzătoare), apoi este apelat submodulul ADD care adaugă valoarea cărţii (**R**) la punctaj (**P**).
- **BYE -** dacă jucătorul nu mai vrea să primească cărţi, trebuie să genereze semnalul corespunzător prin apăsarea butonului de pe tastatură. În urma generării semnalului, se va trece la partea de plată a jucătorului în funcţie de punctajul lui obţinut. Se va genera exact una din următoarele ieşiri în funcţie de punctaj: $\mathbf{W_{20}+}$, $\mathbf{W_{16}+}$, $\mathbf{W_{12}+}$ sau **LOSE**.
- **C (ieşirea din submodulul de comparaţie cu 21) -** în urma comparaţiei lui **P** cu 21, **C** va deveni 1 dacă **P** este mai mare strict decât 21, sau 0 altfel.
    - Pentru a uşura complexitatea proiectului, submodulul CMP-21 a fost integrat în ASM, el fiind apelat doar într-o singură bucată din organigramă. Astfel, pentru a indica "**C** == 1", se va trece într-o stare anume (în organigramă s_8), iar pentru a indica "**C** == 0", se va trece într-o stare anume (în organigramă s_7).

Explicarea ieşirilor din schema bloc

- **RESET_ADD, RESET_RAND -** se apelează la începutul fiecărui joc pentru a reseta submodulele ADD şi RNG.
- **NXT_RND -** se apelează când se doreşte generarea unei valori noi aleatoare din intervalul [3, 10]. Submodulul RNG, implementat asemenea unui numarator va da valori noi lui $\mathbf{R_3, R_2, R_1, R_0}$ după trecerea unui ciclu.
- $\mathbf{P_4, P_3, P_2, P_1, P_0, W_{20}+, W_{16}+, W_{12}+}$**, LOSE (ieşiri folosite de submodulul display) -** biţii lui P sunt folosiţi pentru a arata jucătorului scorul sau curent. $\mathbf{W_{20}+}$ (scor cel puţin 20), $\mathbf{W_{16}+}$ (scor cel puţin 16), (scor cel puţin 12)$\mathbf{W_{12}+}$**,** (scor sub 12)**LOSE** nu pot fi simultan porniţi şi semnifică terminarea unui joc, întotdeauna urmată de întoarcerea în starea iniţială.
    - **LOSE** mai poate fi apelat (urmat de terminarea implicită a jocului) dacă după apelarea submodulului de comparaţie CMP-21 valoarea **C** este 1 (adică punctajul **P** este mai mare strict decât 21)
- $\mathbf{P_4, P_3, P_2, P_1, P_0, R_4, R_3, R_2, R_1, R_0}$ **(ieşiri folosite de submodulul de adunare) -** submodulul efectuează operaţia de adunare pe 5 biţi între $\mathbf{P_{4..0}}$ şi $\mathbf{R_{4..0}}$. Cum submodulul RNG nu foloseşte decât biţii $\mathbf{R_3, R_2, R_1, R_0}$, bitul $\mathbf{R_4}$ este întotdeauna 0.

Descrierea funcționării aparatului

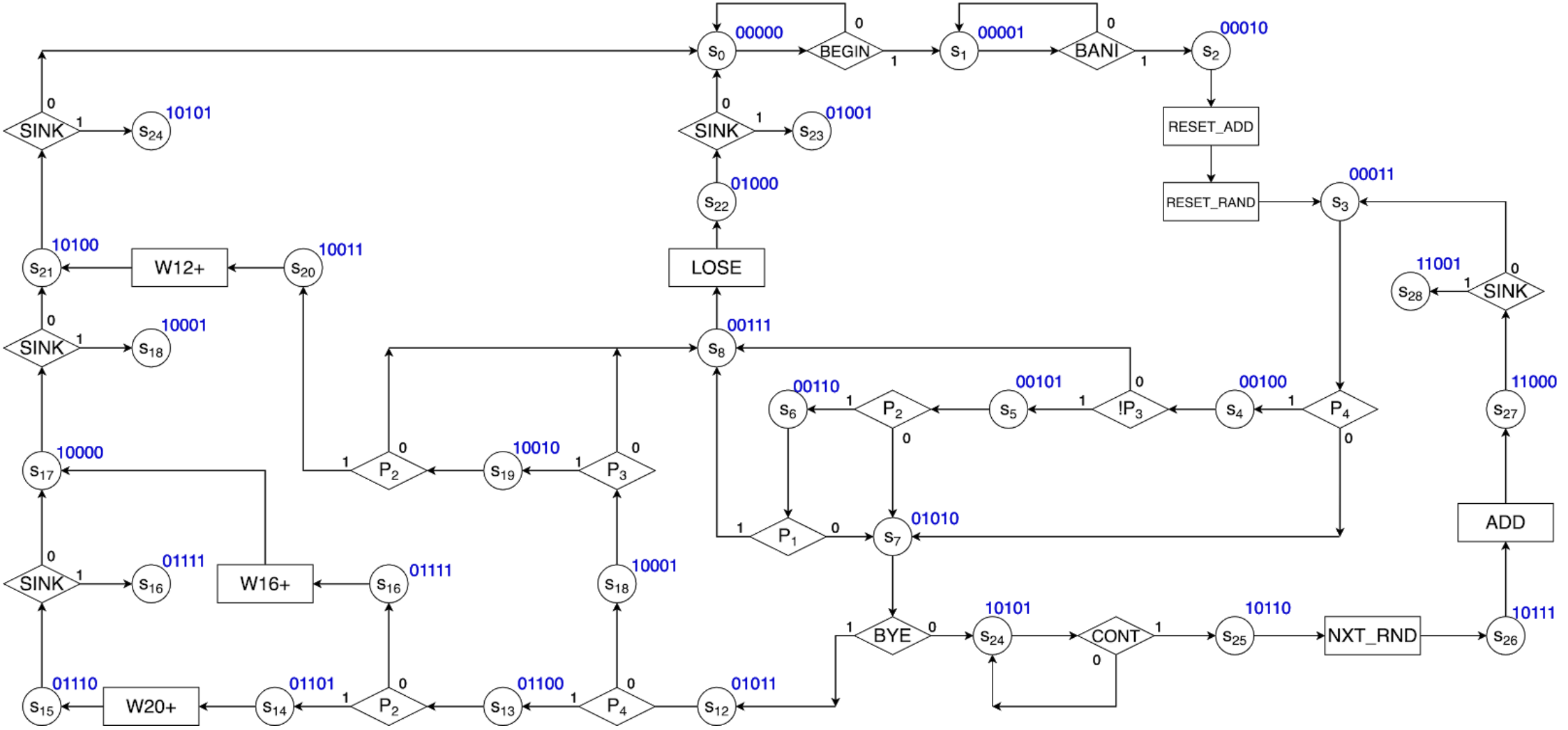
Aparatul se va afla într-un mod pasiv până la trecerea prin semnalele **BEGIN** și **PAY**. Imediat după ce trecem de cele două semnale vom genera ieșirile **RESET_ADD**, **RESET_RAND**. După ce jocul va începe efectiv, se va intra într-o buclă de joc, care constă în următorii pași:

