

Arduino C/C++ programozás 2.

- 9. Függvények
- 10. Hétszegmenses kijelző vezérlése
- 11. Állapotok tárolása, hétszegmenses kijelző 2.
- 12. Tömbök, bitműveletek
- 13. LCD kijelző vezérlése
- 14. PWM kimenetek
- 15. Hangok
- 16. Idő, véletlen számok
- 17. Feladatok

Felhasznált forrás, és ajánlott irodalom:

Ruzsinszki Gábor: Programozható Elektronikák

Felhasznált és ajánlott online szimulációs felület: www.tinkercad.com

9.1. Függvény

Függvények

- vannak előre definiált, beépített függvények:
 - a C nyelv könyvtári függvényei,
 - a használt fejlesztési környezet saját függvényeiezeket csak használni kell, pl. Arduino esetén a delay() vagy a digitalWrite() ...
- de természetesen létrehozhatunk saját függvényeket is
- egy függvény fejrészét (visszatérési érték típusa, függvény neve, és zárójelek között paraméterek) a függvény törzse követi → { } zárójelek között

```
visszatérési_típus függvény_neve(paraméterek) // fejrész
{
    utasítások;                               // a függvény törzse
}
```

Miért használunk függvényeket?

- függvények segítségével tudjuk a programunkat kisebb egységekre, alprogramokra osztani
- így programunk átláthatóbb, egyszerűbb felépítésű lesz →
- könnyebben módosítható, hibakeresés egyszerűbb (?), könnyebb újra felhasználás

9.2. Függvény

Egy C nyelvű program szerkezete:

- függvényekből (alprogramokból) áll, általában minimum 1 függvény kell, ez a **'main'** függvény (általában) !!
- a main függvény a fő függvény, vele kezdődik a program végrehajtása, nem hagyható el ! (Arduino esetén nincs !)
- egy függvény fejrészét (visszatérési érték típusa, függvény neve, és zárójelek között paraméterek) a függvény törzse követi → { } zárójelek között

egy egyszerű C program tehát így néz ki:

```
void main( )    // void → nincs visszatérési érték
{
    definíció1;  // sorok végén megjegyzések lehetnek '/' után !!!
    definíció2;  // definíciók (deklarációk) után pontosvessző kell (;)
    ...
    utasítás1;   // utasítások után szintén pontosvessző kell !!
    utasítás2;
    utasítás3;
    ...
}
```

9.3. Függvény

Arduino esetén kicsit módosul az egész

- a „main” helyett **két függvény !!** → „setup” és „loop”
- a „setup” függvénnyel kezdődik a program végrehajtása, csak egyszer fut le
- ezután a „loop” függvény hajtódik végre, az viszont folyamatosan újra és újra végrehajtódik !! (végtelen ciklus)
- plusz függvényeket természetesen létrehozhatunk

egy egyszerű program Arduino esetén tehát így néz ki:

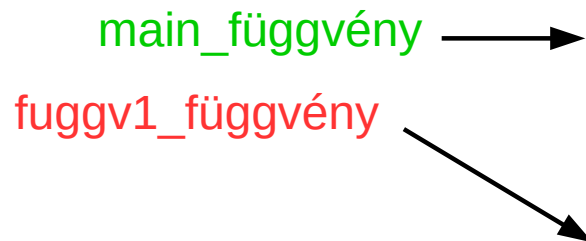
```
void setup( )           // először ez fut le (egyszer)
{
    utasitas1;
    utasitas2;
    ...
}

void loop( )           // majd ez fut folyamatosan, újra és újra
{
    utasitas1;
    utasitas2;
    ...
}
```

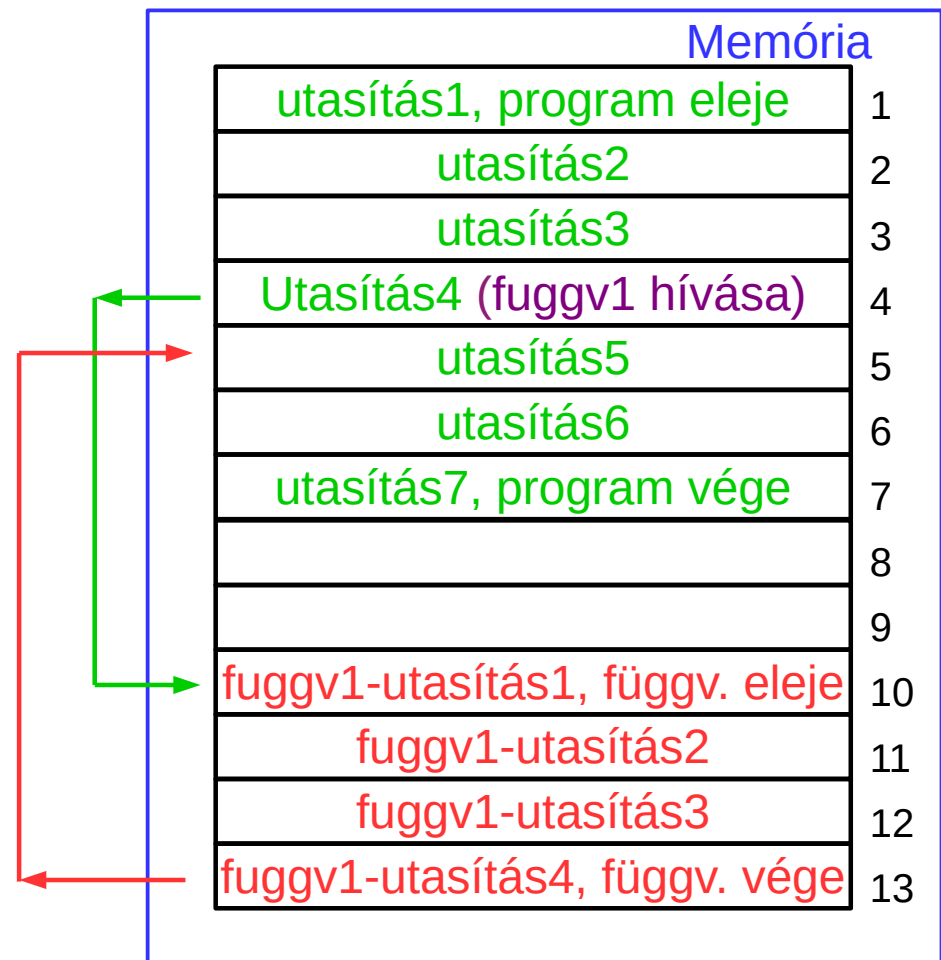
9.4. Függvény használata

Függvény hívása, majd vissza:

- lényegében **ugró utasítással történik** ! (gépi kód szinten)
- alapvetően az utasítások végrehajtása ugyanis egymás után történik, ahogyan a memóriában következnek egymás után (olyan sorrendben, ahogyan mi leírtuk az utasításokat)
- függvény hívás esetén azonban nem a következő memória rekeszben lévő utasítás hajtódik végre, hanem az adott memória címre ugrunk, és ott folytatjuk tovább



A program végrehajtás sorrendje ebben az esetben tehát (melyik számú memóriarekeszben lévő utasítás hajtódik végre)
→ 1-2-3-4-10-11-12-13-5-6-7



9.5. Függvény használata

Kapcsolat egy függvény és a program többi része között

- a függvény bemenete → a paraméterek, segítségével tudunk adatokat átadni a függvénynek (amikor meghívjuk)
- a függvény kimenete → a visszatérési érték, segítségével tudunk adatot visszaadni a hívó programrésznek
- C nyelven függvényen belül nem lehet függvényt létrehozni !!
- függvény hívása → **függvénynév(bemenő_paraméterek);**
pl. **delay(3000);**
- elképzelhető, hogy nincs paramétere a függvénynek
- ha van paramétere, akkor ugyanannyi darab (és ugyanolyan típusú) paramétereket kell átadnunk mikor meghívjuk !
- visszatérési érték felhasználása
pl. **szam = atlag(10,36,60);**
- paraméterek definiálása → típus név
vissza_típus függvény_név(típus1 név1, típus2 név2, ..., típusx névx)

pl. **int atlag3(int a, int b, int c)**
- érték visszaadása a **return** utasítással → pl. **return 10**

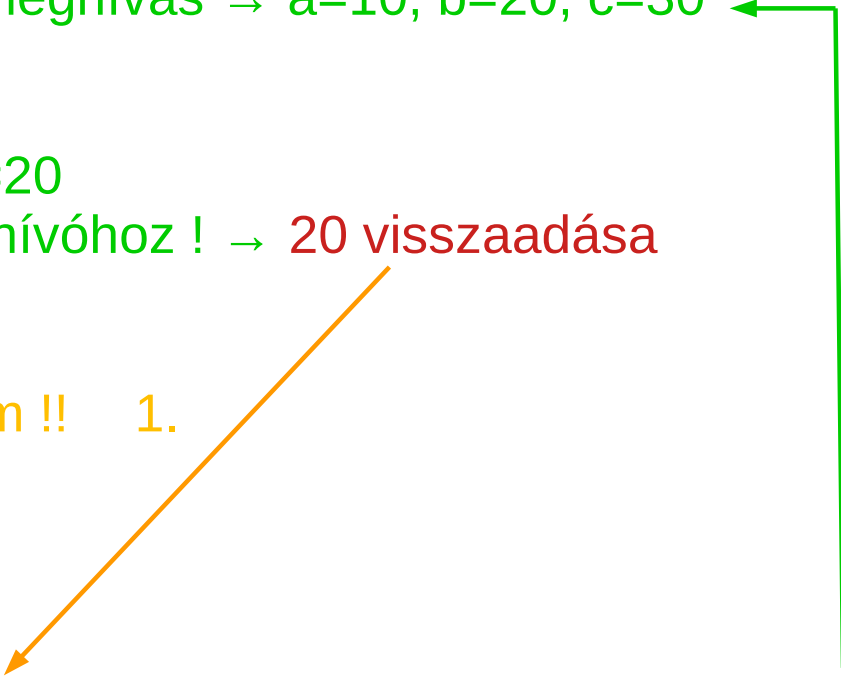
9.6. Függvény használata

Függvény hívása, érték visszaadása

- pl. 3 szám átlagát kiszámoló függvény

```
float atlag3(float a,float b,float c) // 6. meghívás → a=10, b=20, c=30
{
    float atl;           // 7.
    atl=(a+b+c)/3;       // 8. atl=60/3=20
    return atl;          // 9. vissza a hívóhoz ! → 20 visszaadása
}

void main( )             // itt indul a program !! 1.
{
    float szam1;          // 2.
    float szam2=20;       // 3.
    float szam3;          // 4.
    szam1=atlag3(10,szam2,30); // 5. → atlag3 meghívása → ugrás
    ...                  // 10. → szam1=20
}
```



9.7. Függvény használata

Függvény meghívása többször

- sokszor meghívható egy függvény, más-más paraméterekkel
- ez csökkenti a sorok számát (a függvényt csak egyszer kell leírni)

```
float atlag3(float a,float b,float c) // 6. meghívás → a=10, b=20, c=30
{
    // 12. meghívás → a=20, b=20, c=38
    float atl;           // 7. // 13.
    atl=(a+b+c)/3;       // 8. atl=60/3=20 // 14. atl=78/3=26
    return atl;          // 9. ugrás, 20 visszaadása // 15. ugrás, 26 vissza
}

void main( )             // itt indul a program !! 1.
{
    float szam1;          // 2.
    float szam2=20;       // 3.
    float szam3;          // 4.
    szam1=atlag3(10,szam2,30); // 5. → atlag3 meghívása → ugrás
                                // 10. → szam1=20
    szam3=atlag3(szam2,szam2,38); // 11. → atlag3 meghívása → ugrás
                                    // 16. szam3=26
}
```


9.8. Helyi és globális változók

Helyi változók

- A függvényen belül definiált változók → a függvény helyi (saját) változói → más függvények ezeket nem érik el !!
- **különböző függvényeknek lehetnek azonos nevű helyi változói → de ezen változók teljesen függetlenek egymástól !!**
(hasonlóan mint egy Miskolcon lévő Petőfi utcának semmi köze egy Debrecenben lévő Petőfi utcához)

```
fuggv1() {  
    char i;  
    char szam1;  
    ..... }
```

```
fuggv2() {  
    char j;  
    char szam1;  
    ..... }
```

```
main() {  
    char szam1;  
    char szam2;  
    ..... }
```

szam1, **szam1** és **szam1** változó 3 különböző változó !! → semmi közük egymáshoz

fuggv1 számára 'j' és 'szam2' változó nem létezik

fuggv2 számára 'i' és 'szam2' változó nem létezik

main számára 'j' és 'i' változó nem létezik

9.9. Helyi és globális változók

Globális változók

- A függvényeken kívül definiált változók (a program elején) !!
- minden függvény eléri ezeket a változókat
(kivéve ha ugyanilyen néven helyi változója is van !!)

```
char i;  
char szam3;
```

```
fuggv1() {  
    char i;  
    char szam1;  
    ..... }  
}
```

```
fuggv2() {  
    char j;  
    char szam1;  
    ..... }  
}
```

```
main() {  
    char szam1;  
    char szam2;  
    ..... }  
}
```

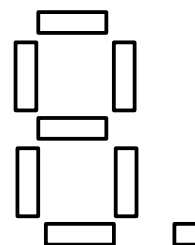
A globális 'i' és 'szam3' változókat minden függvény látja → tudja azokat használni

DE !! függv1 tartalmaz helyi 'i' változót → a globális 'i' helyett azt látja !!

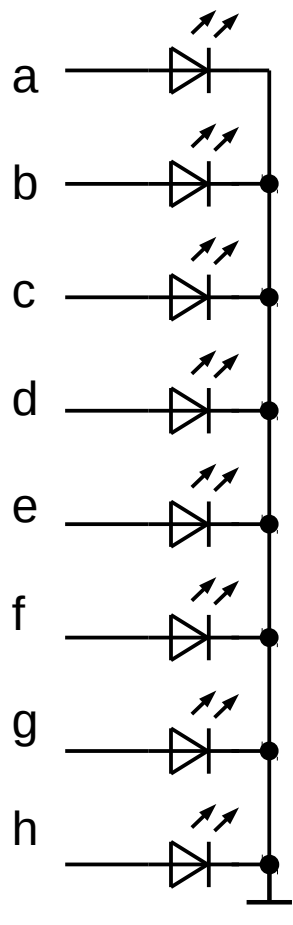
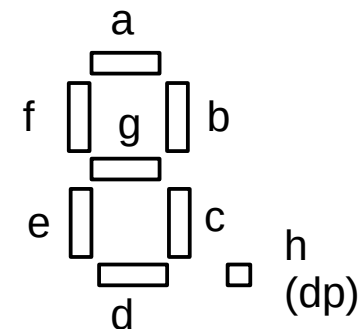
10.1. Hétszegmenses kijelző

Hétszegmenses kijelző

7 + 1 szegmensből áll
amelyek LEDek igazából
→ kétféle bekötés !!



szegmens nevek

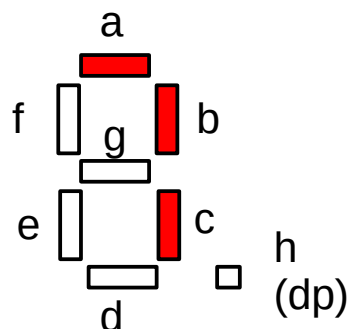


Közös katódú

vezérlés:

- amelyik szegmensre **1-et adunk** → **világít**
- amelyekre **0-át adunk** → **nem világít**

