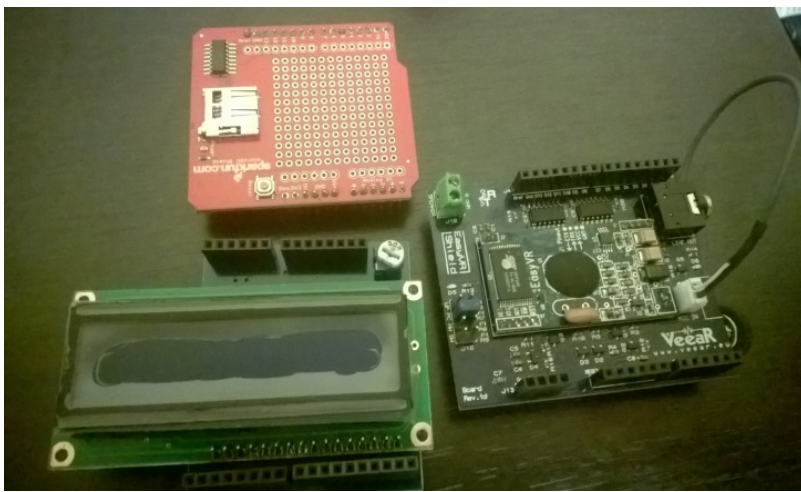


Laborator 3 – Utilizarea afișorului LCD și a sistemului de întreruperi

1. Utilizarea shield-ului LCD

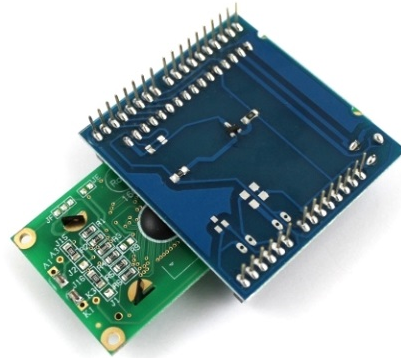
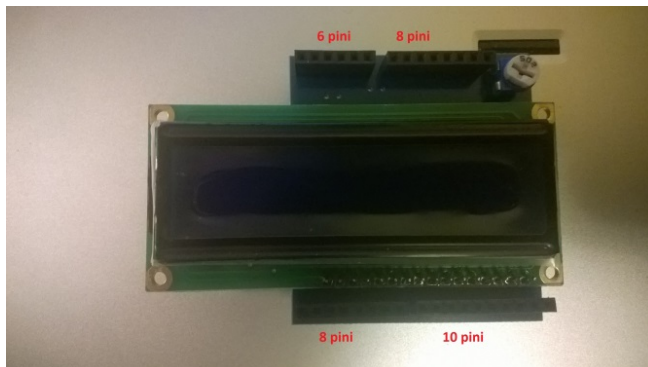
Shield-urile sunt PCB-uri (plăci de circuit imprimat – Printed Circuit Board) care pot fi plasate deasupra plăcilor Arduino, extinzându-le astfel capacitățile. Există o varietate foarte mare de astfel de shield-uri, precum: XBee shield, SD-Shield, H-Bridge shield, etc. Pe PCB-urile Shield-urilor, anumite fire sunt trase către barețele de pini care urmează să fie introduse în Arduino. Așadar, trebuie să fim atenți la pinii folosiți de componenta electronică, pentru a o putea mapa corect în program și pentru a nu avea conflicte în utilizarea acestor pini. Pinii folosiți sunt în general specificați în foaia tehnică a produsului. Figura de mai jos conține câteva exemple de shield-uri (shield LCD, un shield SD card și un shield de recunoaștere vocală EasyVR).



