

ÓBUDAI EGYETEM  
KANDÓ KÁLMÁN VILLAMOSMÉRNÖKI KAR



**Sándor Tamás**

## Programozás II.

ÓE-KVK 2149

Budapest, 2018.

Készítette:

**Sándor Tamás, adjunktus, Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar, Műszertechnikai és Automatizálási Intézet**

Köreműködött:

**Bedő Sándor, villamosmérnök**

**Mészáros Dániel, villamosmérnök, Óbudai Egyetem Kandó Kálmán Villamosmérnöki Kar, Műszertechnikai és Automatizálási Intézet**

**Borsos Döníz, villamosmérnök, biztonságtechnikai mérnök**

Lektorálta:

**dr. Illés Zoltán PhD, egyetemi docens, Eötvös Lóránd Tudományegyetem Informatikai Kar**

**ISBN 978-963-449-099-9**

# Tartalom

1	Előszó.....	11
2	Számítógép architektúrák.....	12
2.1	Neumann architektúra .....	12
2.2	Harvard architektúra.....	13
2.3	CPU, központi műveletvégző egység.....	14
3	8 bites Harvard architektúra bevezetés .....	15
3.1	Regiszterek.....	16
3.1.1	Általános felhasználású regiszterek (r0-r31) .....	16
3.1.2	Speciális felhasználású regiszterek .....	17
3.1.3	Státusz regiszter (Status Register).....	17
3.1.4	Utasításmutató (IP, Instruction Pointer) /Program számláló (PC, Program Counter) .....	18
3.1.5	Veremmutató (Stack Pointer).....	18
3.2	Memória felépítése .....	19
3.2.1	Adatmemória, a RAM terület felépítése .....	19
3.2.2	Programmemória, a Flash terület felépítése .....	20
3.3	Portkezelés .....	21
3.3.1	I/O portok.....	21
3.3.2	I/O lábak funkciói.....	22
4	Assembly nyelvű programozás.....	26
4.1	Assembly nyelv és az assembler .....	26
4.2	Nyelvi elemek .....	26
4.2.1	Utasítások .....	26
4.2.2	Címzési módok .....	26
4.2.3	Assembly alapkód felépítése .....	27
4.2.4	Assembly forráskódú program fordításának lépései .....	28
4.2.5	Programutasítások csoportok .....	28
5	8 bites Harvard architektúra .....	40
5.1	Megszakítások.....	40
5.1.1	Fogalma, forrásai.....	40
5.1.2	Megszakítás vektortábla .....	40
5.1.3	Atmega128 Timer megszakítások .....	40
5.2	Időzítők / Számlálók az ATmega128 -ban .....	41
5.2.1	Timer/Interruptok .....	41
5.2.2	CTC mód (Clear Timer on Compare Match) .....	43

5.2.3	Timer0 CTC mód mintakódok.....	47
5.2.4	Timer0 OVF mintakódok.....	50
5.2.5	Timer1 beállítása .....	52
5.2.6	Timer1 CTC mód mintakódok.....	57
5.2.7	Timer1 OVF megszakítás mintakódok.....	59
5.3	PWM.....	61
5.3.1	Timer/Counter0 PWM.....	62
5.3.2	Timer0 PWM mintakódok .....	64
5.3.3	Timer/Counter1 PWM.....	66
5.3.4	Timer1 PWM mintakódok .....	67
5.4	Soros portkezelés.....	70
5.4.1	Bevezetés .....	70
5.4.2	ATmega128 USART jellemzői .....	70
5.4.3	Mintakód.....	73
5.5	SPI.....	75
5.5.1	Bevezetés .....	75
5.5.2	ATmega128 SPI jellemzői .....	76
5.5.3	Mintakódok.....	77
5.6	TWI.....	79
5.6.1	Bevezetés .....	79
5.6.2	ATmega128 TWI jellemzői .....	80
5.6.3	Mintakódok.....	82
6	T-bird3 .....	84
6.1	Hardveres felépítése .....	84
6.1.1	Csatlakozók ismertetése .....	84
6.1.2	JTAG debugger használata .....	87
6.1.3	Fontos tudnivalók.....	89
6.1.4	Alapáramkör .....	90
6.1.5	LED-ek és GOMB-ok.....	91
6.1.6	T-bird szalagkábel kiosztás .....	103
6.2	Kiegészítő áramkör: T-Bird – Expansion Board .....	104
6.2.1	Megjelenítő eszközök.....	104
6.2.2	Bemeneti eszközök.....	117
6.2.3	Csatlakozó kiosztás .....	123
	Programozás II. laboratórium AVR Assembly .....	125
	1. labor .....	126

1	Bevezetés.....	126
1.1	Linkek, letöltések .....	126
1.2	AVR Studio 4 használata .....	127
1.2.1	Új projekt létrehozása.....	127
1.2.2	Fordítás .....	130
1.2.3	Programozó beállítása .....	130
1.2.4	Programozás.....	132
1.2.5	Súgó.....	134
1.2.6	Debug .....	134
1.3	Alap Assembly kód .....	135
1.4	Önálló feladatok.....	135
2	Alapvető utasítások megismerése, debugger .....	136
2.1	Adatmozgató utasítások.....	136
2.1.1	Közös feladat: Adatmozgató utasítások.....	137
2.2	Aritmetikai utasítások és logikai utasítások.....	138
2.2.1	Közös feladat: Aritmetikai utasítások.....	138
2.2.2	Közös feladat: Logikai utasítások .....	140
2.3	Bit és bitteszt utasítások .....	141
2.3.1	Közös feladat: Bit utasítások .....	141
2.4	Önálló feladatok .....	142
3	Portkezelés alapjai: LED-ek.....	143
3.1	LED-ek .....	143
3.2	LED(ek) bekapcsolása .....	144
3.2.1	Közös feladat: LED bekapcsolása.....	144
3.2.2	Közös feladat: Műveletvégrés eredményének megjelenítse LED-eken.....	145
3.3	Önálló feladatok .....	146
4	Makrók .....	147
4.1	LED init makró .....	147
4.1.1	Közös feladat .....	147
4.2	Önálló feladatok.....	148
5	Gyakorló feladatok .....	149
2. labor	.....	150
1	„Ciklusok” .....	150
1.1	Feltételes ugró utasítások.....	150
1.1.1	Közös feladat: Összegzés, BREQ, BRNE, CPI .....	151
1.2	Késleltetés, LED villogtatás.....	153

1.2.1	Közös feladat: LED0 villogtatása .....	153
1.3	Önálló feladat.....	154
2	Futófény .....	155
2.1.1	Közös feladat: Egyszerű (körbe) futófény .....	155
2.1.2	Közös feladat: Körbefutófény .....	157
2.2	Önálló feladat.....	158
3	Osztás.....	159
3.1	Közös feladat: Osztás művelet visszavezetése kivonásra.....	159
3.2	Önálló faladat.....	160
4	Portkezelés alapjai: Nyomógombok .....	161
4.1	Gombkezelés.....	161
4.1.1	Közös feladat: Lenyomott gomb számának megfelelő LED világít .....	161
4.1.2	Közös feladat: GOMB menü készítése.....	163
4.2	Önálló feladat.....	164
5	Gyakorló feladatok .....	165
3. labor	.....	166
1	Szubrutinok .....	167
1.1	Közös feladat: STACK_init makró, PUSH, POP.....	167
1.2	Közös feladat: LED_out szubrutin .....	169
1.3	Önálló feladat.....	170
2	Adatmemória írása .....	171
2.1	Közös feladat: Memória írása növekvő címekre .....	171
2.2	Közös feladat: Memória írása csökkenő címekre .....	173
2.3	Közös feladat: 2bájtos szám memóriába írása .....	175
2.4	Önálló feladatok .....	176
3	Adatmemória olvasása .....	177
3.1	Közös feladat: Adatmemóriába írás, olvasás, összegzés .....	177
3.2	Közös feladat: Memóriába írás, olvasás, összegzés, szubrutin, 16bit .....	179
3.3	Önálló feladatok .....	180
4	Másolás .....	181
4.1	Közös feladat: Memóriaterület másolása.....	181
4.2	Önálló feladat.....	182
5	Gyakorló feladatok .....	183
4. labor	.....	184
1	Timer0 CTC, OVF .....	185
1.1	Közös feladat: Timer0, /1024/250, rotálás.....	185

1.2	Közös feladat: TIM0 CTC, LED2 villogtatása .....	187
1.3	Közös feladat: Timer0 OVF, 1024 előosztás, LED0 villogtatása .....	189
1.4	Önálló feladat.....	190
2	Portkezelés alapjai: 7szegmenses kijelző 1digit.....	191
2.1	Közös feladat: 7szegmenses kijelző 1digit .....	192
2.2	Önálló feladatok.....	193
3	Portkezelés alapjai: 7szegmenses kijelző 4digit.....	194
3.1	Közös feladat: 4jegyű szám megjelenítése a 7szegmenses kijelzőn .....	194
3.2	Extra közös feladat: 7szegmenses kijelző 4 digit, delay .....	197
3.3	Önálló feladat.....	199
4	Megszakítások alkalmazása 1. ....	200
4.1	Közös feladat: 7szegmenses 4digit, megszakítás.....	200
4.2	Extra közös feladat: 7szegmenses kijelző 4digit, megszakítás .....	203
4.3	Önálló feladatok.....	206
5	Gyakorló feladatok .....	207
5. labor	.....	208
1	Timer1 CTC, OVF .....	210
1.1	Közös feladat: TIM1 OVF, LED0 villogtatása .....	210
5.1	Közös feladat: TIM1 COMPA, LED villogtatás 1s.....	212
5.2	Önálló feladat.....	213
2	Megszakítások alkalmazása 2. ....	214
2.1	Közös feladat: Számláló a 7szegmenses kijelzőre, 1 digiten (0-9) .....	214
2.2	Közös feladat: 7szegmenses kijelző, számláló (0-99).....	217
2.3	Közös feladat: Másodperces számláló a 7szegmenses kijelzőre 1.....	221
2.4	Közös feladat: 7szegmenses kijelző, számláló (0-99), delay nélkül .....	225
2.5	Önálló feladatok.....	228
3	PWM .....	229
3.1	Közös feladat: Timer0 PWM .....	229
3.2	Közös feladat: TIM1 PWM .....	231
3.3	Önálló feladatok.....	232
4	RGB LED .....	233
4.1	Közös feladat: RGB piros LED villogtatása.....	233
4.2	Extra közös feladat: RGB zöld LED PWM .....	235
5	Gyakorló feladatok .....	237
6. labor	.....	238
1	Portkezelés alapjai: Mátrixbillentyűzet.....	239

1.1	Közös feladat: Lenyomott szám megjelenítése.....	239
1.2	Közös feladat: Megszakítással .....	242
2	Mátrix billentyűzet összeadás .....	245
2.1	Közös feladat: Összeadás .....	245
3	Mátrix billentyűzet kivonás.....	249
3.1	Közös feladat: Kivonás.....	249
4	Mátrix billentyűzet: Szorzás.....	253
4.1	Önálló feladat.....	253
	Programozás II. laboratórium AVR C .....	254
1.	Labor.....	255
1	Portkezelés: LED-ek .....	256
1.1	Közös feladat: Műveletvégzés megjelenítése LED-eken .....	256
1.2	Közös feladat: LED villogtatás, delay .....	257
1.3	Közös feladat: Futófény 1.....	258
1.4	Közös feladat: Futófény 2.....	259
1.5	Önálló feladatok .....	259
2	Portkezelés: Nyomógombok .....	260
2.1	Közös feladat: Gombok beolvasása .....	260
2.2	Közös feladat: Gomb menü .....	261
2.4	Közös feladat: Gomb kapcsoló – számláló .....	262
2.5	Önálló feladat.....	263
3	Timer0 .....	264
3.1	Közös feladat: Timer0 CTC mód, számláló.....	264
3.2	Közös feladat: Timer0 OVF IT .....	265
3.3	Közös feladat: Timer0 CTC IT 1s .....	266
3.4	Önálló feladat.....	267
4	Timer0 alkalmazása .....	268
4.1	Közös feladat: Timer0 HW PWM LED0.....	268
4.2	Közös feladat: Timer0 SW PWM.....	269
4.3	Önálló feladatok .....	270
5	Gyakorló feladatotok .....	271
2.	Labor.....	272
1	Portkezelés: 7szegmenses kijelző .....	273
1.1	Közös feladat: 7szegmenses kijelző 1 digit .....	273
1.2	Közös feladat: 7szegmenses kijelző 4digit delay .....	274
1.3	Közös feladat: 7szegmenses kijelző 1digites számláló .....	275

1.4	Önálló feladatok .....	276
2	Timer1 .....	277
2.1	Közös feladat: Timer1 CTC mód megszakítás 1s .....	277
2.2	Közös feladat Timer1 OVF IT .....	278
2.3	Önálló feladatok .....	278
3	Megszakítások alkalmazása: számlálók .....	279
3.1	Közös feladat: 7szegmenses kijelző 4digites szám kiírása megszakítással .....	279
3.2	Közös feladat: 7szegmenses kijelző 2digites számláló .....	280
3.3	Közös feladat: 2digites számláló megszakítással .....	282
3.4	Önálló feladatok .....	283
4	Megszakítások alkalmazása: Óra .....	284
4.1	Közös/önálló feladat .....	284
5	Gyakorló feladatok .....	285
3.	Labor .....	286
1	Mátrix billentyűzet .....	286
1.1	Közös feladat: Lenyomott mátrix bill megjelenítése a 7szegmenses kijelzőn .....	286
1.2	Közös feladat: Mátrix bill olvasása megszakítással .....	288
1.3	Önálló feladat: .....	289
2	Mátrix billentyűzet: Menü .....	290
2.1	Önálló feladat .....	292
3	Mátrix billentyűzet: többjegyű szám bevitel .....	293
3.1	Önálló feladat .....	294
4	Timer-ek alkalmazása: Timer1/3 PWM .....	295
4.1	Közös feladat: Timer1 Fast PWM ICR1 .....	295
4.2	Közös feladat: RGB zöld PWM .....	296
4.3	Önálló feladatok .....	296
5	Gyakorló feladatok .....	297
4.	Labor .....	298
1	Mátrix billentyűzet: Számológép1 .....	299
1.1	Közös feladat: Egyjegyű számológép .....	299
1.2	Önálló feladat .....	301
2	Mátrix billentyűzet: 2jegyű számológép .....	302
2.1	Közös feladat: Számológép 2 .....	302
2.2	Önálló feladat .....	304
3	Mátrix billentyűzet: Kódzár .....	305
3.1	Közös feladat: kódzár .....	305

3.2	Önálló feladat.....	307
4	Gyakorló feladatok .....	308
5.	Assembly ZH-ra való felkészüléshez .....	309
6.	AVR C ZH-ra való felkészüléshez.....	314
	Megszakítás vektortábla .....	318

## 1 Előszó

A Programozás II. tantárgy célja, hogy a korábban megismert programozási alapismereteket már valós, mikrokontrolleres környezetben fejleszteni tudják a hallgatók. A tárgy kereteiben először a mikrokontroller felépítésével ismerkedhetünk meg építve a Digitális technika téma területén elsajátított ismeretekre, majd az assembly nyelvű programozás alapjai kerülnek elsajátításra, és végül ismertetésre kerül ugyanannak a mikrokontrollernek a magas szintű programozási nyelven történő kódolása.

A jegyzetben ismertetésre kerül tárgyhoz kapcsolódó laborban (Programozás II. laboratórium) már 2009 óta használt T-bird nevű fejlesztő board, amely megfelelő segítséget nyújt az Atmel cég Atmega128 8 bites mikrokontrollerének megismeréséhez, illetve ehhez a fejlesztő boardhoz kapcsolódó kiegészítő boardon található perifériák megismeréséhez (hétszegmenses kijelző, billentyűzetmatrix, 4x16 karakteres LCD kijelző, háromszínű LED, analóg hőmérséklet érzékelő, USB, RS485).

Az eszköz természetesen amellett, hogy a mikrokontroller programozás hatékony megismerését segíti, alkalmas későbbi tárgyak (Beágyazott rendszerek, Projekt I., Projekt II. tárgyak, Információs rendszerek) önálló feladatainak megvalósításához, hiszen a önműködő robotuktól, az intelligens kaspón keresztül a beszélő T-bird-ig sokféle alkalmazás készült felhasználásával, illetve a kiegészítő board felhasználásra került 32 bites mikrokontrollerek oktatásában, illetve FPGA-hoz (Nexys4) is illesztésre került.

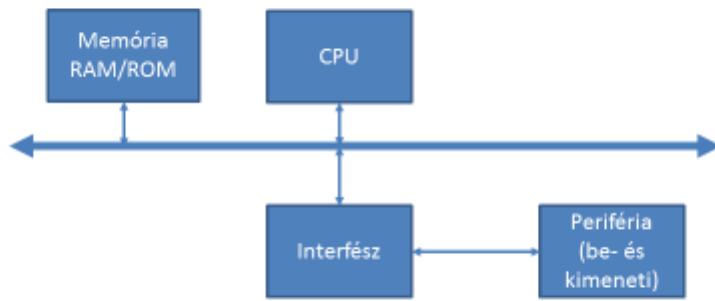
Budapest, 2015.03.31.

*Szerző*

## 2 Számítógép architektúrák

### 2.1 Neumann architektúra

#### Neumann féle architektúra



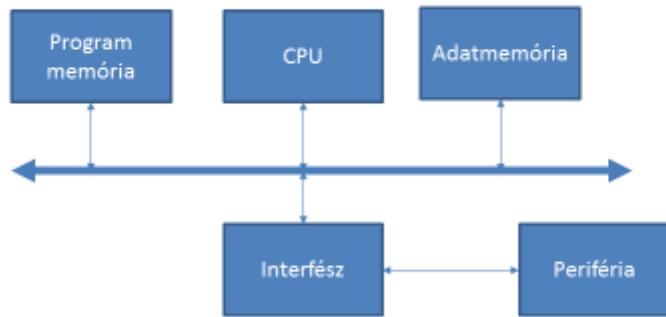
1. ábra - Neumann architektúra

A Neumann architektúra főbb részei a következők:

- Központi vezérlőegység (CPU, Central Processing Unit),
- Memóriák,
  - o RAM (Random Access Memory),
  - o ROM (Read Only Memory),
  - o FLASH,
- Interfészek,
  - o Általános felhasználású be- és kimentek (GPIO, General Purpose Input Output),
- Szabványos interfészek ( $I^2C$ , TWI, SPI, ).

## 2.2 Harvard architektúra

### Harvard féle architektúra



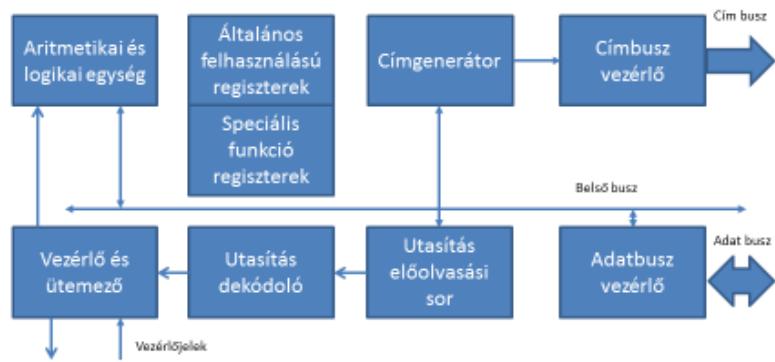
2. ábra - Harvard architektúra

*A Harvard architektúra főbb részei a következők:*

- Központi vezérlőegység (CPU, Central Processing Unit),
- Memóriák,
  - o RAM (Random Access Memory),
  - o ROM (Read Only Memory),
  - o FLASH,
- Interfészek,
  - o Általános felhasználású be- és kimentek (GPIO, General Purpose Input Output),
- Szabványos interfészek ( $I^2C$ , TWI, SPI).

## 2.3 CPU, központi műveletvégző egység

### Központi műveletvégző egység (CPU, Central Processing Unit)

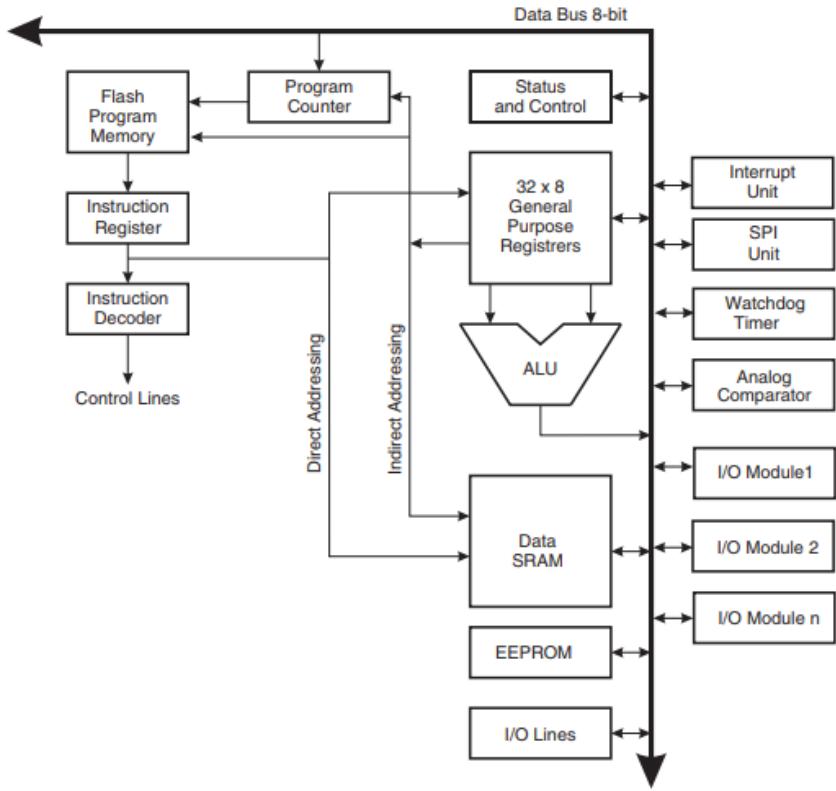


3. ábra - Központi műveletvégző egység

*A CPU főbb egységei a következők:*

- ALU (Aritmetikai és logikai egység)
- Regiszterek
  - o Általános felhasználású regiszterek
  - o Speciális funkciójú regiszterek
- Utasítás dekódoló
- Vezérlő- és ütemező egység

### 3 8 bites Harvard architektúra bevezetés



4. ábra – ATmega CPU megja

Az Atmega128 mikrokontroller az alábbi főbb egységeket tartalmazza:

- ALU (Arithmetic and Logic Unit), aritmetikai és logikai egység, amelynek a feladata az operandusokkal történő műveletvégzés. A műveletvégzés bemeneti forrásainak ismertetése későbbiekben az egyes assembly utasításoknál kerül részletesebben ismertetésre.
- General Purpose Registers, általános felhasználású regiszterek (32 darab, r0-r31), amelyek 8 bites nagyságúak, de r26-r31-as regiszterek regiszterpárként is értelmezhetőek, X, Y és Z regiszterek index regiszterként is használatosak. Ezek a regiszterek a RAM terület első 32 bájtját adják.
- 4 kbyte DATA SRAM, amelyet 0x100 címtől már a felhasználó is szabadon programozhat.
- 128 kbyte Flash program memória, ahova a felhasználó a programjait feltöltheti.

### 3.1 Regiszterek

A regiszterek (belőlő tároló területek, amelyeket a mikrokontroller közvetlenül felhasználhat műveletvégzéshez) az SRAM területen találhatók.

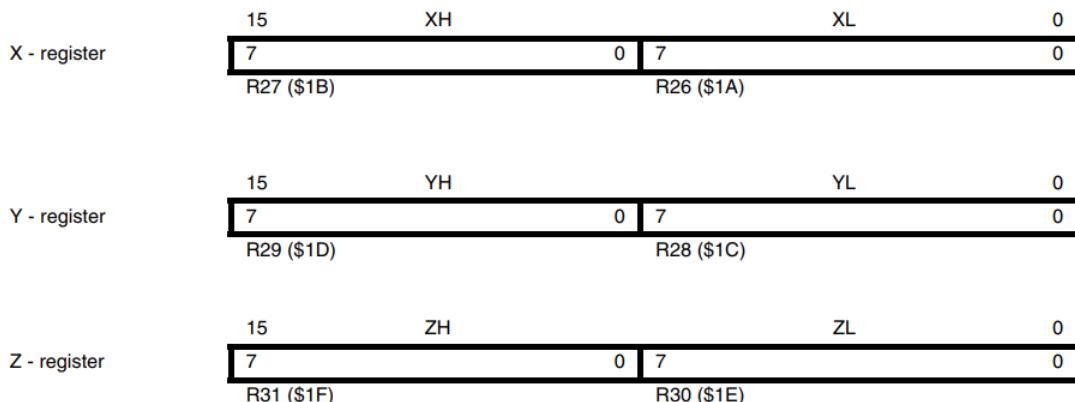
#### 3.1.1 Általános felhasználású regiszterek (r0-r31)

General Purpose Working Registers	7	0	Addr.
	R0		\$00
	R1		\$01
	R2		\$02
	...		
	R13		\$0D
	R14		\$0E
	R15		\$0F
	R16		\$10
	R17		\$11
	...		
	R26		\$1A
	R27		\$1B
	R28		\$1C
	R29		\$1D
	R30		\$1E
	R31		\$1F
			X-register Low Byte
			X-register High Byte
			Y-register Low Byte
			Y-register High Byte
			Z-register Low Byte
			Z-register High Byte

5. ábra - Általános felhasználású regiszterek

#### Általános felhasználású regiszterek:

- 32 darab 8 bites általános felhasználású regiszter, ebből:
  - o 26 darab 8 bites regiszter általános
  - o 3 db 16 bites indirekt címzés megvalósítására szolgáló indexregiszter:
    - X index regiszter, adatterület címzésére használatos indexregiszter.
    - Y index regiszter, adatterület címzésére használatos indexregiszter.
    - Z index regiszter, kódterület címzésére használatos indexregiszter (Flash program memória címzés).



6. ábra - Indexregiszterek

### 3.1.2 Speciális felhasználású regiszterek

- PC (Program Counter), Utasítás mutató (IP)
- SP (Stack Pointer), veremmutató, a verem következő szabad helyére mutat.
- SR (Status Register), a mikrokontroller állapotregisztere, általában a mikrokontroller állapotát jelzi műveletvégzést követően.
- MCU Control Register (IVCE, IVSEL).

### 3.1.3 Státusz regiszter (Status Register)

A státusz regiszter bitek a mikrokontroller állapotát jelzik műveletvégzést követően.

Nézzük az egyes bitek funkcióját!

Bit	7	6	5	4	3	2	1	0	SREG
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

- I Global Interrupt Enable, általános megszakítás engedélyező bit, ha az értéke 0, akkor tiltott minden maszkolható megszakítás, ha az értéke 1, akkor általánosan engedélyezett a megszakítások használata. Ekkor még egyedileg az engedélyezéseket el kell végezni.
- T Bit Copy Storage (BLD vagy BST forrás vagy cél), a felhasználó által szabadon állítható bit.
- H Half Carry Flag (BCD), a 3. bitről a 4. bitre történő átvitel, amely bitnek a jelentősége BCD számokkal történő műveletvégzés esetén van.
- S Sign Bit ( $S = N \oplus V$ ), előjele bit, amely előjeles számokkal történő műveletvégzés esetén használatos.
- V Overflow, túlcordulás bit, előjeles számábrázolás esetén műveletvégzés után a számtartomány túllépését jelzi.
- N Negative, műveletvégzés után az eredmény MSB (Most Significant Bit) bitjének másolata.
- Z Zero, műveletvégzés után a bit értéke 1, ha az eredmény 0.
- C Carry, átvitel bit, előjele nélküli számábrázolás esetén a műveletvégzést követően a számtartomány túllépését jelzi.

### 3.1.4 Utasításmutató (IP, Instruction Pointer) /Program számláló (PC, Program Counter)

A következő végrehajtandó utasítás címét tartalmazza. Értékét az utasítás dekóder a beolvastott utasítás dekódolása után állítja. Továbbá állíthatja:

- Szubrutin hívás,
- Megszakítási rutin hívás,
- Elágazó utasítás.

### 3.1.5 Veremmutató (Stack Pointer)

A verem (STACK) az általános SRAM területen foglalt hely (max mérete a SRAM területe). A veremmutató segítségével lehet megcímzni a beírásnál, illetve a kiolvasásnál az adatokat. Stack Pointer (SP) egy 16 bites regiszter, amely kezelhetünk két darab 8 bites regiszterként is (SPH és SPL). A veremterület a RAM terület végétől kezdődik (RAMEND), és egészen a speciális felhasználású regiszterekig tarthat (0x100), ha a felhasználói programnak nincsen változók számára lefoglalt területe. Verem használatra szubrutin vagy megszakításhíváskor van szükség, így a veremmutató (SP) inicializálása nélkül ezek a programegységek nem is fognak működni, ha assembly programot készítünk.

**Példa a verem inicializálására assembly<sup>1</sup> nyelvű programozáskor:**

```
ldi      r16,  high RAMEND
out     SPH,   r16
ldi      r16,  low RAMEND
out    SPL,   r16
```

Ekkor a veremmutató (SP) a RAM terület végére fog mutatni.

**Verem (Stack) megoldások:**

- Szoftveres verem:
  - o Általános SRAM területen foglalt hely (max mérete a SRAM területe), Atmel AVR
- Hardveres verem:
  - o Fixen kijelölt belső tároló terület, amely LIFO (Last In First Out) módon működik, PIC

---

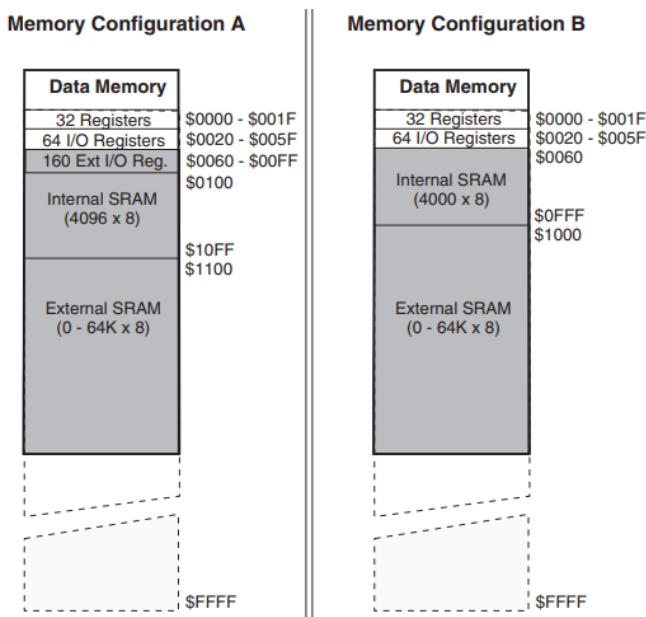
<sup>1</sup> Az Assembly nyelvű programozással a következő fejezet foglalkozik.

## 3.2 Memória felépítése

### 3.2.1 Adatmemória, a RAM terület felépítése

Az adatmemória (SRAM) felépítése a 4.ábrán látható. A memória terület első 32 bájtnál az általános hozzáférésű regiszterek (r0-r31) találhatók, majd ezt követi a 64 bájton keresztül az I/O regiszterek, és végül 160 bájton keresztül kiterjesztett I/O terület következik 0xFF címig. A felhasználói programok 0x100-tól kezdődhet, és 4kbyte nagyságú SRAM területet használhatják. Természetesen nem szabad elfelejtkezni a verem területről, amely SRAM terület végétől (RAMEND) építkezik visszafele.

Az egyes assembly utasítások különböző címtartományokban használhatók, nem szabad összekeverni a használatukat, mert a fordító nem fog jelezni szintaktikai hibát, de a program nem azt fogja végrehajtani, amit várnánk tőle.



7. ábra - Adatmemória (SRAM) terület felépítése

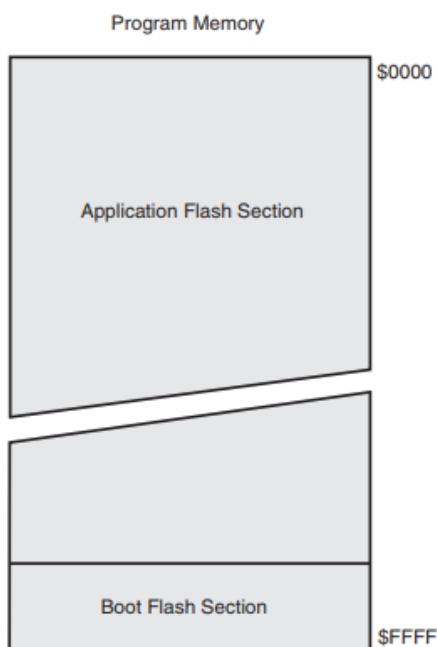
Az alábbiakban összefoglalásra kerül, hogy melyik címtartományban melyik adatmozgató utasítás használható:

- 0x00 – 0x1F közvetlen bitenkénti kezelés (SBI, CBI), vagy adatmozgató utasítás (MOV),
- 0x00 – 0x3F IN vagy OUT utasítással,
- 0x60 – 0xFF ST/STS/STD, LD/LDS/LDD.

### 3.2.2 Programmemória, a Flash terület felépítése

Az elkészített felhasználói programokat JTAG programozó vagy ISP programozó segítségével a programkód területre lehet letölteni. A Harvard architektúrából adódóan az adat és a programkód terület külön címtartományban található, így a programkód terület is a 0x0000 címtől kezdődik. Ennek a területnek az elején találhatjuk meg a megszakítás vektortáblát, amely a függelékben megadott táblázatban található megszakítást kiszolgáló rutinoknak a szabványos címeit tartalmazza.

Lehetőség van arra is, hogy a mikrokontroller bootloader üzemmódban induljon el, ekkor a Boot Flash Section (a bootloader üzemmódnak megfelelő) programterületen található kód indul el.



**8. ábra - Programmemória, a Flash terület felépítése**

Ez a funkció nagyon jól használható programozó eszköz nélkül történő programfrissítésre, hiszen, ha a Boot Flash Section területre olyan kód kerül letöltésre (természetesen ekkor még alkalmazni kell programozó eszközt), amelyik például soros porton keresztül feltölti az új programot az Application Flash Section-be, akkor ezzel megoldhatjuk azt is, hogy egy távoli eszközre töltsünk le programot (tipikusan például mérésadatgyűjtő eszközök távoli firmware frissítése).

### 3.3 Portkezelés

#### 3.3.1 I/O portok

Az AVR-ek meglehetősen sok és különféle bemeneti/kimenet portokkal rendelkeznek attól függően, hogy milyen típusú mikrokontrollert választunk. Esetünkben a Programozás II. laboratórium keretein belül az ATmega128 típusú mikrokontrollerrel találkozhatunk.

A laboron használt mikrokontroller 53 darab I/O lábbal rendelkezik, amelyeket 7 portra osztottak szét: PORTA, PORTB, PORTC, PORTD, PORTE, PORTF, PORTG [0-4].

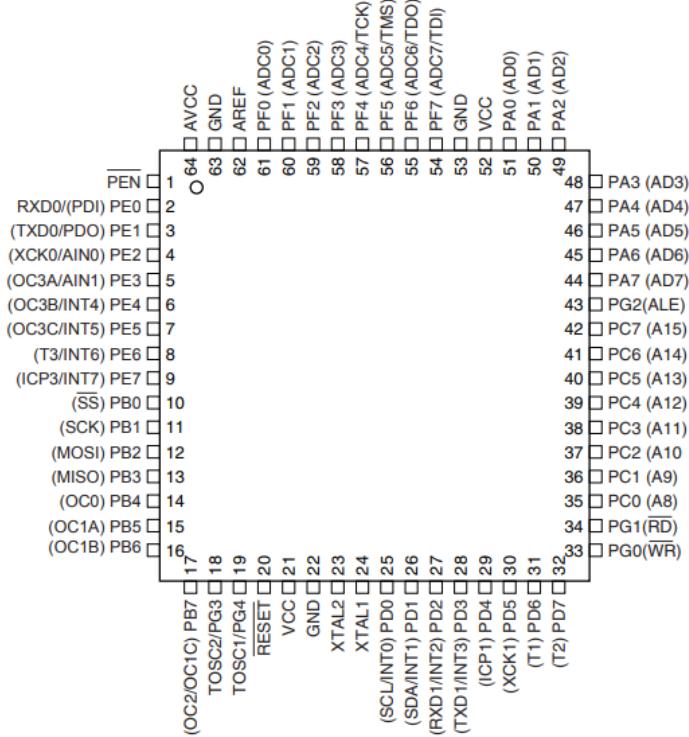
Ezen felül a legfontosabb lábak a következők:

Láb neve	Megnevezése
VCC	pozitív tápfeszültség
GND	föld
XTAL1-2	külső kvarc bemenet
RESET	reset bemenet
AREF	ADC referencia feszültsége
AVCC	analóg rész pozitív feszültsége
AGND	analóg rész föld

Fontos megjegyezni, hogy az AVR mikrokontroller kimenete portonként maximálisan 125mA-t képes kiadni magából, és a bemeneti feszültség nem haladhatja meg a tápfeszültséget. Ezen értékek figyelembevétele nélkül a mikrokontroller károsodást szenvedhet, tönkremehet.

### 3.3.2 I/O lábak funkciói

Az AVR lábainak funkciót minden esetben megtaláljuk az adatlapban, de esetünkben az ATmega128 mikrokontrollert ebben a részben egy kicsit részletesebben megvizsgáljuk.



9. ábra - ATmega 128 lábkiosztása

Mint a képen is látható a lábak mellett zárójelben találhatóak úgynevezett második funkciók. Nézzük is meg ezek jelentéseit:

Megnevezés	Funkció leírása
RXD <sub>x</sub> / TXD <sub>x</sub>	UART <sub>x</sub> fogadó és küldő lábai (bemenet, kimenet)
AIN <sub>x</sub>	ADC analóg bemenetei
INT <sub>x</sub>	külső megszakítás bemenetek
SCL / SDA	TWI busz adatvonalai
OC <sub>x</sub>	8 bites Timer/Counter kimenetek
OCxA/B/C	16 bites Timer/Counter kimenetek
TCK, TMS, TDI, TDO	JTAG programozó adatvonalai (ha JTAG programozával programozzuk a mikrokontrollert, akkor azok a lábak nem használhatók fel I/O lábként!)
MOSI, MISO, SCK, SS	SPI busz adatvonalai

A táblázatban említett funkciók használatát a későbbi fejezetekben részletesen taglalni fogjuk.

Vizsgáljuk meg, hogy az AVR I/O lábait hogyan tudjuk beállítani:

- **bemenetnek / kimenetnek állítás**

Először is fontos megemlíteni, hogy ha bármit is be szeretnénk állítani az AVR-en meg kell hívni az “avr/io.h” header fájlt, mert ez a fájl tartalmazza az AVR belső felépítésének megfelelően definiált kifejezéseit, így ennek köszönhetően nem memória címeket kell írni a kódunkba, hanem ezeknek megfelelő hivatkozásokat.

Az I/O lábak irányait a “DDRx” regiszterrel tudjuk állítani, ahol az “x” az adott port nevét jelenti. Nézzünk is egy példát a D port beállítására:

DDRD |= 0b11110000;

A D port (PORTD) felső 4 bitjét kimenetnek, az alsó 4-et pedig bemenetnek állítottuk be. Ebből rögtön kiderül, hogy ha az irány beállító regiszterbe ’1’-et írunk, akkor azt a port bitet kimenetként, ha ’0’-t írunk, akkor pedig bemenetként fogjuk tudni használni.

- **kimenet beállításai**

A láb logikai jelszintjét a “PORTx” regiszter segítségével lehet beállítani, ahol az “x” itt is a port nevét jelenti. Példa a D port beállítására:

PORTD |= 0b11110000;

A D port (PORTD) felső 4 lábán logikai ’1’, az alsó 4 lábon pedig logikai ’0’ jelszint fog megjelenni.

- **bemenet további beállításai**

Ha a láb bemenetnek van állítva, akkor bekapsolható az AVR belső felhúzó ellenállása, amely olyan esetekben nagyon hasznos, ha egy gombnak hardveresen nem kötöttek be felhúzó ellenállást, és minden lenyomásnál jelentkezik a prell jelenség.

A beállítása a következő:

DDRx	PORTx	I/O	Belső felhúzó ellenállás
0	0	bemenet	nem
0	1	bemenet	igen

- **bemenet vizsgálata**

Sokszor adódik olyan helyzet, hogy vizsgálni kell AVR felhasználásával egy jel logikai értékét. Ekkor az adott láb bemenetnek állítását követően a “PINx” definiált értékből lehet kiolvasni, ahol az “x” a port nevét jelenti.

T-bird-ön a gombok a G porton kerültek elhelyezésre, így a példa ezeknek a gomboknak a használatára mutat példát:

```
1 #include <avr/io.h>
2
3 int main()
4 {
5     PORT_init();
6     while(1)
7     {
8         if(PING & 0b00000001)
9         {
10             LED_out(0x10);
11         }
12         else
13         {
14             LED_out(0x04);
15         }
16     }
17 }
18
19 void LED_out(unsigned char szam)
20 {
21     PORTD = szam & 0xF0;
22     PORTB = (szam << 4);
23 }
24 void PORT_init(void)
25 {
26     DDRG |= 0b00000000; //gombok bemenetek
27     DDRD |= 0b11110000; //LED-ek kimenetek
28     DDRB |= 0b11110000; //LED-ek kimenetek
29 }
```

Nézzük ennek az egyszerű programnak a leírását:

1. Általánosan használatos header fájl, az AVR definíciók használatához (pl. DDRB, PORTD, ...)
- 3.-17. main függvény feje, illetve a függvény törzse
5. PORT\_init() függvény meghívása, az irányregiszterek beállítása (DDRB, DDRD, DDRG)
- 6.-16. egy végtelen while ciklusban a PING0-án levő gomb lenyomását figyelve, ennek megfelelően a PORTD4-re, illetve PORTB6-ra kötött LED-ek aktiválódnak.
- 19.-23. LED\_out függvény segítségével egy 0 és 255 közötti számot megjeleníthetünk a T-bird-ön található 8 db LED-en. A T-bird adatlapjából kiderül, hogy a LED0-LED3 a PORTB7-PORTB4-re, illetve a LED7-LED4 a PORTD7-PORTD4-re kerültek elhelyezésre.
- 24.-29. PORT\_init függvény definíciós része, ahol a konkrét irányregiszter beállításokat találhatjuk (emlékeztetőül '0': bemeneti irány, '1': kimeneti irány).

## 4 Assembly nyelvű programozás

### 4.1 Assembly nyelv és az assembler

Az assembly programozási nyelv a mikrokontroller utasításkészletére épül (Atmega128 esetében ez 133 utasítás), és ezeket az utasításokat 2, 3 vagy 4 betűs memonikok segítségével írja le.

Az assembler az egy compiler típusú fordító program, amely a forráskódú állományból állít elő tárgykódú állományt, amelyből a linker fogja előállítani a letölthető .hex állományt.

### 4.2 Nyelvi elemek

#### 4.2.1 Utasítások

- **Fordítónak szóló utasítások**
  - o Direktíva vagy pszeudo utasítás.
- **Programutasítások**
  - o Aritmetikai és logikai utasítások
  - o Vezérlésátadó utasítások
    - Ugró utasítások
    - Szubrutin hívások
    - Komparáló utasítások
  - o Bitállító és -tesztelő utasítások
    - Interrupt utasítások
  - o Adatmozgató utasítások
  - o MCU vezérlő utasítások

#### 4.2.2 Címzési módok

**3 féle kódterület címzéssel találkozhatunk:**

- Közvetlen kódterület cím
  - Abszolút cím
    - Feltételes ugró utasítás
    - Feltétel nélküli ugró utasítás
    - Szubrutin hívás

- Relatív cím
  - Feltételes ugró utasítás
  - Feltétel nélküli ugró utasítás
  - Szubrutin hívás
- Indirekt cím
  - Indirekt szubrutin hívás

**5 féle adatterület címzési mód van:**

- Regiszter címzés
- Regiszter indirekt címzés
- Memória címzés
- Memória indirekt címzés
- Verem terület címzése

#### 4.2.3 Assembly alapkód felépítése

```
.INCLUDE "m128def.inc"      ; Include definitions Atmega128
.DSEG
;
.CSEG
.ORG 0x0
    rjmp start           ; jump to start
.ORG 0x100
;-----
; MACRO
;-----
;-----
; SUBRUTINE
;-----
;-----
; INTERRUPT
;-----
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
;-----
; PROGRAM
;-----
loop:
    rjmp loop
```

#### 4.2.4 Assembly forráskódú program fordításának lépései

Forráskód	*.asm
Compiler	
Tárgykód	*.obj
Linker	
Hex kód	*.hex
Debugger	
Debugger kód	*.elf

#### 4.2.5 Programutasítások csoportok

##### 4.2.5.1 Adatmozgató utasítások

Programozás során lapvető feladat az adatok mozgatása, amit az adatmozgató utasítások segítségével valósíthatunk meg. Az adatmozgató utasítások lényege, hogy hatására a céloperandus felveszi a forrásoperandus értékét.

Az egyes regiszterek címe befolyásolja azt, hogy mely adatmozgató utasítást tudjuk használni:

- Általános hozzáférésű I/O regiszter
  - o 0x00 – 0x1F közvetlen bitenkénti kezelés (SBI, CBI)
  - o 0x00 – 0x3F IN vagy OUT utasítással, Közvetlen elérésű I/O terület
  - o 0x60 – 0xFF ST/STS/STD, LD/LDS/LDD, Külső elérésű (SRAM) I/O terület
- GPIO0, GPIO1, GPIO2: R/W

###### 4.2.5.1.1 Move

A MOV (Move Between Registers) utasítással regiszterek (0-31) közötti adatmozgatást tudunk megvalósítani. Egy regiszter másolatát képezi egy másik regiszterbe. A forrásregiszter változatlan marad, a célregiszterbe a forrásregiszter értéke töltődik.

MOV Rd, Rr;               $Rd \leftarrow Rr$

A MOVW (Copy Register Word) utasítás egy regiszterpár másolatát egy másik regiszterpárba helyezi. A forrás regiszterpár értéke változatlan marad, a célregiszterpárba töltődik. A d és az r index páros értékű lehet 0-31 között.

MOVW Rd, Rr;               $Rd+1:Rd \leftarrow Rr+1:Rr$

#### 4.2.5.1.2 Load, store

A **load** utasítások egyike a LDI (Load Immediate) utasítás, amely egy 8bites konstans értéket közvetlenül tölt be regiszterbe (16-31).

LDI Rd, K;    Rd  $\leftarrow$  K

Az LDS (Load Direct from SRAM) utasítás az adattérből 1 bájtot tölt be közvetlenül egy regiszterbe (0-31) 16bites címzés alkalmazásával.

LDS Rd, k;    Rd  $\leftarrow$  (k), SRAM

Az STS (Store Direct to SRAM) utasítás az LDS ellentéte. Egy regiszter értékét tudjuk vele eltárolni az adattérben. Szintén a 0-31 regiszterek egyikét és 16bites címet kell alkalmaznunk.

STS k, Rr;    (k)  $\leftarrow$  Rr

Az LD (Load Indirect) utasítást használatának több módja lehetséges. Az X, Y és Z index regisztereit felhasználva az adattérből közvetlenül tölthetünk értéket regiszterbe (0-31).

LD Rd, X;    Rd  $\leftarrow$  (X)

Lehetőség van az adott indexregiszter utó-inkrementálására (Load Indirect and Post-Inc.), ekkor az utasítás végrehajtása után fog eggyel növekedni az értéke.

LD Rd, X+;    Rd  $\leftarrow$  (X), X  $\leftarrow$  X + 1

Ezen felül elő-dekrementálásra (Load Indirect and Pre-Dec) is van mód, ekkor az indexregiszter értéke eggyel csökken az adatmozgatás előtt.

LD Rd, -X;    X  $\leftarrow$  X - 1, Rd  $\leftarrow$  (X)

Az LDD (Load Indirect with Displacement) utasítás lehetőséget nyújt arra, hogy az Y és a Z indexregisztereit 0-63 közötti értékkal növeljük és az adattérből onnan helyezzük át értéket egy regiszterbe (0-31).

LDD Rd, Y+q;    Rd  $\leftarrow$  (Y + q)

A **store** utasítások pont fordítva működnek, mint a load utasítások. Regiszterből (0-31) töltünk értéket az indexregiszterben eltáron adatterületre.

ST X, Rd ;    (X)  $\leftarrow$  Rr

ST X+, Rd;    (X)  $\leftarrow$  Rr, X  $\leftarrow$  X + 1

ST -X, Rd;    X  $\leftarrow$  X - 1, (X)  $\leftarrow$  Rr

STD Y+q, Rd;    (Y + q)  $\leftarrow$  Rr

Lehetőségünk van a programmemoriából egy bájtot regiszterbe (0-31) tölteni a Z indexregisztert alkalmazva. Az LPM (Load Program Memory) utasítást önmegában használva az érték az R0 regiszterbe kerül.

LPM;    R0  $\leftarrow$  (Z)

LPM Rd, Z;    Rd  $\leftarrow$  (Z)  
LPM Rd, Z+;    Rd  $\leftarrow$  (Z), Z  $\leftarrow$  Z+1

Az SPM (Store Program Memory) utasítással 2bájtos értéket írhatunk a programmemóriába. Az R1 regiszter használatos a felső bájt értékére, az R0 pedig az alsóra. Az utasítás a teljes program memória területén alkalmazható.

SPM;                (Z)  $\leftarrow$  R1:R0

#### 4.2.5.1.3 PUSH, POP

A veremkezelés esetén a PUSH (Push Register on Stack) és a POP (Pop Register from Stack) utasítások használatosak. A PUSH utasítással tárolhatjuk a megadott regiszter (0-31) értékét a veremben. A utasítás végrehajtása után a veremmutató értéke egygyel csökken.

PUSH Rr;            STACK  $\leftarrow$  Rr

A POP utasítás a verem egy bájtját tölti be egy regiszterbe (0-31). Az utasítás végrehajtása előtt a veremmutató értéke egygyel növekszik.

POP Rd;            Rd  $\leftarrow$  STACK

#### 4.2.5.1.4 Bemeneti és kimeneti utasítások

A IN és OUT utasítások a portokhoz, timerekhez, konfigurációkhoz tartozó regiszterek (0-63) írása és olvasása.

IN Rd, P            ; Rd  $\leftarrow$  P

OUT P, Rr;        P  $\leftarrow$  Rr

Példa adatmozgató utasítások használatára:

```
ldi r16, 60          ;60 betöltése az r16 regiszterbe
ldi r18, 10          ;10 betöltése az r18 regiszterbe
mov r17, r18          ;r18 regiszter másolása r17 regiszterbe
movw r21:r20,r17:r16 ;r17, r16 másolása az r21, r20 regiszterekbe
```

#### 4.2.5.2 Aritmetikai és logikai utasítások

##### 4.2.5.2.1 Add

Az ADD (Add two Registers) utasítással két regiszter adható össze, anélkül, hogy a műveletvégzés során Carry flag befolyásolná az eredményt.

ADD Rd, Rr;        Rd  $\leftarrow$  Rd + Rr

Az ADC (Add with Carry two Registers) utasítást alkalmazva a Carry flag értéke is hozzáadódik a két regiszter összegéhez.

ADC Rd, Rr;        Rd  $\leftarrow$  Rd + Rr + C

Az ADIW (Add Immediate to Word) utasítás egy regiszterpárhoz ad hozzá értéket (0-63). Az

utasítás csak a felső 4 regiszterpáron alkalmazható. Tipikus felhasználása az indexregiszterekkel való műveletvégzés.

ADIW RdI, K;       $Rdh:Rdl \leftarrow Rdh:Rdl + K$

Az INC (Increment) utasítás inkrementálásra szolgál, az adott regiszter (0-31) értékét növeli eggyel.

INC Rd;       $Rd \leftarrow Rd + 1$

#### 4.2.5.2.2 Subtract

A SUB (Subtract two Registers) utasítás két regiszter (0-31) egymásból való kivonására szolgál.

SUB Rd, Rr;     $Rd \leftarrow Rd - Rr$

A SUBI (Subtract Constant from Register) utasítással egy regiszterből vonhatunk ki egy konstans értéket (0-255). A műveletvégzésre csak a 16-31 regiszterek valamelyike alkalmazható.

SUBI Rd, K;     $Rd \leftarrow Rd - K$

Az SBC (Subtract with Carry two Registers) utasítás olyan kivonást tesz lehetővé két regiszter (0-31) kötözz, ahol az eredményben a Carry flag értékének kivonása is megjelenik.

SBC Rd, Rr;     $Rd \leftarrow Rd - Rr - C$

Az SBCI (Subtract with Carry Constant from Reg.) utasítás az SBI és az SBC együttes hatását képviseli. Regiszterből (16-31) konstans értéket (0-255) és a Carry flag értékét vonja le.

SBCI Rd, K;     $Rd \leftarrow Rd - K - C$

Az SBIW (Subtract Immediate from Word) utasítás regiszterpárból von ki konstans értéket (0-63). Az ADIW utasításhoz hasonlóan, itt is csak a felső 4 regiszterpár alkalmazható.

SBIW RdI, K;  $Rdh:Rdl \leftarrow Rdh:Rdl - K$

A DEC (Decrement) utasítás dekrementálásra szolgál, az adott regiszter (0-31) értékét csökkenti eggyel.

DEC Rd;       $Rd \leftarrow Rd - 1$

#### 4.2.5.2.3 Multiply

A MUL (Multiply Unsigned) utasítás két regiszter (8bites, előjel nélküli érték) szorzására használatos. Az eredmény az R1 (felső bájt) és R0 (alsó bájt) regisztereiken fog tárolódni. Amennyiben a szorzás során felhasználásra kerül az R1 és/vagy R0 regiszter, az adott érték felül fog íródni a műveletvégzés után.

MUL Rd, Rr;  $R1:R0 \leftarrow Rd \times Rr$

A MULS (Multiply Signed) esetén előjeles értékekkel történik a műveletvégzés, ezt leszámítva

a működése a MUL utasítással azonos. Még egy különbség, hogy csak a 16-31 regisztereket használhatjuk a műveletvégzésre.

MULS Rd, Rr;      R1:R0  $\leftarrow$  Rd x Rr

Ha előjel nélküli számot kívánunk összeszorozni előjelessel, akkor a MULSU (Multiply Signed with Unsigned) utasítást kell használnunk. Ebben az esetben csak a 16-23 regiszterek közül választhatunk.

MULSU Rd, Rr;      R1:R0  $\leftarrow$  Rd x Rr

Példa aritmetikai utasítások használatára:

```
ldi r16, 60          ;60 betöltése az r16 regiszterbe
ldi r18, 10          ;10 betöltése az r18 regiszterbe
add r16, r18         ;r16 és r18 összeadása, eredmény r16 regiszterbe
inc r18              ;r18 inkrementálása
sub r18, r16         ;r18-ból r16 kivonása, eredmény r18 regiszterbe
dec r16              ;r16 dekrementálása
```

#### 4.2.5.2.4 AND

Az AND utasítás logikai és kapcsolatot hajt végre két regiszter (0-31) között.

AND Rd, Rr;      Rd  $\leftarrow$  Rd • Rr

Az ANDI utasítással logikai és kapcsolat valósítható meg egy regiszter (16-31) és egy konstans érték (0-255) között.

ANDI Rd, K;      Rd  $\leftarrow$  Rd • K

#### 4.2.5.2.5 OR, EOR

Az OR utasítás logikai vagy kapcsolatot hajt végre két regiszter (0-31) között.

OR Rd, Rr;      Rd  $\leftarrow$  Rd v Rr

Az ORI utasítással logikai vagy kapcsolat valósítható meg egy regiszter (16-31) és egy konstans (0-255) között.

ORI Rd, K;      Rd  $\leftarrow$  Rd v K

Az EOR utasítás logikai kizáró-vagy kapcsolatot hajt végre két regiszter (0-31) között.

EOR Rd, Rr;      Rd  $\leftarrow$  Rd  $\oplus$  Rr

#### 4.2.5.2.6 Complement

A COM utasítással képezhetjük egy regiszter (0-31) egyes komplementensét. Gyakorlatilag a regiszter értékét kivonja a 0xFF értékből.

COM Rd;      Rd  $\leftarrow$  0xFF – Rd

A NEG utasítással képezhetjük egy regiszter (0-31) kettes komplemensét. Gyakorlatilag a regiszter értékét kivonja a 0x00 értékből. A 0x80 szám értéke a műveletvégzés után változatlan marad.

NEG Rd;       $Rd \leftarrow 0x00 - Rd$

#### 4.2.5.2.7 Set, clear, test

Az SBR utasítással egy regiszter (16-31) megadott bitjeit (0-255) állíthatjuk egyesbe. Az ORI utasításra vezethető vissza

SBR Rd, K;     $Rd \leftarrow Rd \vee K$

Az CBR utasítással egy regiszter (16-31) megadott bitjeit (0-255) törli, azaz nulla értéket kapnak. Az ANDI utasításra vezethető vissza.

CBR Rd, K;  $Rd \leftarrow Rd \bullet (0xFF - K)$

A TST utasítással tesztelhetjük, hogy a regiszter (0-31) nullával vagy negatív értékkel egyenlő-e. Gyakorlatilag a regiszter önmagával vett és kapcsolata. A regiszter értéke nem változik.

TST Rd;       $Rd \leftarrow Rd \bullet Rd$

Egy regiszert (0-31) törölni CLR utasítással tudunk. Az utasítás a regiszter kizáró vagy kapcsolatát veszi önmagával, ekkor a regiszter értéke nulla lesz.

CLR Rd;       $Rd \leftarrow Rd \oplus Rd$

A SER utasítás az összes bitet egyesbe állítja egy regiszterben (16-31). Az utasítás hatására a 0xFF érték töltődik a regiszterbe.

