2. Portul digital de intrare-ieșire

Regiștrii importanți
Portul digital de intrare
Portul digital de ieșire
Aplicații cu porturi digitale de intrare-ieșire

Porturile digitale de intrare-ieșire, referite în continuare ca porturi I/O, reprezintă capacitatea unui microcontroler de a transmite sau monitoriza stări logice în circuitul electric în care acesta este folosit. În general, un microcontroler deține 2 sau 3 porturi I/O care pot fi conectate direct la circuitul electric din care face parte, bineînțeles cu respectarea specificaţiilor electrice ale microcontrolerului. În cadrul laboratorului se va folosi Microcontrolerul Atmega328P. Acesta deține 28 de pini metalici. Dintre aceștia, 23 pot fi folosiți ca pini I/O, restul de 5 pini fiind pini cu funcţii specifice: alimentare (GND și VCC), respectiv referință (AREF). Fiecare pin din cei 23 reprezintă un singur canal de legătură cu exteriorul.

Din punct de vedere software, un port I/O este reprezentat prin intermediul unui byte, adică este format din 8 biți. Fiecare bit al fiecărui port I/O este conectat la un pin metalic al integratului microcontrolerului. Acesta poate fi folosit ca ieșire, ca intrare sau ca intrare/ieșire. Din considerente de optimizare, pe lângă funcția de intrare/ieșire digitală, un pin metalic poate îndeplini funcționalități precum: întrerupere, comunicație serială, convertor analog-digital, etc (vezi figura 2.1). Aceste funcționalități nu pot fi folosite în paralel pe același pin. Este de datoria utilizatorului să asigure că celelalte funționalități paralele sunt dezactivate atunci când pinul este folosit într-un anumit mod. De exemplu, pentru a folosi pinul 2 al microcontrolerului Atmega328P, utilizatorul trebuie să asigure dezactivarea modulului de comunicație serială, deoarece pinul 0 al Portului D (PD0) mai poate îndeplini și funcția de recepție semnal serial (RxD).

Termenul *digital* se referă la faptul că un microcontroler folosește cele 2 nivele logice 1, respectiv 0 pentru a defini o stare. Circuitul electric în care microcontrolerul este utilizat are la bază o tensiune electrică, care, în general, este de 5V sau 3.3V. Așadar, un pin al microcontrolerului poate avea tensiunea de 5V sau 0V (GND). Microcontrolerul digitalizează această valoare prin maparea ei la una dintre cele două stări logice (1, respectiv 0). Așadar, din perspectiva microcontrolerului, orice tensiune peste valoarea de 2V va fi mapată la starea

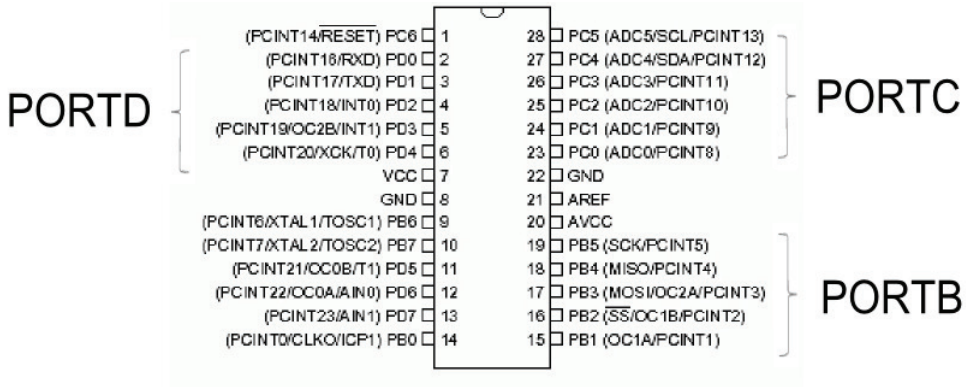

Fig. 2.1 Porturile I/O a microcontrolului Atmega328P.

logică 1 (eng. *high*) pentru tehnologia TTL, respectiv orice tensiune sub valoare de 0.8V (GND) va fi mapată către starea logică 0 (eng. *low*). Pentru orice tensiune între cele 2 praguri menționate, pinul microcontrolerului se va afla în starea de înaltă impedanță.

2.1 Regiștrii portului de intrare-ieșire

Un microcontroler deține multiple resurse. Porturile I/O sunt un exemplu. Fiecare resursă poate fi utilizată în mai multe feluri și cu diferiți parametrii, viteze, etc.. Regiștri sunt elementele ce permit configurarea funcționalităților resurselor unui microcontroler. Intuitiv, un registru poate fi comparat cu un panou cu butoane (vezi figura 2.2).

Fig. 2.2 Panou cu butoane de comandă.

Un buton poate avea două stari: deschis (0 logic) sau închis (1 logic). Închis înseamnă că funcționalitatea la care este atașat butonul este activă. Mai multe butoane cu funcționalități similare pot fi grupate și formează un *registru*. Grupurile se pot forma din multipli de 8 butoane. Un registru este foarte asemănător cu o adresă de memorie, diferența fiind aceea că o memorie stochează un număr, în timp ce un registru stochează o configurație a unei resurse a microcontrolerului. De exemplu, dacă un registru ar arăta ca cel din figura 2.3, microcontrolerul ar fi configurat să activeze pe pini 0, 6 și 7 ai unui port de ieșire starea logică 1, fapt ce ar fi determinat aprindere unor LED-uri. Pentru că utilizăm un sistem digital, un buton este echivalentul unui bit. Practic cele 8 butoane vor constitui 1 byte de configurare, adică un registru.

Configurarea unui port de intrare/ieșire se realizează prin intermediul următorilor regrstrii (vezi figura 2.4):

- PORTx - reprezintă registrul în care se setează starea (1 sau 0) pinilor de ieșire;
- DDRx - reprezintă registrul în care se configurează sensul piniilor portului ca fiind de intrare sau ieșire;

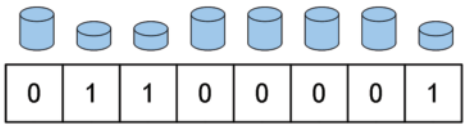

Fig. 2.3 Registru de 8 biți.

– PINx - reprezintă registrul din care se citește starea (1 sau 0) pinilor de intrare.

Pentru că un microcontroler poate avea mai multe porturi, la sfârșitul denumirii fiecărui registru de tip port se adaugă o literă ce identifică unic acel port. De exemplu microcontrolerul Atmega328P are 3 porturi I/O identificate prin literel B, C și D (PORTB, PORTC și PORTD). Așadar, registrul portului D în care se va face configurarea direcției pinilor portului va fi denumit DDRD.