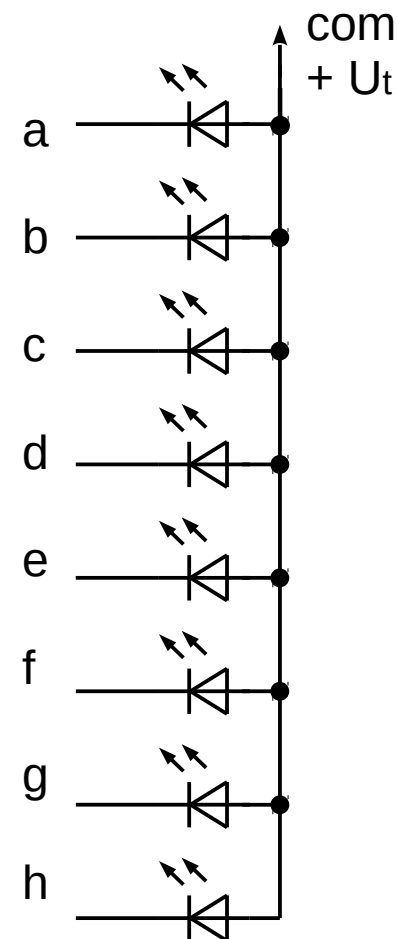
pl. a=b=c=1,
a többi 0



Közös anódú

vezérlés:

- amelyik szegmensre **0-át adunk** → **világít**
- amelyekre **1-et adunk** → **nem világít**

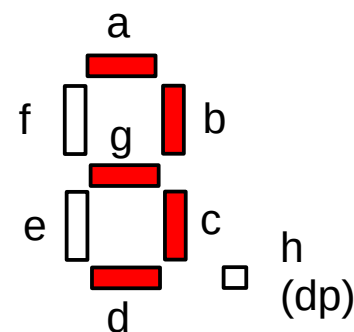
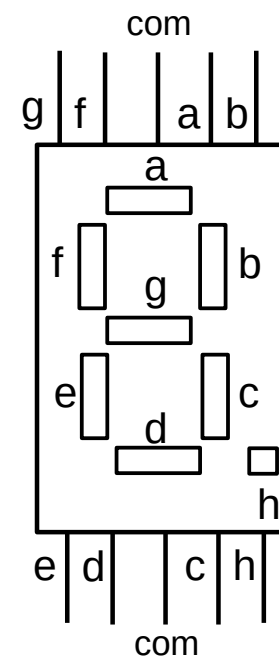
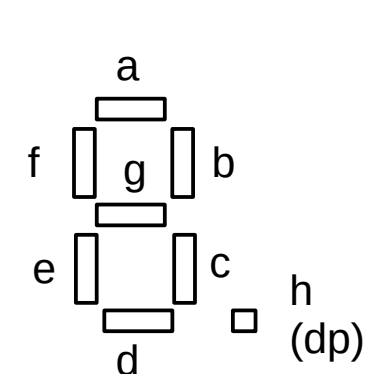
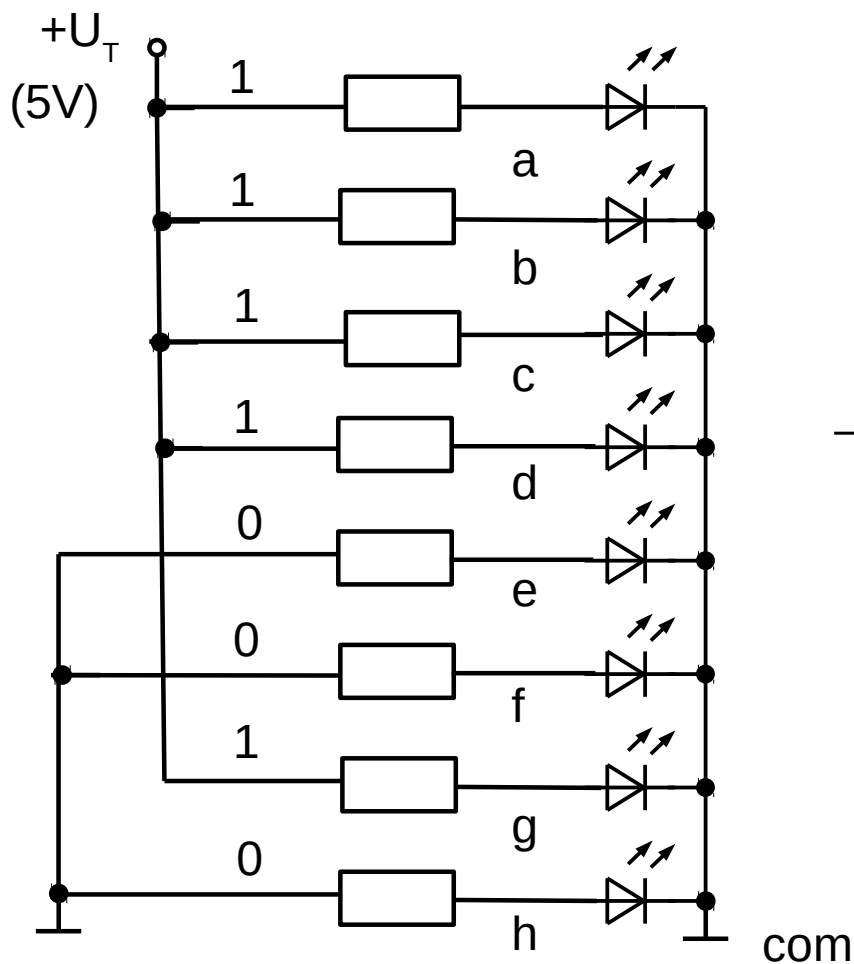


10.2. Hétszegmenses kijelző

Hétszegmenses kijelző

Közös katódú

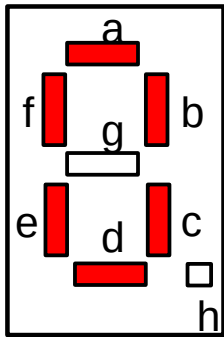
- amelyik szegmensre **1-et adunk** → **világít**
- amelyekre **0-át adunk** → **nem világít**



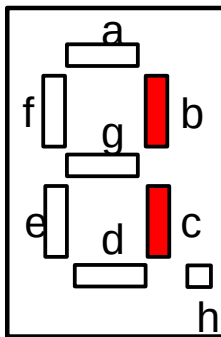
10.3. Hétszegmenses kijelző

Hogyan kell vezérelni a szegmenseket ?

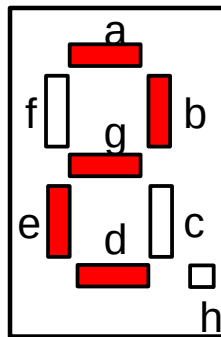
0



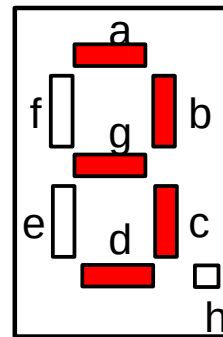
1



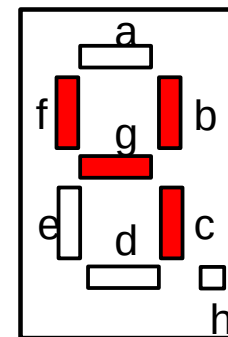
2



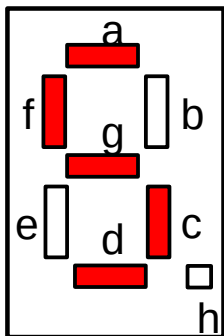
3



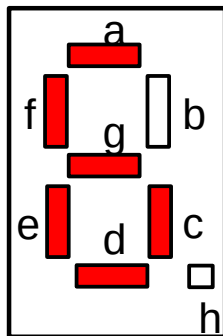
4



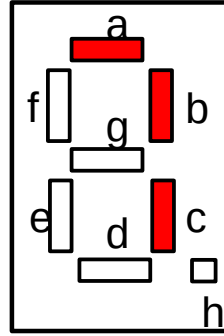
5



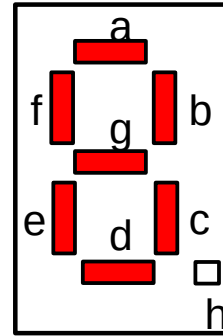
6



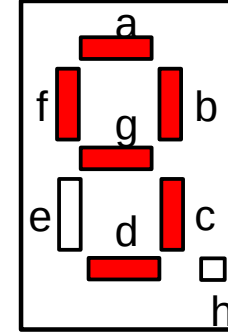
7



8

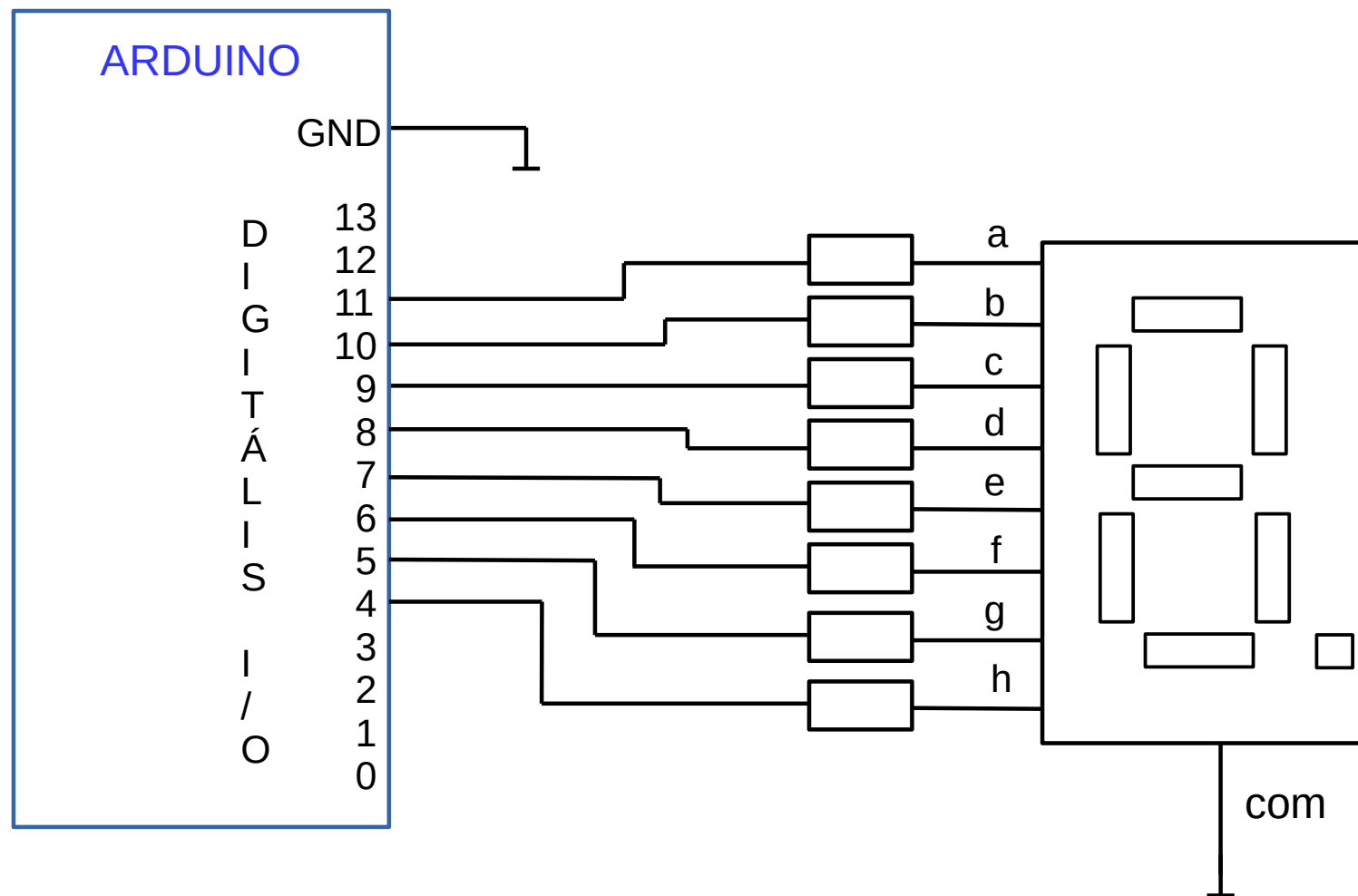


9

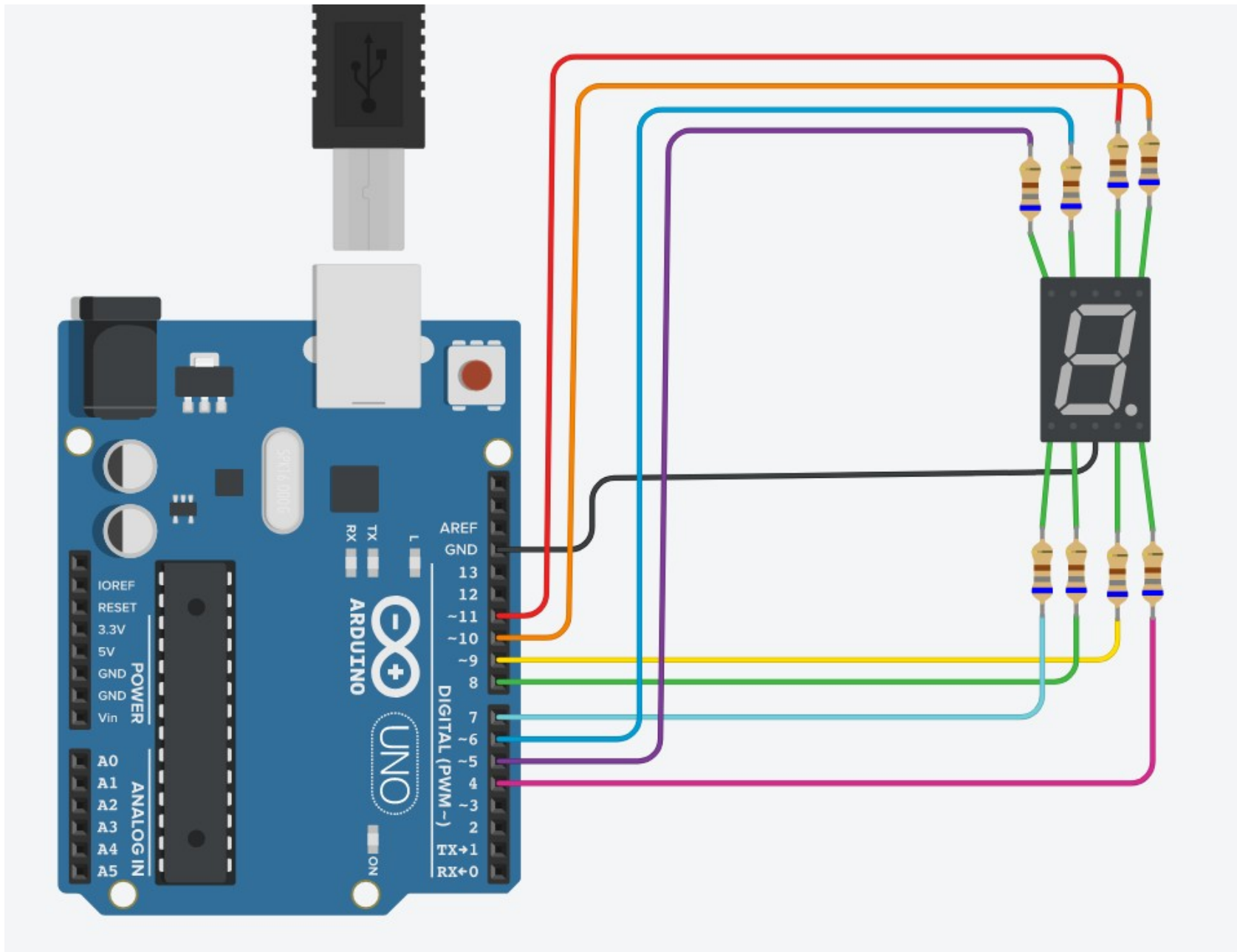


10.4. A hardver

Hétszegmenses kijelző (közös katódú) digitális kimeneteken



10.5. A hardver



10.6. Hétszegmenses kijelző vezérlése

1. mintafeladat

A kijelzőn felváltva, egyszer az összes szegmens világítson, majd egyik sem !