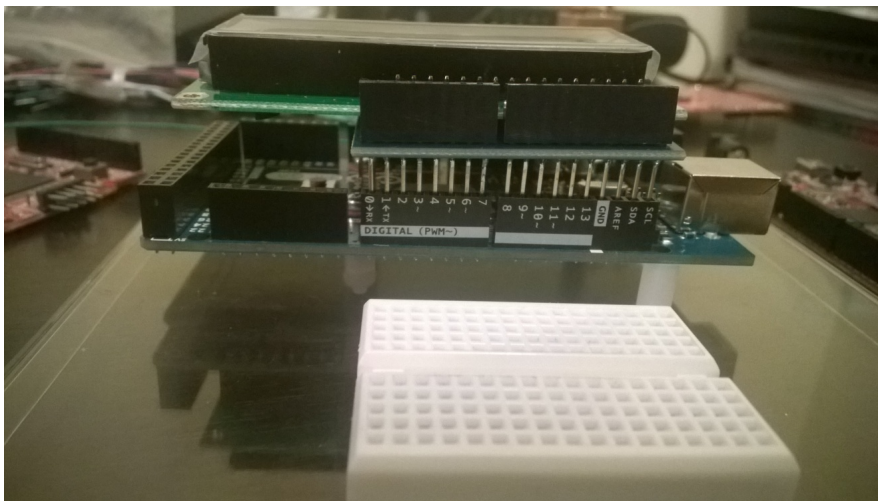
Dat fiind faptul că plăcile Arduino au avut un succes foarte mare pe piață mulți dintre competitorii platformei de dezvoltare au folosit același factor de formă când au proiectat plăcile lor. Acest lucru este avantajos întrucât și alte plăci, cu alte procesoare, pot folosi anumite shield-uri Arduino. În figura de mai jos puteți observa alte plăci de dezvoltare care, au un design similar cu al plăcilor Arduino (placa din stânga este un Fez Domino cu un procesor ARM, care se programează în .Net, iar placa din dreapta este un Chip Kit care conține un procesor de tip PIC și este programabilă în c++).



În acest laborator vom folosi shield-ul LCD, care conține un afișor cu cristale lichide și un potențiomtru pentru reglarea intensității luminii.

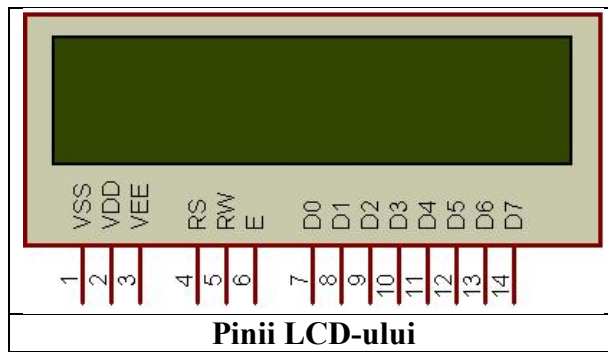


Shield-ul este amplasat deasupra plăcii Arduino Mega astfel încât baretele mai lungi (8, respectiv 10 pini) să fie în dreptul pinilor digitali, iar cele scurte în dreptul pinilor analogici, ca în figura de mai jos. **NU scoateți shield-ul de pe placa Arduino (există riscul să îndoiți pinii acestuia) !!!**



LCD-ul utilizează pinii digitali de la 2 la 7, astfel: pinul digital 7 - RS; pinul digital 6 - EN; pinul digital 5 - DB4; pinul digital 4 - DB5; pinul digital 3 - DB6; pinul digital 2 - DB7 (explicația semnalelor pe pagina următoare).

Shield-ul este bazat pe controllerul clasic care se folosește la LCD-uri, Hitachi HD44780. LCD-urile care folosesc cel mult 80 de caractere distincte și cel mult 4 linii de afișare au nevoie de un singur controller. LCD-urile care au mai mult de 80 de caractere au nevoie de două controlere. Imaginea cu pinout-ul LCD-ului și semnificația pinilor o găsiți în figura și tabelul de pe pagina următoare.