- Comparația punctajului **P** cu 21 în “submodulul” CMP-21.
    - Dacă **P** este prea mare, se va ieși din buclă și se va intra într-o ramură în care se generează semnalul **LOSE** și ne întoarcem în starea pasivă.
    - Dacă **P** este mai mic sau egal cu 21, continuăm să ne plimbăm prin buclă.
- Așteptăm o acțiune din partea jucătorului: aceasta poate fi ori **CONT**, ori **BYE**.
    - Dacă acțiunea este **BYE**, vom intra în partea de “plată” a jucătorului (ieșind efectiv din buclă), pe care o vom detalia mai jos.
    - Dacă acțiunea este **CONT**, înseamnă că jucătorul dorește să mai primească o carte și continuăm în buclă.
- Se generează un semnal **NEXT_RND** trimis către submodulul **RNG**, care ne va oferi un număr aleator nou din intervalul [3, 10], corespunzător valorii cărții date jucătorului. După ce stăm un ciclu, ne putem aștepta ca valorile biților din CBB-urile corespunzătoare **R**-urilor să fie actualizate, astfel încât putem trece la următorul pas.
- Se generează un semnal **ADD**, care va actualiza valorile biților din CBB-urile corespunzătoare P-urilor, cu respect față de valorile din CBB-urile corespunzătoare **R**-urilor. Acțiunea are loc în submodulul de adunare.
- După ce mai stăm un ciclu, vom termina bucla curentă și ne întoarcem la primul pas, cel de comparație.

Partea de plată a jucătorului se va desfășura astfel:

- Știm sigur că dacă am intrat în partea de plată, punctajul este în intervalul [0, 21].
- Dacă $\mathbf{P_4}$ este 1, atunci punctajul este în intervalul [16, 21]. Cum reprezentarea lui 21 în bază 2 este 10101, rezulta ca daca $\mathbf{P_4}$ este 1, atunci $\mathbf{P_3}$ este 0.
    - Dacă $\mathbf{P_2}$ este 1, atunci punctajul este în intervalul [20, 21]. Se genereaza ieșirea $\mathbf{WIN_{20}+}$ și se revine în starea inițială, mod pasiv.
    - Dacă $\mathbf{P_2}$ este 0, atunci punctajul este în intervalul [16, 19]. Se genereaza ieșirea $\mathbf{WIN_{16}+}$ și se revine în starea inițială.
- Dacă $\mathbf{P_4}$ este 0, atunci punctajul este în intervalul [0, 15].
    - Dacă $\mathbf{P_3}$ este 0, atunci punctajul este în intervalul [0, 7]. Se genereaza ieșirea **LOSE** și se revine în starea inițială.
    - Dacă $\mathbf{P_3}$ este 1, atunci punctajul este în intervalul [8, 15].
        - Dacă $\mathbf{P_2}$ este 1, atunci punctajul este în intervalul [12, 15], se genereaza ieșirea $\mathbf{WIN_{12}+}$ și se revine în starea inițială.
        - Dacă $\mathbf{P_2}$ este 1, atunci punctajul este în intervalul [8, 11], se genereaza ieșirea **LOSE** și se revine în starea inițială.

Organigrama aparatului (https://cutt.ly/pl-organigrama-blackjack)



Submodulul RNG (Randomizator)

Pentru acest proiect, am avut nevoie de un generator de numere aleatoare din intervalul [3, 10]. Deoarece intervalul este de lungime relativ mică, am construit un generator de perioadă 16. Unul dintre cele mai simple generatoare de implementat este **generatorul liniar congruenţial**. Acesta se foloseşte de următoarele valori:

- $seed$, sau valoarea veche a generatorului;
- *a*, un multiplicator, număr natural, care este de obicei mare;
- *b*, un număr natural, de obicei cu câteva ordine de mărime mai mic decât *a*;
- mod, un număr mai mare decat *a* sau *b*.

Valoarea nouă a lui seed se obţine după formula: $(a \cdot seed + b) \,\%\, mod$.

De obicei, vrem $gcd(a, mod) = 1$ şi $gcd(b, mod) = 1$. Putem să luăm mod prim pentru a garanta acest lucru. În practică, *b* este foarte mic, iar mod este luat ca o putere mare a lui 2, astfel transformând operaţia costisitoare de modulo într-o shiftare pe biţi.

Codul în Python care generează primele 16 numere din intervalul [3,10] din secvenţa cu $a = 25214903917$, $b = 13$, $mod = 2^{48} + 17$ este:

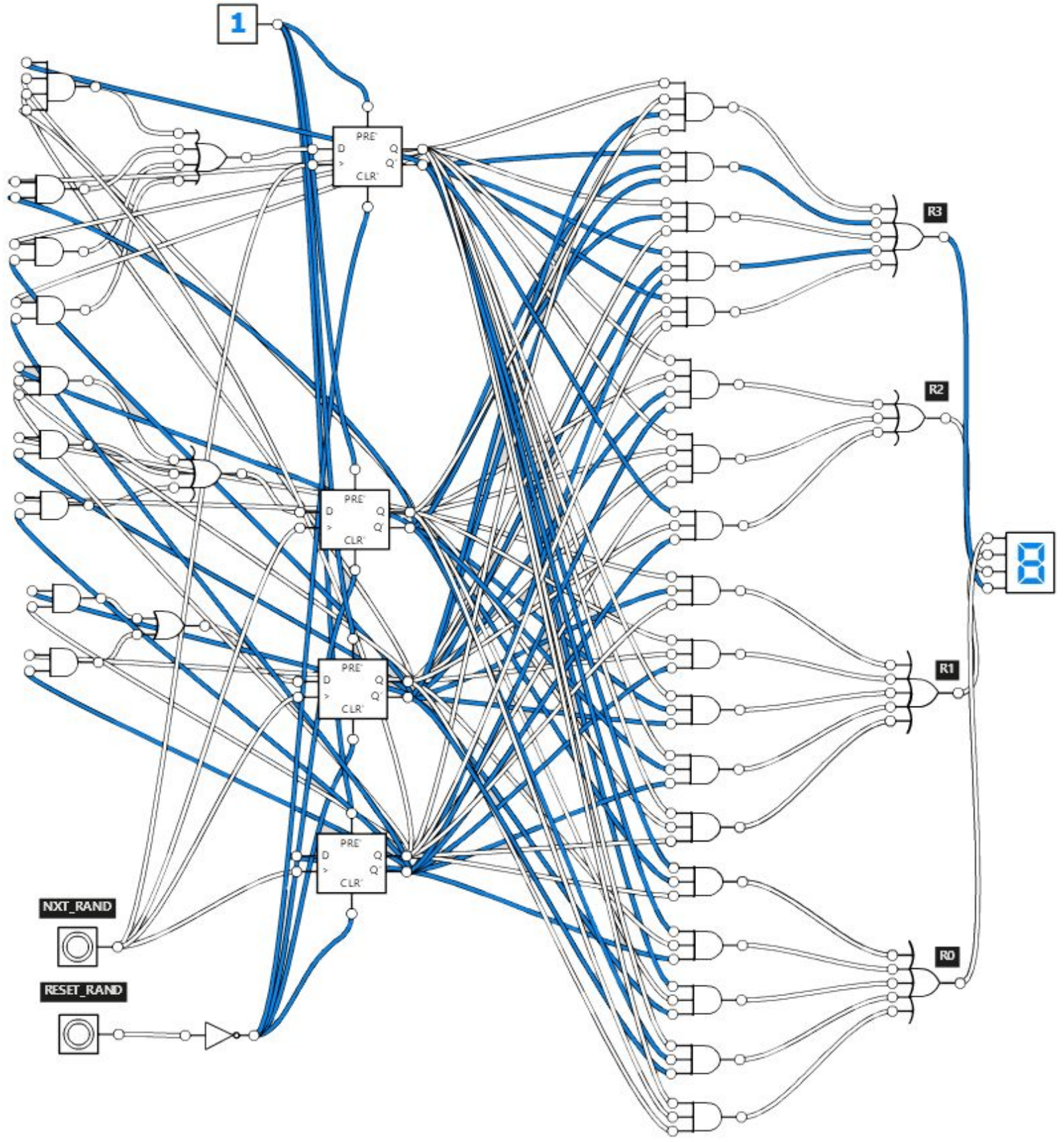
```
import math

def lcg():
    a = 25214903917
    b = 13
    mod = (1<<48) + 17
    seed = 0
    while True:
        seed = (seed * a + b) % mod
        yield seed

i = 1
lo, hi = 3, 10
stop_at = 21
print(lo, hi, "stop_at =", stop_at)
for seed in lcg():
    print(seed % (hi - lo + 1) + lo, end = ' ')
    i += 1
    if (i > 16):
        break

#8, 9, 5, 9, 9, 3, 7, 8, 3, 3, 10, 8, 4, 10, 3, 7
```

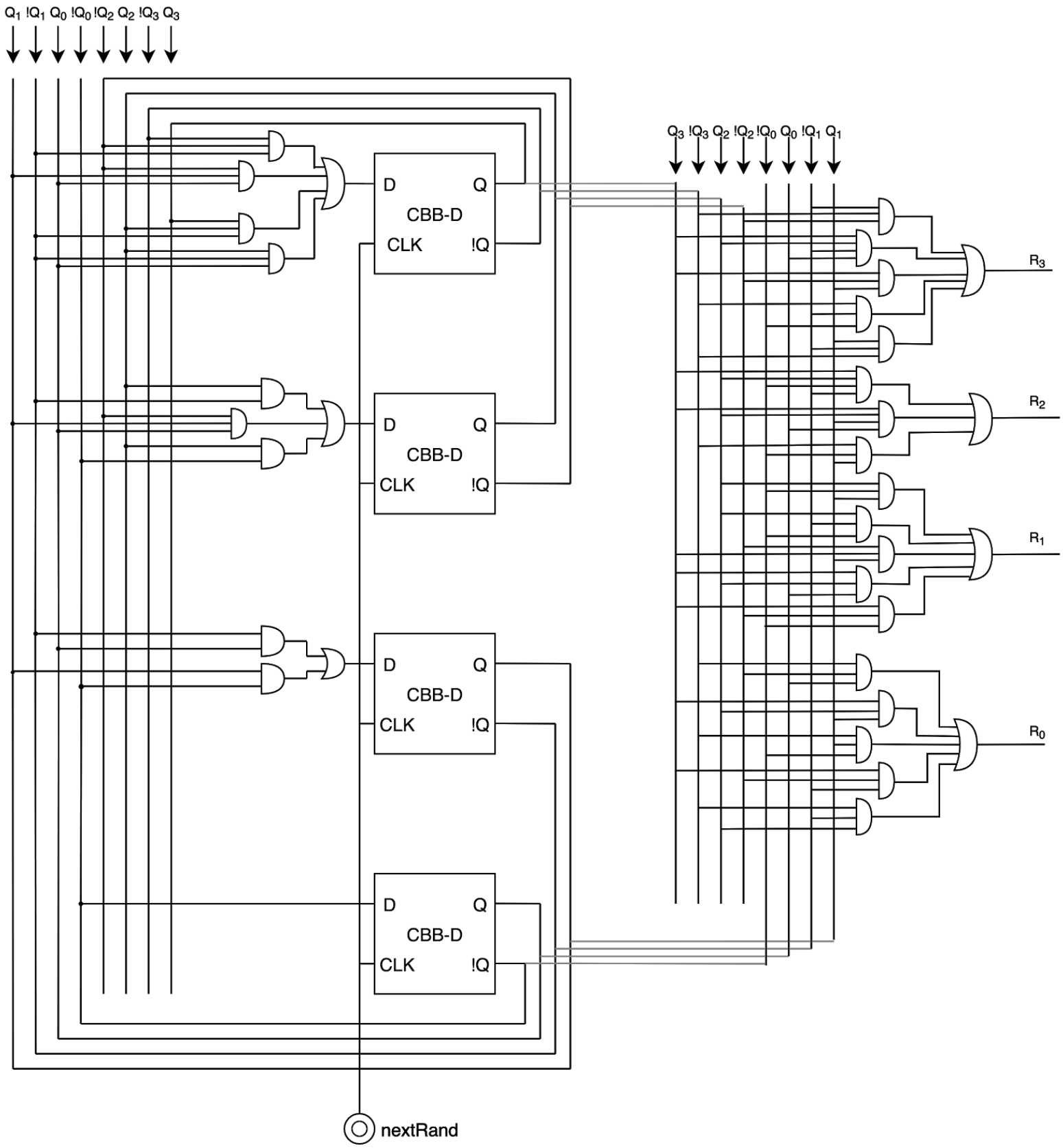
Primele 16 numere ale secvenţei sunt: 8, 9, 5, 9, 9, 3, 7, 8, 3, 3, 10, 8, 4, 10, 3, 7.

Am ales să implementez modulul “Random” cu 4 CBB-uri de tip D. Mai jos se găseşte tabelul tranziţiilor. Ultima coloană reprezintă ieşirea dorită în starea de pe prima coloană. Coloanele R_3,R_2, R_1,R_0 reprezintă ieşirile propriu-zise, atribuite numărului aleator dorit.

