

UNIVERSITATEA POLITEHNICĂ TIMIȘOARA  
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DISCIPLINA  
PROIECTAREA CU MICROPROCESOARE

# **MICROSISTEM CU MICROPROCESORUL 8086**

AUTOR

CONSTANTIN CĂTĂLINA-VIVIANA

GRUPA 3C.01.2

PROF. COORDONATOR

PETRUC SEBASTIAN-IOAN

ANUL UNIVERSITAR

2024 – 2025

## 1. TEMA PROIECTULUI

Proiectul consistă în proiectarea unui microsistem cu următoarea structură:

- unitate centrală cu microprocesorul 8086
- 256KB memorie EPROM, utilizând circuite 27C2048
- 64KB memorie SRAM, utilizând circuite 62512
- interfață serială, cu circuitul 8251, plasată în zona 0AF0H – 0AF2H sau 0BF0H – 0BF2H, în funcție de poziția microcomutatorului S1
- interfață paralelă, cu circuitul 8255, plasată în zona 0D70H – 0D76H sau 0C70H – 0C76H, în funcție de poziția microcomutatorului S2
- o minitastatură cu 12 contacte
- 10 led-uri
- un modul de afișare cu 7 segmente, cu 6 ranguri

Toate programele în limbaj de ansamblare sunt concepute sub formă de subrutine. Programele incluse în acest proiect sunt:

- rutinele de programare ale circuitelor 8251 și 8255
- rutinele de emisie / recepție caracter pe interfața serială
- rutina de emisie caracter pe interfața paralelă
- rutina de scanare a minitastaturii
- rutina de aprindere / stingere a unui led
- rutina de afișare a unui caracter hexa pe un rang cu segmente

## 2. DESCRIEREA HARDWARE

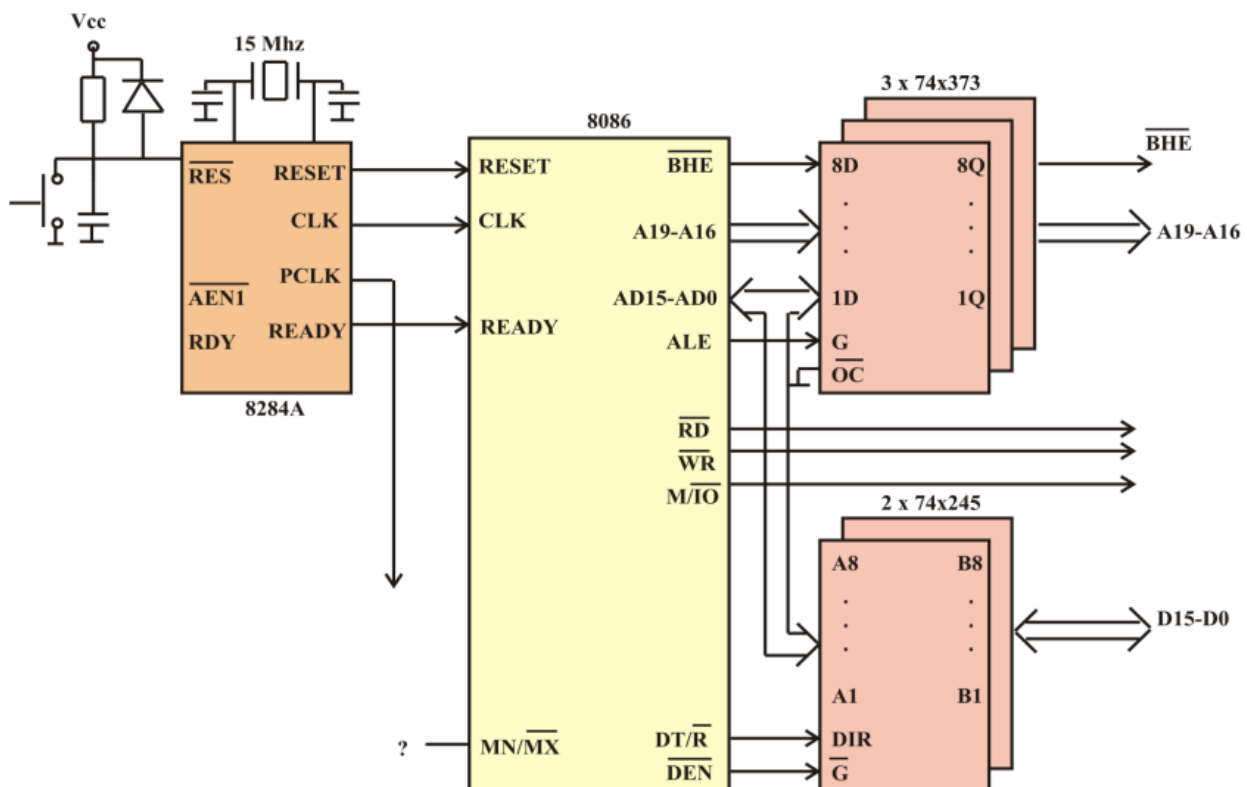
Schema hardware a microsistemului cu microprocesor 8086 este formată din: unitatea centrală cu microprocesor 8086, circuite de memorie, decodicatorul de porturi, interfețe seriale și paralele, minitastatură, modul de afișare cu 7 segmente și led-uri.

### 2.1 Unitatea centrală a microsistemului

Unitatea centrală a unui microsistem, adesea denumită și Unitatea Centrală de Procesare (CPU), este componenta principală responsabilă pentru procesarea instrucțiunilor și coordonarea activităților în cadrul sistemului.

În cazul nostru, aceasta este alcătuită din:

- Microprocesorul 8086
- Generatorul de tact 8284A
- Circuite pentru amplificarea și demultiplexarea magistrelor de adrese și date



#### 2.1.1 Microprocesorul 8086

Microprocesorul 8086 este primul procesor care are registrele interne și magistrala de date externă pe 16 biți, iar astăzi este cel mai răspândit microprocesor. Acesta are posibilitatea de a adresa direct 1 MB de memorie, unde, datorită frecvenței ridicate a tactului de 5 MHz, se permite aducerea în avans a instrucțiunilor.

## Configurația pinilor:

**RESET** (System Reset) – intrare pentru inițializarea microprocesorului

**CLK** (System Clock) – intrare de tact cu frecvența uzuală de 5 MHz și factor de umplere de 1/3

**READY** (Wait State Control) – intrare pentru sincronizarea cu circuitele de memorie și porturile mai lente

**MN /  $\sim$ MX** – intrare care indică modul de lucru al procesorului

**DT /  $\sim$ R** (Data Transmit / Receive) – ieșire cu trei stări, care indică sensul transferului pe magistrala de date: 1 indică transmisie de date, 0 indică recepție de date

**$\sim$ DEN** (Data Enable) – ieșire cu trei stări, care validează transferul de date pe magistrală

**M /  $\sim$ IO** (Memory / Input – Output Control) – dacă are valoarea 1 înseamnă că se execută un ciclu de acces de memorie, iar dacă are valoarea 0 înseamnă că se execută un ciclu de transfer cu porturile de intrare / ieșire

**$\sim$ WR** (Write Control) – ieșire cu trei stări, activă atunci când microprocesorul execută un ciclu de scriere sau de ieșire

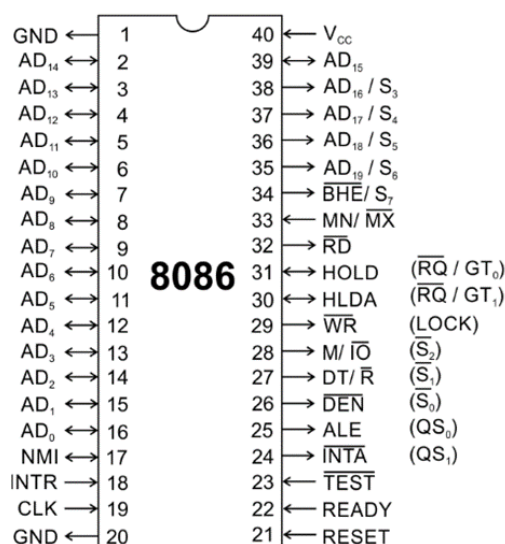
**$\sim$ RD** (Read Control) – ieșire cu trei stări, activă atunci când microprocesorul execută un ciclu de citire sau de intrare

**ALE** (Address Latch Enable) – ieșire care se activează atunci când pe magistrala multiplexată de adrese / date sunt active adresele; se poate folosi pentru demultiplexarea magistralei prin încărcarea adreselor în registre

**AD15 – AD0** – magistrala multiplexată de adrese / date cu trei stări

**A19 – A16** – rangurile 19 – 16 din magistrala de adrese

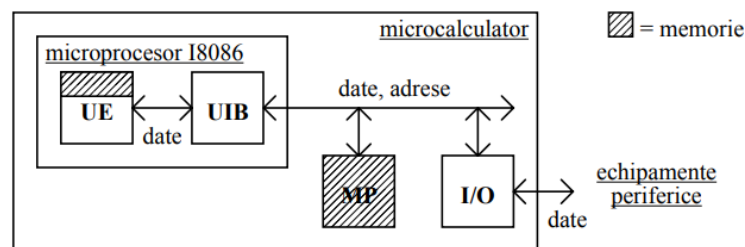
**$\sim$ BHE** (Bus High Enable) – ieșire care indică dacă are sau nu loc un transfer pe jumătatea superioară a magistralei de date



## Structura internă:

Microprocesorul 8086 cuprinde două unități funcționale, care lucrează asincron și independent una față de cealaltă:

- **UE** (Unitatea de Execuție) – execută instrucțiunile
- **UIB** (Unitatea de Interfațare cu Magistrale) – aduce instrucțiunile din memorie și transferă operanzii între EU și exteriorul microprocesorului



Microprocesorul poate lucra în modurile:

- **MINIM** – pentru aplicații relativ simple, în care microprocesorul generează el însuși semnalele necesare transferurilor cu memoria și cu porturile de intrare / ieșire
- **MAXIM** – pentru aplicații complexe, inclusiv sisteme multiprocesor, în care semnalele de comandă pentru memorii și porturi sunt generate de un controler de magistrală

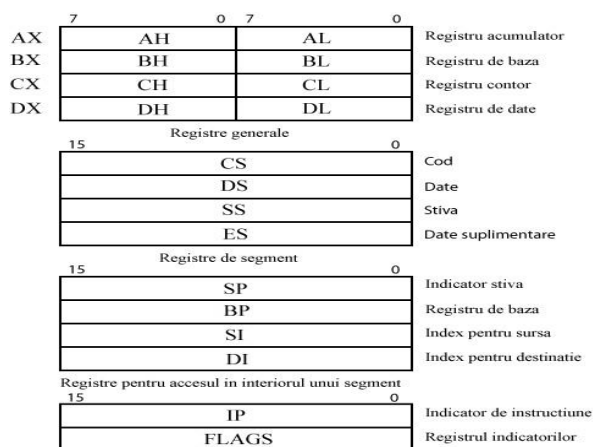
Trecerea dintr-un mod în altul se face prin hardware, prin terminalul MN / MX la care, prin 1 logic, se cere modul minim, iar prin 0 logic se cere modul maxim.

## Setul de registre:

Registrele sunt specializate pe funcții, ele fiind grupate în 5 categorii:

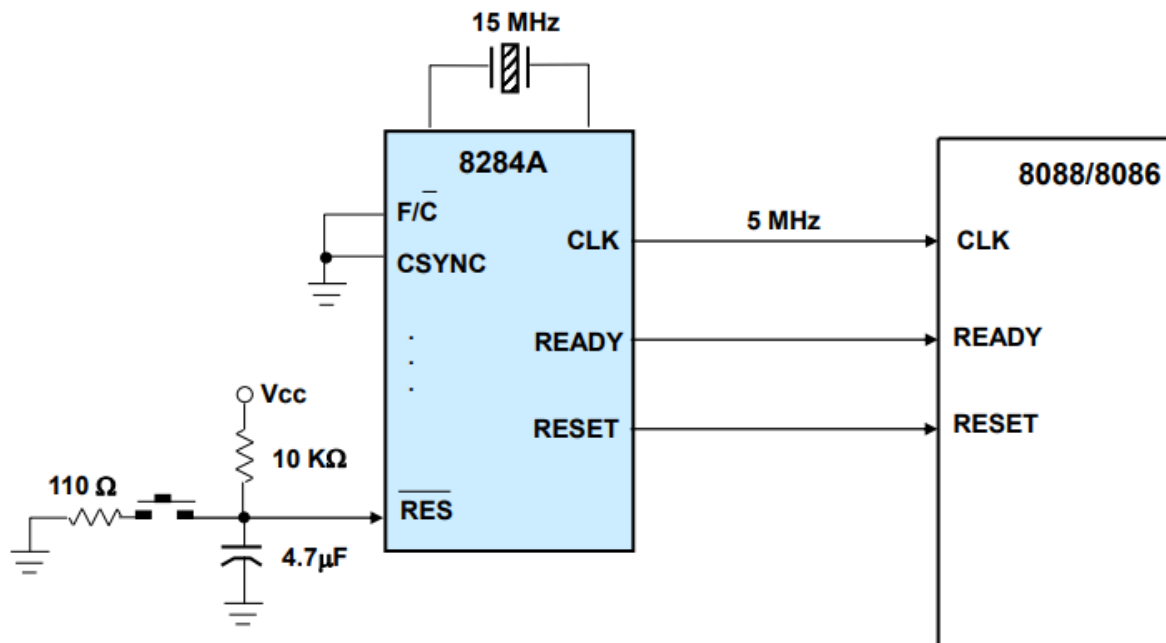
- registre de date (AX, BX, CX, DX)
- registre index, pentru accesul în interiorul unui segment (SP, BP, DI, SI)
- registre de segment (CS, DS, SS, ES)
- registru indicator de adresă (IP)
- registru de stare (F)

Primele două categorii alcătuiesc registrele de uz general. Utilizatorul are acces la toate registrele, dimensiunea lor fiind de 16 biți, egală cu dimensiunea magistralei de date.

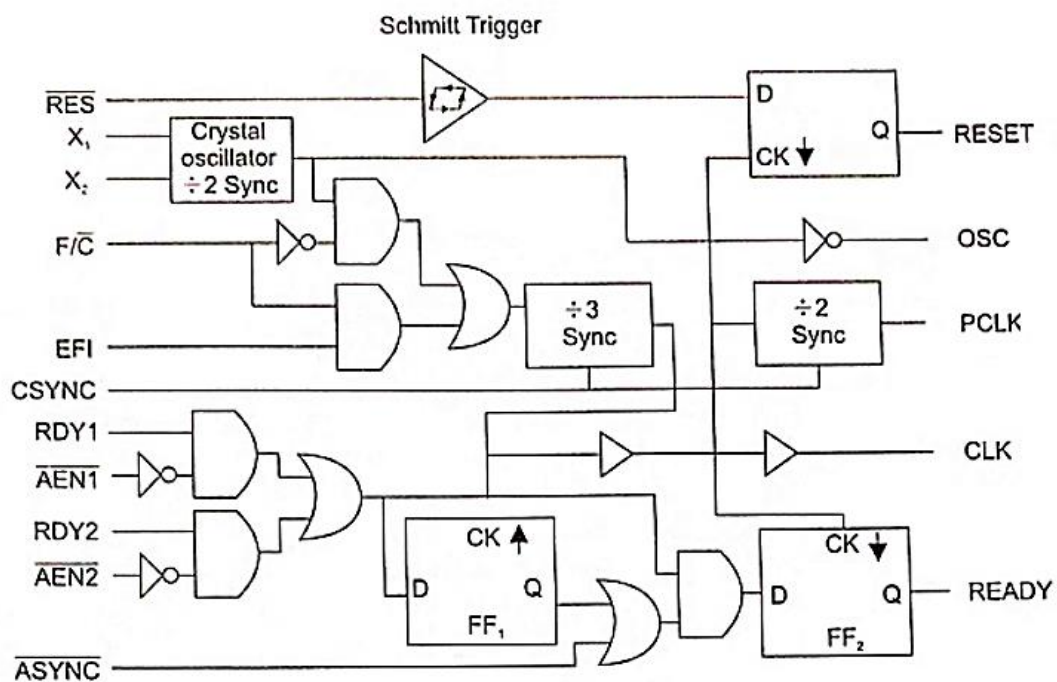


### 2.1.2 Generatorul de tact 8284A

Circuitul 8284A este folosit pentru a genera semnalul de tact CLK către microprocesor și către circuitele specializate ale interfeței seriale, dar și pentru a sincroniza cu semnalul de tact și genera semnalele READY și RESET.