Pin Nr	Nume	Descriere
1	VSS	Power supply (GND)
2	VCC	Power supply (+5V)
3	VEE	Contrast adjust
4	RS	Register Select 0 = Instruction input 1 = Data input
5	R/W	0 = Write to LCD module 1 = Read from LCD module
6	EN	Enable signal
7	D0	Data bus line 0 (LSB)
8	D1	Data bus line 1
9	D2	Data bus line 2
10	D3	Data bus line 3
11	D4	Data bus line 4
12	D5	Data bus line 5
13	D6	Data bus line 6
14	D7	Data bus line 7(MSB)
Semnificația pinilor LCD		

Controllerul HD44780 conține două registre pe 8 biți: registrul de date și registrul de instrucțiuni. Registrul de instrucțiuni e un registru prin care LCD primește comenzi (shift, clear etc). Registrul de date este folosit pentru a acumula datele care vor fi afișate pe display. Când semnalul Enable al LCD-ului este activat, datele de pe pinii de date sunt puse în registrul de date, apoi mutate în DDRAM (memoria de afișaj, Display Data RAM) și afișate pe LCD. Registrul de date nu este folosit doar pentru trimiterea datelor către DDRAM ci și către CGRAM, memoria care stochează caracterele create de către utilizator (Character Generator RAM).





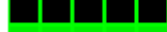


Display Data Ram (DDRAM) stochează datele de afișare, reprezentate ca și caractere de 8 biți. Capacitatea extinsă a memoriei este de 80 X 8 biți, sau 80 de caractere. Memoria rămasă liberă poate fi folosită ca un RAM generic. Pe LCD-ul nostru se afișează doar 2x16 caractere, deoarece aceasta este dimensiunea afișorului, dar controllerul poate stoca 80.

Forma grafică efectivă a unui caracter afișat pe LCD este dată de conținutul memoriei CGROM (Character Generator Read Only Memory). Această memorie conține matricele de puncte pentru fiecare caracter (5x8 puncte sau 5x10 puncte – depinde de dispozitiv).

Upper 4 Bits	Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx	0000	CG ROM (1)		0	1	2	3	4	5	6	7	8	9	A	B	C	D
xxxx	0001	(2)	!	1	A	Q	a	q				。	ア	チ	△	ä	q
xxxx	0010	(3)	"	2	B	R	b	r				「	イ	ツ	×	β	θ
xxxx	0011	(4)	#	3	C	S	c	s				」	ウ	テ	ε	∞	
xxxx	0100	(5)	\$	4	D	T	d	t				、	エ	ト	†	μ	Ω
xxxx	0101	(6)	%	5	E	U	e	u				・	オ	ナ	1	∞	Ü
xxxx	0110	(7)	&	6	F	V	f	v				ヲ	カ	ニ	ヨ	ρ	Σ
xxxx	0111	(8)	'	7	G	W	g	w				ア	キ	ヌ	ラ	g	π
xxxx	1000	(1)	<	8	H	X	h	x				イ	ク	ネ	リ	∫	×
xxxx	1001	(2)	>	9	I	Y	i	y				ウ	ケ	ル	リ	∫	∫
xxxx	1010	(3)	*	:	J	Z	j	z				エ	コ	ン	レ	j	≠
xxxx	1011	(4)	+	;	K	[k	[オ	サ	ヒ	ロ	*	万
xxxx	1100	(5)	,	<	L	¥	l	l				カ	シ	フ	ワ	φ	円
xxxx	1101	(6)	-	=	M]	m]				ユ	ズ	ン	ン	も	÷
xxxx	1110	(7)	.	>	N	^	n	^				ヨ	セ	ホ	ッ	ん	
xxxx	1111	(8)	/	?	O	_	o	_				ツ	ソ	マ	°	ö	■

Codul caracterelor pentru caractere formate din 5 x 8 puncte

Aspectul caracterelor de la 0x00 la 0x07 poate fi definit de utilizator. Se va specifica pentru fiecare caracter o matrice de 8 octeți, cate unul pentru fiecare rând. Cei mai putini semnificativi 5 biți din fiecare rând vor specifica care pixeli vor fi aprinși și care nu (vezi exemplul de mai jos).