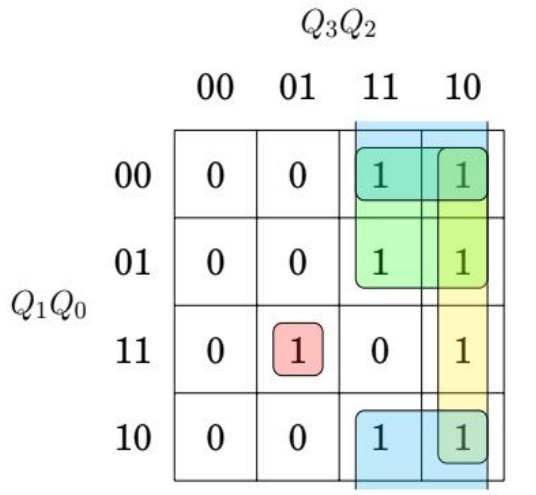
	Q_3^t	Q_2^t	Q_1^t	Q_0^t	Q_3^{t+1}	Q_2^{t+1}	Q_1^{t+1}	Q_0^{t+1}	D_3	D_2	D_1	D_0	R_3	R_2	R_1	R_0	R
s_0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	8
s_1	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	9
s_2	0	0	1	0	0	0	1	1	0	0	1	1	0	1	0	1	5
s_3	0	0	1	1	0	1	0	0	0	1	0	0	1	0	0	1	9
s_4	0	1	0	0	0	1	0	1	0	1	0	1	1	0	0	1	9
s_5	0	1	0	1	0	1	1	0	0	1	1	0	0	0	1	1	3
s_6	0	1	1	0	0	1	1	1	0	1	1	1	0	1	1	1	7
s_7	0	1	1	1	1	0	0	0	1	0	0	0	1	0	0	0	8
s_8	1	0	0	0	1	0	0	1	1	0	0	1	0	0	1	1	3
s_9	1	0	0	1	1	0	1	0	1	0	1	0	0	0	1	1	3
s_{10}	1	0	1	0	1	0	1	1	1	0	1	1	1	0	1	0	10
s_{11}	1	0	1	1	1	1	0	0	1	1	0	0	1	0	0	0	8
s_{12}	1	1	0	0	1	1	0	1	1	1	0	1	0	1	0	0	4
s_{13}	1	1	0	1	1	1	1	0	1	1	1	0	1	0	1	0	10
s_{14}	1	1	1	0	1	1	1	1	1	1	1	1	0	0	1	1	3
s_{15}	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	7

Implementare în logicly: (fișierul se poate descărca de aici: https://cutt.ly/nextrand-pl2-logicly.)

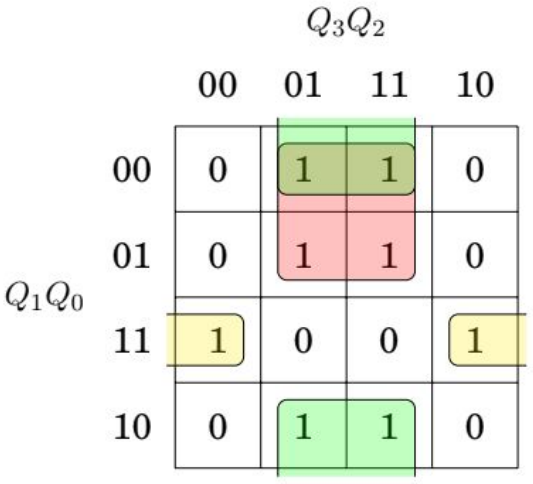



Q1 !Q1 Q0 !Q0 !Q2 Q2 !Q3 Q3
D
Q
CBB-D
CLK
!Q
Q3 !Q3 Q2 !Q2 !Q0 Q0 !Q1 Q1
R3
R2
R1
R0
nextRand

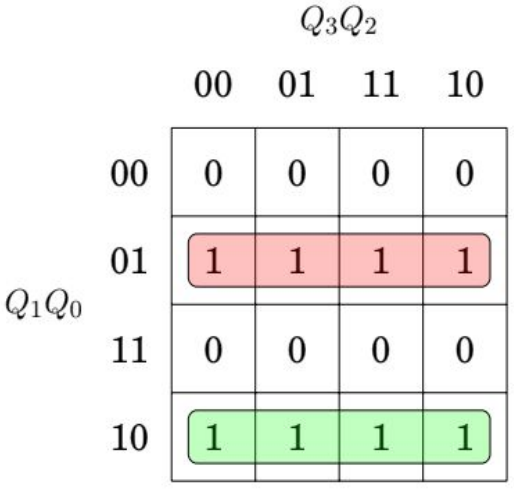
Diagramele Karnaugh pentru stările următoare



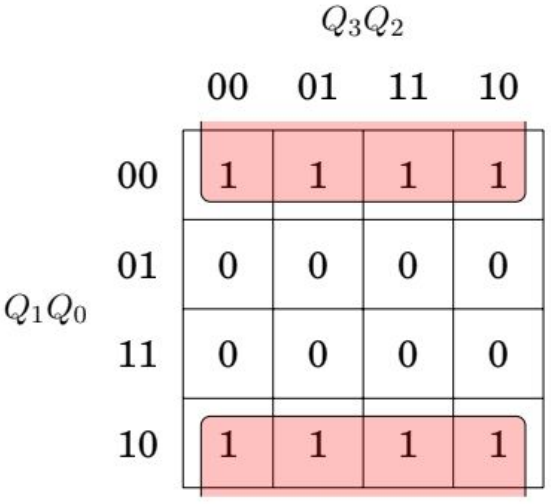
$$Q_3^{t+1} = D_3^t = \overline{Q}_3Q_2Q_1Q_0 + Q_3\overline{Q}_2 + Q_3\overline{Q}_1 + Q_3\overline{Q}_0$$



$$Q_2^{t+1} = D_2^t = \overline{Q}_2Q_1Q_0 + Q_2\overline{Q}_1 + Q_2\overline{Q}_0$$