SER Rd;       $Rd \leftarrow 0xFF$

Példa logikai utasítások használatára:

```
ldi r16, 60          ;60 betöltése az r16 regiszterbe
ldi r18, 10          ;10 betöltése az r18 regiszterbe
and r16, r18          ;r16 és r18 logikai és kapcsolata
eor r16, r18          ;r16 és r18 logikai és kizáró-vagy kapcsolata
neg r16               ;r16 kettes komplemense
ser r18               ;r18 regiszter bitjeinek egyesbe állítása
```

#### 4.2.5.3 Vezérlést átadó utasítások

- Ugró utasítás
  - o Feltétel nélküli
  - o Feltételes
    - Branch if
    - Skip if

- Szubrutin hívás
- Komparáló utasítás

#### 4.2.5.3.1 Feltétel nélküli ugró utasítások

**Direkt ugrás:** ugrás a 8kbyte programmemória területen bármelyik címre.

JMP k; PC  $\leftarrow$  k

**Indirekt ugrás:** közvetett ugrás a Z indexregiszterben tárol 16 bites címre (0-128kbyte).

IJMP; PC  $\leftarrow$  Z

**Relatív ugrás:** relatív ugrás a PC - 2Kword +1 és a PC + 2Kword (2Kword=4kbyte) közötti címtartományra. Ha a programmemória nem haladja meg a 8kbyte-ot, akkor a teljes memória elérhető bármelyik címről.

RJMP k; PC  $\leftarrow$  PC + k + 1

#### 4.2.5.3.2 Feltételes ugró utasítások

Abban az esetben, ha valami feltételhez kötve kívánjuk az ugrást végezni, ha a feltétel nem teljesül, akkor a következő utasítások végrehajtása a cél, a feltételes ugró utasítások használatosak.

##### 4.2.5.3.2.1 Skip

Ezek az utasítások igaz feltétel esetén kihagyják a következő utasítás végrehajtását.

Az SBRC (Skip if Bit in Register Cleared) a következő utasítást figyelmen kívül hagyja, amennyiben a regiszter (0-31) megadott bitje (0-7) nulla értékű.

SBRC Rr, b if (Rr(b)=0) PC  $\leftarrow$  PC + 2 or 3

Az SBRS (Skip if Bit in Register is Set) utasítás egyes bitérték esetén teszi ugyan ezt.

SBRS Rr, b if (Rr(b)=1) PC  $\leftarrow$  PC + 2 or 3

Az SBIC (b Skip if Bit in I/O Register Cleared) és SBIS (Skip if Bit in I/O Register is Set) utasítások az SBRC és SBRS utasításokhoz hasonlóan működnek, de I/O regiszterekre (0-31) vonatkoztatva.

SBIC P, b if (P(b)=0) PC  $\leftarrow$  PC + 2 or 3

SBIS P, b if (P(b)=1) PC  $\leftarrow$  PC + 2 or 3

##### 4.2.5.3.2.2 Branch

A BRBS (Branch if Status Flag Set) és a BRBC (Branch if Status Flag Cleared) utasítások a SR egy bitjét vizsgálják, amennyiben igaz az állítás a (PC-63) - (PC-64) közötti megadott területre való ugrást hajtanak végre.

BRBS s, k if (SREG(s) = 1) then PC  $\leftarrow$  PC+k + 1

BRBC s, k      if (SREG(s) = 0) then  $PC \leftarrow PC + k + 1$

A BREQ (Branch if Equal) és a BRNE (Not Equal) utasítások az egyenlőséget vizsgálják. Gyakorlatilag a Z flag értékét figyelik. Összehasonlítást követően a Z flag értéke 1, ha az egyenlőség fennáll, 0, ha az egyenlőtlenség.

BREQ k            if ( $Z = 1$ ) then  $PC \leftarrow PC + k + 1$

BRNE k            if ( $Z = 0$ ) then  $PC \leftarrow PC + k + 1$

A BRSH (Branch if Same or Higher) és BRLO (Branch if Lower) utasítások előjele nélküli értékek összehasonlítását végezik. Az utasításokat CP, CPI, SUB vagy SUBI utasítások valamelyike előzi meg a sorban, a C flag értkét veszi figyelembe.

BRSH k            if ( $C = 0$ ) then  $PC \leftarrow PC + k + 1$

BRLO k            if ( $C = 1$ ) then  $PC \leftarrow PC + k + 1$

A BRGE és BRLT utasítások előjeles értékek összehasonlítását végezik. Az S flaget veszik figyelembe a vizsgálatkor, ami a N és a V flag kizáró-vagy kapcsolata.

BRGE k            if ( $N \oplus V = 0$ ) then  $PC \leftarrow PC + k + 1$

BRLT k            if ( $N \oplus V = 1$ ) then  $PC \leftarrow PC + k + 1$

A flag értékéhez közvetlenül kapcsolódóan is megtalálhatjuk a branch utasításokat, ezek: BRCS (Branch if Carry Set), BRCC (Branch if Carry Cleared), BRMI (Branch if Minus), BRPL (Branch if Plus), BRHS (Branch if Half Carry Flag Set), BRHC (Branch if Half Carry Flag Cleared), BRTS (Branch if T Flag Set), BRTC (Branch if T Flag Cleared), BRVS (Branch if Overflow Flag is Set), BRVC (Branch if Overflow Flag is Cleared).

BRCS k            if ( $C = 1$ ) then  $PC \leftarrow PC + k + 1$

BRCC k            if ( $C = 0$ ) then  $PC \leftarrow PC + k + 1$

BRMI k            if ( $N = 1$ ) then  $PC \leftarrow PC + k + 1$

BRPL k            if ( $N = 0$ ) then  $PC \leftarrow PC + k + 1$

BRHS k            if ( $H = 1$ ) then  $PC \leftarrow PC + k + 1$

BRHC k            if ( $H = 0$ ) then  $PC \leftarrow PC + k + 1$

BRTS k            if ( $T = 1$ ) then  $PC \leftarrow PC + k + 1$

BRTC k            if ( $T = 0$ ) then  $PC \leftarrow PC + k + 1$

BRVS k            if ( $V = 1$ ) then  $PC \leftarrow PC + k + 1$

BRVC k            is Cleared if ( $V = 0$ ) then  $PC \leftarrow PC + k + 1$

#### 4.2.5.3.3 Szubrutin hívó utasítások, feltétel nélküli szubrutin hívás

Direkt szubrutin hívás meghív egy szubrutint a program memórián belül. A visszatérési cím a verembe kerül betöltésre. A veremmutató értéke utólagosan eggyel csökken.

CALL k;      PC  $\leftarrow$  k

Indirekt szubrutin hívás: működése a CALL utasításhoz hasonló, de itt a Z indexregiszter kerül felhasználásra konstans érték helyett.

ICALL;      PC  $\leftarrow$  Z

Relatív szubrutin hívás: esetében a PC - 2Kword + 1 és PC + 2Kword címtartományban történhet a szubrutin hívása.

RCALL k;      PC  $\leftarrow$  PC + k + 1

Szubrutinból való visszatérés: a RET parancs hatására a visszatérési cím a veremből töltődik be. A veremmutató értéke előzetesen eggyel növekszik.

RET    PC  $\leftarrow$  STACK

#### 4.2.5.3.4 Komparáló utasírások

A CP (Compare) utasítás összehasonlít két regisztert (0-31), a regiszterek értékét nem befolyásolja. minden féltételes ugró utasítás alkalmazható a komparáló utasítások után.

CP Rd,Rr      Rd - Rr

A CPC (Compare with Carry) komparáló utasítás már a C falg értékét is figyelembe veszi.

CPC Rd,Rr      Rd - Rr - C

A CPI (Compare Register with Immediate) utasítás összehasonlítást végez egy regiszter (16-31) és egy konstans között (0-255).

CPI Rd,K      Rd - K

Példa feltételes utasítások használatára:

```
ldi r16, 60          ;60 betöltése az r16 regiszterbe
ldi r18, 10          ;10 betöltése az r18 regiszterbe

cp r16, r18          ;összehasonlítás
brne not_eq          ;ugrás a not_eq címkére, ha r16 nem egyenlő r18-val
brsh s_h              ;ugrás az s_h címkére, ha r16 nagyobb vagy egyenlő, mint r18

not_eq:               ;not_eq címke
ldi r16, 1
rjmp loop            ;relatív ugrás a loop címkére

s_h:
ldi r16, 2
rjmp loop
```

#### 4.2.5.4 Bit és bitteszt utasítások

##### 4.2.5.4.1 Bitállító utasítások

A bitállító utasításokkal tudjuk az egyes regiszterek bitjeit állítani vagy törölni. Nem csak általános felhasználású regiszterek esetében van erre lehetőség, hanem a SR bitjei esetében is. Az SBI (Set Bit in I/O Register) utasítással az I/O regiszterek (0-31) bitjei állíthatók.

SBI P,b;      I/O(P,b)  $\leftarrow$  1

A CBI (Clear Bit in I/O Register) utasítással pont az ellenkező eredmény érhető el, mint az SBI utasítással, a bitek nullába állíthatók.

CBI P,b;      I/O(P,b)  $\leftarrow$  0

A BSET (Flag Set) utasítással a SR megadott egyetlen bitje (flag) állítható 1-es értékbe. Itt egy utasítás végrehajtásával minden csak egy bitérték állítható.

BSET s;      SREG(s)  $\leftarrow$  1

A BCLR (Flag Clear) utasítással a SR egy bitje törlhető, azaz nulla értékbe állítható. Itt is minden csak egy flag-en hajtható végre a művelet.

BCLR s;      SREG(s)  $\leftarrow$  0

A BST (Bit Store from Register to T) utasítással egy regiszter (0-31) megadott bitjét (0-7) tárolhatjuk el a SR T flag-ében.

BST Rr, b      T  $\leftarrow$  Rr(b)

A BLD (Bit load from T to Register) utasítással a SR T flag értékét másolhatjuk egy regiszter (0-31) megadott bitjére (0-7).

BLD Rr, b      Rd(b)  $\leftarrow$  T

Az imént felsorolt utasításokon felül további olyan utasítások is találhatók a készletben, melyek a SR megadott bitjeit állítják vagy törlik. Ezek felépítése úgy alakul, hogy állítás esetén a SE előtaggal kezdődik az utasítás, törlés esetén a CL rövidítéssel, majd utána következik a flag betűjele:

SEC/N/Z/I/S/O/T/H

CLC/N/Z/I/S/O/T/H

Példa bitállító utasítások használatára:

```
LDI r16, 0xFF
BSET 6          ;Set T flag
BCLR 6          ;Reset T flag
BST r16, 1      ;T flag r16 regiszter 1. bitjének értékére állítása
```

#### 4.2.5.4.2 Bitléptető és bitforgató utasítások

##### 4.2.5.4.2.1 Logikai shift

Az LSL (Logical Shift Left) a regiszterben (0-31) található összes bit értékét egygyel balra tolja. A 0. bit értéke törlődik. A 7. bit értéke a Carry flag-be töltődik. Gyakorlatilag kettővel való szorzást valósít meg az utasítás.

LSL Rd;       $Rd(n+1) \leftarrow Rd(n)$ ,  $Rd(0) \leftarrow 0$

Az LSR (Logical Shift Right) a regiszterben (0-31) található összes bit értékét egygyel jobbra tolja. A 7. bit értéke törlődik. A 0. bit értéke a Carry flag-be töltődik. Gyakorlatilag kettővel való osztást valósít meg az utasítás.

LSR Rd;       $Rd(n) \leftarrow Rd(n+1)$ ,  $Rd(7) \leftarrow 0$

##### 4.2.5.4.2.2 Rotálás Carry-n keresztül

A ROL (Rotate Left Through Carry) a regiszterben (0-31) található minden bitet balra tolja eggyel. A Carry flag tolódik a 0. bit helyére, a 7 bit pedig a Carry flag-ére.

ROL Rd;       $Rd(0) \leftarrow C$ ,  $Rd(n+1) \leftarrow Rd(n)$ ,  $C \leftarrow Rd(7)$

A ROR (Rotate Right Through Carry) a regiszterben (0-31) található minden bitet jobbra tolja eggyel. A Carry flag tolódik a 7. bit helyére, a 0. bit pedig a Carry flag-ére.

ROR Rd;       $Rd(7) \leftarrow C$ ,  $Rd(n) \leftarrow Rd(n+1)$ ,  $C \leftarrow Rd(0)$

##### 4.2.5.4.2.3 Aritmetikai shift

Az ASR (Arithmetic Shift Right) esetében a regiszter (0-31) bitjei eggyel jobbra tolódnak, viszont a 7. bit értéke nem változik. A 0. bit a Carry flag helyére töltődik. A művelet előjeles szám osztása kettővel, anélkül, hogy az előjel megváltozna.

ASR Rd;       $Rd(n) \leftarrow Rd(n+1)$ ,  $n=0..6$

A SWAP (Swap Nibbles) utasítás a regiszter (0-31) alsó és felső 4 bitjének cseréjét végzi.

SWAP Rd;       $Rd(3..0) \leftarrow Rd(7..4)$ ,  $Rd(7..4) \leftarrow Rd(3..0)$

Példa bitléptető és bitforgató utasítások használatára:

```
LDI r16, 0b01110011
LSR r16          ;r16 bitjeinek eltolása jobbra: 00111001, C=1
ROR r16          ;r16 bitjeinek eltolása jobbra: 10011100
SWAP r16         ;r16 alsó és felső 4 bitjének felcserélése: 11001001
```

#### *4.2.5.5 Megszakítások*

A SEI (Global Interrupt Enable) utasítással engedélyezhetjük a globális megszakításokat. Az utasítás hatására a SR I flag értéke 1 lesz. A bit- és bitteszt utasítások csoportjába tartozik.

SEI

A RETI (Interrupt Return) utasítással térhetünk vissza egy megszakításból. A visszatérési cím a veremből töltődik be. Az utasítás végrehajtása előtt a veremmutató értéke eggyel nő. A vezérlést átadó utasítások csoportjába tartozik.

RETI

#### *4.2.5.6 Processzorvezérlő utasítások*

A NOP (No Operation) utasítás nem hajt végre semmilyen műveletet, a programszámláló értékét eggyel növeli.

NOP

A SLEEP utasítás hatására az áramkör alvó módba kerül, melyet a MCU kontrol regiszter határoz meg.

SLEEP

A WDR (Watchdog Reset) utasítás a Watchdog Timer-t reset-eli.

WDR

A BREAK utasítást az On-Chip Debug rendszer használja, és általában nem használatos az alkalmazásszoftverben. Bizonyos beállítások mellett a MCU a BREAK utasítást NOP-ként kezeli.

BREAK

## 5 8 bites Harvard architektúra

### 5.1 Megszakítások

#### 5.1.1 Fogalma, forrásai

- megszakítási esemény
- futó program felfüggesztése
- megszakítás kérés kiértékelése
- Engedélyezett megszakítás esetén:
  - Megszakítás Vektor Táblából a megszakítás kiszolgáló rutin címe betöltődik a IP-be (vagy PC-be), a visszatérési cím pedig a verembe
  - Megszakítási rutin lefutása után (reti-vel fejeződik be a rutin) IP-be visszatöltődik a veremből a visszatérési cím
- a felfüggesztett program futása folytatódik tovább

#### 5.1.2 Megszakítás vektortábla

Az Atmega128 megszakítás vektor táblája a programkód terület 0x0000 címétől kezdődik, és a mellékletben található táblázatban levő címeken találhatók az egyes kiszolgáló rutinok címe, 2 bájtonként 0x0045-ig.

#### 5.1.3 Atmega128 Timer megszakítások

A Timer-ek interrupt meghívásának 2 általános módja van AVR esetén:

- Overflow interrupt
- Compare Match interrupt

Overflow interrupt (Túlcordulás interrupt)

Akkor hívódik meg, ha a Timer elérte a maximális értékét és túlcordul. 8 bites timer esetén 255, 16 bites timer esetén 65535.

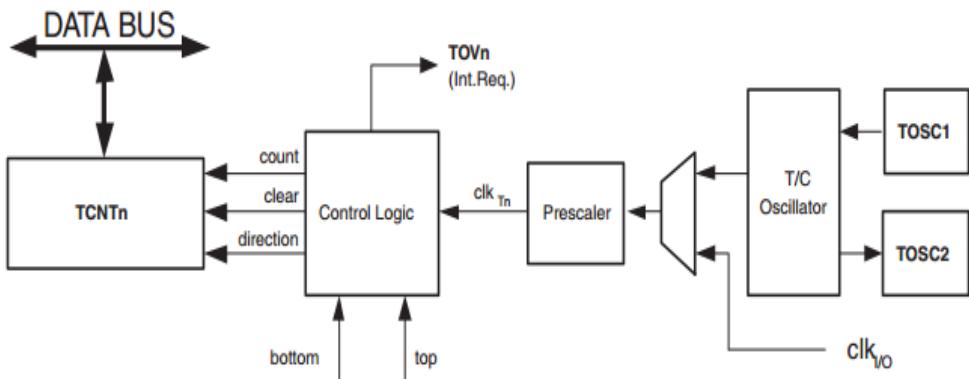
Compare match interrupt

Komparálási szint elérésekor hívódik meg az interrupt. A komparálási szintet az előzőekben bemutatott OCRn regiszter értékével lehet állítani.

## 5.2 Időzítők / Számlálók az ATmega128 -ban

### 5.2.1 Timer/Interruptok

Minden programnak szüksége van időzítésre, hogy a különböző utasításokat a megfelelő időben hajtsa végre. Erre a célra használjuk a timer-eket (időzítő). A timer az alap órajelét az AVR kvarc bemenetére (XTAL1-2) kötött kvarcról, vagy a belső oszcillátoráról kaphatja. Ez az órajelforrás biztosítja a timer lépései frekvenciát beállító logika bemenetét, amit ez az alábbi képen is látható.



10. ábra - Timer blokkvázlata

A Timer léptető logikája egy programozható frekvencia osztó áramkörön keresztül van összeköttetésben az órajel forrással, melyet majd a szoftver felhasználásával állíthatunk, és a Timer aktuális értéke a TCNT<sub>n</sub> regiszterben található, melynél az “n” a Timer számát jelenti.

A jegyzetben kettő időzítésről lesz szó, amelynél a Timer-t használjuk fel:

- **pontos időzítés**

Az időzített végrehajtást megszakítások felhasználásával végezzük el. Nézzük, hogy mi is az az megszakítás. Interrupt (megszakítás) olyan művelet sorozat, amely a program futását megszakítja, a megszakítás kiszolgáló függvényben lévő utasításokat végrehajtja, majd visszatér a megszakított program futtatásához.

- **időzített végrehajtás**

*A pontos időzítésekre és az időzített végrehajtásokra példa:*

```
Timer0Init();      //Timer beállítása
while(1)
{
    if(TCNT > 100)
    {      //ha a timer értéke nagyobb, mint 100
        LEDOn(); //kapcsolja be a LED-eket
    }
    else
    {
        LEDOff(); //kikapcsolja a LED-eket
    }
}
```

Fontos megemlíteni már a Timer-ek felhasználásánál a megszakítás fogalmát, mivel a Timer-ek is képesek megszakításokat generálni, és majd ezt a képességet felhasználva tudunk időzített végrehajtásokat elvégezni.

ATmega128 időzítői:

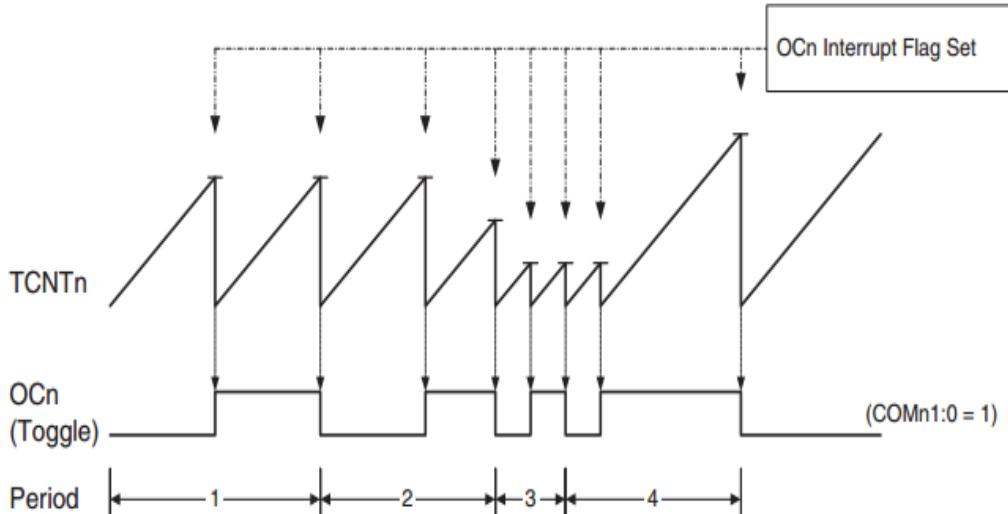
- 2 db 8 bites timer (Timer 0, Timer 2)
- 2 db 16 bites timer (Timer 1, Timer 3)

### 5.2.2 CTC mód (Clear Timer on Compare Match)

Mint fentebb említésre került, a Timer nem más, mint egy számláló, amelynek ismerjük a lépései frekvenciáját, és ezt kihasználva létre tudunk hozni nagyon pontos időzítéseket.

A mód fontossága, mint a nevéből is kiderül, törli a Timer tartalmát egy megadott érték elérésekor, melyet a szoftverből szabadon tudunk módosítani.

Egy kép a reprezentálásához az adatlapból:



11. ábra - CTC mód

Látható, hogy az időzítő értéke folyamatosan nő az idő teltével, amíg el nem éri a komparálási értéket, melyet az “ $OCR_n$ ” regiszterben állíthatunk be. Ekkor az értéke 0 lesz, és kezdi előlről a számolást.

#### 5.2.2.1 Timer 0 beállítása

A Timer0 egy 8 bites időzítő, ami az jelenti, hogy az időzítés maximális értéke 255 lehet. Fontos megjegyezni, hogy az időzítő csak akkor kezdi el a számolást, ha annak nullánál nagyobb értékét állítottuk be (pl.: egy órajel osztást).

**Példa 1: Készítsünk egy programot, mely 32ms-os periódussal növeli a LED-eken megjelenített számok értékét!**

```
1 #include <avr/io.h>
2
3 void PORT_init(void);
4 void Timer0_init(void);
5 void LED_out(unsigned char szam);
6
7 int main()
8 {
9     unsigned char szamolo = 0;
10    PORT_init();
11    Timer0_init();
12
13    while(1)
14    {
15        if(TCNT >= 249) //32 ms
16        {
17            szamolo++;
18            LED_out(szamolo);
19        }
20    }
21 }
22 void PORT_init()
23 {
24     DDRB = 0xF0;
25     DDRD = 0xF0;
26 }
27 void Timer0_init()
28 {
29     TCCR0 = (1<<CS02) //Órajelosztás [1024]
30             | (1<<CS01)
31             | (1<<CS00)
32             | (1<<WGM01) //CTC mód
33     OCR0 = 250; //Timer maximális értéke
34 }
35 void LED_out(unsigned char szam)
36 {
37     PORTD = szam & 0xF0;
38     PORTB = (szam << 4);
39 }
```

## Magyarázat:

Az első a Timer0Init() függvény (29.-33. sorok), amelyben a Timer0-t inicializáljuk.

Bit	7	6	5	4	3	2	1	0	TCCR0
Read/Write	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	
Initial Value	0	0	0	0	0	0	0	0	

Az adatlapban is megtalálható a TCCR0 regiszter belső felépítése, amely pontosan leírja, hogy az egyes bitek beállításának hatását.

Vizsgáljuk meg pontosabban azokat a biteket, amelyek fontosok a laboratóriumi gyakorlat elsajátításához:

- WGM00, WGM01 - ezekkel a bitekkel tudjuk beállítani, hogy az időzítő milyen üzemmódban szeretnénk üzemeltetni

Mode	WGM01 <sup>(1)</sup> (CTC0)	WGM00 <sup>(1)</sup> (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0 at	TOV0 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

Mint az adatlapban is látható, a WGM01 bit beállításával állíthatjuk be a Timer0-t CTC módba.

$$\text{TCCR0} = (1 << \text{WGM01});$$

- COM00, COM01 - Különböző módok további beállításai érhetők el vele. Esetünkben a CTC mód beállításai.

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Például a COM01 beállításával érhetjük el azt, hogy a OC0 nullázódjon a komparálási szint elérésekor, vagyis amikor eléri az OCR0 értékét.

$$\text{TCCR0} = (1 << \text{COM01});$$

- CS01, CS02, CS03 - A Timer belső frekvencia osztóját tudjuk állítani velük, vagyis a Timer lépései frekvenciáját

<b>CS02</b>	<b>CS01</b>	<b>CS00</b>	<b>Description</b>
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{TOS}}/(\text{No prescaling})$
0	1	0	$\text{clk}_{\text{TOS}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{TOS}}/32$ (From prescaler)
1	0	0	$\text{clk}_{\text{TOS}}/64$ (From prescaler)
1	0	1	$\text{clk}_{\text{TOS}}/128$ (From prescaler)
1	1	0	$\text{clk}_{\text{TOS}}/256$ (From prescaler)
1	1	1	$\text{clk}_{\text{TOS}}/1024$ (From prescaler)

A CS0n bitek beállításával jelen esetben egy 1024-es osztást állítottunk be a Timer0-nak, ami a következőt jelenti:

Tegyük fel, hogy az AVR 8MHz-ről működik. Akkor az 1024-es osztással 7812,5Hz-et kapunk, tehát a Timer lépései frekvenciáját kapjuk meg. Ezt időre átszámolva:

$$T = \frac{1}{f} = \frac{1}{7812,5\text{Hz}} = 0,000128 \text{ s} = 128\mu\text{s}$$

vagyis a timer 1 lépést 128μs alatt tesz meg. Ami 256 lépésre átszámolva, vagyis a maximális érték eléréséhez szükséges idő: 32,768 ms.

Jelen esetben nekünk csak 32ms-ra van szükségünk, ezért be kell állítani egy komparálási szintet az OCR0 regiszterben, ami 249. Ugyanis  $250 * 128\mu\text{s}$  az 32ms-al egyenlő. Ezen felül a CTC mód használata szükséges, mivel gondoljunk csak bele, hogy ha nem nulláznánk a timert a komparálási szinten, akkor hozzáadódna még  $6 * 128\mu\text{s}$  ami az az idő ameddig eléri a maximális értékét, ami 255.

### 5.2.3 Timer0 CTC mód mintakódok

A következő mintakód a Timer0 CTC módjára egy példa. A CTC mód, 1024-es előosztás kerül beállításra és a megszakítást engedélyezésre. A példaprogram megközelítőleg egy másodpercenként villogtatja a LED2-t (16MHz-es órajel mellett). A C nyelven megírt mintakód is hasonlóképen működik.

#### 5.2.3.1 AVR Assembly TIM0 CTC mintakód

```
;-----TIM0 CTC-----
INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    tmp    =      r16
.DEF    LED    =      r17
.DEF    szam   =      r18

.ORG 0x0
rjmp start                      ; jump to start

.ORG 0x1E                      ;$001E TIMER0 CTC Timer/Counter0 Compare Match
rjmp TIM0_CTC_IT

.ORG 0x100
;-----;
; MACRO
;-----;
.macro STACK_init
ldi    tmp, HIGH(RAMEND)
out   SPH, tmp
ldi    tmp, LOW(RAMEND)
out   SPL, tmp
.endmacro

.macro LED_init
ldi    tmp, 0xF0
out   DDRB, tmp
out   DDRD, tmp
.endmacro

.macro TIM0_CTC_init            ;TIM0 CTC init
ldi    tmp, 0b00001111          ;1024-es előosztás beállítása

out   TCCR0, tmp    ;Timer/Counter Control Register - TCCR0
;7   6   5   4   3   2   1   0
;FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00
ldi    tmp, 0x02          ;Timer/Counter0 Overflow Interrupt Enable

out   TIMSK, tmp    ;Timer/Counter Interrupt Mask Register - TIMSK
;7   6   5   4   3   2   1   0
;                           OCIE0 TOIE0
ldi    tmp, 249          ;16000000/1024/250 8000000/1024/250 32ms 16ms
out   OCR0, tmp
sei   ;Sets the Global Interrupt flag (I) in SREG (status register).
.endmacro
```

```

;-----
; SUBROUTINE
;-----
LED_out:
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
ret
;-----
; INTERRUPT
;-----
;-----  

TIM0_CTC_IT:          ;TIM0 CTC megszakítást kiszolgáló rutin
    inc szam           ;szam növelése
reti
;-----  

; PROGRAM
;-----  

start:  

;-----  

; INIT
;-----  

STACK_init
LED_init
TIM0_CTC_init
;-----  

; PROGRAM
;-----  

ldi    LED, 0x04          ;LED2
eor    szam, szam         ;szam 0 kezdőérték
ldi    tmp, 0x04

loop:
    cpi    szam,62          ;16000000/1024/250/62      ~1s
    breq   kitesz
    brne   loop

kitesz:
    eor    szam, szam         ;szam törlése
    call   LED_out            ;LED kitétele a LED-ekre
    eor    LED, tmp            ;"villgotatás"

njmp  loop

```

### 5.2.3.2 AVR C TIM0 CTC mintakód

```
//-----TIM0 CTC-----
#include <avr/io.h>
#include <avr/interrupt.h>

void PORT_init(void);
void Timer0_init(void);
unsigned char szamlalo=0;

int main()
{
    PORT_init();
    Timer0_init();
    PORTB=0x40;
    while(1)
    {
        }
    }

void PORT_init()
{
    DDRB = 0xF0;
    DDRD = 0xF0;
}
void Timer0_init()
{
    TCCR0 = (1<<CS02) | (1<<CS01) | (1<<CS00) | (1<<WGM01);
                //CTC, órajelosztás [1024]
    TIMSK|=(1<<OCIE0);
    OCR0 = 249;           //Timer maximális értéke - 8bites max 255
    sei();
}

ISR(TIMER0_COMP_vect)
{
    szamlalo++;
    if (szamlalo>=62)
    {
        PORTB^=0x40;
        szamlalo=0;
    }
}
```

## 5.2.4 Timer0 OVF mintakódok

A példaprogram a Timer0 OVF megszakításának beállítását demonstrálja. Előosztásnak 1024 kerül beállításra, ezen felül csak az overflow megszakítást kell engedélyezni. A mintakód a LED0-t villogtatja megközelítőleg 16ms-onként. Ez az érték 16MHz órajel mellett igaz, 8MHz-es órajel esetén ez megközelítőleg 32ms. A C nyelven megírt mintaprogram működése hasonló.

### 5.2.4.1 AVR Assembly TIM0 OVF mintakód

```
;-----TIM0 OVF-----  
.INCLUDE "m128def.inc" ; Include definitions Atmega128  
.DSEG  
;  
.CSEG  
.DEF tmp = r16  
.DEF LED = r17  
.DEF szam = r18  
  
.ORG 0x0  
rjmp start ; jump to start  
  
.ORG 0x20 ;$0020 TIMER0 OVF Timer/Counter0 Overflow  
rjmp TIM0_OVF_IT  
.ORG 0x100  
;  
;  
; MACRO  
;  
.macro STACK_init  
ldi tmp, HIGH(RAMEND)  
out SPH, tmp  
ldi tmp, LOW(RAMEND)  
out SPL, tmp  
.endmacro  
  
.macro LED_init  
ldi tmp, 0xF0  
out DDRB, tmp  
out DDRD, tmp  
.endmacro  
  
.macro TIM0_OVF_init ;TIM0 OVF init  
ldi tmp, 0b00000111 ;1024-es előosztás beállítása  
  
out TCCR0, tmp ;Timer/Counter Control Register - TCCR0  
;7 6 5 4 3 2 1 0  
;FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00  
ldi tmp, 0x01 ;Timer/Counter0 Overflow Interrupt Enable  
  
out TIMSK, tmp ;Timer/Counter Interrupt Mask Register - TIMSK  
;7 6 5 4 3 2 1 0  
; OCIE0 TOIE0  
sei ;Sets the Global Interrupt flag (I) in SREG (status register).  
.endmacro
```

```

;-----
; SUBROUTINE
;-----
LED_out:
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
ret
;-----
; INTERRUPT
;-----
TIM0_OVF_IT:           ;TIM0 OVF megszakítást kiszolgáló rutin
    call   LED_out          ;LED villogtatás
    eor    LED, tmp
reti
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
STACK_init
LED_init
TIM0_OVF_init
;-----
; PROGRAM
;-----
ldi     LED, 0x01
ldi     tmp, 0x01

loop:
    rjmp   loop

```

#### 5.2.4.2 AVR C TIM0 OVF mintakód

```

//-----TIM0 OVF-----
#include <avr/io.h>
#include <avr/interrupt.h>

void Timer0Init() {
    TCCR0  =  (1<<CS02) | (1<<CS01) | (1<<CS00); //1024-es osztás
    TIMSK |=  (1<< TOIE0);                      //OVF interrupt enable
    sei();                                         //Globális interruptok engedélyezése
}

int main() {
    DDRB = 0xF0;
    Timer0Init(); //Timer beállítása

    while(1) {

    }

    ISR(TIMER0_OVF_vect) {                      //Timer0 overflow interrupt
        PORTB^=0x10;
    }
}

```

## 5.2.5 Timer1 beállítása

**Példa 2:** Készítsünk egy programot, amely 800 ms -os periódussal meghív egy függvényt, amely növeli a LED-eken megjelenített szám értékét.

Komplexebb időzítési feladatok elvégzéséhez már interruptokat használunk.

A második fontos dolog, egy ilyen feladat megtervezésénél, a timer kiválasztása. Ugyan is ha 800ms-os végrehajtást szeretnénk létrehozni ahoz már a 8 bites timer nem elég. Nézzük is meg egy egyszerű számítással levezetve:

- **8 bites timer (8MHz-es órajel)**

A timer maximális előosztója 1024, és a maximális értéke 255.

$$\frac{1}{8000000/1024} * 256 = 0,032768 \text{ s} \Rightarrow 32,768 \text{ ms}$$

Tehát látható, hogy egy 8 bites Timer felhasználásával maximum 32,768ms-os interrupt meghívást tudunk létrehozni.

- **16 bites timer (8MHz-es órajel)**

A timer maximális előosztója itt is 1024 és a maximális értéke 65535.

$$\frac{1}{8000000/1024} * 65536 = 8,3886 \text{ s}$$

Látható, hogy ezzel a timerrel már több mint 8 másodperces interrupt meghívásokat is létre tudunk hozni.

Következzen a feladat megoldása Timer 1 felhasználásával:

```
#include <avr/io.h>
#include <avr/interrupt.h>

void LED_out(unsigned char szam) {
    PORTD = szam & 0xF0;
    PORTB = (szam << 4);
}

void Timer1Init() {

    TCCR1B = (1<<WGM12) | (1<<CS12) | (1<<CS10);      //ctc          //1024
    TCCR1C = 0;
    OCR1A = 6249;                                         // max 65535 (16bit-es)
    //8000000/1024/6250 -->800ms

    TIMSK |= (1<<OCIE1A);                                //engedélyezések
    sei();
}

unsigned char szamlalo = 0;

int main() {
    DDRD = 0xF0; //LEDek beállítása
    DDRB = 0xF0;

    Timer1Init(); //Timer beállítása

    while(1) {
        LED_out(szamlalo); //szamlalo értéke a LED-en
    }
}

ISR(TIMER1_COMPA_vect) { //Timer1 komparálási interrupt
    szamlalo++;
}
```

### Magyarázat:

Először is pár szó a 16 bites Timer-ről. Alapjában véve nagyon hasonlít a 8 bites Timer-re, annyi különbséggel, hogy több funkcióval rendelkezik és a beállításait több regiszterre bontották szét.

Az “avr/io.h” header fájl mellett most már használnunk kell az “avr/interrupt.h” header fájlt is, ami az AVR mikrokontroller esetében az interrupt(megszakítások) kezelését tartalmazza.

Bit	7	6	5	4	3	2	1	0	TCCR1A
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

A TCCR1A regiszter felépítése a következő:

- COM1x1, COM1x0 : Mint ahogy a 8 bites esetén a timer különböző funkcióit állíthatjuk be velük
- WGM10, WGM11 : A timer futási módjait állíthatjuk be vele, mint például az előzőekben ismertetett CTC módot.

COMnA1/COMnB1/ COMnC1	COMnA0/COMnB0/ COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	Toggle OCnA/OCnB/OCnC on compare match.
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level).
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level).

Nézzük a kiszínezett sort, mely beállításához a COM1A1-es regisztert kell 1-be állítani a TCCR1A-ban. Ezzel azt érjük el, hogy ha a timer elérte a megadott komparálási szintet, akkor OCn nullázódik. Ebben az esetben az OC1A funkciójú pinen (PB5) meg is jelenik az aktuális állapota. Ez a mi esetünkben a LED1.

Bit	7	6	5	4	3	2	1	0	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

A TCCR1B regiszter felépítése a következő:

- WGM12, WGM13: Ezekkel és a TCCR1A regiszterben lévő bitekkel tudjuk beállítani a timert különböző funkciókba.
- CS10, CS11, CS12: Ezekkel a bitekkel tudjuk beállítani a timer előosztását, melye a timer0 beállításánál részletesebben elmagyarázásra került.

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation <sup>(1)</sup>	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	-	-	-
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Mint a táblázatban látható, a timer CTC módjának a beállításához a WGM11-t kell beállítani, hogy a komparálási szintje az OCR1A legyen.

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{IO}}/1$ (No prescaling)
0	1	0	$\text{clk}_{\text{IO}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{IO}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{IO}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{IO}}/1024$ (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

A timer előosztását 1024-re szeretnénk beállítani, ezért a CS10 és a CS12 bitet kell beállítani a TCCR1B regiszterben.

### Nézzünk is rá egy példát:

A mikrokontrollerre kötött órajelforrás az 8MHz.

Tehát az  $F_{\text{CPU}} = 8000000$

Az előosztás: 1024

A Timer lépései ideje:

$$\frac{8\ 000\ 000 \text{ Hz}}{1024} = 7812,5 \text{ Hz}$$

$$\frac{1}{7812,5 \text{ Hz}} = 0,000128 \text{ s} = 128 \mu\text{s}$$

Tehát  $128 \mu\text{s}$  szükséges ahhoz, hogy a timer lépjön egyet.

Ahhoz, hogy a feladatban megfogalmazott 800 ms-os időzítést létre tudjuk hozni ki kell számolni az OCR1A értékét, amely a komparálási szint.

$$\frac{800 \text{ ms}}{128 \mu\text{s}} = 6250$$

Bit	7	6	5	4	3	2	1	0	
ReadWrite	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Initial Value	0	0	0	0	0	0	0	0	

Most már csak azt kell beállítani, hogy ha a timer értéke elérte a komparálási szintet, akkor generáljon egy interruptot. Ezt a TIMSK regiszterben állíthatjuk be.

Számunkra most csak az OCIE1A és a TOIE1 bit magyarázata a fontos.

Általában a Timerek kettő fajta interruptot tudnak generálni:

- Overflow interrupt: amikor a timer elérte a maximális értékét és úm. túlcordul
- Compare match interrupt: amikor beállítunk egy komparálási értéket, és amikor ezt eléri a timer, akkor generál egy interruptot

Fontos, hogy az interruptok megnevezéseit az “iom128.h” file-ban találhatjuk, melyet az AVR Studio egy fordítás után automatikusan legenerál a “Dependencies” mappába.

*Egyszerre tud Overflow és Compare match interruptot is generálni.*

## 5.2.6 Timer1 CTC mód mintakódok

A következő két mintakód (ASM és C) a Timer1 komparálási megszakításának beállítását mutatja be. Előosztásnak 1024 kerül beállításnak, komparálási értéknek 15624, azaz egy másodperces időzítés. A megszakítást kiszolgáló rutinban a LED3 kerül villogtatásra. A beállított értékek itt is 16MHz-es órajelet feltételeznek.

### 5.2.6.1 AVR Assembly TIM1 CTC mintakód

```
;-----TIM1 CTC COMPA-----
INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    tmp      =      r16
.DEF    LED      =      r17

.ORG 0x0
    njmp start           ; jump to start
.ORG 0x18      ;$0018 TIMER1 COMPA Timer/Counter1 Compare Match A
    njmp TIM1_COMPA
.ORG 0x100
;-----
; MACRO
;-----
.macro TIM1_CTC_init
;TCCR1B
;ICNC1 ICES1 - WGM13 WGM12 CS12 CS11 CS10
;0     0     0 0   1   1   0   1
    ldi    tmp, 0b00001101 ;CTC mód 1024 előosztás
    out   TCCR1B, tmp

    ldi    tmp, 0x3D       ;ELŐSZÖR OCR1AH!!!!!!
    out   OCR1AH, tmp
    ldi    tmp, 0x08       ;15625-1      max 65535
    out   OCR1AL, tmp
;TIMSK
;OCIE2 TOIE2 TICIE1 OCIE1A OCIE1B TOIE1 OCIE0 TOIE0
;0     0     0     1     0     0     0     0
    ldi    tmp, 0b00010000
    out   TIMSK, tmp
    sei
.endmacro

.macro STACK_init
    ldi    tmp, HIGH(RAMEND)
    out   SPH, tmp
    ldi    tmp, LOW(RAMEND)
    out   SPL, tmp
.endmacro

.macro LED_init
    ldi    tmp, 0xF0
    out   DDRB, tmp
    out   DDRD, tmp
.endmacro
```

```

;-----
; SUBROUTINE
;-----
LED_out:
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
ret
;-----
; INTERRUPT
;-----
TIM1_COMPA:
call    LED_out          ;LED villogtatása
eor    LED, tmp
reti
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
STACK_init
TIM1_CTC_init
LED_init
;-----
; PROGRAM
;-----
ldi LED, 0x08            ;LED3
ldi tmp, 0x08
loop:
    rjmp   loop

```

### 5.2.6.2 AVR C TIM1 CTC mintakód

```

//-----TIM1 CTC-----
#include <avr/io.h>
#include <avr/interrupt.h>

void Timer1Init() {
    TCCR1B = (1<<WGM12) | (1<<CS12) | (1<<CS10);      //ctc      //1024
    TCCR1C = 0;
    OCR1A = 15624;           // max 65535 (16bit-es)
    TIMSK |= (1<<OCIE1A); //engedélyezés
    sei();
}
int main() {
    //LEDEk beállítása
    DDRB = 0xF0;
    PORTB=0x80;
    Timer1Init(); //Timer beállítása
    while(1) {

    }
}
ISR(TIMER1_COMPA_vect) { //Timer1 komparálási interrupt
    PORTB^=0x80;
}
```

## 5.2.7 Timer1 OVF megszakítás mintakódok

A példaprogram a Timer1 overflow megszakításának beállítását mutatja be. Itt is 1024-es előosztást állítottunk, ezen kívül csak a OVF megszakítás kerül engedélyezésre. Ezen beállítások mellett villogtatjuk a LED0-t, amely több mint 4 másodpercenként fog állapotot váltani 16MHz-es órajel mellett. A C nyelven elkészített kód működése az előzőkben ismertetettek alapján történik.

### 5.2.7.1 AVR Assembly TIM1 OVF mintakód

```
;-----TIM1 OVF-----  
.INCLUDE "m128def.inc" ; Include definitions Atmega128  
.DSEG  
;  
.CSEG  
.DEF tmp = r16  
.DEF LED = r17  
  
.ORG 0x0  
rjmp start ; jump to start  
.ORG 0x1C ;$001C TIMER1 OVF Timer/Counter1 Overflow  
rjmp TIM1_OVF  
.ORG 0x100  
;-----  
; MACRO  
;-----  
.macro TIM1_OVF_init ;TCCR1B  
;ICNC1 ICES1 - WGM13 WGM12 CS12 CS11 CS10  
;0 0 0 0 0 1 0 1  
ldi tmp, 0b00000101 ;CTC mód 1024 előosztás  
out TCCR1B, tmp  
;TIMSK  
;OCIE2 TOIE2 TICIE1 OCIE1A OCIE1B TOIE1 OCIE0 TOIE0  
;0 0 0 0 0 1 0 0  
ldi tmp, 0b00000100  
out TIMSK, tmp  
sei  
.endmacro  
  
.macro STACK_init  
ldi tmp, HIGH(RAMEND)  
out SPH, tmp  
ldi tmp, LOW(RAMEND)  
out SPL, tmp  
.endmacro  
  
.macro LED_init  
ldi tmp, 0xF0  
out DDRB, tmp  
out DDRD, tmp  
.endmacro
```

```

; -----
; SUBROUTINE
; -----
LED_out:
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
ret
; -----
; INTERRUPT
; -----
TIM1_OVF:
call    LED_out          ; LED villogtatása
eor    LED, tmp
reti
; -----
; PROGRAM
; -----
start:
; -----
; INIT
; -----
STACK_init
TIM1_OVF_init
LED_init
; -----
; PROGRAM
; -----
ldi LED, 0x01            ; LED0
ldi tmp, 0x01
loop:
    rjmp   loop

```

#### 5.2.7.2 AVR C TIM1 OVF mintakód

```

//-----TIM1 OVF-----
#include <avr/io.h>
#include <avr/interrupt.h>

void Timer1Init() {
    TCCR1B = (1<<CS12)| (1<<CS10);      //1024 előosztás
    TIMSK |= (1<<TOIE1);                  //OVF engedélyezés
    sei();
}

unsigned char szamlalo = 0;
int main() {
    DDRB = 0xF0;
    PORTB = 0x10;
    Timer1Init(); //Timer beállítása
    while(1) {

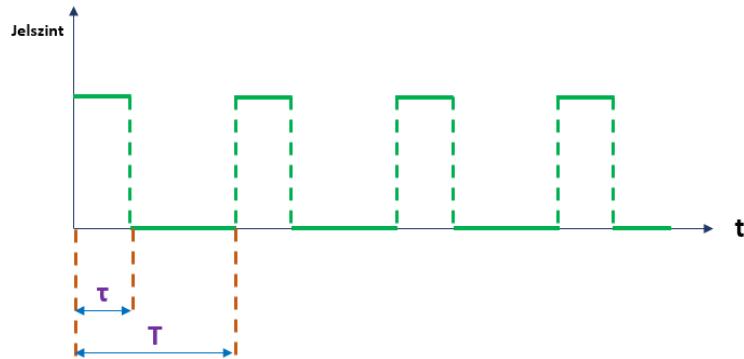
    }
}

ISR(TIMER1_OVF_vect) { //Timer1 overflow interrupt
    PORTB ^= 0x10;
}

```

### 5.3 PWM

A PWM az angol Pulse Width Modulation rövidítése, impulzusszélesség-modulációt jelent.

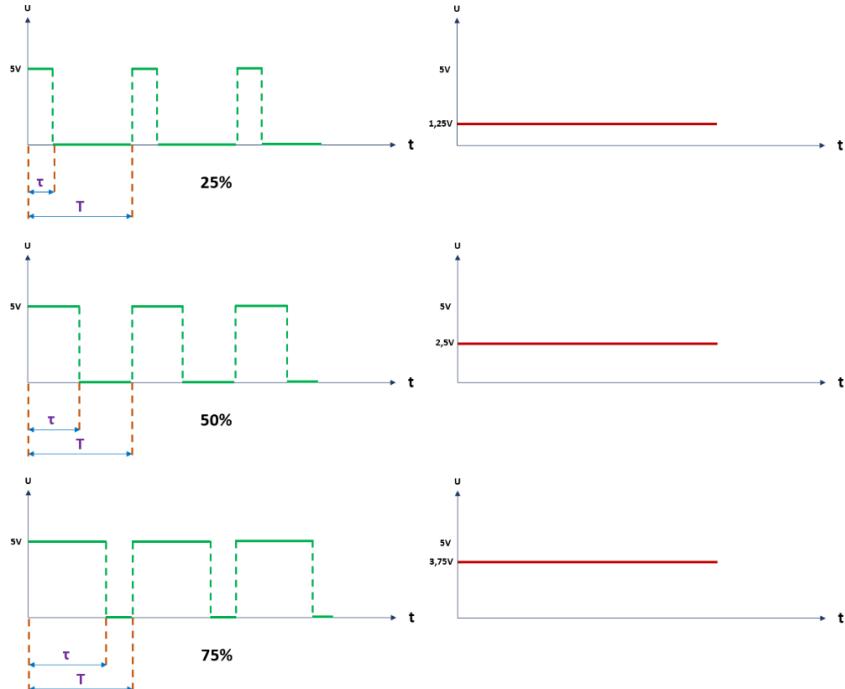


12. ábra – Négyszögjel

Az ábrán látható négyzetgyel periódus idejét a  $T$  jelöli. A  $\tau$  azt az időt jelöli, amennyi ideig magas a jelszint egy perióduson belül. Ez az idő 0 és  $T$  között alakulhat. A  $\tau/T$  hányadost kitöltési tényezőnek nevezik, mely 0-1 között vehet fel értékeket. A magas jelszintet impulzusnak hívjuk.

Négyzetgyel előállítása során az impulzusok hosszúsága információt hordozhat. A kitöltési tényező változtatásával megvalósíthatunk különböző vezérléseket.

A következő ábrán látható a különböző kitöltési tényezőjű négyzetgyelek alakulása.



13. ábra - Négyzetgyelek különböző kitöltéssel

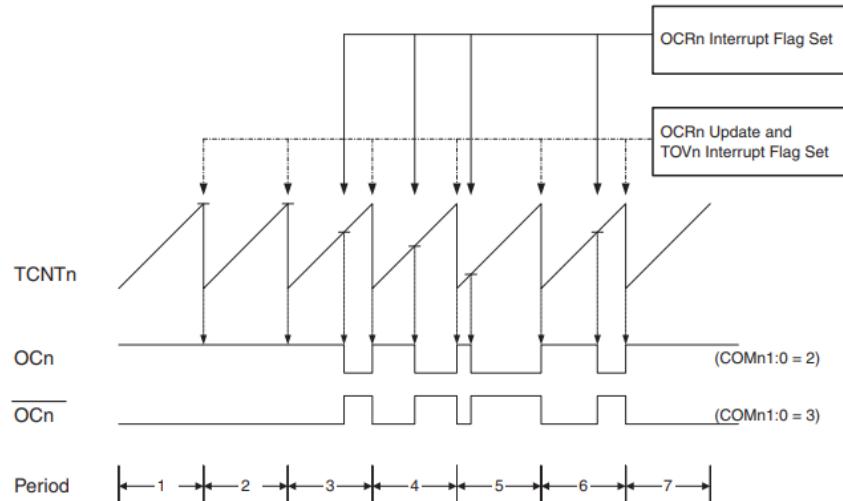
Az ATmega128 mikrokontroller több PWM módot is kínál, nézzük ezeket sorra.

### 5.3.1 Timer/Counter0 PWM

Timer0 esetében a következő PWM módokat különböztethetünk meg:

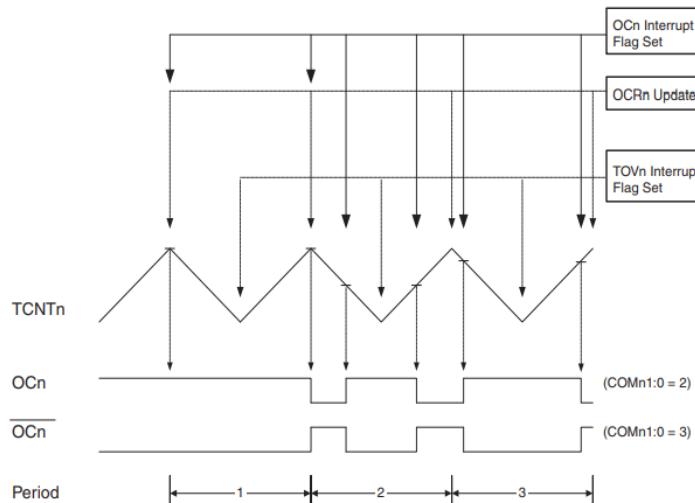
- Fast PWM
- Phase Correct PWM

A kettő között a különbség, hogy a Fast PWM-mel előállíthatók kétszer olyan gyors frekvenciák is. A timer számláló az alsó értéktől számol a maximális értékig, majd ismét az alsótól növekszik. Ez az egyszeres meredekségű működés. A PWM frekvenciát az órajelet az előosztással és 256-tal elosztva kapjuk meg. A kitöltési tényező mértékét a komparálási érték határozza meg. Az ábrán a FastPWM mód idődiagrammja látható.



**14. ábra - Timer0 Fast PWM mód**

A Phase Correct PWM alapja a kettős meredekségű működés. A timer számláló az alsó értéktől számol a maximum értékig, majd vissza az alsóig. A PWM frekvenciát az órajelet az előosztással és 510-zel elosztva kapjuk meg. Az ábrán a Phase Correct PWM mód idődiagramja látható:



**15. ábra - Timer0 Phase Correct PWM mód**

Az adatlapban található táblázat alapján a WGM01, WGM00 : 0, 1 beállításával választhatjuk ki a Phase Correct PWM módot, az 1, 1 beállítással pedig a Fast PWM módot, melyet a következő táblázat mutat.

<b>Mode</b>	<b>WGM01 (CTC0)</b>	<b>WGM00 (PWM0)</b>	<b>Timer/Counter Mode of Operation</b>	<b>TOP</b>	<b>Update of OCR0 at</b>	<b>TOV0 Flag Set on</b>
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

### 5.3.2 Timer0 PWM mintakódok

A következő mintakód a Timer0 Phase Correct PWM módjának beállítását mutatja be. Itt fontos, hogy ne válasszunk túl nagy előosztást, mert LED-ek esetében alkalmazva nem lesz látványos a végeredmény. Általában a 64-es előosztás már jó választás. PWM esetében nem kell megszakítást engedélyeznünk, nincs rá szükség.

A COM01 regiszter bitet kell még 1-esbe állítani, ezzel a beállítással érjük el azt, hogy a LED0 fényerejét tudjuk változtatni. A LED0 a PORTB4-en helyezkedik el, ennek a lábnak az alternatív funkciója a OC0.

A COM01 állításával a timer számláló felfelé számlálásakor a komparálási értéket elérve az OC0 0 lesz, lefelé számláláskor a komparálási értéket elérve pedig 1. A kitöltést az OCR0 regiszterrel tudjuk állítani.

#### 5.3.2.1 AVR Assembly TIM0 PWM mintakód

```
;-----TIM0 PWM-----
.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF tmp      = r16
.DEF LED      = r17
.DEF PWM_d   = r18
.ORG 0x0
rjmp start          ; jump to start

.ORG 0x100
;-----
; MACRO
;-----
.macro TIM0_PWM_init
    ldi tmp, 0b01100100          ;TIM0 Phase Correct PWM Mode init
                                    ;64-es előosztás beállítása
                                    ;Phase Correct PWM Mode kiválasztás,
                                    ;Clear OC0 on Compare Match
                                    ;when up-counting.
                                    ;Set OC0 on Compare Match
                                    ;when downcounting.
    out TCCR0, tmp             ;Timer/Counter Control Register - TCCR0
                                ;7 6 5 4 3 2 1 0
                                ;FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00
                                ;0 1 1 0 0 1 0 0
.endmacro

.macro STACK_init
    ldi tmp, HIGH(RAMEND)
    out SPH, tmp
    ldi tmp, LOW(RAMEND)
    out SPL, tmp
.endmacro
```

```

.macro LED_init
    ldi    tmp, 0xF0
    out    DDRB, tmp
    out    DDRD, tmp
.endmacro
;-----
; SUBROUTINE
;-----
PWM_duty:
    mov    tmp, PWM_d                      ;kitöltési tényező állítása
    out    OCR0, tmp                        ;0-255
ret
;-----
; INTERRUPT
;-----
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
STACK_init
LED_init
TIM0_PWM_init
;-----
; PROGRAM
;-----
ldi          PWM_d, 10                  ;kitöltési tényező állítása
call    PWM_duty
loop:
    rjmp   loop

```

### 5.3.2.2 AVR C TIM0 PWM mintakód

```

/*-----TIM0 PWM-----*/
#include <avr/io.h>

int PWM_duty0= 5;                                //kitöltés állítása
void Timer0Init() {
    TCCR0 = (1<<CS02) | (0<<CS01) | (0<<CS00) | (1 << WGM00);      //64-es osztás /PWM, Phase Correct
    TCCR0 |= (1 << COM01);           //Clear OC0 on Compare Match
    OCR0 = PWM_duty0;                //0-255               PORTB4 -> LED0
}

int main() {
    DDRB = 0xF0;                     //LED-ek beállítása
    Timer0Init();                   //Timer beállítása
    PORTB|=0x20;                    //LED1 bekacsolása

    while(1) {

}
}

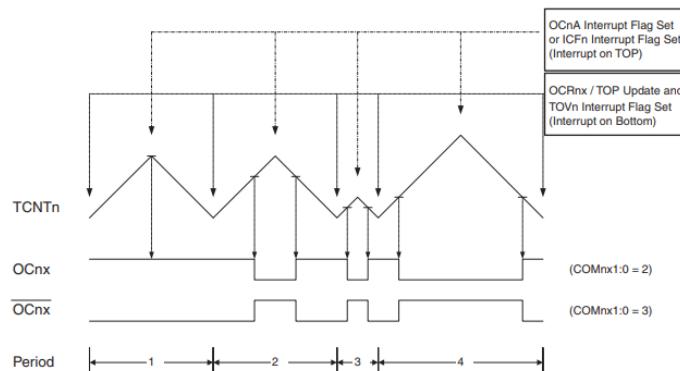
```

### 5.3.3 Timer/Counter1 PWM

Timer1 esetében a következő PWM módokat különböztethetjük meg:

- Fast PWM
  - o 8bit, 9bit, 10bit, 16bit
- Phase Correct PWM
  - o 8bit, 9bit, 10bit, 16bit
- Phase and Frequency Correct PWM

Működésük hasonló a Timer0 PWM azonos módjaihoz. Fast PWM módban a PWM frekvencia az órajel az előosztással és a beállított maximális értékkel plusz egyelőre elosztva számolható. Phase Correct PWM mód esetén a PWM frekvencia az órajel elosztva az előosztással és a beállított tom érték kétszeresével, ez igaz a Phase and Frequency Correct PWM módra is. Az ábrán a Phase and Frequency Correct PWM mód idődiagramja látható.



**16. ábra - Timer1 Phase and Frequency Correct PWM mód**

A következő táblázat mutatja a különböző PWM módok kiválasztásának lehetőségét. A táblázat megmutatja többek között a komparálási szinteket is.