Logica internă a generatorului de tact 8284A este reprezentată în figura de mai jos:



### Descrierea pinilor terminali ai generatorului de tact 8284A:

**X<sub>1</sub> și X<sub>2</sub>** – pini de intrare conectați la un cristal de cuarț extern, utilizat ca sursă de frecvență pentru generatorul de tact; frecvența cristalului va fi aproximativ de trei ori frecvența necesară microprocesorului

**CLK** – pin de ieșire care furnizează semnalul de tact, ce este utilizat ca semnal de intrare în sistemul microprocesorului 8086 / 8088; semnalul generat va fi 1/3 din valoarea frecvenței cristalului

**PCLK** – un semnal de ieșire (Peripheral Clock) care este 1/6 din frecvența cristalului și poate fi utilizat ca semnal de tact pentru echipamentele periferice din sistemul 8086 / 8088

**RESET** (Ieșire Reset) – conectat la pinul de intrare RESET al procesorului 8086 / 8088; semnalul trebuie să fie sincronizat cu semnalul de tact

**~RES** (Intrare Reset) – utilizat pentru a iniția un semnal de resetare către procesorului 8086 / 8088

**READY** – pin de ieșire READY conectat la intrarea READY a procesorului 8086 și este utilizat pentru a controla durata ciclurilor de magistrală în cazul în care dispozitivele de memorie sau de I/O sunt mai lente și nu pot finaliza transferul de date la viteza procesorului

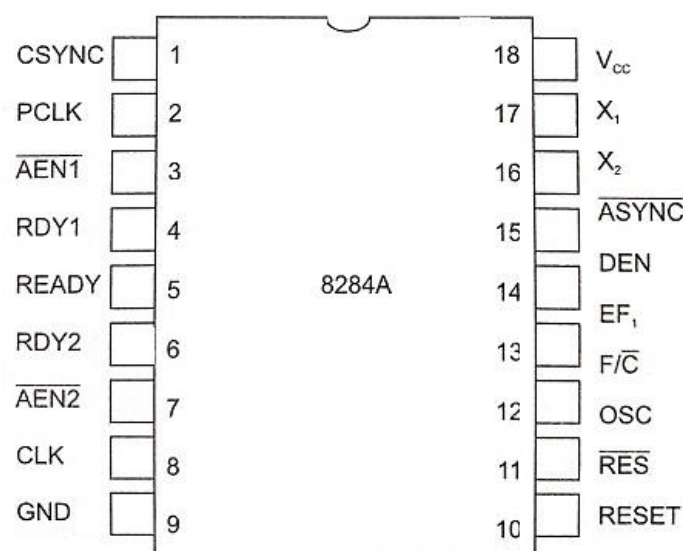
**RDY** – intrare de pregătire pentru stările de așteptare furnizată pentru a susține un sistem bazat pe 8086 / 8088; prin această intrare dispozitivele lente de memorie sau I/O pot solicita extinderea ciclurilor de magistrală

**CSYNC** – pin utilizat pentru sincronizarea semnalelor de tact într-un sistem multiprocesor, unde mai multe procesoare trebuie să primească un semnal de tact sincronizat pentru a funcționa coerent și în paralel

**~AEN1** – intrări furnizate pentru a arbitra prioritățile de magistrală ori de câte ori RDY este activă

**VCC** – pin conectat la + 5V ± 10% pentru alimentare

**GND** – pin care trebuie conectat la GND



### 2.1.3 Circuitul amplificator / separator bidirecțional 74x245

Circuitul integrat 74x245 este un amplificator și separator bidirecțional pe 8 biți, utilizat pentru a transfera date între două magistrale (bus-uri) bidirecționale. Acesta este alcătuit din 8 perechi de porți cu câte 3 intrări bidirecționale.

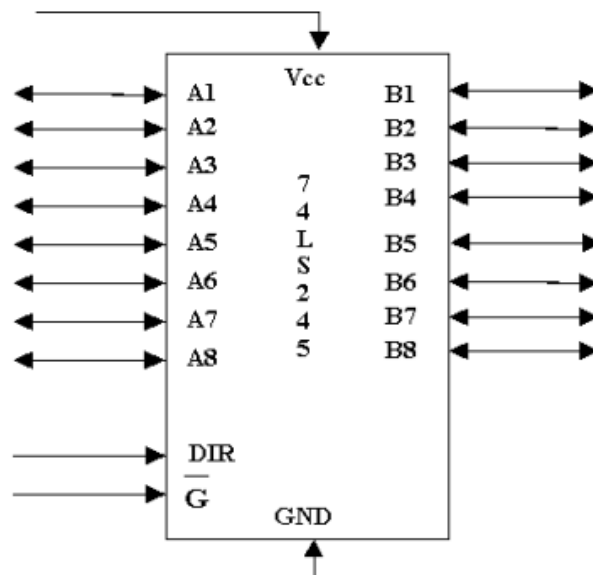
#### Configurația pinilor:

**A1 – A8** – magistrala de date pe 8 biți (canalul A)

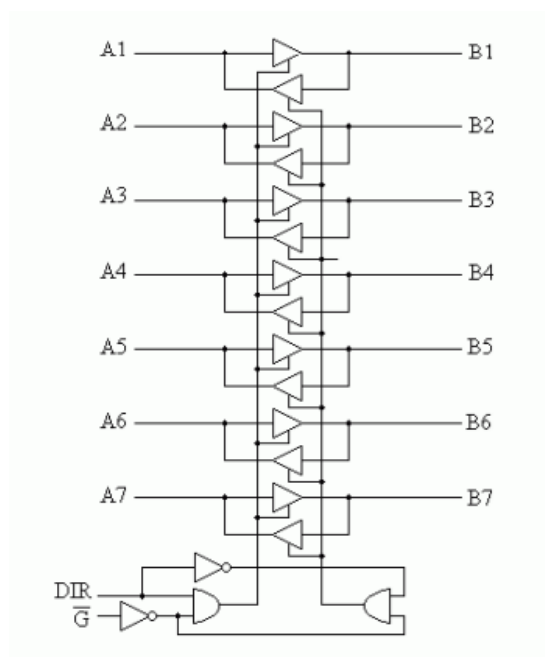
**B1 – B8** – magistrala de date pe 8 biți (canalul B)

**DIR** (Direction Control) – pinul de control al direcției de transfer al datelor din magistralele A și B

**$\sim G$**  (Output Enable) – pinul de activare al ieșirilor ce controlează starea de tri-state



#### Schema internă:





### Funcționarea:

Atunci când DIR este activ, are valoarea 1 logic, datele sunt transferate din A spre B, iar când este inactiv, datele sunt transferate din B spre A.

Când  $\sim G$  este activ, adică are valoarea 0 logic, circuitul permite transferul de date între A și B, în direcția specificată de DIR. În caz contrar, ieșirile sunt în stare de înaltă impedanță.

$\sim G$	DIR	A8 – A1	B8 – B1
0	0	B8 – B1	Intrări
0	1	Intrări	A8 – A1
1	X	A 3-a stare	A 3-a stare

### 2.1.4 Circuitul registrul 74x373

Circuitul 74x373 este un circuit integrat de tip registru folosit pentru demultiplexarea magistralei de adrese. Fiind cunoscut și ca latch de tip D pe 8 biți cu intrare tri-state, acesta conține ieșiri cu 3 stări și este alcătuit din 8 ranguri (bistabile).

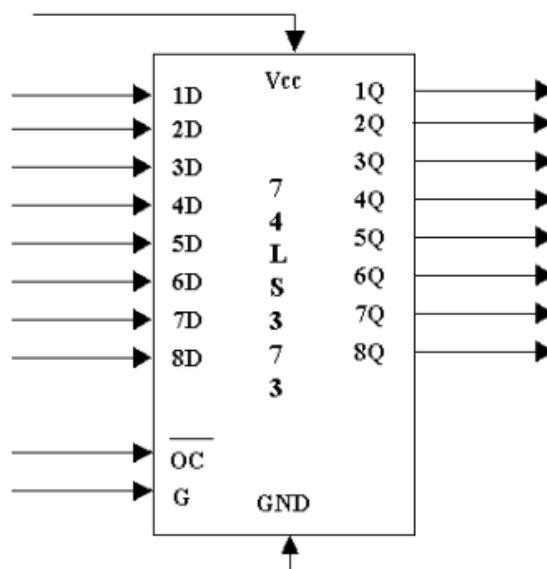
#### Configurația pinilor:

**Intrări de date (D1 – D8)** – intrări pe 8 biți, care sunt transferate la ieșirile corespunzătoare atunci când pinul G este activ

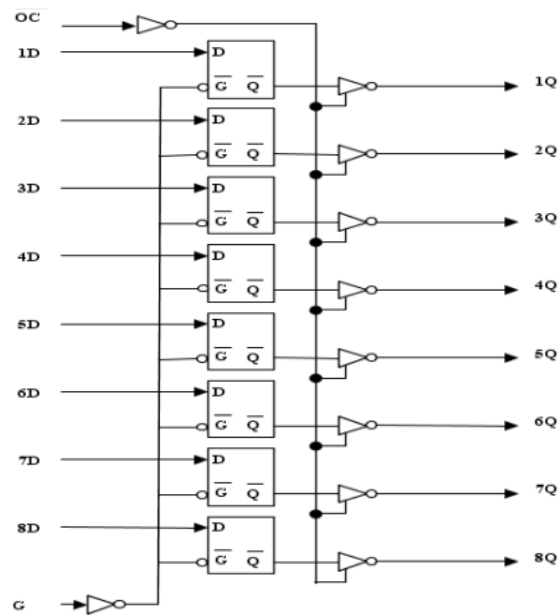
**Ieșiri (Q1 – Q8)** – ieșiri conectate la magistrală ce redau datele stocate în latch-uri

**G** (Latch Enable) – pinul de activare a stocării datelor în latch

**$\sim OC$**  (Output Control) – pinul de control al ieșirii ce determină dacă ieșirile Q sunt active pe magistrală sau se află în starea de înaltă impedanță



### Schema internă:



### Funcționarea:

Atunci când G este activ, adică are valoarea 1 logic, datele de la intrările D sunt transferate la ieșiri. Dacă G este inactiv, datele de la ieșirile Q rămân constante, chiar dacă intrările D se schimbă.

Când  $\sim OC$  este activ, adică are valoarea 0 logic, ieșirile sunt conectate la magistrală și afișează valorile stocate în latch. Dacă  $\sim OC$  este inactiv, atunci ieșirile intră în starea de înaltă impedență.

$\sim OC$	G	8Q – 1Q
0	0	Vechiul conținut
0	1	8D – 1D
1	X	A 3-a stare

## 2.2 Conectarea memoriei

În cadrul acestui proiect vom folosi următoarele circuite:

- 256 KB de memorie EPROM: un circuit EPROM 27C2048
- 64 KB de memorie SRAM: un circuit SRAM 62512

**Memoria SRAM** (Static Random-Access Memory) este un tip de memorie volatilă utilizată în electronică și informatică pentru stocarea temporară a datelor. Acestea folosesc circuite logice combinaționale pentru a memora fiecare bit. Spre deosebire de alte tipuri de memorie, cum ar fi DRAM (Dynamic RAM), memoria SRAM nu necesită reîmprospătare periodică pentru a menține datele, ceea ce o face mai rapidă și mai fiabilă pentru anumite aplicații.

**Memoria EPROM** (Erasable Programmable Read Only Memory) este un tip de memorie nevolatilă, adică o memorie care își păstrează datele chiar și când îi este întreruptă alimentarea cu curent electric, ce poate fi programată și ștearsă pentru a fi reutilizată. Aceasta este construită din celule de memorie care conțin tranzistoare cu grilă flotantă. Atunci când este programată, electronii sunt trimiși pe grila flotantă, încărcând-o negativ, ceea ce modifică starea tranzistorului și permite stocarea datelor. EPROM-urile au o fereastră transparentă de cuarț care permite pătrunderea luminii ultraviolete pentru a șterge memorie (aducerea ei la starea zero).

### 2.2.1 Circuitul de memorie EPROM 27C2048

Circuitul 27C2048 este de tip EPROM și are o capacitate de stocare de 2Mb ce este organizată în 128K locații a câte 16 biți fiecare. Timpii de acces la memorie sunt de maxim 55 ns.

#### Configurația pinilor:

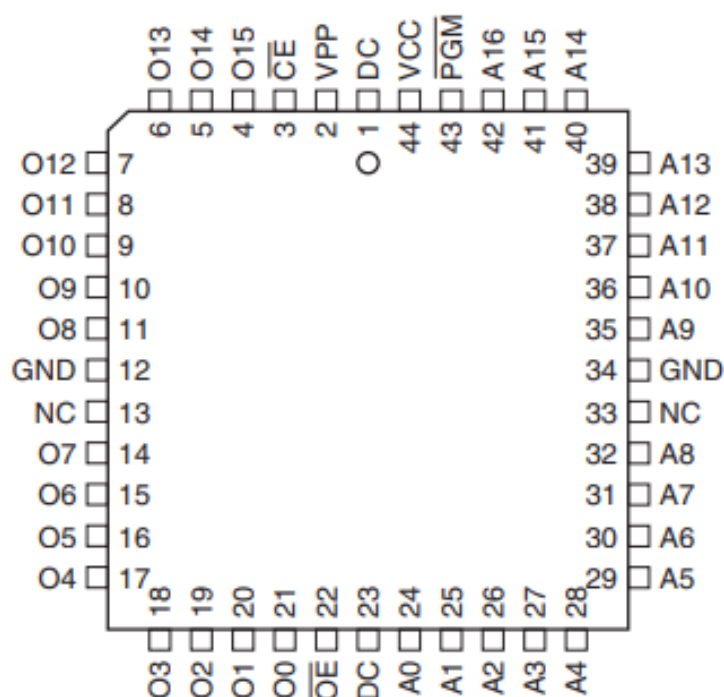
**A16 – A0** – pini pentru liniile de adrese

**O15 – O0** – pini pentru ieșirea de date ce citirea unui cuvânt de 16 biți din locația selectată de memorie

**~CE** – pin ce activează memoria SRAM; dacă ~CE este 0 memoria este activă și poate citi sau scrie date, iar dacă este 1, memoria nu răspunde la comenzi

**~OE** (Output Enable) – pin ce permite citirea datelor din memorie; dacă ~OE este 0 atunci datele de la adresa specificată sunt trimise spre pinii I/O7 -I/O0 pentru a fi citite, altfel liniile de date rămân în stare de înaltă impedanță

**~PGM** (Program Enable) – pin utilizat în procesul de programare al cipului; setarea acestuia permite scrierea datelor în cip



## 2.2.2 Circuitul de memorie SRAM 62512

Circuitul 62512 este de tip SRAM și indică o configurație de 64K x 8 biți, ceea ce înseamnă că memoria are o capacitate de stocare de 512 Kb = 64 KB (sau 64K de locații a câte 8 biți fiecare). Timpii de acces la memorie variază între 10 și 70 ns în funcție de model.