// globális változók tárolják, hogy az egyes szegmensek

// melyik lábbal vezérelhetők

byte a7=11; // 'a' szegmens 11. lábon van

byte b7=10; // 'b' szegmens 10. lábon

byte c7=9; // 'c' szegmens

byte d7=8; // 'd' szegmens

byte e7=7; // 'e' szegmens

byte f7=6; // 'f' szegmens

byte g7=5; // 'g' szegmens

byte h7=4; // 'h' szegmens

void setup()

{

pinMode(a7, OUTPUT); // 'a7' kimenet lesz

pinMode(b7, OUTPUT); // 'b7' kimenet lesz

pinMode(c7, OUTPUT); // 'c7' kimenet lesz

pinMode(d7, OUTPUT); // 'd7' kimenet lesz

pinMode(e7, OUTPUT); // 'e7' kimenet lesz

pinMode(f7, OUTPUT); // 'f7' kimenet lesz

pinMode(g7, OUTPUT); // 'g7' kimenet lesz

pinMode(h7, OUTPUT); // 'h7' kimenet lesz

}

10.7. Hétszegmenses kijelző vezérlése

1. mintafeladat folytatása

A kijelzőn felváltva, egyszer az összes szegmens világítson, majd egyik sem !

// az összes szegmens, illetve az egyik sem bekapcsolását külön függvényekben írjuk meg
// és a loop függvényből csak folyamatosan meghívjuk azokat

```
void loop( )
{
    szegm_mind();           // függvény hívás !!
    delay(1000);
    szegm_egysem();        // függvény hívás !!
    delay(1000);
}

void szegm_mind()          // az összes szegmens felkapcsolása
{
    digitalWrite(a7, HIGH);
    digitalWrite(b7, HIGH);
    digitalWrite(c7, HIGH);
    digitalWrite(d7, HIGH);
    digitalWrite(e7, HIGH);
    digitalWrite(f7, HIGH);
    digitalWrite(g7, HIGH);
    digitalWrite(h7, HIGH);
}
```

10.8. Hétszegmenses kijelző vezérlése

1. mintafeladat folytatása

A kijelzőn felváltva, egyszer az összes szegmens világítson, majd egyik sem !

// az összes szegmens, illetve az egyik sem bekapcsolását külön függvényekben írjuk meg
// és a loop függvényből csak folyamatosan meghívjuk azokat

```
void szegm_egysem()
{
    digitalWrite(a7, LOW);
    digitalWrite(b7, LOW);
    digitalWrite(c7, LOW);
    digitalWrite(d7, LOW);
    digitalWrite(e7, LOW);
    digitalWrite(f7, LOW);
    digitalWrite(g7, LOW);
    digitalWrite(h7, LOW);
}
```

10.9. Hétszegmenses kijelző vezérlése

2. mintafeladat

A kijelzőn felváltva az 1-es illetve a 2-es szám jelenjen meg !

A feladat eleje megegyezik az előző feladattal ! (globális változók és setup fv.)

```
// globális változók tárolják, hogy az egyes szegmensek
```

```
// melyik lábbal vezérelhetők
```

```
byte a7=11;           // 'a' szegmens 11. lábon van
```

```
byte b7=10;           // 'b' szegmens 10. lábon
```

```
byte c7=9;            // 'c' szegmens
```

```
byte d7=8;            // 'd' szegmens
```

```
byte e7=7;            // 'e' szegmens
```

```
byte f7=6;            // 'f' szegmens
```

```
byte g7=5;            // 'g' szegmens
```

```
byte h7=4;            // 'h' szegmens
```

```
void setup( )
```

```
{
```

```
    pinMode(a7, OUTPUT); // 'a7' kimenet lesz
```

```
    pinMode(b7, OUTPUT); // 'b7' kimenet lesz
```

```
    pinMode(c7, OUTPUT); // 'c7' kimenet lesz
```

```
    pinMode(d7, OUTPUT); // 'd7' kimenet lesz
```

```
    pinMode(e7, OUTPUT); // 'e7' kimenet lesz
```

```
    pinMode(f7, OUTPUT); // 'f7' kimenet lesz
```

```
    pinMode(g7, OUTPUT); // 'g7' kimenet lesz
```

```
    pinMode(h7, OUTPUT); // 'h7' kimenet lesz
```

```
}
```

10.10. Hétszegmenses kijelző vezérlése

2. mintafeladat folytatása

A kijelzőn felváltva az 1-es illetve a 2-es szám jelenjen meg !

// a számok megjelenítését külön függvényekben írjuk meg
// és a loop függvényből csak folyamatosan meghívjuk azokat

```
void loop( )  
{  
    szam1();           // függvény hívás !!  
    delay(1000);  
    szam2();           // függvény hívás !!  
    delay(1000);  
}  
  
void szam1()           // az 1-es szám megjelenítése  
{  
    digitalWrite(a7, LOW);  
    digitalWrite(b7, HIGH);  
    digitalWrite(c7, HIGH);  
    digitalWrite(d7, LOW);  
    digitalWrite(e7, LOW);  
    digitalWrite(f7, LOW);  
    digitalWrite(g7, LOW);  
    digitalWrite(h7, LOW);  
}
```

10.11. Hétszegmenses kijelző vezérlése

2. mintafeladat folytatása

A kijelzőn felváltva az 1-es illetve a 2-es szám jelenjen meg !

// a számok megjelenítését külön függvényekben írjuk meg
// és a loop függvényből csak folyamatosan meghívjuk azokat

```
void szam2()          // a 2-es szám megjelenítése
{
    digitalWrite(a7, HIGH);
    digitalWrite(b7, HIGH);
    digitalWrite(c7, LOW);
    digitalWrite(d7, HIGH);
    digitalWrite(e7, HIGH);
    digitalWrite(f7, LOW);
    digitalWrite(g7, HIGH);
    digitalWrite(h7, LOW);
}
```

10.12. Feladatok

Az előbbi minta feladatokat bővítsd ki új függvényekkel, amelyek a többi számot is megjelenítik (3, 4, 5, ... 9, 0)

Írj programokat a 7 szegmenses kijelző vezérlésére

1. feladat

- Az 1-3-5-7-9 értékek kijelzése egymás után (1s-ig mindegyik)

2. feladat

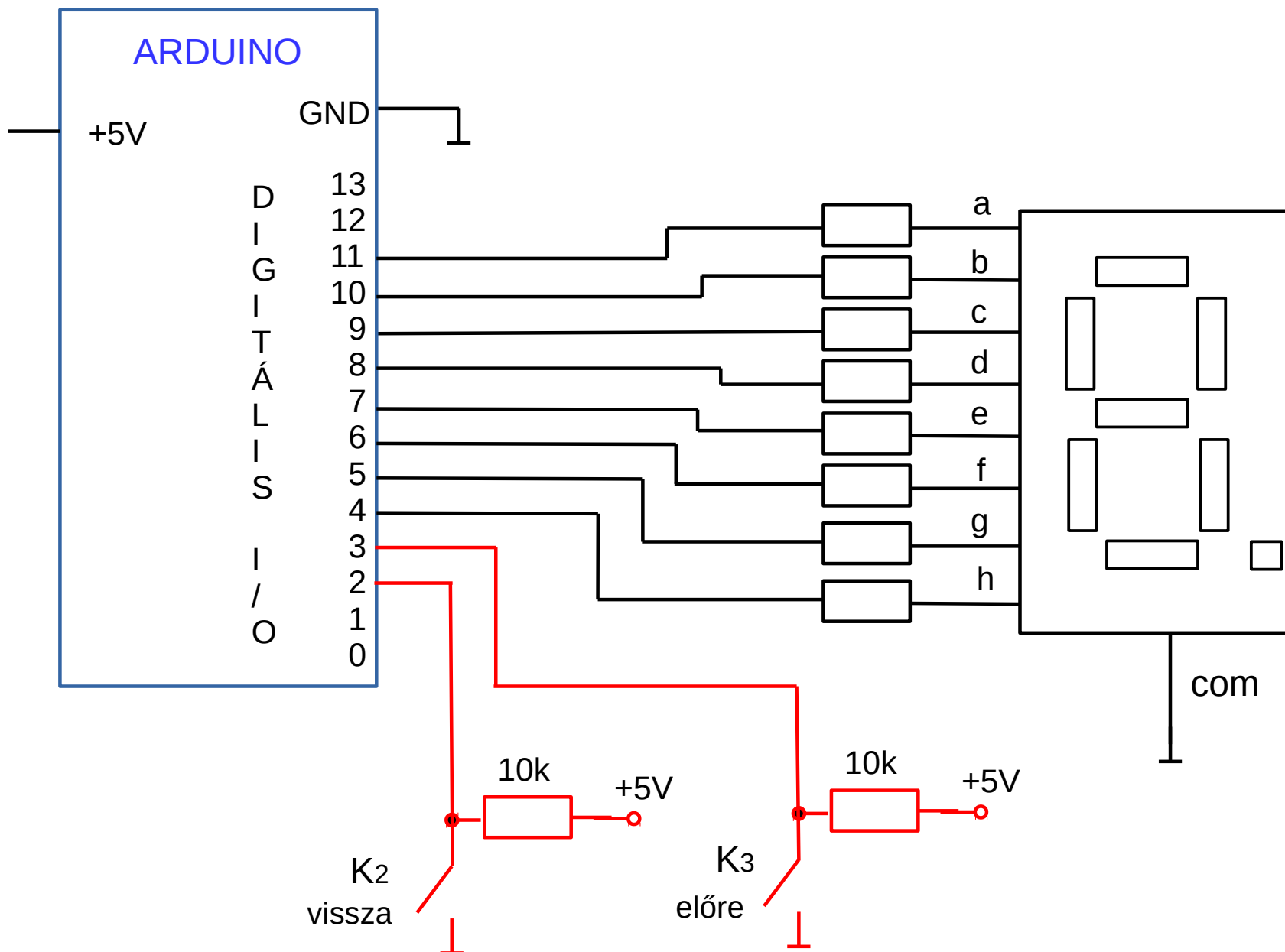
- A 0-2-4-6-8-0-2-4-6-8-0-... értékek kijelzése folyamatosan (2s-ig mindegyik)

3. feladat

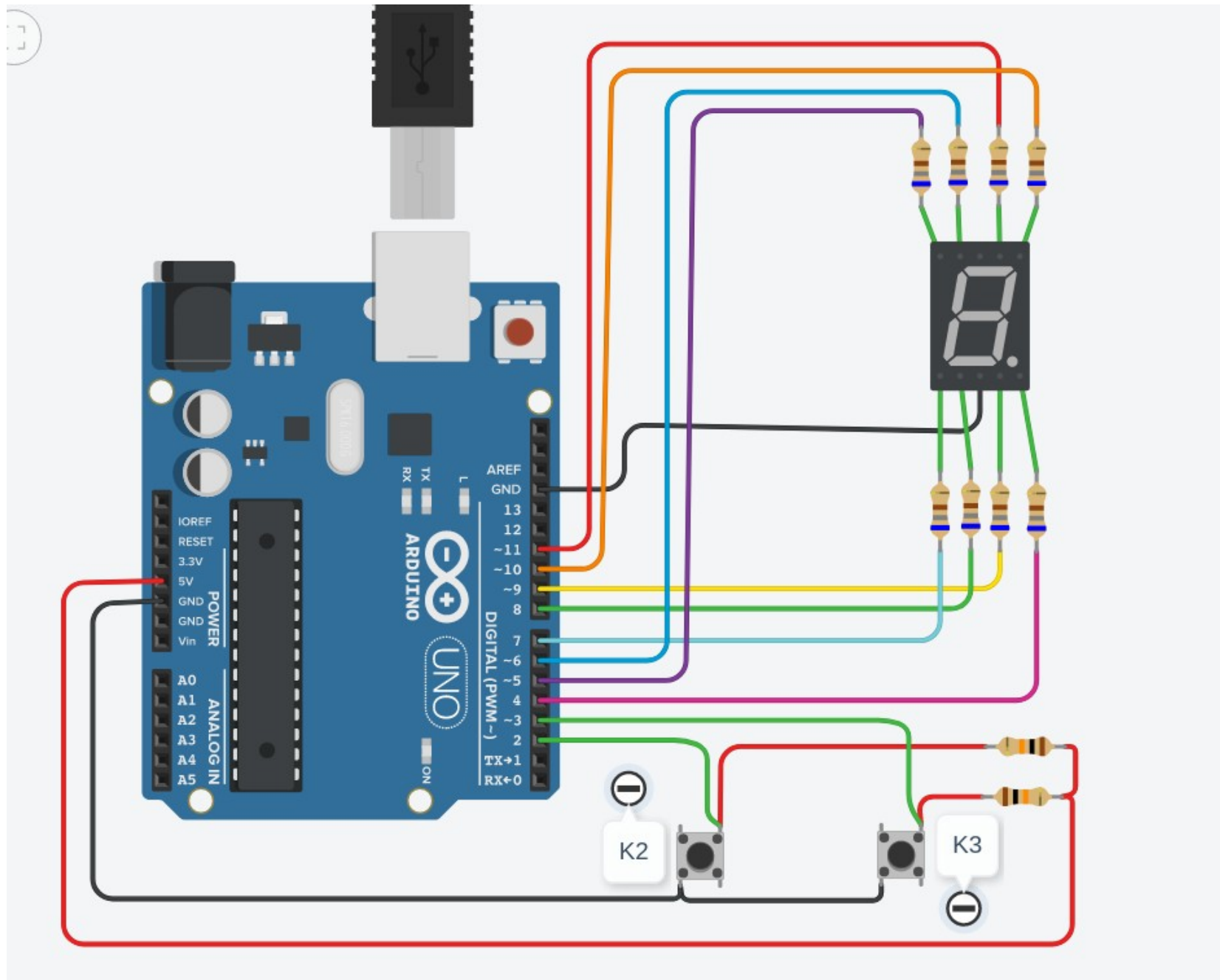
- Induláskor a 0-9-0-9 értékek kijelzése egymás után (1s-ig mindegyik)
- Majd ezután folyamatos számlálás 0-1-2-3-4-5-6-7-8-9-0-1-2-...
(2s-ig mindegyik)

11.1. Hétszegmenses kijelző 2.

Hétszegmenses kijelző (közös katódú) digitális kimeneteken



11.2. Hétszegmenses kijelző 2.



11.3. Hétszegmenses kijelző vezérlése 2.

1. mintafeladat

A kijelzőn sorban számoljunk !

- vagy előre: 1-2-3-1-2-3-1- ... vagy visszafelé: 3-2-1-3-2-1-...
- A két nyomógombbal tudunk a számlálás irányán változtatni
- egy változóban tároljuk, hogy melyik gomb volt legutoljára lenyomva (most éppen melyik irányba kell számolni)
- hogy egyszerűen tudjuk kezelni a számok kiírását, az előzőekben megírt, minden számra külön-külön függvényeket összevonjuk egy függvénybe, és egy paraméterrel adjuk meg, hogy éppen melyik számot kell kiírni !