Custom Pattern	Decimal	Hex
	Row 1: 4	0x04
	Row 2: 14	0x0E
	Row 3: 14	0x0E
	Row 4: 14	0x0E
	Row 5: 31	0x1F
	Row 6: 0	0x00
	Row 7: 4	0x04

Un exemplu de model creat de utilizator

Afișoarele LCD pot comunica cu microcontrolerul în două moduri: pe 8 biți și pe 4 biți. De obicei se preferă conectarea pe 4 biți, pentru că sunt mai putini biți de interfațat, și în consecință rămân mai mulți pini pentru alte aplicații. *Shield-ul LCD pe care îl veți utiliza este*

gata configurat pentru a folosi comunicarea pe 4 biți. Acest mod folosește doar 7 pini de pe placa Arduino; modul pe 8 biți ar fi folosit 11;

Exemplu: afișarea șirurilor de caractere și a valorilor numerice pe LCD

În primul exemplu vom afișa un șir de caractere pe prima linie a LCD-ului, și numărul de secunde care au trecut de la începerea programului.

```
//include biblioteca pentru lucrul cu LCD
#include <LiquidCrystal.h>
/* LCD RS pinul digital 7
 * LCD Enable pinul 6
 * LCD D4 pinul 5
 * LCD D5 pinul 4
 * LCD D6 pinul 3
 * LCD D7 pinul 2
Al șaptelea pin este pinul de la potentiometrul de reglare a iluminării */

//inițializează lcd-ul la valorile stabilite ale pinilor
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
unsigned long time;

void setup()
{
  //setează numărul de rânduri și coloane ale LCD-ului
  lcd.begin(16, 2);
}

void loop()
{
  //luăm numărul de secunde trecute de la reset (sau pornirea programului)
  time = millis() / 1000;
  //setăm cursorul la coloana 0 rândul 1
  lcd.setCursor(0, 0);
  //scriem un text
  lcd.print("Hello Children");
  //setăm cursorul la mijlocul liniei a doua
  lcd.setCursor(7, 1);
  //scriem timpul
  lcd.print(time);
}
```

În figura următoare puteți vedea rezultatul.



Rezultatul afisarii pe LCD

Exemplu: generarea de caractere utilizator

Acest exemplu ilustrează crearea caracterelor de către utilizator, și folosirea lor.

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

```
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
```

```
// Masca pentru primul caracter, fiecare linie de biți reprezintă o linie a caracterului
```

```
byte happy[8] = {
```

```
B00000,
```

```
B11011,
```

```
B11011,
```

```
B00000,
```

```
B00000,
```

```
B10001,
```

```
B01110,
```

```
};
```

```
// Masca pentru al doilea caracter
```

```
byte sad[8] = {
```

```
B00000,
```

```
B11011,
```

```
B11011,
```

```
B00000,
```

```
B00000,
```

```
B01110,
```

```
B10001,
```

```
};
```

```
void setup() {
```



```

lcd.begin(16, 2);
// cele două caractere sunt stocate în CGROM, zona utilizator, pozițiile 0 și 1
lcd.createChar(0, happy);
lcd.createChar(1, sad);

// Afișare prima linie, un text standard urmat de primul caracter utilizator
lcd.setCursor(0, 0);
lcd.print("Happy ");
lcd.write(byte(0)); // Observați diferența dintre print și write
/* când referiți caracterul „0” trebuie să faceți un cast la byte. Altfel compilatorul va
semnala o eroare. Excepție este cazul în care referiți o variabilă:
byte zero = 0;
lcd.write(zero);
*/

// Afișare pe a doua linie
lcd.setCursor(0, 1);
lcd.print("Sad ");
lcd.write(1); // când referiți caractere diferite de „0” nu mai este necesar cast-ul;
}

// Funcția loop rămâne nefolosită, puteți să o folosiți pentru a extinde funcționalitatea
void loop()
{
}

```

2. Folosirea sistemului de întreruperi

Întreruperile sunt evenimente care necesită atenția imediată a microcontrolerului. În momentul în care se întâmplă un eveniment care are ca efect declanșarea unei întreruperi, microcontrolerul oprește programul pe care îl execută și începe să se ocupe de întrerupere, executând o ISR (Interrupt Service Routine), o procedură atașată întreruperii respective. Pentru ca microcontrolerul să răspundă la cereri de întreruperi, trebuie activat bitul care controlează sistemul global de întreruperi (Global Interrupt Enable), și bitul corespunzător întreruperii. Următoarele lucruri sunt esențiale pentru ca sistemul de întreruperi să poată fi folosit în mod corect:

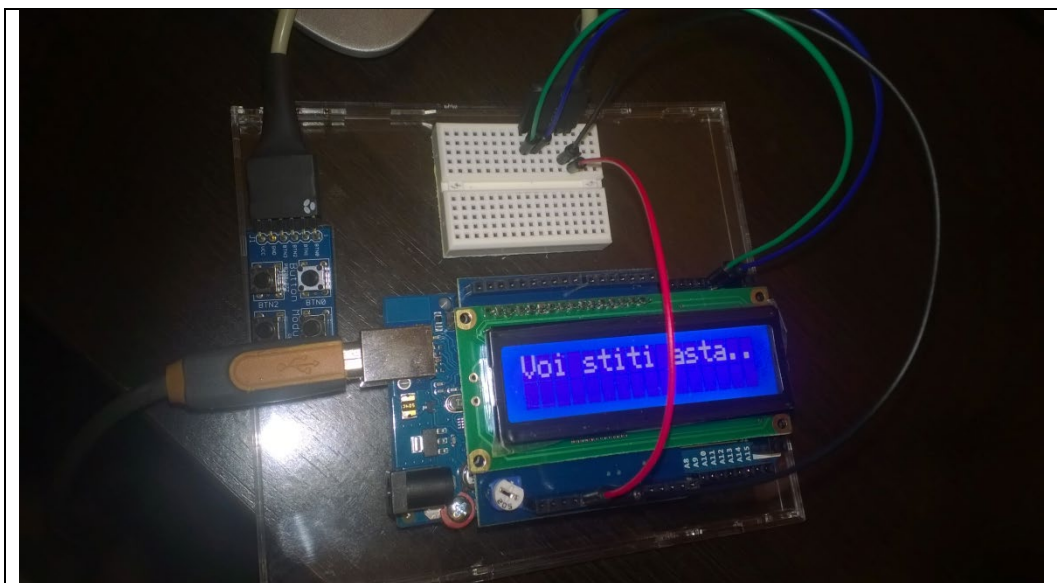
- Întreruperea trebuie să fie activată prin bitul ei corespunzător.
- Bitul Global corespunzător întreruperilor (bitul I) din SREG trebuie să fie activat.
- Stiva trebuie inițializată. Acest lucru se realizează în mod automat dacă se lucrează în mediul Arduino.
- Fiecare rutină se finalizează cu instrucțiunea RETI. În acel moment microcontrolerul știe să se întoarcă la execuția programului întrerupt. La Arduino, instrucțiunea este plasată în mod automat de compilator.

Înteruperile sunt de două feluri: interne și externe. Înteruperile interne sunt asociate cu perifericele microcontrolerului (Timer/Counter, Analog comparator etc.) Înteruperile externe sunt declanșate de pini externi (de exemplu, la acești pini se pot atașa butoane).

Fiecare AVR are o listă de înteruperi, care include tipul de eveniment care va declanșa înteruperia. În momentul în care înteruperile sunt activate și unul din aceste evenimente se întâmplă, procesorul va realiza un salt în memoria program la o anumită locație/adresa (referita de vectorul înteruperii, adresa pe care o va găsi în lista/tabela de înteruperi). Scriind o procedură ISR și apoi făcând un o legătură spre această procedură la adresa înteruperii corespunzătoare, putem să determinăm sistemul să realizeze o acțiune specifică în momentul în care un anumit eveniment se produce.

În primul exemplu vom folosi o înterupere declanșată de butoane (externa), și vom afișa pe LCD un mesaj corespunzător butoanelor. În momentul în care butonul este apăsat se declanșează cererea de înterupere, și programul afișează un mesaj pe LCD. Pentru acest exemplu aveți nevoie de placa Arduino Mega, un afișor LCD, un bloc de butoane PModBtn și de bread board. ATmega 2560 are 6 pini de înterupere. Pentru a vedea lista de pini și înteruperile corespunzătoare accesați <http://arduino.cc/en/Hacking/PinMapping2560>.

Vom conecta butoanele la pinii corespunzători înteruperilor INT0 și INT1, adică pinii digitali 20 și 21. În figura următoare se observă o imagine a conexiunilor necesare, iar mai jos codul corespunzător programului.