### Configurația pinilor:

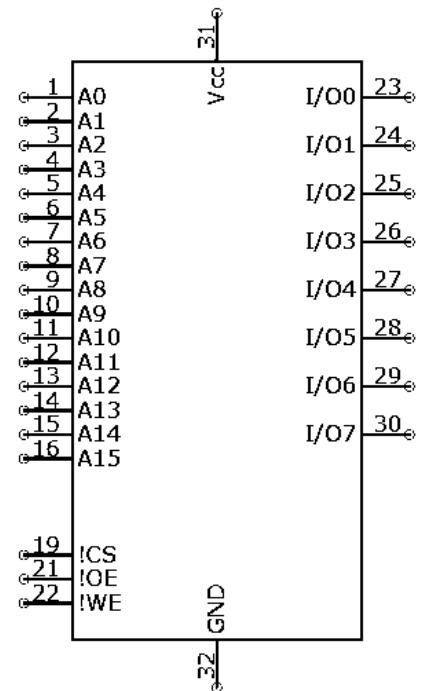
**A15 – A0** – pini pentru liniile de adrese

**I/O7 – I/O0** – linii bidirecționale de date, fiecare dintre ele fiind capabilă să primească sau să livreze un bit

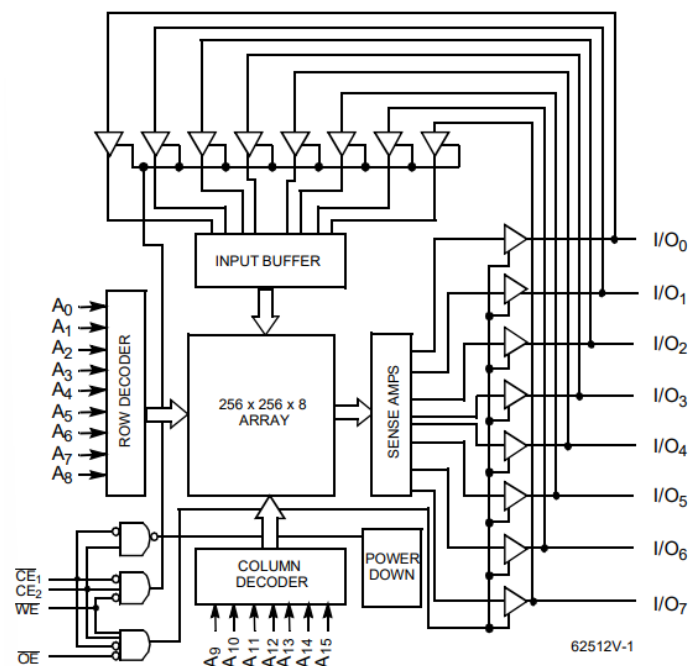
**~CS** – pin ce activează memoria SRAM; dacă ~CS este 0 memoria este activă și poate citi sau scrie date, iar dacă este 1, memoria nu răspunde la comenzi

**~OE** (Output Enable) – pin ce permite citirea datelor din memorie; dacă ~OE este 0 atunci datele de la adresa specificată sunt trimise spre pinii I/O7 -I/O0 pentru a fi citite, altfel liniile de date rămân în stare de înaltă impedanță

**~WE** (Write Enable) – pin ce permite scrierea datelor în memorie; dacă ~WE este 0 atunci datele de la pinii I/O7 - I/O0 sunt stocate în memorie la adresa specificată de liniile de adresă, altfel memoria se poate afla în modul de citire sau inactivitate



### Schema internă:

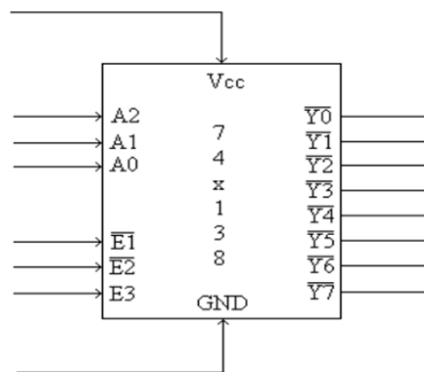


### 2.2.3 Circuitul decodificator 74x138

Decodificatorul este un circuit logic combinațional cu intrări de adresă, intrări de activare și ieșiri selectate în funcție de adresa furnizată. Acest dispozitiv convertește o adresă binară de 3 biți primită la intrare, în una dintre cele 8 ieșiri active.

#### Configurația pinilor:

Circuitul are 3 intrări (A2 – A0), 3 intrări de validare ( $\sim E1$ ,  $\sim E2$ , E3) și 8 ieșiri ( $\sim Y7$  -  $\sim Y0$ ). Acesta activează câte o ieșire numai dacă  $\sim E1 = \sim E2 = 0$  și  $E3 = 1$ , altfel toate ieșirile sunt inactive.



#### Funcționarea:

E3	/E2	/E1	A2	A1	A0	/Y7	/Y6	/Y5	/Y4	/Y3	/Y2	/Y1	/Y0
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1
0	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	1	X	X	X	1	1	1	1	1	1	1	1

#### Semnalele de selecție ale porturilor:

În cadrul acestui proiect vor exista următoarele semnale de selecție ale porturilor:

- SA1 – SA6: cele 6 ranguri ale modului de afișare cu segmente
- ST1 – ST2: minitastatura
- SL1 – SL2: cele 10 led-uri
- S1\_1 și S1\_2: interfața serială
- S2\_1 și S2\_2: interfața paralelă

## 2.2.4 Decodificarea memoriilor

### 256 KB memorie EPROM:

$2^8 * 2^{10} = 2^{18}$  biți  $\Rightarrow$  Numărul de locații în hexa = 40000h (se pune un 1 pe poziția 18: 0100 0000 0000 0000 0000)

$\Rightarrow$  Spațiul de adresare: **00000h** (adresa de început) – **3FFFFh** (adresa finală)

### 64KB memorie SRAM:

$2^6 * 2^{10} = 2^{16}$  biți  $\Rightarrow$  Numărul de locații în hexa = 10000h (se pune un 1 pe poziția 16: 0001 0000 0000 0000 0000)

$\Rightarrow$  Spațiul de adresare: **40000h** (adresa de început) – **4FFFFh** (adresa finală)

## 2.2.5 Decodificarea completă a memoriilor

CIRCUIT	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	ZONA
EPROM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00000h
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFFh
SRAM	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40000h
	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4FFFFh

În urma decodificării liniilor de adresă a celor două zone de memorie, liniile de adresă generează următoarele semnale de selecție:

$$SELe_{prom} = (\sim A_{19}) * (\sim A_{18})$$

$$SELs_{ram} = (\sim A_{19}) * A_{18} * (\sim A_{17}) * (\sim A_{16})$$

## 2.3 Interfața serială și paralelă

**Interfața serială** constă în totalitatea circuitelor și programelor de bază care asigură comunicarea între unitatea centrală și echipamentele periferice, aceasta fiind de tip bit după bit, cu sau fără fir. Datorită costului mai redus și a rezistenței la perturbații, transferul de tip serial este util atunci când informația trebuie transmisă pe distanțe mari (peste 3 metri). Costul este determinat de numărul firelor care leagă cele 2 echipamente, numărul mai mic de fire determinând un cost mai redus.

**Interfața paralelă** constă în totalitatea circuitelor și programelor de bază care asigură comunicarea între unitatea centrală și echipamentele periferice în care se transferă 8 biți simultan, iar transferul este însoțit și de semnale de dialog.

Astfel, există un semnal de dialog către modulul care primește datele. Prin acesta se specifică dacă datele sunt stabile pe linii și dacă pot fi preluate. Pe lângă acesta există și cel puțin un semnal de dialog de la modulul care primește datele, prin care acesta comunică primirea acestora sau eventuala indisponibilitate a sa de a primi datele.

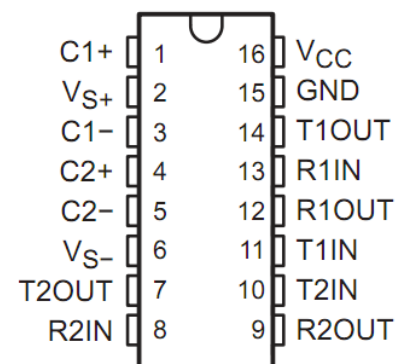
În cadrul acestui proiect vom folosi:

- circuitul MAX232
- circuitul 8251 pentru interfața serială
- circuitul 8255 pentru interfața paralelă
- circuitul decodicator 74x138 pentru conectarea la porturi

### 2.3.1 Circuitul MAX232

Circuitul MAX32 este un circuit integrat (IC) utilizat pentru conversia semnalelor logice între nivelurile de tensiune TTL (0V – 5V) și RS-232 ( $\pm 12V$ ), care este un standard de comunicație serială folosit adesea pentru a conecta diferite dispozitive.

**TXD** (Transmit Data) și **RXD** (Receive Data) ale circuitului **8251** sunt semnale TTL. Acestea trebuie conectate la pinii **T1IN** (transmitere intrare) și **R1OUT** (recepționare ieșire) ale **MAX232**.



**T1OUT** și **R1IN** ale **MAX232** vor fi conectate la pinii corespunzători de **TX** și **RX** ai dispozitivului care utilizează semnale **RS-232** (de exemplu, un port serial RS-232 al unui computer sau al unui modem).

MAX232 necesită 4 condensatori externi (de obicei  $1\mu F$  sau  $10\mu F$ ) pentru a asigura tensiunile de  $+12V$  și  $-12V$  necesare semnalelor RS-232.

### 2.3.2 Interfața serială cu circuitul 8251

Circuitul 8251 este un circuit specializat pentru transferurile seriale ce face parte din categoria circuitelor de tip USART (Universal Synchronous Asynchronous Receiver Transmitter), având 2 moduri de lucru: sincron și asincron. Acesta poate să primească un octet în paralel de la unitatea centrală, să-l serializeze și să-l transmită la un echipament serial sau să preia de pe linie, de la un echipament periferic serial, un octet, să-l assembleze și să-l predea, în paralel, unității centrale. Circuitul comunică, prin program sau prin întreruperi, unității centrale când are un caracter gata pentru ea sau când a terminat de transmis un octet și poate prelua altul.

Interfața serială varianta 1:

- port date: **0AF0h**
- port comenzi: **0AF2h**

Interfața serială varianta 2:

- port date: **0BF0h**
- port comenzi: **0BF2h**

### Configurația pinilor:

**C / ~ D** – selectează modul de operare al lui 8251; dacă este 1 este sincron, altfel este asincron

**RXD** (Receive Data) – semnal de recepționare a datelor de la portul serial

**~ RXC** (Receive Clock) – semnal utilizat pentru sincronizarea fluxului de date recepționate

**RXRDY** (Receive Ready) – semnal ce indică faptul că datele sunt disponibile pentru a fi citite de la 8251

**SYNDET / BD** (Syndrome / Break Detection) – semnal ce semnalează detectarea unui sindrom, adică o eroare de formă a datelor, sau o întrerupere pe linia serială

**TXD** (Transmit Data) – semnal de transmitere a datelor către portul serial

**~ TXC** (Transmit Clock) – semnal utilizat pentru sincronizarea transmisiei de date în modul sincron al 8251

**TXRDY** (Transmit Ready) – semnal folosit pentru a indica disponibilitatea unui registru de transmitere pentru date noi

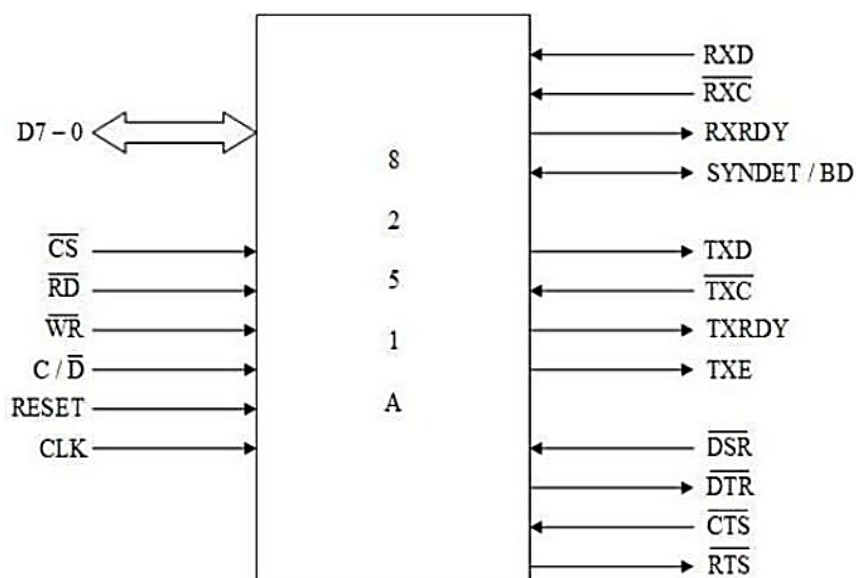
**TXE** (Transmit Enable) – semnal de control pentru a activa transmiterea

**~ DSR** (Data Set Ready) – semnal de stare care indică dacă echipamentul de la celălalt capăt al liniei este gata pentru a trimite date

**~ DTR** (Data Terminal Ready) – semnal care indică faptul că echipamentul local este gata pentru a începe comunicarea

**~ CTS** (Clear To Send) – semnal folosit pentru controlul fluxului

**~ RTS** (Request To Send) – semnal de control al transiterii datelor, folosit pentru a solicita permisiunea de a transmite date





### Funcționarea:

/CS	/RD	/WR	C//D	Operație
1	X	X	X	Magistrala de date este în a 3-a stare
0	1	1	X	Magistrala de date este în a 3-a stare
0	0	1	1	Citirea octetului de stare
0	0	1	0	Citirea datelor
0	1	0	1	Scrierea cuvintelor de comandă
0	1	0	0	Scrierea datelor

### Decodificarea adreselor de memorie:

VARIANTA	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	ADRESA
S1_1	0	0	0	0	1	0	1	0	1	1	1	1	0	0	0	0	0AF0h
	0	0	0	0	1	0	1	0	1	1	1	1	0	0	1	0	0AF2h
S1_2	0	0	0	0	1	0	1	1	1	1	1	1	0	0	0	0	0BF0h
	0	0	0	0	1	0	1	1	1	1	1	1	0	0	1	0	0BF2h

$$S1\_1 = (\sim A10) * A9 * (\sim A8) = \sim Y2$$

$$S1\_2 = (\sim A10) * A9 * A8 = \sim Y3$$

### 2.3.3 Interfața paralelă cu circuitul 8255

Circuitul 8255 are 24 de pini de I/O care pot fi configurate în mai multe feluri în funcție de modul de lucru ales: 2 grupe de câte 12 linii fără semnale de dialog, 2 grupe a câte 8 linii cu semnale de dialog, o grupă de 8 linii bidirecționale cu semnale de dialog.

Comunicarea cu circuitul 8255 se face prin intermediul a 4 adrese de port, corespunzătoare porturilor A, B, C și portului pentru cuvântul de comandă.