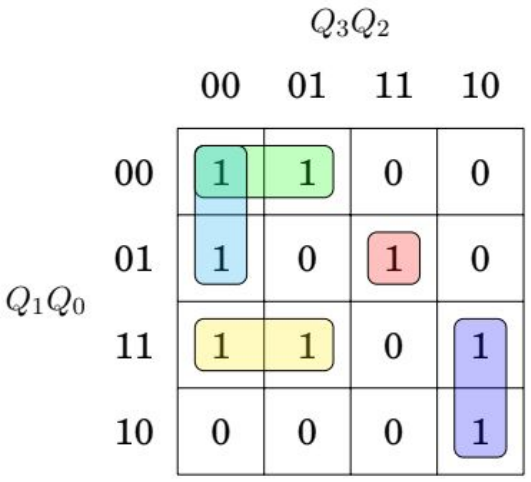


$$Q_1^{t+1} = D_1^t = \overline{Q}_1Q_0 + Q_1\overline{Q}_0(Q_1XORQ_0)$$

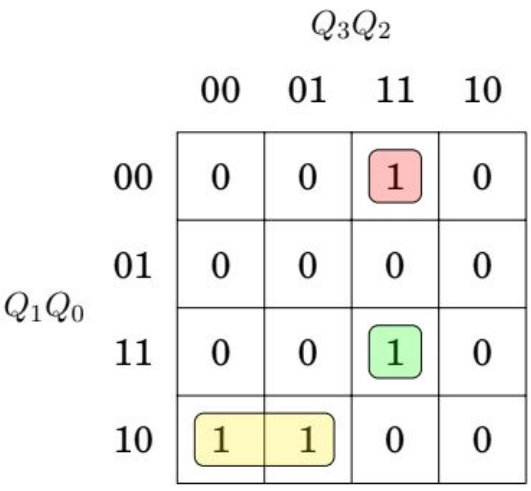


$$Q_0^{t+1} = D_0^t = \overline{Q}_0$$

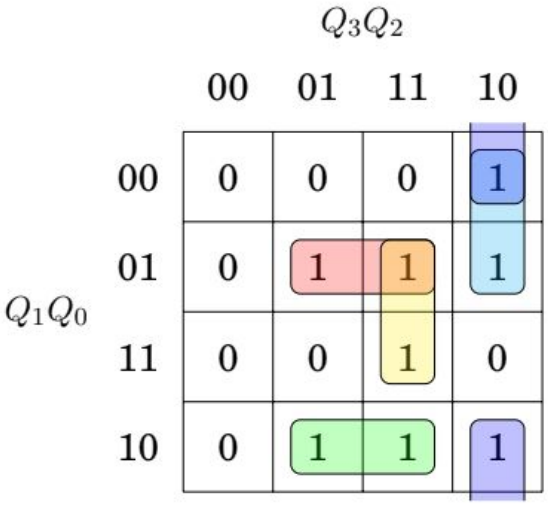
Diagramele Karnaugh pentru ieşiri



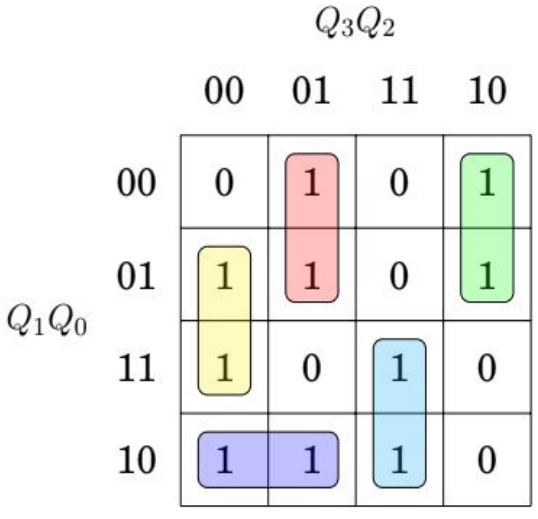
$$R_3 = \overline{Q}_3\overline{Q}_2\overline{Q}_1 + Q_3Q_2\overline{Q}_1Q_0 + Q_3\overline{Q}_2Q_1 + \overline{Q}_3\overline{Q}_1\overline{Q}_0 + \overline{Q}_3Q_1Q_0$$



$$R_2 = Q_3Q_2\overline{Q}_1\overline{Q}_0 + Q_3Q_2Q_1Q_0 + \overline{Q}_3Q_1\overline{Q}_0$$



$$R_1 = Q_2\overline{Q}_1Q_0 + Q_2Q_1\overline{Q}_0 + Q_3\overline{Q}_2\overline{Q}_1 + Q_3\overline{Q}_2\overline{Q}_0 + Q_3Q_2Q_0$$



$$R_0 = \overline{Q}_3\overline{Q}_2Q_0 + \overline{Q}_3Q_1\overline{Q}_0 + \overline{Q}_3Q_2\overline{Q}_1 + Q_3\overline{Q}_2\overline{Q}_1 + Q_3Q_2Q_1$$

Submodulul ADD (Adunare)

Pentru a ţine minte scorul actual al jucătorului, trebuie să avem o variabilă anume (denumită în acest proiect "P", de la "Punctaj"; valoarea ei se explicitează prin biţii P_4, P_3, P_2, P_1, P_0, de la cel mai semnificativ la cel mai puţin semnificativ) la care vom aduna în mod constant numerele de pe cărţile primite, sau în cazul nostru numerele generate de Randomizatorul din Anexa 1.

Informal, vrem să implementăm operaţia "$+=$" regăsită în mai multe limbaje de programare, astfel încât operaţia să arate astfel:

$$P += R$$

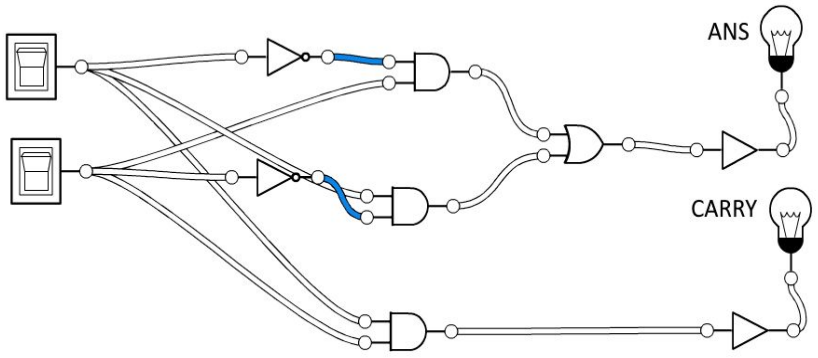
Trebuie să avem grijă să nu depăşim la adunare formatul de 5 biţi al punctajului. Deoarece cel mai mare punctaj valid este 21, iar cel mai mare număr valid care poate fi reprezentat pe 5 biţi este 31, putem aduna maxim 10 pentru a ne asigura că nu vom da "overflow". Pentru a nu include cărţi cu valoare prea mică, am restrâns intervalul valorilor la $[3, 10]$.

Mai precis, trebuie să modificăm un adunător pe 5 biţi astfel încât rezultatul să se atribuie (şi să se păstreze) în primul termen al adunării.