Conexiunile necesare primului exemplu

```
//includem headerul responsabil pentru operatii cu intreruperi pentru avr
#include "avr/interrupt.h"
//include biblioteca de manipulat LCD
#include <LiquidCrystal.h>
```

```
//initializeaza lcd-ul la valorile stabilite ale pinilor
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
volatile int buttonVariable; //variabila publica ce poate fi modificata de o intrerupere
```



```

void setup(void)
{
    buttonVariable = 0; //initializam variabila shared intre ISR si programul principal

    //seteaza numarul de randuri si coloane ale LCD-ului
    lcd.begin(16, 2);
    lcd.print("Incepe experimentul");
    delay(1000); //facem o scurta pauza pentru a vizualiza mesajul de pe ecran

    //setam pinul 21 ca si pin de intrare; pinul pe care se afla intreruperea INT0
    pinMode(21, INPUT);
    //setam pinul 20 ca si pin de intrare; pinul pe care se afla intreruperea INT1
    pinMode(20, INPUT);

    pinMode(13, OUTPUT); //setam pinul 13 ca si iesire
    digitalWrite(13, HIGH); //aprindem led-ul atasat la pinul 13

    delay(1000);

    EIMSK |= (1 << INT0); //activam punctual intreruperea INT0
    EIMSK |= (1 << INT1); //activam punctual intreruperea INT1

    EICRA |= (1 << ISC01); //activam intreruperea 0 pentru front descrescator.
    EICRA |= (1 << ISC11); //ca si mai sus, pentru intreruperea 1
    sei(); //activam intreruperile la nivel global

    digitalWrite(13, LOW); // stingem led-ul atasat la pinul 13

    lcd.clear(); //stergem ecranul LCD
}

void loop()
{
    //daca a fost executata o ISR trebuie sa stergem ecranul si sa afisam iar mesajul
    // principal
    if(buttonVariable == 1)
    {
        lcd.clear(); //se sterge ecranul LCD
        buttonVariable = 0; // variabila este re-initializata
    }

    delay(1000);
    lcd.setCursor(0,0); //setam cursorul
    lcd.print("Liniste..."); //afisam un mesaj
}

//Rutina pentru tratarea intreruperii atasata la INT0
ISR(INT0_vect)
{

```

```

digitalWrite(13, !digitalRead(13)); //schimba starea pin 13
lcd.setCursor(0,0); //pozitionam cursorul stanga sus
lcd.print("Intrerupem");//afisam mesaj
lcd.setCursor(0,1);
lcd.print("ptr stirea zilei");
buttonVariable = 1;
}

//Rutina pentru tratarea intreruperii atasata la INT0
ISR(INT1_vect)
{
digitalWrite(13, !digitalRead(13));
lcd.clear();
lcd.setCursor(0,0);
lcd.print("Stirea Doi");
buttonVariable = 1;
}

```

Este bine ca o procedură de tip ISR să poată fi executată cât mai repede, deoarece programul principal este oprit în aceste momente, așteptând terminarea procedurii.

Țineți minte că `delay()` și `millis()` nu funcționează „pe parcursul” unei rutine de întrerupere. Aceste funcții folosesc și ele sistemul de întreruperi, iar acesta este dezactivat atunci când suntem într-o procedură de tip ISR. În general, în timpul execuției unei ISR sistemul nu poate răspunde la alte cereri. Din acest motiv, este recomandat ca aceste proceduri să poată fi executate într-un timp foarte scurt.

Dacă se dorește modificarea unei variabile în interiorul unei proceduri ISR, și doriți ca valoarea modificată a acestei variabile să fie vizibilă în tot programul, declarați această variabilă ca **volatile**. Prin această declarație, compilatorul știe că variabila poate fi modificată în orice moment, renunțând la eventuale optimizări și plasând variabila în memoria RAM.

Arduino ne permite să utilizăm sistemul de întreruperi și fără a avea cunoștințe despre mecanismul specific microcontrolerului, punând la dispoziție anumite funcții generice. Prima funcție pe care o vom prezenta este **attachInterrupt()**. Funcția aceasta are rolul de a atașa o funcție ISR la o întrerupere externă, înlocuind orice funcție precedentă care a fost atașată anterior, și de a activa întreruperea. Sintaxa este: **attachInterrupt(interrupt, ISR, mode)**. Primul parametru al funcției `attachInterrupt` reprezintă numărul întreruperii.