Mode	WGMr3	WGMr2 (CTCn)	WGMr1 (PWMr1)	WGMr0 (PWMr0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	—	—	—
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

### 5.3.4 Timer1 PWM mintakódok

A Timer1 PWM-ek esetében hasonló a dolgunk van a Timer0-hoz, viszont itt több a beállítási lehetőség.

A példaprogram a Timer1 Fast PWM módjának beállítását mutatja be. Az előosztást itt is 64-re választottuk, a komparálási értéket az egyszerűség kedvéért 250-re (ICR1-ben, ez lesz a számláló maximális értéke). Ennek két oka van, az egyik, hogy 8bites értéket Assembly-ben könnyebb beállítani, illetve a kitöltési tényező számolása egy 0-250-es tartományban egyszerűbb.

Az OC1A, OC1B, OC1C sorra a LED1-3-ra csatlakoznak, tehát a Timer1-gyel 2 LED-re tudunk hardveres PWM-et megvalósítani. A kimenetek értéke, például OC1A esetén 1 lesz az OCR1A-ban beállított komparálási értékig, majd utána nulla, a számláló alsó értékének elérésekor ismét 1. Ez igaz a B-re és C-re is.

A mintakódban különböző kitöltések kerültek beállításra.

#### 5.3.4.1 AVR Assembly TIM1 PWM mintakód

```
; -----TIM1 PWM-----
.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF tmp    =      r16
.DEF LED    =      r17
.ORG 0x0
njmp start           ; jump to start
.ORG 0x100
;
; MACRO
;
.macro TIM1_PWM_init
;TCCR1A
;COM1A1 COM1A0 COM1B1 COM1B0 COM1C1 COM1C0 WGM11 WGM10
;1 0 1 0 1 0 1 0
ldi tmp,0b10101010
out TCCR1A, tmp
;TCCR1B
;ICNC1 ICES1 - WGM13 WGM12 CS12 CS11 CS10
;0 0 0 1 1 0 1 1
ldi tmp, 0b000011011       ;64 előosztás, Fast PWM
out TCCR1B, tmp
ldi tmp, 0                 ;Komparálási érték, 250
out ICR1H, tmp
ldi tmp, 0xFA
out ICR1L, tmp
ldi tmp, 0
out OCR1AH, tmp
ldi tmp, 0
out OCR1BH, tmp
```

```

ldi    tmp, 0
sts    OCR1CH, tmp
;Kitöltés állítása
ldi    tmp, 1
out    OCR1AL, tmp
ldi    tmp, 10
out    OCR1BL, tmp
ldi    tmp, 100
sts    OCR1CL, tmp
.endmacro

.macro STACK_init
ldi    tmp, HIGH(RAMEND)
out    SPH, tmp
ldi    tmp, LOW(RAMEND)
out    SPL, tmp
.endmacro

.macro LED_init
ldi    tmp, 0xF0
out    DDRB, tmp
out    DDRD, tmp
.endmacro

;-----
; SUBRUTINE
;-----
LED_out:
out    PORTD, LED
swap   LED
out    PORTB, LED
swap   LED
ret
;-----
; INTERRUPT
;-----
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
STACK_init
TIM1_PWM_init
LED_init
;-----
; PROGRAM
;-----


loop:
    rjmp   loop

```

### 5.3.4.2 AVR C TIM1 PWM mintakód

```
/*-----TIM1 PWM-----*/
#include <avr/io.h>
#include <avr/interrupt.h>

int PWM_duty0=1;
int PWM_duty1=10;
int PWM_duty2=100;

void Timer1Init() {
    TCCR1A = (0<<COM1A0) | (1<<COM1A1) | //clear
    (0<<COM1B0) | (1<<COM1B1) |
    (0<<COM1C0) | (1<<COM1C1) |
    (0<<WGM10) | (1<<WGM11); //fast pwm ICRn

    TCCR1B = (0 << ICNC1) | (0 << ICES1) |
    (1 << WGM13) | (1 << WGM12) |
    (1<<CS11) | (1<<CS00); //F_CPU/64

    TCCR1C = 0;
    ICR1 = 250;
    OCR1A = PWM_duty0; //duty 0-250
    OCR1B = PWM_duty1; //duty 0-250
    OCR1C = PWM_duty2; //duty 0-250
    sei();
}

void LED_init()
{
    DDRD = 0xF0; //LEDEk beállítása
    DDRB = 0xF0;
}

int main()
{
    LED_init(); //LED-ek beállítása
    Timer1Init(); //Timer beállítása

    while(1) {
    }
}
```

## 5.4 Soros portkezelés

### 5.4.1 Bevezetés

Az RS-232 interfész a CCITT nemzetközi távközlési bizottság határozta meg a soros adatátviteli interféssz elektromos jellemzőit és feladatait. Ezt az adatátvitelt először számítógépek közötti kommunikációra használták, de a későbbiekben a mikrokontrollerek elterjedésének köszönhetően már minden ilyen nagyobb chip tartalmaz legalább egy soros interfész, melyet UART<sup>2</sup> vagy USART<sup>3</sup> –nak neveznek. A soros kommunikáció során az adatcsomagok egy start bittel kezdődnek, amely a csomag kezdetét jelzik, majd ezt követi a 7-9 -ig terjedő adatbit, amelyet a mikrokontrollereknél külön be tudunk állítani, majd következik egy paritás bit, és az üzenet egy stop bit zárja le. Az adatátviteli sebességét baudrate –ben határozzuk meg, amely a másodpercenként átvitt bitek számát jelenti. Ezen kívül lehetőségünk nyílik beállítani a stopbitek számát, melynek akkor van jelentősége, hogy ha az eszközök feldolgozási teljesítménye nem elegendő, akkor van még egy órajel ideje feldolgozni az adatot. A mikrokontrollerekben az UART általában duplán bufferelt, amely azt jelenti, hogy a fogadott adatot rögtön áttölти egy másik bufferbe és majd onnan történik az adat feldolgozása.

### 5.4.2 ATmega128 USART jellemzői

Jellemzők:

- USART: USART0, USART1
- Full-duplex működés
- Szinkron, asszinkron működés
- 5-9 adatbit, 1-2 STOP bit, paritás ellenőrzés
- adat túlfutás, kerethiba jelzés
- zajszűrés
- 3 megszakítás
- multiprocesszoros működés
- sebességduplázás

Az USART0 a T-bird jobb oldali USB csatlakozóján keresztül érhető el, ezen keresztül kapcsolódhat számítógéphez, ahol egy terminálprogramon keresztül egyszerűen megvalósíthatjuk a kommunikációt. Az USART0 RX kivezetése a PORTE0-n található, a TX

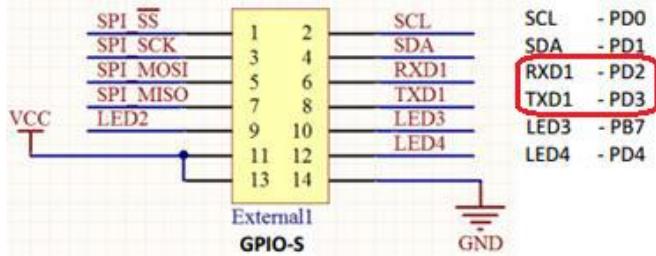
---

<sup>2</sup> Universal asynchronous Receiver/transmitter

<sup>3</sup> (Universal Synchronous Asynchronous Receiver/Transmitter

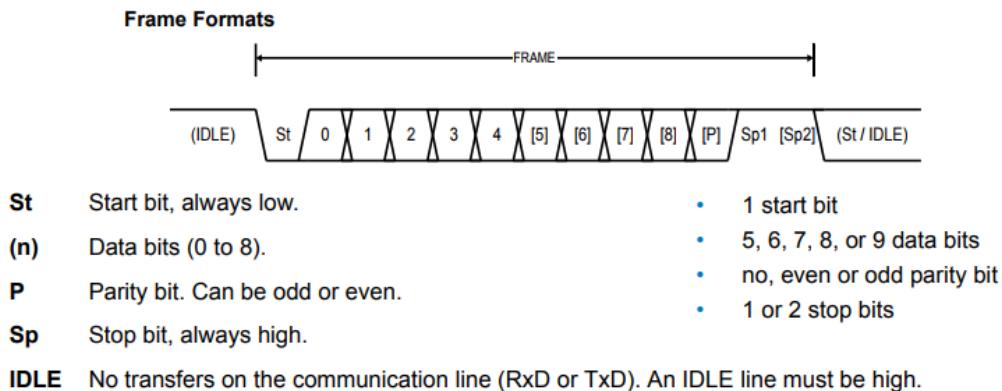
pedig a PORTE1 pinen.

Az UART1 a fejlesztőeszköz szabad felhasználású csatlakozására van kivezetve (az ábrán a T-bird3 GPIO-S csatlakozója látható). Ezen keresztül összekapcsolhatunk két T-bird-öt egymással, vagy külső modulok illesztésére is van lehetőség.



Az USART1 RX kivezetése a PORTD2-n található, a TX pedig a PORTD3 pinen.

Az adatkeret felépítése a következő ábra látható:



17. ábra – Adatkeret

Az adatkeret felépítése *UCSRnC (USART Control and Status Register C)* regiszter bitjeinek beállításával módosítható.

UCSRnC – USART Control and Status Register C <sup>(1)</sup>	Bit	7	6	5	4	3	2	1	0	UCSRnC
	Read/Write	R/W								
	Initial Value	0	0	0	0	0	1	1	0	

Az átviteli sebesség beállítására kétféle módszert kínál a gyártó. Az egyik egy képlet, mellyel kiszámolható az órajel függvényében beállítandó érték. A képlet a következőképpen alakul:

$$\text{BAUD} = \frac{f_{\text{OSC}}}{16(\text{UBRR} + 1)}$$

$$\text{UBRR} = \frac{f_{\text{OSC}}}{16\text{BAUD}} - 1$$

Az UBRR regiszterben (*USART Baud Rate Register*) lehet beállítani az értéket, mely 2db 8 bites regiszterből áll (UBRRH, UBRLR).

Bit	15	14	13	12	11	10	9	8	UBRRnH
	-	-	-	-	UBRRn[11:8]			UBRRnL	UBRRnL
	UBRRn[7:0]								UBRRnL
Read/Write	7 R R/W	6 R R/W	5 R R/W	4 R R/W	3 R/W R/W	2 R/W R/W	1 R/W R/W	0 R/W R/W	UBRRnL
Initial Value	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	UBRRnL

A képlettel való számolás esetében problémát okozhat a kerekítés, ezért pontosabb beállítást egy másik módszerrel tehetünk. Az adatlapban található egytáblázat, mely megadja azt, hogy adott órajel mellett milyen értéket kell beállítani az UBRR-ben.

Baud Rate [bps]	$f_{osc} = 8.0000MHz$				Baud Rate [bps]	$f_{osc} = 16.0000MHz$				
	U2X = 0		U2X = 1			U2X = 0		U2X = 1		
	UBRR	Error	UBRR	Error		UBRR	Error	UBRR	Error	
2400	207	0.2%	416	-0.1%	2400	416	-0.1%	832	0.0%	
4800	103	0.2%	207	0.2%	4800	207	0.2%	416	-0.1%	
9600	51	0.2%	103	0.2%	9600	103	0.2%	207	0.2%	
14.4k	34	-0.8%	68	0.6%	14.4k	68	0.6%	138	-0.1%	
19.2k	25	0.2%	51	0.2%	19.2k	51	0.2%	103	0.2%	
28.8k	16	2.1%	34	-0.8%	28.8k	34	-0.8%	68	0.6%	
38.4k	12	0.2%	25	0.2%	38.4k	25	0.2%	51	0.2%	
57.6k	8	-3.5%	16	2.1%	57.6k	16	2.1%	34	-0.8%	
76.8k	6	-7.0%	12	0.2%	76.8k	12	0.2%	25	0.2%	
115.2k	3	8.5%	8	-3.5%	76.8k	12	0.2%	25	0.2%	

A táblázat még tartalmazza a hibaszázalékokat is. Lehetőség van az átviteli sebesség duplázsára, ekkor csökkenthető az átvitel hibája. Ehhez az U2X regiszter bitet kell 1-be állítani.

Az UCSRnB (*USART Control and Status Register B*) regiszterrel engedélyezhető a küldés és a fogadás (RXEN0, TXEN0 bitekkel).

Bit	7	6	5	4	3	2	1	0	UCSRnB
	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	UCSRnB
	0	0	0	0	0	0	0	0	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	UCSRnB
Initial Value	0	0	0	0	0	0	0	0	UCSRnB

Az adat az UDRn (*USART I/O Data Register*) regiszterbe kerül.

Bit	7	6	5	4	3	2	1	0	UDnR (Read)
	RXB[7:0]								UDnR (Write)
	TXB[7:0]								UDnR (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	UDnR (Read)
Initial Value	0	0	0	0	0	0	0	0	UDnR (Write)

Küldés, fogadás esetén az *UCSRnA* (*USART Control and Status Register A*) bitjeit kódunkban figyelve megállapítható a küldés/fogadás vége és a különböző hibák. Ezen felül, a sebesség duplázása és a multiprocesszoros működés is itt állítható be.

UCSRnA – USART Control and Status Register A								
Bit	7	6	5	4	3	2	1	0
ReadWrite	R	R/W	R	R	R	R	R/W	R/W
Initial Value	0	0	1	0	0	0	0	0

### 5.4.3 Mintakód

A mintakód az UART0 beállítását ismerteti. A mintakód (az adatlapban hasonló megtalálható) 8adatbittel, 2 stopbittel és 0 paritással dolgozik. Adatátviteli sebességnak 9600bps kerül beállításra. Az inicializáláson kívül a legegyszerűbb küldés és fogadás is a kód részét képezi. A mintaprogram, csak kódrészlet, kipróbáláshoz kiegészítés megírása szükséges, az alapvető beállítások megmutatása a célja.

#### 5.4.3.1 AVR ASM kódrészlet

```

USART0_Init:
    ; Set baud rate
    out    UBRR0H, r17
    out    UBRR0L, r16
    ; Enable receiver and transmitter
    ldi   r16, (1<<RXEN0) | (1<<TXEN0)
    out   UCSRnB, r16
    ; Set frame format: 8data, 2stop bit
    ldi   r16, (1<<USBS0) | (3<<UCSZ00)
    out   UCSR0C, r16
ret

USART0_Transmit:
    ; Wait for empty transmit buffer
    sbis  UCSR0A, UDRE0
    rjmp  USART0_Transmit
    ; Put data (r16) into buffer, sends the data
    out   UDR0, r16
ret

USART0_Receive:
    ; Wait for data to be received
    sbis  UCSR0A, RXC0
    rjmp  USART0_Receive
    ; Get and return received data from buffer
    in    r16, UDR0
ret

```

#### 5.4.3.2 AVR C kódrészlet

```
#define FOSC 16000000// Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1

void main( void )
{
    ...
    USART0_Init ( MYUBRR );
    ...
}

void USART0_Init( unsigned int ubrr )
{
    /* Set baud rate */
    UBRR0H = (unsigned char)(ubrr>>8);
    UBRR0L = (unsigned char)ubrr;
    /* Enable receiver and transmitter */
    UCSR0B = (1<<RXEN0)|(1<<TXEN0);
    /* Set frame format: 8data, 2stop bit */
    UCSR0C = (1<<USBS0)|(3<<UCSZ00);
}

void USART0_Transmit( unsigned char data )
{
    /* Wait for empty transmit buffer */
    while ( !( UCSR0A & (1<< UDRE0) )
    ;
    /* Put data into buffer, sends the data */
    UDR0 = data;
}

unsigned char USART0_Receive( void )
{
    /* Wait for data to be received */
    while ( !(UCSR0A & (1<< RXC0)) )
    ;
    /* Get and return received data from buffer */
    return UDR0;
}
```

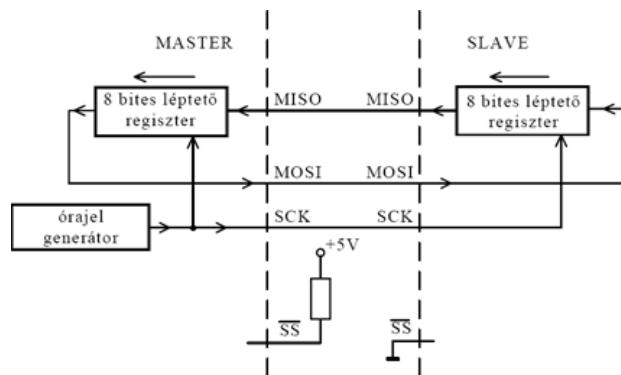
## 5.5 SPI

### 5.5.1 Bevezetés

Az SPI buszt a Motorola fejlesztette ki, amely egy full duplex protokoll. Mint ahogyan a nevéről is adódik ez egy soros szinkron interfész. Az SPI kommunikáció is a master-slave kapcsolatot alakítja ki, annyi különbséggel az I<sup>2</sup>C buszhoz képest, hogy itt a címzés nem az adat buszon történik, hanem minden slave eszköz rendelkezik egy ún. „<sup>4</sup>CS” lábbal, amely állításával tudja megcímzni az eszközt a master. A megcímzett eszköz felől ezt a lábat „<sup>5</sup>SS” - nék jelölik a gyártók. minden mikrokontroller rendelkezik hardveresen chip select lábakkal, de ezeket szoftveresen a felhasználó is tudja állítani. Az SPI kommunikációjánál a szerepek hardveresen kötöttek, így a master – slave szerep felcserélése nem lehetséges (egy mikrokontroller és egy külső periféria között). Viszont, ha kettő darab mikrokontroller kommunikál egymással, akkor ezek egy újra konfigurálással könnyedén megoldhatók. Magához a kommunikációhoz 4 darab vezetékre van szükség:

- MOSI (Master Output Slave Input)
- MISO (Master Input Slave Output)
- SCK (Serial Clock)
- CS (Chip Select)

A kommunikáció szinkronizálása az SCK vezetéken történő órajelre történik. És a MOSI és a MISO lábakon a master és a slave eszköz úm. adatot cserél bitenként (mindig a master generálja az órajelet). Ezt a kommunikációt úgy is elkövethetjük, mint kettő darab shift regisztert, amelyek az órajelre bitenként adatot cserélnek. Az 13. ábra a MOSI és a MISO kapcsolatát és a busz bekötését szemlélteti.



18. ábra - SPI kommunikáció összeköttetés

<sup>4</sup> CS - Chip Select

<sup>5</sup> SS - Slave Select

Ha a mikrokontrollerrel szeretnénk egy SPI kommunikációt kialakítani, akkor lehetőségünk adódik, beállítani az órajel fázisát és polaritását, melynek köszönhetően különféle megoldású soros elemek is összekapcsolhatók az SPI rendszerrel. Ezen kívül több beállítási mód is létezik, hogy az adattranszfer az órajel mely „részénél” történjen meg. Lehetőségünk van fel- és lefutó ére is beállítani, ún. alap és fordított polaritásnál is.

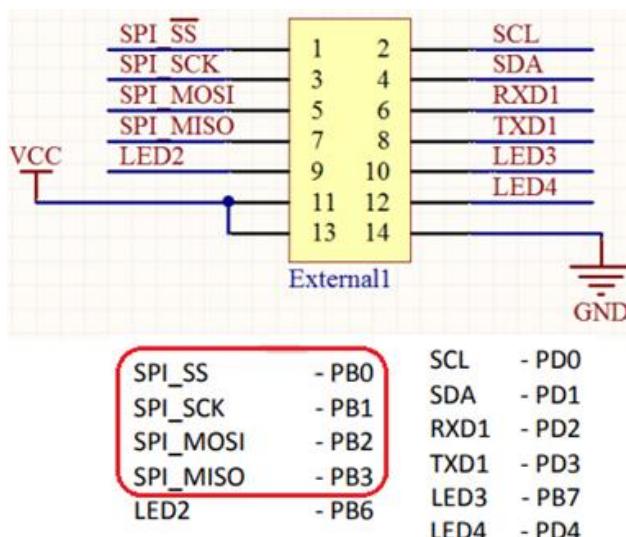
Az SPI buszt általában 2MHz-ig használják, de pl.: a Xicor cég X2565 soros adatkezelésű EEPROM IC-je, amely SPI jelleggel kezelhető, akár az 5MHz es működési frekvenciát is elérheti.

### 5.5.2 ATmega128 SPI jellemzői

Jellemzők:

- Full-duplex, 3 vezetékes szinkron adatátvitel
- Master vagy slave működés
- Programozható átviteli sebesség
- Átvitel végét jelző megszakítás
- Írásütközés védelmi flag
- Shiftelés iránya
  - o MSB -> Most Significant Bit
  - o LSB -> Least Significant Bit

Az SPI SS kivezetése a PORTB0, az SCK a PORTB1, a MOSI a PORTB2, a MISO pedig a PORTB3 pinen található. Az ábra szemlélteti a T-bird3 szabadfelhasználású csatlakozóját és az egyes lábak funkcióját.



Az SPCR (SPCR – SPI Control Register) regiszter bitjeivel engedélyezhetjük az SPI kommunikációt, megszakítást, beállíthatjuk az adatirányt, az órajel polaritását, az órajel osztását és választhatunk master és slave működés között.

SPCR – SPI Control Register								
Bit	7	6	5	4	3	2	1	0
0x0D (0x2D)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
ReadWrite	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

Az SPSR (SPI Status Register) 7. bitje az SPI Interrupt Flag (SPIF). Az átvitel után a flag értéke 1, abban az esetben, ha a megszakítás engedélyezve lett, megszakítást generál.

SPSR – SPI Status Register								
Bit	7	6	5	4	3	2	1	0
0x0E (0x2E)	SPIF	WCOL	-	-	-	-	-	SPI2X
ReadWrite	R	R	R	R	R	R	R	R/W
Initial Value	0	0	0	0	0	0	0	0

Az SPDR (SPI Data Register) regiszterben tárolódik az adat átvitelkor.

SPDR – SPI Data Register								
Bit	7	6	5	4	3	2	1	0
0x0F (0x2F)	MSB							LSB
ReadWrite	R/W							
Initial Value	X	X	X	X	X	X	X	X

### 5.5.3 Mintakódok

Az SPI beállítására szintén kódrészletek olvashatók a következőkben. A mintakódok ismertetik a master/slave mód beállításait, illetve az ezekhez tartozó küldés és fogadás megvalósítását.

#### 5.5.3.1 AVR Assembly SPI MASTER kódrészlet

```

SPI_MasterInit:
    ; Set MOSI and SCK output, all others input
    ldi    r17, (1<<DD_MOSI)|(1<<DD_SCK)
    out   DDR_SPI, r17
    ; Enable SPI, Master, set clock rate fck/16
    ldi    r17, (1<<SPE)|(1<<MSTR)|(1<<SPR0)
    out   SPCR, r17
ret

SPI_MasterTransmit:
    ; Start transmission of data (r16)
    out   SPDR, r16
Wait_Transmit:
    ; Wait for transmission complete
    sbis  SPSR, SPIF
    rjmp  Wait_Transmit
ret

```

#### 5.5.3.2 AVR C SPI MASTER kódrészlet

```
void SPI_MasterInit(void)
{
    /* Set MOSI and SCK output, all others input */
    DDR_SPI = (1<<DD_MOSI)|(1<< DD_SCK);
    /* Enable SPI, Master, set clock rate fck/16 */
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}

void SPI_MasterTransmit(char cData)
{
    /* Start transmission */
    SPDR = cData;
    /* Wait for transmission complete */
    while (!(SPSR & (1<<SPIF)))
    ;
}
```

#### 5.5.3.3 AVR Assembly SPI SLAVE kódrészlet

```
SPI_SlaveInit:
    ; Set MISO output, all others input
    ldi    r17, (1<< DD_MISO)
    out    DDR_SPI, r17
    ; Enable SPI
    ldi    r17, (1<<SPE)
    out    SPCR, r17
ret

SPI_SlaveReceive:
    ; Wait for reception complete
    sbis   SPSR, SPIF
    njmp  SPI_SlaveReceive
    ; Read received data and return
    in     r16,   SPDR
ret
```

#### 5.5.3.4 AVR C SPI SLAVE kódrészlet

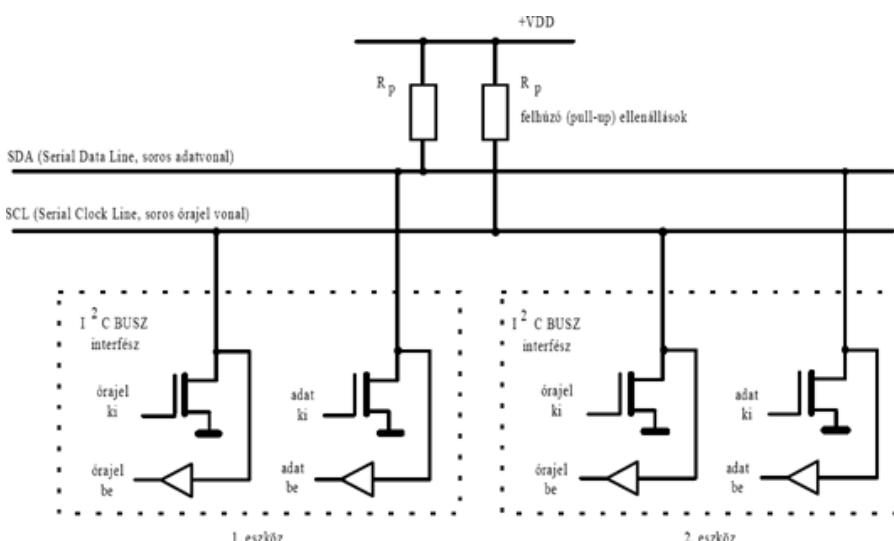
```
void SPI_SlaveInit(void)
{
    /* Set MISO output, all others input */
    DDR_SPI = (1<< DD_MISO);
    /* Enable SPI */
    SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
    /* Wait for reception complete */
    while (!(SPSR & (1<<SPIF)))
    ;
    /* Return data register */
    return SPDR;
}
```

## 5.6 TWI

### 5.6.1 Bevezetés

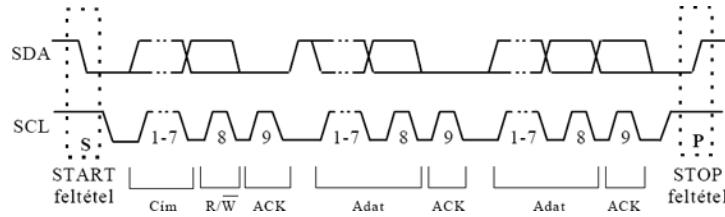
Története: Az 1980- as évek elején a Philips fejlesztette ki abból a célból, hogy a Televíziójukban a központi egység minél kevesebb vezetéken tudjon kommunikálni a perifériákkal. Az I<sup>2</sup>C egy soros szinkron adatátviteli rendszer. A kommunikáció két vezetéken történik: SDA (Serial Data Line), SCL (Serial Clock Line). Az SDA vonalon történik az adatátvitel az SCL vonal pedig szolgáltatja az adatátvitel ütemezéséhez szükséges órajelet, a start és a stop bitet. A buszra csatlakoztatott eszközök a két vezetékre nyitott draines ill. nyitott kollektoros kimenettel csatlakoznak (19. ábra), ezért fontos, hogy az SDA és az SCL vezetéket is általánosan 2k2 – 4k7 –es ellenállásokkal pozitív tápra húzzuk. Az ellenállások értéke annak függvénye, hogy milyen sebességen kommunikálnak az eszközök és hogy milyen a kapacitása a vezetékeknek. Fontos megjegyezni, hogy a buszon lévő eszközök külön címekkel rendelkeznek.



19. ábra - Open kollektoros csatlakozások

A kommunikáció folyamata: Az adat küldést mindenkor a master kezdeményezi úgy, hogy amíg az órajel magas állapotában nullára változtatja az SDA- n lévő értéket. Ez jelenti a start bit- et. Majd ezt követi az eszköz címe, amely általában 7 bit, mivel a 8.bit a R/W bit, amely azt jelzi, hogy a master olvasni vagy írni akar a slave eszkösről. Ha ez megtörtént, akkor a megcímzett slave eszköz visszaküld egy ACK bit- et mellyel visszaigazolja a vételt. Ezután megkezdődik az adatbritek átküldése, fogadása melyeket mindenkor egy ACK bit zár le az adatellenőrzés miatt.

Majd, ha befejeződött az adatátvitel, akkor a master egy magas órajel állapotban magas állapotba állítja az SDA vonalon lévő bitet, ezzel jelezvén az adatátvitel végét. A 20.ábra ezt szemlélteti.



**20. ábra - I2C kommunikáció folyamata**

Tehát összefoglalva: az I<sup>2</sup>C adatátvitel két vezetéken történik (SDA, SCL), minden eszköz saját címmel rendelkezik, multi master- es, a busz időzítését mindig a master végzi, a maximális működési frekvenciája: -eredetileg 100kb/sec, –fast módban 3.2Mb/sec, –ultra fast módban 5Mb/sec

### 5.6.2 ATmega128 TWI jellemzői

Jellemzők:

- slave és master működés támogatott
- küldés, fogadás
- programozható 7bites cím → 128 különböző salve cím
- multi-master arbitráció támogatás
- 400kHz adatátviteli sebesség
- zajelnyomás

A TWBR (TWI Bit Rate Register) regiszterrel állíthatjuk az órajel osztását az átvitelhez, a bitkombinációkhöz tartozó osztásokat az adatlapban található képlettel számolhatjuk:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{\text{TWPS}}}$$

A képletben a TWBR a regiszterben beállított értéket jelzi, a TWPS a TWI státusz regiszterben beállított előosztás bitek értékét.

TWBR – TWI Bit Rate Register								
Bit (0x70)	7	6	5	4	3	2	1	0
Read/Write	R/W							
Initial Value	0	0	0	0	0	0	0	0

A TWCR (TWI Control Register) regiszterrel irányíthatjuk a TWI működését. Itt található többek között a TWI interrupt flag, az engedélyező bit, start/stop felétel bit.

TWCR – TWI Control Register								
Bit (0x74)	7	6	5	4	3	2	1	0
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W
Initial Value	0	0	0	0	0	0	0	0

A TWSR (TWI Status Register) regiszter felső 5 bitje adja meg a TWI státuszát, ezt az adatlapban található összefoglaló táblázat foglalja össze. Az alsó két bit pedig az előosztást adja meg, mely az előzőekben említett képletben kerül felhasználásra.

TWSR – TWI Status Register								
Bit	7	6	5	4	3	2	1	0
(0x71)	TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0
Read/Write	R	R	R	R	R	R	R/W	R/W
Initial Value	1	1	1	1	1	0	0	0

Fontos még megemlíteni a TWDR (TWI Data Register) regisztert, melyeben átvitelkor tárolódik a következő átvienő byte.

TWDR – TWI Data Register								
Bit	7	6	5	4	3	2	1	0
(0x73)	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0
Read/Write	R/W							
Initial Value	1	1	1	1	1	1	1	1

A TWAR (TWI (Slave) Address Register) regiszterben állítható be a 7 bites slave cím, a regiszter felső 7 bitjén.

TWAR – TWI (Slave) Address Register								
Bit	7	6	5	4	3	2	1	0
(0x72)	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
Read/Write	R/W							
Initial Value	1	1	1	1	1	1	1	0

### 5.6.3 Mintakódok

A TWI mintakód szintén egy kódrészlet, mely egy tipikus küldési folyamatot mutat be.

#### 5.6.3.1 AVR Assembly kódrészlet

```
; Send START condition
ldi    r16, (1<<TWINT) | (1<<TWSTA) | (1<<TWEN)
out   TWCR, r16
;Wait for TWINT flag set.
;This indicates that the START condition has been transmitted
wait1:
in    r16, TWCR
sbrs  r16, TWINT
rjmp  wait1
;Check value of TWI Status Register.
;Mask prescaler bits. If status different from START go to ERROR
in    r16, TWSR
andi  r16, 0xF8
cpi   r16, START
brne  ERROR
;Load SLA_W into TWDR Register.
;Clear TWINT bit in TWCR to start transmission of address
ldi   r16, SLA_W
out  TWDR, r16
ldi   r16, (1<< TWINT) | (1<<TWEN)
out  TWCR, r16
;Wait for TWINT flag set. This indicates that the SLA+W has been transmitted,
;and ACK/NACK has been received.
wait2:
in    r16, TWCR
sbrs  r16, TWINT
rjmp  wait2
; Check value of TWI Status Register. Mask prescaler bits.
;If status different from MT_SLA_ACK go to ERROR
in    r16, TWSR
andi  r16, 0xF8
cpi   r16, MT_SLA_ACK
brne  ERROR
; Load DATA into TWDR Register.
;Clear TWINT bit in TWCR to start transmission of data
ldi   r16, DATA
out  TWDR, r16
ldi   r16, (1<< TWINT) | (1<<TWEN)
out  TWCR, r16
;Wait for TWINT flag set. This indicates that the DATA has been transmitted,
;and ACK/NACK has been received.
wait3:
in    r16, TWCR
sbrs  r16, TWINT
rjmp  wait3
;Check value of TWI Status Register. Mask prescaler bits.
;If status different from MT_DATA_ACK go to ERROR
in    r16, TWSR
andi  r16, 0xF8
cpi   r16, MT_DATA_ACK
brne  ERROR
;Transmit STOP condition
ldi   r16, (1<<TWINT) | (1<<TWEN) | (1<<TWSTO)
out  TWCR, r16
```

### 5.6.3.2 AVR C kód részlet

```
//Send START condition
TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<<TWEN)

/* Wait for TWINT flag set. This indicates that the START condition has
been transmitted*/
while (!(TWCR & (1<< TWINT)))
;

/* Check value of TWI Status Register. Mask prescaler bits. If status dif-
ferent from START go to ERROR*/
if ((TWSR & 0xF8) != START) ERROR();

/* Load SLA_W into TWDR Register. Clear TWINT bit in TWCR to start
transmission of address*/
TWDR = SLA_W;
TWCR = (1<<TWINT)|(1<<<TWEN);

/* Wait for TWINT flag set. This indicates that the SLA+W has been
transmitted, and ACK/NACK has been received.*/
while (!(TWCR & (1<< TWINT)))
;

/* Check value of TWI Status Register. Mask prescaler bits. If status dif-
ferent from MT_SLA_ACK go to ERROR*/
if ((TWSR & 0xF8) != MT_SLA_ACK) ERROR();

/* Load DATA into TWDR Register. Clear TWINT bit in TWCR to start transmis-
sion of data*/
TWDR = DATA;
TWCR = (1<< TWINT)|(1<<<TWEN);

/*Wait for TWINT flag set. This indicates that the DATA has been transmit-
ted, and ACK/NACK has been received.*/
while (!(TWCR & (1<< TWINT)))
;

/* Check value of TWI Status Register. Mask prescaler bits. If status dif-
ferent from MT_DATA_ACK go to ERROR*/
if ((TWSR & 0xF8) != MT_DATA_ACK) ERROR();

/* Transmit STOP condition*/
TWCR = (1<<TWINT)|(1<<TWEN)|(1<<<TWSTO);
```

## 6 T-bird3

### 6.1 Hardveres felépítése

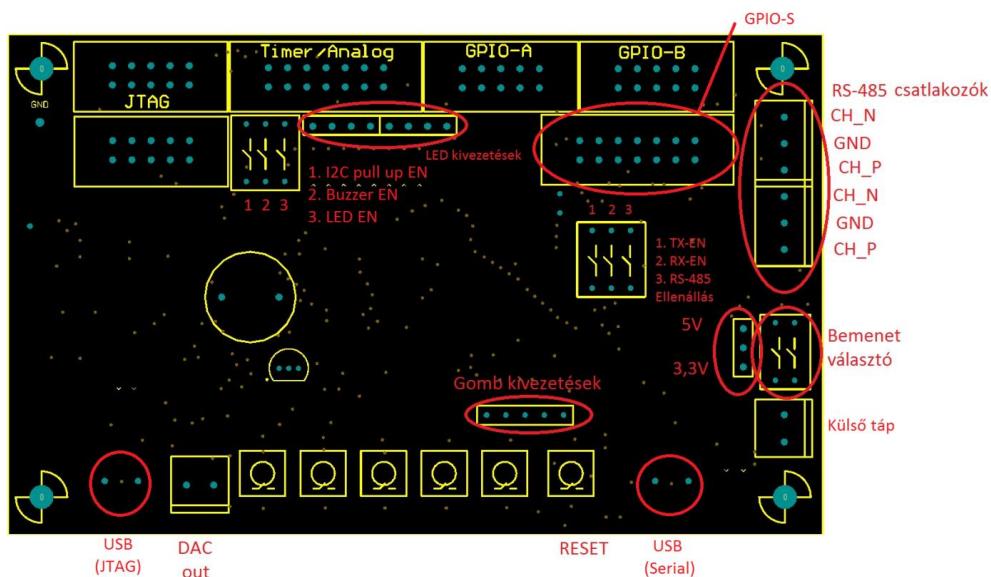
#### Főbb tulajdonságok

- ATMEL AVR – Atmega128 típusú mikrokontroller
- USB – soros átalakító
- valós idejű óra (RTC – Real Time Clock, PCF8563)
- Piezzo buzzer
- Analóg hőmérséklet szenzor (LM35)
- RS-485 interfész (SN75176, belső védelemmel)
- USB és külső tápellátás lehetőség
- 5 db nyomógomb
- 8 db LED
- Integrált JTAG debugger (JTAG ICE)
- védőbiztosíték
- Digitál – Analóg Átalakító

A T-bird 3 fejlesztőpanel az előző verziók teljes értékű helyettesítése. A fejlesztő panel csatlakozói és lábkiosztása megegyezik az előző verziók kiosztásával, ezért a kiegészítő panelok csatlakoztatása nem okoz gondot. A JTAG ICE programozónak köszönhetően a panel a vásárlás után nem igényel semmi plusz alkatrészt.

#### 6.1.1 Csatlakozók ismertetése

A T-bird 3 fejlesztői panelre minden külső eszközt hagyományos szalagkábel csatlakozón keresztül tud rácsatlakoztatni.



**Külső táp:** A fejlesztőpanel külső tápellátására szolgáló csatlakozó. **A bemeneti feszültség maximális értéke +5V lehet.**

**Bemenet választó:** A DIP-SWITCH kapcsoló segítségével választhatjuk ki, hogy mely bemeneti feszültséggel szeretnénk megtáplálni a fejlesztőpanelt.

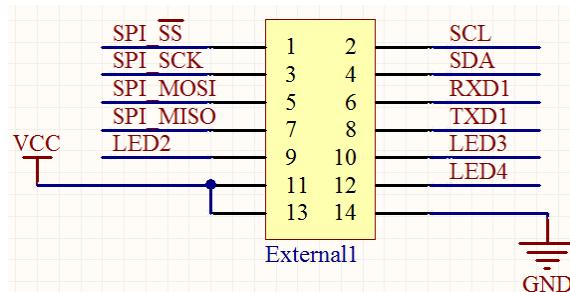
1. kapcsoló: külső csatlakozó
2. kapcsoló: USB csatlakozó

Egyes modellek esetében ezenfelül a csatlakozó mellett található jumperrel választhatjuk ki, hogy a panelt milyen feszültségszintről szeretnénk üzemeltetni (3.3V vagy 5V).

**RS-485 csatlakozók:** A csatlakozók duplikálása megkönnyíti a fejlesztőpanel buszba való beépítését (továbbvezetés).

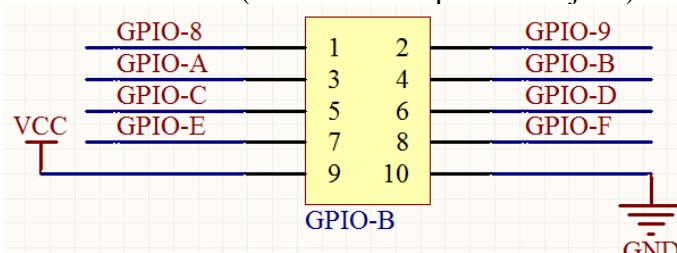
**RS-485 DIP SWITCH:** Ezen a kapcsolón keresztül tudjuk letiltani az RS-485 meghajtó IC-t, és állítani a buszlezáró ellenállást. **Fontos, hogy ha 3,3V-ról működteti a panelt, akkor ezek a kapcsolók be legyenek kapcsolva!**

**GPIO-S:** Általános célú IO kivezetés. (bővebben a kapcsolási rajzon)



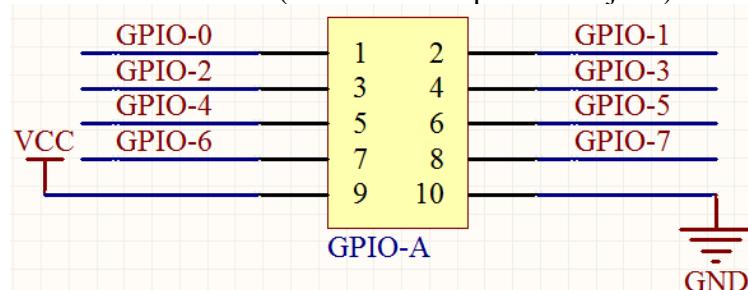
SPI_SS	- PB0	SCL	- PD0
SPI_SCK	- PB1	SDA	- PD1
SPI_MOSI	- PB2	RXD1	- PD2
SPI_MISO	- PB3	TXD1	- PD3
LED2	- PB6	LED3	- PB7
VCC		LED4	- PD4
External			
GND			

**GPIO-B:** Általános célú IO kivezetés. (bővebben a kapcsolási rajzon)



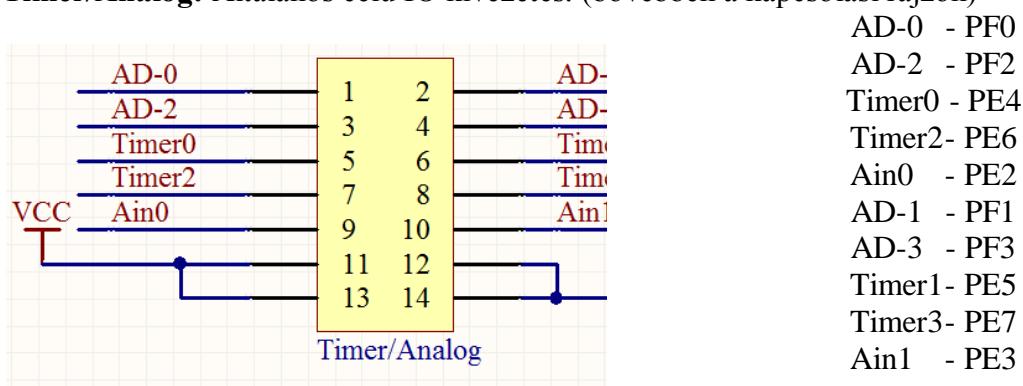
GPIO-8	- PC0	GPIO-C	- PC4
GPIO-9	- PC1	GPIO-D	- PC5
GPIO-A	- PC2	GPIO-E	- PC6
GPIO-B	- PC3	GPIO-F	- PC7

**GPIO-A:** Általános célú IO kivezetés. (bővebben a kapcsolási rajzon)

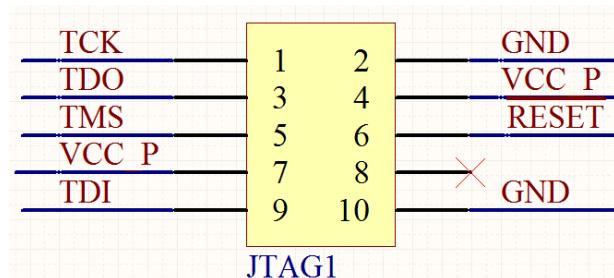


GPIO-0	- PA0	GPIO-4	- PA4
GPIO-1	- PA1	GPIO-5	- PA5
GPIO-2	- PA2	GPIO-6	- PA6
GPIO-3	- PA3	GPIO-7	- PA7

**Timer/Analog:** Általános célú IO kivezetés. (bővebben a kapcsolási rajzon)



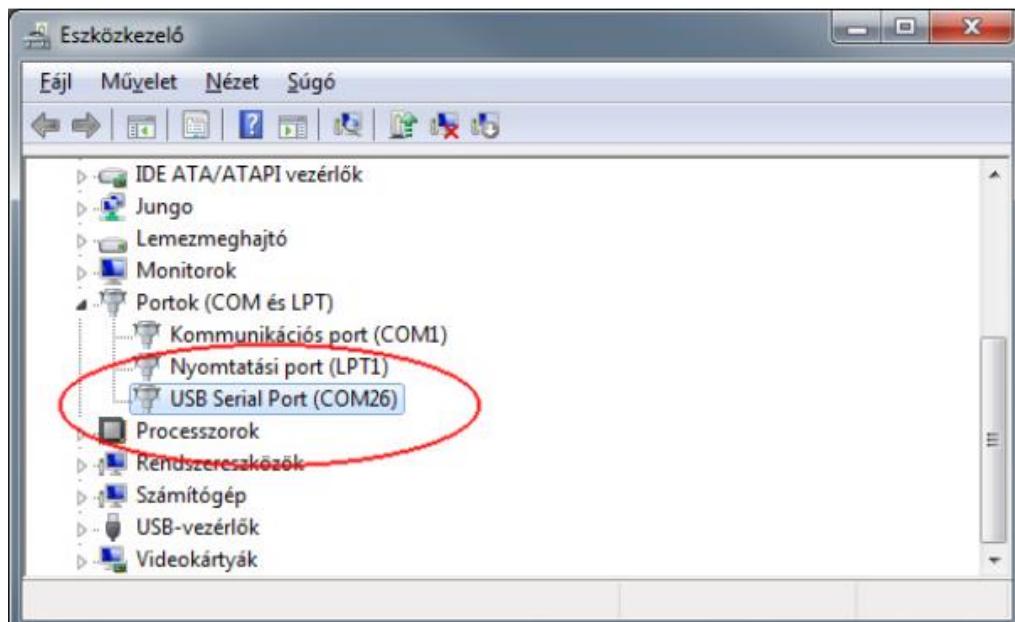
**JTAG csatlakozó:** Az integrált JTAG debugger csatlakozó felülete.



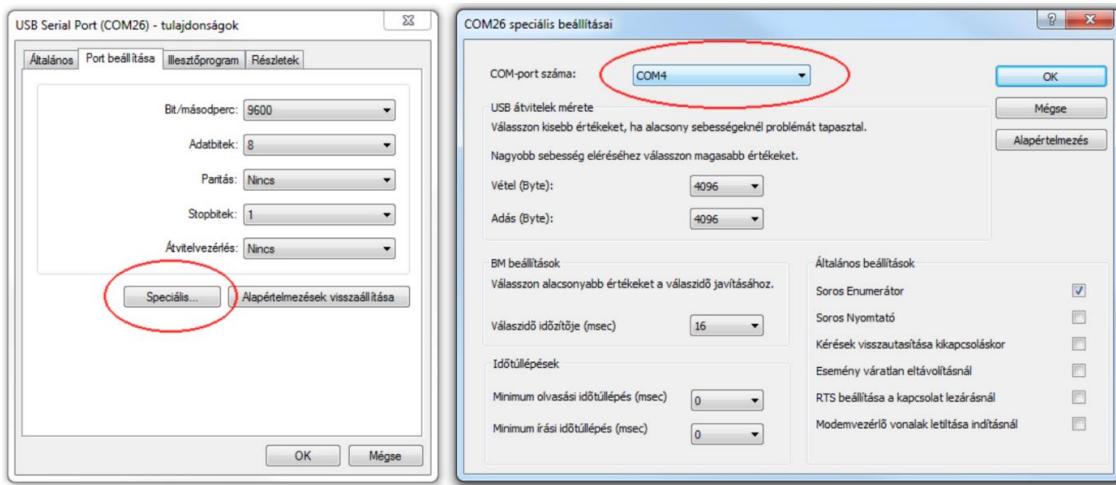
### 6.1.2 JTAG debugger használata

Az USB (JTAG) csatlakozón keresztül csatlakoztatjuk a számítógéphez a fejlesztői panelt. Ekkor a számítógép automatikusan eszközillesztő szoftvert keres, az előre beállított mappákban. Amennyiben az eszközillesztő szoftver telepítése sikertelen, a legfrissebb illesztőprogram letöltése ajánlott a [www.ftdichip.com](http://www.ftdichip.com) oldalról, az FT232RL típusú USB-Soros illesztő IC-hez. Figyelem! Egyes AVR Studio verziók (pl. v 4) nem képesek kezelní a magasabb port számokra kerülő virtuális soros porti JTAG debuggereket. Így szükséges lehet az eszközillesztő szoftver telepítése után a port számot módosítani, az alábbi módon:

1. Nyissuk meg a Számítógép > **Eszközkezelő** ablakot a módosítani kívánt portot (jelen esetben COM26)



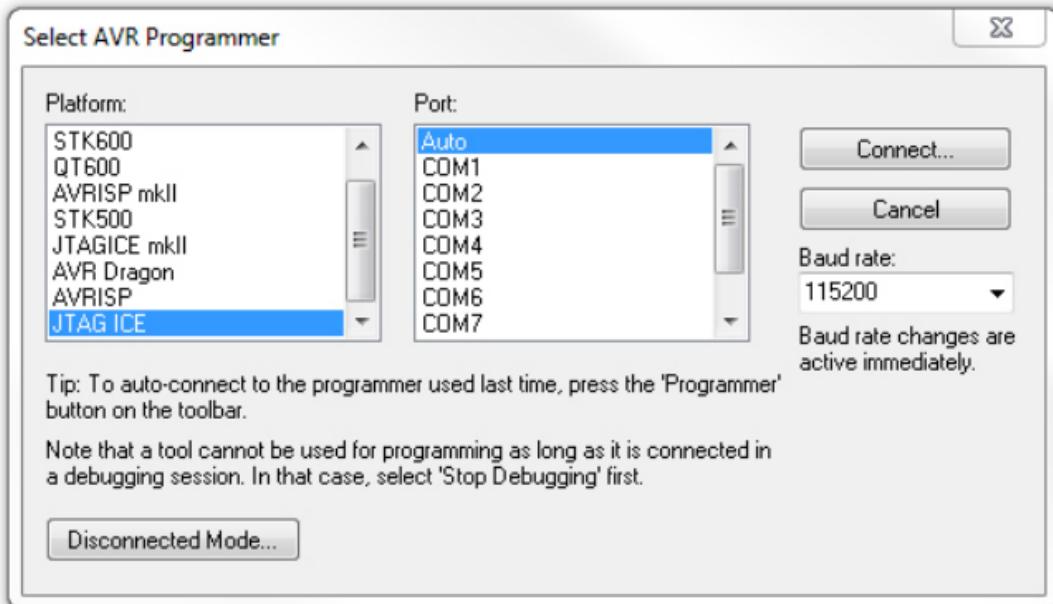
2. Jobb kíkk, Tulajdonságok, majd a Port beállítása fülön kattintsunk a Speciális... gombra.
3. Állítsuk át a COM-port száma mezőt egy COM1-9-ig terjedő értékre (jelen példában COM4), majd az OK gomb megnyomásával térjünk vissza az eszközkezelőbe.



4. Ezzel a virtuális soros port mostantól COM4-en érhető el, ezt kell kiválasztani az AVR Studio programban.

A következő lépésként nyissuk meg az AVR Studio 4 programot.

Válasszuk ki a Tools > Program AVR > Connect menüpontot, majd a megjelenő listából válasszuk a JTAG ICE eszközt.



A Connect... gomb megnyomásával az AVR Studio csatlakozott is.

### 6.1.3 Fontos tudnivalók

A programozásnál és a Fuse-bit beállításoknál fokozottan ügyeljünk a JTAG, ISP és oszcillátor beállításokra. Amennyiben helytelen értékre állítjuk ezen biteket, a teljes fejlesztői panel működésképtelenné válhat, ez az eset nem tartozik a garanciális meghibásodások körébe.

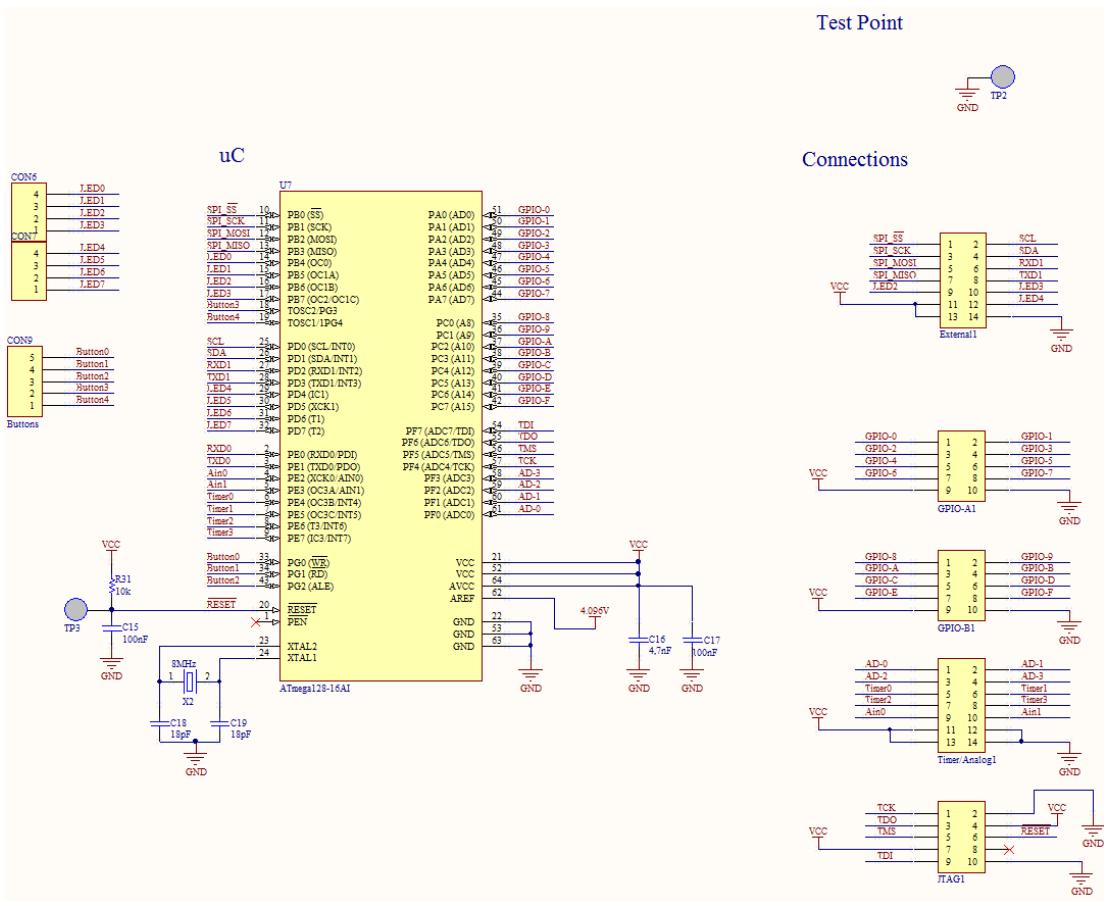
Amennyiben a JTAG és ISP Fuse biteket kikapcsoljuk, úgy abban az esetben az ATmega128 mikrovezérlő minden további programozását letiltjuk, így használhatatlanná válik a teljes fejlesztői panel. Fokozottan ügyeljünk ezen bitek beállításaira!

Amennyiben a PWR LED nem világít, de a tápfeszültséget valamelyik USB csatlakozón vagy külső tápfeszültség csatlakozón keresztül biztosítottuk, úgy abban az esetben az olvadó biztosíték szakadt meg rövidzár miatt. Távolítsuk el az áramkörből a fejlesztői panelt, és vizsgáljuk meg mivel okozhattuk a rövidzárat. Ezt követően az olvadó biztosíték cseréje szükséges.

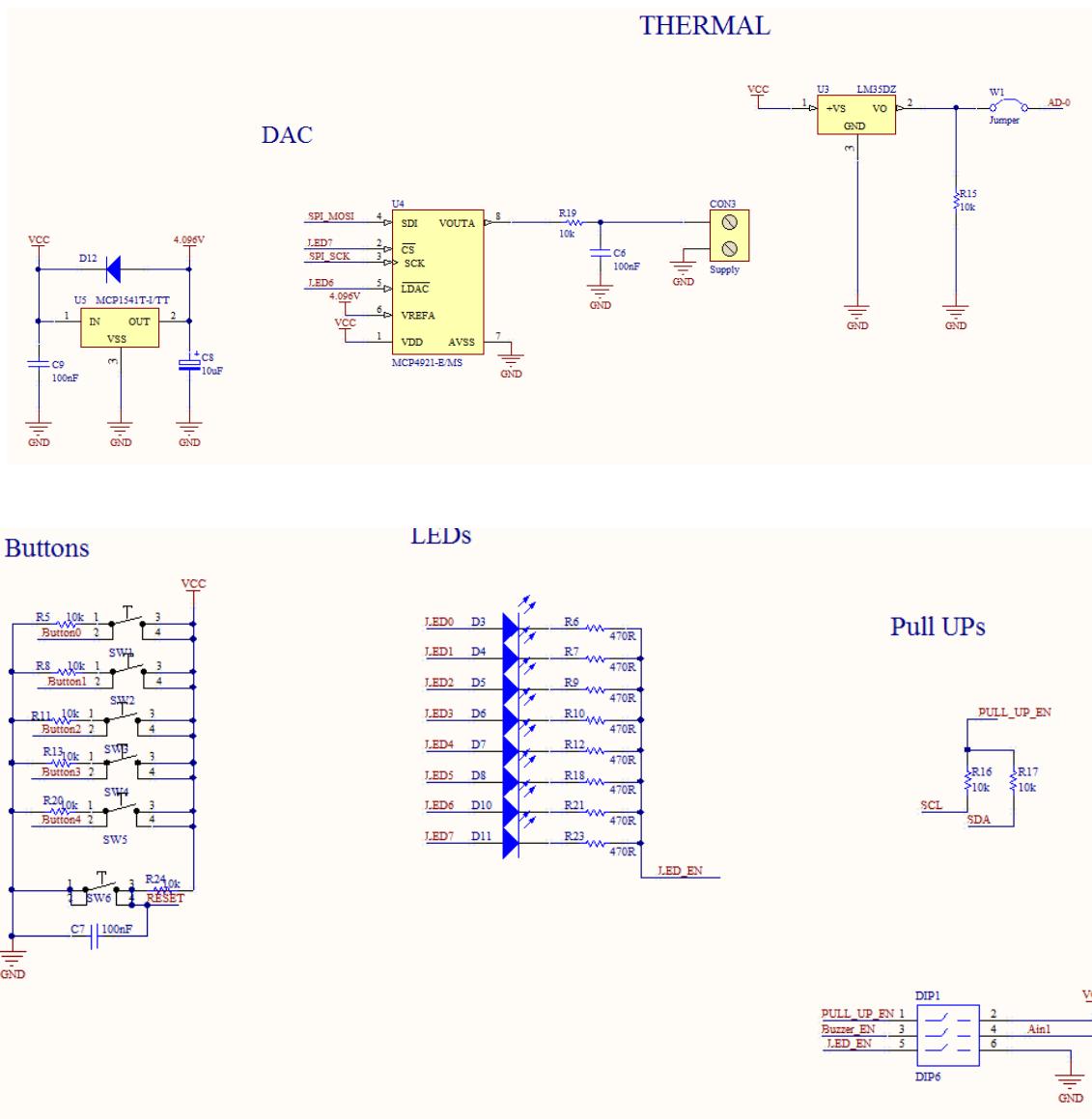
A fejlesztői panel nem rendelkezik túlfeszültség védelemmel, így fokozottan figyeljünk külső tápfeszültség forrás alkalmazása esetén.