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Fig. 2.4 Regiștrii de configurare ai portului D.

2.2 Portul digital de ieșire

Este folosit de microcontroler pentru a transmite semnale logice către circuitul electric în care acesta este utilizat. Un pin al portului digital de ieșire configurat să furnizeze starea logică 1 asigură existența unei tensiuni de 5 volți și un curent de maxim 40 mA. Aceste caracteristici sunt suficiente pentru a aprinde un LED sau pentru a controla numeroși senzori, însă insuficiente pentru a alimenta un releu sau diverse motare de curent continuu. Scurcicuitarea sau suprasolicitarea pot conduce la defectarea pinilor portului de ieșire. Totuși, această situație nu va afecta restul funcționalităților microcontrloerului, ele putând fi folosite în continuare.

Utilizarea portului digital I/O ca port de ieșire necesită configurarea stării logice 1 pentru fiecare bit din registrul DDRx ce se dorește a fi folosit ca ieșire. Stabilirea unei tensiuni (0V sau 5V) pe pinii de ieșire ai portului microcontrolerului se configurează prin intermediul

registrului PORT. Astfel, configurarea stării logice 1 a unui bit al acestui registru determină aplicarea tensiunii de 5V la pinul de ieșire aferent.

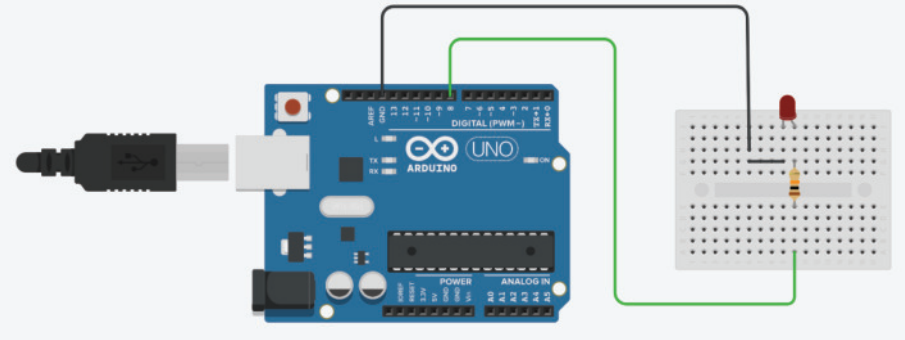

Fig. 2.5 Schema electrică pentru aprinderea unui LED.

2.2.1 Exemplu de aplicaţie

Se consideră schema electrică formata dintr-o placă de dezvoltare Arduino UNO, un LED și un rezistor de 10 KOhmi (vezi figura 2.5). Montajul a fost realizat folosind simulatorul disponibil pe pagina de web *https://www.tinkercad.com*. Obiectivul este acela de a aprinde, respectiv stinge, LED-ul la un interval de 1000 milisecunde.

Programul scris în IDE-ul Arduion 1.8.5 este următorul:

Algoritmul 2.1 Exemplu de utilizare a portului de ieșire al unui microcontroler pentru a aprinde, respectiv a stinge un LED

```
1  void setup()
2  {
3      // stabileste directia pinului 8 al placii Arduino ca pin de
       ↪ iesire
4      pinMode(8, OUTPUT);
5  }
6
7  void loop()
8  {
9      // aplica 1 logic pe  pinul 8 pentru a aprinde LED-ului placii
       ↪  Arduino
10     digitalWrite(8, HIGH);
11     // asteapta 1000 milisecunde(s)
12     delay(1000);
13     // aplica 0 logic pe  pinul 8 pentru a stinge LED-ul placii
       ↪ Arduino
14     digitalWrite(8, LOW);
15     //  Asteapta 1000 milisecunde(s)
16     delay(1000);
17 }
```

În programul anterior au fost realizate următoarele setări:

- linia 4: configurarea pinului 8 ca iesirea (care este conectat la bitul 0 al portului B al microcontrolerului);
- linia 10: configurarea stării logice 1 pentru bitul 0 al portului B;
- linia 14: configurarea stării logice 0 pentru bitul 0 al portului B.

Programul prezentat mai sus conține funcții ce îl ajută pe programator să scrie cod într-o manieră intuitivă, corelată cu placa de dezvoltare Arduino UNO. De exemplu, linia de cod 9 determină setarea tensiunii de 5V pe pinul numărul 8 al plăcii de dezvoltare. Funcţia *digitalWrite* se va ocupa mai departe de legătura acestuia cu pinul microcontrolerului. Practic, programatorul nici nu trebuie să știe la ce pin al microcontrolerului este conectat pinul 8 al plăcii de dezvoltare. Aceeaşi funcționalitate a algoritmului anterior poate fi rescrisă, fără a utiliza aceste funcții, după cum urmează:

Algoritmul 2.2 Exemplu de utilizare a portului de ieșire pentru a aprinde un LED utilizând explicit registrele microcontrolerului.

```
1  #define DDB0            0
2  #define PORTB0          0
3
4  void setup()
5  {
6      // stabileste directia bitului 0 al portului B ca pin de
       ↪ iesire
7      // bitul 0 al portului B este conectat la pinul 8 respectiv
       ↪ LED-ul placii Arduino
8      DDRB = 1 << DDB0;
9  }
10
11 void loop()
12 {
13     // aplica 1 logic pe  pinul 8 ceea ce determine aprinderea LED
       ↪ -ului placii Arduino
14     PORTB = 1 << PORTB0;
15     // asteapta 1000 milisecunde(s)
16     delay(1000);
17     // aplica 0 logic pe  pinul 8 ceea ce determine aprinderea LED
       ↪ -ului placii Arduino
18     PORTB = 0 << PORTB0;
19     //  asteapta 1000 milisecunde(s)
20     delay(1000);
21 }
```

În programul anterior au fost realizate următoarele setări:

– linia 1: redefinirea pinului 0 din registrul DDRB, sub numele de *DDB0*;
– linia 2: redefinirea pinului 0 din registrul PORTB, sub numele de *PORTB0*;
– linia 8: configurarea lui DDB0 (bitul 0) ca pin de ieșire în registrul de direcție DDRB;
– linia 14: configurarea stării logice 1 pentru PORTB0 (bitul 0) în registrul PORTB;
– linia 18: configurarea stării logice 0 pentru PORTB0 (bitul 0) în registrul PORTB.