Programarea circuitului se realizează prin transmiterea unui cuvânt de comandă la adresa portului cuvântului de comandă. Se poate cere porturilor să lucreze în unul din următoarele moduri:

- modul 0, numit și mod de I/O, pentru porturile A, B și C
- modul 1, numit și mod de I/O cu dialog, pentru porturile A și B
- modul 2, numit și mod bidirecțional cu dialog pentru portul A

Interfața paralelă varianta 1:

- port A: **0D70h**
- port B: **0D72h**
- port C: **0D74h**
- port RCC (registrul de comandă și control): **0D76h**

Interfața paralelă varianta 2:

- port A: **0C70h**
- port B: **0C72h**
- port C: **0C74h**
- port RCC: **0C76h**

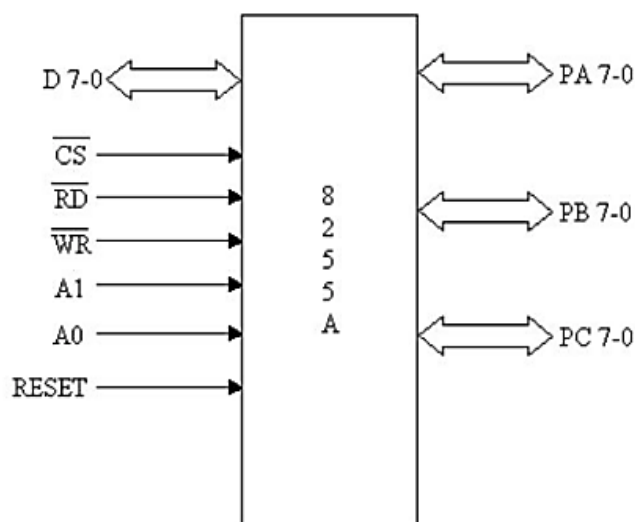
### Configurația pinilor:

**A0 și A1** – semnale de adresă folosite pentru a seta modul de operare al 8255 și pentru a selecta modul de lucru al porturilor

**PA7 – PA0** – port de 8 biți pentru I/O

**PB7 – PB0** – port de 8 biți pentru I/O

**PC7 – PC0** – pini de control pentru gestionarea semnalelor de stare sau date suplimentare



### Funcționarea:

/CS	/RD	/WR	A1	A0	Operație
0	1	0	0	0	Scrisoare în portul A
0	1	0	0	1	Scrisoare în portul B
0	1	0	1	0	Scrisoare în portul C
0	1	0	1	1	Scrisoare în portul cuvântului de comandă
0	0	1	0	0	Citire din portul A
0	0	1	0	1	Citire din portul B
0	0	1	1	0	Citire din portul C
0	0	1	1	1	Fără operație – magistrala de date este în a 3-a stare
0	1	1	X	X	Fără operație – magistrala de date este în a 3-a stare
1	X	X	X	X	Magistrala de date este în a 3-a stare

## Decodificarea adreselor de memorie:

VARIANTA	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	ADRESA
S2_1	0	0	0	0	1	1	0	1	0	1	1	1	0	0	0	0	0D70h
	0	0	0	0	1	1	0	1	0	1	1	1	0	0	1	0	0D72h
	0	0	0	0	1	1	0	1	0	1	1	1	0	1	0	0	0D74h
	0	0	0	0	1	1	0	1	0	1	1	1	0	1	1	0	0D76h
S2_2	0	0	0	0	1	1	0	0	0	1	1	1	0	0	0	0	0C70h
	0	0	0	0	1	1	0	0	0	1	1	1	0	0	1	0	0C72h
	0	0	0	0	1	1	0	0	0	1	1	1	0	1	0	0	0C74h
	0	0	0	0	1	1	0	0	0	1	1	1	0	1	1	0	0CF6h

$$S2\_1 = A10 * (\sim A9) * A8 = \sim Y5$$

$$S2\_2 = A10 * (\sim A9) * (\sim A8) = \sim Y4$$

## 2.4 Conectarea afișajelor

În cadrul acestui proiect vom folosi următoarele componente pentru interfața cu utilizatorul:

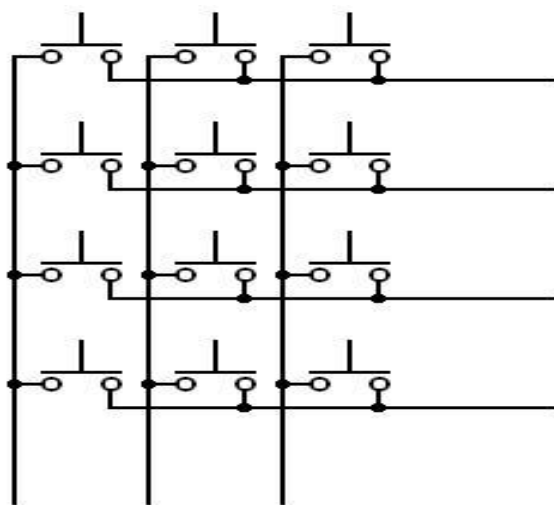
- o minitatură cu 12 contacte
- 10 led-uri
- 6 ranguri de modul de afișare cu segmente

### 2.4.1 Minitastatura cu 12 contacte

**Minitastatura** este o structură matriceală care are taste la intersecția liniilor și coloanelor. Rândurile sunt conectate la porturi de intrare (porți cu 3 stări), iar coloanele sunt conectate la porturi de ieșire ce are posibilitatea de memorare (registru). Se efectuează baleierea (scanning) coloanelor prin trimiterea succesivă a unui semnal logic "0" pe fiecare coloană, în timp ce celelalte coloane sunt setate pe logic "1". În acest timp, se citesc semnalele de pe rânduri. Dacă o tastă este apăsată, aceasta creează o conexiune între coloană și rând, iar semnalul "0" va fi detectat pe rândul corespunzător. Prin combinarea poziției coloanei active cu rândul pe care se detectează un semnal "0", microcontrolerul poate determina care tastă a fost apăsată.

Pentru protecția ieșirilor de la portul de ieșire, fiecare tastă include o diodă conectată în serie. Aceasta permite curentului să circule într-o singură direcție și împiedică interferențele între linii și coloane, reducând riscul unui scurtcircuit.

În cadrul acestui proiect vom folosi o minitastatură formată din 12 contacte, cu o structură matriceală 4x3, 4 rezistențe, 3 diode conectate la ieșiri, un registru 74x373 și un circuit 74x244. Aceasta este selectată de semnalele **ST1** (intrare) și **~ST2** (ieșire).



Fie **ST1** semnalul de selecție pentru zona de memorie a portului de intrare al minitastaturii

⇒ spațiul de adresare va fi **0D10h – 0D17h**

Fie **ST2** semnalul de selecție pentru zona de memorie a portului de ieșire al minitastaturii

⇒ spațiul de adresare va fi **0D18h – 0D1Fh**

ZONA	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	ADRESA
ST1	0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	0	0D10h
	0	0	0	0	1	1	0	1	0	0	0	1	0	1	1	1	0D17h
ST2	0	0	0	0	1	1	0	1	0	0	0	1	1	0	0	0	0D18h
	0	0	0	0	1	1	0	1	0	0	0	1	1	1	1	1	0D1Fh

$$ST1 = (\sim A5) * A4 * (\sim A3) = \sim Y2$$

$$ST2 = (\sim A5) * A4 * A3 = \sim Y3$$

#### 2.4.1.1 Circuitul amplificator / separator unidirecțional 74x244

Circuitul 74x244 este un circuit integrat logic din familia TTL sau CMOS utilizat în principal pentru buffering (amplificarea și izolarea semnalelor) în aplicații digitale. Este cunoscut ca un buffer/driver unidirecțional cu ieșiri tri-state, ceea ce îl face util în controlul magistralelor comune și în protejarea componentelor electronice. Acesta separă magistralele unidirecționale astfel încât să nu existe interferențe între diferite componente conectate la o linie comună.

### Configurația pinilor:

**1A1 – A14** – grupul 1 de intrări

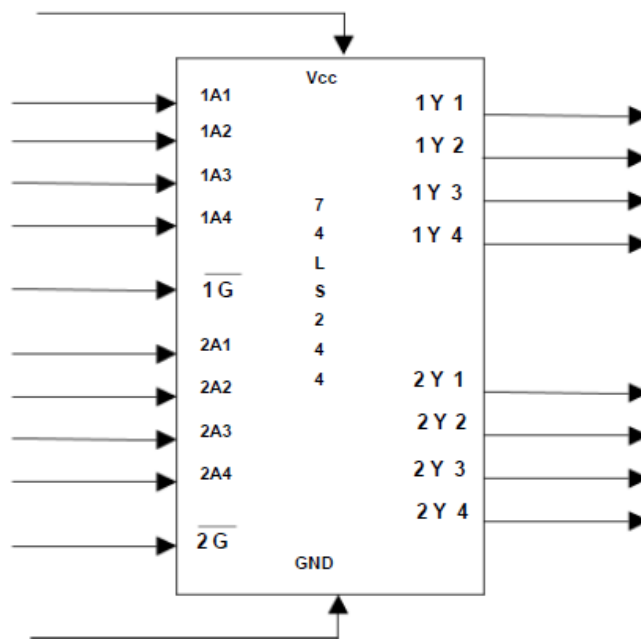
**2A1 – 2A4** – grupul 2 de intrări

**1Y1 – 1Y4** – grupul 1 de ieșiri

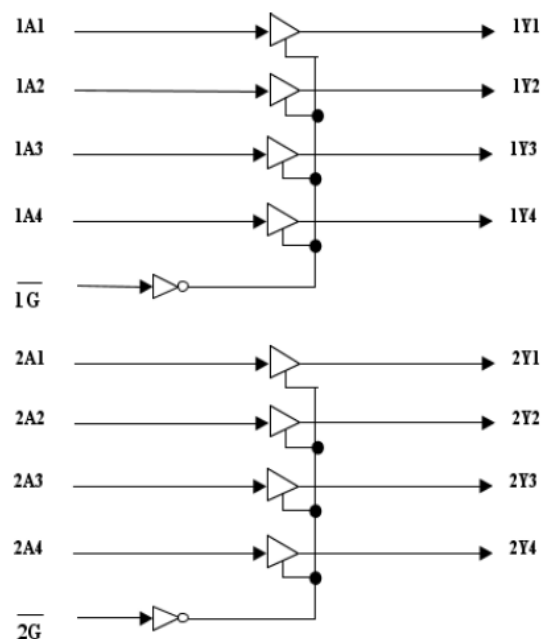
**2Y1 – 2Y4** – grupul 2 de ieșiri

**~ 1G** – controlează grupul 1 (1A1–1A4 → 1Y1–1Y4)

**~ 2G** – controlează grupul 2 (2A1–2A4 → 2Y1–2Y4)



### Schema internă:



### Funcționarea:

Dacă **1G** sau **2G** este Low (0 logic), ieșirile grupului activat urmează direct valorile intrărilor.

Dacă **1G** sau **2G** este High (1 logic), ieșirile grupului respectiv sunt deconectate (stare de impedanță mare), ceea ce permite altor dispozitive să utilizeze magistrala fără conflicte.

1/G	2/G	1Y1 – 1Y4	2Y1 – 2Y4
0	0	1A4 – 1A1	2A4 – 2A1
0	1	1A4 – 1A1	A 3-a stare
1	0	A 3-a stare	2A4 – 2A1
1	1	A 3-a stare	A 3-a stare

### Conectarea cu minitastatura:

Circuitul 74x244 va fi utilizat pentru a conecta minitastatura la microprocesorul 8086, eficientizând și simplificând comunicarea între cele două. Acest circuit este utilizat cu mai multe roluri: pentru izolarea semnalelor tastaturii de microcontroler și protejarea magistralei de intrări a acestuia de eventualele fluctuații sau interferențe care pot apărea în tastatură; pentru multiplexare: pentru reducerea complexității conexiunilor între tastatură și microprocesor; permite ca aceeași tastatură să fie utilizată de mai multe module fără conflicte pe magistrală chiar și atunci când sunt în stare de impedanță mare.

#### 2.4.2 Led-uri

**LED** (light emitting diode) este o diodă semiconductoare, care emite lumină la polarizarea directă joncțiunii p-n. Pentru ca un led să fie aprins la ieșirea portului trebuie să fie 0 logic, iar pentru stingerea acestuia se încarcă 1 logic pe poziția corespunzătoare. Se folosește un registru 74x373 conectate la magistrală pentru a le menține aprinse sau stinse.

Pentru acest microsistem se vor folosi 2 registre 74x373 conectate la magistrală pentru a menține cele 10 leduri aprinse sau stinse.

Fie **SL1** semnalul de selecție pentru zona de memorie a primelor 8 led-uri

⇒ spațiul de adresare va fi **0D20h – 0D27h**

Fie **SL2** semnalul de selecție pentru zona de memorie a următoarelor 2 led-uri

⇒ spațiul de adresare va fi **0D28h – 0D2Fh**

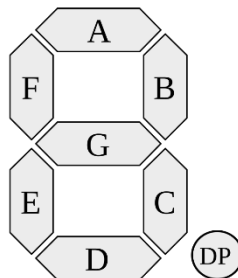
ZONA	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	ADRESA
SL1	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0D20h
	0	0	0	0	1	1	0	1	0	0	1	0	0	1	1	1	0D27h
SL2	0	0	0	0	1	1	0	1	0	0	1	0	1	0	0	0	0D28h
	0	0	0	0	1	1	0	1	0	0	1	0	1	1	1	1	0D2Fh

$$SL1 = A5 * (\sim A4) * (\sim A3) = \sim Y4$$

$$SL2 = A5 * (\sim A4) * A3 = \sim Y5$$

### 2.4.3 Modul de afișaj cu 7 segmente

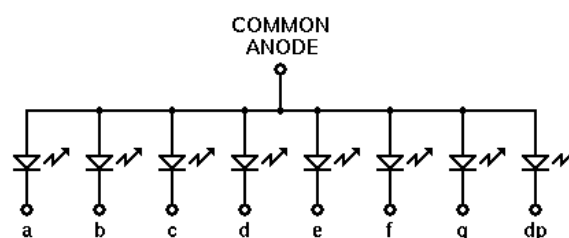
**Modulul de afișaj cu șapte segmente** este un dispozitiv electronic care permite afișarea numerelor zecimale (sau hexazecimale). Acesta este format din șapte LED-uri, sau segmente, aranjate într-un model dreptunghiular pentru afișarea numărului și încă un LED pentru punctul decimal. Primul segment, notat cu "a", este cel din partea de sus, iar celelalte segmente se notează în sensul acelor de ceas, segmentul de mijloc fiind notat cu "g". Pentru afișajele care utilizează și punct, acesta este notat cu "dp" (decimal point).



















În cadrul acestui proiect vom folosi 6 module de afișaj cu șapte segmente de tip anod comun, 6 registre 74x373 pentru control și 6 rezistențe.

#### Tipul de afișaj cu anod comun:

În afișajul cu anod comun, toate conexiunile anod ale segmentelor LED sunt legate între ele la "1" logic. Segmentele individuale sunt iluminate prin aplicarea unui semnal de masă, "0" logic sau LOW, prin intermediul unui rezistor limitator de curent adecvat la catodul segmentului particular.