Minden T-Bird 3 fejlesztői panel részletesen, minden funkcióját tesztelve kerül forgalomba.

### 6.1.4 Alapáramkör



### 6.1.5 LED-ek és GOMB-ok



### 6.1.5.1 LED-ek kezelése mintakódok

A LED-ek egyik látványos felhasználása a különböző futófények megvalósítása. A következőkben 3 félle futófényre látható példa Assembly és C programozási nyelveken. Az első futófény egy egyszerű körbefutást mutat be, a második egy ide-oda futófényt, a harmadik pedig a híres Knight Rider.

#### 6.1.5.1.1 AVR Assembly balra-futófény mintakód

```
;-----Balra-futófény-----  
  
.INCLUDE "m128def.inc" ; Include definitions Atmega128  
.DSEG  
;  
.CSEG  
.DEF LED = r18  
.ORG 0x0  
rjmp start ; jump to start  
  
.ORG 0x100  
;  
;  
; MACRO  
;  
.macro LED_init  
ldi r16, 0xF0  
out DDRB, r16  
out DDRD, r16  
.endmacro  
  
.macro LED_out  
out PORTD, LED  
swap LED  
out PORTB, LED  
swap LED  
.endmacro  
  
.macro delay_1s ;Késleltetés  
ldi r26, 0xFF  
ldi r25, 0xFF  
ldi r24, 0x5F  
delay_loop:  
dec R26 ; lefele számol, ha 0 lesz, akkor Z flag=1  
brne delay_loop ; Z flag-et nézi, ha Z flag=0, akkor ugrik,  
dec R25  
brne delay_loop;  
dec R24  
brne delay_loop  
  
.endmacro  
;  
; SUBRUTINE  
;  
;  
; INTERRUPT  
;  
  
;  
; PROGRAM
```

```

;-----
; start:
;-----
; INIT
;-----
; LED_init
;-----
; PROGRAM
;-----
ldi    LED, 0x01

loop:
    LED_out
    delay_1s
    lsl    LED          ;balra shift
    brcs  c_s          ;BRCS: Branch if Carry Set, ugrás
                        ;ha a C flag értéke 1 a c_s címkére
    brcc  loop          ;BRCC: Branch if Carry Cleared, ugrás
                        ;ha a C flag értéke 0 a loop címkére

c_s:
    ldi    LED, 0x01

rjmp  loop

```

#### 6.1.5.1.2 AVR Assembly ide-oda futófény mintakód

```
;-----Oda-Vissza-----  
.INCLUDE "m128def.inc"           ; Include definitions Atmega128  
.DSEG  
;  
.CSEG  
.DEF LED      =      r18  
.ORG 0x0  
rjmp start           ; jump to start  
  
.ORG 0x100  
;-----  
; MACRO  
;  
.macro LED_init  
    ldi r16, 0xF0  
    out DDRB, r16  
    out DDRD, r16  
.endmacro  
  
.macro LED_out  
    out PORTD, LED  
    swap LED  
    out PORTB, LED  
    swap LED  
.endmacro  
  
.macro delay_1s          ;Késleltetés  
    ldi r26, 0xFF  
    ldi r25, 0xFF  
    ldi r24, 0x5F  
    delay_loop:  
        dec R26          ; lefele számol, ha 0 lesz, akkor Z flag=1  
        brne delay_loop ; Z flag-et nézi, ha Z flag=0, akkor ugrik,  
        dec R25  
        brne delay_loop;  
        dec R24  
        brne delay_loop  
.endmacro  
;  
; SUBRUTINE  
;  
;  
; INTERRUPT  
;  
;  
; PROGRAM  
;  
;  
start:  
;  
; INIT  
;  
    LED_init
```

```
;-----  
; PROGRAM  
;-----  
ldi    LED, 0x01  
  
loop:  
  
balra:  
    LED_out  
    delay_1s  
    lsl    LED  
    cpi    LED, 0x80  
    breq   jobbra  
  
    brne   balra  
  
jobbra:  
    LED_out  
    delay_1s  
    lsr    LED  
    cpi    LED, 0x01  
    breq   loop  
    brne   jobbra  
  
rjmp   loop
```

#### 6.1.5.1.3 AVR Assembly Knight Rider futófény mintakód

```
;-----Knight Rider-----  
.INCLUDE "m128def.inc" ; Include definitions Atmega128  
.DSEG  
;  
.CSEG  
.DEF LED = r18  
.DEF LED1 = r19  
.DEF LED2 = r20  
.ORG 0x0  
rjmp start ; jump to start  
  
.ORG 0x100  
;  
;-----  
; MACRO  
;  
.macro LED_init  
ldi r16, 0xF0  
out DDRB, r16  
out DDRD, r16  
.endmacro  
  
.macro LED_out  
out PORTD, LED  
swap LED  
out PORTB, LED  
swap LED  
.endmacro  
  
.macro delay_1s ;Késleltetés  
ldi r26, 0xFF  
ldi r25, 0xFF  
ldi r24, 0x5F  
delay_loop:  
dec R26 ; lefele számol, ha 0 lesz, akkor Z flag=1  
brne delay_loop ; Z flag-et nézi, ha Z flag=0, akkor ugrik,  
dec R25  
brne delay_loop;  
dec R24  
brne delay_loop  
.endmacro  
;  
;-----  
; SUBRUTINE  
;  
;  
;-----  
; INTERRUPT  
;  
;  
;  
;-----  
; PROGRAM  
;  
;  
start:  
;  
;  
; INIT  
;  
;  
LED_init
```

```
;-----  
; PROGRAM  
;-----  
ldi    LED, 0x18  
ldi    LED1, 0x08  
ldi    LED2, 0x10  
  
loop:  
  
ki:  
    LED_out  
    delay_1s  
    lsr    LED1  
    lsl    LED2  
    eor    LED, LED  
    or     LED, LED1  
    or     LED, LED2  
    cpi    LED, 0x81  
    breq   be  
    brne   ki  
  
be:  
    LED_out  
    delay_1s  
    lsl    LED1  
    lsr    LED2  
    eor    LED, LED  
    or     LED, LED1  
    or     LED, LED2  
    cpi    LED, 0x18  
    breq   ki  
    brne   be  
  
rjmp   loop
```

#### 6.1.5.1.4 AVR C balra-futófény mintakód

```
//-----Run-----  
  
#define F_CPU 16000000UL //órajel frekvencia  
  
#include <avr/io.h>  
#include <util/delay.h>  
  
void Init(void);  
void led_out(unsigned char ertek);  
  
int main(void)  
{  
  
    unsigned char i;  
    Init();  
  
    i=1;  
    while(1)  
    {  
        i=i<<1;  
        led_out(i);  
  
        if(i>=128)  
        {  
            led_out(i);  
            _delay_ms(200);  
            i=1; led_out(i);  
        }  
        _delay_ms(200);  
    }  
  
    return 0;  
}  
  
void led_out(unsigned char ertek)  
{  
  
    PORTD=ertek&0xF0;  
    PORTB=ertek<<4;  
}  
  
void Init(void)  
{  
  
    DDRB=0xF0;  
    DDRD=0xF0;  
}
```

#### 6.1.5.1.5 AVR C ide-oda futófény mintakód

```
//-----Left-Right-----  
  
#define F_CPU 16000000UL          //órakezelő frekvencia  
  
#include <avr/io.h>  
#include <util/delay.h>  
  
void Init(void);  
void led_out(unsigned char ertek);  
  
int main(void)  
{  
  
    unsigned char i, irany=0;  
    Init();  
  
    i=1;  
    while(1)  
    {  
        if(irany==0) {i=i<<1; led_out(i); _delay_ms(100); if(i==128) {irany=1;}}  
        if(irany==1) {i=i>>1; led_out(i); _delay_ms(100); if(i==1) {irany=0;}}  
    }  
  
    return 0;  
}  
  
void led_out(unsigned char ertek)  
{  
  
    PORTD=ertek&0xF0;  
    PORTB=ertek<<4;  
}  
  
void Init(void)  
{  
  
    DDRB=0xF0;  
    DDRD=0xF0;  
}
```

#### 6.1.5.1.6 AVR C Knight Rider futófénny mintakód

```
//-----Knight Rider-----  
#define F_CPU 8000000UL //órajel frekvencia  
  
#include <avr/io.h>  
#include <util/delay.h>  
  
void Init(void);  
void led_out(unsigned char ertek);  
  
int main(void)  
{  
  
    unsigned char i, j, irany=0;  
    Init();  
  
    i=1; j=128;  
    while(1)  
    {  
        if(irany==0) {i=i<<1; j=j>>1; led_out(i|j); _delay_ms(100);}  
        if(i==128) {irany=1;}  
        if(irany==1) {i=i>>1; j=j<<1; led_out(i|j); _delay_ms(100);}  
        if(i==1) {irany=0;}  
    }  
  
    return 0;  
}  
  
void led_out(unsigned char ertek)  
{  
  
    PORTD=ertek&0xF0;  
    PORTB=ertek<<4;  
}  
  
void Init(void)  
{  
  
    DDRB=0xF0;  
    DDRD=0xF0;  
}
```

### 6.1.5.2 Gombkezelés mintakódok

A beviteli eszközök legegyszerűbb példája a nyomógomb. A T-Bird esetében ebből 5 darabtalálkozhatunk. A mintaprogram beolvassa a gombok állapotát, majd a lenyomott gombnak megfelelő számú LED-et (LED-eket) bekapcsolja.

#### 6.1.5.2.1 AVR Assembly gombkezelés mintakód

```
;-----GOMB olvasása-----
INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
.CSEG
.DEF    btn     =      r16
.DEF    LED     =      r17
.ORG 0x0
    rjmp start           ; jump to start
.ORG 0x100
;-----
; MACRO
;-----
.macro init
    ldi   r16, 0xF0
    out  DDRB, r16
    out  DDRD, r16
    ldi   r16, 0x00
    sts  DDRG, r16       ;gombok-hoz tartozó PIN-ek bemenetre állítása
.endmacro

.macro LED_out
    out  PORTD, LED
    swap LED
    out  PORTB, LED
    swap LED
.endmacro
;-----SUBRUTINE
;-----
;-----INTERRUPT
;-----
;-----PROGRAM
;-----
start:
;-----INIT
;-----
init
;-----PROGRAM
;-----
loop:
    lds   btn, PING          ;G port olvasása btn-be
    andi  btn, 0b00011111    ;alsó 5 bit maszkolása (G4-G0)
    mov   LED, btn
    LED_out                  ;értékek kitétele LED-ekre

    rjmp loop
```

#### 6.1.5.2.2 AVR C gombkezelés mintakód

```
//-----Gombok-----
#include <avr/io.h>

void LED_out(unsigned char szam);
void init();

int main()
{
    init();
    while (1)
    {
        LED_out(PING&0x1F);           //gombok beolvasása és kitétele LED-ekre
    }

    return 0;
}

void init()
{
    DDRG=0x00;                      //gomb bemenet
    DDRB=0xF0;                      //LED kimenet
    DDRD=0xF0;
}

void LED_out(unsigned char szam) {
    PORTD = szam & 0xF0;
    PORTB = szam<<4;
}
```

### 6.1.6 T-bird szalagkábel kiosztás

**T-bird3 Ports Pins**

bit	7	6	5	4	3	2	1	0	
PORT									
<b>A</b>	Enable	Dot	Digit Sel1	Digit Sel0	BCD Data3	BCD Data2	BCD Data1	BCD Data0	7 Segments Display
I/O	Out	Out	Out	Out	Out	Out	Out	Out	
<b>B</b>	LED3	LED2	LED1	LEDO	SPI_MISO	SPI莫斯I	SPI_SCK	SPI_SS	LEDs first nibble
I/O	Out	Out	Out	Out					
<b>C</b>	RED	KBrow3	KBrow2	KBrow1	KBrow0	KBcol2	KBcol1	KBcol0	Keyboard
I/O	Out	Out	Out	Out	Out	In	In	In	
<b>D</b>	LED7	LED6	LED5	LED4	TXD1/INT3	RXD1/INT2	SDA/INT1	SCL/INT0	LEDs second nibble
I/O	Out	Out	Out	Out	Out/In	In	Out/In	Out/In	
<b>E</b>	LCD_Data7	LCD_Data6	LCD_Data5	LCD_Data4	GREEN	BLUE	TXD0/PDO	RXD0/PDI	LCD Data
I/O	In/Out	In/Out	In/Out	In/Out	Out	Out	Out	In	
<b>F</b>	TDI	TDO	TMS	TCK	LCD_Enable	LCD_R/W	LCD_RS	LM35	LCD Control
I/O	JTAG	JTAG	JTAG	JTAG	Out	Out	Out	In	Analog
<b>G</b>	NC	NC	NC	K4	K3	K2	K1	K0	Pushbutton
I/O				In	In	In	In	In	

## 6.2 Kiegészítő áramkör: T-Bird – Expansion Board



A T-Bird fejlesztőpanelhez készült kiegészítő board számos lehetőséget foglal magában. A panelen megjelenítésre és bevitelre alkalmas eszközök kerültek elhelyezésre.

### 6.2.1 Megjelenítő eszközök

#### 6.2.1.1 LCD kijelző, 4x16 karakter, HD44780 kompatibilis

Az LCD kijelző 4 adatbites üzemmódban használható, így 3 vezérlő és 4 adat vonallal rendelkezik. Egy karakter kiírása ebből adódóan két írási ciklussal jár.

A port bitekkel történő takarékosság érdekében az LDC kijelző az 4 bites üzemmódban vezérelhető, így a teljes LCD modul vezérléséhez összesen 7 bit került felhasználásra (4 bit adat és 3 bit vezérlőjel). Az LCD modul adatbitjei (D4-D7) a mikrokontroller PF4-PF7 bitjein találhatók. A vezérlőjelek pedig következőképpen kerültek bekötésre:

PF7	PF 6	PF 5	PF 4	PF 3	PF 2	PF 1	PF0
LCD D7	LCD D6	LCD D5	LCD D4	Enable	RW	RS	Hőszenzor

Az LCD kijelző háttérvilágítása a kijelző jobb alsó sarkában található potenciometter használatával szabályozható. Ennek beállítására esetlegesen akkor lehet szükség, ha a kijelzőn semmilyen információt nem látunk. A fényerő maximális értékre történő állítása esetén a 4 sorban az egyes karakterek foltjai fognak látszani, mutatva azt, hogy a kijelző bekapcsolt állapotban van.

#### **Leírások:**

[http://en.wikipedia.org/wiki/HD44780\\_Character\\_LCD](http://en.wikipedia.org/wiki/HD44780_Character_LCD)

<http://lcd-linux.sourceforge.net/pdfdocs/hd44780.pdf>

#### 6.2.1.1.1 AVR Assembly LCD mintakód

```
.include "m128def.inc"

.CSEG

.def    lcd_com      =      r16
.def    lcd_dat      =      r17
.def    tmp          =      r18
.def    tmp2         =      r19
.def    tmp3         =      r20

.org   0x00
        rjmp  start

.org   0x100

.macro      stack_init
    ldi      tmp,  high(RAMEND)
    out      SPH,  tmp
    ldi      tmp,  low(RAMEND)
    out     SPL,  tmp
.endmacro

.macro EN
    call   delays_ms
    in     tmp,  PINF
    ori     tmp,  0b00001000
    sts     PORTF, tmp
    call   delays_ms
    in     tmp,  PINF
    andi   tmp,  0b11110111
    sts     PORTF, tmp
.endmacro

.macro      port_init
    ldi      tmp,  0b11110000
    out     DDRE, tmp
    ldi      tmp,  0b00001110
    sts     DDRF, tmp
.endmacro

lcd_init:
    call   delays_ms
    call   delays_ms
    ldi    lcd_com, 0b00110000
    call   lcd_command
    call   delays_ms
    call   delays_ms
    ldi    lcd_com, 0b00110000
    call   lcd_command
    call   delays_ms
    call   delays_ms
    ldi    lcd_com, 0b00110000
    call   lcd_command
    call   delays_ms
    call   delays_ms
    ldi    lcd_com, 0b00001110
    call   lcd_command
    call   delays_ms
```

```

call  delays_ms
ldi   lcd_com, 0x02
call  lcd_command
ldi   lcd_com, 0x06
call  lcd_command
ldi   lcd_com, 0x01
call  lcd_command
ldi   lcd_com, 0x0C
call  lcd_command
ret

lcd_command:
push  tmp
in    tmp,  PINF
andi  tmp,  0b11111101
sts   PORTF, tmp
mov   tmp,  lcd_com
andi  tmp,  0xf0;
out   PORTE, tmp
EN
swap  lcd_com
mov   tmp,  lcd_com
andi  tmp,  0xf0;
out   PORTE, tmp
EN
swap  lcd_com
pop   tmp
ret

lcd_data:
push  tmp
in    tmp,  PINF
ori   tmp,  0b00000010
sts   PORTF, tmp
mov   tmp,  lcd_dat
andi  tmp,  0xf0;
out   PORTE, tmp
EN
swap  lcd_dat
mov   tmp,  lcd_dat
andi  tmp,  0xf0;
out   PORTE, tmp
EN
swap  lcd_dat
pop   tmp
ret

delays_ms:
eor   tmp2,  tmp2
ldi   tmp3,  0x30
k_d:
dec   tmp2
brne  k_d
dec   tmp3
brne  k_d
ret

start:
stack_init
port_init
call  lcd_init

```

```

ldi    ZH,    high(szoveg<<1)
ldi    ZL,    low(szoveg<<1)
kiir:
lpm   lcd_dat,      Z+
cpi   lcd_dat,      0
breq  loop
call  lcd_data
rjmp  kiir
loop:
rjmp  loop

szoveg: .db "Kis Ibolya", 0

```

### 6.2.1.1.2 AVR C LCD mintakód

```

#define F_CPU 16000000UL
#include <util/delay.h>
#include <inttypes.h>
#include <avr/io.h>

#define LCD_CMD_DDR  (DDRF)
#define LCD_DATA_DDR (DDRE)

#define LCD_CMD_PORT (PORTF)
#define LCD_DATA_PORT (PORTE)
#define LCD_DATA_PIN  (PINE)

#define LCD_RS        (PF1)
#define LCD_RW        (PF2)
#define LCD_EN        (PF3)

#define LCD_E  2      //enable
#define LCD_CUR 1    //cursor
#define LCD_BL 0      //blink

void LCD_init();
void LCD_cmd(uint8_t cmd);
void LCD_clock();
void LCD_data(uint8_t data);

void LCD_Puts (char*s);

int main(void)
{
LCD_init();
LCD_cmd(0x01);
LCD_Puts("Kis Ibolya");

while (1)
{
}
}

```

```

void LCD_init(){

    DDRE |= 0xF0;
    //data7...4 out
    DDRF |= (1<<LCD_RS) | (1<<LCD_RW) | (1<<LCD_EN);      //RS, R/W, EN out
    //írás
    PORTF &= ~(LCD_RW);
    //R/W<-0 ->write
    PORTF &= ~(1<<LCD_RS);
    //RS <-0 ->parancs
    PORTE = 0x20;
    //0x20 parancs: Function set - Sets interface data length
    LCD_clock();
    //delay
    LCD_clock();
    //__| __
    LCD_clock();
    //4 bites üzemmód, 8x5pixel, 2soros
    //üzemmód választás
    //nem tudjuk, hogy bekapcsolás után éppen milyenben van
    LCD_cmd(0x28);
    LCD_cmd(0x28);
    LCD_cmd(0x28);
    //lcd alaphelyzetbe
    LCD_cmd(0x02);
    //lcd törlése
    LCD_cmd(0x01);
    //kurzor villog, aláhúzás be, LCD be
    //LCD_command(0x08 | (1<<LCD_E) | (1<<LCD_CUR) | (1<<LCD_BL));
    //kurzor villog ki, aláhúzás be, LCD be
    //LCD_command(0x08 | (1<<LCD_E) | (1<<LCD_CUR) | (0<<LCD_BL));
    //kurzor villog ki, aláhúzás ki, LCD be
    LCD_cmd(0x08 | (1<<LCD_E) | (0<<LCD_CUR) | (0<<LCD_BL));
}

void LCD_busy(void){                                //BF olvasása
    uint8_t busy;
    LCD_DATA_DDR &= ~(1<<PE7);           //ott olvassuk majd a BF-et (D7-PE7)
    LCD_CMD_PORT &= ~(1<<LCD_RS);        //Státusz infó
    LCD_CMD_PORT |= (1<<LCD_RW);         //olvasás
    do{
        busy = 0;
        LCD_CMD_PORT |= (1<<LCD_EN);      //EN<-1
        _delay_us(1);                      //felfutó
        busy=(LCD_DATA_PIN&(1<<PE7));    //átadjuk a BF értékét
        LCD_CMD_PORT &= ~(1<<LCD_EN);     //EN<-0
        _delay_us(1);
        LCD_CMD_PORT |= (1<<LCD_EN);      //EN<-1
        _delay_us(1);
        LCD_CMD_PORT &= ~(1<<LCD_EN);     //EN<-0
        _delay_us(1);
    }while(busy);
    LCD_CMD_PORT &= ~(1<<LCD_RW);          //R/W<-0 write
    LCD_DATA_DDR |= (1<<PE7);              //PE7<-1
}

void LCD_cmd(uint8_t cmd){
    LCD_busy();                            //Megvárjuk még felszabadul
    LCD_CMD_PORT &= ~(1<<LCD_RS);        //Parancs
    LCD_CMD_PORT &= ~(1<<LCD_RW);        //Küldés
    LCD_CMD_PORT &= ~(1<<LCD_EN);        //EN<-0
}

```

```

LCD_DATA_PORT &= ~(0xF0);
LCD_DATA_PORT |= (cmd&0x0F);           //felső 4 bit küldése
LCD_clock();                          //__| |_
LCD_DATA_PORT &= ~(0xF0);
LCD_DATA_PORT |= ((cmd<<4)&0x0F);    //alsó 4 bit küldése
LCD_clock();                          //__| |_
}

void LCD_data(uint8_t data){
LCD_busy();                         //Megvárjuk még felszabadul
LCD_CMD_PORT |= (1<<LCD_RS);       //Adatregiszter
LCD_CMD_PORT &= ~(1<<LCD_RW);     //írás
LCD_CMD_PORT &= ~(1<<LCD_EN);     //EN<-0

LCD_DATA_PORT &= ~(0xF0);          //4 felső bit kitétele
LCD_DATA_PORT |= (data&0x0F);      //__| |_
LCD_clock();

LCD_DATA_PORT &= ~(0xF0);
LCD_DATA_PORT |= ((data<<4)&0x0F); //alsó 4 bit kitétele
LCD_clock();
}

void LCD_clock(){
LCD_CMD_PORT |= (1<<LCD_EN);      //__| |
_delay_us(2);                      //|
LCD_CMD_PORT &= ~(1<<LCD_EN);    //        |_
_delay_us(2);
}

void LCD_Puts (char*s)
{
while (*s)
{
LCD_data(*s);
s++;
}
}

```

### 6.2.1.2 4db hétszegmenses kijelző

A hétszegmenses kijelzők multiplexerrel, valamint meghajtó áramkörrel vannak ellátva. A meghajtó áramkörnek BCD kódban kell megadni a kiírni kívánt számot, a multiplexernek pedig szintén BCD kódban kell megmondani, hogy melyik kijelzőre szeretnénk kiírni az adott számot. A legkisebb cím a jobboldali kijelzőhöz tartozik (0), majd balra növekszik (egészen háromig). Lehetőség van még két db LED-et is kiválasztani a multiplexerrel (4-es cím), amelyek a hétszegmenses kijelzők között találhatóak meg, így könnyedén kialakítható egy óra.

A 7 szegmenses kijelző vezérlése a mikrokontroller A portján keresztül történik. A kiküldendő információ felépítése a táblázatban található. Adott digitre kiküldendő érték a bájt alsó 4 bitjén (PA0-PA3) lehet megadni binárisan. A négy digit közül, hogy melyikre történjen ezen

információ kiírása, azt a digit választó bitek (PA4-PA5) segítségével lehet kijelölni.

A két-két digit között található LED vezérlése a PA6 bit segítségével valósítható meg. A kijelző engedélyezését a PA7 bit végzi. A kiíratás során ügyelni kell arra, hogy egyszerre csak egy digitre lehet információt kivinni, így, ha folyamatos kiíratást akarunk elérni, akkor a digitekre való kiíratást javasolt kb. 400 Hz-es sűrűséggel ismételni.

A 4 db hétszegmenses kijelző vezérlő bájtja:

PA7	PA 6	PA 5	PA 4	PA 3	PA 2	PA 1	PA0
Hétszegmenses kijelző engedélyezés	Középső LED	Digit választó 1	Digit választó 0	BCD D	BCD C	BCD B	BCD A

A hétszegmenses kijelző digit választása:

Digit választó: 11	Digit választó: 10	Középső LED	Digit választó: 01	Digit választó: 00

A hétszegmenses kijelzőre való kiíratást ugyanúgy, mint a billentyűzet lekezelését is érdemes egy Timer megszakítási rutinban elhelyezni, amely megszakítás a rendszer időalapját is képezheti.

#### Leírások:

[http://www.nxp.com/documents/data\\_sheet/HEF4511B.pdf](http://www.nxp.com/documents/data_sheet/HEF4511B.pdf)  
<http://ics.nxp.com/products/hc/datasheet/74hc238.74hct238.pdf>

#### 6.2.1.2.1 AVR ASM 7szegmenses kijelző mintakód

```
;-----7szegmenses kijelző---1234-----
```

```
.INCLUDE "m128def.inc" ; Include definitions Atmega128
.DSEG
.org 0x100
digit_s: .BYTE 4 ;A 4jegyű szám
.CSEG
.DEF tmp = r16
.DEF LED = r17
.DEF EN = r18
.DEF digit = r19
.DEF szam = r20
.DEF szamjegy = r21
```

```

.ORG 0x0
    rjmp start ; jump to start
.ORG 0x1E ;$001E TIMER0 CTC Timer/Counter0 Compare Match
    rjmp TIM0_CTC_IT
.ORG 0x100
;-----
; MACRO
;-----
.macro init
    ldi    tmp, 0xF0 ;LED-ek
    out   DDRB, tmp
    out   DDRD, tmp
    ldi    tmp, 0xFF ;7szegmenses kijelző, kimenetre állítás
    out   DDRA, tmp
.endmacro

.macro STACK_init ;KÖTELEZŐ!
    ldi  tmp, high(RAMEND)
    out SPH, tmp
    ldi  tmp, low(RAMEND)
    out SPL, tmp
.endmacro

.macro TIM0_CTC_init ;TIM0 CTC init
    ldi  tmp, 0b00001111 ;1024-es előosztás beállítása

    out  TCCR0, tmp ;Timer/Counter Control Register - TCCR0
                ;7 6 5 4 3 2 1 0
                ;FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00
    ldi  tmp, 0x02 ;Timer/Counter0 Overflow Interrupt Enable
    out  TIMSK, tmp ;Timer/Counter Interrupt Mask Register - TIMSK
                ;7 6 5 4 3 2 1 0
                ;          OCIE0 TOIE0
    ldi  tmp, 30 ;16000000/1024/30 8000000/1024/30 ~520Hz
    out  OCR0, tmp
    sei           ;Sets the Global Interrupt flag (I) in SREG
.endmacro
;-----
; SUBROUTINE
;-----
sub_7seg_digit:
    ldi  EN, 0x80 ;0x80|digit|szám
    eor  tmp, tmp
    or   tmp, EN ; engedélyezés
    swap digit
    or   tmp, digit ; digit 0-3
    swap digit
    or   tmp, szam ; szám 0-9
    out  PORTA, tmp
ret

sub_7seg:
    cpi  digit, 4 ;összehasonlítás
    breq nullaz

kiir:
    ld   szam, Z+ ;memóriából érték betöltése
    call sub_7seg_digit ;megjelenítés az aktuális digiten
    inc  digit ;digit növelése
    rjmp vege

```

```

nullaz:                                ;digit nullázása
    eor    digit, digit
    ldi    ZL, low(digit_s)      ;visszaállítás a memória címre 0x100
    ldi    ZH, high(digit_s)

vege:
ret
;-----
; INTERRUPT
;-----
TIM0_CTC_IT:
    call   sub_7seg
reti
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
init
STACK_init
TIM0_CTC_init
;-----
; PROGRAM
;-----
ldi    ZL, low(digit_s)      ;digit_s kezdőcímére visszaállítás
ldi    ZH, high(digit_s)
eor    digit, digit          ;törölés
eor    szam, szam
ldi    szamjegy, 4

tolt:
    mov    tmp, szamjegy        ;1234 betöltése
    st     Z+, tmp
    dec   szamjegy
    brne  tolt

ldi    ZL, low(digit_s)      ;digit_s kezdőcímére visszaállítás
ldi    ZH, high(digit_s)

loop:
    rjmp  loop

```

#### 6.2.1.2.2 AVR C 7szegmenses kijelző mintakód

```

/*-----7szegmenses kijelző-----*/
#include <avr/io.h>
#include <avr/interrupt.h>

void init();

void SEVSEG_putdigit(uint8_t szam, uint8_t digit);
void SEVSEG_putnumber (int val);

uint8_t digit[4]={0};
int j=0;

```

```

int main(void)
{
    init();
    // SEVSEG_putdigit(0,0);
    // SEVSEG_putdigit(1,1);
    // SEVSEG_putdigit(2,2);
    // SEVSEG_putdigit(3,3);

    while (1)
    {

    }

    void init()
    {
        DDRA=0xFF;                                //kimenetre állítás
        //timer0 setup
        TCCR0 = (1<<CS01)| (1<<CS00);
        TIMSK=(1<<TOIE0);
        sei();
    }

    ISR(TIMER0_OVF_vect)
    {
        SEVSEG_putnumber (1234);
    }

    void SEVSEG_putdigit(uint8_t szam, uint8_t digit)
    {
        if (digit>3) return;
        if (szam>9) return;
        PORTA=0x80 | (digit)<<4 | szam;      //kiíratás 0x80=enable;
        //PA7 engedélyezés
        //PA6 :
        //                           digit3          digit2          digit1          digit0
        //PA5-4 digit választó:   11             10             01             00
    }

    void SEVSEG_putnumber (int val)
    {
        digit[0]=val%10;                          //előállítás digitenként
        digit[1]=(val/10)%10;
        digit[2]=(val/100)%10;
        digit[3]=(val/1000)%10;
        j = (j+1)%4;                            //folyamatos kiírás
        SEVSEG_putdigit(digit[j], j);
    }
}

```

#### 6.2.1.3 3 színű LED

A panelen megtalálható egy 3 színű LED, amelyet például 3 szoftveres PWM jelrel meg hajtva könnyedén előállítható 256 (4 bites PWM) színkombináció vagy akár 16 Millió (8 bites PWM). A háromszínű LED vezérlése a PC7 (RED), a PE2 (GREEN) és a PE3 (BLUE) port biteken keresztül történhet. A vezérlés során lehetőség van a LED-ek fényerejének változtatására is PWM jel segítségével. A három szín megfelelő arányú keverésével tetszőleges szín előállítható, illetve ezen szín fényereje is állítható.

### 6.2.1.3.1 AVR ASM RGB mintakód

```

;-----RGB színkeverés-----
INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF  pwm1      =    r16
.DEF  p2        =    r17
.DEF  pwm3      =    r18
.DEF  szamlalo  =    r19
.DEF  tmp        =    r20

.ORG 0x0
rjmp start                   ; jump to start

.ORG 0x20                  ;$0020 TIMER0 OVF Timer/Counter0 Overflow
rjmp TIM0_OVF_IT
.ORG 0x100
;-----
; MACRO
;-----
.macro STACK_init
ldi   tmp, HIGH(RAMEND)
out  SPH, tmp
ldi   tmp, LOW(RAMEND)
out  SPL, tmp
.endmacro

.macro RGB_init
ldi   tmp, 0b10000000
out  DDRC, tmp
ldi   tmp, 0b000001100
out  DDRE, tmp
.endmacro

.macro TIM0_OVF_init          ;TIM0 OVF init
ldi   tmp, 0b00000001          ;előosztás beállítása (nincs)
out  TCCR0, tmp               ;Timer/Counter Control Register - TCCR0
                           ;7 6 5 4 3 2 1 0
                           ;FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00
ldi   tmp, 0x01                ;Timer/Counter0 Overflow Interrupt Enable

out  TIMSK, tmp               ;Timer/Counter Interrupt Mask Register - TIMSK
                           ;7 6 5 4 3 2 1 0
                           ;          OCIE0 TOIE0
sei                         ;Sets the Global Interrupt flag (I) in SREG
.endmacro
;-----
; SUBROUTINE
;-----
;-----INTERRUPT-----
;-----TIM0_OVF_IT:          ;TIM0 OVF megszakítást kiszolgáló rutin
inc  szamlalo
brcc vege
eor  szamlalo, szamlalo
vege:
reti

```

```

;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
STACK_init
RGB_init
TIM0_OVF_init
;-----
; PROGRAM
;-----
eor    tmp, tmp
eor    szamlalo, szamlalo
;világos kékeszöld
ldi    pwm1, 226
ldi    p2, 107
ldi    pwm3, 53

loop:
rgb_p_be:           ;RGB R bekapcsolása
cp    pwm1, szamlalo
brlo  rgb_p_ki       ;ha kisebb a számlálónál, akkor kikapcsoljuk
ldi    tmp, 0b10000000 ;különben bekapcsoljuk
out   PORTC, tmp
rjmp  rgb_z_be

rgb_p_ki:           ;RGB R kikapcsolása
ldi    tmp, 0b00000000
out   PORTC, tmp

rgb_z_be:           ;RGB Z bekapcsolása
cp    p2, szamlalo
brlo  rgb_z_ki
in    tmp, PINE
ori   tmp, 0b00000100
out   PORTE, tmp
rjmp  rgb_k_be

rgb_z_ki:           ;RGB Z kikapcsolása
in    tmp, PINE
andi  tmp, 0b00001000
out   PORTE, tmp

rgb_k_be:           ;RGB K bekapcsolása
cp    pwm3, szamlalo
brlo  rgb_k_ki
in    tmp, PINE
ori   tmp, 0b00001000
out   PORTE, tmp
rjmp  loop

rgb_k_ki:           ;RGB K kikapcsolása
in    tmp, PINE
andi  tmp, 0b00000100
out   PORTE, tmp

rjmp  loop

```

#### 6.2.1.3.2 AVR C RGB mintakód

```
/*-----RGB színkeverés-----*/  
  
#include <avr/io.h>  
#include <inttypes.h>  
#include <avr/interrupt.h>  
  
//Világos kékészöld  
uint8_t pwm1=29;  
uint8_t pwm2=148;  
uint8_t pwm3=202;  
uint8_t szamlalo=0;  
  
void Timer0Init() {  
    TCCR0 = (0<<CS02) | (0<<CS01) | (1<<CS00); //nincs osztás  
    TIMSK |= (1<< TOIE0); //OVF interrupt enable  
    sei(); //Globális interruptok engedélyezése  
}  
  
int main() {  
    // PC7 (RED), a PE2 (GREEN) és a PE3 (BLUE)  
    DDRC = (1<<PC7); //RGB beállítása  
    DDRE = (1<<PE2) | (1<<PE3);  
  
    Timer0Init(); //Timer beállítása  
  
    while(1) {  
        if (pwm1>=szamlalo){  
            PORTC|=(1<<PC7);  
        }  
        else  
            PORTC&=(0<<PC7);  
        if (pwm2>=szamlalo){  
            PORTE|=(1<<PE2);  
        }  
        else  
            PORTE&=(0<<PE2);  
        if (pwm3>=szamlalo){  
            PORTE|=(1<<PE3);  
        }  
        else  
            PORTE&=(0<<PE3);  
    }  
}  
ISR(TIMER0_OVF_vect) { //Timer0 overflow interrupt  
    szamlalo++;  
    if (szamlalo==255){  
        szamlalo=0;  
    }  
}
```

## 6.2.2 Bemeneti eszközök

### 6.2.2.1 3x4-es billentyűzet mátrix

A billentyűzet mátrix soraira aktív jelet adva (időben egyszerre minden csak egyre), és az oszlop jeleket beolvasva könnyedén megállapítható, hogy adott pillanatban melyik gomb került lenyomásra.

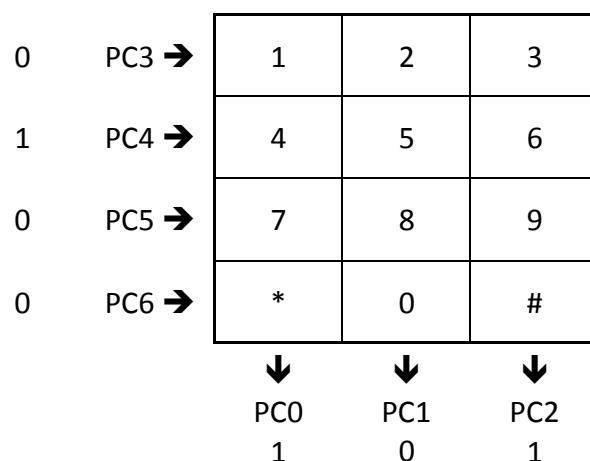
A billentyűzet mátrix megcímzése a mikrokontroller PC3-PC6 port bitjein keresztül lehetséges. A kiválasztani kívánt sorra logikai 1-et, a másik három címző vezetékre logikai 0-t adva, az adott sorban levő lenyomott billentyűzet értékét tudjuk vissza olvasni a PC0-PC2 port vezetékeken keresztül.

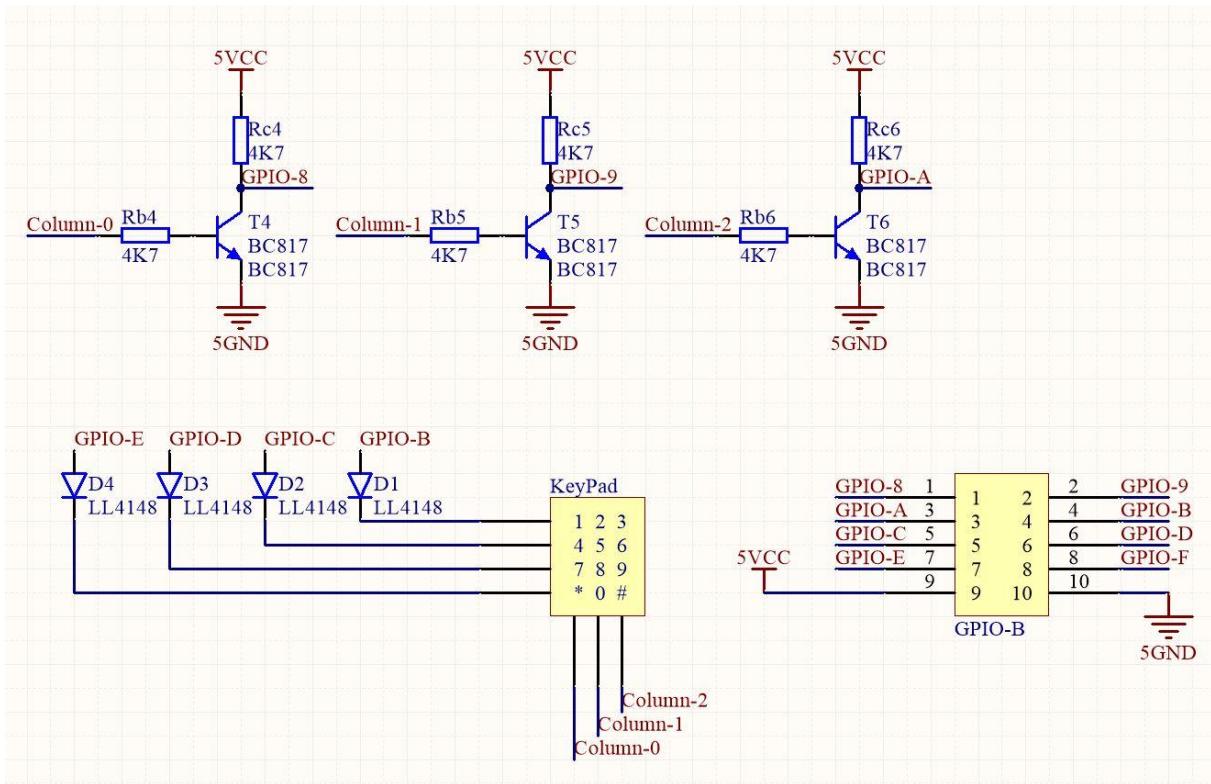
Ciklikusan végig címezve a többi vezetéket, az éppen megcímzett sorokban is vizsgálhatjuk a lenyomott billentyűket. A ciklikusság gyakoriságát érdemes olyan sűrűre választani, hogy az egyes billentyűk pergése már ne okozzon problémát, de ne is kelljen túl sokáig nyomni a gombot.

PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
RGB LED RED	Billentyű- zet címző D	Billentyű- zet címző C	Billentyű- zet címző B	Billentyű- zet címző A	Billentyű- zet olvasó C	Billentyű- zet olvasó B	Billentyű- zet olvasó A

**Példa:**

PC3-PC6-ra küldjünk ki 0010 értéket – ez a második sort címzi meg – és olvassuk vissza a PC0-PC2 vonalakat. Ha a visszaolvasott érték maszkolás után 101, akkor a középső oszlopból levő billentyű került lenyomásra, a második sor esetén az 5-ös billentyű.





### Olvasóvezetékek

GPIO-8	PC0
GPIO-9	PC1
GPIO-A	PC2

### Címzővezetékek

GPIO-B	PC3
GPIO-C	PC4
GPIO-D	PC5
GPIO-E	PC6

### 6.2.2.1.1 AVR Assembly mátrix billentyűzet mintakód

```

;-----Mátrix-----
.INCLUDE "m128def.inc" ; Include definitions Atmega128
.DSEG
.CSEG
.DEF keyb_val = r5
.DEF keyb_but = r6
.DEF keyb_num = r7
.DEF tmp = r16
.DEF tmp2 = r17
.DEF tmp3 = r18
.DEF tmp4 = r19
.DEF tmp5 = r21
.DEF keyb_row = r22
.DEF szam = r23
.DEF digit = r24
.ORG 0x0
    njmp start ; jump to start
.ORG 0x20 ;$0020 TIMER0 OVF Timer/Counter0 Overflow
    njmp TIM0_OVF_IT
.ORG 0x100
;-----
; MACRO
;-----
.macro PORT_init
    in    tmp,DDRA ; 7 szegmenses kijelző
    ori   tmp,0xff
    out   DDRA, tmp
    in    tmp,DDRC ; Mátrix billentyűzet
    ori   tmp,0xF8
    out   DDRC,tmp
.endmacro

.macro STACK_init
    ldi   tmp, HIGH(RAMEND)
    out   SPH, tmp
    ldi   tmp, LOW(RAMEND)
    out   SPL, tmp
.endmacro

.macro TIM0_OVF_init ;TIM0 OVF init
    ldi   tmp, 0b000000111 ;1024-es előosztás beállítása
    out   TCCR0, tmp ;Timer/Counter Control Register - TCCR0
                ;7 6 5 4 3 2 1 0
                ;FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00
    ldi   tmp, 0x01 ;Timer/Counter0 Overflow Interrupt Enable
    out   TIMSK, tmp ;7 6 5 4 3 2 1 0
    sei   ;OCIE0 TOIE0
.endmacro ;Sets the Global Interrupt flag (I) in SREG
;-----
; SUBROUTINE
;-----
sub_7seg_digit:
    eor   tmp, tmp ; engedélyezés
    ori   tmp, 0x80
    swap  digit
    or    tmp, digit ; digit 0-3
    swap  digit
    or    tmp, szam ; szám 0-9
    out   PORTA, tmp
ret

```

```

sub_matrix_olvas_kiir:
keyb_init:
    ldi    keyb_row, 0x08          ;1. sor
keyb_init_1:
    ldi    ZH, HIGH(keyb_nums<<1) ;számok beolvasása
    ldi    ZL, LOW(keyb_nums<<1) ;összehasonlítás
    cpi   keyb_row, 0x80          ;címzés kezdése ismét az 1. sornál
    breq  keyb_init              ;Sor címzése
    out   PORTC, keyb_row        ;PORT C olvasása
    in    keyb_but, PINC         ;Következő sor
    lsl   keyb_row               ;12 alapérték, ha nincs egyezés
    ldi   tmp2, 12
back_keyb:
    lpm   keyb_val, Z+           ;beolvasás, majd egyet lépteti az indexet
    cp    keyb_val, keyb_but     ;Egyezés keresése a beolvasott értékkel
    breq  find_keyb             ;Egyezés esetén
    inc   ZL                     ;Cím növelése
    dec   tmp2                  ;Végig nézzük az összes értéket
    brne back_keyb
    jmp   keyb_init_1
find_keyb:
    lpm   szam, Z               ;Lenyomott gomb számának kitétele
    eor   digit, digit          ;A 7seg kijelzőre, 0. digit
    call  sub_7seg_digit
ret
;-----
; INTERRUPT
;-----
TIM0_OVF_IT:
    call  sub_matrix_olvas_kiir
reti
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
PORT_init
STACK_init
TIM0_OVF_init
;-----
; PROGRAM
;-----

loop:
    rjmp  loop

keyb_nums: .DB
69,0,14,1,13,2,11,3,22,4,21,5,19,6,38,7,37,8,35,9,67,10,70,11
; DB-Define constant byte(s) in program memory or E2PROM memory

```

#### 6.2.2.1.2 AVR C mátrix billentyűzet mintakód

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

void init();
void SEVSEG_putdigit(uint8_t szam, uint8_t digit);
void beolvas_kiir();
uint8_t m_button = 0;
uint8_t row = 0x08;

int main(void)
{
    init();

    while (1)
    {

    }

}

void init()
{
    DDRA=0xFF;                      //7seg kimenetre állítás
    //timer0 setup
    DDRC = 0x78;                    //mátrix
    TCCR0 = (1<<CS01) | (1<<CS00);
    TIMSK=(1<<TOIE0);
    sei();
}

ISR(TIMER0_OVF_vect)
{
    beolvas_kiir();
}

void SEVSEG_putdigit(uint8_t szam, uint8_t digit)
{
    PORTA=0x80 | (digit)<<4 | szam; //kiíratás 0x80=enable;
}

void matrix()
{
    const unsigned char billtomb[12] =
        { 69 , 14, 13, 11, 22, 21, 19, 38, 37, 35, 70, 67 };
        //0,   1,   2,   3,   4,   5,   6,   7,   8,   9,   #,   *
    unsigned char num, bill;

    PORTC = row;                  //címzés           //kezdőcím: 0x08, első sor
    _delay_ms( 5 );              //várakozás
    bill = PINC & 0x7f;          //maszkolás 0b01111111
    num = 0;                     //tömb index törlése

    while( num<12 )
    {
        if( bill == billtomb[num] ){ //tömb elemek ellenőrzése
            m_button = num;       //ha egyezés van kiküldjük a számot
            while(PINC == billtomb[num] );
            break;                //folyamat befejezése
        }
    }
}
```

```

        else{
            m_button = 12;
            num++;
        }
    }
    if( row == 0x40 )row = 0x08;           //sorcímzés léptetése
    else row = row << 1;
}