2.3 Portul digital de intrare

Este folosit de microcontroler pentru a monitoriza/citi semnale din circuitul electric în care acesta este utilizat. Cel mai des, pinii porturilor de intrare sunt folosiți pentru a evalua starea unor butoane (închis/deschis). Două registre sunt importante în acest sens: registrul DDR și registrul PIN. Configurând valoarea 0 pe un bit al registrului DDR al unui port, îi spunem microcontrolerului că dorim să utilizam acel pin, ca pin de intrare. Citirea stări/valorii logice a pinului portului se realizează prin intermediul registrului PIN corespunzător (vezi figura 2.4).

2.3.1 Exemplu de aplicaţie

Se consideră schema electrică formată dintr-o placă Arduino UNO, un LED, un rezistor și un buton fără reținere (vezi figura 2.6). Montajul a fost realizat folosind simulatorul disponibil pe pagina de web *https://www.tinkercad.com*. Obiectivul este acela de a aprinde, respectiv a stinge un LED la fiecare apăsare a butonului.

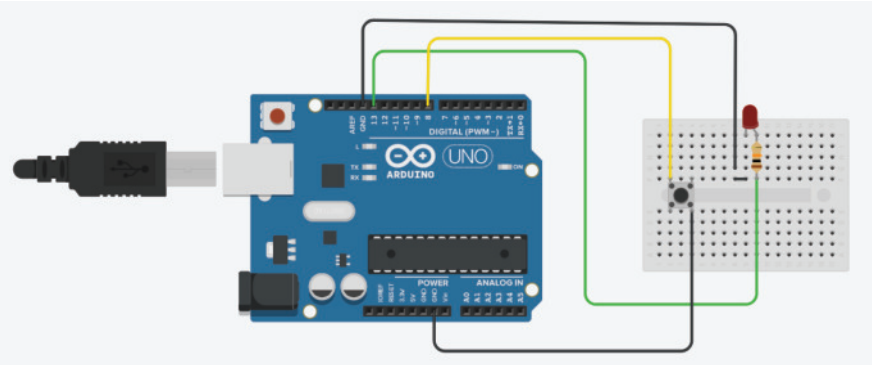

Fig. 2.6 Schema electrică pentru citirea stării unui buton.

Citirea stării butonului se face pe pinul numărul 8, de pe placa de dezvoltare. Conform cablajului circuitului plăcii de dezvoltare, pinul 8 este conectat la bit-ul 0 al portului B al microcontrolerului. Astfel, când butonul este apăsat, starea logică a pin-ului este 0 (0V). Când acesta este relaxat, starea logică a pinului este necunoscută. Practic, pe firul galben din figura 2.6 nu avem nici 5V și nici 0V (stare de înaltă impedanță). Pentru a evita această situație se va folosi un rezistor care are rolul de a menține o stare logică cunoscută atunci când butonul nu este apăsat, de exemplu 1 logic dacă rezistorul este conectat la 5V. Acest rezistor poartă denumirea de rezistor de *pull-up*. Valoarea rezistorului trebuie să fie de aproximativ 20 KOhmi. Microcontrolerul Atmega328P are încapsulat în integrat-ul lui un astfel de rezistor pentru fiecare pin al fiecărui port de intrare. Acești rezistori pot fi activați prin configurarea stării logice 1 în bitul corespunzător al registrului PORT (vezi

figura următoare).

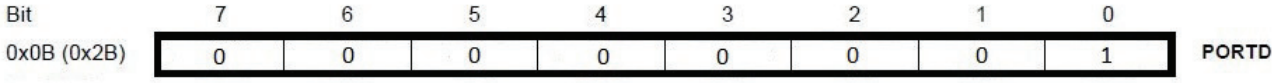

Fig. 2.7 Activarea rezistorului de pull-up pentru pinul 0 al portului D.

Programul scris în IDE-ul Arduino 1.8.5 este următorul:

Algoritmul 2.3 Exemplu de utilizare a portului de intrare pentru a citi starea unui buton și a aprinde, respectiv a stinge un LED

```
1  // pin-ul conectat la buton
2  const byte pinButon = 8;
3  // pin-ul conectat la LED
4  const byte pinLED = 13;
5
6  // starea curenta a LED-ului
7  byte stareLED = LOW;
8
9  void setup()
10 {
11     // stabileste directie pin ca intrare
12     pinMode(pinButon, INPUT);
13     // stabileste directie pin ca iesire
14     pinMode(pinLED, OUTPUT);
15
16     // activeaza registru de pull-up
17     digitalWrite(pinButon, HIGH);
18     // setare stare initiala LED
19     digitalWrite(pinLED, stareLED);
20 }
21
22 void loop()
23 {
24     // se verifica daca butonul a fost apasat,
25     // adica daca pin-ul e in starea LOW
26     if (digitalRead(pinButon) == LOW)
27     {
28         // asteptam ca butonul sa nu mai fie apasat
29         // inainte de a schimba starea, adica asteptam
30         // atat timp cat pin-ul e in starea LOW
31         while (digitalRead(pinButon) == LOW)
```

```
32          {
33              // asteptare...
34          }
35
36          // basculare stare LED
37          stareLED = !stareLED;
38          // setare stare LED
39          digitalWrite(pinLED, stareLED);
40      }
41  }
```

În programul anterior au fost realizate următoarele setări:

- linia 2: definirea variabilei ce configurează legătura dintre placa de dezvoltare, pe care se află conectat LED-ul, și pinul 5 al portului B al microcontrolerului;
- linia 4: definirea variabilei ce configurează legătura dintre placa de dezvoltare și pinul 0 al portului B al microcontrolerului;
- linia 7: variablila pentru a stoca starea LED-ului;
- linia 12: configurarea bitului 5 al registrului DDRB, ca pin de ieşire;
- linia 14: configurarea bitului 0 al registrului DDRB, ca pin de intrare;
- linia 17: configurarea stării logice 1 pentru bitul 0 al registrului PORTB, pentru a activa rezistorul de pull-up;
- linia 19: configurarea stării inițiale a LED-ului.