Din păcate, parametru **interrupt** nu corespunde cu numărul întreruperii externe din microcontrollerul ATmega2560, și nici cu pinul digital al plăcii. Este un simplu număr de ordine. Tabelul de mai jos ne arată diferența dintre parametrul **interrupt**, întreruperea externă a ATmega2560, pinul de pe chip, și pinul digital de pe placă.

attachInterrupt	Name	Pin on chip	Pin on board
		(TQFP)	
0	INT4	6	D2
1	INT5	7	D3
2	INT0	43	D21
3	INT1	44	D20
4	INT2	45	D19
5	INT3	46	D18

Din acest motiv, se recomandă utilizarea funcției **digitalPinToInterrupt(pin)**, care va returna numărul întreruperii atașată pinului digital (dacă aceasta există).

Al doilea parametru al funcției **attachInterrupt** este procedura de tratare a întreruperii (ISR). Al treilea parametru este condiția de declanșare a întreruperii (front crescător: **RISING**, descrescător: **FALLING**, pe nivel 0: **LOW**, sau nivel 1: **HIGH**, sau la orice schimbare de nivel pe pin: **CHANGE**).

Funcția **noInterrupts()** dezactivează întreruperile. Acestea pot fi reactivate cu funcția **interrupts()**.

Funcția **detachInterrupt()** dezactivează întreruperea cu numărul specificat ca parametru. Sintaxa este **detachInterrupt(interrupt)**, **interrupt** având aceeași semnificație ca în cazul funcției **attachInterrupt()**.

Exemplul anterior a fost implementat folosind registre de configurare ale AVR, fiind astfel dependent de specificațiile microcontrolerului. În continuare vom prezenta un exemplu cu funcționalitate similară, realizat cu ajutorul funcțiilor specifice Arduino.

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

```
LiquidCrystal lcd(7,6,5,4,3,2);
volatile int buttonVariable;
```

```
void setup()
{
  buttonVariable = 0;
  lcd.begin(16,2);
  lcd.print("A inceput");
  lcd.setCursor(0,1);
  lcd.print("din nou");
  delay(1000);
```

```
// cei doi pini de intrerupere, 21 si 20, declarati ca intrare si rezistente pull up active
pinMode(20 ,INPUT);
pinMode(21 ,INPUT);
digitalWrite(20, HIGH);
digitalWrite(21, HIGH);
```

```
// atasm pinilor 21 si 21, corespunzatori INT1 si INT0, functii ISR
attachInterrupt(digitalPinToInterrupt(20), functieUnu, RISING);
```

```

attachInterrupt(digitalPinToInterrupt(21), functieDoi, CHANGE);
}

void loop()
{
//aici sunt taskuri care se executa in mod normal cand se ruleaza programul
  lcd.print("Programul principal");
  delay(1000);
}

//prima procedură ISR
void functieDoi()
{
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Functia Doi");
}

//a doua procedură ISR
void functieUnu()
{
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Functia Unu");
}

```

Lucru individual

1. Rulați exemplele prezentate
2. Folosind generatorul de caractere, creați o animație care indică trecerea timpului (un ceas cu braț care se rotește, o clepsidră, etc)
3. Folosind întreruperile, creați un cronometru. Unul din butoane va porni/opri cronometru, iar celălalt buton va avea rol de reset.
4. Folosiți animația de la punctul 2 împreună cu cronometrul creat. Animația va rula cât timp cronometrul merge, și se va opri când acesta e oprit.
5. Bonus: folosiți o placă pentru a genera semnale de diferite frecvențe și cu diferiți factori de umplere (timpul cât semnalul este 1), folosind modificarea stării unui pin (prin digitalWrite) la intervale de timp controlate prin delay() sau millis(). Folosiți întreruperile externe de pe altă placă pentru a determina caracteristicile acestui semnal, și a le afișa pe LCD.

Referințe

<https://www.arduino.cc/en/Reference/LiquidCrystal>
<https://www.arduino.cc/en/Reference/AttachInterrupt>