**Codificarea caracterelor:**

HEXA	dp	g	f	e	d	c	b	a	COD	AFIȘAJ
0	1	1	0	0	0	0	0	0	C0 h	
1	1	1	1	1	1	0	0	1	F9 h	
2	1	0	1	0	0	1	0	0	A4 h	
3	1	0	1	1	0	0	0	0	B0 h	
4	1	0	0	1	1	0	0	1	99 h	
5	1	0	0	1	0	0	1	0	92 h	
6	1	0	0	0	0	0	1	0	82 h	
7	1	1	1	1	1	0	0	0	F8 h	
8	1	0	0	0	0	0	0	0	80 h	
9	1	0	0	1	0	0	0	0	90 h	
A	1	0	0	0	1	0	0	0	88 h	
B	1	0	0	0	0	0	1	1	83 h	
C	1	1	0	0	0	1	1	0	C6 h	
D	1	0	1	0	0	0	0	1	A1 h	
E	1	0	0	0	0	1	1	0	86 h	
F	1	0	0	0	1	1	1	0	8E h	



### Harta memoriei:

Fie **SA1** semnalul de selecție pentru zona de memorie pentru primul afișaj cu șapte segmente

⇒ spațiul de adresare va fi **0E00h – 0E07h**

Fie **SA2** semnalul de selecție pentru zona de memorie pentru al doilea afișaj cu șapte segmente

⇒ spațiul de adresare va fi **0E08h – 0E0Fh**

Fie **SA3** semnalul de selecție pentru zona de memorie pentru al treilea afișaj cu șapte segmente

⇒ spațiul de adresare va fi **0E10h – 0E17h**

Fie **SA4** semnalul de selecție pentru zona de memorie pentru al patrulea afișaj cu șapte segmente

⇒ spațiul de adresare va fi **0E18h – 0E1Fh**

Fie **SA5** semnalul de selecție pentru zona de memorie pentru al cincilea afișaj cu șapte segmente

⇒ spațiul de adresare va fi **0E20h – 0E27h**

Fie **SA6** semnalul de selecție pentru zona de memorie pentru al șaselea afișaj cu șapte segmente

⇒ spațiul de adresare va fi **0E28h – 0E2Fh**

ZONA	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	ADRESA
SA1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0E00h
	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0E07h
SA2	0	0	0	0	1	1	1	0	0	0	0	0	1	0	0	0	0E08h
	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	1	0E0Fh
SA3	0	0	0	0	1	1	1	0	0	0	0	1	0	0	0	0	0E10h
	0	0	0	0	1	1	1	0	0	0	0	1	0	1	1	1	0E07h
SA4	0	0	0	0	1	1	1	0	0	0	0	1	1	0	0	0	0E18h
	0	0	0	0	1	1	1	0	0	0	0	1	1	1	1	1	0E1Fh
SA5	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0E20h
	0	0	0	0	1	1	1	0	0	0	1	0	0	1	1	1	0E27h
SA6	0	0	0	0	1	1	1	0	0	0	1	0	1	0	0	0	0E28h
	0	0	0	0	1	1	1	0	0	0	1	0	1	1	1	1	0E2Fh

În urma decodificării liniilor de adresă se generează următoarele semnale de selecție:

$$SA1 = (\sim A5) * (\sim A4) * (\sim A3) = \sim Y0$$

$$SA2 = (\sim A5) * (\sim A4) * (A3) = \sim Y1$$

$$SA3 = (\sim A5) * (A4) * (\sim A3) = \sim Y2$$

$$SA4 = (\sim A5) * (A4) * (A3) = \sim Y3$$

$$SA5 = (A5) * (\sim A4) * (\sim A3) = \sim Y4$$

$$SA6 = (A5) * (\sim A4) * (A3) = \sim Y5$$

### 3. SUBROUTINE ÎN LIMBAJ DE ASAMBLARE

#### 3.1 Rutina de programare a circuitului 8251

**PROGRAMARE\_8251:**

; Determinarea bazei de adrese în funcție de poziția S1

; Intrare: AH - poziția comutatorului semnalului de selecție /S51

**CMP AH, 1** ; verificăm dacă AH este 1 (0AF0H - 0AF2H)

**JE BASE\_0BF2** ; dacă AH este 1, folosim adresa 0BF2H

**MOV DX, 0AF2H** ; dacă AH nu este 1, folosim adresa 0AF2H

**JMP CONTINUE** ; sărim peste configurarea pentru 0BF2H

**BASE\_0BF2:** ; etichetă pentru cazul în care AH nu este 1

**MOV DX, 0BF2H** ; setăm adresa la 0BF2H

**CONTINUE:** ; etichetă pentru continuarea programului

**MOV AL, 0CEH** ; încărcăm registrul AL cu valoarea 0CEH ce reprezintă un cuvânt de mod pentru circuitul 8251 (comunicație asincronă, fără paritate, factor de umplere x16, cu 8 biți de date și 2 biți de stop)

**OUT DX, AL** ; trimitem conținutul registrului AL la portul DX

**MOV AL, 15H** ; încărcăm registrul AL cu valoarea 15H ce reprezintă un cuvânt de comandă pentru circuitul 8251 (activarea transmițătorului și a receptorului)

**OUT DX, AL** ; trimitem conținutul registrului AL la portul DX

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

### 3.2 Rutina de emisie a unui caracter pe interfața serială

#### **EMISIE\_8251:**

; Determinarea bazei de adrese în funcție de poziția S1  
; Intrare: AH – poziția comutatorului semnalului de selecție /S51  
; CL – caracterul ce va fi transmis

**CMP AH, 1** ; verificăm dacă AH este 1 (0AF0H – 0AF2H)

**JE BASE\_0BF2** ; dacă AH este 1, folosim adresa 0BF2H

**MOV DX, 0AF2H** ; dacă AH nu este 1, folosim adresa 0AF2H

**JMP CONTINUE** ; sărim peste configurarea pentru 0BF2H

**BASE\_0BF2:** ; etichetă pentru cazul în care AH nu este 1

**MOV DX, 0BF2H** ; setăm adresa la 0BF2H

**CONTINUE:** ; etichetă pentru continuarea programului

**IN AL, DX** ; citire și testare rang TxRDY din cuvântul de stare pentru a vedea dacă interfața este pregătită pentru transmisie

**RCR AL, 1** ; rotim registrul AL spre dreapta și mutăm MSB (ce reprezintă TxRDY) în CF (Carry Flag)

**JNC CONTINUE** ; dacă CF = 0 (interfața nu este pregătită), continuăm să verificăm ; starea portului într-un loop (așteptăm până devine disponibil)

**MOV AL, CL** ; mutăm caracterul care trebuie transmis în registrul AL

**SUB DX, 2** ; calculăm adresa portului de date

**OUT DX, AL** ; trimitem conținutul registrului AL la portul DX

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

### 3.3 Rutina de recepție a unui caracter pe interfața serială

#### **RECEPTIE\_8251:**

; Determinarea bazei de adrese în funcție de poziția S1  
; Intrare: AH - poziția comutatorului semnalului de selecție /S51  
; CL - caracterul ce va fi recepționat

**CMP AH, 1** ; verificăm dacă AH este 1 (0AF0H - 0AF2H)

**JE BASE\_0BF2** ; dacă AH este 1, folosim adresa 0BF2H

**MOV DX, 0AF2H** ; dacă AH nu este 1, folosim adresa 0AF2H

**JMP CONTINUE** ; sărim peste configurarea pentru 0BF2H

**BASE\_0BF2:** ; etichetă pentru cazul în care AH nu este 1

**MOV DX, 0BF2H** ; setăm adresa la 0BF2H

**CONTINUE:** ; etichetă pentru continuarea programului

**IN AL, DX** ; citire și testare rang TxRDY din cuvântul de stare pentru a vedea dacă interfața este pregătită pentru transmisie

**RCR AL, 2** ; rotim registrul AL spre dreapta de două ori și mutăm RxRDY (ce indică dacă un caracter a fost primit și poate fi citit) în CF

**JNC CONTINUE** ; dacă CF = 0 (interfața nu este pregătită), continuăm să verificăm ; starea portului într-un loop (așteptăm până devine disponibil)

**SUB DX, 2** ; calculăm adresa portului de date

**IN AL, DX** ; citim caracterul recepționat din portul de date (DX) și îl stocăm în registrul AL

**MOV CL, AL** ; mutăm caracterul din AL în registrul CL

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

### 3.4 Rutina de programare a circuitului 8255

#### **PROGRAMARE\_8255:**

; Determinarea bazei de adrese în funcție de poziția S2

; Intrare: AH – poziția comutatorului semnalului de selecție /S55

**CMP AH, 1** ; verificăm dacă AH este 1 (0C70H – 0C76H)

**JE BASE\_0C76** ; dacă AH este 1, sărim la setarea adresei pentru 0C76H

**MOV DX, 0D76H** ; dacă AH nu este 1, folosim adresa 0D76H

**JMP CONTINUE** ; sărim peste configurarea pentru 0C76H

**BASE\_0C76:** ; etichetă pentru cazul în care AH nu este 1

**MOV DX, 0C76H** ; setăm adresa la 0C76H

**CONTINUE:** ; etichetă pentru continuarea programului

**MOV AL, 81H** ; încărcăm registrul AL cu valoarea 81H ce reprezintă un cuvânt de comandă pentru circuitul 8255

**OUT DX, AL** ; trimitem conținutul registrului AL la portul DX

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

### 3.5 Rutina de emisie caracter pe interfața paralelă

#### **EMISIE\_8255:**

```
; Determinarea bazei de adrese în funcție de poziția S2
; Intrare: AH - poziția comutatorului semnalului de selecție /S55
;          CL - caracterul ce va fi transmis

    CMP AH, 1 ; verificăm dacă AH este 1 (0C70H - 0C76H)
    JE BASE_0C74 ; dacă AH este 1, sărim la setarea adresei pentru 0C74H
    MOV DX, 0D74H ; dacă AH nu este 1, folosim adresa 0D74H
    JMP CONTINUE ; sărim peste configurarea pentru 0C74H
```

**BASE\_0C74:** ; etichetă pentru cazul în care AH nu este 1

```
    MOV DX, 0C74H ; setăm adresa la 0C74H
```

**CONTINUE:** ; etichetă pentru continuarea programului

```
    IN AL, DX ; se citește starea portului C pentru a verifica dacă interfața
    este pregătită să primească date

    RCR AL, 1 ; rotim bitul cel mai semnificativ (MSB) în CF (Carry Flag)

    JNC CONTINUE ; dacă CF = 0 (interfața nu este pregătită), continuăm să
    verificăm ; starea portului într-un loop (așteptăm până devine disponibil)

    MOV AL, CL ; mutăm caracterul care trebuie transmis (din CL) în AL

    SUB DX, 4 ; calculăm adresa portului A

    OUT DX, AL ; trimitem conținutul registrului AL la portul DX

    OR AL, 01H ; activăm semnalul STROBE, setând bitul 0 din registrul AL pe
    1; semnalul STROBE indică interfeței paralele că datele sunt pregătite

    ADD DX, 2 ; calculăm adresa portului B

    OUT DX, AL ; Scriem semnalul STROBE pe portul de control DX

    AND AL, 00H ; resetăm semnalul STROBE, setând toți biții din AL pe 0

    OUT DX, AL ; scriem valoarea 0 pe portul de control, dezactivând semnalul
    STROBE

    OR AL, 01H ; reactivăm semnalul STROBE pentru operații ulterioare

    OUT DX, AL ; scriem valoarea actualizată pe portul de control

    RET ; marchează finalul subrutinei și transferul controlului înapoi la
    apelant
```