Programul anterior poate fi rescris, fără a utiliza funcții specifice Arduino, după cum urmează:

Algoritmul 2.4 Exemplu de utilizare a portului de intrare pentru a citi starea unui buton și a aprinde, respectiv a stinge un LED, utilizând explicit registrele microcontrolerului.

```
1  #define PIN0     0
2  #define PIN5     5
3
4  byte stareLED = 0;
5
6  void setup()
7  {
8      // stabileste directiile celor doi pini,
9      // buton (PIN0) pe input si LED (PIN5) pe output
10     DDRB = (1 << PIN5) | (0 << PIN0);
11
12     // activeaza registru pull-up si setare
13     // stare initiala pentru LED
14     PORTB = (stareLED << PIN5) | (1 << PIN0);
15 }
```

```
16
17 void loop()
18 {
19     // se verifica daca butonul a fost apasat,
20     // adica daca pin-ul e in starea LOW (0)
21     if ((PINB & (1 << PIN0)) == 0)
22     {
23         // asteptam ca butonul sa nu mai fie apasat
24         // inainte de a schimba starea, adica asteptam
25         // atat timp cat pin-ul e in starea LOW (0)
26         while ((PINB & (1 << PIN0)) == 0)
27         {
28             // asteptare...
29         }
30
31         stareLED = !stareLED; // basculare stare LED
32         // resetare stare LED anterioara
33         PORTB &= ~(1 << PIN5);
34         // setare stare LED actuala
35         PORTB |= stareLED << PIN5;
36     }
37 }
```

2.4 Aplicații cu porturi digitale de intrare-ieșire

În cadrul acestui sub-capitol vor fi prezentate trei dintre cele mai utile moduri de folosire a porturilor digitale de intrare / ieșire ale unui microcontroler. Pentru fiecare dintre acestea se va prezenta: schema electrică, programul software aferent și o serie de explicații utile.

2.4.1 Utilizarea porturilor digitale de ieșire pentru comanda afişajelor cu 7 segmente

Afișarea valorii temperaturii, afișarea orei, respectiv a minutului unui ceas digital, afișarea valorii unui numărător, sunt doar câteva aplicații în care un microcontroler trebuie să transmită informații către utilizator, prin intermediul un dispozitiv de afișare. O soluție este afișajul cu 7 segmente. Acesta poate fi controlat prin intermediul porturilor digitale de ieșire. Practic, 8 leduri sunt așezate pentru a descrie vizual o valoare numerică între 0 și 9 (vezi figura următoare).

Modul de conectare al unui afișaj cu 7 segmente la un microcontroler, precum și modul de configurare al registrului PORT pentru a afișa valoarea 5, este prezentat în figura următoare:

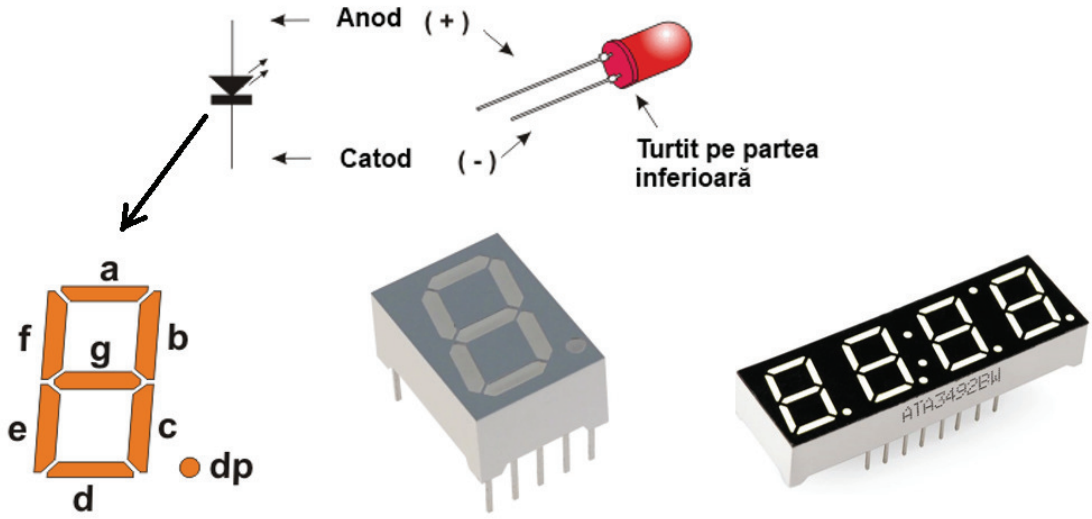

Fig. 2.8 Afișaj cu 7 segmente. (sus): o diodă LED. (jos): stânga: adnotarea segmenelor; (jos) mijloc: afișaj pentru o singură valoare; (jos) dreapta: afișaj pentru 4 valori.

2.4.1.1 Exemplu de aplicaţie

Se propune o aplicație care să contorizeze număul de pasageri ai unui avion. Îmbarcarea unui pasager va fi simulată prin apăsarea unui buton care va incrementa numărul de pasageri. Omolog, debarcarea va fi simulată prin apăsarea unui al doilea buton care va decrementa numărul de pasageri. Resetarea numărului de pasageri se va face la apăsarea celui de-al treilea buton. Se consideră că, într-un avion pot încăpea aproximativ 200 de pasageri. Pentru a afișa o astfel de valoare vor fi necesare 3 module cu 7 segmente. Astfel. pentru a controla toate aceste 7 segmente vor fi necesari 3x7 = 21 pini de ieșire. Majoritatea microcontrolerelor nu beneficiază de aşa mulți pini de ieșire, motiv pentru care toate cele trei module cu 7 segmente vor fi conectate în paralel la un singur port de ieșire. Practic, va fi ca o magistrală pe care se vor transmite valori, urmând ca fiecare afișaj să fie activat, pe rand, prin intermediul unui tranzistor care va controla catodul modului cu 7 segmente. Datorită frecvenței mari de activare, respectiv dezactivare, secvențială a fiecărui modul, vizul se va avea impresia că ambele sunt aprinse (active) în același timp. Schema electrică, prezentată în figura 2.10, necesită următoarele componente electrice:

- 1x arduino Uno R3;
- 13 resistori x 1k ohm;
- 3x butoane fără reținere;
- 3x afișaje cu 7 segmente - catod comun;
- 3x tranzistoare de tip PNP.