// globális változók a szegmensek és nyomógombok lábakhoz rendelésére

```
byte a7=11;    // 'a' szegmens 11. lábon van
byte b7=10;    // 'b' szegmens 10. lábon
byte c7=9;     // 'c' szegmens
byte d7=8;     // 'd' szegmens
byte e7=7;     // 'e' szegmens
byte f7=6;     // 'f' szegmens
byte g7=5;     // 'g' szegmens
byte h7=4;     // 'h' szegmens
byte k2=2;     // hátra nyomógomb a 2-es lábon
byte k3=3;     // előre nyomógomb
```

// globális változó (számláló), kezdőérték adással

```
byte i=0;
// változó annak tárolására, hogy éppen előre vagy
// visszafelé kell számolni (1 - előre, 0 - hátra)
byte elore=1;
```

11.4. Hétszegmenses kijelző vezérlése 2.

1. mintafeladat folytatása

// Megmondjuk melyik lábak kimenetek (a hétszegmenses kijelzőt vezérlők)
// illetve melyik lábak bemenetek (amelyekre a nyomógombokat kötöttük)

```
void setup( )  
{  
    pinMode(a7, OUTPUT);    // 'a7' kimenet lesz  
    pinMode(b7, OUTPUT);    // 'b7' kimenet lesz  
    pinMode(c7, OUTPUT);    // 'c7' kimenet lesz  
    pinMode(d7, OUTPUT);    // 'd7' kimenet lesz  
    pinMode(e7, OUTPUT);  
    pinMode(f7, OUTPUT);  
    pinMode(g7, OUTPUT);  
    pinMode(h7, OUTPUT);  
    pinMode(k2, INPUT);      // 'k2' bemenet lesz (nyomógomb)  
    pinMode(k3, INPUT);      // 'k3' bemenet lesz (nyomógomb)  
}
```

11.5. Hétszegmenses kijelző vezérlése 2.

1. mintafeladat folytatása

```
void loop()
{
    szam7szegm(i);           // számot kijelző függvény hívása
    delay(1000);

    // nyomógombok lekérdezése, és állapot változó állítása, ha kell
    if(digitalRead(k3)==0) előre=1;    // előre számolunk
    if(digitalRead(k2)==0) előre=0;    // visszafelé számolunk
    if(előre==1)               // számolás előre
    {
        i++;                  // "i" változó növelése 1-el
        if(i>3)               // ha i>3
            { i=0; }          // akkor "i" értéke legyen 0
    }
    else if(előre==0)          // számolás visszafelé
    {
        if(i>0)               // ha i>0
            { i--; }          // akkor "i" csökkentése
        else i=3;
    }
}
```

11.6. Hétszegmenses kijelző vezérlése 2.

1. mintafeladat folytatása

```
void szam7szegm(byte x)
{
    if(x==1) {
        digitalWrite(a7, LOW); digitalWrite(b7, HIGH);
        digitalWrite(c7, HIGH); digitalWrite(d7, LOW);
        digitalWrite(e7, LOW); digitalWrite(f7, LOW);
        digitalWrite(g7, LOW); digitalWrite(h7, LOW); }

    else if(x==2) {
        digitalWrite(a7, HIGH); digitalWrite(b7, HIGH);
        digitalWrite(c7, LOW); digitalWrite(d7, HIGH);
        digitalWrite(e7, HIGH); digitalWrite(f7, LOW);
        digitalWrite(g7, HIGH); digitalWrite(h7, LOW); }

    else if(x==3) {
        digitalWrite(a7, HIGH); digitalWrite(b7, HIGH);
        digitalWrite(c7, HIGH); digitalWrite(d7, HIGH);
        digitalWrite(e7, LOW); digitalWrite(f7, LOW);
        digitalWrite(g7, HIGH); digitalWrite(h7, LOW); }
}
```

11.7. Feladatok

Az előbbi minta feladatot bővítsd ki úgy, hogy a többi számot is meg tudjuk jeleníteni (4, 5, ... 9, 0)

Írj programokat a 7 szegmenses kijelző vezérlésére

1. feladat

- folyamatos számlálás előre (0-1-2-3-4-5-6-7-8-9-0-1-2-...) vagy hátra,
- a számlálás irányát a két nyomógombbal tudjuk állítani

2. feladat

- Induláskor a 0-9-0-9 értékek kijelzése egymás után (1s-ig mindegyik)
- ezután folyamatos számlálás előre (0-1-2-3-4-5-6-7-8-9-0-1-2-...)
- a K2 nyomógomb lenyomására a számlálás leáll ! (stop)
- a K3 nyomógomb lenyomására a számlálás újra elindul előre! (start)

12.1. Tömbök

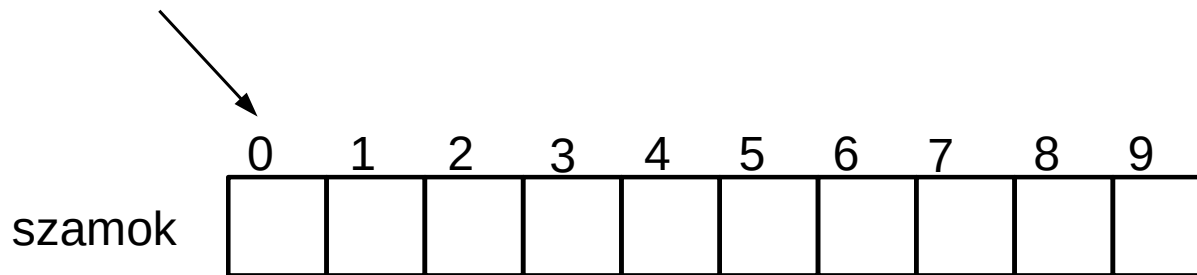
Tömb

- olyan változó, amely sok azonos típusú elemet (pl. egész számot) tárol
- tömb deklarációja

típus tömb_név[elemek_száma];

pl.

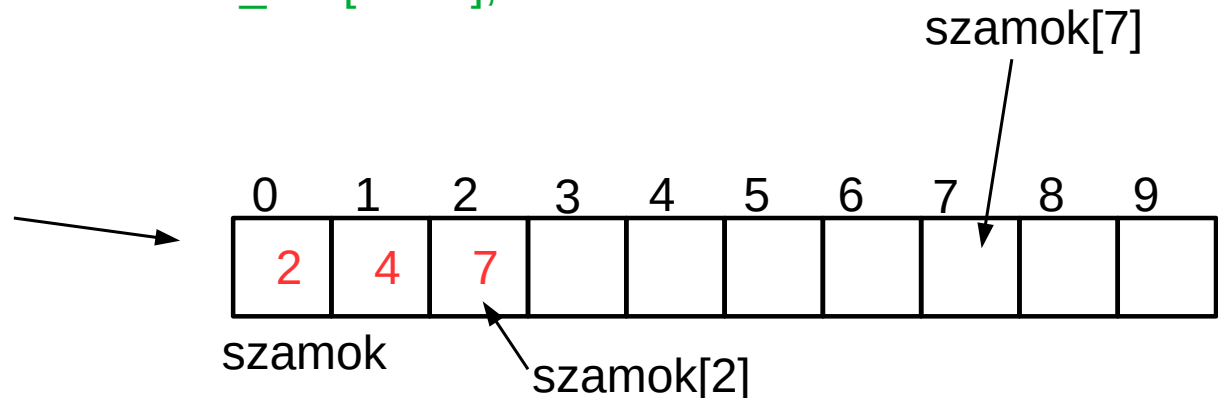
int számok[10]; // 10 elemű tömb, 10db egész szám tárolására
 az elemek számozása 0-val kezdődik !!!



- hivatkozás a tömb elemeire → tömb_név[index]
 értékkadás → tömb_név[index]=érték;
 kiolvasás → változó=tömb_név[index];

pl.

számok[0]=2;
számok[1]=4;
számok[2]=számok[1]+3;



12.2. Tömb használata

Tömb és ciklus

- ha az összes tömbelemet akarjuk kiolvasni vagy értéket adni nekik → ciklus felhasználásával tudunk egyszerűen végig lépkedni a tömb elemeken
pl.

```
byte szamok[10];           // 10 elemű tömb létrehozása
i=0;                       // index változó
while (i<10)               // ismétlés 10-szer, i=0,1,2,...9
{
    szamok[i]=2*i;         // tömbelem értéke legyen 2*index
    i++;                  // index növelése (következő ciklus számára)
}
```



	0	1	2	3	4	5	6	7	8	9
szamok	0	2	4	6	8	10	12	14	16	18

12.3. Tömb használata

Tömb, kezdőérték adás

- a tömb deklarálásakor azonnal megadhatjuk a tömbelemek értékeit is, ha már ismertek, így nem kell a feltöltéssel később vesződni

típus tömb_név[elemek_száma]={1.elem, 2.elem, 3.elem, ... k.elem};

vagy típus tömb_név[]={1.elem, 2.elem, 3.elem, ... k.elem};

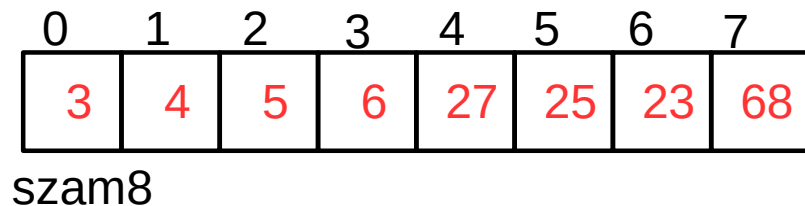
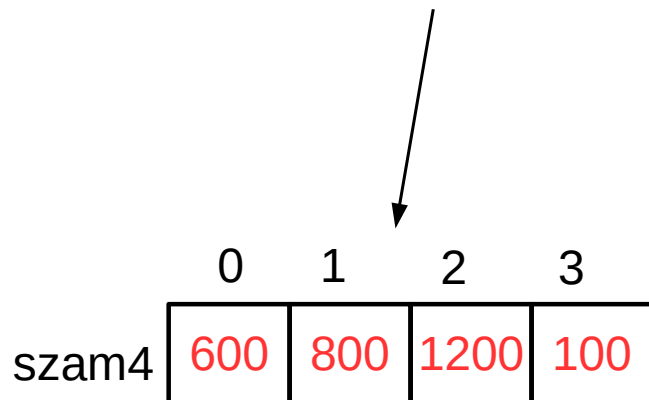
pl.

byte szam8[]={3,4,5,6,27,25,23,68};

// 8 elemű tömb létrehozása

int szam4[4]={600,800,1200,100};

// 4 elemű tömb létrehoz



12.4. Bitműveletek

Bitműveletek

- bitenkénti ÉS (AND): $\&$ `szam1&szam2`
pl. `0b1011&0b1101 → 0b1001`
- bitenkénti VAGY (OR): $|$ `szam1|szam2`
pl. `0b1010|0b0100 → 0b1110`
- bitenkénti NEM (NOT): \sim `~szam1`
pl. `~0b1011 → 0b0100`
- kizáró VAGY (XOR): \wedge `szam1^szam2`
pl. `0b1110^b0100 → 0b1010`
- eltolás balra (left shift) $<<$ `szam1<<x` *// eltolás ← x bittel*
pl. `0b00001001<<2 → 0b00100100`
- eltolás jobbra (right shift) $>>$ `szam1>>x` *// eltolás → x bittel*
pl. `0b11001010>>3 → 0b00011001`

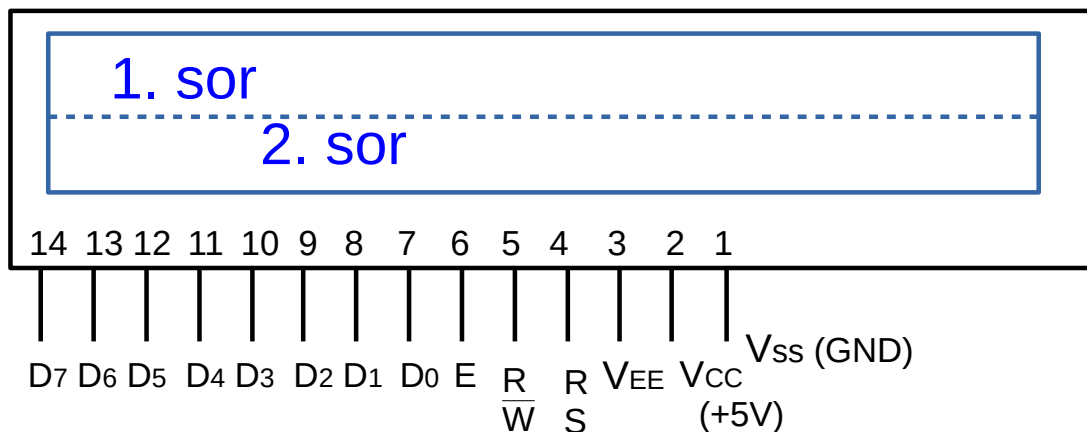
13.1. Alfánumerikus LCD kijelző

Alfánumerikus LCD kijelzők

- LCD (folyadék kristályos) kijelző sokféle létezik, most csak az alfánumerikus kijelzőkkel foglalkozunk, azok közül is a HD44780 (kvázi ipari szabvány) vezérlővel ellátott kijelző modulokkal
 - ebből is létezi többféle: 2 soros, 4 soros, ... → 2x16, 4x20, .. karakteres
 - A 2x16 karakteres kijelző programozását fogjuk áttekinteni
- Erről van egy jó leírás a következő oldalon → <http://esca.atomki.hu/PIC24/lcd.html>

2x16 karakteres LCD kijelző

- 16 vagy 14 kivezetéssel rendelkeznek



adat vezetékek

- 8 bites mód → D0-D7 (vagy DB0-DB7) lábak
- 4 bites mód → D4-D7 (vagy DB4-DB7) lábak

V_{EE} kontraszt beállítása → 0-5V közötti feszültséggel (poti)

E (vagy EN) - engedélyezés (órjel)

- 6-os láb
- egy ideig (0,3-0,5us) 1-es szintre kell állítani (a pozitív élre olvassa be a bemeneteket)

RS - Regiszter kiválasztása

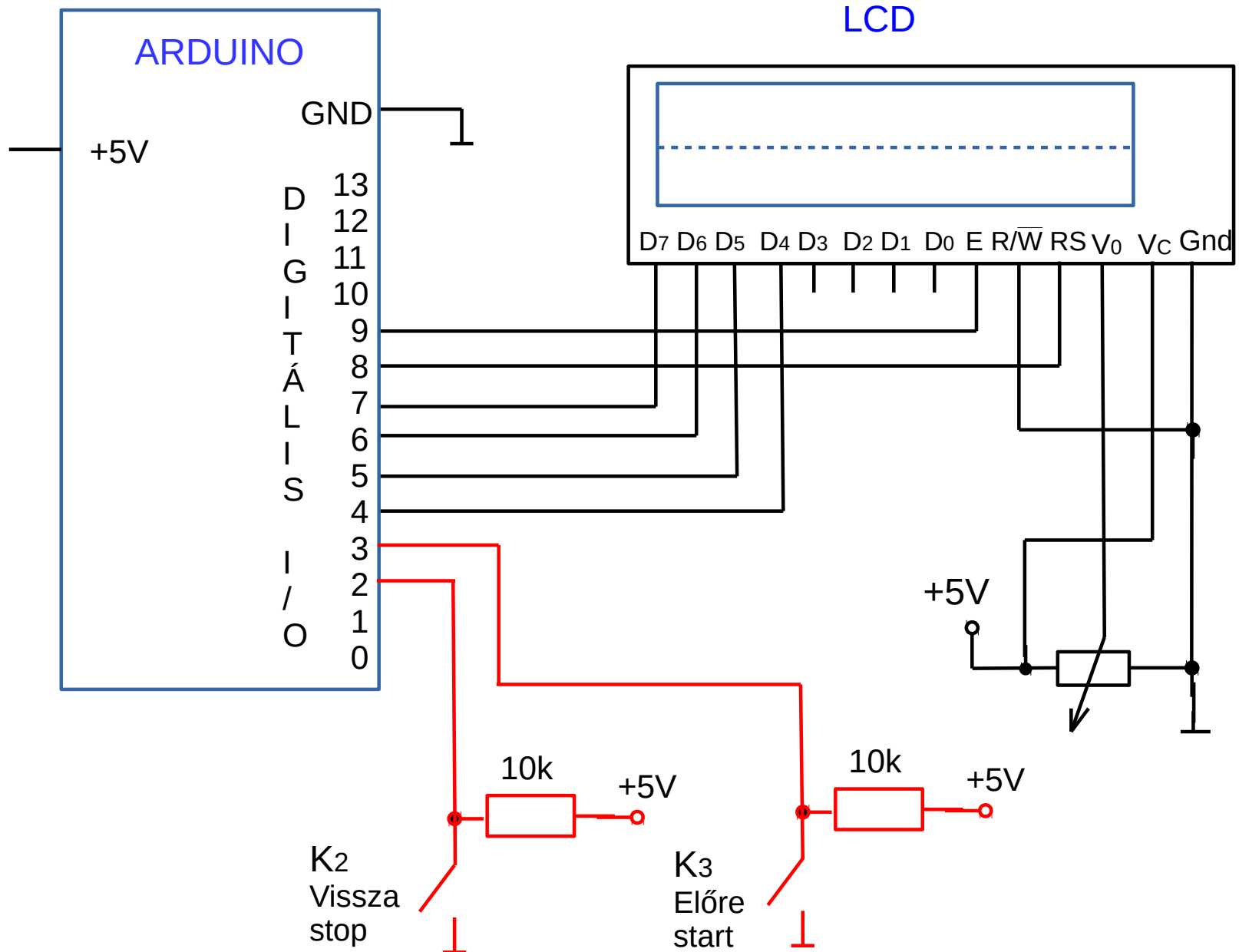
- RS=0 → vétel utasítás regiszterbe (IR) → parancs
- RS=1 → vétel az adat regiszterbe (DR) → adat