### 3.6 Rutina de scanare a minitastaturii

#### TASTATURA:

; se pune 0 pe prima coloană și se verifică dacă s-au acționat tastele 1,4,7,\*

**MOV AL, 0FEH** ; punem 0 pe prima coloană și 1 pe celelalte (11111110B)

**MOV DX, 0D10H** ; scriem în DX adresa portului de intrare pentru coloane

**OUT DX, AL** ; scriem masca la port pentru a activa prima coloană

; citim starea rândurilor

**MOV DX, 0D18H** ; scriem în DX adresa portului de ieșire pentru rânduri

**IN AL, DX** ; citim starea rândurilor în AL

**AND AL, 01H** ; testăm primul bit, corespunzător tastei 1

**JZ TASTA1** ; dacă bitul este 0, atunci tasta este apăsată și sărim la rutina ei

**IN AL, DX** ; citim starea rândurilor în AL

**AND AL, 02H** ; testăm al doilea bit, corespunzător tastei 4

**JZ TASTA4** ; dacă bitul este 0, atunci tasta este apăsată și sărim la rutina ei

**IN AL, DX** ; citim starea rândurilor în AL

**AND AL, 04H** ; testăm al treilea bit, corespunzător tastei 7

**JZ TASTA7** ; dacă bitul este 0, atunci tasta este apăsată și sărim la rutina ei

**IN AL, DX** ; citim starea rândurilor în AL

**AND AL, 08H** ; testăm al patrulea bit, corespunzător tastei \*

**JZ TASTA\_STEA** ; dacă bitul este 0, atunci tasta este apăsată și sărim la rutina ei

; se pune 0 pe a doua coloană și se verifică dacă s-au acționat tastele 2,5,8,0

**MOV AL, 0FDH** ; punem 0 pe a doua coloană și 1 pe celelalte (11111101B)

**MOV DX, 0D10H** ; scriem în DX adresa portului de intrare pentru coloane

**OUT DX, AL** ; scriem masca la port pentru a activa prima coloană



; citim starea rândurilor

**MOV DX, 0D18H** ; scriem în DX adresa portului de ieșire pentru rânduri

**IN AL, DX** ; citim starea rândurilor în AL

**AND AL, 01H** ; testăm primul bit, corespunzător tastei 2

**JZ TASTA2** ; dacă bitul este 0, atunci tasta este apăsată și sărim la rutina ei

**IN AL, DX** ; citim starea rândurilor în AL

**AND AL, 02H** ; testăm al doilea bit, corespunzător tastei 5

**JZ TASTA5** ; dacă bitul este 0, atunci tasta este apăsată și sărim la rutina ei

**IN AL, DX** ; citim starea rândurilor în AL

**AND AL, 04H** ; testăm al treilea bit, corespunzător tastei 8

**JZ TASTA8** ; dacă bitul este 0, atunci tasta este apăsată și sărim la rutina ei

**IN AL, DX** ; citim starea rândurilor în AL

**AND AL, 08H** ; testăm al patrulea bit, corespunzător tastei 0

**JZ TASTA0** ; dacă bitul este 0, atunci tasta este apăsată și sărim la rutina ei

; se pune 0 pe a treia coloană și se verifică dacă s-au acționat tastele 3,6,9,#

**MOV AL, 0FBH** ; punem 0 pe a doua coloană și 1 pe celelalte (11111011B)

**MOV DX, 0D10H** ; scriem în DX adresa portului de intrare pentru coloane

**OUT DX, AL** ; scriem masca la port pentru a activa prima coloană

; citim starea rândurilor

**MOV DX, 0D18H** ; scriem în DX adresa portului de ieșire pentru rânduri

**IN AL, DX** ; citim starea rândurilor în AL

**AND AL, 01H** ; testăm primul bit, corespunzător tastei 3

**JZ TASTA3** ; dacă bitul este 0, atunci tasta este apăsată și sărim la rutina ei

**IN AL, DX ;** citim starea rândurilor în AL

**AND AL, 02H ;** testăm al doilea bit, corespunzător tastei 6

**JZ TASTA6 ;** dacă bitul este 0, atunci tasta este apăsată și sărim la rutina ei

**IN AL, DX ;** citim starea rândurilor în AL

**AND AL, 04H ;** testăm al treilea bit, corespunzător tastei 9

**JZ TASTA9 ;** dacă bitul este 0, atunci tasta este apăsată și sărim la rutina ei

**IN AL, DX ;** citim starea rândurilor în AL

**AND AL, 08H ;** testăm al patrulea bit, corespunzător tastei #

**JZ TASTA\_HASHTAG ;** dacă bitul este 0, atunci tasta este apăsată și sărim la rutina ei

**JMP TASTATURA ;** se reia baleierea

#### **TASTA0:**

**CALL DELAY ;** se așteaptă stabilizarea contactelor

**AST0: IN AL, DX ;** se citește din nou starea rândurilor

**AND AL, 08H ;** se testează dacă tasta rămâne apăsată

**JZ AST0 ;** dacă tasta este apăsată, așteptăm până se dezactivează

**CALL DELAY ;** se introduce un mic delay pentru a preveni bounce-ul tastei

**MOV CL, 00H ;** se setează valoarea corespunzătoare tastei 0 în CL

**RET ;** marchează finalul subrutinei

#### **TASTA1:**

**CALL DELAY ;** se așteaptă stabilizarea contactelor

**AST1: IN AL, DX ;** se citește din nou linia și se așteaptă dezactivarea tastei

**AND AL, 01H ;** se testează dacă tasta rămâne apăsată

**JZ AST1 ;** dacă tasta este apăsată, așteptăm până se dezactivează

**CALL DELAY ;** se introduce un mic delay pentru a preveni bounce-ul tastei

**MOV CL, 01H ;** se setează valoarea corespunzătoare tastei 1 în CL

**RET ;** marchează finalul subrutinei

**TASTA2:**

**CALL DELAY** ; se așteaptă stabilizarea contactelor

**AST2:**

**IN AL, DX** ; se citește din nou linia și se așteaptă dezactivarea tastei

**AND AL, 01H** ; se testează dacă tasta rămâne apăsată

**JZ AST2** ; dacă tasta este apăsată, așteptăm până se dezactivează

**CALL DELAY** ; se introduce un mic delay pentru a preveni bounce-ul tastei

**MOV CL, 02H** ; se setează valoarea corespunzătoare tastei 2 în CL

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

**TASTA3:**

**CALL DELAY** ; se așteaptă stabilizarea contactelor

**AST3:**

**IN AL, DX** ; se citește din nou linia și se așteaptă dezactivarea tastei

**AND AL, 01H** ; se testează dacă tasta rămâne apăsată

**JZ AST3** ; dacă tasta este apăsată, așteptăm până se dezactivează

**CALL DELAY** ; se introduce un mic delay pentru a preveni bounce-ul tastei

**MOV CL, 03H** ; se setează valoarea corespunzătoare tastei 3 în CL

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

**TASTA4:**

**CALL DELAY** ; se așteaptă stabilizarea contactelor

**AST4:**

**IN AL, DX** ; se citește din nou linia și se așteaptă dezactivarea tastei

**AND AL, 02H** ; se testează dacă tasta rămâne apăsată

**JZ AST4** ; dacă tasta este apăsată, așteptăm până se dezactivează

**CALL DELAY** ; se introduce un mic delay pentru a preveni bounce-ul tastei

**MOV CL, 04H** ; se setează valoarea corespunzătoare tastei 4 în CL

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

**TASTA5:**

**CALL DELAY** ; se așteaptă stabilizarea contactelor

**AST5:**

**IN AL, DX** ; se citește din nou linia și se așteaptă dezactivarea tastei

**AND AL, 02H** ; se testează dacă tasta rămâne apăsată

**JZ AST5** ; dacă tasta este apăsată, așteptăm până se dezactivează

**CALL DELAY** ; se introduce un mic delay pentru a preveni bounce-ul tastei

**MOV CL, 05H** ; se setează valoarea corespunzătoare tastei 5 în CL

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

**TASTA6:**

**CALL DELAY** ; se așteaptă stabilizarea contactelor

**AST6:**

**IN AL, DX** ; se citește din nou linia și se așteaptă dezactivarea tastei

**AND AL, 02H** ; se testează dacă tasta rămâne apăsată

**JZ AST6** ; dacă tasta este apăsată, așteptăm până se dezactivează

**CALL DELAY** ; se introduce un mic delay pentru a preveni bounce-ul tastei

**MOV CL, 06H** ; se setează valoarea corespunzătoare tastei 6 în CL

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

**TASTA7:**

**CALL DELAY** ; se așteaptă stabilizarea contactelor

**AST7:**

**IN AL, DX** ; se citește din nou linia și se așteaptă dezactivarea tastei

**AND AL, 04H** ; se testează dacă tasta rămâne apăsată

**JZ AST7** ; dacă tasta este apăsată, așteptăm până se dezactivează

**CALL DELAY** ; se introduce un mic delay pentru a preveni bounce-ul tastei

**MOV CL, 07H** ; se setează valoarea corespunzătoare tastei 7 în CL

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

**TASTA8:**

**CALL DELAY** ; se așteaptă stabilizarea contactelor

**AST8:**

**IN AL, DX** ; se citește din nou linia și se așteaptă dezactivarea tastei

**AND AL, 04H** ; se testează dacă tasta rămâne apăsată

**JZ AST8** ; dacă tasta este apăsată, așteptăm până se dezactivează

**CALL DELAY** ; se introduce un mic delay pentru a preveni bounce-ul tastei

**MOV CL, 08H** ; se setează valoarea corespunzătoare tastei 8 în CL

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

**TASTA9:**

**CALL DELAY** ; se așteaptă stabilizarea contactelor

**AST9:**

**IN AL, DX** ; se citește din nou linia și se așteaptă dezactivarea tastei

**AND AL, 04H** ; se testează dacă tasta rămâne apăsată

**JZ AST9** ; dacă tasta este apăsată, așteptăm până se dezactivează

**CALL DELAY** ; se introduce un mic delay pentru a preveni bounce-ul tastei

**MOV CL, 09H** ; se setează valoarea corespunzătoare tastei 9 în CL

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

**TASTA\_STEA:**

**CALL DELAY** ; se așteaptă stabilizarea contactelor

**ASTA:**

**IN AL, DX** ; se citește din nou starea rândurilor

**AND AL, 08H** ; se testează dacă tasta rămâne apăsată

**JZ ASTA** ; dacă tasta este apăsată, așteptăm până se dezactivează

**CALL DELAY** ; se introduce un mic delay pentru a preveni bounce-ul tastei

**MOV CL, 0AH** ; se setează valoarea corespunzătoare tastei \* în CL

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

**TASTA\_HASTAG:**

**CALL DELAY** ; se așteaptă stabilizarea contactelor

**ASTB:**

**IN AL, DX** ; se citește din nou starea rândurilor

**AND AL, 08H** ; se testează dacă tasta rămâne apăsată

**JZ ASTB** ; dacă tasta este apăsată, așteptăm până se dezactivează

**CALL DELAY** ; se introduce un mic delay pentru a preveni bounce-ul tastei

**MOV CL, 0BH** ; se setează valoarea corespunzătoare tastei # în CL

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

### 3.7 Rutina de aprindere a unui led

; Anodul este comun, deci LED-ul va fi aprins la „0”

**APRINDERE\_LED1:** ; subrutină ce aprinde primul LED

**MOV DX, 0D20H** ; setăm adresa portului de ieșire (pentru primul grup de LED-uri) în DX

**MOV AL, 11111110B** ; primul bit va fi „0” (= aprins), restul pe „1”

**OUT DX, AL** ; transmitem conținutul lui AL către portul DX

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

**APRINDERE\_LED9:** ; subrutină ce aprinde LED-ul 9

**MOV DX, 0D28H** ; setăm adresa portului de ieșire (pentru al doilea grup de LED-uri) în DX

**MOV AL, 11111110B** ; primul bit va fi „0” (= aprins), restul pe „1”

**OUT DX, AL** ; transmitem conținutul lui AL către portul DX

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

**APRINDERE\_LED1-6:** ; subrutină ce aprinde primele 6 LED-uri

**MOV DX, 0D20H** ; setăm adresa portului de ieșire (pentru primul grup de LED-uri) în DX

**MOV AL, 11000000B** ; primii 6 biți vor fi „0” (= aprins), restul pe „1”

**OUT DX, AL** ; transmitem conținutul lui AL către portul DX

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

**APRINDERE\_LED1-8:** ; subrutină ce aprinde primele 8 LED-uri

**MOV DX, 0D20H** ; setăm adresa portului de ieșire (pentru primul grup de LED-uri) în DX

**MOV AL, 00H** ; punem toți biții pe „0”, aprinzând primele 8 LED-uri

**OUT DX, AL** ; transmitem conținutul lui AL către portul DX

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

### 3.8 Rutina de stingere a unui led

; Anodul este comun, deci LED-ul va fi stins la „1”

**STINGERE\_LED1-8:** ; subrutină ce stinge primele 8 LED-uri

**MOV DX, 0D20H** ; setăm adresa portului de ieșire (pentru primul grup de LED-uri) în DX

**MOV AL, FFH** ; punem toți biții pe „1”, stingând primele 8 LED-uri

**OUT DX, AL** ; transmitem conținutul lui AL către portul DX

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant

**STINGERE\_LED9-10:** ; subrutină ce stinge LED-urile 9 și 10

**MOV DX, 0D28H** ; setăm adresa portului de ieșire (pentru primul grup de LED-uri) în DX

**MOV AL, FFH** ; punem toți biții pe „1”, stingând LED-urile 9 și 10

**OUT DX, AL** ; transmitem conținutul lui AL către portul DX

**RET** ; marchează finalul subrutinei și transferul controlului înapoi la apelant



### 3.9 Rutina de afișare a unui caracter hexa pe un rang cu segmente

; Intrare: AH - rangul pe care se va afișa caracterul

; CL - caracterul ce va fi afișat

**SET\_RANG:** ; determinăm rangul pe care se va face afișarea

**CMP AH, 1** ; verificăm dacă rangul este 1

**JE RANG1**

**CMP AH, 2** ; verificăm dacă rangul este 2

**JE RANG2**

**CMP AH, 3** ; verificăm dacă rangul este 3

**JE RANG3**

**CMP AH, 4** ; verificăm dacă rangul este 4

**JE RANG4**

**CMP AH, 5** ; verificăm dacă rangul este 5

**JE RANG5**

**CMP AH, 6** ; verificăm dacă rangul este 6

**JE RANG6**

**RANG1:** ; îi atribuim lui DX adresa corespunzătoare rangului 1

**MOV DX, 0E00H**

**JMP AFISARE**

**RANG2:** ; îi atribuim lui DX adresa corespunzătoare rangului 2

**MOV DX, 0E08H**

**JMP AFISARE**

**RANG3:** ; îi atribuim lui DX adresa corespunzătoare rangului 3

**MOV DX, 0E10H**

**JMP AFISARE**

**RANG4:** ; îi atribuim lui DX adresa corespunzătoare rangului 4

**MOV DX, 0E18H**

**JMP AFISARE**

**RANG5:** ; îi atribuim lui DX adresa corespunzătoare rangului 5

**MOV DX, 0E20H**

**JMP AFISARE**

**RANG6:** ; îi atribuim lui DX adresa corespunzătoare rangului 6

**MOV DX, 0E28H**

**JMP AFISARE**

**AFISARE:** ; testăm caracterul din CL

**CMP CL, 00H ; verificăm dacă caraterul este 0**

**JE AFIS\_0**

**CMP CL, 01H ; verificăm dacă caraterul este 1**

**JE AFIS\_1**

**CMP CL, 02H ; verificăm dacă caraterul este 2**

**JE AFIS\_2**

**CMP CL, 03H ; verificăm dacă caraterul este 3**

**JE AFIS\_3**

**CMP CL, 04H ; verificăm dacă caraterul este 4**

**JE AFIS\_4**

**CMP CL, 05H ; verificăm dacă caraterul este 5**

**JE AFIS\_6**

**CMP CL, 06H ; verificăm dacă caraterul este 6**

**JE AFIS\_6**

**CMP CL, 07H ; verificăm dacă caraterul este 7**

**JE AFIS\_7**

```
CMP CL, 08H ; verificăm dacă caraterul este 8
JE AFIS_8

CMP CL, 09H ; verificăm dacă caraterul este 9
JE AFIS_9

CMP CL, 0AH ; verificăm dacă caraterul este A
JE AFIS_A

CMP CL, 0BH ; verificăm dacă caraterul este B
JE AFIS_B

CMP CL, 0CH ; verificăm dacă caraterul este C
JE AFIS_C

CMP CL, 0DH ; verificăm dacă caraterul este D
JE AFIS_D

CMP CL, 0EH ; verificăm dacă caraterul este E
JE AFIS_E

CMP CL, 0FH ; verificăm dacă caraterul este F
JE AFIS_F
```

```
AFIS_0: ; afișăm caracterul 0
MOV AL, C0H
OUT DX, AL
RET
```

```
AFIS_1: ; afișăm caracterul 1
MOV AL, F9H
OUT DX, AL
RET
```