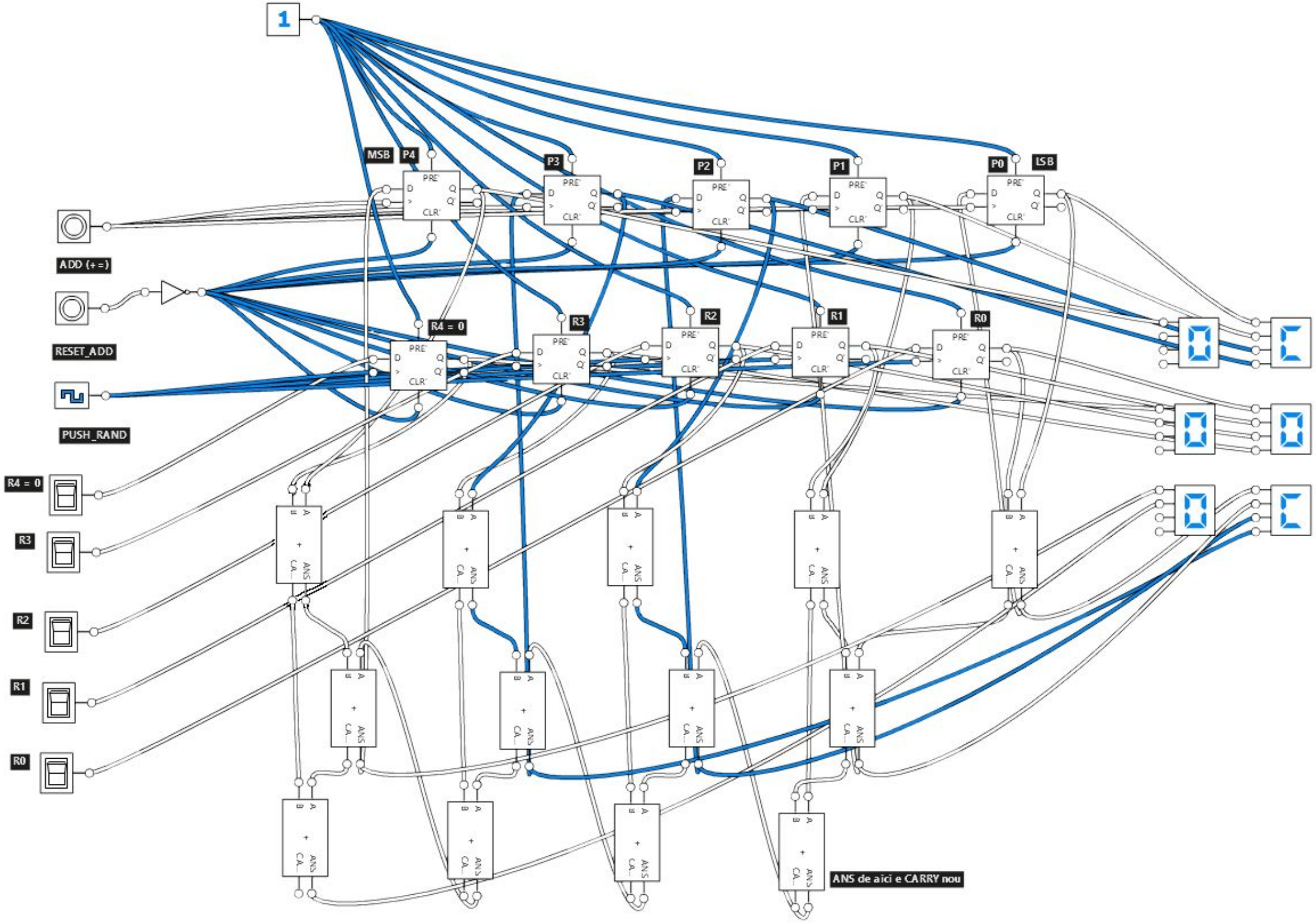
P este întotdeauna primul termen al adunării;
R este întotdeauna al doilea termen al adunării;
$P + R$, rezultatul, va fi atribuit primului termen din adunare.

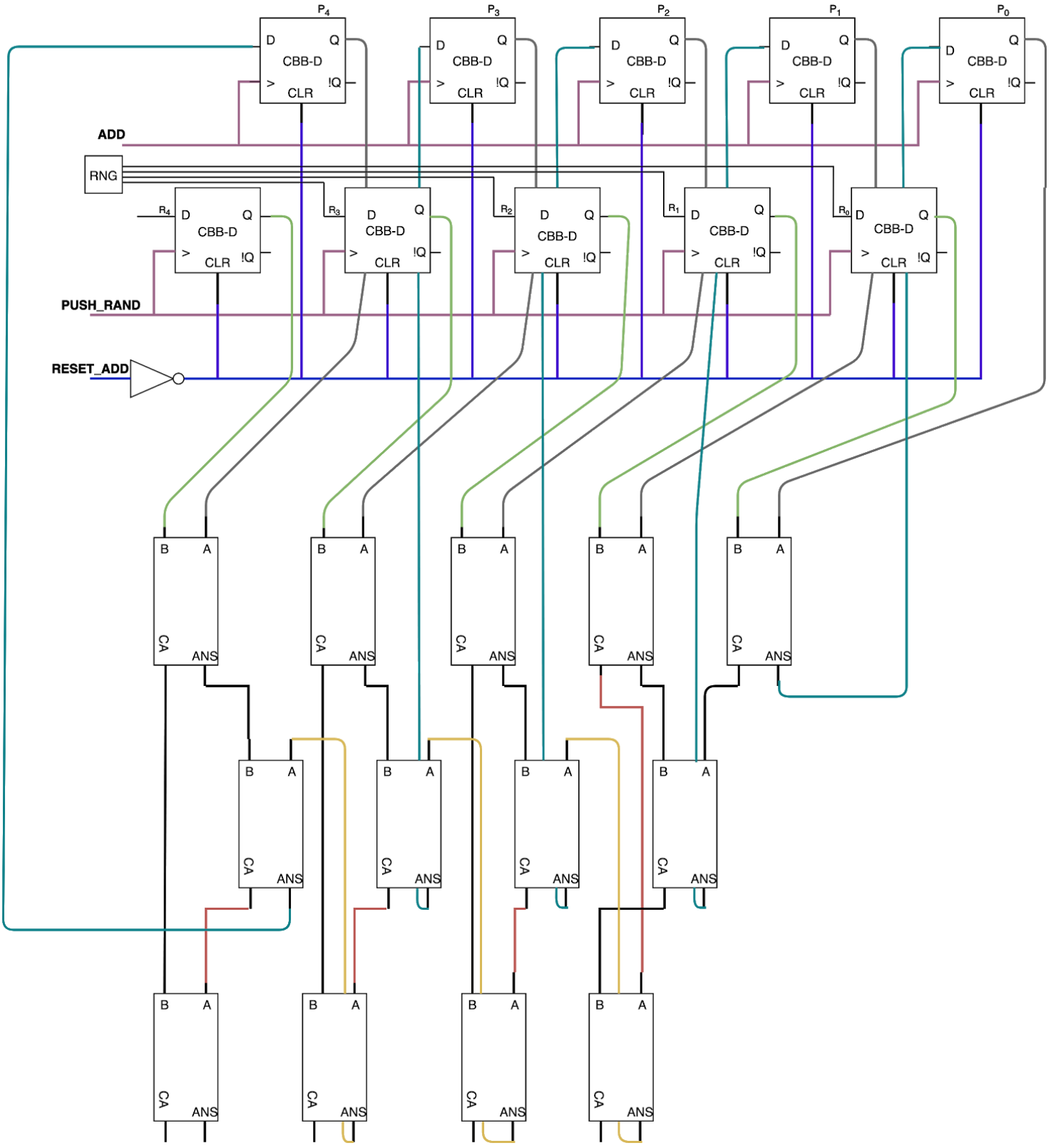
- Valorile lui P şi R se păstrează pe 5 CBB-uri de tip D fiecare. Cum R_4 nu este generat niciodată, valoarea CBB-ului său va fi întotdeauna 0.
- CBB-urile asignate lui Rsunt guvernate de un ceas sincron, numit PUSH_RAND. În urma trimiterii unui semnal NXT_RAND, corespunzător dorinţei primirii unei noi cărţi, trebuie să înregistrăm imediat valorile biţilor primite din submodulul "Randomizer", pregătindu-ne astfel pentru adunare.
- Când dorim să facem adunarea, trimitem un semnal ADD, care ia valorile de pe CBB-urile lui P şi face operaţia "$+=$", cu respect faţă de valorile de pe CBB-urile lui R.
- După operaţia de început a jocului ($BANI = 1$) trebuie trimis un semnal pe ieşirea RESET_ADD din submodulul ADD.