void beolvas_kiir()
{
    matrix();
    if (m_button<=9)
    {
        SEVSEG_putdigit(m_button, 0);
    }
}

```

#### 6.2.2.2 LM35 hőszenzor

A hőmérséklet érzékelő szenzor kimenete a mikrokontroller analóg bemenetére van kötve. A kontroller belső A/D átalakítóját használva lehet digitalizálni, majd egy skálatényezővel megmondani, hogy mekkora a hőmérséklet értéke.

Az LM35-ös hőmérsékletérzékelő a mikrokontroller PF0 analóg bemenetére kapcsolódik, amely analóg jelet a belső referencia tápfeszültség felhasználásával vizsgálhatjuk meg.

**Leírás:**

<http://www.national.com/ds/LM/LM35.pdf>

Az LM35, A/D átalakító használata túlmutat a Programozás II. tárgy keretein, ezért mintakód sem kerül ismertetésre.

### 6.2.3 Csatlakozó kiosztás

A kiegészítő board teljes egészében illeszkedik a T-Bird csatlakozó kiosztásához. A lenti táblázatban található meg a csatlakozók kiosztása:

Csatlakozó	[lábszám]	I/O	Funkció
<b>GPIO-A</b>	[1]	Output	7seg. meghajtó BCD kód „A”
	[2]	Output	7seg. meghajtó BCD kód „B”
	[3]	Output	7seg. meghajtó BCD kód „C”
	[4]	Output	7seg. meghajtó BCD kód „D”
	[5]	Output	7seg. demux „A”
	[6]	Output	7seg. demux „B”
	[7]	Output	7seg. demux „C”
	[8]	Output	7seg. demux enable
	[9]	Power	5V
	[10]	Power	GND
<b>GPIO-B</b>	[1]	Input	Bill. mátrix bal oldali oszlop
	[2]	Input	Bill. mátrix középső oszlop
	[3]	Input	Bill. mátrix jobb oldali oszlop
	[4]	Output	Bill. mátrix első sor
	[5]	Output	Bill. mátrix második sor
	[6]	Output	Bill. mátrix harmadik sor
	[7]	Output	Bill. mátrix negyedik sor
	[8]	Output	RGB LED „Red”
	[9]	Power	5V
	[10]	Power	GND
<b>Timer/Analog</b>	[1]	Input	Hőszenzor, analóg bemenet
	[2]	Output	LCD RS
	[3]	Output	LCD R/W
	[4]	Output	LCD Enable
	[5]	I/O	LCD data[4]
	[6]	I/O	LCD data[5]
	[7]	I/O	LCD data[6]
	[8]	I/O	LCD data[7]
	[9]	Output	RGB LED „Blue”
	[10]	Output	RGB LED „Green”
	[11]	Power	5V
	[12]	Power	GND
	[13]	Power	5V
	[14]	Power	GND

T-bird	Csatlakozó	Kiegészítő panel
	<b>GPIO-A</b>	
PA3-PA0	4-1	BCD kód a kiválasztott digitre
PA5-PA4	6-5	digit választó
PA6	7	digitek közötti középső pontok
PA7	8	demux enable
	<b>GPIO-B</b>	
PC2-PC0	3-1	Billentyűzetmátrix olvasása
PC6-PC3	7-4	Billentyűzetmátrix címzése (normál)
PC7	8	RGB LED RED
	<b>Timer-Analog</b>	
PF0	1	Hőszenzor
PF1	2	LCD RS
PF2	3	LCD RW
PF3	4	LCD Enable
PF7-PF4	8-5	LCD data 7-4
PE3	9	RGB LED BLUE
PE2	10	RGB LED GREEN
	<b>TWI</b>	
PD0	SCL	
PD1	SDA	
	<b>USART</b>	
PD2	RX	
PD3	TX	
	<b>SPI</b>	
PB0	SS	
PB1	SCK	
PB2	MOSI	
PB3	MISO	

# Programozás II. laboratórium AVR Assembly

## **Labor célja**

Az AVR Assembly utasításainak bemutatása Atmega mikrokontroller család segítségével, illetve megismertetni az Assembly nyelven történő program tervezés és megvalósítás technikáit. minden laboron 4 résztémakör kerül érintésre, amelyek feldolgozandó és elsajátítandó tananyag jól elkülöníthető egységeit képezik, illetve a következő laboron számonkérésre is kerülnek. A számonkérésre való felkészülést az útmutatóban kidolgozott és laboron megoldott önálló feladatokon túl az 5. részben található Gyakorló feladatok is segítik. A labor tematikája felsorolásban található oldalszámok ezen jegyzet adott laborhoz tartozó elméleti ismeretanyagait tartalmazzák. Érdemes ezen oldalakon levő ismereteket a labor anyag áttekintése előtt megtanulni vagy átismételni.

Sikeres tanulást kívánnak

a Szerzők.

# 1. labor

## Labor tematikája:

1. AVR Studio 4 használata, assembly nyelvi forráskód felépítése (15-25. oldal)
2. Alapvető utasítások megismerése, Debugger (26-39. oldal)
3. Portkezelés alapjai: LED-ek (91. oldal, 21-26. oldal)
4. Makrók

## 1 Bevezetés

### 1.1 Linkek, letöltések

MAI WIKI – Programozás II. laboratórium:

[http://wiki.mai.kvk.uni-obuda.hu/category/programozas\\_2\\_laboratorium/](http://wiki.mai.kvk.uni-obuda.hu/category/programozas_2_laboratorium/)

Fejlesztői környezet:

<http://wiki.mai.kvk.uni-obuda.hu/avr-fejlesztokornyezet-telepitese-64-bites-windows-rendszerekre/>

T-Bird portkiosztás:

<http://wiki.mai.kvk.uni-obuda.hu/t-bird-portkiosztas/>

AVR Assembler:

<https://www.microchip.com/webdoc/avr assembler/index.html>

ATmega128 adatlap:

<http://ww1.microchip.com/downloads/en/DeviceDoc/doc2467.pdf>  
<http://ww1.microchip.com/downloads/en/DeviceDoc/2467S.pdf>

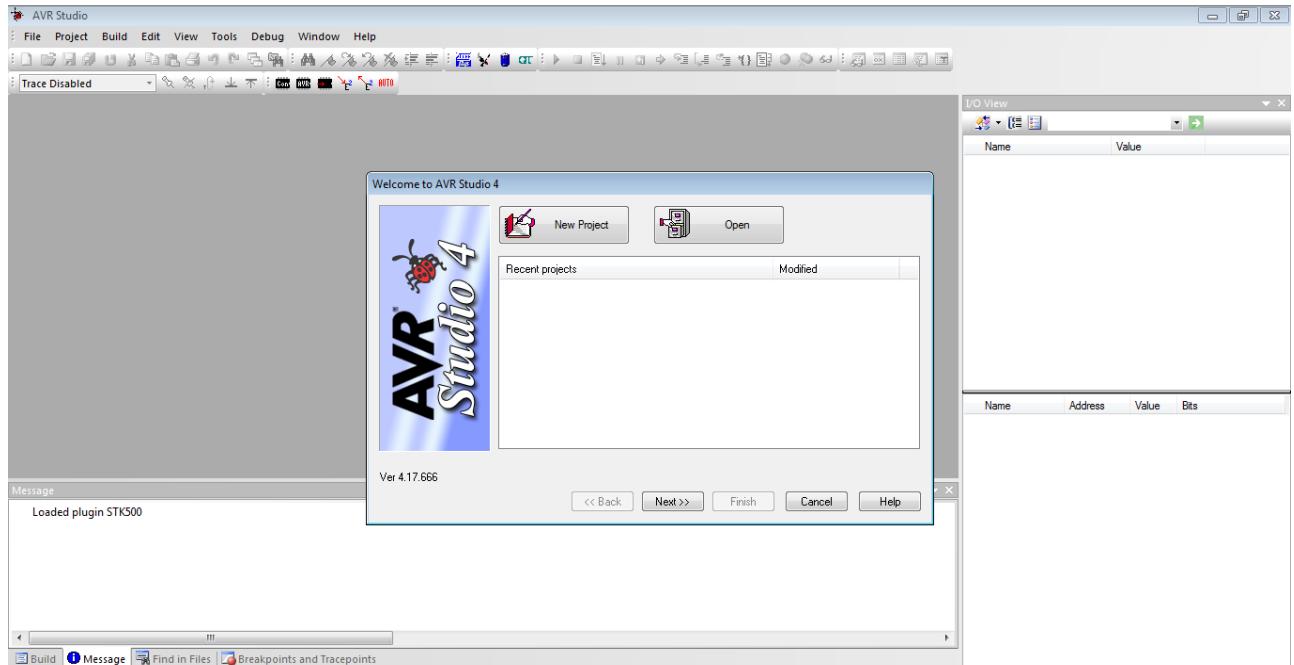
ATmega64 adatlap:

[http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2490-8-bit-AVR-Microcontroller-ATmega64-L\\_datasheet.pdf](http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2490-8-bit-AVR-Microcontroller-ATmega64-L_datasheet.pdf)  
[http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2490-8-bit-AVR-Microcontroller-ATmega64-L\\_summary.pdf](http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2490-8-bit-AVR-Microcontroller-ATmega64-L_summary.pdf)

## 1.2 AVR Studio 4 használata

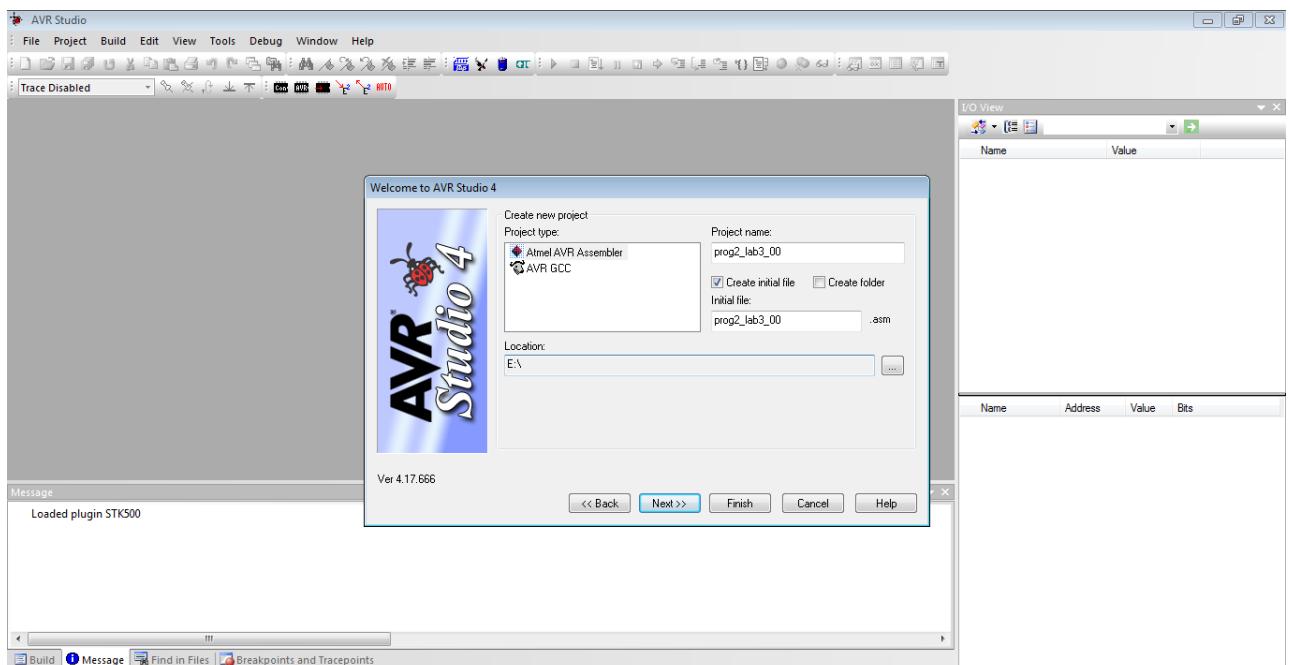
### 1.2.1 Új projekt létrehozása

Az AVR Studio 4-et megnyitva egy üdvözlő ablak jelenik meg. A **New Project** gombra kattintva tudunk elindítani az új projekt létrehozását. Amennyiben kódolás közben szeretnénk egy újabb projektet létrehozni, azt a **Project** menü **New Project** opciójával tehetjük meg.

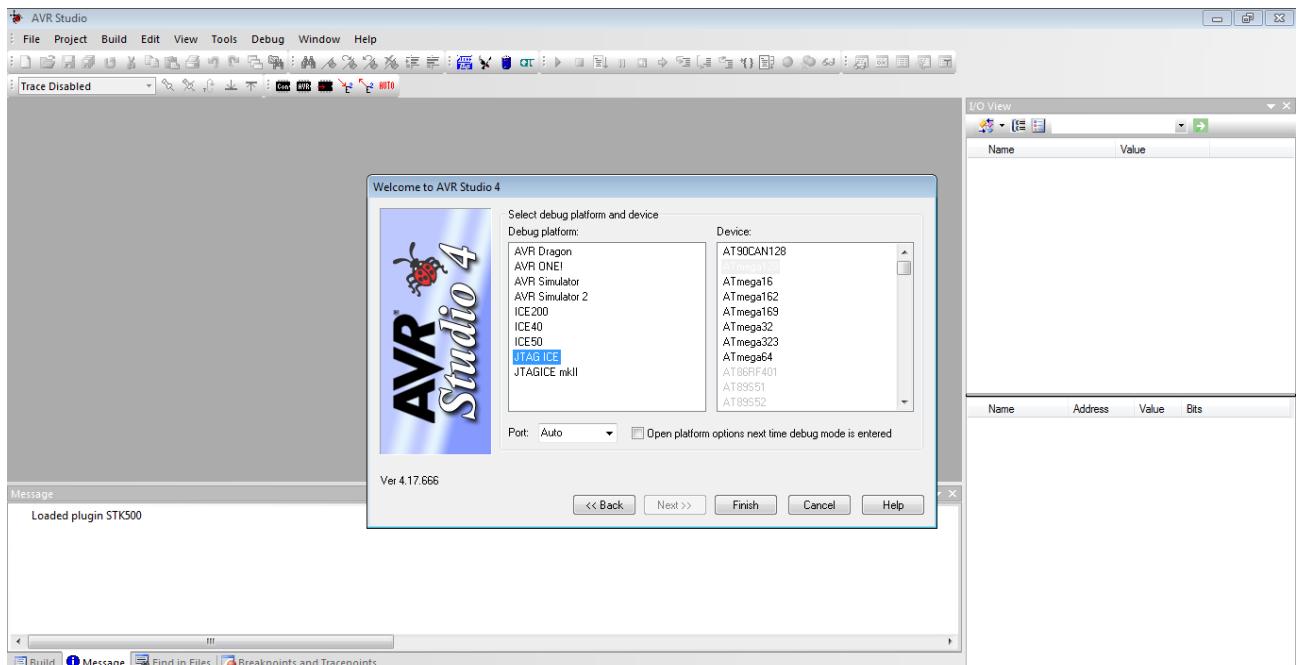


A következő megjelenő ablakban kell kiválasztani a projekt típusát. Assembly esetén az **Atmel AVR Assembler** választandó, C kód esetén pedig az **AVR GCC**.

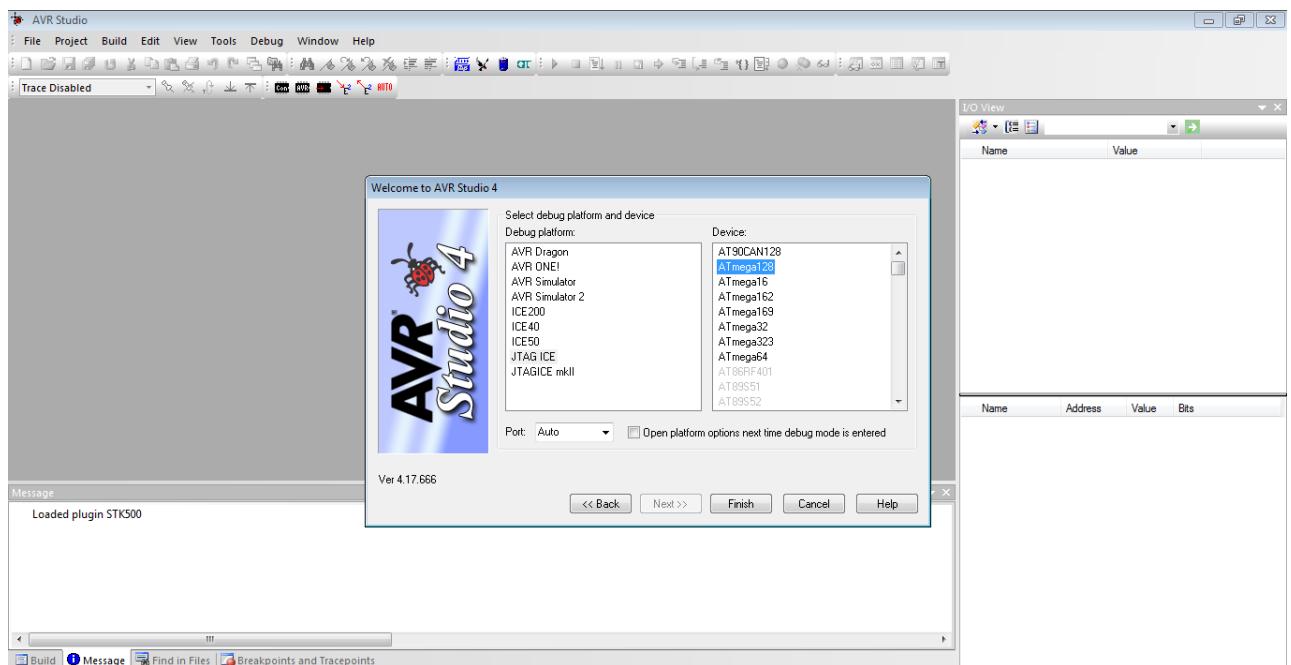
Meg kell adni a projekt nevét (**Project Name**) és a helyét (**Location**) is. Itt arra kell figyelni, hogy ezek egyike se tartalmazzon ékezetes vagy speciális karaktert. Ezután a **Next** gombra kattintva folythatunk a projekt készítésével.



Ezután a **Debug platformot** kell kiválasztani. Amennyiben csak szimulációt szeretnénk használni, akkor az **AVR Simulator-t** kell választani, különben a **JTAG ICE**.



Az eszköz az ablak jobb oldalán választható ki: **ATmega128**, **ATmega64**. Az eszköz kiválasztása után a **Finish** gombra kattintva létrejön az új projektünk.



### 1.2.2 Fordítás

A kódunk megírása után lefordíthatjuk azt. Ez menüből is elérhető, de a legegyszerűbb módja, ha az ábrán látható két kék nyilas gomb egyikére kattintunk. Az egyik opción az **Assemble and run** (fordítás és futtatás), amely a *Ctrl+F7* billentyűkombinációval is elérhető. A másik opción az **Assemble** (fordítás), mely az *F7* billentyűt lenyomva is elérhető.

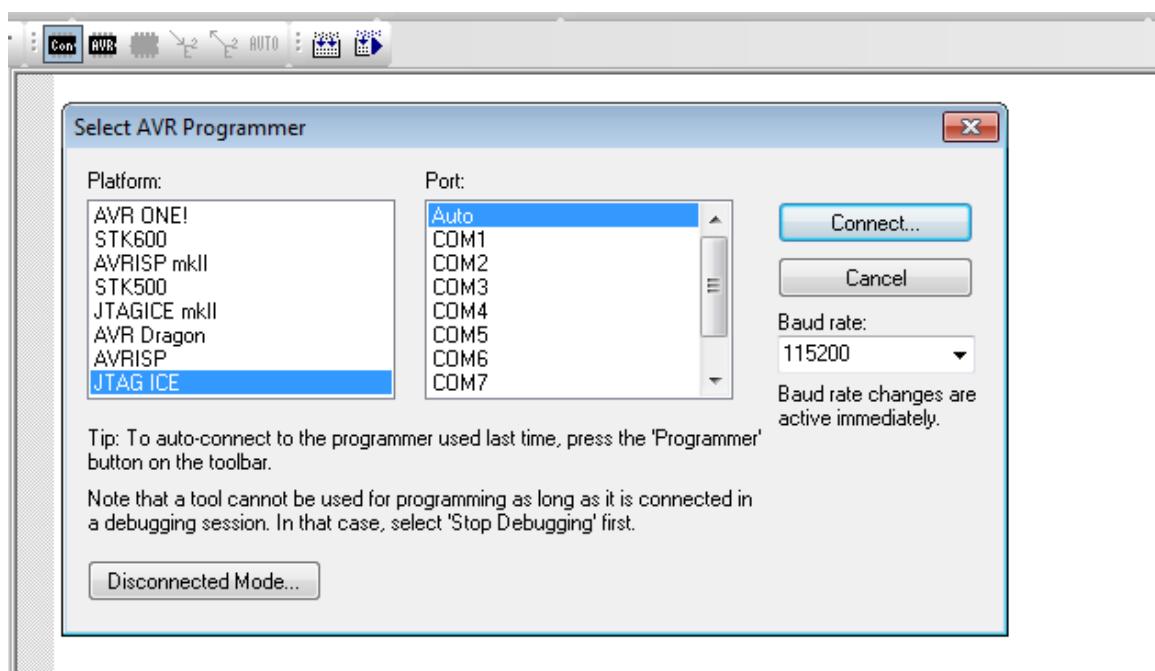


```
.include "m128def.inc"
.CSEG
.org 0x00
jmp start
.org 0x100

start:
        rjmp start
```

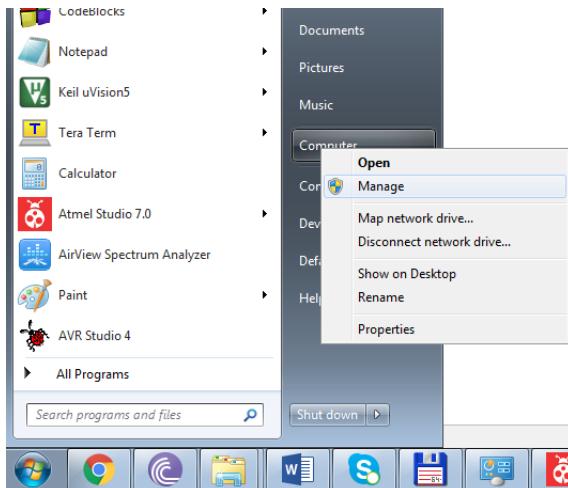
### 1.2.3 Programozó beállítása

Az eszközünk programozása előtt a számítógéppel csatlakozni kell hozzá. Ezt a **Con** feliratú fekete gombbal tehetjük meg. (**Display the 'Connect' dialog**). Bal oldalon a **JTAG ICE** platformot kell kiválasztani, a jobb oldalon a Port választására van lehetőség. Amennyiben nem tudjuk, hogy melyik **COM1-9** porton található az eszközünk, választhatjuk az **Auto** csatlakozási módot. A **Connect...** gombra kattintva indul meg a csatlakozás.

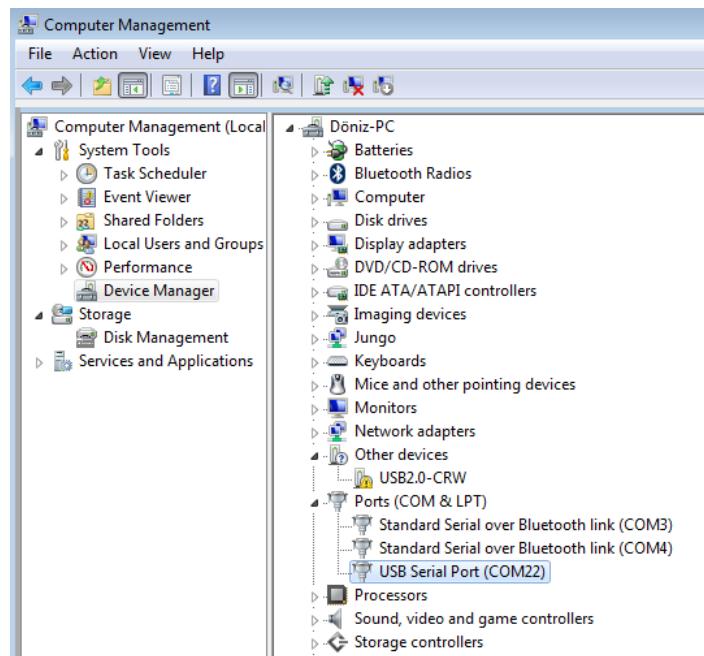


Ha nem csatlakozik Auto üzemmódban, akkor valószínűleg 9-nél nagyobb COM port-on van az eszköz. Ebben az esetben COM1-9 valamelyikére kell átállítanunk. (Sok esetben a az újból csatlakoztatás is megoldja a problémát.) Ezt a következőképen tehetjük meg. Az eszközkezelőbe kell belépní (**Computer**-

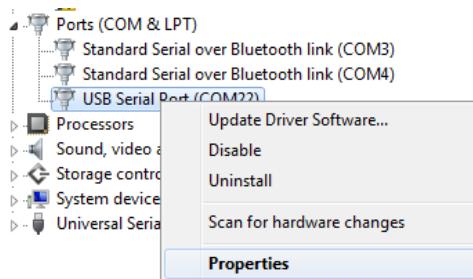
**Manage-Device Manager).**



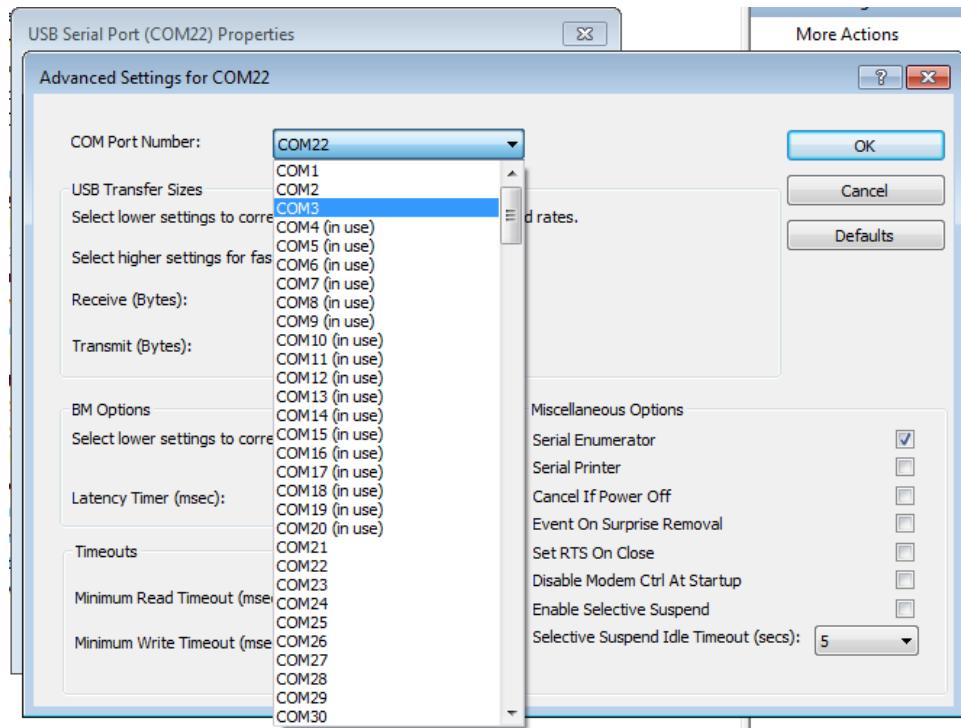
Az eszközkezelőben találjuk a Portok alatt a fejlesztőeszközünket. Az ábra alapján a COM22-n.



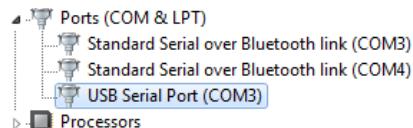
Jobb egérgombbal kattintva érjük el a beállítási lehetőségeket.



Itt lesz lehetőségünk a Port számának átállítására. A port számának 1-9 választandó, lehetőleg olyan, ami nincs használatban (választhatunk már korábban foglalt portot is az 1-9 tartományban, mert a Windows akkor is foglaltnak jelzi, ha korábban használtuk azt a portot, de azóta már nem használjuk).

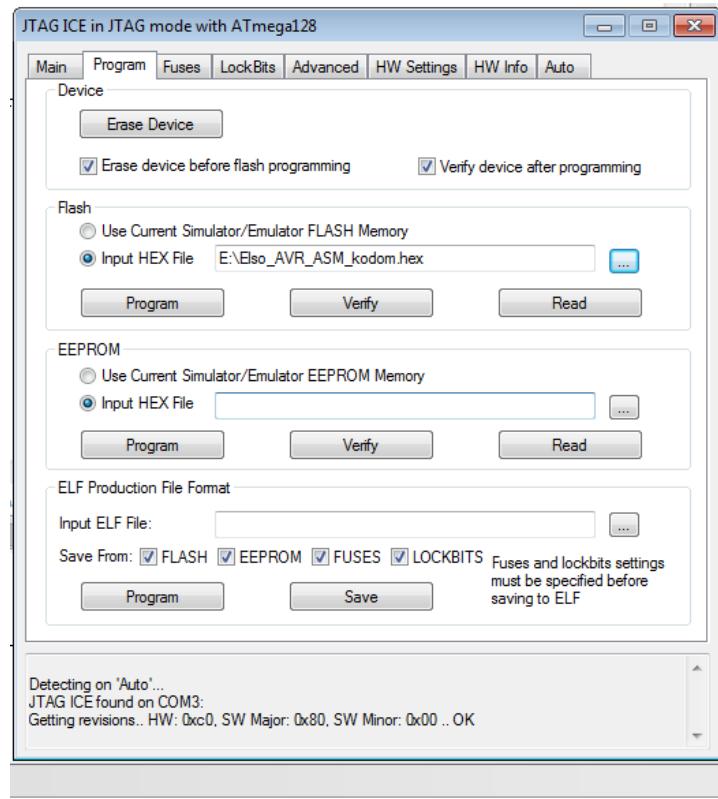


Jelen esetben a COM3 lett beállítva. Így már tudunk csatlakozni az eszközünkhöz.

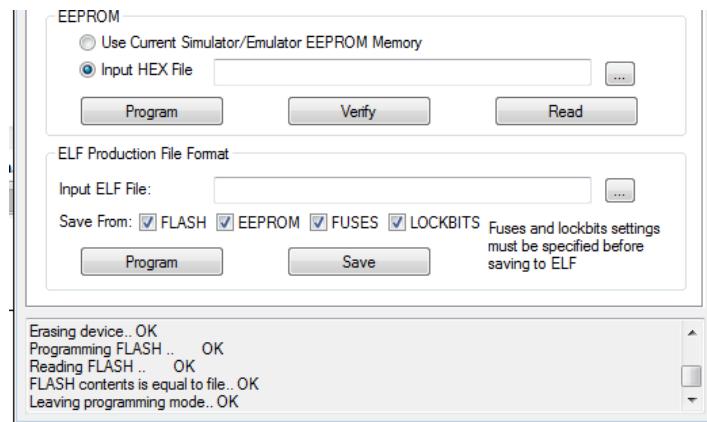


#### 1.2.4 Programozás

Sikeres csatlakozás után megjelenik a programozói ablak. Meg kell adni a program által létrehozott **.hex** file elérési útját. Ha ez megtörtént, akkor a **Program** gombra elindíthatjuk a felprogramozást.



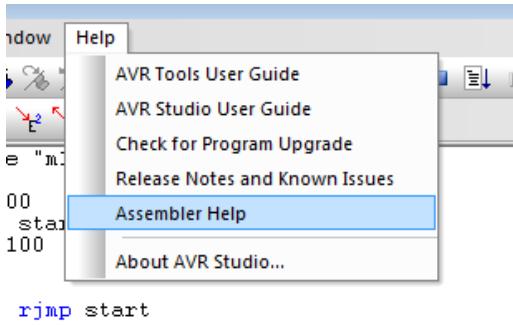
Sikerességek esetén a programozás ablak alsó információs részében csupa OK üzenetet kell kapunk.



Ha ezt az ablakot nem zárjuk be, akkor a továbbiakban a programozás egy kattintással végezhető, az előzetesen megadott beállítások mellett.



## 1.2.5 Súgó

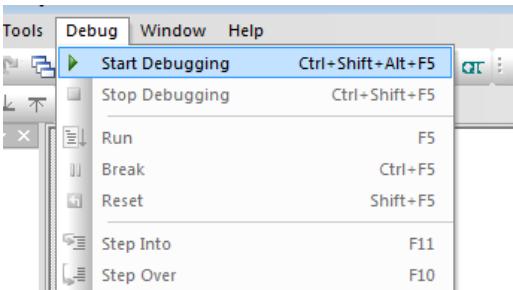


A fejlesztői környezet súgója a **Help** menü **Assembler Help** elmenüjéből érhető el. Itt adott utasításokra vagy utasítás csoportokra is rákereshetünk.

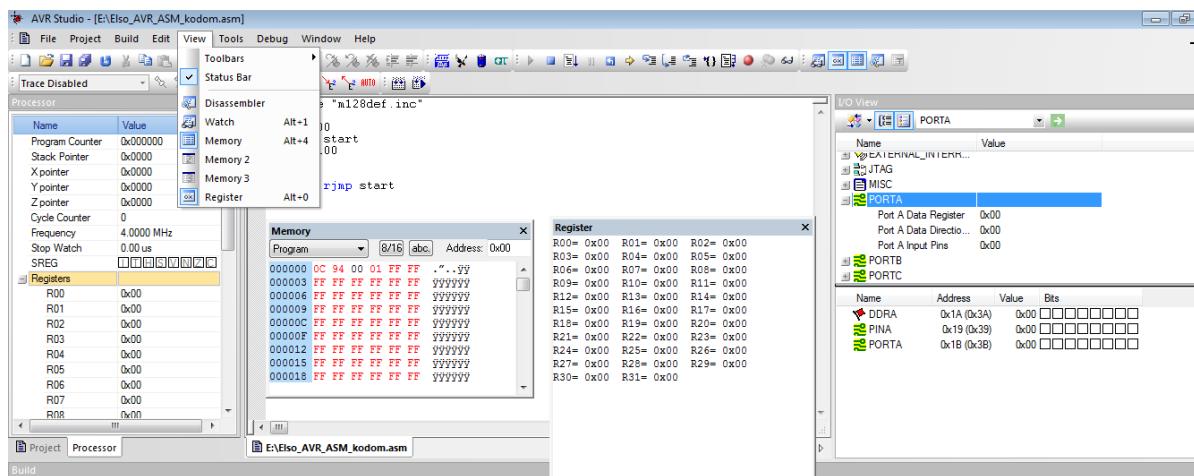
Ha egy kódban valamely utasításról szeretnénk információt, akkor az utasításra kattintva és az **F1** billentyű lenyomva megjelenik az adott utasítás súgója.

## 1.2.6 Debug

Debuggolás esetén a **Debug menu - Start Debugging** opciónál kell választani.



A **View** menüben kijelölhetők **Memory** ablakok és **Register** ablak, amelyeken az értékek változásai követhetők.



Alapbeállítások mellett jobb oldalon látható a **Processor** ablak, ahol a speciális regiszterek és az általános felhasználású regiszterek változásai követhetők. Bal oldalon található az **I/O View** ablak, ahol többek között a Portok változásai láthatók.

A kódban való léptetés az F10 és F11 billentyűkkel lehetséges.

### 1.3 Alap Assembly kód

```
;-----Üres Assembly kód-----
;Nem csinál semmit, de minden szintaktikai elemet tartalmaz
;és a későbbi programjaink alapját fogja képezni.
;

.INCLUDE "m128def.inc"    ; Include definitions Atmega128
;.INCLUDE "m64def.inc"    ; Include definitions Atmega64

.DSEG
;
.CSEG
.ORG 0x0
    rjmp start           ; jump to start

.ORG 0x100
;-----
; MACRO
;-----
;-----
; SUBRUTINE
;-----
;-----
; INTERRUPT
;-----
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
;-----
; PROGRAM
;-----
loop:
;-----
```

rjmp loop

### 1.4 Önálló feladatok

1. Hozzon létre egy projektet `elso_feladat` néven! Az alapkódot másolja be. A következő közös feladatok során ezt használja alapként!
2. Fordítsa le a létrehozott projektet, és töltse fel a T-Bird-re!

## 2 Alapvető utasítások megismerése, debugger

Utasításlista összefoglaló:

- Instruction Set Summary
  - o ATmega128 adatlap 365-367. oldal
  - o ATmega64 adatlap 395-397. oldal

### 2.1 Adatmozgató utasítások

DATA TRANSFER INSTRUCTIONS						
MOV	Rd, Rr	Move Between Registers	Rd $\leftarrow$ Rr	None	1	
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd $\leftarrow$ Rr+1:Rr	None	1	
LDI	Rd, K	Load Immediate	Rd $\leftarrow$ K	None	1	
LD	Rd, X	Load Indirect	Rd $\leftarrow$ (X)	None	2	
LD	Rd, X+	Load Indirect and Post-Inc.	Rd $\leftarrow$ (X), X $\leftarrow$ X + 1	None	2	
LD	Rd, - X	Load Indirect and Pre-Dec.	X $\leftarrow$ X - 1, Rd $\leftarrow$ (X)	None	2	
LD	Rd, Y	Load Indirect	Rd $\leftarrow$ (Y)	None	2	
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd $\leftarrow$ (Y), Y $\leftarrow$ Y + 1	None	2	
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y $\leftarrow$ Y - 1, Rd $\leftarrow$ (Y)	None	2	
LDD	Rd, Y+q	Load Indirect with Displacement	Rd $\leftarrow$ (Y + q)	None	2	
LD	Rd, Z	Load Indirect	Rd $\leftarrow$ (Z)	None	2	
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd $\leftarrow$ (Z), Z $\leftarrow$ Z+1	None	2	
LD	Rd, - Z	Load Indirect and Pre-Dec.	Z $\leftarrow$ Z - 1, Rd $\leftarrow$ (Z)	None	2	
LDD	Rd, Z+q	Load Indirect with Displacement	Rd $\leftarrow$ (Z + q)	None	2	
LDS	Rd, k	Load Direct from SRAM	Rd $\leftarrow$ (k)	None	2	
ST	X, Rr	Store Indirect	(X) $\leftarrow$ Rr	None	2	
ST	X+, Rr	Store Indirect and Post-Inc.	(X) $\leftarrow$ Rr, X $\leftarrow$ X + 1	None	2	
ST	- X, Rr	Store Indirect and Pre-Dec.	X $\leftarrow$ X - 1, (X) $\leftarrow$ Rr	None	2	
ST	Y, Rr	Store Indirect	(Y) $\leftarrow$ Rr	None	2	
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) $\leftarrow$ Rr, Y $\leftarrow$ Y + 1	None	2	
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y $\leftarrow$ Y - 1, (Y) $\leftarrow$ Rr	None	2	
STD	Y+q, Rr	Store Indirect with Displacement	(Y + q) $\leftarrow$ Rr	None	2	
ST	Z, Rr	Store Indirect	(Z) $\leftarrow$ Rr	None	2	
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) $\leftarrow$ Rr, Z $\leftarrow$ Z + 1	None	2	
ST	- Z, Rr	Store Indirect and Pre-Dec.	Z $\leftarrow$ Z - 1, (Z) $\leftarrow$ Rr	None	2	
STD	Z+q, Rr	Store Indirect with Displacement	(Z + q) $\leftarrow$ Rr	None	2	
STS	k, Rr	Store Direct to SRAM	(k) $\leftarrow$ Rr	None	2	
LPM		Load Program Memory	Rd $\leftarrow$ (Z)	None	3	
LPM	Rd, Z	Load Program Memory	Rd $\leftarrow$ (Z)	None	3	
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd $\leftarrow$ (Z), Z $\leftarrow$ Z+1	None	3	
ELPM		Extended Load Program Memory	Rd $\leftarrow$ (RAMPZ:Z)	None	3	
ELPM	Rd, Z	Extended Load Program Memory	Rd $\leftarrow$ (RAMPZ:Z)	None	3	
ELPM	Rd, Z+	Extended Load Program Memory and Post-Inc	Rd $\leftarrow$ (RAMPZ:Z), RAMPZ:Z $\leftarrow$ RAMPZ:Z+1	None	3	
SPM		Store Program Memory	(Z) $\leftarrow$ R1:R0	None	-	
IN	Rd, P	In Port	Rd $\leftarrow$ P	None	1	
OUT	P, Rr	Out Port	P $\leftarrow$ Rr	None	1	
PUSH	Rr	Push Register on Stack	STACK $\leftarrow$ Rr	None	2	
POP	Rd	Pop Register from Stack	Rd $\leftarrow$ STACK	None	2	

### 2.1.1 Közös feladat: Adatmozgató utasítások

```
;-----Adatmozgató utasítások-----
;Feladat:      LDI, MOV, MOVW utasítások megismerése és kipróbálása.
;              Használja a debuggert és
;              figyelje a regiszterek, SR, PC változását
;-----  
;LDI, MOV, MOVW  
  
.INCLUDE "m128def.inc"    ; Include definitions Atmega128
;.INCLUDE "m64def.inc"    ; Include definitions Atmega64  
  
.DSEG
;  
.CSEG
.ORG 0x0
    rjmp start           ; jump to start  
  
.ORG 0x100
;-----  
; MACRO
;-----  
;-----  
; SUBRUTINE
;-----  
;-----  
; INTERRUPT
;-----  
;-----  
; PROGRAM
;-----  
start:  
;-----  
  
ldi     r16, 13          ;r16 regiszterbe 13 decimális szám
töltése
ldi     r17, 0xF1          ;r17 regiszterbe 0xF1  hexadecimális
szám töltése
ldi     r18, 0b10001101    ;r18 regiszterbe 10001101 bináris
szám töltése  
  
mov     r19, r17          ;r17 regiszter másolása r19 regiszterbe
movw   r20:r21, r18:r19    ;r18-r19 regiszterpár másolása r20-r21 regiszterpárba  
  
loop:  
    rjmp   loop
```

## 2.2 Aritmetikai utasítások és logikai utasítások

Mnemonics	Operands	Description	Operation	Flags	#Clocks
<b>ARITHMETIC AND LOGIC INSTRUCTIONS</b>					
ADD	Rd, Rr	Add two Registers	$Rd \leftarrow Rd + Rr$	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	$Rd \leftarrow Rd + Rr + C$	Z,C,N,V,H	1
ADIW	Rd,K	Add Immediate to Word	$Rdh:Rdl \leftarrow Rdh:Rdl + K$	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	$Rd \leftarrow Rd - Rr$	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	$Rd \leftarrow Rd - K$	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	$Rd \leftarrow Rd - Rr - C$	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	$Rd \leftarrow Rd - K - C$	Z,C,N,V,H	1
SBIW	Rd,K	Subtract Immediate from Word	$Rdh:Rdl \leftarrow Rdh:Rdl - K$	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	$Rd \leftarrow Rd \bullet Rr$	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	$Rd \leftarrow Rd \bullet K$	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	$Rd \leftarrow Rd \vee Rr$	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	$Rd \leftarrow Rd \vee K$	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	$Rd \leftarrow Rd \oplus Rr$	Z,N,V	1
COM	Rd	One's Complement	$Rd \leftarrow \$FF - Rd$	Z,C,N,V	1
NEG	Rd	Two's Complement	$Rd \leftarrow \$00 - Rd$	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	$Rd \leftarrow Rd \vee K$	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	$Rd \leftarrow Rd \bullet (\$FF - K)$	Z,N,V	1
INC	Rd	Increment	$Rd \leftarrow Rd + 1$	Z,N,V	1
DEC	Rd	Decrement	$Rd \leftarrow Rd - 1$	Z,N,V	1
TST	Rd	Test for Zero or Minus	$Rd \leftarrow Rd \bullet Rd$	Z,N,V	1
CLR	Rd	Clear Register	$Rd \leftarrow Rd \oplus Rd$	Z,N,V	1
SER	Rd	Set Register	$Rd \leftarrow \$FF$	None	1
MUL	Rd, Rr	Multiply Unsigned	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
MULS	Rd, Rr	Multiply Signed	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	$R1:R0 \leftarrow Rd \times Rr$	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	$R1:R0 \leftarrow (Rd \times Rr) \lll 1$	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	$R1:R0 \leftarrow (Rd \times Rr) \lll 1$	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	$R1:R0 \leftarrow (Rd \times Rr) \lll 1$	Z,C	2

### 2.2.1 Közös feladat: Aritmetikai utasítások

```
;-----Aritmetikai utasítások-----
;Feladat:      ADD, ADC, ADIW, SUB, SBC, SBIW,
;              INC, DEC, MUL, SER, CLR
;              utasítások megismerése és kipróbálása.
;              Használja a debuggert és
;              figyelje a regiszterek, SR, PC változását
;-----  

;add, adc, adiw, sub, sbc, sbiw, inc, dec, mul, ser, clr
.INCLUDE "m128def.inc" ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    szam1 =      r16
.DEF    szam2 =      r17
.ORG 0x0
        rjmp start ; jump to start
.ORG 0x100
;-----  

; MACRO
;-----  

;-----  

; SUBROUTINE
;-----  

;-----  

; INTERRUPT
;-----
```

```

;-----
; PROGRAM
;-----
start:
;-----
;

ldi      szam1, 12          ;szam1-be 12 decimális szám töltése
ldi      szam2, 84          ;szam2-be 84 decimális szám töltése

;---Összeadás, kivonás

add      szam1, szam2      ;szam1 és szam2 összeadása, eredmény szam1-ben
sub      szam2, szam1      ;szam2 és szam1 kivonása, eredménye szam2-ben

;---Inkrementálás, dekrementálás

incszam1
dec      szam2            ;szam1 növelése eggyel
                          ;szam2 csökkentése eggyel

;---Szorzás

mul      szam1, szam2      ;szam1 és szam2 szorzása
                          ;eredmény r1(H) és r0(L) regiszterekben

;---művelet 16bites számokkal 300

ldi      r24, low(895)     ;895 alsó bájtjának töltése r24 regiszterbe
ldi      r25, high(895)    ;895 felső bájtjának töltése r25 regiszterbe

adiw    r25:r24, 12        ;r25-24 regiszterpárhoz 12 decimális szám hozzáadása

sbiw    r25:r24, 12        ;r25-24 regiszterpárból 24 decimális szám kivonása

ldi      r16, low(300)     ;300 alsó bájtjának töltése r16 regiszterbe
ldi      r17, high(300)    ;300 alsó bájtjának töltése r17 regiszterbe

add      r16, r24          ;16bites számok összeadása
adc      r17, r25          ;alsó bájtok összeadása
                          ;felső bájtok összeadása Carry-vel

sub      r24, r16          ;16bites számok kivonása
sbc      r25, r17          ;alsó bájtok kivonása
                          ;felső bájtok kivonása Carry-vel

;---Komplemens

com      szam2;           ;egyes komplemens
neg      szam1;           ;kettes komplemens

;---Clear, set

clr      szam2;           ;regiszter törlése
ser      szam1;           ;regiszter bitjeinek 1-be állítása

loop:    rjmp   loop

```

## 2.2.2 Közös feladat: Logikai utasítások

```

;-----Logikai utasítások-----
;Feladat:      AND, ANDI, OR, ORI, EOR
;              utasítások megismerése és kipróbálása.
;              Használja a debuggert és
;              figyelje a regiszterek, SR, PC változását
;-----
;and, or, eor, ori, andi
.INCLUDE "m128def.inc"                                ; Include definitions
Atmega128
.DSEG
;
.CSEG
.DEF    szam1 =     r16
.DEF    szam2 =     r17
.ORG 0x0
    rjmp start                                     ; jump to start

.ORG 0x100
;-----
; MACRO
;-----
;-----
; SUBROUTINE
;-----
;-----
; INTERRUPT
;-----
;-----
; PROGRAM
;-----
start:

ldi    szam1, 12          ;szam1-be 12 decimális szám töltése
ldi    szam2, 84          ;szam2-be 84 decimális szám töltése

;--ÉS
and   szam1, szam2        ;r16, r17 logikai és kapcsolata
andi  szam1, 0x01          ;r16 és 0x01 hexadecimális érték logikai és kapcsolata

;--VAGY
or    szam1, szam2        ;r16, r17 logikai vagy kapcsolata
ori   szam2, 0x11          ;r17 és 0x11 hexadecimális érték logikai vagy
kapcsolata

;--kizáró-vagy
eor   szam1, szam2        ;r16, r17 logikai kizáró-vagy kapcsolata

loop:
    rjmp loop

```

## 2.3 Bit és bitteszt utasítások

BIT AND BIT-TEST INSTRUCTIONS					
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)←Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)←Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1

### 2.3.1 Közös feladat: Bit utasítások

```

;-----Bit utasítások-----
;Feladat:      LSL, LSR, ROL, ROR, SWAP utasítások megismerése és kipróbálása.
;              Használja a debuggert és figyelje a regiszterek, SR, PC változását
;-----
;lsl, lsr, rol, ror, swap

.INCLUDE "m128def.inc"          ; Include definitions Atmega128
.DSEG
.CSEG
.DEF    szam1 =      r16
.DEF    szam2 =      r17
.ORG 0x0
    rjmp  start           ; jump to start
.ORG 0x100
;-----
; MACRO
;-----
;-----
; SUBROUTINE
;-----
;-----
; INTERRUPT
;-----
;-----
; PROGRAM
;-----
start:
;-----
ldi    szam1, 0b01001101      ;szam1-be 0b01101101 bináris érték töltése
ldi    szam2, 0b00100110      ;szam2-be 0b00100110 bináris érték töltése

lsl    szam1                   ;szam1 logikai shift balra 3x
lsl    szam1
lsl    szam1

ror   szam2                   ;szam2 rotálása jobbra 3x
ror   szam2
ror   szam2

swap  szam1                   ;szam1 alsó és felső 4 bitjének cseréje

```

```
loop:  
    rjmp  loop
```

## 2.4 Önálló feladatok

1. Töltsön az r22 regiszterbe 0xAB hexadecimális értéket. A CLI utasítás felhasználása nélkül törlje a regisztert! A feladat elvégzéséhez logikai utasítás használjon!
2. Töltsön az r16 regiszterbe 0b00100110 bináris értéket, majd shift-telje jobbra háromszor! Figyelje az r16 regiszter és a Status Register változását!
3. Töltsön az r16 regiszterbe 0b10010110 bináris értéket, majd rotálja balra háromszor! Figyelje az r16 regiszter és a Status Register változását!

### 3 Portkezelés alapjai: LED-ek

#### 3.1 LED-ek

##### Register Description for I/O Ports

- ATmega128 adatlap 86-88. oldal, ATmega64 adatlap 87-89. oldal
  - DDRx
  - PORTx
  - PINx

##### T-Bird portkiosztás

bit	7	6	5	4	3	2	1	0	
PORT									
A	ENABLE	SEL2	SEL1	SEL0	DATA3	DATA2	DATA1	DATA0	7 SEGMENT DISPLAY
I/O	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	
B	Led3	Led2	Led1	Led0					LED/lo 4bit
I/O	OUT	OUT	OUT	OUT					
C	RED	KBD4row	KBD3row	KBD2row	KBD1row	KBD_right	KBD_centr	KBD_left	Keyboard
I/O	OUT	OUT	OUT	OUT	OUT	IN	IN	IN	
D	Led7	Led6	Led5	Led4					LED/hi 4bit
I/O	OUT	OUT	OUT	OUT					
E	LCD_DATA7	LCD_DATA6	LCD_DATA5	LCD_DATA4	GREEN	BLUE			LCD data
I/O	OUT	OUT	OUT	OUT	OUT	OUT			
F					LCD_E	LCD_R/W	LCD_RS	LM35	LCD Control
I/O					OUT	OUT	OUT	IN	Analog
G	NC	NC	NC	K4	K3	K2	K1	K0	Pushbutton
I/O	X	X	X	IN	IN	IN	IN	IN	
bit	7	6	5	4	3	2	1	0	

PD7	PD6	PD5	PD4					
LED7	LED6	LED5	LED4	LED3	LED2	LED1	LEDO	
				PB7	PB6	PB5	PB4	

### 3.2 LED(ek) bekapcsolása

### 3.2.1 Közös feladat: LED bekapcsolása

```

;-----LED bekapcsolása-----
;Feladat:          Kapcsolja be a LED0-t.
;-----;
;DDRn, OUT, PORTn

.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.ORG 0x0
    rjmp start                ; jump to start

.ORG 0x100
;-----
; MACRO
;-----
; SUBROUTINE
;-----
; INTERRUPT
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----


ldi      r16, 0xF0             ;8 LED-hez tartozó PIN kimenetre állítása
out     DDRB, r16              ;r16 regiszterbe 0F hexadecimális érték töltése
out     DDRD, r16              ;Data Direction Register B port– DDRB regiszterbe r16
;Data Direction Register D port– DDRD regiszterbe r16
;-----


; PROGRAM
;-----


ldi      r16, 0b_0000_0001   ;r16 regiszterbe 00000001 bináris érték töltése
swap   r16                    ;r16 regiszter alsó és felső 4 bitjének cseréje
out    PORTB, r16              ;Data Register B port– PORTB regiszterbe r16
;-----


loop:
        rjmp loop

```

### 3.2.2 Közös feladat: Műveletvégrés eredményének megjelenítse LED-eken

```
;-----Műveletvégzés eredménye LED-ekre-----
;Feladat:      Adja össze a következő számokat: 45, 8.
;              A műveletvégzés eredményét jelenítse meg a LED-eken.
;-----
```

```
.INCLUDE "m128def.inc"          ; Include definitions Atmega128
.DSEG
;
.CSEG
.ORG 0x0
    rjmp start                ; jump to start

.ORG 0x100
;-----
```

```
; MACRO
;-----
```

```
;-----
```

```
; SUBRUTINE
;-----
```

```
;-----
```

```
; INTERRUPT
;-----
```

```
;-----
```

```
; PROGRAM
;-----
```

```
start:
;-----
```

```
; INIT
;-----
```

```
ldi    r16, 0xF0           ;r16 regiszterbe 0F hexadecimális érték töltése
out   DDRB, r16            ;Data Direction Register B port– DDRB regiszterbe r16
out   DDRD, r16            ;Data Direction Register D port– DDRD regiszterbe r16
;-----
```

```
; PROGRAM
;-----
```

```
ldi    r17, 8               ;r17 regiszterbe 8 decimális érték töltése
ldi    r18, 45              ;r18 regiszterbe 45 decimális érték töltése
add   r17, r18              ;r17 és r18 összeadása, eredmény r17 regiszterbe
mov   r16, r17              ;r17 regiszter másolása r16 regiszterbe
```

```
out   PORTD, r16            ;Data Register D port– PORTD regiszterbe r16
swap  r16                  ;r16 regiszter alsó és felső 4 bitjének cseréje
out   PORTB, r16            ;Data Register B port– PORTB regiszterbe r16
```

```
loop:
    rjmp loop
```

### 3.3 Önálló feladatok

1. A közösen megoldott feladatok mintájára és a T-Bird portkiosztását felhasználva kapcsolja be a LED6 és a LED2 LED-eket a boardon!
2. Adja össze a következő 8 bites hexadecimális számokat: 12, A1, 28. A kapott eredménye rotálja kettőt jobbra, majd tegye ki a LED-ekre a kapott értéket!

## 4 Makrók

### 4.1 LED init makró

#### 4.1.1 Közös feladat

```
;-----LED_init macro-----
;Feladat:      Írja meg a LED_init makrót:
;                  a LED-eket tartalmazó port irány regiszterének beállítását.
;-----
;.macro          .endmacro
.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    tmp    =      r16
.ORG 0x0
rjmp start                     ; jump to start
.ORG 0x100
;-----
; MACRO
;-----
.macro LED_init                 ;LED_init macro
    ldi    tmp, 0xF0           ;r16 (tmp) regiszterbe 0F hexadecimális érték töltése
    out   DDRB, tmp           ;Data Direction Register B port
    out   DDRD, tmp           ;DDRB regiszterbe r16 (tmp)
    out   DDRD, tmp           ;Data Direction Register D port
    out   DDRD, tmp           ;DDRD regiszterbe r16 (tmp)
.endmacro                      ;macro vége
;-----
;SUBRUTINE
;-----
;-----
;INTERRUPT
;-----
;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
LED_init
;-----
;PROGRAM
;-----
ldi    r16, 0b_0000_0010     ;r16 regiszterbe 00000001 bináris érték töltése
swap  r16                      ;r16 regiszter alsó és felső 4 bitjének cseréje
out   PORTB, r16               ;Data Register B port – PORTB regiszterbe r16
loop:
rjmp  loop
```

## 4.2 Önálló feladatok

1. Írjon egy makrót LED\_out néven. A makró tegyen ki egy 8 bites számot a 8db LED-re!
2. Végezzen a következő 16 bites számokkal összeadást: 895, 300. Az eredmény felső 8 bitjét jelenítse meg a LED-eken!
3. Végezzen a következő 16 bites számokkal kivonást: 1398, 294. Az eredmény alsó 8 bitjét jelenítse meg a LED-eken!
4. Végezzen szorzást a következő számokkal: 45, 100. Az eredmény felső 8 bitjét jelenítse meg a LED-eken!

## 5 Gyakorló feladatok

1. Vegye a következő értékek logikai VAGY kapcsolatát: 0xF1, 0x14, majd az eredményt tegye ki a LED-ekre!
2. Adjon hozzá a 876 decimális értékhez 43-at, az eredmény alsó bájtját tegye ki a LED-ekre!
3. Vonjon ki a 1234 decimális értékből 52-t, az eredmény felső bájtját tegye ki a LED-ekre!
4. Szorozza össze a következő számokat: 12, 100, majd az eredmény alsó bájtját tegye ki a LED-ekre!
5. A 12 decimális értéket szorozza meg 4-gyel. A művelet elvégzéséhez ne használja a MUL utasítást, a feladatot bit utasítással oldja meg!

## 2. labor

### Labor tematikája:

1. „Ciklusok” (26-39. oldal)
2. Futófény variációk (91. oldal, 92-100. oldal)
3. Osztás algoritmus
4. Portkezelés alapjai: Nyomógombok (91. oldal, 21-26. oldal)

## 1 „Ciklusok”

### 1.1 Feltételes ugró utasítások

BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC $\leftarrow$ PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC $\leftarrow$ Z	None	2
JMP	k	Direct Jump	PC $\leftarrow$ k	None	3
RCALL	k	Relative Subroutine Call	PC $\leftarrow$ PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC $\leftarrow$ Z	None	3
CALL	k	Direct Subroutine Call	PC $\leftarrow$ k	None	4
RET		Subroutine Return	PC $\leftarrow$ STACK	None	4
RETI		Interrupt Return	PC $\leftarrow$ STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC $\leftarrow$ PC + 2 or 3	None	1 / 2 / 3
CP	Rd,Rr	Compare	Rd $\leftarrow$ Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd $\leftarrow$ Rr - C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd $\leftarrow$ K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC $\leftarrow$ PC + 2 or 3	None	1 / 2 / 3
SBRSC	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC $\leftarrow$ PC + 2 or 3	None	1 / 2 / 3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC $\leftarrow$ PC + 2 or 3	None	1 / 2 / 3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC $\leftarrow$ PC + 2 or 3	None	1 / 2 / 3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC $\leftarrow$ PC+k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC $\leftarrow$ PC+k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N $\oplus$ V= 0) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N $\oplus$ V= 1) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC $\leftarrow$ PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC $\leftarrow$ PC + k + 1	None	1 / 2

### 1.1.1 Közös feladat: Összegzés, BREQ, BRNE, CPI

```
;-----Feltételes ugró utasítások-----
;Feladat:      Összegezzen 10db növekvő számot 0-tól kezdődően.
;              Összegezzen 10db csökkenő számot 10-től kezdődően,
;              majd az eredményt jelenítse meg LED-eken.
;-----;
;BREQ, BRNE, CPI

.INCLUDE "m128def.inc"                                ; Include definitions
Atmega128
.DSEG
;
.CSEG
.DEF    szam    =      r16
.DEF    osszeg  =      r17
.DEF    LED     =      r18
.ORG 0x0
    rjmp  start                         ; jump to start
.ORG 0x100
;-----;
; MACRO
;-----;
.macro LED_init
    ldi   r16, 0xF0
    out  DDRB, r16
    out  DDRD, r16
.endmacro

.macro LED_out
    out  PORTD, LED
    swap LED
    out  PORTB, LED
    swap LED
.endmacro
;-----;
;SUBRUTINE
;-----;
;INTERRUPT
;-----;
;PROGRAM
;-----;
start:
;-----;
;INIT
;-----;
LED_init
;-----;
;PROGRAM
```

```

;-----
ldi      szam, 0          ;szam1-be 0 decimális érték töltése
ldi      osszeg, 0        ;szam2-be 0 decimális érték töltése

;--10 darab növekvő szám összeadása 0-tól kezdődően
osszegez:
add    osszeg, szam       ;osszeg és szam összeadása
inc    szam               ;szam inkrementálása
cpi    szam, 10          ;szam összehasonlítása 10 decimális értékkel
brne   osszegez          ;BRNE: Branch if Not Equal, ugrás
                           ;ha nem egyenlőek, az osszegez címkére

;--10 darab csökkenő szám összeadása 10-től kezdődően
eor    osszeg, osszeg     ;osszeg nullázása

osszegez2:
add    osszeg, szam       ;osszegez2 címke
dec    szam               ;osszeg és szam összeadása
cpi    szam, 0            ;szam dekrementálása
breq   osszegez2_vege    ;szam és 0 decimális érték összehasonlítása
                           ;BREQ: Branch if Equal, ugrás
                           ;ha egyenlőek, az osszegez2_vege címkére
rjmp   osszegez2          ;relatív jump az osszegez2 címkére
osszegez2_vege:
                           ;osszegez2_vege címke

;--Összeg kitétele LED-ekre
mov    LED, osszeg        ;osszeg másolása LED-be
LED_out
                           ;LED_out makró

loop:
rjmp   loop

```

## 1.2 Késleltetés, LED villogtatás

### 1.2.1 Közös feladat: LED0 villogtatása

```
;-----LED villogtatása-----
;Feladat:      Villogtassa a LED0 LED-et ~1s-os időközönként.
;                  A villogtatáshoz írjon delay makró.
;-----;
;BRNE, „delay”, LED, EOR

.INCLUDE "m128def.inc" ; Include definitions
Atmega128
.DSEG
;
.CSEG
.DEF     LED      =      r18
.ORG 0x0
    rjmp start           ; jump to start

.ORG 0x100
;-----;
; MACRO
;-----;
.macro LED_init
    ldi     r16, 0xF0
    out    DDRB, r16
    out    DDRD, r16
.endmacro

.macro LED_out
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
.endmacro

.macro delay_1s
    ldi     r26, 0xFF
    ldi     r25, 0xFF
    ldi     r24, 0x5F
    delay_loop:
        dec    R26          ; ha 0 → Z flag=1
        brne delay_loop    ; Z flag-ét figyeli, ha Z flag=0 → ugrik
        dec    R25
        brne delay_loop;
        dec    R24
        brne delay_loop
.endmacro
;-----;
;SUBRUTINE
;-----;
;
```

```

; INTERRUPT
;-----

;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
    LED_init
;-----
; PROGRAM
;-----
ldi r20, 1          ; xor művelethez segédérték
mov     LED, r20

loop:
    LED_out
    eor     LED, r20      ; kizáró vagy - villogtatás
    delay_1s

rjmp   loop

```

### 1.3 Önálló feladat

1. 310-től kezdődően összegezzen 20db növekvő páros számot. Az összeg alsó bájtját tegye ki a LED-ekre.

## 2 Futófény

### 2.1.1 Közös feladat: Egyszerű (körbe) futófény

```
;-----Egyszerű (körbe)futófény-----
;Feladat:          Rotálással készítsen egyszerű körbefutófénnyt.
;-----
;BRNE, ROL

.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF      LED      =      r18
.ORG 0x0
        rjmp start           ; jump to start

.ORG 0x100
;-----
; MACRO
;-----
.macro LED_init
    ldi      r16, 0xF0
    out     DDRB, r16
    out     DDRD, r16
.endmacro

.macro LED_out
    out     PORTD, LED
    swap   LED
    out     PORTB, LED
    swap   LED
.endmacro

.macro delay_1s
    ldi      r26, 0xFF
    ldi      r25, 0xFF
    ldi      r24, 0x5F
    delay_loop:
        dec     R26           ; lefele számol, ha 0 lesz, akkor Z flag=1
        brne  delay_loop      ; Z flag-et nézi, ha Z flag=0, akkor ugrik,
        dec     R25
        brne  delay_loop;
        dec     R24
        brne  delay_loop
.endmacro
;-----
;SUBRUTINE
;-----
```

```
;-----  
; INTERRUPT  
;-----  
;  
;-----  
; PROGRAM  
;-----  
start:  
;-----  
; INIT  
;-----  
    LED_init  
;-----  
; PROGRAM  
;-----  
ldi      LED, 0x01  
  
loop:  
    LED_out  
    rol      LED          ; balra rotálás  
    delay_1s  
  
rjmp    loop
```

### 2.1.2 Közös feladat: Körbefutófény

```
;-----Körbefutófény-----
;Feladat:      SHIT-eléssel készítsen folyamatos körbefutófényt.
;                  Figyelje az átvitelt jelző bit értékét.
;-----
;BRCS, BRCC, CPI

.INCLUDE "m128def.inc" ; Include definitions
Atmega128
.DSEG
;
.CSEG
.DEF      LED      =      r18
.ORG 0x0
        rjmp start ; jump to start
.ORG 0x100
;-----
; MACRO
;-----
.macro LED_init
    ldi    r16, 0xF0
    out   DDRB, r16
    out   DDRD, r16
.endmacro

.macro LED_out
    out   PORTD, LED
    swap  LED
    out   PORTB, LED
    swap  LED
.endmacro

.macro delay_1s
    ldi    r26, 0xFF
    ldi    r25, 0xFF
    ldi    r24, 0x5F
    delay_loop:
        dec   R26          ; lefele számol, ha 0 lesz, akkor Z flag=1
        brne delay_loop    ; Z flag-et nézi, ha Z flag=0, akkor ugrik,
        dec   R25
        brne delay_loop;
        dec   R24
        brne delay_loop

.endmacro
;-----
;SUBROUTINE
;-----
;-----
;INTERRUPT
;-----
```

```

;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
    LED_init
;-----
; PROGRAM
;-----
ldi LED, 0x01

loop:
    LED_out
    delay_1s
    lsl     LED          ; balra shift
    brcs   c_s           ; BRCS: Branch if Carry Set, ugrás
                           ; ha a C flag értéke 1 a c_s címkére
                           ; BRCC: Branch if Carry Cleared, ugrás
                           ; ha a C flag értéke 0 a loop címkére
    brcc   loop
c_s:
    ldi     LED, 0x01

rjmp   loop

```

## 2.2 Önálló feladat

1. Készítsen ide-oda futófényt a következő állapotok szerint:

7	6	5	4	3	2	1	0
x	x	x	x	x	x	0	0
x	x	x	x	x	0	0	x
x	x	x	x	0	0	x	x
x	x	x	0	0	x	x	x
x	x	0	0	x	x	x	x
x	0	0	x	x	x	x	x
0	0	x	x	x	x	x	x
x	0	0	x	x	x	x	x
x	x	0	0	x	x	x	x
...							
x	x	x	x	x	x	0	0
x	x	x	x	x	0	0	x
...							

A feladatot feltételes ugró utasítások használatával valósítsa meg. Az 1. sor a LED-ek számát jelzi, a 0 jelölés a világító LED-ekre vonatkozik, az x pedig a nem világítókra.

### 3 Osztás

#### 3.1 Közös feladat: Osztás művelet visszavezetése kivonásra

```
;-----Osztás-----
;Feladat:      Valósítson meg osztást, kivonásra visszavezetve.
;              A feladatot úgy valósítsa meg, hogy a maradék értékét is tárolja.
;              Teszteléshez használja a következőket: 20/4, 20/6.
;-----;
;BREQ, BRMI