Algoritmul 2.5 Utilizarea afișajului cu 7 segmente.

```
1 // numarul de afisoare cu 7 segmente
2 const byte N = 3;
3
```

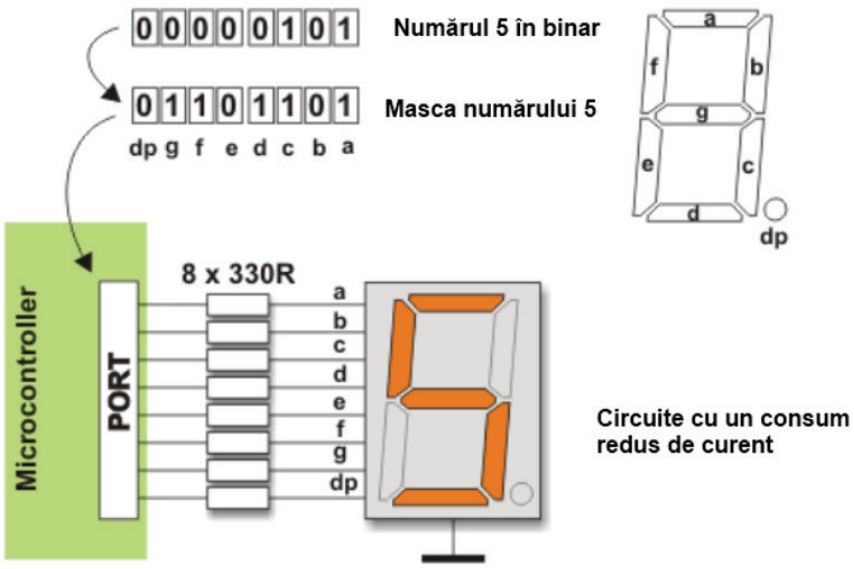

Fig. 2.9 Afișaj cu 7 segmente configurat să afișeze valoarea decimală 5.

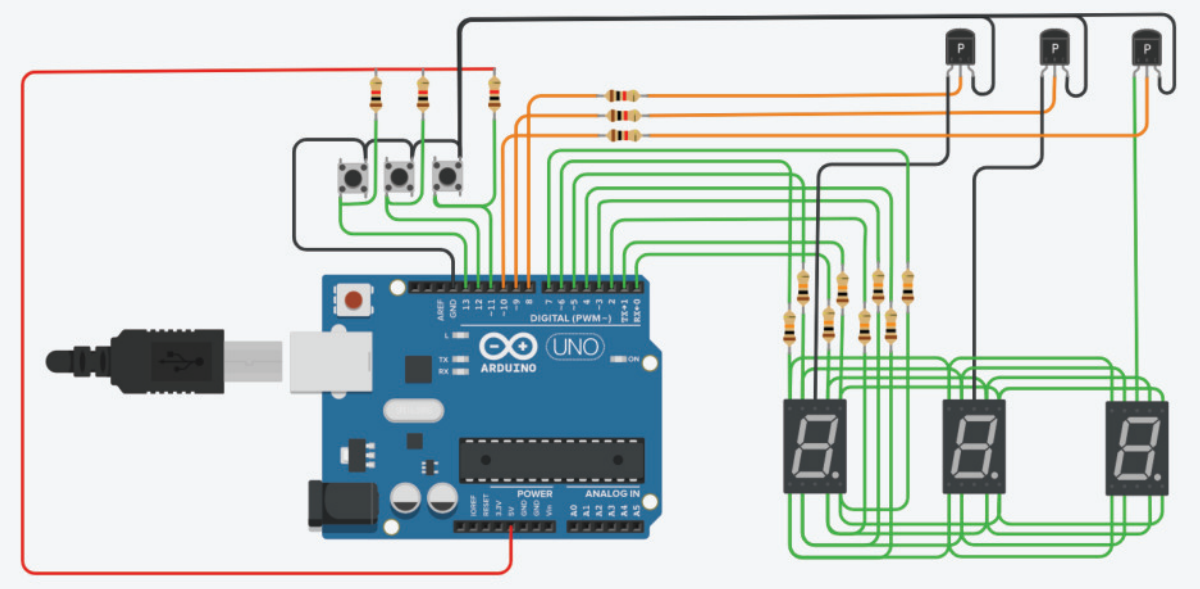

Fig. 2.10 Schema electrică pentru contorizarea numărului de pasageri ai unui avion.

```
4  // pin buton incrementare
5  const byte pinInc = 5;
6  // pin buton decrementare
7  const byte pinDec = 4;
8  // pin resetare numar pasageri
9  const byte pinReset = 3;
10
11 // pini afisaje
12 const byte pinsDisplay[N] = { 0, 1, 2 };
13
14 // starea pinilor pentru fiecare numar
```

```
15 const byte starePortNumere[10] =
16 {
17     // gfedcba (abcdefg invers)
18     0b00111111, // 0
19     0b00000110, // 1
20     0b01011011, // 2
21     0b01001111, // 3
22     0b01100110, // 4
23     0b01101101, // 5
24     0b01111101, // 6
25     0b00000111, // 7
26     0b01111111, // 8
27     0b01101111  // 9
28 };
29
30 int contor = 0;
31
32 void setup()
33 {
34     // setare tot portul D ca iesire
35     DDRD = 0b11111111;
36     // setare pini intrare/iesire port B
37     DDRB = 0b00000111;
38 }
39
40 // functie care verifica daca butonul
41 // de pe un anumit pin a fost apasat, adica
42 // daca starea lui e LOW (0); in plus, functia
43 // asteapta pana cand butonul nu mai e apasat
44 bool buton_apasat(byte pin_buton)
45 {
46     // verificare daca starea butonului e LOW (0)
47     if ((PINB & (1 << pin_buton)) == 0)
48     {
49         // asteptare cat timp butonul e LOW (0)
50         while ((PINB & (1 << pin_buton)) == 0)
51         {
52             // asteptare...
53         }
54
55         // butonul este apasat, pentru ca este LOW
```

```
56          return true;
57      }
58
59      // butonul nu este apasat
60      return false;
61  }
62
63  void loop()
64  {
65      if (buton_apasat(pinInc))
66      {
67          ++contor;
68      }
69
70      if (buton_apasat(pinDec))
71      {
72          --contor;
73      }
74
75      if (buton_apasat(pinReset))
76      {
77          contor = 0;
78      }
79
80      // pentru fiecare afisaj, afisam numarul
81      // respectiv, si activam/dezactivam ceea
82      // ce e necesar
83      for (byte i = 0; i < N; ++i)
84      {
85          // primul  afisaj, afisare sute
86          if (i == 0)
87          {
88              PORTD = starePortNumere[(contor / 100) % 10];
89          }
90          // al doilea afisaj, afisare zeci
91          else if (i == 1)
92          {
93              PORTD = starePortNumere[(contor / 10) % 10];
94          }
95          // al treilea afisaj, afisare unitati
96          else if (i == 2)
```

```
97          {
98              PORTD = starePortNumere[contor % 10];
99          }
100
101         // dezactivare afisaje
102         PORTB |= 0b00000111;
103         // activare doar cel curent
104         PORTB &= ~(1 << pinsDisplay[i]);
105
106         delay(20);
107     }
108 }
```

2.4.1.2 Aplicație propusă

Să se implementeze o aplicație software pentru microcontrolerul 328 care să permită controlul unei treceri de pietoni semaforizate. Culoarea verde a semaforului pietonilor este solicitată la cerere, prin apăsarea unui buton fără reținere. Culoarea verde a semaforului pietonilor trebuie sa dureze 30 secunde, timp în care semaforul vehiculelor au culoarea roșie. Atât semaforul pentru vehicule cât și semaforul pentru pietoni trebuie să prezinte un contor al timpului rămas până la schimbarea culorii. Simularea aplicației poate fi realizată utilizând pagina web *https://www.tinkercad.com*.