R/W – olvasás/írás mód kiválasztása

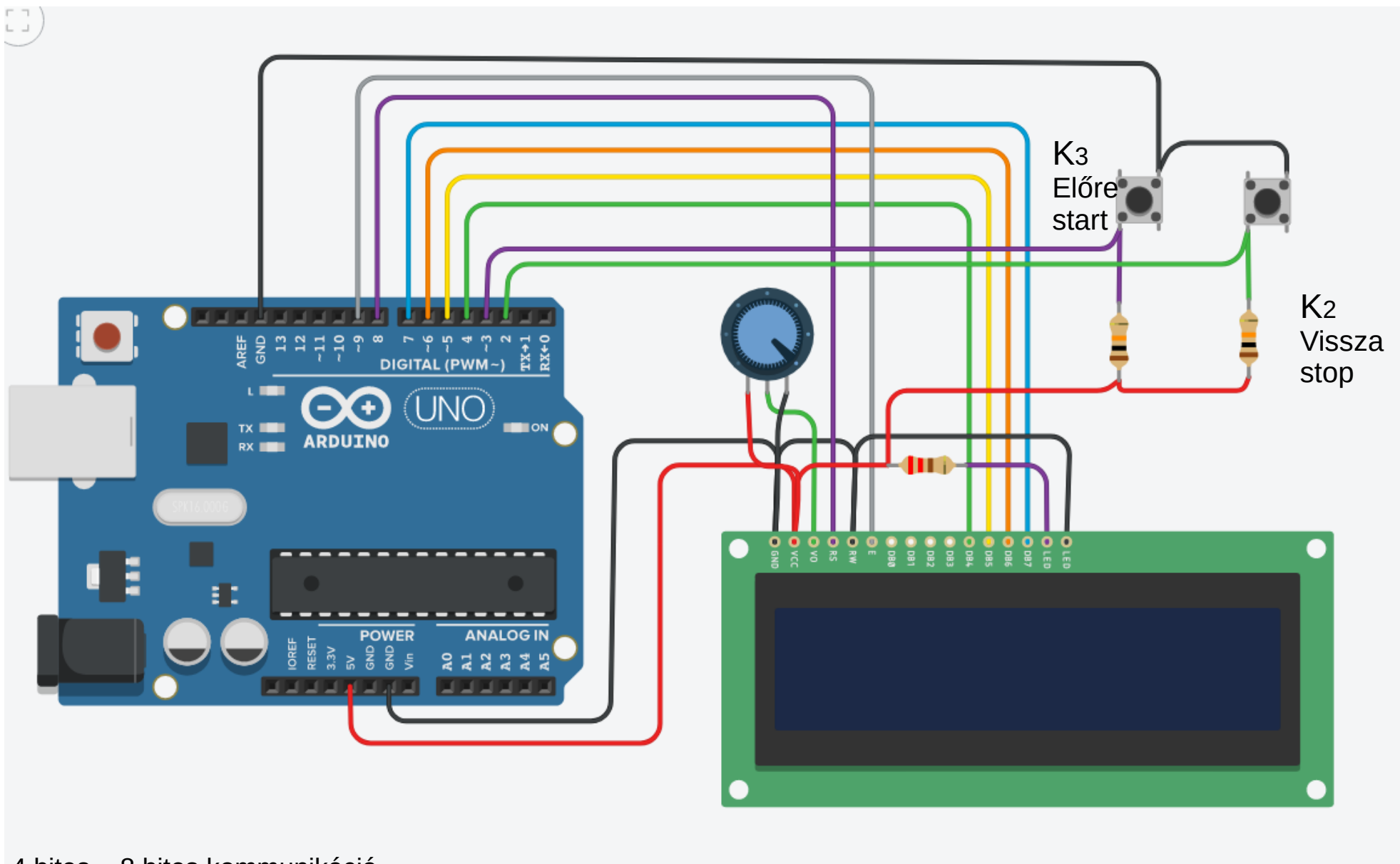
- R/W=1 → adatok olvasása a kijelzőről
- R/W=0 → adatok küldése a kijelzőnek

13.2. LCD kijelző vezérlése

A hardver



13.3. LCD kijelző vezérlése



4 bites – 8 bites kommunikáció

- A kijelző modullal kommunikálhatunk 4 (D7-D4) vagy 8 (D7-D0) adat vezetéken → általában a 4 biteset használjuk mert így a mikrovezérlőnél 4 lábbal kevesebb is elég a kijelző vezérléséhez !!
- De mindkét esetben 8 bites számokat küldünk át ! → 4 bites kommunikáció esetén két lépésben visszük át a 8 bitet → először a felső 4 bit, majd az alsó 4 bit

13.4. LCD vezérlő függvények, objektumok

„LiquidCrystal.h”

Először a programunk elején ezt a header állományt be kell importálnunk, hogy elérjük a szükséges Arduino-s objektumokat, függvényeket !

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

„LiquidCrystal” objektum

Ezután létre kell hozni egy LCD kezelő objektumot, megadva hogy az LCD kijelző kivezetéseit melyik Arduino-s lábakkal akarjuk vezérelni (RS, EN, D4, D5, D6, D7).

Az objektum metódusaival (lényegében függvények) tudjuk vezérelni a kijelzőt !

```
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
```

„begin” metódus

Az lcd objektum típusát, üzemmódját tudjuk beállítani:

```
lcd.begin(16, 2);           // 2 sor, soronként 16 karakter
```

„print” metódus

Szöveg vagy változó kiírása az aktuális kurzor pozíciótól kezdve

```
lcd.print("Hello !");
```

„setCursor” metódus

Kurzor pozíció beállítása (oszlop, sor), a számozás nullával kezdődik !

```
lcd.setCursor(3, 0);        // kurzor pozíció az 1. sor 4. karakternél
```

13.5. LCD kijelző feladatok

1. mintafeladat

- A kijelzőn első sorában jelenítsünk meg egy üdvözlő üzenetet
- a 2. sorban pedig folyamatosan számoljunk → 0-1-2-3-4-5-6-7-8-9-10-11- ...

```
// ezt be kell importálni a kijelző használatához !!
```

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

```
// LCD kijelző bekötése (RS, EN, D4, D5, D6, D7)
```

```
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
```

```
// változó a számláláshoz
```

```
byte szam=0;
```

```
void setup()
```

```
{
```

```
    lcd.begin(16, 2);
```

```
    // 16x2 karakter/soros LCD kijelző beállítása
```

```
    delay(1000);
```

```
    // késleltetünk 1000ms-t
```

```
    lcd.print("Hello !");
```

```
    // szöveg kiírása (1. sorban)
```

```
    delay(2000);
```

```
}
```

13.6. LCD kijelző feladatok

1. mintafeladat folytatása

// lcd.setCursor(7, 1); → kurzor pozíció megadása
// az első paraméter a karakter helye, másik a sor (a számolás 0-val indul !!)

```
void loop()
{
```

```
    lcd.setCursor(7, 1);    // kurzor pozíció 2. sor 8. karakter
    lcd.print("      ");    // előző érték törlése
    lcd.setCursor(7, 1);    // kurzor pozíció 2. sor 8. karakter
```

```
    lcd.print(szam);        // változó értékének kiírása
    szam++;                 // növelés eggyel
    delay(500);             // kicsit késleltetünk
```

```
}
```

13.7. LCD kijelző feladatok

2. mintafeladat

- A kijelzőn első sorában jelenítsünk meg egy üdvözlő üzenetet, a 3. pozíciótól !
- a 2. sorban pedig folyamatosan számoljunk 5 és 30 között ! → 5-6-7-8- ... -30-5-6- ...

// ezt be kell importálni a kijelző használatához !!

#include <LiquidCrystal.h>

// LCD kijelző bekötése (RS, EN, D4, D5, D6, D7)

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

// változó a számláláshoz

byte szam=5; // 5-től számolunk !!

void setup()

{

 lcd.begin(16, 2); // 16x2 karakter/soros LCD kijelző beállítása

 delay(1000); // késleltetünk 1000ms-t
 lcd.setCursor(2, 0); // kurzor pozíció 1. sor 3. karakter
 lcd.print("Hello !"); // szöveg kiíratása (1. sorban)
 delay(2000);

}

13.8. LCD kijelző feladatok

2. mintafeladat folytatása

// lcd.setCursor(0, 1); → kurzor pozíció megadása
// az első paraméter a karakter helye, másik a sor (a számolás 0-val indul !!)

```
void loop()
{
```

```
    lcd.setCursor(0, 1);    // kurzor pozíció 2. sor 1. karakter
    lcd.print("      ");    // előző érték törlése
    lcd.setCursor(0, 1);    // kurzor pozíció 2. sor 1. karakter
```

```
    lcd.print(szam);        // változó értékének kiírása
    szam++;                 // növelés eggyel
    if(szam>30) szam=5;     // kezdés előlről
    delay(500);             // kicsit késleltetünk
```

```
}
```

13.9. LCD kijelző feladatok

3. mintafeladat

- A kijelző első sorában jelenítsünk meg egy üdvözlő üzenetet, kb. középen, 5s ideig !
- majd ezután a 2. sorban folyamatosan számoljunk 0 és 500 között !
 - 0-1-2-3-4- ... -499-500-0-1 ...
- Eközben a felső sorban a „Run” üzenet legyen látható !
- K2 lenyomására a számlálás álljon le, mutatva a legutolsó értéket, és a felső sorban az „Stop” üzenet legyen !
- K3 lenyomására a számlálás folytatódjon !

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
byte k2=2;           // k2 (stop) nyomógomb a 2-es lábon
byte k3=3;           // k3 (start) nyomógomb

int szam=0;          // számláló, de mivel 500-ig számolunk, a byte típus nem jó !!
byte szamol=1;       // állapot tároló, számolunk (1), vagy állunk (0)

void setup()
{
    lcd.begin(16, 2);           // 16x2 karakter/soros LCD kijelző beállítása
    pinMode(k2, INPUT);         // 'k2' bemenet lesz (nyomógomb)
    pinMode(k3, INPUT);         // 'k3' bemenet lesz (nyomógomb)
    delay(1000);                // késleltetünk 1000ms-t
    lcd.setCursor(5, 0);        // kurzor pozíció 1. sor 6. karakter
    lcd.print("Hello !");       // szöveg kiírása (1. sorban)
    delay(5000);

}
```

13.10. LCD kijelző feladatok

3. mintafeladat folytatása

```
void loop()
{
    lcd.setCursor(0, 1);           // kurzor pozíció 2. sor 1. karakter
    lcd.print("      ");          // előző érték törlése
    lcd.setCursor(0, 1);           // kurzor pozíció 2. sor 1. karakter
    lcd.print(szam);               // változó értékének kiírása

    // nyomógombok lekérdezése, és állapot változó állítása, ha kell
    if(digitalRead(k3)==0) szamol=1; // számolunk
    if(digitalRead(k2)==0) szamol=0; // nem számolunk

    if(szamol==1)                 // ha számolunk
    {
        szam++;                  // "szam" változó növelése 1-el
        if(szam>500)              // ha elértük a végét
            { szam=0; }          // akkor kezdjük előlről
    }