**AFIS\_2:** ; afișăm caracterul 2

**MOV AL, A4H**

**OUT DX, AL**

**RET**

**AFIS\_3:** ; afișăm caracterul 3

**MOV AL, B0H**

**OUT DX, AL**

**RET**

**AFIS\_4:** ; afișăm caracterul 4

**MOV AL, 99H**

**OUT DX, AL**

**RET**

**AFIS\_5:** ; afișăm caracterul 5

**MOV AL, 92H**

**OUT DX, AL**

**RET**

**AFIS\_6:** ; afișăm caracterul 6

**MOV AL, 82H**

**OUT DX, AL**

**RET**

**AFIS\_7:** ; afișăm caracterul 7

**MOV AL, F8H**

**OUT DX, AL**

**RET**

**AFIS\_8:** ; afișăm caracterul 8

**MOV AL, 80H**

**OUT DX, AL**

**RET**

**AFIS\_9:** ; afișăm caracterul 9

**MOV AL, 90H**

**OUT DX, AL**

**RET**

**AFIS\_A:** ; afișăm caracterul A

**MOV AL, 88H**

**OUT DX, AL**

**RET**

**AFIS\_B:** ; afișăm caracterul B

**MOV AL, 83H**

**OUT DX, AL**

**RET**

**AFIS\_C:** ; afișăm caracterul C

**MOV AL, C6H**

**OUT DX, AL**

**RET**

**AFIS\_D:** ; afișăm caracterul D

**MOV AL, A1H**

**OUT DX, AL**

**RET**

**AFIS\_E:** ; afișăm caracterul E

**MOV AL, 86H**

**OUT DX, AL**

**RET**

**AFIS\_F:** ; afișăm caracterul F

**MOV AL, 8EH**

**OUT DX, AL**

**RET**

## **4. ANEXE**

Anexa 1 – Unitatea centrală

Anexa 2 – Conectarea memorilor

Anexa 3 – Interfața serială și paralelă

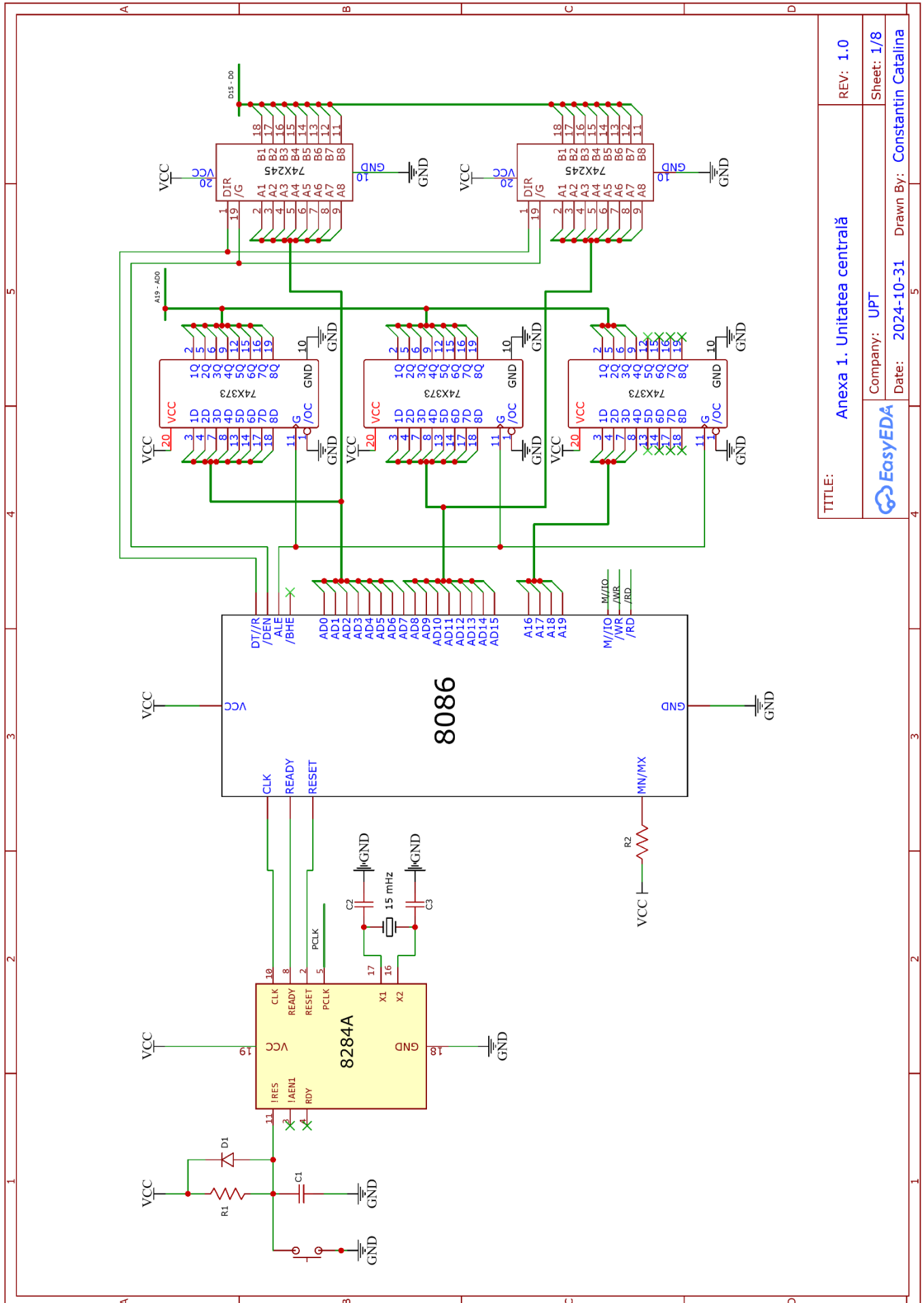
Anexa 4 – Decodificator de porturi

Anexa 5 – Minitastatura

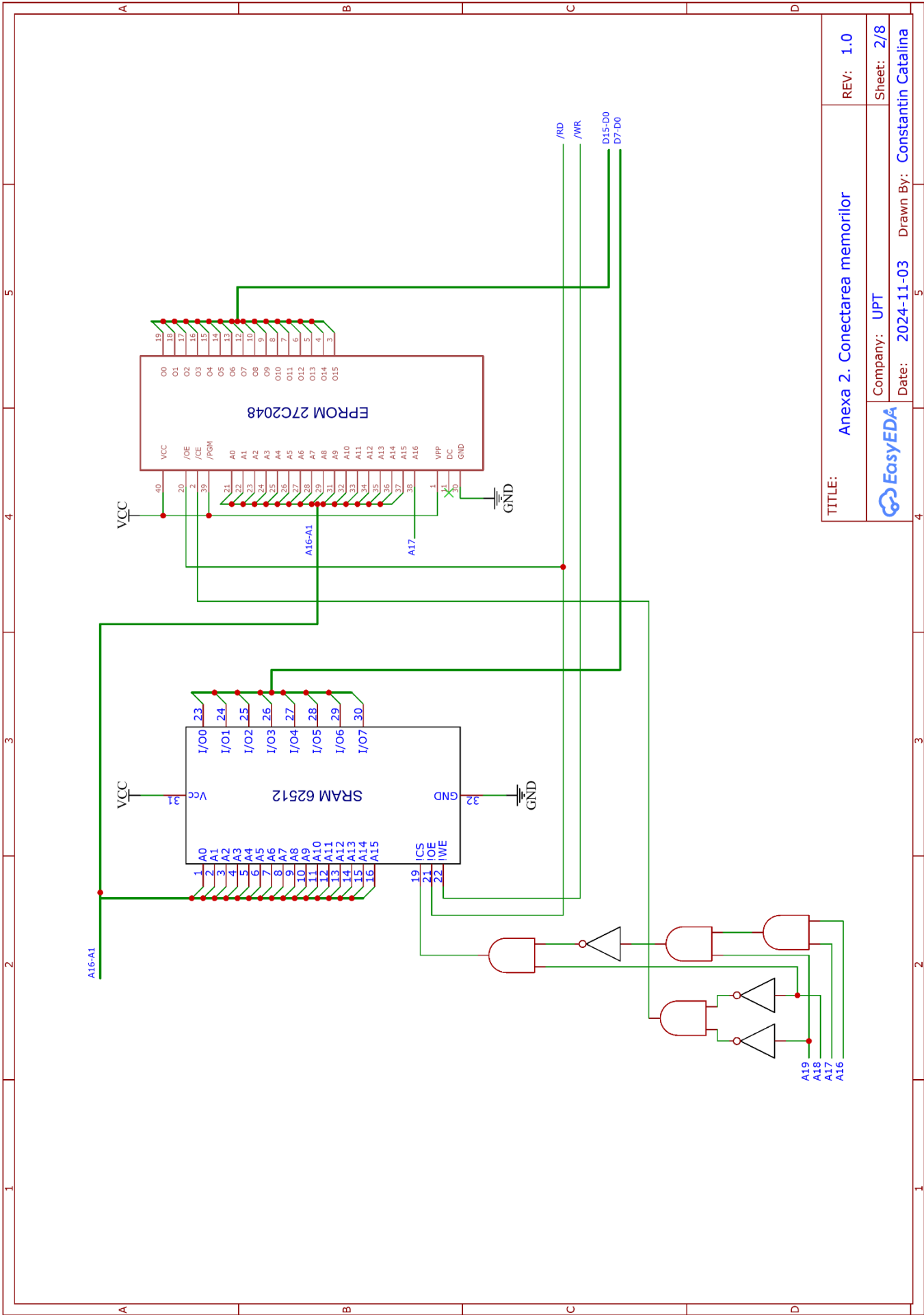
Anexa 6 – Led-uri

Anexa 7 – Afișaj cu segmente

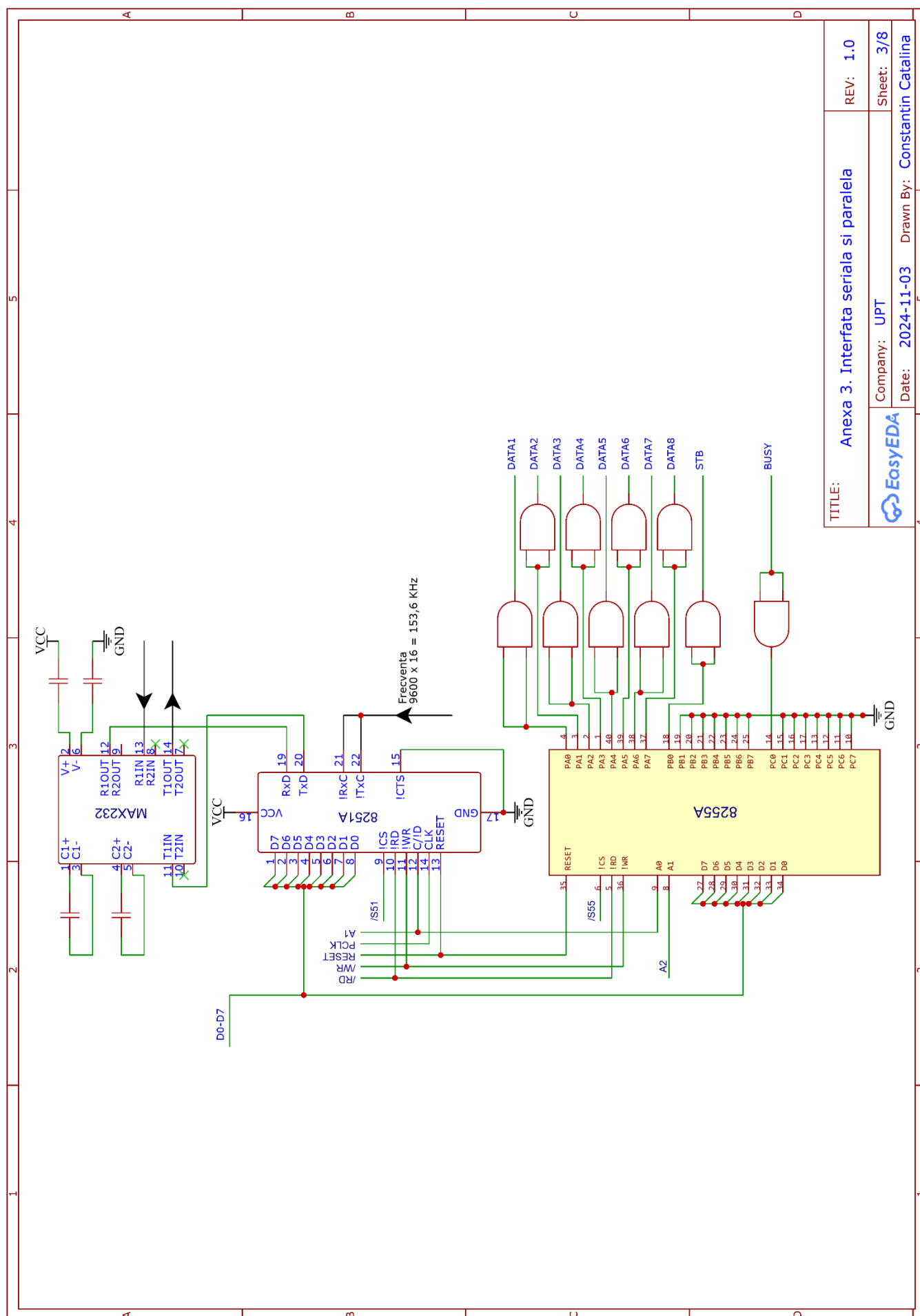
Anexa 8 – Microsistem cu microprocesorul 8086

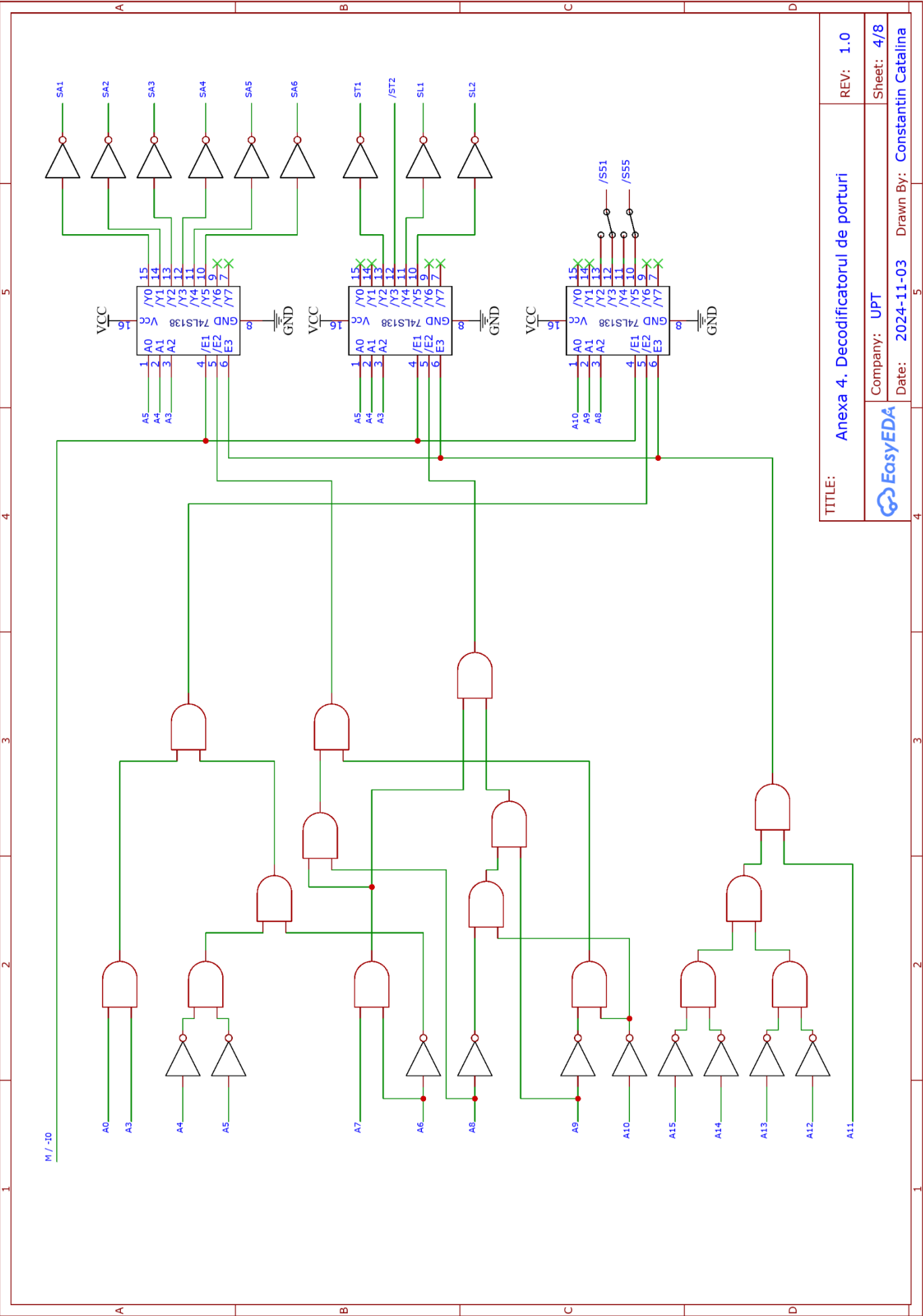




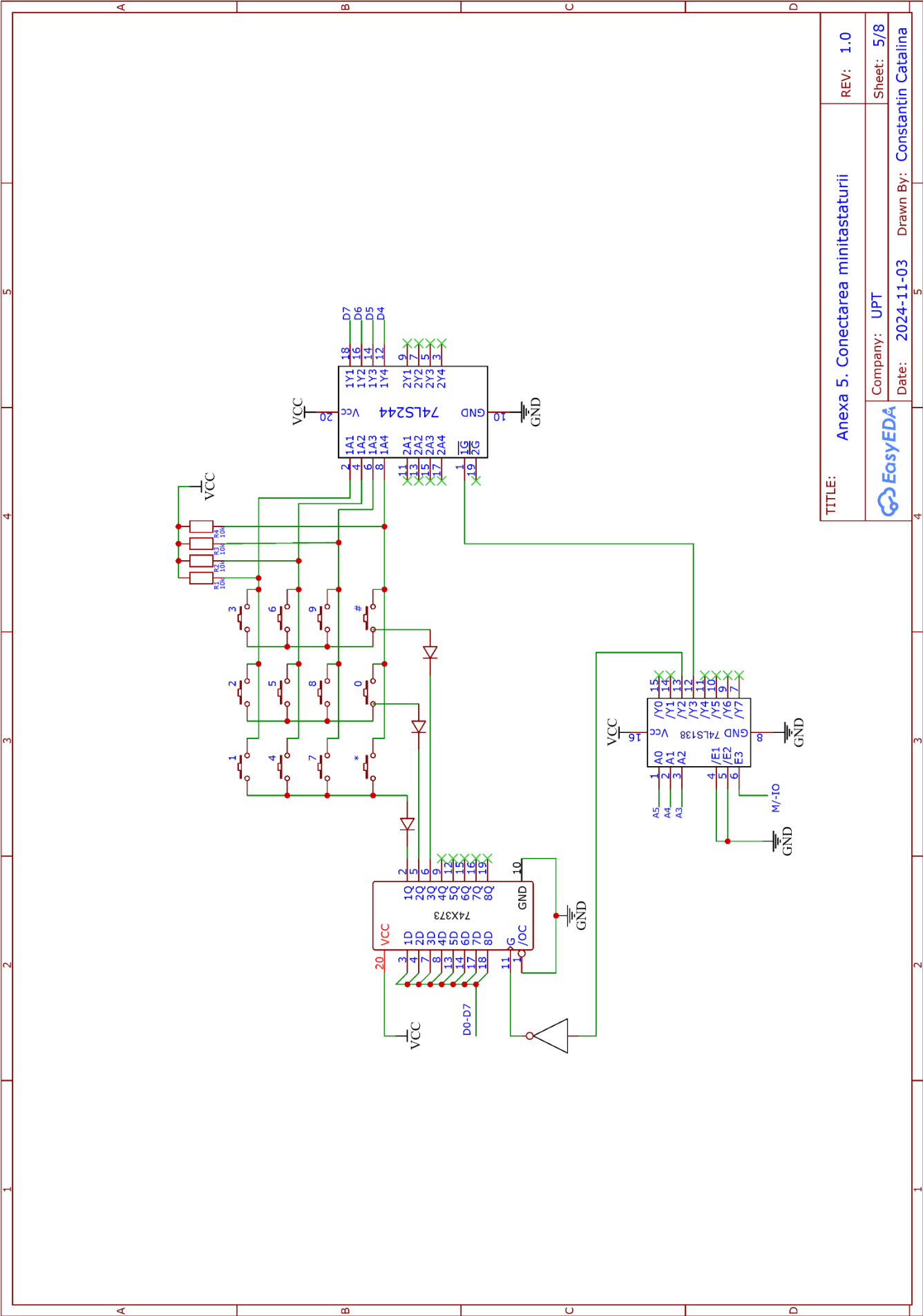


TITLE: Anexa 2. Conectarea memorilor		REV: 1.0
Company: UPT		Sheet: 2/8
Date: 2024-11-03	Drawn By: Constantin Catalina	