2.4.2 Utilizarea porturilor digitale I/O pentru comanda afișajelor LCD

Cel mai comod şi mai răspândit mod pentru afişarea caracterelor alfanumerice (consacrate sau definite de utilizator) este prin utilizarea dispozitivelor de afişare cu cristale lichide (LCD), în format matricial. Un astfel de dispozitiv este constituit dintr-o matrice de dimenisune fixă (de exemplu 8x8 sau 7x8 celule lichide) care devin opace, sau își schimbă culoarea, sub influența unui curent sau câmp electric. Unul dintre cele mai întâlnite afișaje LCD grupează pe 2 rânduri câte 16 astfel de matrici. Acest afișaj LCD este suficient pentru majoritatea aplicațiilor complexe care implică descrierea vizuală a unor informații. Pentru a putea realiza o astfel de aplicaţie, managementul datelor afişate este asigurat de un microcontroler, conectat la un LCD prin intermediul unui port de intrare-ieșire.

2.4.2.1 Modul de conectare al unui afişaj cu cristale lichide

Un afișaj LCD poate primii date/informații de la un microcontroler utilizând 4 sau 8 linii de comunicație (bus de date). Astfel, pentru cazul cu 8 linii (biți) de comunicare, afişajul are nevoie de: o tensiune de alimentare de +5V, GND, 8 linii I/O și 3 linii de control. Pentru cazul cu 4 linii de comunicare sunt necesare: o tensiune de alimentare de +5V, GND, 4 linii I/O și 3 linii de control. Când afişajul LCD nu este pornit liniile de date sunt TRI-STATE, ceea ce înseamnă că ele sunt în stare de înaltă impedanţă (ca şi cum ar

fi deconectate) şi astfel nu interferează cu funcționarea microcontrolerului. Microcontrolerul realizează controlul LCD-ului cu ajutorul a 3 linii de control, acestea având următoarea semnificaţie:

- linia *Enable* (E) permite accesul la afişaj prin intermediul liniilor R/W şi RS. Când această linie este LOW, LCD-ul este dezactivat şi ignoră semnalele de la R/W şi RS. Când linia (E) este HIGH, LCD-ul verifică starea celor două linii de control şi răspunde corespunzător;
- linia *Read/Write* (R/W) stabileşte direcţia datelor dintre LCD şi microcontroler. Când linia este LOW, datele sunt scrise în LCD. Când linia este HIGH, datele sunt citite de la LCD;
- linia *Register Select* (RS), este utilizată de LCD pentru interpretarea tipului de date, de pe liniile de date. Când este LOW, o instrucţiune este scrisă în LCD, iar când este HIGH un caracter este scris în LCD.

Starea logică în care se găseşte fiecare linie de control este următoarea:

Tab. 2.1 Semnificația stării logice a liniilor de control ale unui LCD.

E:	0 – Accesul la LCD dezactivat	1 – Accesul la LCD activat
R/W:	0 – Scrie date în LCD	1 – Citeşte date din LCD
RS:	0 – Instrucţiuni	1 – Caracter

Procedura pentru scrierea unui caracter pe afișajul LCD-ului este următorul:

- se seteaza bitul R/W pe valoarea logică 0;
- se seteaza bitul de RS pe valoarea logică 0 sau 1, în funcţie de utilizarea unei instrucțiuni sau de utilizarea datelor;
- se trimite date către liniile de date (dacă se execută o scriere);
- se setează linia E pe valoarea logică 1;
- se citesc date de la liniile de date (dacă se execută o citire).

Procedura pentru citirea datelor se realizează la fel, cu excepţia setării bitului R/W pe valoarea logică 1. Pentru a genera un caracter special se utilizează reprezentarea din figura 2.11. Afişajul LCD mai conţine o memorie CGRAM (Character Generator RAM). Această memorie este rezervată pentru caracterele definite de utilizator. Datele din CGRAM sunt reprezentate sub formă de caractere bitmap, de 8 biți. Pentru a afişa caracterul bitmap pe LCD, trebuie setată adresa CGRAM la punctul de start (de obicei 0) şi apoi scrise datele în afişaj. Pentru a-i ajuta pe utilizatori, IDE-ul Arduino conține o librărie care definește deja funcții specifice pentru utilizarea facilă a afișajelor de tip LCD. Aceste funcții pot fi utilizate odată cu apelarea directivei:

```
1    #include <LiquidCrystal.h>
```

în aplicația software. Descrierea completă poate fi urmărită la adresa web: *https://www.arduino.cc/en/Reference/LiquidCrystal*.

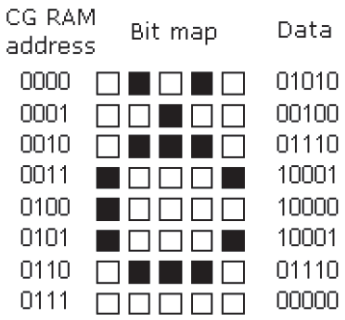

Fig. 2.11 Modul de formare al unui caracter al unui afișaj LCD.

2.4.2.2 Exemplu de aplicație

Să se elaboreze și simuleze, utilizându-se Tinkercard, un program care să afișeze un text și o valoare pe un afișaj de tip LCD. Schema electrică a aplicației propuse este prezentată în figura 2.12.

Componentele utilizate în schemă sunt:

- 1x LCD 16x2;
- 1x arduino Uno R3;
- 1x potențiometru 500 ohmi;
- 1x rezistor 1 kohm.