    lcd.setCursor(5, 0);          // kurzor pozíció 1. sor 6. karakter
    lcd.print("      ");          // előző érték törlése
    lcd.setCursor(5, 0);          // kurzor pozíció 1. sor 6. karakter
    if(szamol==1) lcd.print("Run");
    if(szamol==0) lcd.print("Stop");
    delay(500);
}
```

13.11. Feladatok

1. feladat

Az LCD kijelzőn menjen induláskor egy számláló az 1. sor, 1. karakter pozícióban

0-1-2-3-4-5-6-7-8-9-0-1-2-...

- a K3 kapcsolót megnyomva a számlálás a 14. karakter pozíciójában folytatódjon,
- a K2 kapcsolót megnyomva a számlálás az 1. karakter pozíciójában folytatódjon

2. feladat

Induláskor →

- az 1. sorban írjunk ki egy üzenetet (pl. 'Helo'),
- a 2. sorban folyamatos számlálás 0 és 20 között előre (0-1-2-3-4-...-17-18-19-20-0-1-2-...),

Majd ezután →

- a K2 és K3 nyomógombokkal tudjuk a számlálás irányát (előre vagy hátra) állítani

3. feladat

- az 1. sorban írjunk ki egy üzenetet (pl. 'Helo'), majd a 2. sorban egy '0' számot az 1. karakterpozícióba
- a K3 nyomógombot nyomva tartva → a '0' lépjen egyet jobbra kb. 1 másodpercenként
- a K2 nyomógombot nyomva tartva → a '0' lépjen egyet balra kb. 1 másodpercenként

13.12. LCD kijelző vezérlése bővebben *

1. HD44780 LCD display parancsai

00000001 clear display

0000001x cursor return home

000001MS entry mode set

M-cursor move, 1--> right, 0--> left

S - display shift, 1--> yes, 0--> no

00001DCB display control

D - display, 1--> on, 0--> off

C - cursor, 1--> on, 0--> off

B - cursor blink, 1--> on, 0--> off

0001SDxx cursor/display shift

S-shift-move, 1--> display shift,

0--> cursor move

D-direction, 1--> right, 0--> left

001BRFxx funtion set

B - bit mode, 1--> 8bit, 0--> 4bit

R - row, 1--> 2 rows, 0--> 1 row

F - font, 1--> font 5x10, 0--> font 5x8

01aaaaaa set CGRAM address (user character ---> 5x8)

user character (max. 8):

1.row B4 B3 B2 B1 B0	pl. - - x - -	0b00100
2.row B4 B3 B2 B1 B0	- x x x -	0b01110
3.row B4 B3 B2 B1 B0	x - x - x	0b10101
4.row B4 B3 B2 B1 B0	- - x - -	0b00100
5.row B4 B3 B2 B1 B0	- - x - -	0b00100
6.row B4 B3 B2 B1 B0	- - x - -	0b00100
7.row B4 B3 B2 B1 B0	- - x - -	0b00100
8.row B4 B3 B2 B1 B0	- - x - -	0b00100

1aaaaaaa set DDRAM address **karakter pozíciók címei !!**

első sor 00 01 02 03 0F

második s. 40 41 42 4F

13.13. LCD kijelző vezérlése bővebben *

2. HD44780 LCD display vezérlése

Adatok küldése a kijelzőnek (írás) → R/ \overline{W} lábra 0 szint

A HD44780 vezérlőnek 2db 8 bites regiszterébe írhatunk, az RS lábra adott jellel választjuk ki, hogy melyikbe:

- ha parancsot akarunk küldeni → RS lábra 0 adása → a küldött 8 bit az IR regiszterbe kerül
- ha adatot akarunk küldeni → RS lábra 1 adása → a küldött 8 bit a DR regiszterbe kerül
- RS=0 → vétel utasítás regiszterbe (IR) → parancs
- RS=1 → vétel az adat regiszterbe (DR) → adat

Adatok kérése a kijelzőtől (olvasás) → R/ \overline{W} lábra 1 szint

Ha ezt a funkciót nem akarjuk használni akkor R/ \overline{W} láb fixen 0 szintre (GND) köthető !

- RS=0 esetén → D7 lábon a busy flag értékét, a D6-D0 lábakon a cím számláló értékét (kurzor pozíció) olvashatjuk
- RS= 1 esetén → ?

4 bites – 8 bites kommunikáció

- A kijelző modullal kommunikálhatunk 4 (D7-D4) vagy 8 (D7-D0) adat vezetéken → általában a 4 biteset használjuk mert így a mikrovezérlőnél 4 lábbal kevesebb is elég a kijelző vezérléséhez !!
- De mindkét esetben 8 bites számokat küldünk át ! → 4 bites kommunikáció esetén két lépésben visszük át a 8 bitet → először a felső 4 bit, majd az alsó 4 bit

13.14. LCD kijelző vezérlése bővebben *

3. HD44780 LCD display beállítási, vezérlési példa

4 bites üzemmódban 4 bit küldésének ütemezése

- R/W lábra 0 szint → írás
- RS lábra 0 szint (vagy 1) → parancs küldése (vagy adat)
- D7,D6,D5,D4 lábakra → a parancs vagy adat, felső vagy alsó 4 bitje
- kis késleltetés, ~ 50-100ns !!
- E lábra 1 szint → engedélyezés, kijelző beolvassa a biteket
- kis késleltetés, ~ 300-500ns !!
- E lábra 0 szint → adatfogadás tiltás
- kis késleltetés, ~ 20-50 !!

A nyolc bites parancsokat, adatokat tehát két részletben, hasonló ütemezéssel kell küldenünk

Üzem mód beállítás (kijelző inicializálása)

Mielőtt bármit is ki tudnánk írni be kell állítani a használni kívánt üzemmódot

- legelőször RS=0 és 0010 kód küldése → megmondjuk, hogy 4 bites üzemmódot használunk
(ez a parancs felső 4 bitje, az alsó 4 bitjét most nem küldjük el !)
- RS=0 és 00101100 kód küldése → 4 bites üzemmód, 2 sor, karakter méret 5x10
- RS=0 és 00001100 kód küldése → Display ON, Cursor Off
- RS=0 és 00000001 kód küldése → Display törlése
- RS=0 és 00000110 kód küldése → Entry Mode, Increment cursor position, No display shift

Karakter kiírása a kijelzőre

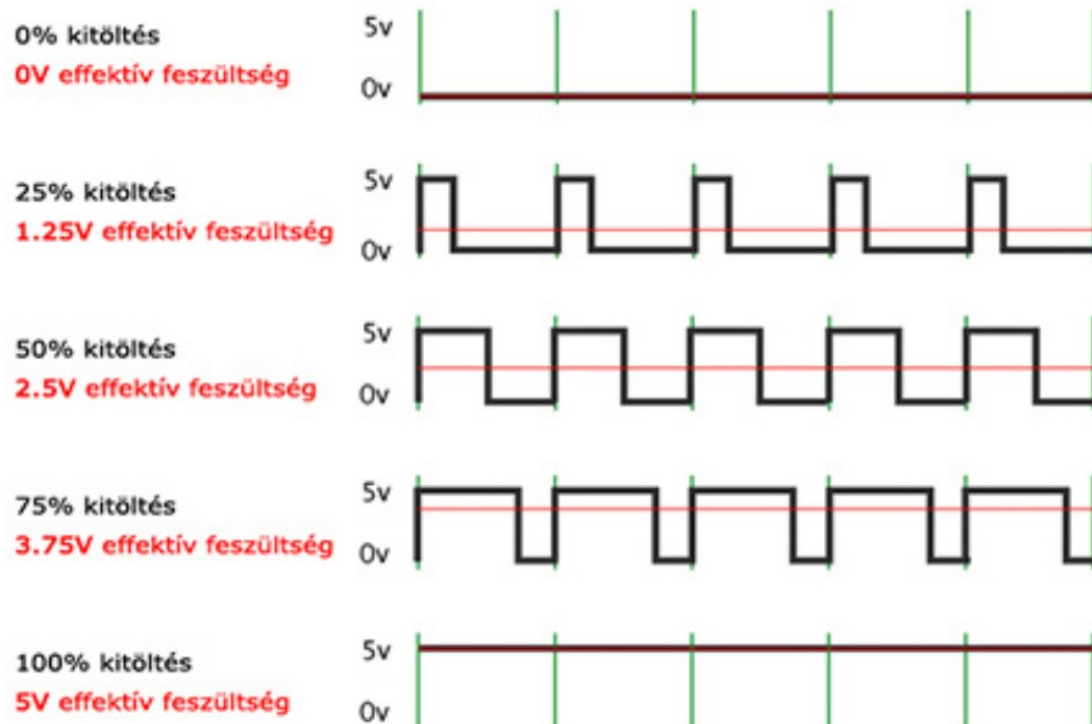
Kétféleképpen lehetséges

- először egy parancsként elküldjük a karakter pozíciójának címét (sor, karakter) → 1aaaaaaaa
majd utána a karakter 8 bites ASCII kódját
- vagy egyből a karakter 8 bites ASCII kódját küldjük → ekkor az aktuális kurzor pozícióba íródik
pl. RS=0 és 10000000 kód küldése → karakter pozíció beállítása 1. sor 1. karakter
RS=1 és 00110000 kód küldése → '0' karakter kiírása

14.1. PWM

PWM

- PWM (Pulse Width Modulation → impulzus szélesség moduláció)
- Egy négyszögjel kitöltési tényezőjét változtatjuk ! (mennyi ideig van magas szinten ill. alacsony szinten)
- A négyszögjel frekvenciája és amplitúdója nem változik.
- a kitöltési tényezőt százalékban szokták megadni. Egy 50%-os PWM jel azt jelenti, hogy a jel az idő felében be, míg a másik felében ki van kapcsolva.
- Az effektív feszültség a kitöltési tényezővel lesz arányos ! → tehát a teljesítményt tudjuk így „analóg” módon változtatni



forrás:
Ruzsinszki Gábor:
Programozható Elektronikák

14.2. PWM Arduinóval

„analogWrite” függvény

Digitális kimeneten hardveres PWM jel küldése.

A digitális kimenetként beállított kivezetésre a 2. paraméter alapján beállított kitöltési tényezőjű PWM jelet ad ki.

Meghívása: `analogWrite(pin, value);`

value: 8 bites szám, 0 – 255 !!

0 → 0%-os kitöltési tényező

255 → 100%-os kitöltési tényező

pl. `analogWrite(3, 80);` // 3. lábon $100 \cdot 80/255 \sim 31\%$ -os kitöltési tényező

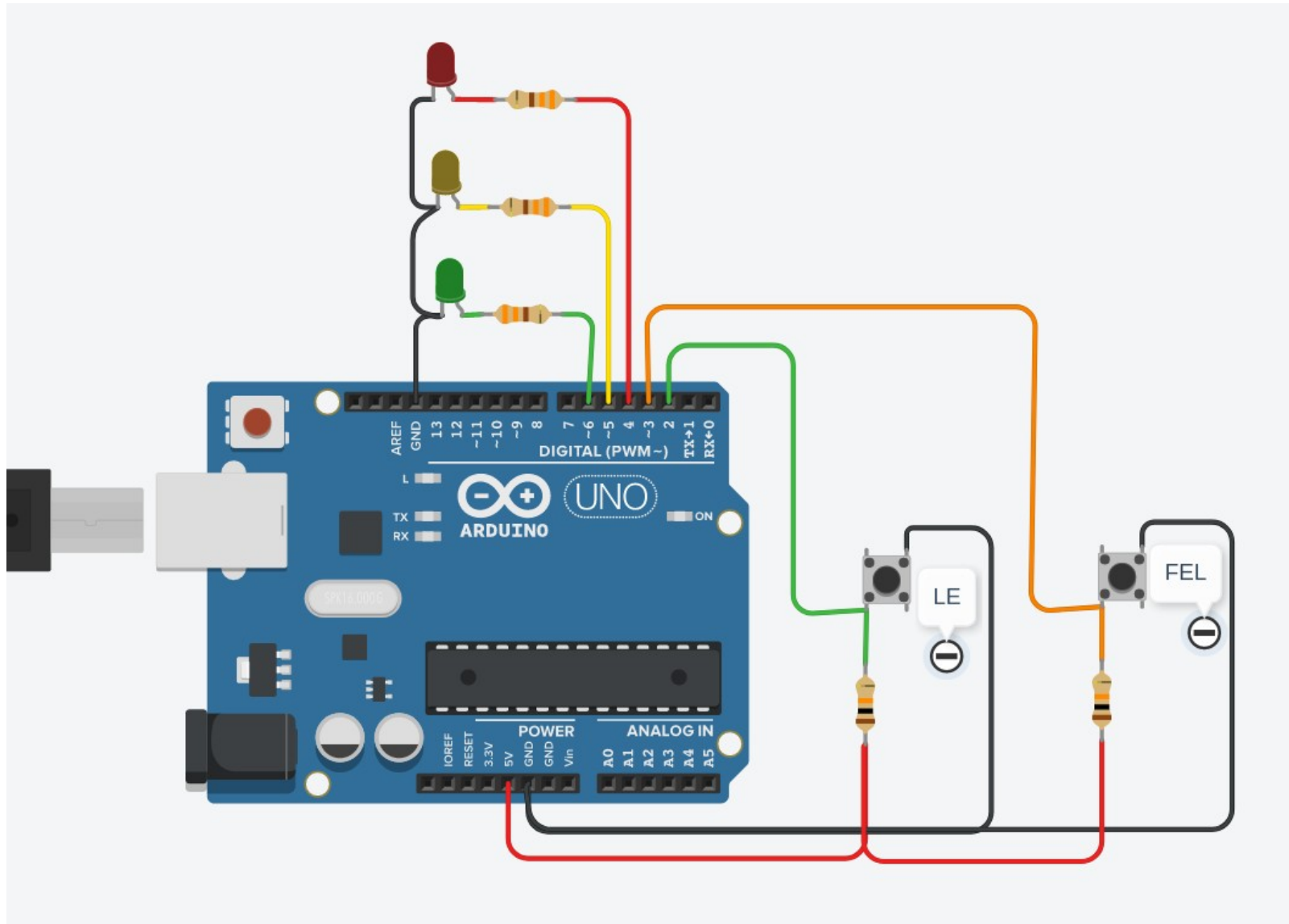
A használni kívánt lábat előtte kimenetként kell beállítani !

Nem minden láb képes PWM kimenetre !! (Arduino típustól is függ)

pl. az Uno a 3,5,6,9,10,11 lábakat tudja ilyen célra használni
(a lábak száma melletti hullámvonal jelöli)

14.3. PWM kimenetek

LED PWM vezérlése, hardver



A zöld és sárga LED-et (6. és 5. láb) tudjuk hardveres PWM jellel vezérelni, a 4-es lábon lévő pirosat nem !

14.4. PWM kimenetek

1. mintafeladat

A zöld LED fényereje először folyamatosan növekedjen, majd csökkenjen !

```
byte zold=6;
byte k=1;      // kitöltési tényező
void setup()
{
    pinMode(zold, OUTPUT);    // 'zöld' kimenet lesz
    delay(1000);
}
void loop()
{
    k=0;
    while(k<245)
    {
        analogWrite(zold,k);    // PWM jel beállítása
        delay(100);
        k=k+10;                 // kitöltési tényező növelése
    }
    k=255;
    while(k>10)
    {
        analogWrite(zold,k);    // PWM jel beállítása
        delay(100);
        k=k-10;                 // kitöltési tényező csökkentése
    }
}
```

14.5. PWM kimenetek

2. mintafeladat

- ha a 'FEL' gombot nyomjuk le → sárga LED fényereje folyamatosan növekedjen !
- ha a 'LE' gombot nyomjuk le → sárga LED fényereje folyamatosan csökkenjen !

```
byte sarga=5; // LED az 5-ös lábon  
byte fel=3;   // nyomógomb a 3-as lábon  
byte le=2;    // nyomógomb a 2-es lábon
```

```
byte k=0;      // kitöltési tényező  
byte felvagyle=0; // változó, annak tárolására, hogy mit kell csinálni  
                // 0 - semmit, 1 – fényerő növelés, 2 – fényerő csökkentés
```

```
void setup()  
{  
    pinMode(sarga, OUTPUT); // 'sarga' kimenet lesz  
    pinMode(fel, INPUT);    // 'fel' bemenet lesz  
    pinMode(le, INPUT);     // 'le' bemenet lesz  
    digitalWrite(sarga, LOW);  
    delay(1000);  
}
```

14.5. PWM kimenetek

2. mintafeladat, folytatás

```
void loop()
{
    if(digitalRead(fel)==0)           // 'FEL' lenyomva
        { felvagyle=1; }
    if(digitalRead(le)==0)            // 'LE' lenyomva
        { felvagyle=2; }

    if(felvagyle==1)                  // fényerő növelés
        if(k<245) k=k+10;             // kitöltési tényező növelése

    if(felvagyle==2)                  // fényerő csökkentés
        if(k>10) k=k-10;              // kitöltési tényező csökkentése

    analogWrite(sarga,k);             // PWM jel beállítása

    delay(200);

}
```

14.6. Feladatok

1. feladat

- indulás után 5-ször villanjon fel (0,5 sec.) a piros LED, majd kapcsoljuk fel a zöld LED-et !
- ha a 'LE' gombot nyomjuk le → zöld LED fényereje folyamatosan csökkenjen !

2. feladat

- ha a 'FEL' gombot nyomjuk le → a zöld LED fényereje folyamatosan növekedjen !
- ha a 'LE' gombot nyomjuk le → a zöld LED egyből kapcsoljon ki !

3. feladat

- ha a 'FEL' gombot nyomjuk le → a zöld LED egyből kapcsoljon be !
- ha a 'LE' gombot nyomjuk le → a zöld LED fényereje folyamatosan csökkenjen, ezalatt a piros LED villogjon !

15.1. Hangok Arduinóval

„tone” függvény

Digitális kimeneten megadott frekvenciájú négyszögjel kiadása.

Meghívása: `tone(pin, frequency);` // a frekvencia Hertz-ben !!
// amíg le nem kapcsoljuk addig megy !

vagy: `tone(pin, frequency, time);` // + időtartam ezredmásodpercben (ms)
// csak annyi ideig szól

pl. `tone(4, 800);` // 4. lábra 800 Hz-es jel felkapcsolása

pl. `tone(2, 300, 4000);` // 2. lábra 300 Hz-es jel felkapcsolása 4 másodpercig

„noTone” függvény

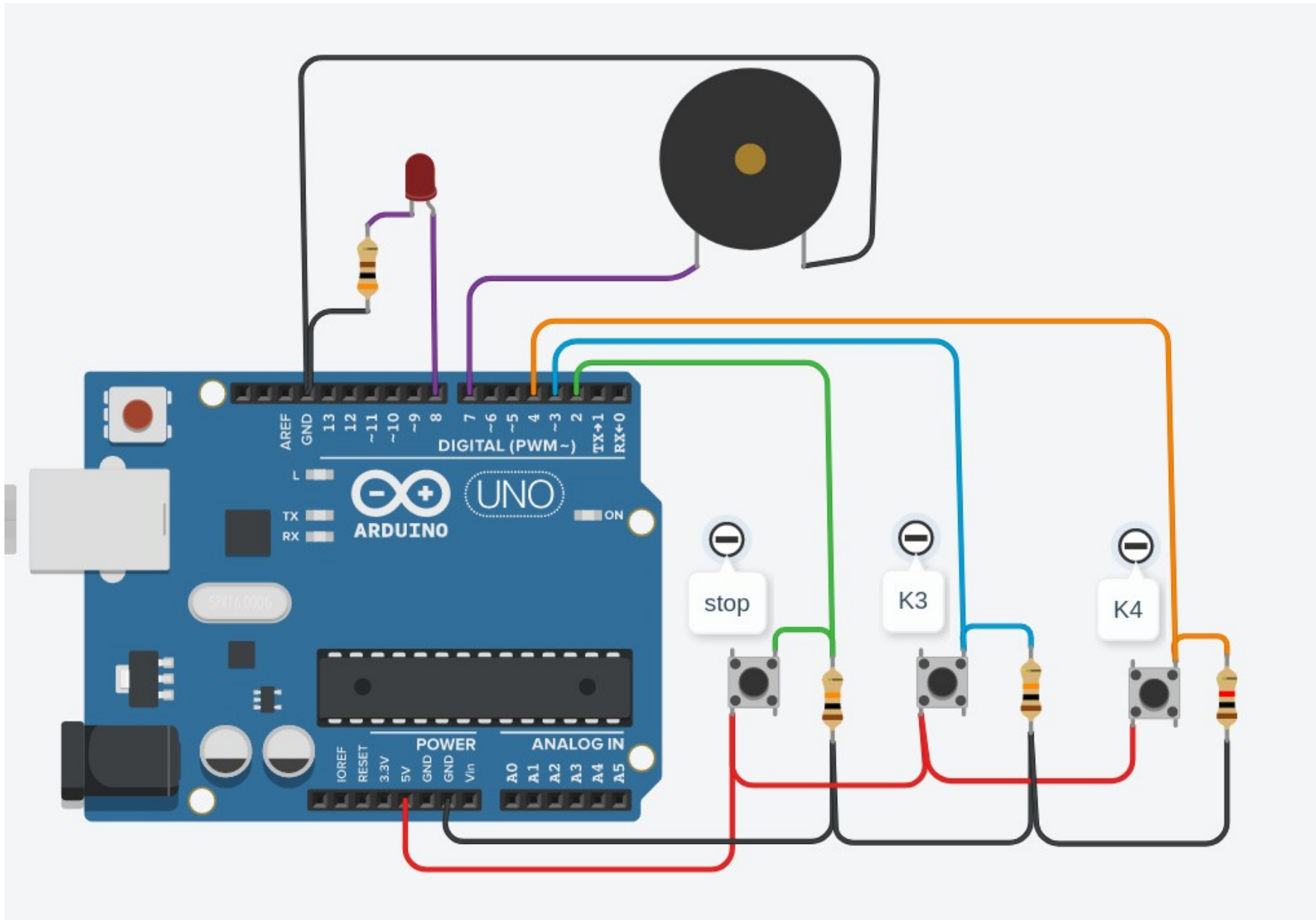
A digitális kimenetre kiadott négyszögjel lekapcsolása

Meghívása: `noTone(pin);`

pl. `noTone(4);` // a 4. lábra kiadott jel lekapcsolása

15.2. Hangok Arduinóval

Piezo hangjelző vezérlése, hardver



15.3. Hangok Arduinóval

1. mintafeladat

Két nyomógommbal (K3, K4) tudjunk két különböző frekvenciájú hang lejátszását indítani. Az egyik néhány másodperc múlva magától is kapcsoljon le. 'Stop' nyomógommbal azonnal leállítható legyen a lejátszás. Amíg nem állítottuk le a lejátszást egy LED is villogjon folyamatosan !