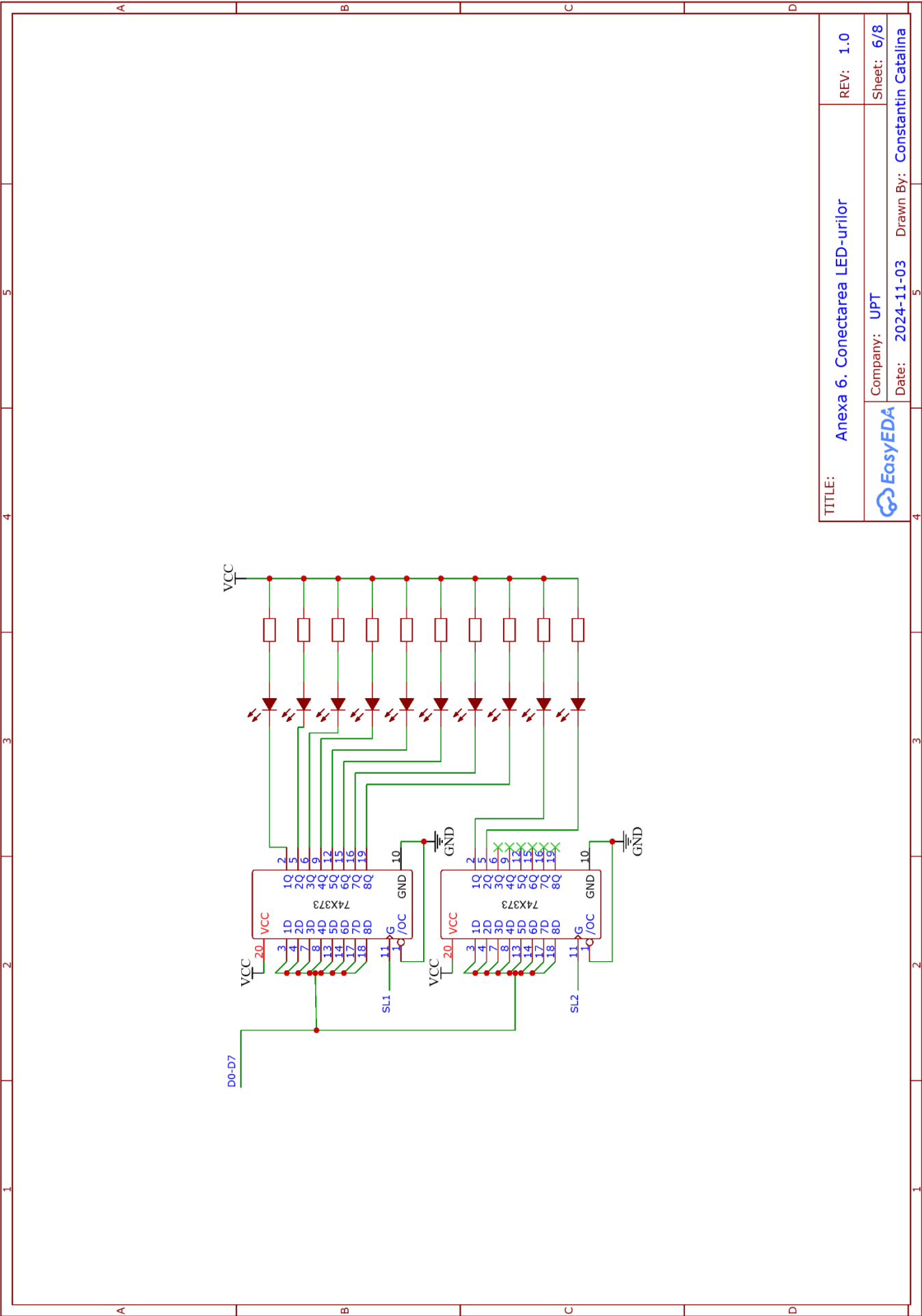




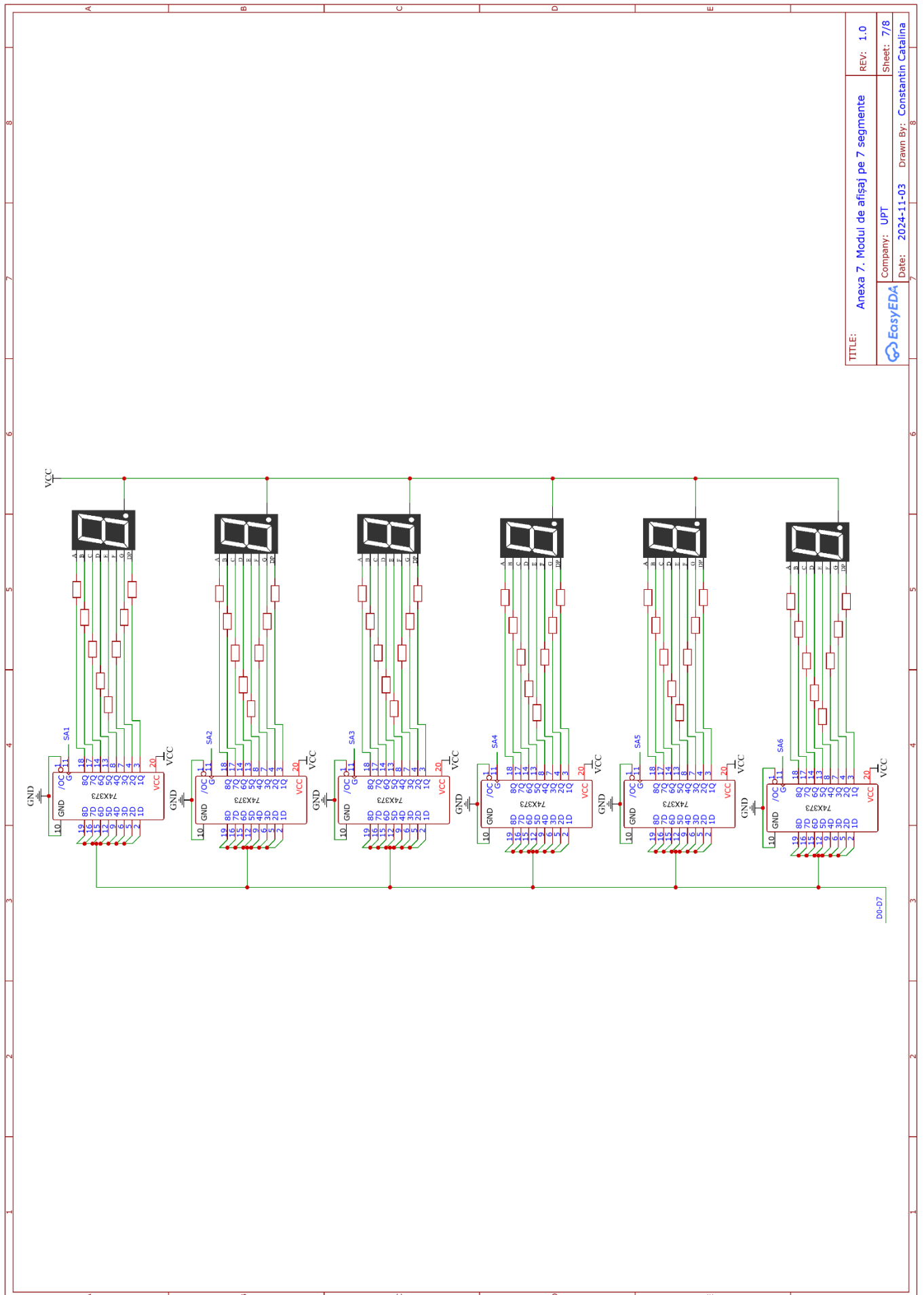
TITLE: Anexa 4. Decodificatorul de porturi		REV: 1.0
Company: UPT		Sheet: 4/8
Date: 2024-11-03	Drawn By: Constantin Catalina	

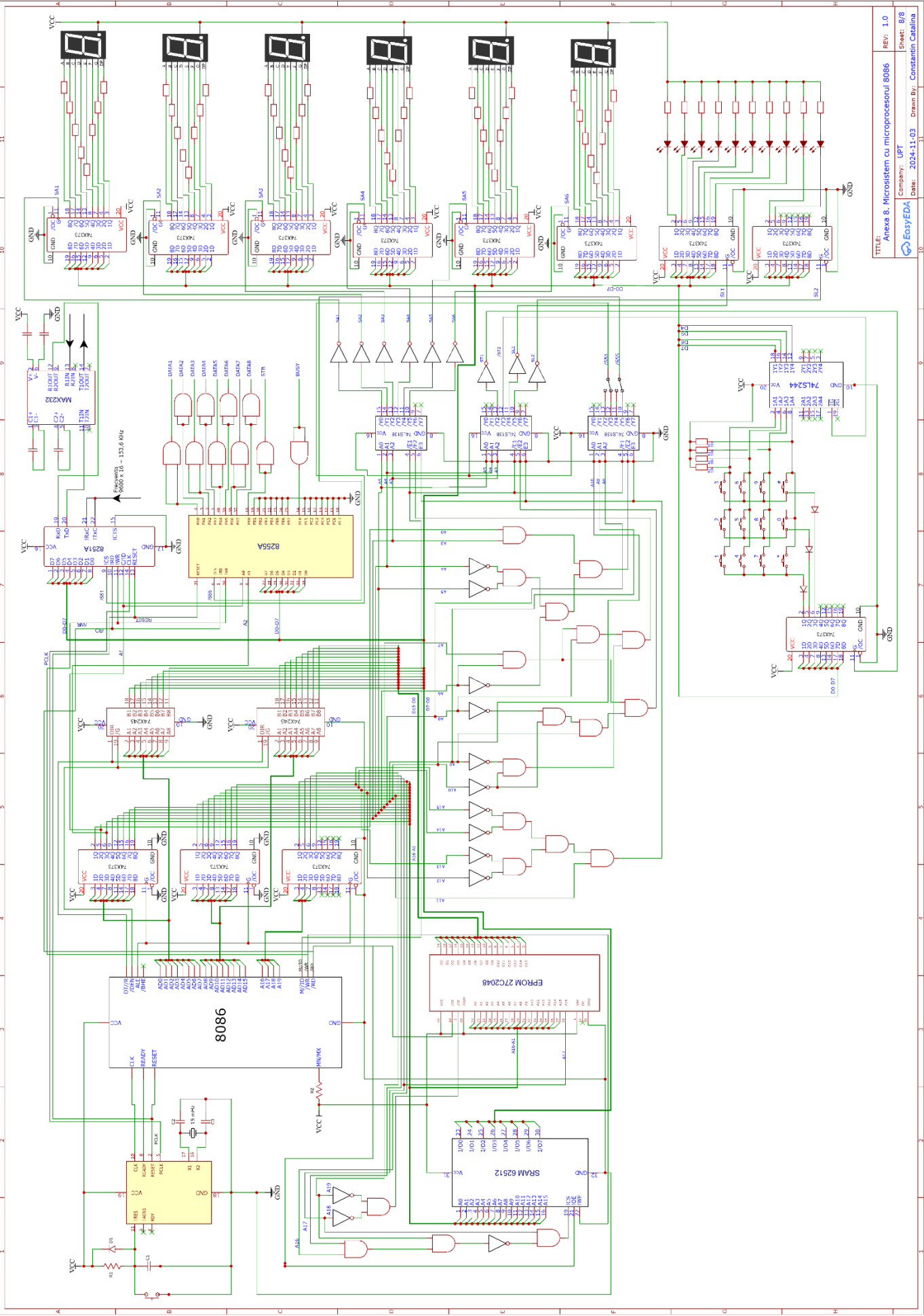


TITLE: Anexa 5. Conectarea minitastaturii	REV: 1.0
	Sheet: 5/8
	Company: UPT
Date: 2024-11-03	Drawn By: Constantin Catalina



TITLE: Anexa 6. Conectarea LED-urilor	REV: 1.0
	Sheet: 6/8
EasyEDA	Company: UPT
	Date: 2024-11-03 Drawn By: Constantin Catalina





## 5. BIBLIOGRAFIE ȘI RESURSE ONLINE

- M. Popa, “Proiectarea microsistemelor digitale”, Orizonturi Universitare, Timișoara, 2003
- M. Popa, “Sisteme cu microprocesoare”, Orizonturi Universitare, Timișoara, 2003
- [https://faculty.ksu.edu.sa/sites/default/files/part2\\_8284a.pdf](https://faculty.ksu.edu.sa/sites/default/files/part2_8284a.pdf)
- [http://discipline.elcom.pub.ro/amp/amp\\_1\\_1.pdf](http://discipline.elcom.pub.ro/amp/amp_1_1.pdf)
- <https://www.alldatasheet.com/html-pdf/74472/MCNIX/27C2048/126/1/27C2048.html>
- <https://www.microchip.com/content/dam/mchp/documents/OTH/ProductDocuments/DataSheets/doc0632.pdf>
- <https://sites.google.com/site/labpmd/Laborator/conectarea-memoriilor>
- <https://www.electrokits.ro/memoria-eprom-erasable-programmable-read-only-memory/>
- <https://ro.wikipedia.org/wiki/EPROM>
- <https://ro.wikipedia.org/wiki/SRAM>
- [https://ro.wikipedia.org/wiki/Afi%C8%99aj\\_cu\\_%C8%99apte\\_segmente](https://ro.wikipedia.org/wiki/Afi%C8%99aj_cu_%C8%99apte_segmente)
- <https://www.tme.eu/ro/news/library-articles/page/56008/afiaje-led-cu-7-segmente-ce-trebuie-sa-titi-despre-ele/>
- <https://sites.google.com/site/bazeleelectronicii/home/circuite-diverse/10-afisaj-cu-7-elemente>