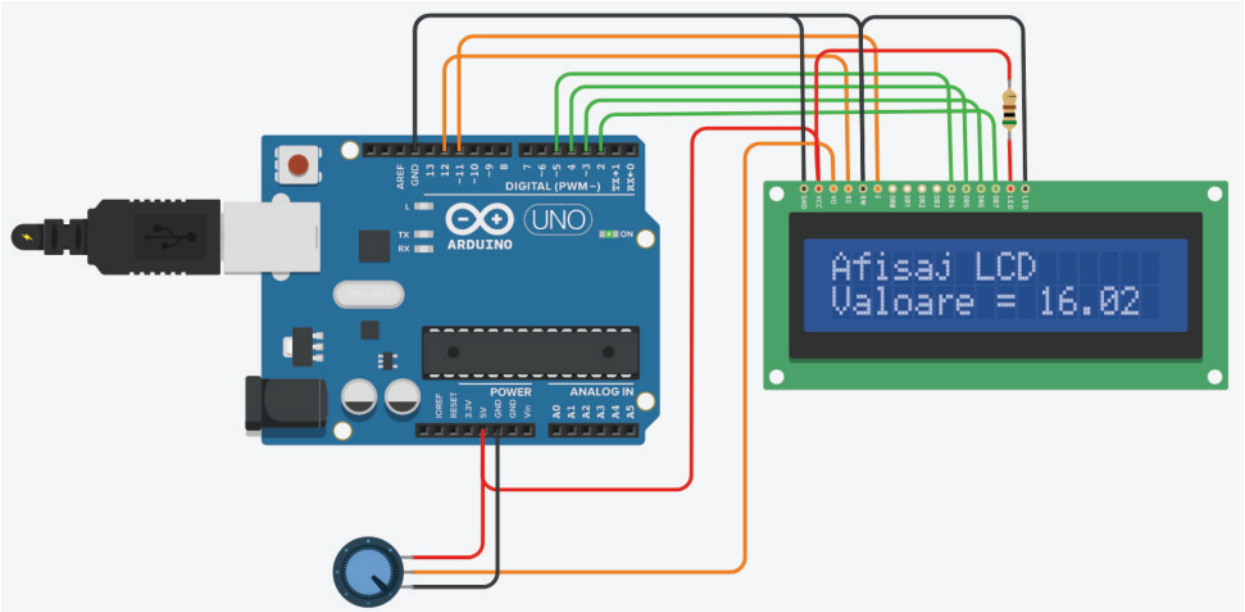

Fig. 2.12 Afișarea de caractere pe LCD.

Programul este următorul:

Algoritmul 2.6 Utilizarea afișajului LCD.

```
1 // include libraria specifica:
2 #include <LiquidCrystal.h>
3
```

```
4  /*
5    Conectarea pinilor la placa Arduino UNO:
6   * LCD RS la pinul digital 12
7   * LCD Enable la pinul digital 11
8   * LCD D4 la pinul digital 5
9   * LCD D5 la pinul digital 4
10  * LCD D6 la pinul digital 3
11  * LCD D7 la pinul digital 2
12  * LCD R/W la pinul de masa
13  * LCD VSS la pinul de masa
14  * LCD VCC la pinul 5V
15  * VO la pinul de iesire al potentiometrului
16 */
17
18 // asociaza pinii LCD-ului catre pinii placii Arduino UNO
19 const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
20 LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
21
22 void setup() {
23   // defineste nr. de lini si coloane ale LCD-ului:
24   lcd.begin(16, 2);
25   // Afiseaza un text pe LCD.
26   lcd.print("Afisaj LCD");
27 }
28
29 void loop() {
30   // seteaza cursorul pe linia 1 coloana 0
31   // atentie, linia 1 este pe cel de-al doilea rand, deoarece
       ↪ numaratoarea incepe de la 0
32   lcd.setCursor(0, 1);
33   // printeaza un text
34   lcd.print("Valoare = ");
35   lcd.setCursor(10, 1);
36   // printeaza o valoare
37   lcd.print(16.02);
38 }
```

2.4.3 Utilizarea porturilor digitale I/O pentru controlul unei tastaturi hexa-numerice

Tastatura numerică reprezintă un mod practic și facil pentru a transmite valori unui microcontroler. Un astfel de dispozitiv este construit dintr-o matrice de butoane, uzual având dimensiunea de 4x4. Aceasta permite utilizarea unui număr decimal cuprins între 0 și 9 precum și definirea a 6 butoane cu funcții speciale. Funcțiile speciale pot fi folosite după cum necesită fiecare aplicație în parte.

2.4.3.1 Modul de funcționare al unei tastaturi hexa-numerice

O tastatură hexa-numerică este compusă din 16 butoane, fără reținere, așezate sub forma unei matrici. Astfel, se definesc 8 conexiuni: 4 pentru rânduri, respectiv 4 pentru coloane. Schema electrică precum și imaginea cu o astfel de tastatură sunt prezentate în figura 2.13.

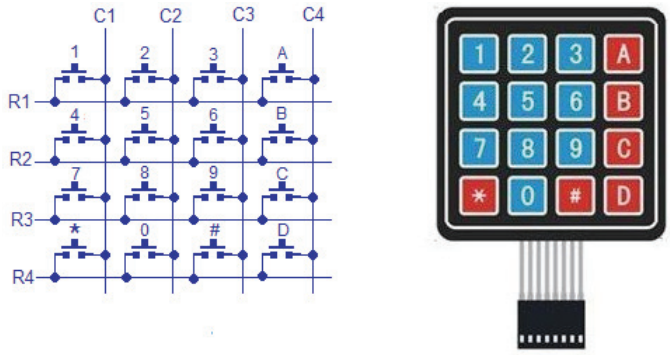

Fig. 2.13 Tastatură numerică (hexazecimală): matrice de butoane (stânga) și forma ei fizică (dreapta).