.INCLUDE "m128def.inc"                                ; Include definitions
Atmega128
.DSEG
;
.CSEG
.DEF    szam        =      r16
.DEF    eredmény   =      r17
.DEF    oszto       =      r18
.DEF    maradek     =      r19
.DEF    LED         =      r20
.ORG 0x0
    rjmp start                                     ; jump to start

.ORG 0x100
;-----;
; MACRO
;-----;
.macro LED_init
    ldi    r16, 0xF0
    out   DDRB, r16
    out   DDRD, r16
.endmacro

.macro LED_out
    out   PORTD, LED
    swap  LED
    out   PORTB, LED
    swap  LED
.endmacro

;-----;
; SUBROUTINE
;-----;
;-----;
; INTERRUPT
;-----;
;-----;
; PROGRAM
;-----;
.start:
```

```

;-----
; INIT
;-----
    LED_init
;-----
; PROGRAM
;-----
ldi    szam, 20
ldi    oszto, 4
ldi    eredmeny, 0

osztas:
    inc   eredmeny           ;eredmény növelése

    sub   szam, oszto        ;szam-ból oszto kivonása
    breq  maradek_nelkuli_osztas ;Z falg figyelése
    brmi  maradekos_osztas   ;BRMI: Branch if Minus

    rjmp  osztas

maradek_nelkuli_osztas:      ;maradék nélküli volt az osztás
    mov   maradek, szam
    rjmp  loop

maradekos_osztas:            ;ha nem maradék nélküli az osztás
    dec   eredmeny          ;eggyel többször végeztünk kivonást
    add   szam, oszto        ;vissza kell állítani a maradékot

;-----Eredmény/maradék kitétele LED-re
;-----Önálló feladat megírni

loop:
    rjmp  loop

```

## 3.2 Önálló feladat

1. A közösen elvégzett osztási feladat eredményét tegye ki a LED-ekre.
2. A 134/12 osztás maradékát tegye ki a LED-ekre.

## 4 Portkezelés alapjai: Nyomógombok

bit	7	6	5	4	3	2	1	0	
PORT									
A	ENABLE	SEL2	SEL1	SEL0	DATA3	DATA2	DATA1	DATA0	7 SEGMENT DISPLAY
I/O	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	
B	Led3	Led2	Led1	Led0					LED/lo 4bit
I/O	OUT	OUT	OUT	OUT					
C	RED	KBD4row	KBD3row	KBD2row	KBD1row	KBD_right	KBD_centr	KBD_left	Keyboard
I/O	OUT	OUT	OUT	OUT	OUT	IN	IN	IN	
D	Led7	Led6	Led5	Led4					LED/hi 4bit
I/O	OUT	OUT	OUT	OUT					
E	LCD_DATA7	LCD_DATA6	LCD_DATA5	LCD_DATA4	GREEN	BLUE			LCD data
I/O	OUT	OUT	OUT	OUT	OUT	OUT			
F					LCD_E	LCD_R/W	LCD_RS	LM35	LCD Control
I/O					OUT	OUT	OUT	IN	Analog
G	NC	NC	NC	K4	K3	K2	K1	K0	Pushbutton
I/O	X	X	X	IN	IN	IN	IN	IN	
bit	7	6	5	4	3	2	1	0	

### 4.1 Gombkezelés

#### 4.1.1 Közös feladat: Lenyomott gomb számának megfelelő LED világítása

```
;-----GOMB olvasása-----
;Feladat: Olvassa be a nyomógombok értékét.
; A lenyomott gomb számának megfelelő LED világítson.
;-----
```

```
;PINn
```

```
.INCLUDE "m128def.inc" ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF btn = r16
.DEF LED = r17
.ORG 0x0
    rjmp start ; jump to start
.ORG 0x100
;-----
; MACRO
;-----
.macro init
    ldi r16, 0xF0
    out DDRB, r16
    out DDRD, r16
    ldi r16, 0x00
    sts DDRG, r16 ; gombok-hoz tartozó PIN-ek bemenetre
```

```

állítása ;(PG4-PG0)
.endmacro

.macro LED_out
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
.endmacro

;-----
;----- ;SUBRUTINE
;----- ;INTERRUPT
;----- ;PROGRAM
;----- start:
;----- ;INIT
;----- init
;----- ;PROGRAM
;----- ;----- ;loop:
    lds    btn, PING           ;G port olvasása btn-be
    andi   btn, 0b00011111     ;alsó 5 bit maszkolása (G4-G0)
    mov    LED, btn            ;értékek kitétele LED-ekre
    LED_out

    rjmp   loop

```

#### 4.1.2 Közös feladat: GOMB menü készítése

```
;-----GOMB menu-----
;Feladat:          Készítsen GOMB menüt az első 3 gombra.
;                  G0 lenyomásakor LED7 világít
;                  G1 lenyomásakor LED6 világít
;                  G2 lenyomásakor LED5 világít
;-----;
;PINn, SBRC

.INCLUDE "m128def.inc"                                ; Include definitions
Atmega128
.DSEG
;
.CSEG
.DEF      btn     =      r16
.DEF      LED     =      r17
.ORG 0x0
    rjmp start                         ; jump to start

.ORG 0x100
;-----;
; MACRO
;-----;
.macro init
    ldi    r16, 0xF0
    out   DDRB, r16
    out   DDRD, r16
    ldi    r16, 0x00
    sts   DDRG, r16
.endmacro

.macro LED_out
    out   PORTD, LED
    swap  LED
    out   PORTB, LED
    swap  LED
.endmacro
;-----;
; SUBRUTINE
;-----;
;-----;
; INTERRUPT
;-----;
;-----;
; PROGRAM
;-----;
.start:
;-----;
; INIT
;-----;
init
```

```

;-----
; PROGRAM
;-----
loop:
    lds    btn, PING           ;G port olvasása btn-be
    sbrc   btn, 0              ;SBRC: Skip if Bit in Register Cleared
    rjmp   menu_1
    sbrc   btn, 1              ;ha a btn 1. bitje 0, akkor átugorja a következő
utasítást
    rjmp   menu_2
    sbrc   btn, 2              ;ha a btn 2. bitje 0, akkor átugorja a következő
utasítást
    rjmp   menu_3
    rjmp   loop

menu_1:                      ;menü1 funkció
    ldi    LED, 0x80
    LED_out
    rjmp   loop

menu_2:                      ;menü2 funkció
    ldi    LED, 0x40
    LED_out
    rjmp   loop
menu_3:                      ;menü3 funkció
    ldi    LED, 0x20
    LED_out

rjmp   loop

```

## 4.2 Önálló feladat

1. Valósítsa meg a következő funkciókat a 3-as és a 4-es gombra. A 4-es gomb lenyomására adjon össze 2 számot (12, 43) és jelenítse meg a kapott értéket a LED-eken. A 3-as gomb lenyomására szorozzon össze két számot (12, 5) és az eredményt tegye ki a LED-ekre.

## 5 Gyakorló feladatok

- Készítse el a következő futófényt:

7	6	5	4	3	2	1	0
x	x	x	0	0	x	x	x
x	x	0	x	x	0	x	x
x	0	x	x	x	x	0	x
0	x	x	x	x	x	x	0
x	0	x	x	x	x	0	x
x	x	0	x	x	0	x	x
x	x	x	0	0	x	x	x
x	x	0	x	x	0	x	x

...

- Készítse el a következő futófényt:

7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	0
x	x	x	x	x	x	0	0
x	x	x	x	x	0	0	0
x	x	x	x	0	0	0	0
x	x	x	0	0	0	0	x
x	x	0	0	0	0	x	x
x	0	0	0	0	x	x	x
0	0	0	0	x	x	x	x
x	0	0	0	0	x	x	x

...

x	x	x	x	0	0	0	0
x	x	x	x	x	0	0	0
x	x	x	x	x	x	0	0
x	x	x	x	x	x	x	0
x	x	x	x	x	x	0	0

...

- Készítsen felfele számlálót a LED-ekre, amely 0-123-ig számol.

- Töltsé fel az r16 regisztert 43-mal, az r17 regisztert 3-mal. Ha r16 maradék nélkül osztható r17-tel, akkor világítson az összes LED, különben a maradék értékét jelenítse meg.

### 3. labor

#### Labor tematikája:

1. Szubrutinok (18, 36. oldal)
2. Adatmemória írása (19. oldal)
3. Adatmemória olvasása
4. Memória másolása

BRANCH INSTRUCTIONS					
CALL	k	Direct Subroutine Call	PC $\leftarrow$ k	None	4
RET		Subroutine Return	PC $\leftarrow$ STACK	None	4

DATA TRANSFER INSTRUCTIONS					
LD	Rd, X	Load Indirect	Rd $\leftarrow$ (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd $\leftarrow$ (X), X $\leftarrow$ X + 1	None	2
LD	Rd, -X	Load Indirect and Pre-Dec.	X $\leftarrow$ X - 1, Rd $\leftarrow$ (X)	None	2
LD	Rd, Y	Load Indirect	Rd $\leftarrow$ (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd $\leftarrow$ (Y), Y $\leftarrow$ Y + 1	None	2
LD	Rd, -Y	Load Indirect and Pre-Dec.	Y $\leftarrow$ Y - 1, Rd $\leftarrow$ (Y)	None	2
LDD	Rd, Y+q	Load Indirect with Displacement	Rd $\leftarrow$ (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd $\leftarrow$ (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd $\leftarrow$ (Z), Z $\leftarrow$ Z + 1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z $\leftarrow$ Z - 1, Rd $\leftarrow$ (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd $\leftarrow$ (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd $\leftarrow$ (k)	None	2
ST	X, Rr	Store Indirect	(X) $\leftarrow$ Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) $\leftarrow$ Rr, X $\leftarrow$ X + 1	None	2
ST	-X, Rr	Store Indirect and Pre-Dec.	X $\leftarrow$ X - 1, (X) $\leftarrow$ Rr	None	2
ST	Y, Rr	Store Indirect	(Y) $\leftarrow$ Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) $\leftarrow$ Rr, Y $\leftarrow$ Y + 1	None	2
ST	-Y, Rr	Store Indirect and Pre-Dec.	Y $\leftarrow$ Y - 1, (Y) $\leftarrow$ Rr	None	2
STD	Y+q, Rr	Store Indirect with Displacement	(Y + q) $\leftarrow$ Rr	None	2
ST	Z, Rr	Store Indirect	(Z) $\leftarrow$ Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) $\leftarrow$ Rr, Z $\leftarrow$ Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z $\leftarrow$ Z - 1, (Z) $\leftarrow$ Rr	None	2
STD	Z+q, Rr	Store Indirect with Displacement	(Z + q) $\leftarrow$ Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) $\leftarrow$ Rr	None	2
LPM		Load Program Memory	R0 $\leftarrow$ (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd $\leftarrow$ (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd $\leftarrow$ (Z), Z $\leftarrow$ Z + 1	None	3

# 1 Szubrutinok

## 1.1 Közös feladat: STACK\_init makró, PUSH, POP

```

-----SATCK Pointer-----
;Feladat:           Írja meg a STACK_init makrót.
;                   A Stack Pointert állítsa a RAM terület végére.
;                   Tárolja a 43, 55 decimális értékeket a veremben.
;                   Tölts vissza 2 értéket a veremből úgy,
;                   hogy a r17-be r18 régi értéke, r18-ba r17 régi
;                   értéke fog betöltődni
;-----;
;PUSH, POP

.INCLUDE "m128def.inc" ; Include definitions Atmega128
.DSEG
;
.CSEG
.ORG 0x0
    rjmp start ; jump to start
.ORG 0x100
;-----;
; MACRO
;-----;
.macro STACK_init ;veremmutató (SP)
;a RAM terület végére fog mutatni
;felső bájt
    ldi      r16, high(RAMEND)
    out     SPH, r16
    ldi      r16, low(RAMEND) ;alsó bájt
    out     SPL, r16
.endmacro
;-----;
; SUBRUTINE
;-----;
; INTERRUPT
;-----;
; PROGRAM
;-----;
start:
;-----;
; INIT
;-----;
STACK_init
;-----;
; PROGRAM
;-----;
ldi      r17, 43
ldi      r18, 55

push    r17 ;r17 értékének tárolása a verembe
push    r18 ;r18 értékének tárolása a verembe

```

```
eor      r17, r17          ;regiszterek törlése
eor      r18, r18
; a következő két utasítás eredményeképpen a r17-be r18 régi értéke
;r18-ba r17 régi értéke fog betöltődni
pop     r17                ;r17-be érték töltése a veremből
pop     r18                ;r18-be érték töltése a veremből
```

loop:

```
rjmp   loop
```

## 1.2 Közös feladat: LED\_out szubrutin

```
;-----LED_out szubrutin-----
;Feladat:      Írja meg a LED_out szubrutint.
;              A szubrutin tegyen ki egy 8bites értéket a 8db LED-re.
;              Adja össze a következő értékeket: 43, 55.
;              Az eredményt jelenítse meg a LED-eken.
;-----;
;RET, CALL

.INCLUDE "m128def.inc"                                ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF     LED      =      r20

.ORG 0x0
    rjmp start                         ; jump to start

.ORG 0x100
;-----
; MACRO
;-----;KÖTELEZŐ!!!
.macro STACK_init          ;veremmutató (SP) a RAM terület végére fog mutatni
    ldi     r16, high(RAMEND)           ;felső bájt
    out    SPH, r16
    ldi     r16, low(RAMEND)            ;alsó bájt
    out    SPL, r16
.endmacro

.macro PORT_init
    ldi     r16, 0xF0
    out    DDRB, r16
    out    DDRD, r16
    eor    r16, r16
    sts    DDRG, r16
.endmacro

;-----
; SUBRUTINE
;-----LED_out:                           ;LED_out szubrutin
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
ret                             ;return szubrutin

;-----
; INTERRUPT
;-----;
```

```

; PROGRAM
;-----
start:

;-----
; INIT
;-----
STACK_init
PORT_init
;-----
; PROGRAM
;-----

ldi r17, 43
ldi r18, 55

add    r17, r18
mov    LED, r17

call   LED_out           ;LED_out szubrutin hívása

loop:
rjmp  loop

```

### 1.3 Önálló feladat

1. Készítsen Menüt, ahol a nyomógombok lenyomásával választhat a menüpontok közül. Az egyes menüfunkciókat szubrutinban valósítsa meg.

GOMB\_0: Két szám összeadása, az összeg megjelenítése a LED-eken.  
 GOMB\_1: Két szám kivonása, az különbség megjelenítése a LED-eken.  
 GOMB\_2: Két szám szorzása, a szorzat alsó bájtjának megjelenítése LED-eken.  
 GOMB\_3: Két szám ÉS kapcsolata.  
 GOMB\_4: Két szám VAGY kapcsolata.

## 2 Adatmemória írása

### 2.1 Közös feladat: Memória írása növekvő címekre

```
;-----Adatmemória írása-----
;Feladat: Írjon 10db növekvő 8 bites számot a memóriába
;           12 kezdőértéktől a 0x123-as memóriacímtől kezdődően
;           növekvő címekre.
;-----;
; X index regiszter, adatterület címzésére használatos indexregiszter.
; Y index regiszter, adatterület címzésére használatos indexregiszter
; Z index regiszter, kódterület címzésére használatos indexregiszter
;           (Flash program memória címzés).

; ST X, Rr - Store Indirect (X) <- Rr
; ST X+, Rr - Store Indirect and Post-Increment (X) <- Rr, X <- X + 1
; ST -X, Rr - Store Indirect and Pre-Decrement X <- X - 1, (X) <- Rr

; LD Rd, X - Load Indirect Rd <- (X)
; LD Rd, X+ - Load Indirect and Post-Increment Rd <- (X), X <- X + 1
; LD Rd, -X - Load Indirect and Pre-Decrement X <- X - 1, Rd <- (X)

; X, Y, Z indexregiszterekkel segítségével írhatjuk és olvashatjuk az adatmemóriát

.INCLUDE "m128def.inc" ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF     szam    =      r20
.DEF     db      =      r21;

.ORG 0x0
        rjmp start ; jump to start

.ORG 0x100
;-----;
; MACRO
;-----;KÖTELEZŐ!!!
.macro STACK_init ;veremmutató (SP) a RAM terület végére fog mutatni
    ldi    r16, high(RAMEND) ;felső bájt
    out   SPH, r16
    ldi    r16, low(RAMEND) ;alsó bájt
    out   SPL, r16
.endmacro
;-----;
; SUBRUTINE
;-----;
;-----;
; INTERRUPT
;-----;
;
```

```
; PROGRAM
;-----
start:
;-----
; INIT
;-----
STACK_init
;-----
; PROGRAM
;-----
```

ldi ZL, low (0x123) ;0x123 alsó bájtja a Z indexregiszter alsó bájtjába  
ldi ZH, high (0x123) ;0x123 felső bájtja a Z indexregiszter felső bájtjába

ldi szam, 12 ;kezdőérték  
ldi db, 10 ;darabszám

tolt:

st	Z+, szam	;szam memóriába töltése, majd Z növelése
inc	szam;	;szam inkrementálása
dec	db	;db dekrementálása
brne	tolt	;vizsgálat Zero flag

loop:

rjmp loop

## 2.2 Közös feladat: Memória írása csökkenő címekre

```
;-----Adatmemória írása-----
;Feladat:    Írjon 15db növekvő 8 bites számot a memóriába
;            125 kezdőértéktől a 0x517-as memóriacímtől kezdődően
;            csökkenő címekre.
;-----


.INCLUDE "m128def.inc"      ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF szam = r20
.DEF db = r21;

.ORG 0x0
rjmp start           ; jump to start

.ORG 0x100
;-----;
; MACRO
;-----;KÖTELEZŐ!!!
.macro STACK_init      ;veremmutató (SP) a RAM terület végére fog mutatni
    ldi r16, high(RAMEND)      ;felső bájt
    out SPH, r16
    ldi r16, low(RAMEND)       ;alsó bájt
    out SPL, r16
.endmacro
;-----;
; SUBRUTINE
;-----;
;-----;
; INTERRUPT
;-----;
;-----;
; PROGRAM
;-----;
start:
;-----;
; INIT
;-----;
STACK_init
;-----;
; PROGRAM
;-----;

    ldi ZL, low (0x517)        ;0x517 alsó bájtja a Z indexregiszter alsó bájtjába
    ldi ZH, high (0x517)        ;0x517 felső bájtja a Z indexregiszter felső bájtjába

    ldi szam, 125              ;kezdőérték
    ldi db, 15                  ;darabszám
```

adiw ZH:ZL, 1 ;az st Z-, szam miatt Z növelése eggyel

tolt:

st -Z, szam ;Z csökkentése, szam memóriába töltése

inc szam ;szam inkrementálása

dec db ;db dekrementálása

brne tolt ;vizsgálat Zero flag

loop:

rjmp loop

## 2.3 Közös feladat: 2bájtos szám memóriába írása

```
;----- Adatmemória írása -----
;Feladat:      Írjon 5db növekvő 16 bites számot a memóriába
;              270 kezdőértéktől a 0x512-as memóriacímtől
;              kezdődően növekvő címekre.
;

.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    szamH = r27
.DEF    szamL = r26
.DEF    db     = r22;

.ORG 0x0
        rjmp start          ; jump to start

.ORG 0x100
;
;----- ; MACRO
;----- ;KÖTELEZŐ!!!
.macro STACK_init      ;veremmutató (SP) a RAM terület végére fog mutatni
    ldi    r16, high(RAMEND)       ;felső bájt
    out   SPH, r16
    ldi    r16, low(RAMEND)        ;alsó bájt
    out   SPL, r16
.endmacro
;
;----- ; SUBROUTINE
;----- ;
;----- ; INTERRUPT
;----- ;
;----- ; PROGRAM
;----- ;
.start:
;
;----- ; INIT
;----- ;
STACK_init
;
;----- ; PROGRAM
;----- ;

ldi    ZL, low (0x512)      ;0x512 alsó bájtja a Z indexregiszter alsó bájtjába
ldi    ZH, high (0x512)      ;0x512 felső bájtja a Z indexregiszter felső bájtjába

ldi    szamL, low(270)       ;kezdőérték
ldi    szamH, high(270)
```

ldi db, 5	;darabszám
tolt:	
st Z+, szamL	;szamL memóriába töltése, majd Z növelése
st Z+, szamH	;szamL memóriába töltése, majd Z növelése
adiw szamH:szamL, 1	;szam inkrementálása
dec db	;db dekrementálása
brne tolt	;vizsgálat Zero flag
loop:	
rjmp loop	

## 2.4 Önálló feladatok

- Írjon a 0x635 memóriaterülettől kezdődően csökkenő memóriacímekre 12db 8bites értéket. A kezdőérték 93 legyen, kettővel csökkenő számokat írjon a megadott területre: 93, 91, 89...
- Írjon 8db csökkenő 16 bites számot a memóriába 310 kezdőértéktől a 0x449-es memóriacímtől kezdődően csökkenő címekre.

### 3 Adatmemória olvasása

#### 3.1 Közös feladat: Adatmemóriába írás, olvasás, összegzés

```
;-----Adatmemória írása és olvasása-----
;Feladat:      Írjon 10db növekvő 8 bites számot a memóriába
;              12 kezdőértéktől a 0x123-as memóriacímtől kezdődően
;              növekvő címekre.
;              Olvassa vissza az adatmemóriába beírt értékeket, összegezze,
;              majd jelenítse meg a LED-eken
;-----
```

```
.INCLUDE "m128def.inc"                                ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    sum    =    r16
.DEF    LED    =    r17
.DEF    szam   =    r20
.DEF    db     =    r21

.ORG 0x0
        rjmp start           ; jump to start

.ORG 0x100
;-----
```

**; MACRO**

```
;-----          ;KÖTELEZŐ!!!
.macro STACK_init      ;veremmutató (SP) a RAM terület végére fog mutatni
    ldi    r16, high(RAMEND)      ;felső bájt
    out    SPH, r16
    ldi    r16, low(RAMEND)       ;alsó bájt
    out    SPL, r16
.endmacro
```

```
.macro LED_init
    ldi    r16, 0xF0
    out    DDRB, r16
    out    DDRD, r16
.endmacro
```

```
;-----
```

**; SUBROUTINE**

```
;-----
```

```
LED_out:
    out    PORTD, LED
    swap  LED
    out    PORTB, LED
    swap  LED
ret
```

```
;-----
```

```

;INTERRUPT
;-----
;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
STACK_init
LED_init
;-----
;PROGRAM
;-----

```

Ildi ZL, low (0x123) ;0x123 alsó bájtja a Z indexregiszter alsó bájtjába  
Ildi ZH, high (0x123) ;0x123 felső bájtja a Z indexregiszter felső bájtjába

Ildi szam, 12	;kezdőérték
Ildi db, 10	;darabszám

tolt:

st	Z+, szam	;szam memóriába töltése, majd Z növelése
inc	szam;	;szam inkrementálása
dec	db	;db dekrementálása
brne	tolt	;vizsgálat Zero flag

Ildi db, 10	;darabszám	
eor	sum, sum	;sum nullázása

Ildi ZL, low (0x123) ;0x123 alsó bájtja a Z indexregiszter alsó bájtjába  
Ildi ZH, high (0x123) ;0x123 felső bájtja a Z indexregiszter felső bájtjába

olvas:

ld	szam, Z+	;memóriából érték betöltése szam-ba, majd Z
növelése		
add	sum, szam	;összegzés
dec	db	;db csökkentése
brne	olvas	;vizsgálat Zero flag

mov	LED, sum	
call	LED_out	;LED_out szubrutinhívás

loop:

rjmp loop

### 3.2 Közös feladat: Memóriába írás, olvasás, összegzés, szubrutin, 16bit

```

;-----Adatmemória írása, olvasása-----
;Feladat: Írjon 12db hárommal csökkenő 16 bites számot a memóriába
;          278 kezdőértéktől a 0x445-ös memóriacímtől kezdődően
;          csökkenő címekre.
;          A memória írását és olvasását külön szubrutinban valósítsa meg.
;

.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    szamH =      r27
.DEF    szamL =      r26
.DEF    sumH =       r29
.DEF    sumL =       r28
.DEF    db     =      r22;

.ORG 0x0
        rjmp start           ; jump to start

.ORG 0x100
;
;-----;
; MACRO
;-----;
;KÖTELEZŐ!!!
.macro STACK_init      ;veremmutató (SP) a RAM terület végére fog mutatni
    ldi    r16, high(RAMEND)   ;felső bájt
    out   SPH, r16
    ldi    r16, low(RAMEND)    ;alsó bájt
    out   SPL, r16
.endmacro
;
;-----;
; SUBROUTINE
;-----;
tolt_sub:             ;memória írása
    ldi    ZL, low (0x445)    ;0x446 alsó bájtja a Z indexregiszter alsó bájtjába
    ldi    ZH, high (0x445)   ;0x446 felső bájtja a Z indexregiszter felső bájtjába
    ldi    szamL, low (278)   ;kezdőérték
    ldi    szamH, high (278)
    ldi    db, 12              ;darabszám
    adiw  ZH:ZL, 1            ;az st Z-, szam miatt Z növelése eggyel

tolt:
    st     -Z, szamH          ;Z csökkentése, szamL memóriába töltése
    st     -Z, szamL          ;Z csökkentése, szamH memóriába töltése
    sbiw  szamH:szamL, 3
    dec   db                  ;db dekrementálása
    brne toltx                ;vizsgálat Zero flag

ret

```

```

olvas_sub:                                ;memória olvasása és összegzés
    ldi    db, 12                          ;darabszám
    eor    sumL, sumL                      ;sumL nullázása
    eor    sumH, sumH                      ;sumH nullázása
    olvas:
        ld     szamH, Z+                  ;memoriából érték betöltése szamH-ba,
        ld     szamL, Z+                  ;majd Z növelése
        add   sumL, szamL                ;memoriából érték betöltése szamL-ba
        adc   sumH, szamH                ;majd Z növelése
        dec   db                         ;db csökkentése
        brne  olvas                     ;vizsgálat Zero falg
ret
;-----
;INTERRUPT
;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
STACK_init
;-----
;PROGRAM
;-----
call   toltsub                      ;szubrutin hívások
call   olvas_sub

loop:
rjmp  loop

```

### 3.3 Önálló feladatok

1. A 0x123-tól kezdődően növekvő címekre írjon 10db csökkenő értéket a memóriába, a kezdőérték 0x23. Olvassa vissza a memóriába írt értékeket és minden másodikat összegezze. Az eredményt tegye ki a LED-ekre.
2. A 0x327 címtől kezdődően csökkenő címekre írjon 11db növekvő értéket a memóriába, a kezdőérték 12. Olvassa vissza a memóriába írt értékeket és vegye a szorzatukat. Az eredmény felső bájtját írja a LED-ekre. A LED\_out-ot szubrutinban valósítsa meg.

## 4 Másolás

### 4.1 Közös feladat: Memóriaterület másolása

```
;-----Másolás-----
;Feladat:      Másolja át az adatmemória területére
;              a 0x543 címtől kezdően a következő
;              10 elemű "tömböt": 1, 3, 5, 7, 11, 25, 42, 14, 9, 8
;              a feladatot szubrutinban valósítsa meg
;

.INCLUDE "m128def.inc"          ; Include definitions Atmega128
.DSEG
    .ORG 0x543                  ;SRAM 0x543-as cím
    cel_tomb: .BYTE 10           ;10byte lefoglalása a cel_tomb-nek

.CSEG
    .DEF ciklus = r20
    .DEF tmp = r16
    .DEF LED = r17
    .ORG 0x0
        rjmp start             ; jump to start

    .ORG 0x110
;

;MACRO
;----- ;KÖTELEZŐ!!!
.macro STACK_init ;veremmutató (SP) a RAM terület végére fog mutatni
    ldi r16, high(RAMEND)      ;felső bájt
    out SPH, r16
    ldi r16, low(RAMEND)       ;alsó bájt
    out SPL, r16
.endmacro
;

;SUBRUTINE
;

masol_sub:
    ldi ZL, low(forras_tomb<<1) ;forrás alsó címe
    ldi ZH, high(forras_tomb<<1) ;forrás felső címe
    ldi XL, low(cel_tomb)         ;cél alsó címe
    ldi XH, high(cel_tomb)        ;cél felső címe
    ldi ciklus, 10                ;ciklus számláló

masol:
    lpm tmp, Z+                 ;forrásból az átmeneti tárolóba
    st X+, tmp                  ;átmeneti tárolóból a célba
    dec ciklus                  ;ciklus számláló csökkent
    brne masol                  ;vissza, amíg el nem fogyott

ret
;

;INTERRUPT
;
```

```

;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
STACK_init
;-----
; PROGRAM
;-----
call masol_sub

loop:
    rjmp loop

forras_tomb: .DB 1, 3, 5, 7, 11, 25, 42, 14, 9, 8
;forras_tomb lista, 10 elem

```

## 4.2 Önálló feladat

1. Másolja át az adatmemória területére a 0x475 címtől kezdően a következő 8 elemű "tömböt": 1, 15, 2, 3, 5, 2, 2, 74 a feladatot szubrutinban valósítsa meg. Olvassa vissza és összegezze az értékeket. Az eredményt jelenítse meg a LED-eken.

## 5 Gyakorló feladatok

1. Készítsen Knight Rider futófényt, melyet szubrutinban valósít meg.
2. A GOMB0 lenyomására adja össze a következő számokat: 43, 52. Írja az eredményt az adatmemóriába a 0x456 címre. A feladatot szubrutinban valósítsa meg.
3. A GOMB1 lenyomására töltön az adatmemóriába 20db növekvő páratlan számot a 0x543 memóriacímtől kezdődően. Az első szám a 257 legyen. A feladatot szubrutinban valósítsa meg.
4. A GOMB2 lenyomására olvassa vissza a 3. feladatban az adatmemóriába írt értékeket és összegyezz azokat. Az eredmény felső bájtját jelenítse meg a LED-eken.
5. A GOMB3 lenyomására töltön az adatmemóriába 20db csökkenő páros számot a 0x245 memóriacímtől kezdődően visszafele. Az első szám a 752 legyen. A feladatot szubrutinban valósítsa meg.
6. A GOMB4 lenyomására olvassa vissza a 5. feladatban az adatmemóriába írt értékeket és összegyezz azokat. Az eredmény alsó bájtját jelenítse meg a LED-eken.

## 4. labor

### **Labor tematikája:**

1. Timer0 CTC, OFV (41-51. oldal)
2. Portkezelés alapjai: 7szegmenses kijelző 1digit (109-113. oldal)
3. Portkezelés alapjai: 7szegmenses kijelző 4digit
4. Megszakítások alkalmazása 1.

### **Timer/Counter0:**

- Timer/Counter0
  - o ATmega64 adatlap 93. oldaltól
  - o ATmega128 adatlap 92. oldaltól

# 1 Timer0 CTC, OVF

## 1.1 Közös feladat: Timer0, /1024/250, rotálás

```
;-----TIM0 CTC-----
;Feladat:      Írja meg a Timer0-hoz tartozó CTC mód init-jét
;              Előosztásnak 1024-et állítson be, komparálási értéknek 249-et:
;              32/16ms
;              Állítja a LED értékét 0x01-re
;              a megszakítást kiszolgáló rutinban rotálja balra
;              és tegye ki a LED-ekre.
;-----;
;SEI, RETI

.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    tmp    =    r16
.DEF    LED    =    r17

.ORG 0x0
    rjmp start           ; jump to start
.ORG 0x1E                 ;$001E TIMERO CTC Timer/Counter0 Compare Match
    rjmp TIM0_CTC_IT
.ORG 0x100
;-----;
; MACRO
;-----;
.macro STACK_init
    ldi    tmp, HIGH(RAMEND)
    out   SPH, tmp
    ldi    tmp, LOW(RAMEND)
    out   SPL, tmp
.endmacro

.macro LED_init
    ldi    tmp, 0xF0
    out   DDRB, tmp
    out   DDRD, tmp
.endmacro

.macro TIM0_CTC_init          ;TIM0 CTC init
    ldi    tmp, 0b00001111  ;1024-es előosztás beállítása
    out   TCCR0, tmp        ;Timer/Counter Control Register – TCCR0
                           ; 7   6   5   4   3   2   1   0
                           ;FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00
    ldi    tmp, 0x02          ;Timer/Counter0 Overflow Interrupt Enable
    out   TIMSK, tmp         ;Timer/Counter Interrupt Mask Register – TIMSK
                           ; 7   6   5   4   3   2   1   0
                           ;          OCIE0 TOIE0
```

```

ldi    tmp, 249      ;16000000/1024/250  8000000/1024/250   32,16ms
out    OCR0, tmp
sei
;Sets the Global Interrupt flag (I) in SREG (status register).
.endmacro
;-----
;SUBROUTINE
;-----
LED_out:
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
ret
;-----
;INTERRUPT
;-----
TIM0_CTC_IT:           ;TIM0 CTC megszakítást kiszolgáló rutin
    call   LED_out        ;LED rotálás
    rol    LED
reti
;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
STACK_init
LED_init
TIM0_CTC_init
;-----
;PROGRAM
;-----
ldi    LED, 0x01

loop:
    rjmp  loop

```

## 1.2 Közös feladat: TIM0 CTC, LED2 villogtatása

```
;-----TIM0 CTC-----
;Feladat:      Villogtassa a LED2-t megközelítőleg másodpercenként.
;              A feladathoz megszakítást használjon.
;-----  
;SEI, RETI

.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    tmp    =    r16
.DEF    LED    =    r17
.DEF    szam   =    r18

.ORG 0x0
    rjmp start             ; jump to start
.ORG 0x1E
Match
    rjmp TIM0_CTC_IT

.ORG 0x100
;-----  
; MACRO
;-----  
.macro STACK_init
    ldi    tmp, HIGH(RAMEND)
    out   SPH, tmp
    ldi    tmp, LOW(RAMEND)
    out   SPL, tmp
.endmacro

.macro LED_init
    ldi    tmp, 0xF0
    out   DDRB, tmp
    out   DDRD, tmp
.endmacro

.macro TIM0_CTC_init          ;TIM0 CTC init
    ldi    tmp, 0b00001111  ;1024-es előosztás beállítása
    out   TCCR0, tmp        ;Timer/Counter Control Register – TCCR0
                           ; 7   6   5   4   3   2   1   0
                           ;FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00
    ldi    tmp, 0x02         ;Timer/Counter0 Overflow Interrupt Enable

    out   TIMSK, tmp        ;Timer/Counter Interrupt Mask Register – TIMSK
                           ; 7   6   5   4   3   2   1   0
                           ;          ;OCIE0 TOIE0
    ldi    tmp, 249          ;16000000/1024/250  8000000/1024/250  32ms 16ms
    out   OCR0, tmp
```

```

        sei          ;Sets the Global Interrupt flag (I) in SREG (status register).
.endmacro
;-----
; SUBROUTINE
;-----
LED_out:
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
ret
;-----
; INTERRUPT
;-----
TIM0_CTC_IT:           ;TIM0 CTC megszakítást kiszolgáló rutin
    inc    szam
reti
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
STACK_init
LED_init
TIM0_CTC_init
;-----
; PROGRAM
;-----
ldi    LED, 0x04          ;LED2
eor    szam, szam         ;szam 0 kezdőérték
ldi    tmp, 0x04

loop:
    cpi   szam,62          ;16000000/1024/250/62      ~1s
    breq  kitesz
    brne  loop

kitesz:
    eor   szam, szam        ;szam törlése
    call   LED_out          ;LED kitétele a LED-ekre
    eor   LED, tmp           ;"villogtatás"

rjmp  loop

```

### 1.3 Közös feladat: Timer0 OVF, 1024 előosztás, LED0 villogtatása

```
;-----TIM0 OVF-----
;Feladat:      Írja meg a Timer0-hoz tartozó OVF mód init-jét
;              Előosztásnak 1024-et állítson be.
;              A megszakítást kiszolgáló rutinban villogtassa a LED0-t.
;-----  
;SEI, RETI

.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    tmp    =      r16
.DEF    LED    =      r17

.ORG 0x0
        rjmp start          ; jump to start

.ORG 0x20                      ;$0020 TIMER0 OVF Timer/Counter0 Over-
flow
        rjmp TIM0_OVF_IT
.ORG 0x100
;-----  
; MACRO
;-----  
.macro STACK_init
    ldi    tmp, HIGH(RAMEND)
    out   SPH, tmp
    ldi    tmp, LOW(RAMEND)
    out   SPL, tmp
.endmacro

.macro LED_init
    ldi    tmp, 0xF0
    out   DDRB, tmp
    out   DDRD, tmp
.endmacro

.macro TIM0_OVF_init          ;TIM0 OVF init
    ldi    tmp, 0b00000111      ;1024-es előosztás beállítása
    out   TCCRO, tmp          ;Timer/Counter Control Register – TCCRO
                                ; 7   6   5   4   3   2   1   0
                                ;FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00
    ldi    tmp, 0x01          ;Timer/Counter0 Overflow Interrupt Enable

    out   TIMSK, tmp          ;Timer/Counter Interrupt Mask Register – TIMSK
                                ; 7   6   5   4   3   2   1   0
                                ;          OCIE0 TOIE0
    sei                          ;Sets the Global Interrupt flag (I) in SREG (status register).
.endmacro
```

```

;-----
; SUBROUTINE
;-----
LED_out:
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
ret
;-----
; INTERRUPT
;-----
TIM0_OVF_IT:                      ;TIM0 OVF megszakítást kiszolgáló rutin
    call   LED_out                 ;LED villogtatás
    eor    LED, tmp
reti
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
STACK_init
LED_init
TIM0_OVF_init
;-----
; PROGRAM
;-----
ldi    LED, 0x01
ldi    tmp, 0x01

loop:
rjmp  loop

```

## 1.4 Önálló feladat

1. Villogtassa a LED6-ot ~0,5s-os időzítéssel, használjon megszakítást a feladat megvalósításához.
2. Készítsen felfele számlálót, mely 0-255-ig számol folyamatosan a LED-eken, ~1s-onként. A feladathoz megszakítást használjon.

## 2 Portkezelés alapjai: 7szegmenses kijelző 1digit

bit	7	6	5	4	3	2	1	0	
PORT									
A	ENABLE	SEL2	SEL1	SEL0	DATA3	DATA2	DATA1	DATA0	7 SEGMENT DISPLAY
I/O	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	
B	Led3	Led2	Led1	Led0					LED/Io 4bit
I/O	OUT	OUT	OUT	OUT					
C	RED	KBD4row	KBD3row	KBD2row	KBD1row	KBD_right	KBD_centr	KBD_left	Keyboard
I/O	OUT	OUT	OUT	OUT	OUT	IN	IN	IN	
D	Led7	Led6	Led5	Led4					LED/hi 4bit
I/O	OUT	OUT	OUT	OUT					
E	LCD_DATA7	LCD_DATA6	LCD_DATA5	LCD_DATA4	GREEN	BLUE			LCD data
I/O	OUT	OUT	OUT	OUT	OUT	OUT			
F					LCD_E	LCD_R/W	LCD_RS	LM35	LCD Control
I/O					OUT	OUT	OUT	IN	Analog
G	NC	NC	NC	K4	K3	K2	K1	K0	Pushbutton
I/O	X	X	X	IN	IN	IN	IN	IN	
bit	7	6	5	4	3	2	1	0	

7 Segment display: E 1:Enable, 0: Disable; SEL 2-1-0: 000:10exp0, 001: 10exp1, 010:10exp2, 011:10exp3, 100:Double Leds

## 2.1 Közös feladat: 7szegmenses kijelző 1digit

```
;-----7szeg-----
;Feladat:      Tegye ki a 7szegmenses kijelző 2. digitre a 7-es számot
;              Írjon szubrutint, amely a 7szegmenses kijelző
;              tetszőleges digitre (0-3)
;              tetszőleges számot (0-9) tesz ki
;-----;
;PORTA

.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    tmp    =    r16
.DEF    LED    =    r17
.DEF    EN     =    r18
.DEF    digit  =    r19
.DEF    szam   =    r20

.ORG 0x0
    rjmp start             ; jump to start
.ORG 0x100
;-----;
; MACRO
;-----;
.macro init
    ldi    tmp, 0xF0
    out   DDRB, tmp          ;LED-ek
    out   DDRD, tmp
    ldi    tmp, 0xFF          ;7szegmenses kijelző, kimenetre állítás
    out   DDRA, tmp
.endmacro

.macro STACK_init                ;KÖTELEZŐ!
    ldi    tmp,high(RAMEND)
    out   SPH,tmp
    ldi    tmp,low(RAMEND)
    out   SPL,tmp
.endmacro

;-----;
;SUBROUTINE
;-----;
sub_7seg_digit:
    ldi    EN, 0x80            ;0x80|digit|szam
    eor    tmp, tmp
    or     tmp, EN              ; engedélyezés
    swap  digit
    or     tmp, digit          ; digit 0-3
    swap  digit
    or     tmp, szam            ; szám 0-9
```

```

        out    PORTA, tmp
ret
;-----
;INTERRUPT
;-----
;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
init
STACK_init
;-----
;PROGRAM
;-----
ldi    szam, 7           ;7-es szám kitétele
ldi    digit, 2          ;a 2. digitre
call   sub_7seg_digit
loop:
rjmp  loop

```

## 2.2 Önálló feladatok

1. A közösen elkészített kódot próbálja ki a további digiterekre is. Tegye ki a digiterekre a következő értékeket:  
 - digit0: 2  
 - digit1: 9  
 - digit3: 4  
 - kapcsolja be a kettőspontot
2. Gombnyomásra (G2) összegezze a következő számokat és jelenítse meg az eredményt a 7szegmenses kijelzőn: 2, 4.

### 3 Portkezelés alapjai: 7szegmenses kijelző 4digit

#### 3.1 Közös feladat: 4jegyű szám megjelenítése a 7szegmenses kijelzőn

```
;-----7szeg-----
;Feladat:      Jelenítse meg a 7szegmenses kijelzőn a 1234 számot.
;           A feladat megoldásához szubrutint használjon.
;-----;
;PORTA

.INCLUDE "m128def.inc"                                ; Include definitions Atmega128
.DSEG
.org 0x100
    digit_s:     .BYTE  4                           ;A 4jegyű szám
.CSEG
.DEF    tmp    =    r16
.DEF    LED    =    r17
.DEF    EN     =    r18
.DEF    digit  =    r19
.DEF    szam   =    r20
.DEF    szamjegy=  r21

.ORG 0x0
    rjmp start                                     ; jump to start

.ORG 0x100
;-----;
; MACRO
;-----;
.macro init
    ldi    tmp, 0xF0
    out   DDRB, tmp                               ;LED-ek
    out   DDRD, tmp
    ldi    tmp,0xFF
    out   DDRA, tmp                               ;7szegmenses kijelző, kimenetre állítás
.endmacro

.macro STACK_init                                ;KÖTELEZŐ!
    ldi    tmp,high(RAMEND)
    out   SPH,tmp
    ldi    tmp,low(RAMEND)
    out   SPL,tmp
.endmacro

;-----;
; SUBROUTINE
;-----;
delay:
    ldi    r26, 0xFF
delay_loop:
    dec   R26                                     ; ha 0 → Z flag=1
```

```

        brne delay_loop           ; Z flag-et figyeli, ha Z flag=0 → ugrik

        ret

sub_7seg_digit:
        ldi    EN, 0x80          ;0x80|digit|szam
        eor    tmp, tmp
        or     tmp, EN           ; engedélyezés
        swap   digit
        or     tmp, digit        ; digit 0-3
        swap   digit
        or     tmp, szam          ; szám 0-9
        out    PORTA, tmp
        ret

sub_7seg:
        cpi    digit, 4          ;összehasonlítás
        breq   nullaz

kiir:
        ld     szam, Z+          ;memóriából érték betöltése
        call   sub_7seg_digit    ;megjelenítés az aktuális digiten
        inc    digit              ;digit növelése
        rjmp   vege

nullaz:
        eor    digit, digit       ;digit nullázása
        ldi    ZL, low(digit_s)  ;visszaállítás a memória címre 0x100
        ldi    ZH, high(digit_s)

vege:
        ret

;-----
;INTERRUPT
;-----
;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
init
STACK_init
;-----
;PROGRAM
;-----
ldi    ZL, low(digit_s)      ;digit_s kezdőcímére visszaállítás
ldi    ZH, high(digit_s)
eor   digit, digit           ; törlés
eor   szam, szam
ldi    szamjegy, 4

```

tolt:

```
    mov    tmp, szamjegy           ;1234 betöltése
    st     Z+, tmp
    dec    szamjegy
    brne  tolt
```

ldi ZL, low(digit\_s) ;digit\_s kezdőcímére visszaállítás

ldi ZH, high(digit\_s)

loop:

```
    call   sub_7seg            ;folyamatos szubrutin hívások
```

```
    call   delay
```

```
    rjmp  loop
```

### 3.2 Extra közös feladat: 7szegmenses kijelző 4 digit, delay

```
;-----7szeg-----
;Feladat:      Jelenítse meg a 7szegmenses kijelzőn a 123-at 4es számrendszerben.
;           A feladat megoldásához szubrutint használjon.
;-----  
;PORTA  
  
.INCLUDE "m128def.inc"          ; Include definitions Atmega128
.DSEG
;  
.CSEG
.DEF    tmp    =    r16
.DEF    LED    =    r17
.DEF    EN     =    r18
.DEF    digit  =    r19
.DEF    szam   =    r20
.DEF    szam_3 =    r21
.ORG 0x0
    rjmp start           ; jump to start  
  
.ORG 0x100
;-----  
; MACRO
;-----  
.macro init
    ldi    tmp, 0xF0
    out   DDRB, tmp          ;LED-ek
    out   DDRD, tmp
    ldi    tmp, 0xFF
    out   DDRA, tmp          ;7szegmenses kijelző, kimenetre állítás
.endmacro  
  
.macro STACK_init           ;KÖTELEZŐ!
    ldi    tmp,high(RAMEND)
    out   SPH,tmp
    ldi    tmp,low(RAMEND)
    out   SPL,tmp
.endmacro  
;  
;SUBRUTINE
;-----  
delay:
    ldi    r26, 0xFF  
  
delay_loop:
    dec   R26                ; ha 0 → Z flag=1
    brne delay_loop          ; Z flag-et figyeli, ha Z flag=0 → ugrik  
  
ret
```

```

sub_7seg_digit:
    ldi    EN, 0x80          ;0x80|digit|szam
    eor    tmp, tmp
    or     tmp, EN           ; engedélyezés
    swap   digit
    or     tmp, digit        ; digit 0-3
    swap   digit
    or     tmp, szam          ; szám 0-9
    out    PORTA, tmp

ret

sub_7seg:
    inc    digit             ;digit növelése

    cpi    digit, 1           ;vizsgálatok
    breq   tizes

    cpi    digit, 2
    breq   szazas

    cpi    digit, 3
    breq   ezres

    eor    digit, digit       ;digit nullázása

egyes:
    mov    tmp, szam_3         ;1-es helyiérték
    andi   tmp, 0x03           ;maszkolás
    mov    szam, tmp
    call   sub_7seg_digit      ;0. digitre megfelelő érték kitétele
    rjmp   vege

tizes:
    mov    tmp, szam_3         ;10-es helyiérték
    lsr    tmp
    lsr    tmp
    andi   tmp, 0x03           ;maszkolás
    mov    szam, tmp
    call   sub_7seg_digit      ;1. digitre megfelelő érték kitétele
    rjmp   vege

szazas:
    mov    tmp, szam_3         ;100-as helyiérték
    swap   tmp
    andi   tmp, 0x03           ;alsó és felső 4 bit cseréje
    mov    szam, tmp           ;maszkolás
    call   sub_7seg_digit      ;2. digitre megfelelő érték kitétele
    rjmp   vege

```

```

ezres:                                ;1000-s helyiérték
    mov    tmp, szam_3
    swap   tmp          ;alsó és felső 4 bit cseréje
    lsr    tmp          ;2 shift jobbra
    lsr    tmp          ;maszkolás
    andi   tmp, 0x03
    mov    szam, tmp
    call   sub_7seg_digit ;3. digitre megfelelő érték kitétele

vege:
ret

;-----
; INTERRUPT
;-----
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
init
STACK_init
;-----
; PROGRAM
;-----
ldi    szam_3, 123
eor    digit, digit
eor    szam, szam

loop:
    call   sub_7seg      ;folyamatos szubrutin hívások
    call   delay
    rjmp  loop

```

### 3.3 Önálló feladat

- Írjon szubrutint, amely 8-as számrendszerben jelenít meg értékeket a 7szegmenses kijelzőn.

## 4 Megszakítások alkalmazása 1.

### 4.1 Közös feladat: 7szegmenses 4digit, megszakítás

```
;-----7szeg-----
;Feladat:      Jelenítse meg a 7szegmenses kijelzőn a 1234 számot.
;              A feladat megoldásához szubrutint és megszakítást használjon.
;-----  
.INCLUDE "m128def.inc"                      ; Include definitions Atmega128
.DSEG
.org 0x100
    digit_s:     .BYTE  4                  ;A 4jegyű szám
.CSEG
.DEF    tmp    =    r16
.DEF    LED    =    r17
.DEF    EN     =    r18
.DEF    digit  =    r19
.DEF    szam   =    r20
.DEF    szamjegy=  r21
.ORG 0x0
    rjmp start                         ; jump to start
.ORG 0x1E
    rjmp TIM0_CTC_IT
.ORG 0x100
;-----  
;  
; MACRO
;-----  
.macro init
    ldi    tmp, 0xF0
    out   DDRB, tmp                     ;LED-ek
    out   DDRD, tmp
    ldi    tmp, 0xFF
    out   DDRA, tmp                     ;7szegmenses kijelző, kimenetre állítás
.endmacro

.macro STACK_init                      ;KÖTELEZŐ!
    ldi    tmp,high(RAMEND)
    out   SPH,tmp
    ldi    tmp,low(RAMEND)
    out   SPL,tmp
.endmacro

.macro TIM0_CTC_init                 ;TIM0 CTC init
    ldi    tmp, 0b00001111            ;1024-es előosztás beállítása
    out   TCCR0, tmp                 ;Timer/Counter Control Register - TCCR0
                                ; 7   6   5   4   3   2   1   0
                                ;FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00
    ldi    tmp, 0x02                 ;Timer/Counter0 Overflow Interrupt Enable
    out   TIMSK, tmp                 ;Timer/Counter Interrupt Mask Register - TIMSK
                                ; 7   6   5   4   3   2   1   0
                                ;          OCIE0 TOIE0
```

```

ldi    tmp, 29      ;16000000/1024/30   8000000/1024/30   ~520Hz
out   OCR0, tmp
sei
;Sets the Global Interrupt flag (I) in SREG (status register).
.endmacro
;-----
;; SUBROUTINE
;-----
sub_7seg_digit:
ldi    EN, 0x80      ;0x80|digit|szam
eor   tmp, tmp
or    tmp, EN        ; engedélyezés
swap  digit
or    tmp, digit    ; digit 0-3
swap  digit
or    tmp, szam     ; szám 0-9
out   PORTA, tmp
ret

sub_7seg:
cpi   digit, 4      ;összehasonlítás
breq nullaz

kiir:
ld    szam, Z+      ;memoriából érték betöltése
call  sub_7seg_digit ;megjelenítés az aktuális digiten
inc   digit         ;digit növelése
rjmp  vege

nullaz:
eor   digit, digit  ;digit nullázása
ldi   ZL, low(digit_s) ;visszaállítás a memória címre 0x100
ldi   ZH, high(digit_s)

vege:
ret
;-----
;; INTERRUPT
;-----
TIM0_CTC_IT:
call  sub_7seg
reti
;-----
;; PROGRAM
;-----
start:
;-----
;; INIT
;-----
init
STACK_init
TIM0_CTC_init

```

```

;-----
; PROGRAM
;-----
ldi    ZL, low(digit_s)      ;digit_s kezdőcímére visszaállítás
ldi    ZH, high(digit_s)
eor   digit, digit          ; törlés
eor   szam, szam
ldi    szamjegy, 4

tolt:
    mov   tmp, szamjegy      ;1234 betöltése
    st    Z+, tmp
    dec   szamjegy
    brne tolta

ldi    ZL, low(digit_s)      ;digit_s kezdőcímére visszaállítás
ldi    ZH, high(digit_s)

loop:
    rjmp  loop

```

## 4.2 Extra közös feladat: 7szegmenses kijelző 4digit, megszakítás

```

;-----7szeg-----
;Feladat: Jelenítse meg a 7szegmenses kijelzőn a 123-at 4es számrendszerben.
; A feladat megoldásához szubrutint használjon.
; Az időzítéshez használjon megszakítást.
;-----PORTA

.INCLUDE "m128def.inc" ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF tmp = r16
.DEF LED = r17
.DEF EN = r18
.DEF digit = r19
.DEF szam = r20
.DEF szam_3 = r21
.ORG 0x0
rjmp start ; jump to start

.ORG 0x1E ;$001E TIMERO CTC Timer/Counter0 Compare Match
rjmp TIM0_CTC_IT
.ORG 0x100
;-----;
; MACRO
;-----
.macro init
    ldi tmp, 0xF0
    out DDRB, tmp ;LED-ek
    out DDRD, tmp
    ldi tmp, 0xFF ;7szegmenses kijelző, kimenetre állítás
    out DDRA, tmp
.endmacro

.macro TIM0_CTC_init ;TIM0 CTC init
    ldi tmp, 0b00001111 ;1024-es előosztás beállítása
    out TCCR0, tmp ;Timer/Counter Control Register - TCCR0
    ; 7   6   5   4   3   2   1   0
    ;FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00
    ldi tmp, 0x02 ;Timer/Counter0 Overflow Interrupt Enable

    out TIMSK, tmp ;Timer/Counter Interrupt Mask Register - TIMSK
    ; 7   6   5   4   3   2   1   0
    ;          OCIE0 TOIE0
    ldi tmp, 29 ;16000000/1024/30 8000000/1024/30 ~520Hz
    out OCR0, tmp
    sei ;Sets the Global Interrupt flag (I) in SREG (status register).
.endmacro

```

```

.macro STACK_init ;KÖTELEZŐ!
    ldi    tmp,high(RAMEND)
    out    SPH,tmp
    ldi    tmp,low(RAMEND)
    out   SPL,tmp
.endmacro
;-----
; SUBROUTINE
;-----
sub_7seg_digit:
    ldi    EN, 0x80      ;0x80|digit|szam
    eor    tmp, tmp
    or     tmp, EN       ; engedélyezés
    swap   digit
    or     tmp, digit   ; digit 0-3
    swap   digit
    or     tmp, szam    ; szám 0-9
    out   PORTA, tmp
ret

sub_7seg:
    inc    digit         ;digit növelése
    cpi    digit, 1      ;vizsgálatok
    breq   tizes
    cpi    digit, 2
    breq   szazas
    cpi    digit, 3
    breq   ezres
    cpi    digit, 4
    breq   nullaz

nullaz:                      ;digit nullázása
    eor    digit, digit
    rjmp   egyes

egyes:                       ;1-es helyiérték
    mov    tmp, szam_3
    andi   tmp, 0x03      ;maszkolás
    mov    szam, tmp
    call   sub_7seg_digit ;0. digitre megfelelő érték kitétele
    rjmp   vege

tizes:                        ;10-es helyiérték
    mov    tmp, szam_3
    lsr    tmp            ;2 shift jobbra
    lsr    tmp
    andi   tmp, 0x03      ;maszkolás
    mov    szam, tmp
    call   sub_7seg_digit ;1. digitre megfelelő érték kitétele
    rjmp   vege

```

```

szazas:                                ;100-as helyiérték
    mov    tmp, szam_3
    swap   tmp
    andi   tmp, 0x03
    mov    szam, tmp
    call   sub_7seg_digit
    rjmp  vege

ezres:                                 ;1000-s helyiérték
    mov    tmp, szam_3
    swap   tmp
    lsr    tmp
    lsr    tmp
    andi   tmp, 0x03
    mov    szam, tmp
    call   sub_7seg_digit

vege:
ret

;-----
;INTERRUPT
;-----
TIM0_CTC_IT:
    call   sub_7seg
reti
;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
init
STACK_init
TIM0_CTC_init
;-----
;PROGRAM
;-----
ldi    szam_3, 123
eor    digit, digit
eor    szam, szam

loop:
    rjmp  loop

```

### 4.3 Önálló feladatok

1. Készítsen egy számlálót, amely a 7szegmenes kijelző 1 digitjén számol 0-tól 9-ig folyamatosan ~1s-onként. A feladathoz megszakítást és szubrutint használjon.
2. Összegezze a következő számokat és az eredményt jelenítse meg a 7szegmenses kijelzőn 4-es számrendszerben: 12, 14, 15, 27. A feladathoz megszakítást és szubrutint használjon.
3. Vegye a 1245, 270 számok különbségét és az eredmény felső báját jelenítse meg a kijelzőn. A feladathoz megszakítást és szubrutint használjon.

## 5 Gyakorló feladatok

- Készítsen futótényt a következő lépésekkel:

7	6	5	4	3	2	1	0
x	x	0	0	0	0	x	x
x	0	0	x	x	0	0	x
0	0	x	x	x	x	0	0
x	0	0	x	x	0	0	x
x	x	0	0	0	0	x	x

...

A feladat elvégzéséhez megszakítást és szubrutint használjon. Két állapot kijelzése között ~0.5s eltérés legyen.

- Készítsen egyszerű gomb menüt az előző laboréhoz hasonlóan, de most az eredményeket a 7szegmenes kijelzőn jelenítse meg. A gombok állapotát nem folyamatosan olvassa. A feladat elvégzéséhez használjon szubrutint.
- Írjon szubrutint, amely 8 bites értékeket 8-as számrendszerben jelenít meg a 7szegmenes kijelzőn. A feladat megvalósításához megszakítást használjon.
- Készítsen 0-9ig számlálót a 7szegmenes kijelző 2. digitjére. A számlálást a G0 nyomógomb lenyomására indítsa, a G1 gombbal pedig lehessen leállítani a számlálást.

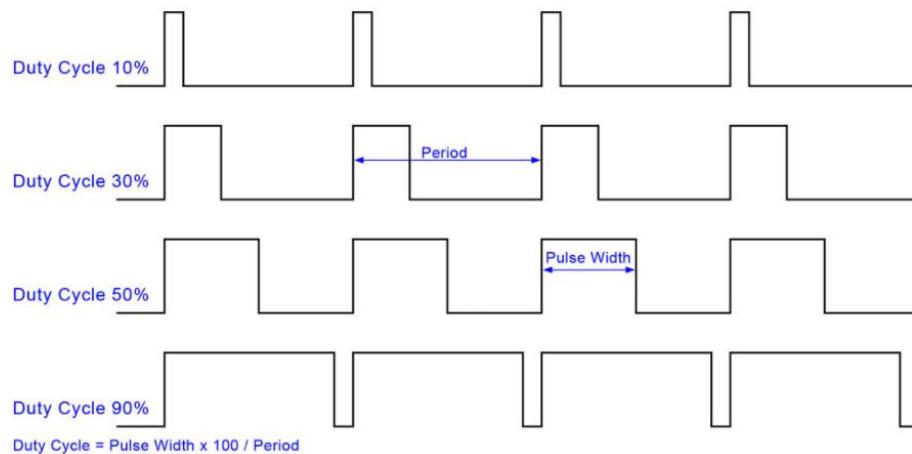
## 5. labor

### Labor tematikája:

1. Timer1 CTC, OVF (47-61. oldal)
2. Megszakítások alkalmazása 2.
3. PWM (61-69. oldal)
4. RGB LED (113-116. oldal)

### Timer/Counter:

- Timer/Counter0
  - o ATmega64 adatlap 93. oldaltól
  - o ATmega128 adatlap 92. oldaltól
- 16-bit Timer/Counter (Timer/Counter 1 and Timer/Counter3)
  - o ATmega64 adatlap 112. oldaltól
  - o ATmega128 adatlap 111. oldaltól



bit	7	6	5	4	3	2	1	0	
PORT	ENABLE	SEL2	SEL1	SEL0	DATA3	DATA2	DATA1	DATA0	7 SEGMENT DISPLAY
I/O	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	
B	Led3	Led2	Led1	Led0					LED/Io 4bit
I/O	OUT	OUT	OUT	OUT					
C	RED	KBD4row	KBD3row	KBD2row	KBD1row	KBD_right	KBD_centr	KBD_left	Keyboard
I/O	OUT	OUT	OUT	OUT	OUT	IN	IN	IN	
D	Led7	Led6	Led5	Led4					LED/hi 4bit
I/O	OUT	OUT	OUT	OUT					
E	LCD_DATA7	LCD_DATA6	LCD_DATA5	LCD_DATA4	GREEN	BLUE			LCD data
I/O	OUT	OUT	OUT	OUT	OUT	OUT			
F					LCD_E	LCD_R/W	LCD_RS	LM35	LCD Control
I/O					OUT	OUT	OUT	IN	Analog
G	NC	NC	NC	K4	K3	K2	K1	K0	Pushbutton
I/O	X	X	X	IN	IN	IN	IN	IN	
bit	7	6	5	4	3	2	1	0	

# 1 Timer1 CTC, OVF

## 1.1 Közös feladat: TIM1 OVF, LED0 villogtatása