```
const byte stop = 2;           // stop nyomógomb a 2-es lábon
const byte k3 = 3;             // K3 nyomógomb a 3-as lábon
const byte k4 = 4;             // K4 nyomógomb a 4-es lábon
const byte hangsz = 7;         // hangszóró a 7-es lábon
const byte led = 8;            // LED a 8-as lábon

byte play = 0;                 // állapot változó, 1-es ha lejátszás, 0 ha stop

void setup()
{
    pinMode(hangsz, OUTPUT);
    pinMode(led, OUTPUT);
    pinMode(stop, INPUT);
    pinMode(k3, INPUT);
    pinMode(k4, INPUT);
    digitalWrite(led, LOW);     // LED lekapcsolása
    noTone(hangsz);             // hang lekapcsolása
}
```

15.4. Hangok Arduinóval

1. mintafeladat folytatása

```
void loop()
{
    if(digitalRead(k3)==1)    // ha lenyomtuk a 'K3' gombot
    {
        play=1;                // állapot változó állítása (play)
        tone(hangsz,300,6000); // 300 Hz-es jel 6s-ig felkapcsolása
    }
    if(digitalRead(k4)==1)    // ha lenyomtuk a 'K4' gombot
    {
        play=1;                // állapot változó állítása (play)
        tone(hangsz,600);      // 600 Hz-es jel felkapcsolása
    }
    if(digitalRead(stop)==1)  // ha lenyomtuk a 'stop' gombot
    {
        play=0;                // állapot változó állítása (stop)
        noTone(hangsz);        // hang lekapcsolása
    }
    if(play==1)               // ha lejátszás van még villogjon a LED is
    {
        digitalWrite(led, HIGH);
        delay(300);
        digitalWrite(led, LOW);
        delay(300);
    }
}
```

15.5. Hangok Arduinóval

2. mintafeladat

K3 nyomógommbal tömbben tárolt hangok lejátszását tudjuk indítani.
Stop nyomógommbal leállítható legyen a lejátszás.

```
const byte stop = 2;      // stop nyomógomb a 2-es lábon
const byte k3 = 3;        // K3 nyomógomb a 3-as lábon
const byte hangsz = 7;    // hangszóró a 7-es lábon

// tömb hangok tárolására
const byte hang= 12;      // hány darab hangunk van
int hangok[hang]= {300,600,300,600,900,900,1200,400,600,400,300,300};

byte play = 0;            // állapot változó,1-es ha lejátszás,0 ha stop
byte i=0;                 // ciklus változó

void setup()
{
    pinMode(hangsz, OUTPUT);
    pinMode(stop, INPUT);
    pinMode(k3, INPUT);
    noTone(hangsz);        // hang lekapcsolása
}
```

15.6. Hangok Arduinóval

2. mintafeladat folytatása

```
void loop()
{
    if(digitalRead(k3)==1)           // ha lenyomtuk a 'K3' gombot
        play=1;                     // állapot változó állítása (play)

    if(digitalRead(stop)==1)         // ha lenyomtuk a 'stop' gombot
    {
        play=0;                     // állapot változó állítása (stop)
        noTone(hangsz);             // hang lekapcsol
        i=0;
    }

    if(play==1)                      // lejátszás
    {
        tone(hangsz,hangok[i]);     // következő hang kiküldése
        i++;                        // lépünk egyet a tömb indexben
        if(i>hang-1) i=0;           // végig értünk,vissza a tömb elejére
    }

    delay(500);
}
```

15.7. Feladatok

1. feladat

- Induláskor adjon két rövid hangjelzést (különböző frekvenciákon)
- Majd a K3 nyomógomb minden lenyomására egy rövid dallam (15 hang tömbben) lejátszása történjen meg egyszer.

2. feladat

- Induláskor adjon két rövid hangjelzést (különböző frekvenciákon)
- Majd a K3 nyomógomb minden lenyomására egy rövid dallam (10 hang tömbben) lejátszása történjen meg háromszor !

3. feladat

- K3 nyomógommbal tömbben tárolt hangok lejátszását tudjuk indítani (dallam1 – 15db hang).
- K4 nyomógommbal egy másik tömbben tárolt hangok lejátszását tudjuk indítani (dallam2 – 20db hang).
- Mindkét dallam lejátszása folyamatosan történjen (ha vége kezdjük előlről), addig míg a 'STOP' nyomógombot meg nem nyomjuk !
- STOP nyomógommbal a lejátszás legyen leállítható.

16.1. Arduino függvények

„millis” függvény

A bekapcsolás óta eltelt időt adja vissza milliszekundumban. Előjel nélküli 32 bites egész a visszatérési érték, kb. 50 nap után túlcsordul, és kezdődik előlről !!

unsigned long b1; // unsigned --> előjel nélküli, csak pozitív szám !!

b1 = millis();

„micros” függvény

A bekapcsolás óta eltelt időt adja vissza mikroszekundumban. Előjel nélküli 32 bites egész a visszatérési érték, kb. 70 perc után túlcsordul, és kezdődik előlről !!
4-8 mikroszekundum pontosságú !

unsigned long b1;

b1 = micros();

„delayMicroseconds” függvény

Késleltetés, mikroszekundumban megadva. Ennyi ideig várunk adott helyen a programban, és csak ha letelt az idő akkor hajtódik végre a következő utasítás.

delayMicroseconds(6000); // késleltetünk 6 milliszekundumot

16.2. Arduino függvények

matematikai függvények

```
abs(x);           // szám abszolút értéke
sqrt(x);          // szám négyzetgyöke
sin(x);           // szám szinusza, paraméter radiánban !!
cos(x);           // szám koszinusza, paraméter radiánban !!
tan(x);           // szám tangense, paraméter radiánban !!
map(x,fromLow,fromHigh,toLow,toHigh); // leképez egy számot
                                           // egy számtartományból, egy másikba
pow(alap,kitevő); // hatványozás
```

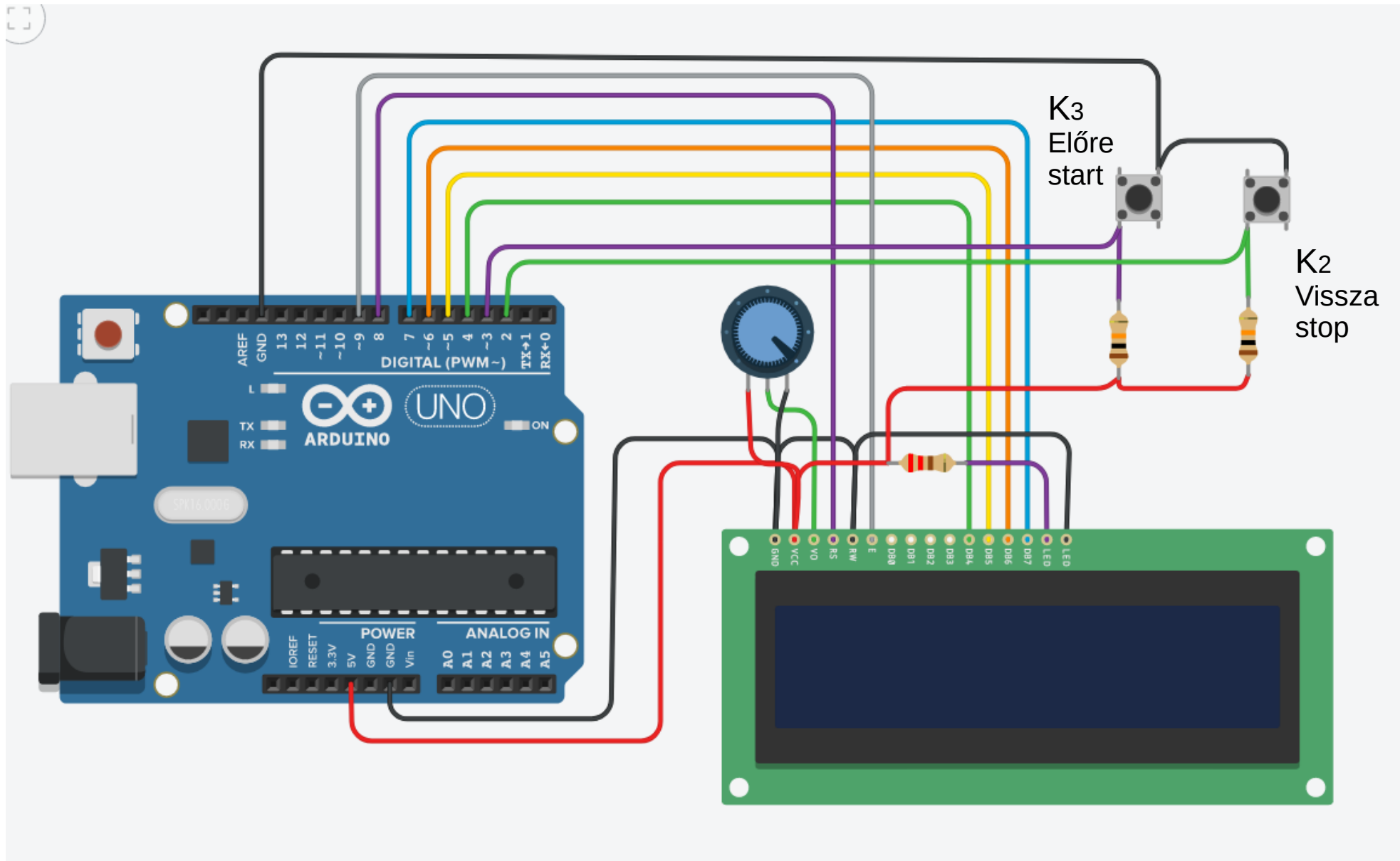
„random” függvény

Véletlenszerűen generál egész számot. (pseudo-random number generator)

```
random(max); // véletlenszám, 0 – max tartományból (max már nincs benne !!)
random(min,max); // véletlenszám, min – max tartományból (max már nincs benne !!)
randomSeed(seed); // inicializálja a random generátort ! célszerű először
                  //ezt meghívni !

pl. randomSeed(analogRead(A0)); // ha A0 nincs használva
    random(1,91); // véletlenszámot ad (lehetséges értékek: 1,2,3,4, ...89, 90)
```

16.3. Feladatok időikkel, számokkal, a hardver



16.4. Feladatok időikkel, számokkal

Beállítások a mintafeladatokhoz

A szögfüggvényekkel való számolásokhoz használhatjuk a következő, az Arduino függvénykönyvtárban definiált konstansokat:

```
PI           //  $\pi$  értéke  
HALF_PI      //  $\pi/2$  értéke  
TWO_PI       //  $2*\pi$  értéke  
DEG_TO_RAD   // fok-radián átszámítás szorzószáma  
RAD_TO_DEG   // radián-fok átszámítás szorzószáma
```

```
#include <LiquidCrystal.h>  
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);  
const byte k2=2;           // k2 (stop) nyomógomb a 2-es lábon  
const byte k3=3;           // k3 (start) nyomógomb
```

```
int i=0;  
float x=0;  
unsigned long ido=0;  
// egyéb szükséges változók definiálása
```

```
void setup()  
{  
    lcd.begin(16, 2);        // 16x2 karakter/soros LCD kijelző beállítása  
    pinMode(k2, INPUT);      // 'k2' bemenet lesz (nyomógomb)  
    pinMode(k3, INPUT);      // 'k3' bemenet lesz (nyomógomb)  
    delay(500);              // késleltetünk  
    lcd.setCursor(5, 0);     // kurzor pozíció 1. sor 6. karakter  
    lcd.print("Hello !");    // szöveg kiíratása (1. sorban)  
    delay(2000);  
}
```

16.5. Feladatok időikkel, számokkal

1. mintafeladat

Írjuk ki az első sorban a szögek (0 foktól 360 fokig, 15 fokonként) szinuszát.
A második sorban pedig az eddig eltelt időt, milliszekundumban

```
void loop()
{
    lcd.clear();           // képernyő törlése
    lcd.setCursor(0, 0);   // kurzor pozíció 1. sorba
    lcd.print(i);
    lcd.print(" sin: ");
    x=sin(DEG_TO_RAD*i);   // szög szinuszának kiszámítása
    lcd.print(x);
    lcd.setCursor(0, 1);   // kurzor pozíció 2. sorba
    ido=millis();          // eltelt idő lekérdezése
    lcd.print(ido);
    i=i+15;                // szög növelése
    if(i>360) i=0;
    delay(2000);
}
```

16.6. Feladatok időikkel, számokkal

2. mintafeladat

Írjuk ki az első sorban a számok (0-tól 20-ig) köbét.

A második sorban pedig az eddig eltelt időt, mikroszekundumban

```
void loop()
{
    lcd.clear();           // képernyő törlése
    lcd.setCursor(0, 0);   // kurzor pozíció 1. sorba
    lcd.print(i);
    lcd.print(" a 3.-on: ");
    x=pow(i,3);            // köb kiszámítása
    lcd.print(x);
    lcd.setCursor(0, 1);   // kurzor pozíció 2. sorba
    ido=micros();          // eltelt idő lekérdezése
    lcd.print(ido);
    i++;                   // szám növelése
    if(i>20) i=0;
    delay(2000);
}
```

16.7. Feladatok időikkel, számokkal

3. mintafeladat

Írjunk ki az első sorban véletlen 5-ös lottó számokat

A második sorban pedig az eddig eltelt időt, mikroszekundumban