Procedura de identificare a butonului apăsat este următoarea. În primul pas se scanează coloanele. Pentru acest lucru, fiecare dintre rânduri este menținut pe rand la starea logică 0, iar celelalte rânduri sunt menținute la starea logică 1. Citirea stăriilogice a fiecărei coloane este realizată de microcontroler. Dacă o anumită coloană este găsită cu valoarea 0, acest lucru înseamnă că butonul care se află pe coloana și rândul respectiv este în scurt-circuit, adică apăsat. Procesul se repetă într-o buclă pentru a capta o nouă apăsare. Relativ la figura 2.13, dacă rândul 1 este 0 și coloana 1 este găsită, în timpul scanării, tot cu valoarea 0, atunci butonul 1 este apăsat. Pentru a evita implementarea manuală a metodei anterior aminite, se recomandă utilizarea librăriei *Keypad* a cărui cod sursă se află la adresa: *https://github.com/Chris–A/Keypad*

2.4.3.2 Exemplu de aplicație

Utilizând libraria menționată anterior, să se elaboreze și simuleze, utilizând Tinkercard, un program care să citească o parolă de la o tastatură hexa-numerică, iar dacă aceasta este identică cu valoarea *1602*, să aprindă un led verde. Pe toată perioada în care parola nu este corectă, respectiv nu toate valorile au fost introduse, să se aprindă un led roșu. Pentru a șterge valorile introduse până în momentul de față se va utiliza tasta $\star$. Resetarea parolei se face cu tasta #. Schema electrică a aplicației propuse este prezentată în figura 2.14

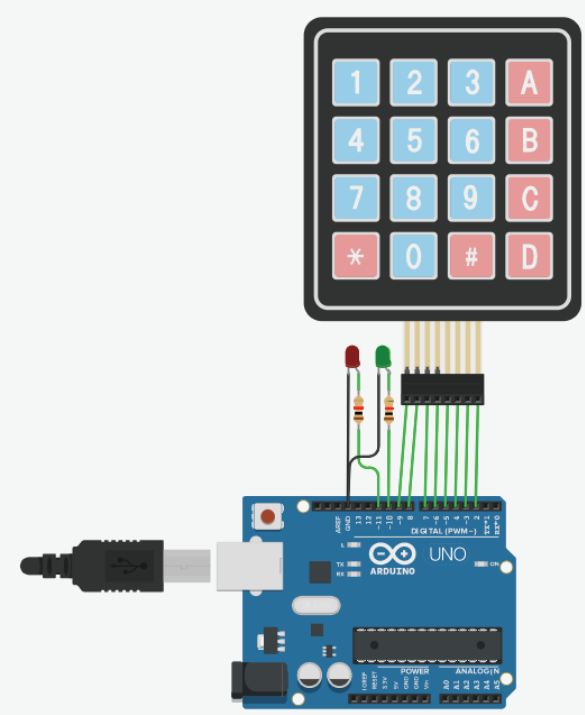

Fig. 2.14 Schema electrică pentru utilizarea tastaturii numerice.

Componentele utilizate în schemă sunt:

- 1x arduino UNO;
- 1x hex keyboard;
- 2x leduri, unul rosu și unul verde;
- 2x rezistor de 1 kohm.

Programul care realizează cerințele impuse este următorul:

Algoritmul 2.7 Utilizarea tastaturii hexa-numerice.

```
1  #include <Keypad.h>
2
3  #define DDRB0    0
4  #define DDRB1    1
5  #define DDRB2    2
6  #define DDRB3    3
7
8  #define PORTB2   2
9  #define PORTB3   3
10
11 #define DIM_PAROLA   4
12
13 #define RANDURI 4
14 #define COLOANE 4
15
16 byte contor = 0;
```

```
17
18 char valoareTasta[RANDURI][COLOANE] = {
19     { '1', '2', '3', 'A' },
20     { '4', '5', '6', 'B' },
21     { '7', '8', '9', 'C' },
22     { '*', '0', '#', 'D' }
23 };
24
25 char parolaCorecta[DIM_PAROLA] = { '1', '6', '0', '2' };
26 char parolaIntrodusa[DIM_PAROLA] = { '0', '0', '0', '0' };
27
28 byte piniRanduri[RANDURI] = { 9, 8, 7, 6 };
29 byte piniColoane[COLOANE] = { 5, 4, 3, 2 };
30
31 Keypad tastatura(
32   makeKeymap(valoareTasta),
33   piniRanduri, piniColoane,
34   RANDURI, COLOANE);
35
36 void setup()
37 {
38     // seteaza tot portul D ca port de intrare
39     DDRD = 0x00;
40     // seteaza pini iesire/intrare in portul B
41     DDRB = (0 << DDRB0) | (0 << DDRB1) | (1 << DDRB2) | (1 <<
       ↪ DDRB3);
42
43     PORTB |= 1 << PORTB3; // aprinde led rosu
44 }
45
46 bool compara_parole()
47 {
48     // pentru fiecare caracter din cele doua
49     // siruri de caractere, daca caracterele
50     // de pe acceasi pozitie nu sunt egale,
51     // atunci parolele nu sunt identice
52     for (byte i = 0; i < DIM_PAROLA; ++i)
53     {
54         if (parolaCorecta[i] != parolaIntrodusa[i])
55         {
56             return false;
```

```
57          }
58      }
59
60      return true;
61  }
62
63  void loop()
64  {
65      const char tastaApasata = tastatura.getKey();
66      if (tastaApasata != 0)
67      {
68          if (tastaApasata == '#')
69          {
70              // resetare parola
71              for (byte i = 0; i < DIM_PAROLA; ++i)
72              {
73                  parolaIntrodusa[i] = '0';
74              }
75          }
76          else
77          {
78              // retine caracterul introdus la pozitia
79              // curenta, si incrementeaza pozitia
80              parolaIntrodusa[contor] = tastaApasata;
81              ++contor;
82
83              // parola are maxim DIM_PAROLA caractere,
84              // deci trebuie resetat la 0 contorul daca
85              // depaseste valoarea respectiva
86              if (contor >= DIM_PAROLA)
87              {
88                  contor = 0;
89              }
90
91              // verifica daca parolele sunt identice
92              if (compara_parole())
93              {
94                  // stinge led rosu
95                  PORTB &= ~(1 << PORTB3);
96                  // aprinde led verde
97                  PORTB |= (1 << PORTB2);
```

```
98                      // asteapta 1 secunda
99                      delay(1000);
100                     // aprinde led rosu
101                     PORTB |= (1 << PORTB3);
102                     // stinge led verde
103                     PORTB &= ~(1 << PORTB2);
104                 }
105             }
106         }
107 }
```

2.4.3.3 Aplicație propusă

Să se implementeze o aplicație software pentru microcontroler care să rezolve operații matematice simple (înmulțire, adunare, scădere, împărțire), utilizând valori zecimale introduse de la o tastatură numerică. Considerându-se tastatura prezentată în figura 2.13, tasta A se va folosi pentru a realiza operația de adunare între două valori, B pentru operația de scădere, C pentru operația de înmulțire, respectiv D pentru operația de împărțire. Tasta $\star$ va avea funcție de ștergere (resetare) a datelor citite de la tastatura, respectiv # va avea funcția de afișare rezultat (=). Vizualizarea valorilor citite de la tastatură, cât și a rezultatelor, se va realiza pe un afișaj LCD 16x2.