În implementarea submodulului de adunare am folosit un adunător pe 1 bit, implementat astfel:



Implementarea submodulului ADD se găseşte mai jos. (fişier: https://cutt.ly/add-logicly.)


1
MSB
P4
P3
P2
P1
P0
LSB
ADD (+=)
RESET_ADD
PUSH_RAND
R4 = 0
R3
R2
R1
R0
R4 = 0
R3
R2
R1
R0
PRE
D
CLR
Q
ANS de aici e CARRY nou


P4
P3
P2
P1
P0
D
Q
CBB-D
>
CLR
!Q
ADD
RNG
R4
R3
R2
R1
R0
PUSH_RAND
RESET_ADD
B
A
CA
ANS

Calculul lungimii microinstrucţiunii pentru format variabil

Codificare intrări	Denumire intrare
0000	SINK
0001	BEGIN
0010	BANI
0011	P_4
0100	P_3
0101	$!P_3$
0110	P_2
0111	P_1
1000	BYE
1001	CONT

Indexare ieşiri	Denumire ieşire
O_0	RESET_ADD
O_1	RESET_RAND
O_2	ADD
O_3	NXT_RAND
O_4	$W_{12}+$
O_5	$W_{16}+$
O_6	$W_{20}+$
O_7	LOSE

- Avem 10 intrări distincte, deci avem nevoie de 4 biţi pe intrare ca să le reprezentăm.
- Avem 8 ieşiri distincte, deci avem nevoie de 8 biţi pentru a le reprezenta.
- În organigramă cea mai mare adresă folosită este 25(11001), deci avem nevoie de 5 biţi pentru a le reprezenta.

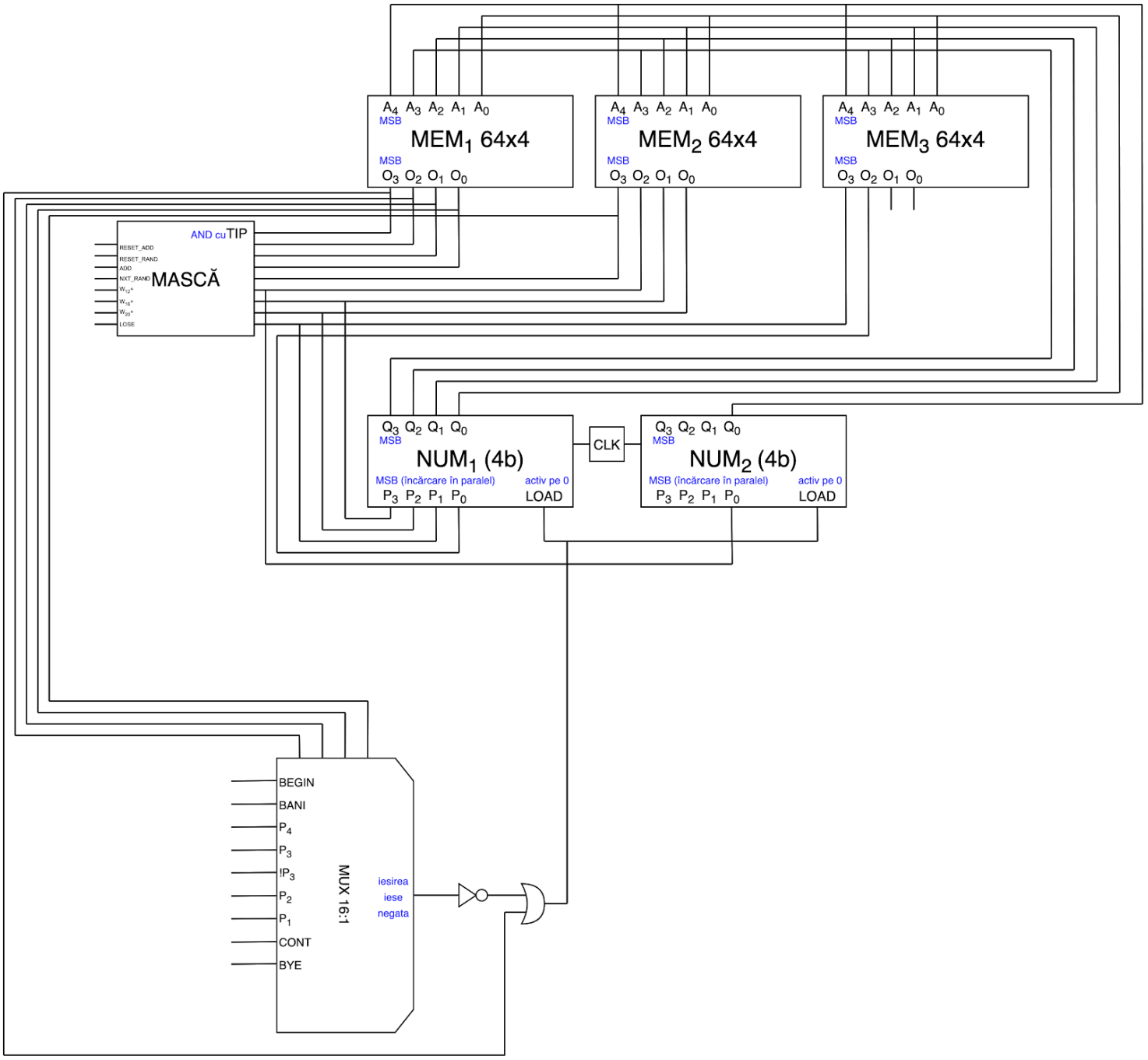
- Pentru adresele de tip 1 lungimea microinstrucţiunii este:
  $$l_{\mu i1} = 1 + NROUT = 1 + 8 = 9$$
- Pentru adresele de tip 0 lungimea microinstrucţiunii este:
  $$l_{\mu i0} = 1 + LGINPUT + LGSTARE = 1 + 4 + 5 = 10$$