```
;-----TIM1 CTC COMPA-----
;Feladat:      Villogtassa a LED0-t TIM1 OVF megszakítással
;              1024-es előosztást állítson
;              Állapítsa meg, hogy körülbelül ez mekkora időzítést jelent
;
INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    tmp    =      r16
.DEF    LED    =      r17

.ORG 0x0
    rjmp start             ; jump to start
.ORG 0x1C
    rjmp TIM1_OVF          ;$001C TIMER1 OVF Timer/Counter1 Overflow
.ORG 0x100
;
;-----;
; MACRO
;-----;
.macro TIM1_OVF_init
        ;TCCR1B
        ;ICNC1 ICES1 - WGM13 WGM12 CS12 CS11 CS10
        ; 0   0   0   0   0   1   0   1
    ldi    tmp, 0b00000101    ;CTC mód 1024 előosztás
    out    TCCR1B, tmp
        ;TIMSK
        ;OCIE2 TOIE2 TICIE1 OCIE1A OCIE1B TOIE1 OCIE0 TOIE0
        ; 0   0   0   0   0   1   0   0
    ldi    tmp, 0b00000100
    out    TIMSK, tmp
    sei
.endmacro

.macro STACK_init                ;KÖTELEZŐ
    ldi    tmp, HIGH(RAMEND)
    out    SPH, tmp
    ldi    tmp, LOW(RAMEND)
    out    SPL, tmp
.endmacro

.macro LED_init
    ldi    tmp, 0xF0
    out    DDRB, tmp
    out    DDRD, tmp
.endmacro
;
```

```

;SUBRUTINE
;-----
LED_out:
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
ret
;-----
;INTERRUPT
;-----
TIM1_OVF:
call    LED_out          ;LED villogtatása
eor    LED, tmp
reti
;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
STACK_init
TIM1_OVF_init
LED_init
;-----
;PROGRAM
;-----
ldi    LED, 0x01          ;LEDO
ldi    tmp, 0x01
loop:
        rjmp   loop

```

## 5.1 Közös feladat: TIM1 COMPA, LED villogtatás 1s

```

;-----TIM1 CTC COMPA-----
;Feladat:      Villogtassa a LED3-t TIM1 COMPA megszakítással
;              1s-os ütemben
;-----  

;.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    tmp    =     r16
.DEF    LED    =     r17

.ORG 0x0
    rjmp start             ; jump to start
.ORG 0x18                 ;$0018 TIMER1 COMPA Timer/Counter1 Compare Match A
    rjmp TIM1_COMPA
.ORG 0x100
;  

;  

; MACRO
;  

;.macro TIM1_CTC_init
        ;TCCR1B
        ;ICNC1 ICES1 – WGM13 WGM12 CS12 CS11 CS10
        ; 0   0   0   0   1   1   0   1
    ldi    tmp, 0b00001101      ;CTC mód 1024 előosztás
    out    TCCR1B, tmp
  

    ldi    tmp, 0x3D            ;ELŐSZÖR OCR1AH!!!!!!!
    out    OCR1AH, tmp
    ldi    tmp, 0x08            ;15625-1      max 65535
    out    OCR1AL, tmp
        ;TIMSK
        ;OCIE2 TOIE2 TICIE1 OCIE1A OCIE1B TOIE1 OCIE0 TOIE0
        ; 0   0   0   1   0   0   0   0
    ldi    tmp, 0b00010000
    out    TIMSK, tmp
    sei
.endmacro
  

.macro STACK_init
    ldi    tmp, HIGH(RAMEND)
    out    SPH, tmp
    ldi    tmp, LOW(RAMEND)
    out    SPL, tmp
.endmacro
  

.macro LED_init
    ldi    tmp, 0xF0
    out    DDRB, tmp
    out    DDRD, tmp
.endmacro

```

```

;-----
; SUBROUTINE
;-----
LED_out:
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
ret
;-----
; INTERRUPT
;-----
TIM1_COMPA:
call    LED_out          ;LED villogtatása
eor    LED, tmp
reti
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
STACK_init
TIM1_CTC_init
LED_init
;-----
; PROGRAM
;-----
ldi    LED, 0x08          ;LED3
ldi    tmp, 0x08

loop:
    rjmp   loop

```

## 5.2 Őnálló feladat

1. Készítsen lefele számlálót a LED-ekre (200-0). Az értékeket 0,5s-onként csökkentse.
2. Valósítson meg egy tetszőleges futófényt. Az egyes állapotok közti idő 0,2s legyen.

## 2 Megszakítások alkalmazása 2.

### 2.1 Közös feladat: Számláló a 7szegmenses kijelzőre, 1 digiten (0-9)

```
;-----TIM1 CTC COMPA-----
;Feladat: Készítsen felfele számlálót a 7szegmenses
; Kijelző 0. digitjére (0-9)
; 1s-os ütemben
;-----

.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
.org 0x100
digit_s:    .BYTE  4             ;A 4jegyű szám
.CSEG
.DEF    tmp    =    r16
.DEF    LED    =    r17
.DEF    digit  =    r18
.DEF    szam   =    r19
.DEF    szamlal =    r20

.ORG 0x0
rjmp start           ; jump to start
.ORG 0x18            ;$0018 TIMER1 COMPA Timer/Counter1 Compare Match A
rjmp TIM1_COMPA
.ORG 0x100
;-----
; MACRO
;-----
.macro TIM1_CTC_init
                ;TCCR1B
                ;ICNC1 ICES1 - WGM13 WGM12 CS12 CS11 CS10
                ; 0   0   0   0   1   1   0   1
ldi      tmp, 0b00001101          ;CTC mód 1024 előosztás
out     TCCR1B, tmp

ldi      tmp, 0x3D               ;ELŐSZÖR OCR1AH!!!!!!
out     OCR1AH, tmp
ldi      tmp, 0x08               ;15625-1      max 65535
out     OCR1AL, tmp
                ;TIMSK
                ;OCIE2 TOIE2 TICIE1 OCIE1A OCIE1B TOIE1 OCIE0 TOIE0
                ; 0   0   0   1   0   0   0   0
ldi      tmp, 0b00010000          ;tmp, 0b00010000
out     TIMSK, tmp
sei
.endmacro

.macro STACK_init
```

```

ldi    tmp, HIGH(RAMEND)
out    SPH, tmp
ldi    tmp, LOW(RAMEND)
out    SPL, tmp
.endmacro
.macro PORT_init
    ldi    tmp, 0xF0
    out    DDRB, tmp
    out    DDRD, tmp
    ldi    tmp, 0xFF          ;7szegmenses kijelző, kimenetre állítás
    out    DDRA, tmp
.endmacro

;-----
; SUBRUTINE
;-----
LED_out:
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
ret

delay:
    ldi    r26, 0xFF
    delay_loop:
        dec    R26           ; ha 0 ' Z flag=1
        brne  delay_loop     ; Z flag-et figyeli, ha Z flag=0 ' ugrik
ret

sub_7seg_digit:                      ;0x80|digit|szam
    eor    tmp, tmp
    ori    tmp, 0x80          ; engedélyezés
    swap   digit
    or     tmp, digit         ; digit 0-3
    swap   digit
    or     tmp, szam          ; szám 0-9
    out    PORTA, tmp
ret

sub_7seg:
    cpi    digit, 4          ;összehasonlítás
    breq  nullaz

kiir:
    ld     szam, Z+          ;memóriából érték betöltése
    call   sub_7seg_digit     ;megjelenítés az aktuális digiten
    inc    digit              ;digit növelése
    rjmp  vege

```

```

nullaz:                                ;digit nullázása
    eor    digit, digit
    ldi    ZL, low(digit_s)           ;visszaállítás a memória címre 0x100
    ldi    ZH, high(digit_s)

vege:
ret
;-----
;INTERRUPT
;-----

TIM1_COMPA:
    ldi    ZL, low(digit_s)           ;visszaállítás a memória címre 0x100
    ldi    ZH, high(digit_s)
    cpi   szamlal, 10                ;ha 9-et túlléptük
    breq  top
    st    Z, szamlal                ;eltároljuk az értéket
    rjmp  end
top:
    eor    szamlal, szamlal         ;törölés - 0
    st    Z, szamlal               ;0 eltárolása
end:
    inc    szamlal                 ;számláló növelése
reti
;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
STACK_init
TIM1_CTC_init
PORT_init
;-----
;PROGRAM
;-----
ldi LED, 0x0A
ldi      ZL, low(digit_s)           ;digit_s kezdőcímére visszaállítás
ldi      ZH, high(digit_s)
eor    digit, digit                ;törölés
ldi szam, 4

tolt:
    ldi    tmp, 0                   ;0000 betöltése
    st    Z+, tmp
    dec   szam
    brne tolta

ldi      ZL, low(digit_s)           ;digit_s kezdőcímére visszaállítás
ldi      ZH, high(digit_s)

eor    szam, szam

```

```

        eor      szamlal, szamlal

loop:
        call    sub_7seg           ;folyamatos szubrutin hívások
        call    delay
        rjmp   loop

```

## 2.2 Közös feladat: 7szegmenses kijelző, számláló (0-99)

```

;-----TIM1 CTC COMPA-----
;Feladat: Készítsen felfele számlálót a 7szegmenses
;          Kijelző 0. digitjére (0-9)
;          1s-os ütemben
;

.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
.org 0x100
digit_s:     .BYTE  4            ;A 4jegyű szám
.CSEG
.DEF tmp = r16
.DEF LED = r17
.DEF digit = r18
.DEF szam = r19
.DEF szamlal = r20
.DEF szamlal1 = r21

.ORG 0x0
rjmp start           ; jump to start
.ORG 0x18           ;$0018 TIMER1 COMPA Timer/Counter1 Compare
Match A
rjmp TIM1_COMPA
.ORG 0x100
;-----
; MACRO
;-----
.macro TIM1_CTC_init
                ;TCCR1B
                ;ICNC1 ICES1 - WGM13 WGM12 CS12 CS11 CS10
                ; 0 0 0 0 1 1 0 1
Iди tmp, 0b00001101           ;CTC mód 1024 előosztás
out TCCR1B, tmp

Iди tmp, 0x3D                 ;ELŐSZÖR OCR1AH!!!!!!
out OCR1AH, tmp
Iди tmp, 0x08                 ;15625-1      max 65535
out OCR1AL, tmp

```

```

;TIMSK
;OCIE2 TOIE2 TICIE1 OCIE1A OCIE1B TOIE1 OCIE0 TOIE0
; 0   0   0   1   0   0   0   0

ldi      tmp, 0b00010000
out     TIMSK, tmp
sei
.endmacro

.macro STACK_init
    ldi      tmp, HIGH(RAMEND)
    out     SPH, tmp
    ldi      tmp, LOW(RAMEND)
    out     SPL, tmp
.endmacro

.macro PORT_init
    ldi      tmp, 0xF0
    out     DDRB, tmp
    out     DDRD, tmp
    ldi      tmp, 0xFF           ;7szegmenses kijelző, kimenetre állítás
    out     DDRA, tmp
.endmacro

;-----
;SUBROUTINE
;-----

LED_out:
    out    PORTD, LED
    swap   LED
    out    PORTB, LED
    swap   LED
ret

delay:
    ldi    r26, 0xFF
delay_loop:
    dec    R26          ; ha 0 ' Z flag=1
    brne delay_loop    ; Z flag-et figyeli, ha Z flag=0 ' ugrik

ret

sub_7seg_digit:           ;0x80|digit|szam
    eor    tmp, tmp
    ori    tmp, 0x80        ; engedélyezés
    swap   digit
    or     tmp, digit       ; digit 0-3
    swap   digit
    or     tmp, szam         ; szám 0-9
    out    PORTA, tmp
ret

```

```

sub_7seg:
    cpi    digit, 4          ;összehasonlítás
    breq   nullaz

kiir:
    ldi    szam, Z+          ;memóriából érték betöltése
    call   sub_7seg_digit    ;megjelenítés az aktuális digiten
    inc    digit              ;digit növelése
    rjmp   vege

nullaz:
    eor    digit, digit      ;digit nullázása
    ldi    ZL, low(digit_s)  ;visszaállítás a memória címre 0x100
    ldi    ZH, high(digit_s)

vege:
ret

;-----
; INTERRUPT
;-----

TIM1_COMPA:
    ldi    ZL, low(digit_s)  ;visszaállítás a memória címre
    ldi    ZH, high(digit_s)
    cpi   szamlal, 10         ;ha 9-et túlléptük
    breq  top
    st    Z, szamlal          ;eltároljuk az értéket
    rjmp  end

top:
    eor   szamlal, szamlal    ;törölés - 0
    st    Z+, szamlal          ;0 eltárolása
    cpi   szamlal1, 10         ;ha 9-et túlléptük
    breq  top1
    inc   szamlal1             ;2. helyiértéken lévő érték növelése
    st    Z, szamlal1          ;2. helyiérték eltárolása
    rjmp  end

top1:
    eor   szamlal1, szamlal1   ;törölés - 0
    st    Z, szamlal1          ;0 eltárolása
end:
    inc   szamlal              ;számláló növelése

reti
;-----
; PROGRAM
;-----

start:
;-----
; INIT
;-----

STACK_init
TIM1_CTC_init
PORT_init

```

```

;-----
; PROGRAM
;-----
ldi      ZL, low(digit_s)          ;digit_s kezdőcímére visszaállítás
ldi      ZH, high(digit_s)
eor     digit, digit             ;törölés
ldi      szam, 4

tolt:
ldi      tmp, 0                  ;0000 betöltése
st       Z+, tmp
dec     szam
brne   tolt

ldi      ZL, low(digit_s)          ;digit_s kezdőcímére visszaállítás
ldi      ZH, high(digit_s)
eor     szam, szam
eor     szamlal, szamlal
eor     szamlal1, szamlal1

loop:
call    sub_7seg                ;folyamatos szubrutin hívások
call    delay
rjmp   loop

```

## 2.3 Közös feladat: Másodperces számláló a 7szegmenses kijelzőre 1

```

;-----TIM1 CTC COMPA-----
;Feladat: Készítsen felfele számlálót a 7szegmenses
;          Kijelző 0. digitjére (0-9)
;          1s-os ütemben
;          A kijelzéshez ne használjon delay-t.
;
;.INCLUDE "m128def.inc" ; Include definitions
Atmega128
.DSEG
.org 0x100
    digit_s: .BYTE 4 ;A 4jegyű szám
.CSEG
.DEF tmp = r16
.DEF LED = r17
.DEF digit = r18
.DEF szam = r19
.DEF szamlal = r20
.DEF count = r21
.ORG 0x0
    rjmp start ; jump to start
.ORG 0x18 ;$0018 TIMER1 COMPA Timer/Counter1 Compare Match A
    rjmp TIM1_COMPA
.ORG 0x100
;-----
; MACRO
;-----
.macro TIM1_CTC_init
    ldi tmp, 0b00001011 ;TCCR1B
    ;ICNC1 ICES1 - WGM13 WGM12 CS12 CS11 CS10
    ; 0 0 0 0 1 0 1 1
    out TCCR1B, tmp ;CTC mód 64 előosztás
    ldi tmp, 0x03 ;ELŐSZÖR OCR1AH!!!!!!
    out OCR1AH, tmp
    ldi tmp, 0xE7 ;1000-1 max 65535 0,004s 4ms
    out OCR1AL, tmp
    ;TIMSK
    ;OCIE2 TOIE2 TICIE1 OCIE1A OCIE1B TOIE1 OCIE0 TOIE0
    ; 0 0 0 1 0 0 0 0
    ldi tmp, 0b00010000
    out TIMSK, tmp
    sei
.endmacro

.macro STACK_init
    ldi tmp, HIGH(RAMEND)
    out SPH, tmp
    ldi tmp, LOW(RAMEND)
    out SPL, tmp
.endmacro

```

```

.macro PORT_init
    ldi    tmp, 0xF0
    out   DDRB, tmp
    out   DDRD, tmp
    ldi    tmp, 0xFF
    out   DDRA ,tmp
;7szegmenses kijelző, kimenetre állítás
.endmacro

;-----
;SUBROUTINE
;-----
LED_out:
    out   PORTD, LED
    swap  LED
    out   PORTB, LED
    swap  LED
ret

sub_7seg_digit:           ;0x80|digit|szam
    eor   tmp, tmp
    ori   tmp, 0x80
    swap  digit
    or    tmp, digit          ; digit 0-3
    swap  digit
    or    tmp, szam           ; szám 0-9
    out   PORTA, tmp
ret

sub_7seg:                 ;összehasonlítás
    cpi   digit, 4
    breq nullaz
kiir:
    ld    szam, Z+
    call sub_7seg_digit      ;memóriából érték betöltése
    inc   digit              ;megjelenítés az aktuális digiten
    rjmp  vege               ;digit növelése

nullaz:                   ;digit nullázása
    eor   digit, digit
    ldi   ZL, low(digit_s)   ;visszaállítás a memória címre 0x100
    ldi   ZH, high(digit_s)

vege:
ret

```

```

;-----
; INTERRUPT
;-----
TIM1_COMPA:
    call    sub_7seg          ; minden megszakításkor kiteszünk egy digitet
    inc     count
    cpi    count, 250         ; minden 250. megszakításkor növeljük a számlálót
    brne   end1

    eor    count, count
    ldi    ZL, low(digit_s)   ; visszaállítás a memória címre 0x100
    ldi    ZH, high(digit_s)
    cpi   szamlal, 10         ; ha 9-et túlléptük
    breq  top
    st     Z, szamlal         ; eltároljuk az értéket
    rjmp  end
    top:
        eor    szamlal, szamlal ; törlés - 0
        st     Z, szamlal       ; 0 eltárolása
    end:
        inc     szamlal         ; számláló növelése
    end1:
reti
;-----
; PROGRAM
;-----
start:
;-----
; INIT
;-----
STACK_init
TIM1_CTC_init
PORT_init
;-----
; PROGRAM
;-----
ldi LED, 0x0A

ldi     ZL, low(digit_s)   ; digit_s kezdőcímére visszaállítás
ldi     ZH, high(digit_s)
eor   digit, digit          ; törlés
eor   count, count

ldi szam, 4

tolt:
    ldi    tmp, 0             ; 0000 betöltése
    st     Z+, tmp
    dec   szam
    brne  tolt

```

ldi ZL, low(digit\_s) ;digit\_s kezdőcímére visszaállítás  
ldi ZH, high(digit\_s)

eor szam, szam  
eor szamlal, szamlal

loop:

rjmp loop

## 2.4 Közös feladat: 7szegmenses kijelző, számláló (0-99), delay nélkül

```

;-----TIM1 CTC COMPA-----
;Feladat: Készítsen felfele számlálót a 7szegmenses
;          Kijelző 0. digitjére (0-9)
;          1s-os ütemben
;

.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
.org 0x100
digit_s:    .BYTE 4             ;A 4jegyű szám
.CSEG
.DEF tmp = r16
.DEF LED = r17
.DEF digit = r18
.DEF szam = r19
.DEF szamlal = r20
.DEF szamlal1 = r21
.def count = r22

.ORG 0x0
rjmp start           ; jump to start
.ORG 0x18           ;$0018 TIMER1 COMPA Timer/Counter1 Compare Match A
rjmp TIM1_COMPA
.ORG 0x100
;-----
; MACRO
;-----
.macro TIM1_CTC_init
;TCCR1B
;ICNC1 ICES1 - WGM13 WGM12 CS12 CS11 CS10
; 0 0 0 0 1 1 0 1
lди tmp, 0b00001011      ;CTC mód 64 előosztás
out TCCR1B, tmp

lди tmp, 0x03           ;ELŐSZÖR OCR1AH!!!!!!
out OCR1AH, tmp
lди tmp, 0xE7           ;1000-1 max 65535
out OCR1AL, tmp
;TIMSK
;OCIE2 TOIE2 TICIE1 OCIE1A OCIE1B TOIE1 OCIE0 TOIE0
; 0 0 0 1 0 0 0 0
lди tmp, 0b00010000
out TIMSK, tmp
sei
.endmacro

```

```

.macro STACK_init
    ldi    tmp, HIGH(RAMEND)
    out    SPH, tmp
    ldi    tmp, LOW(RAMEND)
    out    SPL, tmp
.endmacro

.macro PORT_init
    ldi    tmp, 0xF0
    out    DDRB, tmp
    out    DDRD, tmp
    ldi    tmp, 0xFF                ;7szegmenses kijelző, kimenetre állítás
    out    DDRA, tmp

.endmacro
;-----
;; SUBROUTINE
;-----

LED_out:
    out   PORTD, LED
    swap LED
    out   PORTB, LED
    swap LED
    ret

sub_7seg_digit:                      ;0x80|digit|szam
    eor  tmp, tmp
    ori  tmp, 0x80                  ; engedélyezés
    swap digit
    or   tmp, digit                ; digit 0-3
    swap digit
    or   tmp, szam                 ; szám 0-9
    out  PORTA, tmp

ret

sub_7seg:                            ;összehasonlítás
    cpi  digit, 4
    breq nullaz

kiir:
    ld   szam, Z+                  ;memóriából érték betöltése
    call sub_7seg_digit            ;megjelenítés az aktuális digiten
    inc  digit                     ;digit növelése
    rjmp vege

nullaz:                             ;digit nullázása
    eor  digit, digit
    ldi  ZL, low(digit_s)          ;visszaállítás a memória címre 0x100
    ldi  ZH, high(digit_s)

vege:
ret

```

```

;-----
;INTERRUPT
;-----
TIM1_COMPA:
    call    sub_7seg          ;mindig meghívódik
    inc     count
    cpi    count, 250         ; minden 250. megszakításkor növeljük a számlálót
    brne   vege1

    eor    count, count
    ldi    ZL, low(digit_s)   ;visszaállítás a memória címre
    ldi    ZH, high(digit_s)
    cpi   szamlal, 10         ;ha 9-et túlléptük
    breq  top
    st     Z, szamlal         ;eltároljuk az értéket
    rjmp  end

    top:
        eor    szamlal, szamlal ;törles - 0
        st     Z+, szamlal      ;0 eltárolása
        cpi   szamlal1, 10       ;ha 9-et túlléptük
        breq  top1
        inc   szamlal1          ;2. helyíréken lévő érték növelése
        st     Z, szamlal1       ;2. helyírétek eltárolása
        rjmp  end

    top1:
        eor    szamlal1, szamlal1 ;törles - 0
        st     Z, szamlal1       ;0 eltárolása
    end:
        inc   szamlal           ;számláló növelése

    vege1:
    reti
;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
STACK_init
TIM1_CTC_init
PORT_init
;-----
;PROGRAM
;-----
    ldi    ZL, low(digit_s)   ;digit_s kezdőcímére visszaállítás
    ldi    ZH, high(digit_s)
    eor   digit, digit         ;törles
    ldi    szam, 4
    eor   count, count

```

```

tolt:
ldi    tmp, 0           ;0000 betöltése
st     Z+, tmp
dec   szam
brne  tolt

ldi    ZL, low(digit_s) ;digit_s kezdőcímére visszaállítás
ldi    ZH, high(digit_s)
eor   szam, szam
eor   szamlal, szamlal
eor   szamlal1, szamlal1
loop:
rjmp  loop

```

## 2.5 Önálló feladatok

1. Készítsen másodperc (0-59) kijelzőt a 7szegmenses kijelző 3-2 digitjére. A feladathoz megszakítást használjon.
2. Készítsen lefelé számlálót, amely 99-0-ig számol 0,5s-os ütemben.

### 3 PWM

#### 3.1 Közös feladat: Timer0 PWM

```
;-----TIM0 PWM-----
;Feladat:      Inicializálja a TIM0-hoz tartozó Phase Correct PWM Mode-ot.
;              Állítson be különböző kitöltési tényezőket és figyelje a működést
;-----  
  
.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    tmp          =      r16
.DEF    LED          =      r17
.DEF    PWM_d        =      r18
.ORG 0x0
    rjmp start          ; jump to start  
  
.ORG 0x100
;-----  
;  
; MACRO
;-----  
.macro TIM0_PWM_init          ;TIM0 Phase Correct PWM Mode init
    ldi    tmp, 0b01100100      ;64-es előosztás beállítása
                                ;Phase Correct PWM Mode kiválasztás,
                                ;Clear OC0 on Compare Match
                                ;when up-counting.
                                ;Set OC0 on Compare Match
                                ;when downcounting.
    out   TCCR0, tmp          ;Timer/Counter Control Register - TCCR0
                                ; 7      6      5      4      3      2      1      0
                                ;FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00
                                ; 0      1      1      0      0      1      0      0
.endmacro  
  
.macro STACK_init
    ldi    tmp, HIGH(RAMEND)
    out   SPH, tmp
    ldi    tmp, LOW(RAMEND)
    out   SPL, tmp
.endmacro  
  
.macro LED_init
    ldi    tmp, 0xF0
    out   DDRB, tmp
    out   DDRD, tmp
.endmacro
;-----  
;  
; SUBROUTINE
;-----
```

```

PWM_duty:
    mov    tmp, PWM_d           ;kitöltési tényező állítása
    out    OCRO, tmp            ;0-255
ret
;-----
;INTERRUPT
;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
STACK_init
LED_init
TIM0_PWM_init
;-----
;PROGRAM
;-----
ldi    PWM_d, 4             ;kitöltési tényező: 1,5625
                            ;Próbája ki a következő értékekkel:
                            ;0, 255, 1, 5, 10, 50, 200

call    PWM_duty
loop:
rjmp   loop

```

### 3.2 Közös feladat: TIM1 PWM

```

;-----TIM1 PWM-----
;Feladat:      Inicializálja a TIM1 Fast PWM módját.
;              Előosztásnak 64-et állítson.
;              komparálási értéknek 249-et.
;              OCR1A, OCR1B, OCR1C értékeknek 1, 10, 100-at állítson
;
.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
.CSEG
.DEF    tmp    =      r16
.DEF    LED    =      r17

.ORG 0x0
    rjmp start             ; jump to start
.ORG 0x100
;-----
; MACRO
;-----
.macro TIM1_PWM_init
    ;TCCR1A
    ;COM1A1 COM1A0 COM1B1 COM1B0 COM1C1 COM1C0 WGM11 WGM10
    ;  1      0      1      0      1      0      1      0
    ldi    tmp,0b10101010
    out    TCCR1A, tmp
                    ;TCCR1B
                    ;ICNC1 ICES1 - WGM13 WGM12 CS12 CS11 CS10
    ;  0      0      0      1      1      0      1      1
    ldi    tmp, 0b000011011       ;64 előosztás, Fast PWM
    out    TCCR1B, tmp

    ldi    tmp, 0               ;Komparálási érték, 249
    out    ICR1H, tmp
    ldi    tmp, 0xF9
    out    ICR1L, tmp

    ldi    tmp, 0
    out    OCR1AH, tmp
    ldi    tmp, 0
    out    OCR1BH, tmp
    ldi    tmp, 0
    sts    OCR1CH, tmp
                    ;Kitöltés állítása
    ldi    tmp, 1
    out    OCR1AL, tmp
    ldi    tmp, 10
    out   OCR1BL, tmp
    ldi    tmp, 100
    sts   OCR1CL, tmp

.endmacro

```

```

.macro STACK_init
    ldi    tmp, HIGH(RAMEND)
    out   SPH, tmp
    ldi    tmp, LOW(RAMEND)
    out   SPL, tmp
.endmacro

.macro LED_init
    ldi    tmp, 0xF0
    out   DDRB, tmp
    out   DDRD, tmp
.endmacro

;-----
;SUBROUTINE
;-----
LED_out:
    out   PORTD, LED
    swap  LED
    out   PORTB, LED
    swap  LED
    ret

;-----
;INTERRUPT
;-----
;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
STACK_init
TIM1_PWM_init
LED_init
;-----
;PROGRAM
;-----
loop:
    rjmp  loop

```

### 3.3 Önálló feladatok

1. Pörbálja ki a mintakódokat különböző kitöltési tényezőkkel. Kapcsoljon be egy szomszédos LED-et, hogy jól láthatass a fényerőkülönbséget.

## 4 RGB LED

### 4.1 Közös feladat: RGB piros LED villogtatása

```
;-----TIM1 CTC COMPA-----
;Feladat:      Villogtassa a RGB (piros) LED-et TIM1 COMPA megszakítással
;           1s-os ütemben
;-----


.INCLUDE "m128def.inc"          ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    tmp    =      r16
.DEF    LED    =      r17

.ORG 0x0
rjmp start           ; jump to start
.ORG 0x18
        rjmp TIM1_COMPA
.ORG 0x100
;-----;
; MACRO
;-----;
.macro TIM1_CTC_init
        ;TCCR1B
        ;ICNC1 ICES1 - WGM13 WGM12 CS12 CS11 CS10
        ; 0   0   0   0   1   1   0   1
        ldi    tmp, 0b00001101          ;CTC mód 1024 előosztás
        out    TCCR1B, tmp

        ldi    tmp, 0x3D              ;ELŐSZÖR OCR1AH!!!!!!
        out    OCR1AH, tmp
        ldi    tmp, 0x08              ;15625-1      max 65535
        out    OCR1AL, tmp

        ;TIMSK
        ;OCIE2 TOIE2 TICIE1 OCIE1A OCIE1B TOIE1 OCIE0 TOIE0
        ; 0   0   0   1   0   0   0   0
        ldi    tmp, 0b00010000          ;TIMSK, tmp
        sei

.endmacro
```

```

.macro STACK_init
    ldi      tmp, HIGH(RAMEND)
    out     SPH, tmp
    ldi      tmp, LOW(RAMEND)
    out     SPL, tmp
.endmacro

.macro RGB_LED_init
    ldi      tmp, 0x80          ;RGB RED LED
    out     DDRC, tmp
.endmacro

;-----
; SUBRUTINE
;-----

;-----
; INTERRUPT
;-----

TIM1_COMPA:
    out     PORTC, LED        ;LED villogtatása
    eor     LED, tmp
    reti

;-----
; PROGRAM
;-----

start:
;-----
; INIT
;-----

STACK_init
TIM1_CTC_init
RGB_LED_init
;-----
; PROGRAM
;-----

ldi      LED, 0x80          ;RGB RED LED
ldi      tmp, 0x80
loop:

        rjmp   loop

```

## 4.2 Extra közös feladat: RGB zöld LED PWM

```

;-----TIM3 PWM RGB G-----
;Feladat:      Növelje folyamatosan az RGB zöld LED kitöltési tényezőjét
;              A kitöltés növeléséhez (TIM0) és a PWM-hez is timert használjon
;              (TIM3)
;-----  

.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF    tmp     =      r16

.ORG 0x0
    rjmp start             ; jump to start
.ORG 0x20                 ;$0020 TIMERO OVF Timer/Counter0 Overflow
    rjmp TIMER0_OVF
.ORG 0x100
;  

;  

; MACRO
;  

;.macro port_init
    ldi    tmp, 0x08
    out   DDRE, tmp
.endmacro

;.macro stack_init
    ldi    tmp, HIGH(RAMEND)
    out   SPH, tmp
    ldi    tmp, LOW(RAMEND)
    out   SPL, tmp
.endmacro

;.macro timer0_OVF_init
    ;TCCR0: FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00
    ;        7   6   5   4   3   2   1   0
    ;        0   0   0   0   0   1   1   1
    ldi    tmp, 0b_0000_0111          ;1024 előosztás
    out   TCCR0, tmp

    ;TIMSK: OCIE2 TOIE2 TICIE1 OCIE1A OCIE1B TOIE1 OCIE0 TOIE0
    ;       7   6   5   4   3   2   1   0
    ;       0   0   0   0   0   0   0   1
    ldi    tmp, 0b_0000_0001          ;OVF vect EN
    out   TIMSK, tmp
    sei
.endmacro

```

```

.macro timer3_PWM_init
;TCCR3A:    COM3A1 COM3A0 COM3B1 COM3B0 COM3C1 COM3C0 WGM31 WGM30
;          7      6      5      4      3      2      1      0
;          1      0      0      0      0      0      1      0

ldi    tmp, 0b_1000_0010
; A3 csatornán ; PWM, Phase Correct
sts   TCCR3A, tmp
;TCCR3B: ICNC3 ICES3 - WGM33 WGM12 CS32 CS31 CS30
;          7      6      5      4      3      2      1      0
;          0      0      0      1      0      1      0      0

ldi    tmp, 0b_0001_0100
;top: ICR3
;/256
sts   TCCR3B,tmp
ldi    tmp, 0xFF
sts   ICR3L, tmp
;ICR3=255
.endmacro
;-----
;SUBROUTINE
;-----
;-----
;INTERRUPT
;-----
;-----  

TIMERO_OVF:
inc   tmp
sts   OCR3AL, tmp
;kitöltés növelése
reti
;-----  

;PROGRAM
;-----  

start:
;-----  

;INIT
;-----  

port_init
stack_init
timer0_OVF_init
timer3_PWM_init
eor   tmp, tmp
;-----  

;PROGRAM
;-----  

loop:

        rjmp  loop

```

## 5 Gyakorló feladatok

1. Készítsen LED fényerő szabályozót. Tetszőleges PWM alkalmazásával, nyomógomb lenyomására növelje (G0) és csökkentse (G1) a LED fényerejét.
2. Készítsen szubrutint, amely folyamatosan növeli, majd csökkenti a LED0 fényerejét.
3. Készítsen szubrutint, amely folyamatosan növeli, majd csökkenti a LED2 fényerejét.
4. Készítsen órát, jelezze ki a 7szegmenses kijelzőn a másodperceket és a perceket. Villogtassa a 7szegmenses kijelzőn a kettőspontot 0,5s-onként.
5. Készítsen stoppert. G0 nyomógomb lenyomására indítsa el a 0,1s-os számlálást (0-9999) a 7szegmenses kijelzőn, G1-re állítsa meg, G3-mmal lehessen nullázni.

## 6. labor

### Labor tematikája:

1. Portkezelés alapjai: Mátrixbillentyűzet (117-120. oldal)
2. Mátrix billentyűzet: összeadás
3. Mátrix billentyűzet: kivonás
4. Mátrix billentyűzet: szorzás

bit	7	6	5	4	3	2	1	0	
PORT									
A	ENABLE	SEL2	SEL1	SEL0	DATA3	DATA2	DATA1	DATA0	7 SEGMENT DISPLAY
I/O	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT	
B	Led3	Led2	Led1	Led0					LED/Io 4bit
I/O	OUT	OUT	OUT	OUT					
C	RED	KBD4row	KBD3row	KBD2row	KBD1row	KBD_right	KBD_centr	KBD_left	Keyboard
I/O	OUT	OUT	OUT	OUT	OUT	IN	IN	IN	
D	Led7	Led6	Led5	Led4					LED/hi 4bit
I/O	OUT	OUT	OUT	OUT					
E	LCD_DATA7	LCD_DATA6	LCD_DATA5	LCD_DATA4	GREEN	BLUE			LCD data
I/O	OUT	OUT	OUT	OUT	OUT	OUT			
F					LCD_E	LCD_R/W	LCD_RS	LM35	LCD Control
I/O					OUT	OUT	OUT	IN	Analog
G	NC	NC	NC	K4	K3	K2	K1	K0	Pushbutton
I/O	X	X	X	IN	IN	IN	IN	IN	
bit	7	6	5	4	3	2	1	0	

# 1 Portkezelés alapjai: Mátrixbillentyűzet

## 1.1 Közös feladat: Lenyomott szám megjelenítése

```
;----- Mátrix-----
;Feladat: Olvassa be a mátrix billentyűzet gombjait (0-9).
; A beolvasott billentyűzet számát jelenítse meg a 7szegmenses
; kijelző 0. digitjén.
;----- .INCLUDE "m128def.inc" ; Include definitions Atmega128
.DSEG
;
.CSEG
.DEF keyb_val = r5
.DEF keyb_but = r6
.DEF keyb_num = r7
.DEF tmp = r16
.DEF tmp2 = r17
.DEF tmp3 = r18
.DEF tmp4 = r19
.DEF tmp5 = r21
.DEF keyb_row = r22
.DEF szam = r23
.DEF digit = r24
.ORG 0x0
    rjmp start ; jump to start
.ORG 0x100
;----- ; MACRO
;----- .macro PORT_init
        in    tmp, DDRA ; 7 szegmenses kijelző
        ori   tmp, 0xff
        out   DDRA, tmp
        in    tmp, DDRC ; Mátrix billentyűzet
        ori   tmp, 0xF8
        out   DDRC, tmp
.endmacro

.macro STACK_init
    ldi   tmp, HIGH(RAMEND)
    out  SPH, tmp
    ldi   tmp, LOW(RAMEND)
    out  SPL, tmp
.endmacro

delay_ms:
    eor   tmp4, tmp4
    eor   tmp5, tmp5
delay_1:
    dec   tmp4
```

```

        brne    delay_1
        dec     tmp5
        brne    delay_1
ret
;-----
; SUBROUTINE
;-----
sub_7seg_digit:
;0x80|digit|szam
        eor     tmp, tmp
        ori     tmp, 0x80
        swap   digit
        or      tmp, digit
        swap   digit
        or      tmp, szam
        out    PORTA, tmp
ret

sub_matrix_olvas_kiir:
keyb_init:
ldi    keyb_row, 0x08
;1. sor
keyb_init_1:
ldi    ZH, HIGH(keyb_nums<<1)
;számok beolvasása
ldi    ZL, LOW(keyb_nums<<1)
cpi   keyb_row, 0x80
;összehasonlítás
breq  keyb_init
;címzés kezdése ismét az 1. sornál
out   PORTC, keyb_row
;Sor címzése
in    keyb_but, PINC
;PORT C olvasása
lsl   keyb_row
;Következő sor
ldi   tmp2, 12
;12 alapérték, ha nincs egyezés
back_keyb:
lpm   keyb_val, Z+
;beolvasás, majd egyet lépteti az indexet
cp    keyb_val, keyb_but
;breq find_keyb
;Egyezés keresése a beolvasott értékkel
inc   ZL
;Egyezés esetén
dec   tmp2
;Cím növelése
brne back_keyb
;jmp   keyb_init_1
;Végig nézzük az összes értéket
find_keyb:
lpm   szam, Z
;Lenyomott gomb számának kitétele
eor   digit, digit
call  sub_7seg_digit
;A 7szegmenses kijelzőre, 0. digit
call  delay_ms

ret
;-----
; INTERRUPT
;-----
;-----
; PROGRAM
;-----
start:

```

```
;-----  
; INIT  
;-----  
PORT_init  
STACK_init  
  
;-----  
; PROGRAM  
;-----  
  
loop:  
    call    sub_matrix_olvas_kiir  
    rjmp   loop  
  
keyb_nums: .DB 69,0,14,1,13,2,11,3,22,4,21,5,19,6,38,7,37,8,35,9,67,10,70,11  
; DB-Define constant byte(s) in program memory or E2PROM memory
```

## 1.2 Közös feladat: Megszakítással

```

;----- Mátrix -----
;Feladat: Olvassa be a mátrix billentyűzet gombjait (0-9).
;          A beolvasott bill. számát
;          jelenítse meg a 7szegmenses kijelző 0. digitjén.
;          Kezelés megszakítással.
;
;.INCLUDE "m128def.inc"           ; Include definitions Atmega128
.DSEG
.CSEG
.DEF    keyb_val    =    r5
.DEF    keyb_but    =    r6
.DEF    keyb_num    =    r7
.DEF    tmp         =    r16
.DEF    tmp2        =    r17
.DEF    tmp3        =    r18
.DEF    tmp4        =    r19
.DEF    tmp5        =    r21
.DEF    keyb_row   =    r22
.DEF    szam        =    r23
.DEF    digit       =    r24
.ORG 0x0
    rjmp start           ; jump to start
.ORG 0x20
flow
    rjmp TIM0_OVF_IT
.ORG 0x100
;----- ; MACRO ;-----
;macro PORT_init
    in    tmp, DDRA           ; 7 szegmenses kijelző
    ori   tmp, 0xff
    out   DDRA, tmp
    in    tmp, DDRC           ; Mátrix billentyűzet
    ori   tmp, 0xF8
    out   DDRC, tmp
.endmacro

;macro STACK_init
    ldi   tmp, HIGH(RAMEND)
    out  SPH, tmp
    ldi   tmp, LOW(RAMEND)
    out  SPL, tmp
.endmacro

;macro TIM0_OVF_init      ;TIM0 OVF init
    ldi   tmp, 0b00000111     ;1024-es előosztás beállítása
    out  TCCR0, tmp           ;Timer/Counter Control Register - TCCR0
                                ; 7   6   5   4   3   2   1   0
                                ;FOC0 WGM00 COM01 COM00 WGM01 CS02 CS01 CS00

```

```

ldi    tmp, 0x01           ;Timer/Counter0 Overflow Interrupt Enable

out   TIMSK, tmp    ; 7      6      5      4      3      2      1      0
sei
.endmacro
;-----;
;SUBROUTINE
;-----;
sub_7seg_digit:          ;0x80|digit|szam
    eor   tmp, tmp
    ori   tmp, 0x80
    swap  digit
    or    tmp, digit
    swap  digit
    or    tmp, szam
    out   PORTA, tmp
ret

sub_matrix_olvas_kiir:
keyb_init:
    ldi   keyb_row, 0x08          ;1. sor
keyb_init_1:
    ldi   ZH, HIGH(keyb_nums<<1) ;számok beolvasása
    ldi   ZL, LOW(keyb_nums<<1)
    cpi  keyb_row, 0x80
    breq keyb_init
    out  PORTC, keyb_row
    in   keyb_but, PINC
    lsl  keyb_row
    ldi  tmp2, 12
back_keyb:
    lpm  keyb_val, Z+          ;beolvasás, majd egyet lépteti az indexet
    cp   keyb_val, keyb_but
    breq find_keyb
    inc  ZL
    dec  tmp2
    brne back_keyb
    jmp  keyb_init_1
find_keyb:
    lpm  szam, Z              ;Lenyomott gomb számának kitétele
    eor  digit, digit
    call sub_7seg_digit
ret
;-----;
;INTERRUPT
;-----;
TIM0_OVF_IT:
    call  sub_matrix_olvas_kiir
reti
;-----;
;PROGRAM

```

```
;-----  
start:  
;-----  
; INIT  
;-----  
PORT_init  
STACK_init  
TIM0_OVF_init  
;-----  
; PROGRAM  
;-----  
  
loop:  
  
    rjmp    loop  
  
keyb_nums: .DB 69,0,14,1,13,2,11,3,22,4,21,5,19,6,38,7,37,8,35,9,67,10,70,11  
; DB-Define constant byte(s) in program memory or E2PROM memory
```

## 2 Mátrix billentyűzet összeadás

### 2.1 Közös feladat: Összeadás

```
;----- Mátrix-----
;Feladat: Olvassa be a mátrix billentyűzet gombjait (0-9).
; Valósítson meg összeadást, az eredményt jelenítse meg
; a 7szegmenses kijelzőn.
; G0    1. operandus      0. digit
; G1    2. operandus      1. digit
; G2    Összeadás        2. digit
;

.INCLUDE "m128def.inc" ; Include definitions Atmega128
.DSEG
szamok: .BYTE 2
.CSEG
.DEF keyb_val = r5
.DEF keyb_but = r6
.DEF keyb_num = r7
.DEF tmp = r16
.DEF tmp2 = r17
.DEF tmp3 = r18
.DEF tmp4 = r19
.DEF tmp5 = r21
.DEF keyb_row = r22
.DEF szam = r23
.DEF digit = r24
.DEF btn = r8
.DEF szam1 = r9
.DEF szam2 = r10

.ORG 0x0
rjmp start ; jump to start

.ORG 0x100
;-----;
; MACRO
;-----;
.macro PORT_init
in    tmp, DDRA ; 7 szegmenses kijelző
ori   tmp, 0xff
out   DDRA, tmp
in    tmp, DDRC ; Mátrix billentyűzet
ori   tmp, 0xF8
out   DDRC, tmp
eor   tmp, tmp
sts   DDRG, tmp ; Gombok
.endmacro

.macro STACK_init
ldi   tmp, HIGH(RAMEND)
```

```

        out    SPH, tmp
        ldi    tmp, LOW(RAMEND)
        out    SPL, tmp
.endmacro

delay_ms:
        eor    tmp4, tmp4
        eor    tmp5, tmp5
delay_1:
        dec    tmp4
        brne  delay_1
        dec    tmp5
        brne  delay_1
        ret

;-----
; SUBROUTINE
;-----
sub_7seg_digit:                                ;0x80|digit|szam
        eor    tmp, tmp
        ori    tmp, 0x80          ; engedélyezés
        swap   digit
        or     tmp, digit         ; digit 0-3
        swap   digit
        or     tmp, szam          ; szám 0-9
        out    PORTA, tmp
        ret

sub_matrix_olvas_kiir:
keyb_init:
        ldi    keyb_row, 0x08      ;1. sor
keyb_init_1:
        ldi    ZH, HIGH(keyb_nums<<1) ;számok beolvasása
        ldi    ZL, LOW(keyb_nums<<1)
        cpi   keyb_row, 0x80        ;összehasonlítás
        breq  keyb_init            ;címzés kezdése ismét az 1. sornál
        out   PORTC, keyb_row      ;Sor címzése
        in    keyb_but, PINC       ;PORT C olvasása
        lsl   keyb_row             ;Következő sor
        ldi   tmp2, 12              ;12 alapérték, ha nincs egyezés
back_keyb:
        lpm   keyb_val, Z+          ;beolvasás, majd egyet lépteti az indexet
        cp    keyb_val, keyb_but    ;Egyezés keresése a beolvasott értékkel
        breq  find_keyb            ;Egyezés esetén
        inc   ZL                   ;Cím növelése
        dec   tmp2                  ;Végig nézzük az összes értéket
        brne back_keyb
        jmp   keyb_init_1

find_keyb:
        lpm   szam, Z               ;Lenyomott gomb számának kitétele

```

```

;a 7szegmenses kijelzőre
call    sub_7seg_digit
call    delay_ms

ret

;-----
;INTERRUPT
;-----
;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
PORT_init
STACK_init
;-----
;PROGRAM
;-----
loop:
    lds    btn, PING           ;Nyomógombok olvasása
    ldi    szam, 0x1F          ;Maszkolás
    and    btn, szam
    sbrc   btn, 0              ;Vizsgálatok, hogy melyiket nyomtuk
    rjmp   beolvas1
    sbrc   btn, 1
    rjmp   beolvas2
    sbrc   btn, 2
    rjmp   osszead
    rjmp   loop

    beolvas1:                 ;0. gomb nyomására 1. operandus beolvasása
    ldi    digit, 0            ;0. digitre kitétel
    call   sub_matrix_olvas_kiir
    ldi    XL, low(szamok)     ;cél alsó címe
    ldi    XH, high(szamok)    ;cél felső
    st    X, szam              ;eltároljuk az 1. operandust
    rjmp   loop

    beolvas2:                 ;1. gomb nyomására 2. operandus beolvasása
    ldi    digit, 1            ;1. digitre kitétel
    call   sub_matrix_olvas_kiir
    ldi    XL, low(szamok)     ;cél alsó címe
    ldi    XH, high(szamok)    ;cél felső
    adiw   X, 1
    st    X, szam              ;eltároljuk a 2. operandust
    rjmp   loop

    osszead:                  ;összeadás
    ldi    digit, 2

```

```
ldi    XL, low(szamok)
ldi    XH, high(szamok)
ld    szam1, X+
ld    szam2, X
add   szam1, szam2
mov   szam, szam1
call  sub_7seg_digit ;Összeg kitétele a 2. digitre

rjmp  loop
```

```
keyb_nums: .DB 69,0,14,1,13,2,11,3,22,4,21,5,19,6,38,7,37,8,35,9,67,10,70,11
; DB-Define constant byte(s) in program memory or E2PROM memory
```

### 3 Mátrix billentyűzet kivonás

#### 3.1 Közös feladat: Kivonás

```
;----- Mátrix-----
;Feladat: Olvassa be a mátrix billentyűzet gombjait (0-9).
; Valósítson meg összeadást, az eredményt jelenítse meg
; a 7szegmenses kijelzőn.
; G0    1. operandus      0. digit
; G1    2. operandus      1. digit
; G2    Összeadás        2. digit
; G3    Kivonás          3. digit (nagyobb szám - kisebb szám)
;----- .INCLUDE "m128def.inc" ; Include definitions Atmega128
.DSEG
szamok: .BYTE 2
.CSEG
.DEF keyb_val = r5
.DEF keyb_but = r6
.DEF keyb_num = r7
.DEF tmp = r16
.DEF tmp2 = r17
.DEF tmp3 = r18
.DEF tmp4 = r19
.DEF tmp5 = r21
.DEF keyb_row = r22
.DEF szam = r23
.DEF digit = r24
.DEF btn = r8
.DEF szam1 = r9
.DEF szam2 = r10
.ORG 0x0
    rjmp start ; jump to start
.ORG 0x100
;----- ; MACRO ;----- .macro PORT_init
    in    tmp, DDRA ; 7 szegmenses kijelző
    ori   tmp, 0xff
    out   DDRA, tmp
    in    tmp, DDRC ; Mátrix billentyűzet
    ori   tmp, 0xF8
    out   DDRC, tmp
    eor   tmp, tmp
    sts   DDRG, tmp ; Gombok
.endmacro

.macro STACK_init
    ldi   tmp, HIGH(RAMEND)
    out   SPH, tmp
```

```

ldi    tmp, LOW(RAMEND)
out    SPL, tmp
.endmacro

delay_ms:
    eor    tmp4, tmp4
    eor    tmp5, tmp5
delay_1:
    dec    tmp4
    brne  delay_1
    dec    tmp5
    brne  delay_1
    ret

;-----
; SUBROUTINE
;-----
sub_7seg_digit:
    eor    tmp, tmp
    ori    tmp, 0x80
    swap   digit
    or     tmp, digit
    swap   digit
    or     tmp, szam
    out   PORTA, tmp
    ret

;0x80|digit|szam

sub_matrix_olvas_kiir:
keyb_init:
    ldi    keyb_row, 0x08
    ;1. sor
keyb_init_1:
    ldi    ZH, HIGH(keyb_nums<<1)
    ldi    ZL, LOW(keyb_nums<<1)
    ;számok beolvasása
    cpi   keyb_row, 0x80
    ;összehasonlítás
    breq  keyb_init
    out   PORTC, keyb_row
    ;címzés kezdése ismét az 1. sornál
    in    keyb_but, PINC
    ;Sor címzése
    lsl   keyb_row
    ;Következő sor
    ldi    tmp2, 12
    ;12 alapérték, ha nincs egyezés
back_keyb:
    lpm   keyb_val, Z+
    ;beolvasás, majd egyet lépteti az indexet
    cp    keyb_val, keyb_but
    ;Egyezés keresése a beolvasott értékkel
    breq  find_keyb
    ;Egyezés esetén
    inc   ZL
    ;Cím növelése
    dec   tmp2
    ;Végig nézzük az összes értéket
    brne back_keyb
    jmp   keyb_init_1
find_keyb:
    lpm   szam, Z
    ;Lenyomott gomb számának kitétele
    ;a 7szegmenses kijelzőre
    call   sub_7seg_digit

```

```

    call    delay_ms
ret
;-----
;INTERRUPT
;-----

;-----
;PROGRAM
;-----
start:
;-----
;INIT
;-----
PORT_init
STACK_init
;-----
;PROGRAM
;-----
loop:
    lds    btn, PING           ;Nyomógombok olvasása
    ldi    szam, 0x1F          ;Maszkolás
    and    btn, szam
    sbrc   btn, 0              ;Vizsgálatok, hogy melyiket nyomtuk
    rjmp   beolvas1
    sbrc   btn, 1
    rjmp   beolvas2
    sbrc   btn, 2
    rjmp   osszead
    sbrc   btn, 3
    rjmp   kivon

    rjmp   loop

beolvas1:                      ;0. gomb nyomására 1. operandus beolvasása
    ldi    digit, 0            ;0. digitre kitétel
    call   sub_matrix_olvas_kiir
    ldi    XL, low(szamok)     ;cél alsó címe
    ldi    XH, high(szamok)    ;cél felső
    st    X, szam              ;eltároljuk az 1. operandust
    rjmp   loop

beolvas2:                      ;1. gomb nyomására 2. operandus beolvasása
    ldi    digit, 1            ;1. digitre kitétel
    call   sub_matrix_olvas_kiir
    ldi    XL, low(szamok)     ;cél alsó címe
    ldi    XH, high(szamok)    ;cél felső
    adiw  X, 1
    st    X, szam              ;eltároljuk a 2. operandust
    rjmp   loop

osszead:                      ;összeadás
    ldi    digit, 2

```

```

ldi    XL, low(szamok)
ldi    XH, high(szamok)
ld    szam1, X+
ld    szam2, X
add   szam1, szam2
mov   szam, szam1
call  sub_7seg_digit ;Összeg kitétele a 2. digitre
rjmp  loop

```

kivon:

```

ldi    digit, 3
ldi    XL, low(szamok)
ldi    XH, high(szamok)
ld    szam1, X+
ld    szam2, X
cp    szam1, szam2
brlo  kivon1      ;mindig a nagyobb szám kerül kivonásra a ki-
sebből
sub   szam1, szam2
mov   szam, szam1
rjmp  kivon_vege
kivon1:
sub   szam2, szam1
mov   szam, szam2
kivon_vege:
call  sub_7seg_digit ;Különbség kitétele a 2. digitre
rjmp loop

```

```

keyb_nums: .DB 69,0,14,1,13,2,11,3,22,4,21,5,19,6,38,7,37,8,35,9,67,10,70,11
; DB-Define constant byte(s) in program memory or E2PROM memory

```

## 4 Mátrix billentyűzet: Szorzás

### 4.1 Önálló feladat

```
;----- Mátrix-----
;Feladat: Olvassa be a mátrix billentyűzet gombjait (0-9).
; Valósítson meg összeadást, az eredményt jelenítse meg
; a 7szegmenses kijelzőn.
; G0   1. operandus      0. digit
; G1   2. operandus      1. digit
; G2   Összeadás        2. digit
; G3   Kivonás          3. digit (nagyobb szám - kisebb szám)
; G4   Szorzás          LED
;-----
```

Egészítse ki a közösen megírt kódot a szorzás műveletére. G4 megnyomásra szorozza össze a két operandust és szorzatot tegye ki a LED-ekre.

Egyeszerűsítse a kódot. Az ismétlődő/kiemelhető részeket szubrutinban valósítsa meg.

# Programozás II. laboratórium AVR C

## Labor célja

Az AVR C fejlesztői környezet bemutatása Atmega mikrokontroller család segítségével, illetve megismertetni a C nyelven történő program tervezés és megvalósítás technikáit. minden laboron 4 résztémakör kerül érintésre, amelyek feldolgozandó és elsajátítandó tananyag jól elkülöníthető egységeit képezik, illetve a következő laboron számonkérésre is kerülnek. A számonkérésre való felkészülést az útmutatóban kidolgozott és laboron megoldott önálló feladatokon túl az 5. részben található Gyakorló feladatok is segítik. A labor tematikája felsorolásban található oldalszámok ezen jegyzet adott laborhoz tartozó elméleti ismeretanyagait tartalmazzák. Érdemes ezen oldalakon levő ismereteket a labor anyag áttekintése előtt megtanulni vagy átismételni.

Sikeres tanulást kívánnak

a Szerzők.

## **1. Labor**

### **Labor tematikája:**

1. Portkezelés: LED-ek (91-100. oldal)
2. Portkezelés: Nyomógombok (91. oldal, 101-102. oldal)
3. Timer0 (41-51. oldal)
4. Timer0 alkalmazások (61-69. oldal)

# 1 Portkezelés: LED-ek

## 1.1 Közös feladat: Műveletvégzés megjelenítése LED-eken

```
/*-----LED7_0 műveletvégzések (bevezetés)-----  
 * Feladat: Jelenítse meg a 8db LED-en a műveletek eredményét  
 * A LED_out-ot függvényben írja meg  
 */  
  
#include <avr/io.h>  
  
void LED_init(); //LED-ekhez tartozó PIN-ek kimenetre állítása  
void LED_out(char szam); //8bites szám kitétele a LED-ekre  
  
int main(void) //main függvény  
{  
    LED_init(); //LED_init() függvényhívás  
    char szam1=6; //szam1, szam2 változók  
    char szam2=2; // 2 --> 0b00000010 6 --> 0b000000110  
    char szam=0;  
  
    //mindig csak 1 sor legyen aktív  
    szam=szam1+szam2; //összeadás  
    //szam=szam1-szam2; //kivonás  
    //szam=szam1*szam2; //szorzás  
    //szam=szam1/szam2; //osztás  
    //szam=szam1<<2; //balra shift 2-vel  
    //szam=szam1>>2; //jobbra shift 2-vel  
    //szam=~szam1; //bitenkénti NOT  
    //szam=szam1&szam2; //bitenkénti ÉS  
    //szam=szam1|szam2; //bitenkénti VAGY  
    //szam=szam1^szam2; //bitenkénti XOR  
    //szam++; //inkrement  
    //szam--; //dekrement  
  
    LED_out(szam); //szam kitétele a LED-ekre  
  
    while (1) //végzetlen ciklus  
    {  
        }  
}  
  
void LED_init()  
{  
    DDRB=0xF0; //PORTB felső 4 bitjének kimenetre állítása  
    DDRD=0xF0; //PORTD felső 4 bitjének kimenetre állítása  
}  
  
void LED_out(char szam)  
{  
    PORTD=szam&0xF0; //felső 4 bit kitétele a felső 4LED-re és maszkolás  
    PORTB=(szam<<4)&0xF0; //alsó 4 bit shift-elése balra 4-vel  
    //kitétele az alsó 4 LED-re és maszkolás  
}
```

## 1.2 Közös feladat: LED villogtatás, delay

```
/*-----LED7_0 villogtatás-----  
 * Feladat:          Villogtassa a 8db LED-et 0,5 másodpercenként  
 *                  A LED_out-ot függvényben írja meg  
 */  
  
#include <avr/io.h>  
#define F_CPU 16000000UL           // #define F_CPU 8000000UL      órajel  
#include <util/delay.h>          // _delay_ms header file  
  
void LED_init();                //LED-ekhez tartozó PIN-ek kimenetre állítása  
void LED_out(char szam);        //8bites szám kitétele a LED-ekre  
  
int main(void)                 //main függvény  
{  
    LED_init();                //LED_init() függvényhívás  
    char szam=0xFF;             //szam változó  
  
    while (1)                  //végzetlen ciklus  
    {  
        LED_out(szam);         //szam kitétele a LED-ekre  
        szam^=0xFF;              //kizárá vagy kapcsolat (villogtatáshoz)  
        _delay_ms(500);           //0,5s késleltetés  
    }  
  
    void LED_init()  
    {  
        DDRB=0xF0;              //PORTB felső 4 bitjének kimenetre állítása  
        DDRD=0xF0;              //PORTD felső 4 bitjének kimenetre állítása  
    }  
  
    void LED_out(char szam)  
    {  
        PORTD=szam&0xF0;         //felső 4 bit a felső 4LED-re és maszkolás  
        PORTB=(szam<<4)&0xF0;     //alsó 4 bit shift-elése balra 4-ét  
                                //kitétele az alsó 4 LED-re és maszkolás  
    }  
}
```

### 1.3 Közös feladat: Futófény 1.