```
void setup()
{
    ...
    ...
    randomSeed(analogRead(A0));
}

void loop()
{
    lcd.clear();           // képernyő törlése
    lcd.setCursor(0, 0);   // kurzor pozíció 1. sorba
    for(byte j=0;j<5;j++)  // 5 szám „húzása”
    {
        i=random(1,91);    // véletlen szám 1-90
        lcd.print(i);
        lcd.print(" ");
    }
    lcd.setCursor(0, 1);   // kurzor pozíció 2. sorba
    ido=micros();          // eltelt idő lekérdezése
    lcd.print(ido);
    delay(5000);
}
```

Mi a program hibája ??? Póbáljuk meg módosítani, hogy helyesen működjön !
valószínűleg tömböt kell használnunk :(

16.8. Feladatok időikkel, számokkal

4. mintafeladat

Szimuláljunk dobókockát !

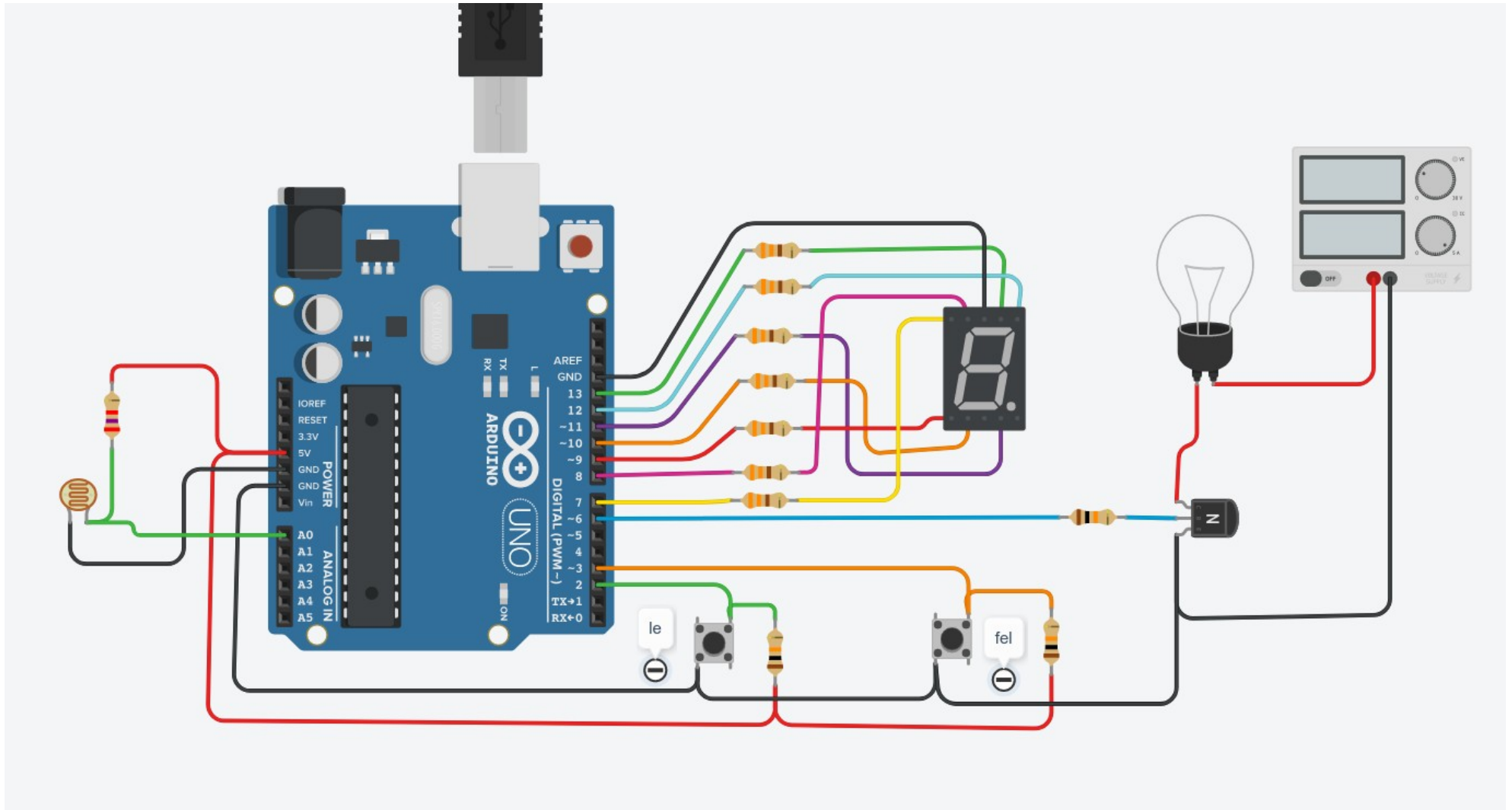
A K3 nyomógomb lenyomására, írjunk ki véletlen számot 1 és 6 között !

```
void setup()
{
    ...
    ...
    randomSeed(analogRead(A0));
}

void loop()
{
    lcd.clear();           // képernyő törlése
    lcd.setCursor(0, 0);   // kurzor pozíció 1. sorba
    lcd.print(" K3 dob");
    while(digitalRead(k3)==1); // várunk K3 lenyomására
    lcd.setCursor(0, 1);   // kurzor pozíció 2. sorba
    i=random(1,7);
    lcd.print("kocka: ");
    delay(1500);
    lcd.print(i);

    delay(3000);
}
```

17.1. Mindenféle feladatok 1., a hardver



17.2. Mindenféle feladatok 1.

1. feladat

A fotoellenállást érő fény erősségétől függően más-más értéket mutat a hétszegmenses kijelző →

ha az analóg érték:

- nagyobb mint 650 → '0' (sötét van)
- 400 és 650 között van → '1'
- 200 és 400 között van → '2'
- 200-nál kisebb → '3'

2. feladat

- Induláskor a 0-1-0-1-0 értékek kijelzése egymás után (1s-ig mindegyik)
- a 'FEL' nyomógomb lenyomása után → számlálás előre 1s-onként → 1-2-3-4 és itt megáll !
- a 'LE' nyomógomb lenyomása után → számlálás hátra 1s-onként → 3-2-1-0 és itt megáll !

3. feladat

- folyamatos számlálás előre kettesével (0-2-4-6-8-0-2-...) vagy hátra kettesével (8-6-4-2-0-8-6-...)
- a számlálás irányát a két nyomógombbal tudjuk állítani

17.2. Mindenféle feladatok 1.

4. feladat

A fotoellenállást érő fény erősségétől függően más-más értéket mutasson a hétszegmenses kijelző, és az izzó fényereje is változzon →

ha az analóg érték:

- nagyobb mint 600 → '0' (sötét van), ilyenkor az izzó is világítson teljes fényerővel !
- 300 és 600 között van → '1' és az izzó közepes erősséggel világítson !
- 300-nál kisebb → '2' az izzó csak gyengén világítson !

5. feladat

- Induláskor a 0-2-0-2-0 értékek kijelzése egymás után (1s-ig mindegyik)
- a 'FEL' nyomógomb lenyomása után → számlálás előre 1s-onként → 1-2-3-4 és itt megáll !
Közben az izzó teljes fényerővel erősséggel világítson !
- a 'LE' nyomógomb lenyomása után → számlálás hátra 1s-onként → 3-2-1-0 és itt megáll !
Közben az izzó közepes fényerővel világítson !

6. feladat

- folyamatos számlálás előre kettesével (0-2-4-6-8-0-2-...) vagy hátra kettesével (8-6-4-2-0-8-6-..)
- a számlálás irányát a két nyomógombbal tudjuk állítani
- Az izzó fényereje is legyen arányos a számláló értékével → PWM kitöltési tényező 0, 20%, 40%, 60%, 80%

17.2. Mindenféle feladatok 1.

7. feladat

Induláskor az 1-2-3-4-5-6 értékek kijelzése egymás után (0,5s-ig mindegyik)

A 'FEL' nyomógomb lenyomása után →

- hétszegmenses kijelző kapcsoljon le
- az izzó fényereje folyamatosan növekedjen nullától maximumig, kb. 2 másodperc alatt !
- majd sorsoljunk ki egy véletlen számot 1-6 között (1,2,3,4,5 vagy 6) és jelenítsük meg a hétszegmenses kijelzőn !

8. feladat

- Induláskor a 0 érték kijelzése a hétszegmenses kijelzőn !
- ha a 'FEL' gombot nyomjuk le → az izzó fényereje folyamatosan növekedjen nullától maximumig, kb. 3 másodperc alatt 9 lépésben, és a fokozatot/lépésszámot jelezzük ki a hétszegmenses kijelzőn (1-2-3-....-9) !
- ha a 'LE' gombot nyomjuk le → az izzó fényereje folyamatosan csökkenjen maximumtól nulláig, kb. 3 másodperc alatt 9 lépésben, és ezt jelezzük ki a hétszegmenses kijelzőn (8-7-6-...-0) !

9. feladat

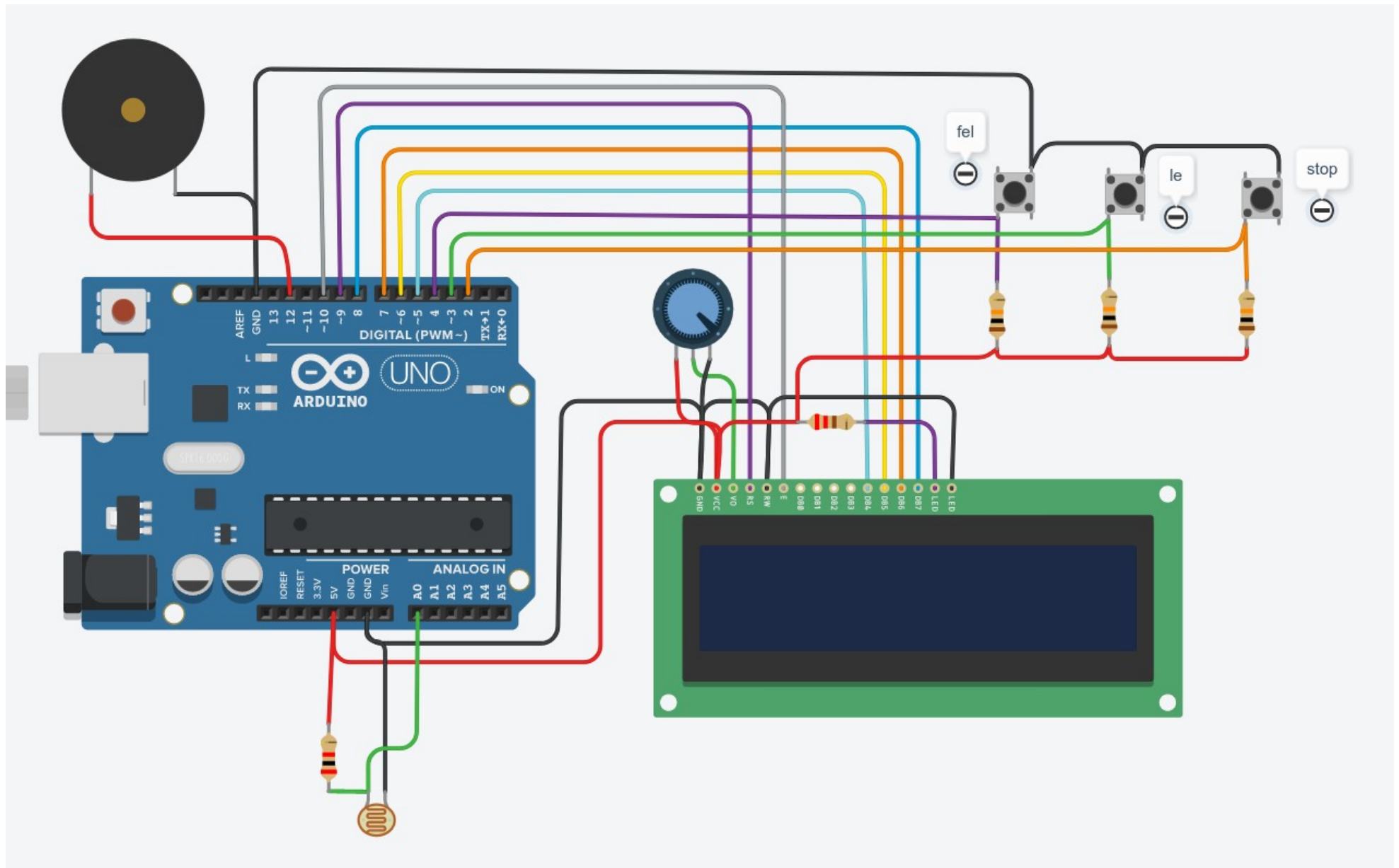
Induláskor az 1-2-3-4-5-6 értékek kijelzése egymás után (0,5s-ig mindegyik), majd kapcsoljuk le a hétszegmenses kijelzőt !

A fotoellenállást érő fény erősségétől függően a következő történjen:

ha az analóg érték:

- kisebb mint 500 (világos van) → 4 másodpercenként sorsoljunk ki egy véletlen számot 1-6 között (1,2,3,4,5 vagy 6) és jelenítsük meg a hétszegmenses kijelzőn !
- nagyobb mint 500 (sötét van) → álljon le a sorsolás, és kapcsoljuk le a hétszegmenses kijelzőt !

17.3. Mindenféle feladatok 2., a hardver



17.4. Mindenféle feladatok 2.

1. feladat

- az LCD kijelző első sorában jelenjen meg egy üzenet
- A fotoellenállást érő fény erősségétől függő beolvasott analóg értéket írjuk ki az LCD kijelző második sorában, kb. 0,5 másodpercenként frissítsük !
- ha az analóg érték nagyobb mint 700 (sötét van) → adjunk hangjelzést !

2. feladat

- Induláskor az LCD kijelző 1. sorában írjunk ki egy üzenetet (pl. 'Helo'), a 2. sorban folyamatos számlálás 0 és 20 között előre 1 másodpercenként (0-1-2-3-4-...-17-18-19-20-0-1-2-...), a 19 és 20 értékeknél rövid hangjelzés legyen !
- Majd ezután →
 - a STOP nyomógombbal tudjuk a számlálást leállítani
 - a FEL nyomógombbal tudjuk a számlálást elindítani

3. feladat

- Induláskor az LCD kijelző 1. sorában írjunk ki egy üzenetet (pl. 'Helo'), és adjunk 4-5 hangból álló hangjelzést !
- a 'FEL' nyomógomb lenyomása után → a 2. sorban számlálás előre 1s-onként → 0-1-2-3-4-5-6-.....8-9-0-1-.....
- a 'LE' nyomógomb lenyomása után → a 2. sorban számlálás hátra 1s-onként → 9-8-7-.....-1-0-9-8-.....

17.4. Mindenféle feladatok 2.

4. feladat

- Induláskor az LCD kijelző első sorában jelenjen meg egy üzenet („Hello”) !
- A 'FEL' nyomógomb lenyomása után →
- adjunk 2 másodperces hangjelzést, majd
- sorsoljunk ki egy véletlen számot 1-6 között (1,2,3,4,5 vagy 6) és jelenítsük meg az LCD kijelző második sorában !

5. feladat

- Induláskor az LCD kijelző 1. sorában írjunk ki egy üzenetet (pl. 'Hello'),
- a LE nyomógommbal tömbben tárolt hangok lejátszását tudjuk indítani (10db hang), jelenítsük meg az LCD kijelző 2. sorában az éppen kiadott hang frekvenciáját !
A dallam lejátszása folyamatosan történjen (ha vége kezdjük előlről), addig míg a STOP nyomógombot meg nem nyomjuk !
- STOP nyomógommbal a lejátszás legyen leállítható.

6. feladat

- Induláskor az LCD kijelző első sorában jelenjen meg egy üzenet („Hatos lotto”) !
- A 'FEL' nyomógomb lenyomása után →
- adjunk 5 másodpercenként adjunk rövid hangjelzést, majd sorsoljunk ki egy véletlen számot 1 és 45 között és jelenítsük meg az LCD kijelző 2. sorában !
Ez folyamatosan történjen, addig míg a STOP nyomógombot meg nem nyomjuk !
- STOP nyomógommbal a sorsolás legyen leállítható.

17.4. Mindenféle feladatok 2.

7. feladat

Induláskor az LCD kijelző első sorában jelenjen meg egy üzenet („Hetes lotto”) !

A fotoellenállást érő fény erősségétől függően a következő történjen,

ha az analóg érték:

- kisebb mint 400 (világos van) →
adjunk rövid hangjelzést, majd sorsoljunk ki egy véletlen számot
1 és 35 között és jelenítsük meg az LCD kijelző 2. sorában !
- nagyobb mint 400 (sötét van) →
álljon le a sorsolás, és töröljük az LCD kijelző 2. sorát !

8. feladat

Induláskor az LCD kijelző 1. sorában jelenjen meg egy üzenet („Negyzetgyok”) !

A 'FEL' nyomógomb lenyomása után →

- folyamatos számlálás 2 és 20 között előre 4 másodpercenként,
 - jelenítsük meg a számot az LCD kijelző 1. sorában, és a négyzetgyökét a 2. sorában !
 - ez folyamatosan történjen, addig míg a STOP nyomógombot meg nem nyomjuk !
- STOP nyomógommbal a számlálás legyen leállítható.