- Aşadar, lungimea microinstrucţiunii este:
  $$l_{\mu i} = \max(l_{\mu i0}, l_{\mu i1}) = \max(9,\ 10) = 10$$

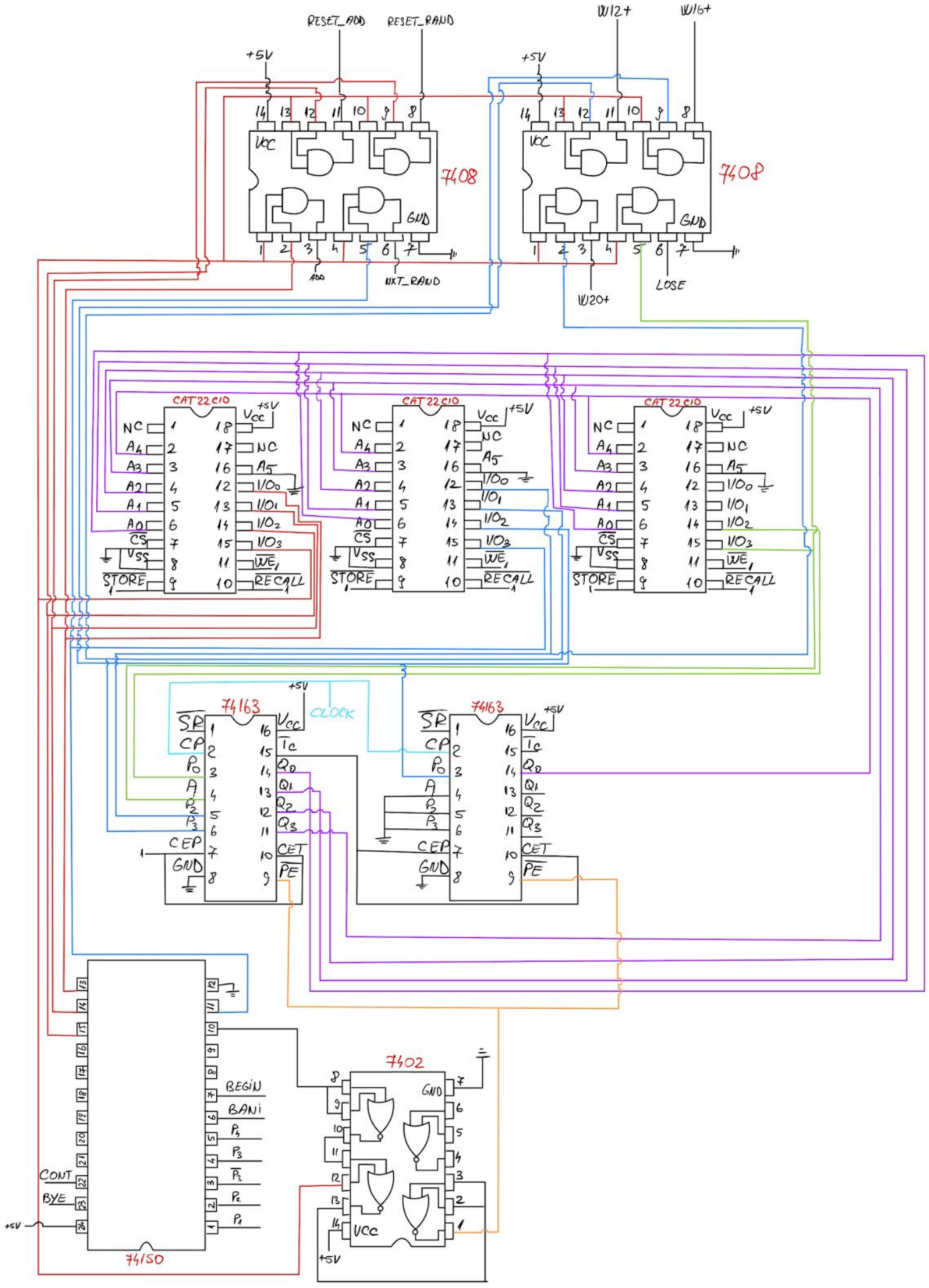
Completarea conținutului memoriei de microprogram

ind. stare	TIP	O_7/IN_A	O_6/IN_B	O_5/IN_C	O_4/IN_D	O_3/ST_A	O_2/ST_B	O_1/ST_C	O_0/ST_D	ST_E
00000	0	0	0	0	1	0	0	0	0	0
00001	0	0	0	1	0	0	0	0	0	1
00010	1	0	0	0	0	0	0	1	1	-
00011	0	0	0	1	1	0	1	0	1	0
00100	0	0	1	0	1	0	0	1	1	1
00101	0	0	1	1	0	0	1	0	1	0
00110	0	0	1	1	1	0	1	0	1	0
00111	1	1	0	0	0	0	0	0	0	-
01000	0	0	0	0	0	0	0	0	0	0
01001	-	-	-	-	-	-	-	-	-	-
01010	0	1	0	0	0	1	0	1	0	1
01011	0	0	0	1	1	1	0	0	0	1
01100	0	0	1	1	0	0	1	1	1	1
01101	1	0	1	0	0	0	0	0	0	-
01110	0	0	0	0	0	1	0	0	0	0
01111	1	0	0	1	0	0	0	0	0	-
10000	0	0	0	0	0	1	0	1	0	0
10001	0	0	1	0	0	0	0	1	1	1
10010	0	0	1	1	0	0	0	1	1	1
10011	1	0	0	0	1	0	0	0	0	-
10100	0	0	0	0	0	0	0	0	0	0
10101	0	1	0	0	1	1	0	1	0	1
10110	1	0	0	0	0	1	0	0	0	-
10111	1	0	0	0	0	0	1	0	0	-
11000	0	0	0	0	0	0	0	0	1	1
11001	-	-	-	-	-	-	-	-	-	-

Proiectarea schemei de comandă microprogramate



Proiectarea cablajului



Referințe

În realizarea proiectului am folosit următoarele resurse pentru conceperea diagramelor:

- https://www.diagrams.net/
- https://logic.ly/
- GoodNotes

Link-ul pentru prezentarea PowerPoint

- https://docs.google.com/presentation/d/1raTuTdT03QW8f0pWgR9YQaJNCj38QYKSgkDH-AsjmTA/edit?usp=sharing