```
/*-----Futófény-----  
 * Feladat:          Készítsen körbefutófényt  
 *                  A LED_out-ot függvényben írja meg  
 */  
  
#include <avr/io.h>  
#define F_CPU 16000000UL      // #define F_CPU 8000000UL      órajel  
#include <util/delay.h>      // _delay_ms header file  
  
void LED_init();           //LED-ekhez tartozó PIN-ek kimenetre állítása  
void LED_out(char szam);   //8bites szám kitétele a LED-ekre  
  
int main(void)             //main függvény  
{  
    LED_init();            //LED_init() függvényhívás  
    char szam=0x01;  
  
    while (1)              //végtelen ciklus  
    {  
        LED_out(szam);    //szam kitétele a LED-ekre  
        szam=szam<<1;     //balra shift  
        _delay_ms(300);     //300ms késleltetés  
        if (szam==0)         //ha végig értünk  
        {  
            szam=0x01;       //kezdjük előlről  
        }  
    }  
  
    void LED_init()  
    {  
        DDRB=0xF0;          //PORTB felső 4 bitjének kimenetre állítása  
        DDRD=0xF0;          //PORTD felső 4 bitjének kimenetre állítása  
    }  
  
    void LED_out(char szam)  
    {  
        PORTD=szam&0xF0;    //felső 4 bit a felső 4LED-re és maszkolás  
        PORTB=(szam<<4)&0xF0; //alsó 4 bit shift-elése balra 4-et  
                           //kitétele az alsó 4 LED-re és maszkolás  
    }  
}
```

#### 1.4 Közös feladat: Futófény 2.

```
-----Futófény-----
* Feladat: 00011000 kezdőállapot, 00100100 ... 10000001, 0000110000...
*           A LED_out-ot függvényben írja meg
*/
#include <avr/io.h>
#define F_CPU 16000000UL      // #define F_CPU 8000000UL      órajel
#include <util/delay.h>       // _delay_ms header file

void LED_init();           //LED-ekhez tartozó PIN-ek kimenetre állítása
void LED_out(char szam);   //8bites szám kitétele a LED-ekre

int main(void)             //main függvény
{
    LED_init();            //LED_init() függvényhívás

    char szam=0;
    char szam1=0x10;
    char szam2=0x08;

    while (1)              //végzetlen ciklus
    {
        szam=szam1|szam2;
        LED_out(szam);      //szam kitétele a LED-ekre
        szam1=szam1<<1;    //balra shift
        szam2=szam2>>1;    //jobbra shift
        _delay_ms(300);      //300ms késleltetés
        if (szam==0x81)      //ha végig értünk
        {
            szam1=0x10;
            szam2=0x08;       //kezdjük előlről
        }
    }
}

void LED_init()
{
    DDRB=0xF0;             //PORTB felső 4 bitjének kimenetre állítása
    DDRD=0xF0;             //PORTD felső 4 bitjének kimenetre állítása
}

void LED_out(char szam)
{
    PORTD=szam&0xF0;       //felső 4 bit a felső 4LED-re és maszkolás
    PORTB=(szam<<4)&0xF0;  //alsó 4 bit shift-elése balra 4-et
                           //kitétele az alsó 4 LED-re és maszkolás
}
```

#### 1.5 Önálló feladatok

1. Készítsen ide-oda futófényt. A Kezdőállapot 00000001 legyen, ha elérte a 10000000 állapotot a következő állapot 01000000 legyen. Késleltetés: 150ms
2. Készítsen körbefutófényt. A kezdőállapot 00000011 legyen, ha elérte a 11000000 állapotot, akkor a következő a 10000001 legyen, majd kezdje előlről az egészet. Késleltetés: 250ms

## 2 Portkezelés: Nyomógombok

### 2.1 Közös feladat: Gombok beolvasása

```
-----GOMB bevezető-----
* Feladat:          Olvassa be a gombok állapotát.
*                  A lenyomott gombnak megfelelő LED világítson
*
*/
#include <avr/io.h>

void PORT_init();      //LED-ekhez, GOMB-okhoz tartozó PIN-ek kimenetre állítása
void LED_out(char szam); //8bites szám kitétele a LED-ekre

int main(void)           //main függvény
{
    PORT_init();          //PORT_init() függvényhívás

    char szam=0;

    while (1)             //végtelen ciklus
    {
        szam=PING&0x1F;   //gombok olvasása és maszkolás
        LED_out(szam);
    }
}

void PORT_init()
{
    DDRB=0xF0;            //PORTB felső 4 bitjének kimenetre állítása
    DDRD=0xF0;            //PORTD felső 4 bitjének kimenetre állítása
    DDRG=0x00;            //gombok bemenetre állítása
}

void LED_out(char szam)
{
    PORTD=szam&0xF0;     //felső 4 bit a felső 4LED-re és maszkolás
    PORTB=(szam<<4)&0xF0; //alsó 4 bit shift-elése balra 4-ét
                           //kitétele az alsó 4 LED-re és maszkolás
}
```

## 2.2 Közös feladat: Gomb menü

```
/*-----GOMB menü-----  
 * Feladat:          Olvassa be a gombok állapotát.  
 *                  Készítsen menüt az 5 gombhoz  
 */  
  
#include <avr/io.h>  
  
void PORT_init();      //LED-ekhez, GOMB-okhoz tartozó PIN-ek kimenetre állítása  
void LED_out(char szam);    //8bites szám kitétele a LED-ekre  
  
int main(void)           //main függvény  
{  
    PORT_init();          //PORT_init() függvényhívás  
  
    char szam=0;  
  
    while (1)             //végtelen ciklus  
    {  
        szam=PING&0x1F;      //gombok olvasása és maszkolás  
  
        switch(szam)         //menü  
        {  
            case 0x01:   LED_out(0x80);    break;  
            case 0x02:   LED_out(0x40);    break;  
            case 0x04:   LED_out(0x20);    break;  
            case 0x08:   LED_out(0x10);    break;  
            case 0x10:   LED_out(0x08);    break;  
            default:    break;  
        }  
    }  
}  
  
void PORT_init()  
{  
    DDRB=0xF0;            //PORTB felső 4 bitjének kimenetre állítása  
    DDRD=0xF0;            //PORTD felső 4 bitjének kimenetre állítása  
    DDRG=0x00;            //gombok bemenetre állítása  
}  
  
void LED_out(char szam)  
{  
    PORTD=szam&0xF0;      //felső 4 bit a felső 4LED-re és maszkolás  
    PORTB=(szam<<4)&0xF0;  //alsó 4 bit shift-elése balra 4-ét  
                           //kitétele az alsó 4 LED-re és maszkolás  
}
```

## 2.4 Közös feladat: Gomb kapcsoló – számláló

```
-----GOMB számláló-----
* Feladat:          Olvassa be a gombok állapotát.
*                  GOMB0 (1.) lenyomására növelje a szám változót és
*                  értékét jelenítse meg a LED-eken
*/
#include <avr/io.h>

void PORT_init();      //LED-ekhez, GOMB-okhoz tartozó PIN-ek kimenetre állítása
void LED_out(char szam); //8bites szám kitétele a LED-ekre

int main(void)           //main függvény
{
    PORT_init();          //PORT_init() függvényhívás

    char gomb=0;
    char gomb_seged=0;
    char szam=0;

    while (1)             //végtelen ciklus
    {
        gomb=PING&0x1F;      //beolvasás

        if( (gomb) && !(gomb_seged)) //ha lenyomtuk, de előtte nem volt lenyomva
        {

            gomb_seged=1;        //gombnyomás jelzése

            if (gomb==1)         //számláló gomb nyomására léptet
            {
                szam++;
                LED_out(szam);      //megjelenítés
                if (szam==100)       //nullázzuk a számlálót
                {
                    szam=0;
                }
            }
        }

        if(!(PING) && (gomb_seged)) gomb_seged=0;
        //már nem nyomjuk, de előtte nyomtuk
    }

    void PORT_init()
    {
        DDRB=0xF0;           //PORTB felső 4 bitjének kimenetre állítása
        DDRD=0xF0;           //PORTD felső 4 bitjének kimenetre állítása
        DDRA=0x00;           //gomb bemenetre állítása
    }

    void LED_out(char szam)
    {
        PORTD=szam&0xF0;      //felső 4 bit kitétele a felső 4LED-re és maszkolás
        PORTB=(szam<<4)&0xF0; //alsó 4 bit shift-elése balra 4-öt
        //kitétele az alsó 4 LED-re és maszkolás
    }
}
```

## 2.5 Önálló feladat

1. Írja meg a gomb menüt úgy, hogy egy gombnyomásra csak egyszer lépjen be az adott menüpontba.
2. Írjon 5 függvényt, melyeket az elkészített menü 5 pontjában hív meg.
  - a. Körbefutófény
  - b. Ide-oda futófény
  - c. LED2 villogtatása
  - d. Összes LED bekapcsolása
  - e. LED-ek kikapcsolása

### 3 Timer0

#### 3.1 Közös feladat: Timer0 CTC mód, számláló

```
/*-----TIM0 CTC számláló-----  
 * Feladat: Készítsen a LED-ekre felfele számlálót.  
 * Timer0 CTC módját és megszakítást használjon.  
 */  
  
#include <avr/io.h>  
#include <avr/interrupt.h>  
  
unsigned char szam = 0;  
  
void LED_init(void);  
void Timer0_init(void);  
void LED_out(char szam);  
  
int main()  
{  
  
    LED_init();  
    Timer0_init();  
  
    while(1)  
    {  
  
    }  
}  
  
ISR(TIMER0_COMP_vect)           //megszakítást kiszolgáló rutin  
{  
    szam++;  
    LED_out(szam);  
}  
  
void LED_init()  
{  
    DDRB = 0xF0;  
    DDRD = 0xF0;  
}  
  
void LED_out (char szam)  
{  
    PORTD=szam&0xF0;  
    PORTB=(szam<<4)&0xF0;  
}  
  
void Timer0_init()  
{  
    TCCR0 = (1<<CS02) | (1<<CS01) | (1<<CS00) | (1<<WGM01); //CTC mód 1024 előosztás  
    TIMSK = (1<<OCIE0);  
    //megszakítás engedélyezés  
    sei();  
    //globális IT EN  
    TCNT0=0;      //Timer counter - Timer aktuális értéke  
    OCR0 = 249;   //Timer maximális értéke - 8bites max 255  
                  //0-249      16000000/1024/250  8000000/1024/250  
}
```

### 3.2 Közös feladat: Timer0 OVF IT

```
/*-----TIM0 OVF IT-----  
 * Feladat:           Készítsen a LED-ekre felfele számlálót.  
 *                      Timer0 OVF módját és megszakítást használjon.  
 */  
  
#include <avr/io.h>  
#include <avr/interrupt.h>  
  
unsigned char szam = 0;  
  
void LED_init(void);  
void Timer0_init(void);  
void LED_out(char szam);  
  
int main()  
{  
  
    LED_init();  
    Timer0_init();  
  
    while(1)  
    {  
  
    }  
}  
  
ISR(TIMER0_OVF_vect)          //megszakítást kiszolgáló rutin  
{  
    szam++;  
    LED_out(szam);  
}  
  
void LED_init()  
{  
    DDRB = 0xF0;  
    DDRD = 0xF0;  
}  
  
void LED_out (char szam)  
{  
    PORTD=szam&0xF0;  
    PORTB=(szam<<4)&0xF0;  
}  
  
void Timer0_init()  
{  
    TCCR0 = (1<<CS02) | (1<<CS01) | (1<<CS00); //1024 előosztás  
    TIMSK = (1<<TOIE0);  
    //megszakítás engedélyezés  
    sei();  
    //globális IT EN  
}
```

### 3.3 Közös feladat: Timer0 CTC IT 1s

```

-----TIM0 CTC IT-----
* Feladat: Készítsen a futófényt. (1s időzítés)
*           Timer0 CTC módját és megszakítást használjon.
*/
#include <avr/io.h>
#include <avr/interrupt.h>

unsigned char szam =0x01;
int mp=0;

void LED_init(void);
void Timer0_init(void);
void LED_out(char szam);

int main()
{
    LED_init();
    Timer0_init();

    while(1)
    {

    }

    ISR(TIMER0_COMP_vect)          //megszakítást kiszolgáló rutin
    {
        mp++;
        if (mp==249)              //16MHz - 249 8MHz - 124
        {                         //számláló
            LED_out(szam);
            szam=szam<<1;
            if (szam==0)
            {
                szam=0x01;
            }
            mp=0;
        }
    }

    void LED_init()
    {
        DDRB = 0xF0;
        DDRD = 0xF0;
    }

    void LED_out (char szam)
    {
        PORTD=szam&0xF0;
        PORTB=(szam<<4)&0xF0;
    }

    void Timer0_init()
    {
        TCCR0 = (1<<CS02) | (1<<CS01) | (0<<CS00) | (1<<WGM01); //CTC mód 256 előosztás
        TIMSK = (1<<OCIE0);           //megszakítás engedélyezés
        sei();                         //globális IT EN
        TCNT0=0;                      //Timer counter - Timer aktuális értéke
        OCR0 = 249;                   //Timer maximális értéke - 8bites max 255
        //0-249      16000000/256/250     8000000/256/250    4ms   8ms
    }
}

```

### 3.4 Önálló feladat

1. Villogtassa a 8db LED-et ~0,5s-os időzítéssel. A feladathoz használja a Timer0 CTC módját.
2. Készítse el a korábban megvalósított futófényeket delay használata helyett megszakítással.
3. Olvassa be a gombok állapotát, de folyamatos beolvasás helyett megszakítást használjon.

## 4 Timer0 alkalmazása

### 4.1 Közös feladat: Timer0 HW PWM LED0

```
/*-----TIM0 PWM-----  
 *Feladat:          Változtassa a LED0 fényerejét.  
 *  
 */  
  
#include <avr/io.h>  
#define F_CPU 16000000UL  
#include <util/delay.h>  
  
unsigned char szam =0x01;  
int mp=0;  
  
void LED_init(void);  
void Timer0_init(void);  
void LED_out(char szam);  
  
int main()  
{  
    LED_init();  
    Timer0_init();  
  
    unsigned char pwm_d=0;  
  
    while(1)  
    {  
        OCR0=pwm_d;                                //kitöltés  
        _delay_ms(15);  
        pwm_d+=1;                                  //kitöltés növelése  
        if (pwm_d>=250)                            //max érték legyen  
        {  
            pwm_d=0;  
            OCR0=pwm_d;  
            _delay_ms(1000);                         //1s-ig nem világít  
        }  
    }  
  
    void LED_init()  
    {  
        DDRB = 0xF0;  
        DDRD = 0xF0;  
    }  
  
    void LED_out (char szam)  
    {  
        PORTD=szam&0xF0;  
        PORTB=(szam<<4)&0xF0;  
    }  
  
    void Timer0_init()  
    {  
        TCCR0 = (0<<CS02) | (1<<CS01) | (1<<CS00) | (1<<COM01) | (1<<WGM00);  
        //PWM Phase Correct 32 előosztás  
    }  
}
```

## 4.2 Közös feladat: Timer0 SW PWM

```
/*-----TIM0 OVF IT SW PWM-----  
 *Feladat: Készítsen a LED-ekre szoftveres PWM-et.  
 */  
  
#include <avr/io.h>  
#include <avr/interrupt.h>  
  
unsigned char szam = 0;  
unsigned char pwm = 1;  
  
void LED_init(void);  
void Timer0_init(void);  
void LED_out(char szam);  
  
int main()  
{  
    LED_init();  
    Timer0_init();  
  
    while(1)  
    {  
  
    }  
}  
  
ISR(TIMER0_OVF_vect) //megszakítást kiszolgáló rutin  
{  
    szam++; //számláló  
    if (szam>pwm) //kikapcsoljuk az összes LED-et  
    {  
        LED_out(0x00);  
        if (szam>100)  
        {  
            szam=0;  
        }  
    }  
    else  
    {  
        LED_out(0xFF); //bekapcsoljuk az összes LED-et  
    }  
}  
  
void LED_init()  
{  
    DDRB = 0xF0;  
    DDRD = 0xF0;  
}  
  
void LED_out (char szam)  
{  
    PORTD=szam&0xF0;  
    PORTB=(szam<<4)&0xF0;  
}  
  
void Timer0_init()  
{  
    TCCR0 = (0<<CS02) | (1<<CS01) | (0<<CS00); //8 előosztás  
    TIMSK = (1<<TOIE0); //megszakítás engedélyezés  
    sei(); //globális IT EN  
}
```

### 4.3 Önálló feladatok

1. Készítsen LED fényerőszabályozót. GOMB0 lenyomására növelje a LED0 fényerejét, GOMB1 lenyomására csökkentse. Ha lenyomja a GOMB2-t, akkor kapcsolja ki a LED-et, GOMB3 nyomására pedig világítson teljes fényerőn.
2. Végezze el az 1-es feladatot a 8db LED-re is.

## 5 Gyakorló feladatok

1. Készítse el az órai gomb menüket úgy, hogy a gombok beolvasásához megszakítást használjon.
2. Készítsen a 8db LED-re számlálót. A számláló 0-150ig számláljon folyamatosan. GOMB0 nyomására induljon el a számlálás, GOMB4 megnyomására pedig álljon meg. A GOMB2-vel törölje a kijelzett értéket (ne világítson egy LED sem).
3. Készítsen tetszőleges futófényt, melynél a léptetés az állapotok között gombnyomásra történik.
4. Készítsen bináris dátum kijelzőt a LED-ekre. GOMB0 megnyomására írja ki az évet (18), GOMB1 megnyomására a hónapot, GOMB2 megnyomására a nap-ot, GOMB3 megnyomására az órát, GOMB4-re pedig a percet. Az értékeket előre állítsa be, a gombolvasásához megszakítást használjon.

## **2. Labor**

### **Labor tematikája:**

1. Portkezelés: 7szegmenses kijelző (109-103. oldal)
2. Timer1 (47-61. oldal)
3. Megszakítások alkalmazása: Számlálók
4. Megszakítások alkalmazása: Óra

# 1 Portkezelés: 7szegmenses kijelző

## 1.1 Közös feladat: 7szegmenses kijelző 1 digit

```
/*-----7szeg 1 digit-----  
*Feladat:    Írjon ki egy tetszőleges számot(0-9)  
*              a 7szegmenses kijelző tetszőleges digitjére(0-3)  
*  
*/  
  
#include <avr/io.h>  
  
void init();  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit);  
  
int main(void)  
{  
    init();  
    SEVSEG_putdigit(1,0);           // (szám, digit)  
  
    while (1)  
    {  
        }  
    }  
  
void init()  
{  
    DDRA=0xFF;                   // kimenetre állítás  
}  
  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit)  
{  
    if (digit>3) return;  
    if (szam>9) return;  
  
    PORTA=0x80 | (digit)<<4 | szam; // kiíratás 0x80=enable;  
  
    //PA7 engedélyezés  
    //PA6 :  
    //                                     digit3      digit2      digit1  
    digit0  
    //PA5-4 digit választó:   11          10          01          00  
}
```

## 1.2 Közös feladat: 7szegmenses kijelző 4digit delay

```
/*-----7szeg 4 digit-----*
*Feladat: Írjon ki egy tetszőleges négyjegyű számot(0000-9999)
*          a 7szegmenses kijelzőre
*/
#include <avr/io.h>
#define F_CPU 16000000UL //80000000UL
#include <util/delay.h>

void init();
void SEVSEG_putdigit(unsigned char szam, unsigned char digit);
void SEVSEG_putnumber (int val);

unsigned char digit[4]={0};
int j=0;

int main(void)
{
    init();

    while (1)
    {
        SEVSEG_putnumber(4567);           //négyjegyű szám kitétele
        _delay_ms(2);
    }
}

void init()
{
    DDRA=0xFF;                      //kimenetre állítás
}

void SEVSEG_putdigit(unsigned char szam, unsigned char digit)
{
    if (digit>3) return;
    if (szam>9) return;

    PORTA=0x80 | (digit)<<4 | szam; //kiíratás 0x80=enable;
}

void SEVSEG_putnumber (int val)
{
    digit[0]=val%10;                //előállítás digitenként
    digit[1]=(val/10)%10;
    digit[2]=(val/100)%10;
    digit[3]=(val/1000)%10;

    j = (j+1)%4;                  //folyamatos kiíráshoz
    SEVSEG_putdigit(digit[j], j);
}
```

### 1.3 Közös feladat: 7szegmenses kijelző 1digites számláló

```
-----7szeg 1digit számláló delay-----
*Feladat:      Készítsen 0-9ig felfele számlálót 0,5s-os időzítéssel.
*
*/
#include <avr/io.h>
#define F_CPU 16000000UL           //8000000UL
#include <util/delay.h>

void init();
void SEVSEG_putdigit(unsigned char szam, unsigned char digit);

int main(void)
{
    init();

    unsigned char numb=0;

    while (1)
    {
        SEVSEG_putdigit(numb, 0);
        numb++;                      //számlálás
        _delay_ms(500);              //késleltetés
        if (numb==10)                //nullázás
        {
            numb=0;
        }
    }
}

void init()
{
    DDRA=0xFF;                   //kimenetre állítás
}

void SEVSEG_putdigit(unsigned char szam, unsigned char digit)
{
    if (digit>3) return;
    if (szam>9) return;

    PORTA=0x80 | (digit)<<4 | szam;   //kiíratás 0x80=enable;
}
```

#### 1.4 Önálló feladatok

1. Villogtassa a 7szegmenses kijelzőhöz tartozó :t 0,5s-os ütemben.
2. Készítse el a 8-0-ig számlálót, mely 0,4s-os ütemben számláljon.
3. Írja ki a 7szegmenses kijelzőre a 123 decimális számot 4es/8as számrendszerben.

## 2 Timer1

### 2.1 Közös feladat: Timer1 CTC mód megszakítás 1s

```
-----Timer1 1s CTC IT-----
*Feladat:          Villogtassa a LED0-t 1s-onként
*
*/
#include <avr/io.h>
#include <avr/interrupt.h>

void Timer1Init();
void LED_init();

int main()
{
    LED_init();
    Timer1Init(); //Timer beállítása
    while(1)
    {
    }

    ISR(TIMER1_COMPA_vect)
    { //Timer1 komparálási interrupt
        PORTB^=0x10;
    }

    void Timer1Init() {
        TCCR1B = (1<<WGM12) | (1<<CS12) | (1<<CS10);      //ctc      //1024
        TCCR1C = 0;
        OCR1A = 15624;           // max 65535 (16bit-es)  1s (16MHz)  2s (8MHz)
                                //8000000/256 31249 -->1s
        TIMSK |= (1<<OCIE1A); //engedélyezés
        sei();
    }

    void LED_init()
    {
        DDRD = 0xF0; //LEDEk beállítása
        DDRB = 0xF0;
    }
}
```

## 2.2 Közös feladat Timer1 OVF IT

```
/*-----Timer1 1s OVF IT-----*
*Feladat:      Villogtassa a LED0-t OVF megszakítással.
*                  Csökkentse az előosztás mértékét és
*                  figyelje a változásokat.
*/
#include <avr/io.h>
#include <avr/interrupt.h>

void Timer1Init();
void LED_init();

int main()
{
    LED_init();
    Timer1Init(); //Timer beállítása
    while(1)
    {
    }

    ISR(TIMER1_OVF_vect)
    { //Timer1 overflow interrupt
        PORTB^=0x10;
    }

    void Timer1Init() {
        TCCR1B = (1<<CS12) | (1<<CS10);           //1024 előosztás
        TCCR1C = 0;
        TIMSK |= (1<<TOIE1);                      //engedélyezés
        sei();
    }

    void LED_init()
    {
        DDRD = 0xF0; //LEDEk beállítása
        DDRB = 0xF0;
    }
}
```

## 2.3 Önálló feladatok

1. Készítsen tetszőleges futófényt Timer1 OVF megszakítással.
2. Készítsen tetszőleges futófényt Timer1 CTC megszakítással 0,2s-os időzítéssel.
3. Készítsen másodperc számlálót a LED-ekre, mely 0-59ig számoljon.

### 3 Megszakítások alkalmazása: számlálók

#### 3.1 Közös feladat: 7szegmenses kijelző 4digites szám kiírása megszakítással

```
-----7szeg 4 digit megszakítás-----
*Feladat:    Írjon ki egy tetszőleges négyjegyű számot(0000-9999)
*              a 7szegmenses kijelzőre
*/
#include <avr/io.h>
#include <avr/interrupt.h>

void init();
void SEVSEG_putdigit(unsigned char szam, unsigned char digit);
void SEVSEG_putnumber (int val);

unsigned char digit[4]={0};
int j=0;

int main(void)
{
    init();

    while (1)
    {

    }

    void init()
    {
        DDRA=0xFF;                                //kimenetre állítás
        TCCR0 = (1<<CS01)| (1<<CS00);          //32 előosztás
        TIMSK=(1<<TOIE0);
        sei();
    }

    ISR(TIMER0_OVF_vect)
    {
        SEVSEG_putnumber (2018);
    }

    void SEVSEG_putdigit(unsigned char szam, unsigned char digit)
    {
        if (digit>3) return;
        if (szam>9) return;
        PORTA=0x80 | (digit)<<4 | szam;      //kiiratás 0x80=enable;
    }

    void SEVSEG_putnumber (int val)
    {
        digit[0]=val%10;                          //előállítás digitenként
        digit[1]=(val/10)%10;
        digit[2]=(val/100)%10;
        digit[3]=(val/1000)%10;

        j = (j+1)%4;                            //folyamatos kiírás
        SEVSEG_putdigit(digit[j], j);
    }
}
```

### 3.2 Közös feladat: 7szegmenses kijelző 2digites számláló

```
/*-----7szeg számláló 2digit delay-----  
*Feladat:           Készítsem 0-59-ig számlálót.  
*                  A megjelenítéshez megszakítást használjon.  
*/  
  
#include <avr/io.h>  
#include <avr/interrupt.h>  
#define F_CPU 16000000UL          //80000000UL  
#include <util/delay.h>  
  
void init();  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit);  
void SEVSEG_putnumber_t();  
  
unsigned char digit[4]={0};  
int j=0;  
  
int main(void)  
{  
    init();  
  
    while (1)  
    {  
        _delay_ms(500);           //0,5s késleltetés  
        digit[0]++;              //1-es helyiérték  
        if (digit[0]==10)         //túlléptük a max értéket  
        {  
            digit[0]=0;           //1-es helyiérték újrakezd  
            digit[1]++;            //10-es növelése  
            if (digit[1]==6)        //59-ig számolunk  
            {  
                digit[1]=0;          //0000-ról újrakezdjük  
            }  
        }  
    }  
}  
  
void init()  
{  
    DDRA=0xFF;                 //kimenetre állítás  
    //timer0 setup  
    TCCR0 = (1<<CS01)| (1<<CS00);      //32 előosztás  
    TIMSK=(1<<TOIE0);  
    sei();  
}  
  
ISR(TIMER0_OVF_vect)  
{  
    SEVSEG_putnumber_t ();  
}  
  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit)  
{  
    if (digit>3) return;  
    if (szam>9) return;  
    PORTA=0x80 | (digit)<<4 | szam;       //kiiratás 0x80=enable;  
}
```

```
void SEVSEG_putnumber_t ()  
{  
    j = (j+1)%4;                                //folyamatos kiírás  
    SEVSEG_putdigit(digit[j], j);  
}
```

### 3.3 Közös feladat: 2digites számláló megszakítással

```
/*-----Timer1 1s CTC 2digit számláló-----*
*Feladat: Készítsen 1digites számlálót a 7szegmenses kijelzőre.
*A megjelenítéshez és a számlálás időzítéséhez is megszakítást használjon.
*Számoljon 0-59ig 1s-os lépésekben.
*/
#include <avr/io.h>
#include <avr/interrupt.h>

void Timer1Init();
void PORT_init();
void Timer0Init();
void SEVSEG_putdigit(unsigned char szam, unsigned char digit);
void SEVSEG_putnumber_t ();

unsigned char digit[4]={0};
int j=0;

int main()
{
    PORT_init();
    Timer1Init();           //Timer beállítása
    Timer0Init();
    sei();
    while(1)
    {
        }
    }

ISR(TIMER1_COMPA_vect)
{
    digit[0]++;
    if (digit[0]==10)
    {
        digit[0]=0;
        digit[1]++;
        if (digit[1]==6)
        {
            digit[1]=0;
        }
    }
}

ISR(TIMER0_OVF_vect)
{
    SEVSEG_putnumber_t();
}
```

```

void Timer1Init() {
    TCCR1B = (1<<WGM12) | (1<<CS12) | (1<<CS10);      //ctc      //1024
    TCCR1C = 0;
    OCR1A = 15624;           //max 65535 (16bit-es)   1s (16MHz)  2s (8MHz)
    TIMSK |= (1<<OCIE1A);                         //8000000/256 31249 -->1s
}

void Timer0Init()
{
    TCCR0=(0<<CS02)|(1<<CS01)|(1<<CS02);
    TIMSK|=(1<<TOIE0);
}

void PORT_init()
{
    DDRA = 0xFF;
}

void SEVSEG_putdigit(unsigned char szam, unsigned char digit)
{
    if (digit>3) return;
    if (szam>9) return;

    PORTA=0x80 | (digit)<<4 | szam;                  //kiíratás 0x80=enable;
}

void SEVSEG_putnumber_t()
{
    j = (j+1)%4;                                     //folyamatos kiírás
    SEVSEG_putdigit(digit[j], j);
}

```

### 3.4 Önálló feladatok

1. Készítsen 0-99ig számlálót a 7szegmenses kijelző 2. és 3. digitjére. A feladatot úgy oldja meg, hogy az alsó két digiten ne kerüljön kijelzésre még a 0 sem. Az időzítés 0,5s legyen.
2. Készítsen másodperc számlálót (stopper) a 7szegmenses kijelzőre. Az számlálást a GOMB0-val indítsa, GOMB1 lenyomásával állítsa meg. A kijelzett érték nullázása a GOMB4 lenyomására törtenjen.

## **4 Megszakítások alkalmazása: Óra**

### **4.1 Közös/önálló feladat**

Készítsen órát a 7szegmenses kijelzőre és a LED-ekre.

A : fél másodpercenként villogjon. A hétszegmenses kijelzőn jelenítse meg az órát és a perct: HH:MM, a LED-eken pedig a másodpercek kijelzése történjen. Minimális kijelzett érték 00:00 00000000, maximális kijelzett érték: 23:59 00111011

A helyes működés ellenőrzése érdekében csökkentse az egyes állapotok közti időt.

## 5 Gyakorló feladatok

1. Készítsen gomb menüt az 5 nyomógombra. A lenyomott gomb számát írja ki a 7szegmenses kijelző 2. digitjére.
2. Készítsen Knight Rider futófényt 0,2s-os időzítéssel. A 22. állapot írtékét írja decimálisan a 7szegmenses kijelzőre.
3. Készítsen számlálót a 7szegmenses kijelző középső 2 digitjére. A számláló 4es számrendszerben számláljon 00-33-ig 0,4s-onként. Az időzítéshez megszakítást használjon.
4. Készítsen számlálót a 7szegmenses kijelzőre, mely 77-00-ig számol 8as számrendszerben. Az időzítés 0,6ms legyen.
5. Az órán megoldott órát egészítse ki azzal, hogy gombnyomásra lehessen beállítani az időt. Kezdőérték 00:00 00000000. Az óra indítása a GOMB4-re történjen, megállítása a GOMB3-ra. A GOMB0 növelje a perc értékét, a GOMB1 csökkentse. Az órát a GOMB3-mal állítsa valamelyik irányba.

### 3. Labor

#### Labor tematikája:

1. Portkezelés: Mátrix billentyűzet (117-121. oldal)
2. Mátrix billentyűzet: Menü
3. Mátrix billentyűzet: többjegyű szám bevitelle
4. Timer-ek alkalmazása: Timer1/3 PWM (61-69. oldal)

## 1 Mátrix billentyűzet

### 1.1 Közös feladat: Lenyomott mátrix bill megjelenítése a 7szegmenses kijelzőn

```
/*-----Mátrix billentyűzet alapkód-----  
 *Feladat:          Jelenítse meg a leütött mátrix billentyű számát  
 *                  a 7szegmenses kijelzőn (0-9).  
 */  
  
#include <avr/io.h>  
#define F_CPU 16000000  
#include <util/delay.h>  
  
void init();  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit);  
void beolvas_kiir();  
  
unsigned char m_button = 0;  
unsigned char row = 0x08;           //1. sor "címe"  
  
int main(void)  
{  
    init();  
  
    while (1)  
    {  
        beolvas_kiir();           //folyamatos beolvasás és megjelenítés  
    }  
}  
  
void init()  
{  
    DDRA=0xFF;                 //7seg kimenetre állítás  
    DDRC = 0x78;                //mátrix  
}  
  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit)  
{  
    if (digit>3) return;  
    if (szam>9) return;  
  
    PORTA=0x80 | (digit)<<4 | szam; //kiíratás 0x80=enable;  
}  
  
void matrix()  
{  
    const unsigned char billtomb[12] = { 69 , 14, 13, 11, 22, 21, 19, 38, 37, 35,  
70, 67 };  
                                //0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  
#,   *  
    unsigned char num, bill;
```

```

PORTC = row;           //címzés          //kezdőcím: 0x08, első sor
_delay_ms( 5 );        //várakozás
bill = PINC & 0x7f;    //maszkolás 0b01111111
num = 0;               //tömbindex törlése

while( num<12 )
{
    if( bill == billtomb[num] ){
        m_button = num;           //tömb elemek ellenőrzése
        break;                   //ha egyezés van kiküldjük a számot
    }
    else{
        m_button = 12;           //12 ha nem nyomunk semmit
        num++;                  //következő tömbindex
    }
}
if( row == 0x40 )row = 0x08;      //első sorra állítás
else row = row << 1;            //sorcímzés léptetése

void beolvas_kiir()
{
    matrix();                 //bill beolvasása
    if (m_button<=9)
    {
        SEVSEG_putdigit(m_button, 0);   //megjelenítés a 0. digiten
    }
}

```

## 1.2 Közös feladat: Mátrix bill olvasása megszakítással

```
/*-----Mátrix billentyűzet IT-----  
 *Feladat:                Jelenítse meg a leütött mátrix billentyű számát  
 *                           a 7szegmenses kijelzőn (0-9).  
 */  
  
#include <avr/io.h>  
#define F_CPU 16000000  
#include <util/delay.h>  
#include <avr/interrupt.h>  
  
void init();  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit);  
void beolvas_kiir();  
  
unsigned char m_button = 0;  
unsigned char row = 0x08;           //1. sor "címe"  
  
int main(void)  
{  
    init();  
  
    while (1)  
    {  
  
    }  
}  
  
ISR(TIMER0_OVF_vect)           //megszakítást kiszolgáló rutin  
{  
    beolvas_kiir();  
}  
  
void init()  
{  
    DDRA=0xFF;                  //7seg kimenetre állítás  
    DDRC = 0x78;                 //mátrix  
    //TIM0 OVF  
    TCCR0 = (1<<CS01)| (1<<CS00);  
    TIMSK=(1<<TOIE0);  
    sei();  
}  
  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit)  
{  
    if (digit>3) return;  
    if (szam>9) return;  
  
    PORTA=0x80 | (digit)<<4 | szam;   //kiíratás 0x80=enable;  
}  
  
void matrix()  
{  
    const unsigned char billtomb[12] = { 69 , 14, 13, 11, 22, 21, 19, 38, 37, 35,  
70, 67 };  
                                //0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  
#,   *  
    unsigned char num, bill;  
  
    PORTC = row;                  //címezés          //kezdőcím: 0x08, első sor  
    bill = PINC & 0x7f;           //maszkolás 0b01111111  
    num = 0;                      //tömbindex törlése
```

```

while( num<12 )
{
    if( bill == billtomb[num] ){
        m_button = num;
        break;
    }
    else{
        m_button = 12;
        num++;
    }
}
if( row == 0x40 )row = 0x08;
else row = row << 1;
}

void beolvas_kiir()
{
    matrix();                                //bill beolvasása
    if (m_button<=9)
    {
        SEVSEG_putdigit(m_button, 0);      //megjelenítés a 0. digiten
    }
}

```

### 1.3 Önálló feladat:

1. A mátrix billentyűzet \* billentyűjének lenyomásakor kapcsolja be a 8db LED-et, a # lenyomására pedig kapcsolja ki azokat.

## 2 Mátrix billentyűzet: Menü

```
/*-----Mátrix billentyűzet menü-----  
*Feladat:           Jelenítse meg a leütött mátrix billentyű számát  
*                  Készítsen mátrix menüt!  
*  
*/  
  
#include <avr/io.h>  
#define F_CPU 16000000  
#include <util/delay.h>  
#include <avr/interrupt.h>  
  
void init();  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit);  
void beolvas_kiir();  
void menu();  
  
unsigned char m_button = 12;  
unsigned char row = 0x08;           //1. sor "címe"  
  
int main(void)  
{  
    init();  
  
    while (1)  
    {  
  
    }  
}  
  
ISR(TIMER0_OVF_vect)           //megszakítást kiszolgáló rutin  
{  
    beolvas_kiir();  
    menu();  
}  
  
void init()  
{  
    DDRA=0xFF;                  //7seg kimenetre állítás  
    DDRC=0x78;                  //mátrix  
    DDRB=0xF0;  
    DDRE|=0x0C;  
    //TIM0 OVF  
    TCCR0 = (1<<CS01)| (1<<CS00);  
    TIMSK=(1<<TOIE0);  
    sei();  
}  
  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit)  
{  
    if (digit>3) return;  
    if (szam>9) return;  
  
    PORTA=0x80 | (digit)<<4 | szam;      //kiiratás 0x80=enable;  
}
```

```

void matrix()
{
    const unsigned char billtomb[12] = { 69 , 14, 13, 11, 22, 21, 19, 38, 37, 35,
70, 67 };
                                //0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
#,   *
    unsigned char num, bill;

PORTC = row;           //címzés          //kezdőcím: 0x08, első sor
bill = PINC & 0x7f;    //maszkolás 0b01111111
num = 0;                //tömbindex törlése

while( num<12 )
{
    if( bill == billtomb[num] ){
        m_button = num;      //tömb elemek ellenőrzése
        break;               //ha egyezés van kiküldjük a számot
                               //folyamat befejezése
    }
    else{
        m_button = 12;      //12 ha nem nyomunk semmit
        num++;              //következő tömbindex
    }
}
if( row == 0x40 )row = 0x08; //első sorra állítás
else row = row << 1;        //sorcímzés léptetése
}

void beolvas_kiir()
{
    matrix();             //bill beolvasása

    if (m_button<=9)
    {
        SEVSEG_putdigit(m_button, 0);
    }
}

void menu()
{
    if (m_button<=9)
    {
        switch (m_button)
        {
            case 0:    PORTB=0xF0;  break;
            case 1:    PORTB=0xA0;  break;
            case 2:    PORTB=0x00;  break;
            case 3:    PORTE=0x0C; break;
            case 4:    PORTE=0x00; break;
            //...
            default:   break;
        }
    }
}

```

## 2.1 Önálló feladat

1. Készítsen mátrix billentyűzet menüt, mellyel LED fényerőt tud szabályozni. 0-s gomb lenyomására kapcsolja ki a LED-eket. 1-es gomb lenyomására legyen 10% a kitöltés, 2-es gomb lenyomására 20%, ..., 9-es gomb lenyomására 90%.

### 3 Mátrix billentyűzet: többjegyű szám bevitelle

```
/*-----Mátrix billentyűzet 2jegy-----  
 *Feladat:          Olvassa be a mátrix billentyűzetet,  
 *                  Jelenítsen meg a 7szegmenses kijelzőn 2jegyű számokat.  
 */  
  
#include <avr/io.h>  
#define F_CPU 16000000  
#include <util/delay.h>  
#include <avr/interrupt.h>  
  
void init();  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit);  
void beolvas_kiir();  
  
unsigned char m_button = 12;  
unsigned char row = 0x08;                                //1. sor "címe"  
  
unsigned char digit[2]={0};  
int j=0;  
int i=1;  
  
int main(void)  
{  
    init();  
  
    while (1)  
    {  
        beolvas_kiir();  
    }  
}  
  
void init()  
{  
    DDRA=0xFF;                                         //7seg kimenetre állítás  
    DDRC = 0x78;                                         //mátrix  
}  
  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit)  
{  
    if (digit>3) return;  
    if (szam>9) return;  
  
    PORTA=0x80 | (digit)<<4 | szam;      //kiiratás 0x80=enable;  
}  
  
void matrix()  
{  
    const unsigned char billtomb[12] = { 69 , 14, 13, 11, 22, 21, 19, 38, 37, 35,  
70, 67 };  
                                //0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  
#,   *  
    unsigned char num, bill;  
  
    PORTC = row;           //címzés           //kezdőcím: 0x08, első sor  
    _delay_ms(5);  
    bill  = PINC & 0x7f;    //maszkolás 0b01111111  
    num  = 0;              //tömbindex törlése
```

```

while( num<12 )
{
    if( bill == billtomb[num] ){
        m_button = num;
        while(PINC == billtomb[num]);
        break;
    }
    else{
        m_button = 12;
        num++;
    }
}
if( row == 0x40 )row = 0x08;
else row = row << 1;
}

void SEVSEG_putnumber_t()
{
    j = (j+1)%2;
    SEVSEG_putdigit(digit[j], j);
}

void beolvas_kiir()
{
    matrix();                                //m.bill beolvasása

    if (m_button<=9)
    {
        i=1;

        while (i)
        {
            digit[i] = digit[i-1];
            i--;
        }

        digit[0] = m_button;
    }
    SEVSEG_putnumber_t();
}

```

### 3.1 Önálló feladat

Olvasson be és jelenítsen meg 4jegyű számokat.

## 4 Timer-ek alkalmazása: Timer1/3 PWM

### 4.1 Közös feladat: Timer1 Fast PWM ICR1

```
/*-----TIM1 PWM-----  
*Feladat:      Inicializálja a Timer1 Fast PWM módját  
*              Állítson be különböző kitöltést a LED-eknek.  
*/  
#include <avr/io.h>  
  
void LED_init();  
void Timer1Init();  
  
int PWM_duty0=1;  
int PWM_duty1=100;  
int PWM_duty2=500;  
  
int main()  
{  
    LED_init();           //LED-ek beállítása  
    Timer1Init();         //Timer beállítása  
    PORTD=0x10;  
  
    while(1)  
    {  
        }  
    }  
  
void Timer1Init()  
{  
    TCCR1A = (0<<COM1A0) | (1<<COM1A1) | //clear  
    (0<<COM1B0) | (1<<COM1B1) |  
    (0<<COM1C0) | (1<<COM1C1) |  
    (0<<WGM10) | (1<<WGM11);           //fast pwm ICRn  
  
    TCCR1B = (0 << ICNC1) | (0 << ICES1) |  
    (1 << WGM13) | (1 << WGM12) |  
    (1<<CS11) | (1<<CS00);           //F_CPU/64  
  
    TCCR1C = 0;  
    ICR1 = 1000;  
    OCR1A = PWM_duty0;                  //duty 0-1000  
    OCR1B = PWM_duty1;                  //duty 0-1000  
    OCR1C = PWM_duty2;                  //duty 0-1000  
}  
  
void LED_init()  
{  
    DDRD = 0xF0; //LEDEk beállítása  
    DDRB = 0xF0;  
}
```

## 4.2 Közös feladat: RGB zöld PWM

```
/*-----TIM3 PWM-----  
*Feladat:      Változtassa az RGB zöld LED fényerejét.  
*  
*/  
#include <avr/io.h>  
  
void LED_init();  
void Timer3Init();  
  
int pwm_d=10;  
  
int main()  
{  
    DDRE|=0x08;           //zöld kimenetre  
    LED_init();           //LED-ek beállítása  
    Timer3Init();         //Timer beállítása  
  
    while(1)  
    {  
        }  
    }  
  
void Timer3Init()  
{  
    TCCR3A = (1<<COM3A1)|(1<< WGM31);  
    TCCR3B = (1<< WGM33) |(1<<CS30)| (1<<CS31) ;           //chase correct pwm ICR3  
                                                               //F_CPU/64  
  
    ICR3=1000;  
    OCR3A=pwm_d;  
}  
  
void LED_init()  
{  
    DDRD = 0xF0; //LEDEk beállítása  
    DDRB = 0xF0;  
}
```

## 4.3 Önálló feladatok

1. Az első feladatot oldja meg Phase Correct PWM mód alkalmazásával.
2. Készítsen tetszőleges színkeverést szoftveres PWM-mel az RGB LED-ekre.
3. Növelje folyamatosan az RGB zöld LED fényerejét, majd csökkentse gombnyomásra.

## 5 Gyakorló feladatok

1. A lenyomott mátrix billentyű számát jelenítse meg a LED-eken.
2. Készítsen mátrix menüt, melyben az 1-es, 4-es, 7-es gombnyomásokra különböző futófények indulnak el.
3. Készítsen nyomógomb lenyomására változtatható RGB színkeverést. 0-2 gombok lenyomására növelje a R(0)-G(1)-B(2) LED-ek fényerejét ~5%-onként, GOMB3 lenyomására világítson az összes maximális fényerőn, GOMB4 lenyomására pedig kapcsolja ki az összeset.

## **4. Labor**

### **Labor tematikája:**

1. Mátrix billentyűzet: számológép 1jegyű számokkal
2. Mátrix billentyűzet: számológép 2jegyű számokkal
3. Mátrix billentyűzet: kódzár

# 1 Mátrix billentyűzet: Számológép1

## 1.1 Közös feladat: Egyjegyű számológép

```
/*-----Mátrix billentyűzet számológép1-----  
 *Feladat: Készítsen egyjegyű számológépet.  
 *  
 */  
  
#include <avr/io.h>  
#define F_CPU 16000000  
#include <util/delay.h>  
#include <avr/interrupt.h>  
  
void init();  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit);  
void beolvas_kiir();  
void LED_out(unsigned char szam);  
void gomb_menu();  
  
unsigned char m_button = 12;  
unsigned char row = 0x08; //1. sor "címe"  
  
unsigned char digit[4]={0};  
int j=0;  
int i=1;  
unsigned char gomb=0;  
unsigned char gomb_seged=0;  
  
int main(void)  
{  
    init();  
  
    while (1)  
    {  
        beolvas_kiir();  
        gomb_menu();  
    }  
}  
  
void init()  
{  
    DDRA=0xFF; //7seg kimenetre állítás  
    DDRC = 0x78; //mátrix  
    DDRB=DDRD=0xF0;  
}  
  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit)  
{  
    if (digit>3) return;  
    if (szam>9) return;  
  
    PORTA=0x80 | (digit)<<4 | szam; //kiiratás 0x80=enable;  
}  
  
void matrix()  
{  
    const unsigned char billtomb[12] = { 69 , 14, 13, 11, 22, 21, 19, 38, 37, 35,  
    70, 67 }; //0, 1, 2, 3, 4, 5, 6, 7, 8, 9,  
#, *  
    unsigned char num, bill;
```

```

PORTC = row;           //címzés          //kezdőcím: 0x08, első sor
_delay_ms(5);
bill = PINC & 0x7f;   //maszkolás 0b01111111
num = 0;               //tömbindex törlése

while( num<12 )
{
    if( bill == billtomb[num] ){
        m_button = num;
        while(PINC == billtomb[num]);
        break;
    }
    else{
        m_button = 12;           //12 ha nem nyomunk semmit
        num++;                  //következő tömbindex
    }
}
if( row == 0x40 )row = 0x08; //első sorra állítás
else row = row << 1;       //sorcímzés léptetése
}

void SEVSEG_putnumber_t()
{
    j = (j+1)%4;           //folyamatos kiíráshoz
    SEVSEG_putdigit(digit[j], j);
}

void beolvas_kiir()
{
    matrix();               //m.bill beolvasása
    if (m_button<=9)
    {
        i=1;
        while (i)
        {
            digit[i] = digit[i-1];
            i--;
        }

        digit[0] = m_button;
    }
    SEVSEG_putnumber_t();
}

void LED_out(unsigned char szam)
{
    PORTD = szam & 0xF0;
    PORTB = szam<<4;
}

void gomb_menu()
{
    gomb=PINC&0x1F;
    if(gomb && !(gomb_seged))
    {
        gomb_seged=1;
        switch(gomb)

```

```

{
    case 1: LED_out(gomb); digit[2]=digit[1]+digit[0]; break;
              //összeadás
    case 2: LED_out(gomb); break;                      //kivonás
    case 4: LED_out(gomb); break;                      //szorzás
    case 8: LED_out(gomb); break;                      //osztás
    case 16: LED_out(gomb); break;
default:      break;
};

}
if(!(gomb) && (gomb_seged)) gomb_seged=0;
}

```

## 1.2 Önálló feladat

1. Készítse el a számológép kivonás, szorzás, osztás részét is.
2. A beolvasáshoz megszakítást használjon.

## 2 Mátrix billentyűzet: 2jegyű számológép

### 2.1 Közös feladat: Számológép 2.

```
/*-----Mátrix billentyűzet számológép 2-----  
 *Feladat:           Készítsen 2jegyű számológépet  
 */  
  
#include <avr/io.h>  
#define F_CPU 16000000  
#include <util/delay.h>  
#include <avr/interrupt.h>  
  
void init();  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit);  
void beolvas_kiir();  
void LED_out(unsigned char szam);  
void gomb_menu();  
  
unsigned char m_button = 12;  
unsigned char row = 0x08;                                //1. sor "címe"  
  
unsigned char digit[4]={0};  
int j=0;  
int i=1;  
unsigned char gomb=0;  
unsigned char gomb_seged=0;  
int szam1=0;  
int szam2=0;  
  
int main(void)  
{  
    init();  
  
    while (1)  
    {  
        beolvas_kiir();  
        szam1=digit[1]*10+digit[0];                //kétjegyű számok előállítása  
        szam2=digit[3]*10+digit[2];  
        gomb_menu();  
    }  
}  
  
void init()  
{  
    DDRA=0xFF;                                         //7seg kimenetre állítás  
    DDRC = 0x78;                                         //mátrix  
    DDRB=DDRD=0xF0;  
}  
  
void SEVSEG_putdigit(unsigned char szam, unsigned char digit)  
{  
    if (digit>3) return;  
    if (szam>9) return;  
  
    PORTA=0x80 | (digit)<<4 | szam;    //kiiratás 0x80=enable;  
}
```

```

void matrix()
{
    const unsigned char billtomb[12] = { 69 , 14, 13, 11, 22, 21, 19, 38, 37, 35,
70, 67 };
                                            //0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
#,   *
    unsigned char num, bill;

    PORTC = row;                      //címzés           //kezdőcím: 0x08, első sor
    _delay_ms(5);
    bill = PINC & 0x7f;             //maszkolás 0b01111111
    num = 0;                         //tömbindex törlése

    while( num<12 )
    {
        if( bill == billtomb[num] ){          //tömb elemek ellenőrzése
            m_button = num;                  //ha egyezés van
            while(PINC == billtomb[num]);    //megfogjuk még nyomjuk
            break;                          //folyamat befejezése
        }
        else{                                //12 ha nem nyomunk semmit
            m_button = 12;                  //következő tömbindex
            num++;
        }
        if( row == 0x40 )row = 0x08;         //első sorra állítás
        else row = row << 1;               //sorcímzés léptetése
    }

    void SEVSEG_putnumber_t()
    {
        j = (j+1)%4;                     //folyamatos kiíráshoz
        SEVSEG_putdigit(digit[j], j);
    }

    void beolvas_kiir()
    {
        matrix();                        //m.bill beolvasása

        if (m_button<=9)
        {
            i=4;
            while (i)
            {
                digit[i] = digit[i-1];
                i--;
            }

            digit[0] = m_button;
        }

        SEVSEG_putnumber_t();
    }
}

void LED_out(unsigned char szam)
{
    PORTD = szam & 0xF0;
    PORTB = szam<<4;
}

```

```

void gomb_menu()
{
    gomb=PING&0x1F;
    if(gomb && !(gomb_seged))
    {
        gomb_seged=1;

        switch(gomb)
        {
            case 1: LED_out(szam1+szam2); break; //összeadás
            case 2: LED_out(gomb); break; //kivonás
            case 4: LED_out(gomb); break; //szorzás
            case 8: LED_out(gomb); break; //osztás
            case 16: LED_out(gomb); break;
            default: break;
        };
    }
    if(!(gomb) && (gomb_seged)) gomb_seged=0;
}

```

## 2.2 Önálló feladat

1. Készítse el a számológép kivonás, szorzás, osztás részét is.
2. A beolvasáshoz megszakítást használjon.

### 3 Mátrix billentyűzet: Kódzár

#### 3.1 Közös feladat: kódzár

```
-----Mátrix billentyűzet kódzár-----
*Feladat:          Készítsen kódzárat a mátrix billentyűzetre.
*                  Helyes kód esetén világítson a Zöld LED,
*                  helytelen kód esetén pedig ne.
*/

#include <avr/io.h>
#define F_CPU 16000000
#include <util/delay.h>
#include <avr/interrupt.h>

void init();
void SEVSEG_putdigit(unsigned char szam, unsigned char digit);
void beolvas_kiir();
void compare();

unsigned char m_button = 12;
unsigned char row = 0x08;           //1. sor "címe"

unsigned char digit[4]={0};
char kod[4]={4,3,2,1};            //1234 a kód
int j=0;
int i=1;
int seged=3;

int main(void)
{
    init();

    while (1)
    {
        beolvas_kiir();
        if (segéd==0)           //ha nem egyezik, kikapcsoljuk a zöld LED-et
        {
            PORTE&=(0<<PINE3);
        }
        else
        {
            PORTE=(1<<PINE3); //ha egyezik, bekapcsoljuk a zöld LED-et
        }
    }
}

void init()
{
    DDRA=0xFF;                   //7seg kimenetre állítás
    DDRC = 0x78;                 //mátrix
    DDRE=0x0C;                   //kék, zöld LED
}

void SEVSEG_putdigit(unsigned char szam, unsigned char digit)
{
    if (digit>3) return;
    if (szam>9) return;

    PORTA=0x80 | (digit)<<4 | szam; //kiíratás 0x80=enable;
}
```

```

void matrix()
{
    const unsigned char billtomb[12] = { 69 , 14, 13, 11, 22, 21, 19, 38, 37, 35,
70, 67 };
                                            //0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
#,   *
    unsigned char num, bill;

    PORTC = row;                      //címzés           //kezdőcím: 0x08, első sor
    _delay_ms(5);
    bill = PINC & 0x7f;             //maszkolás 0b01111111
    num = 0;                         //tömbindex törlése

    while( num<12 )
    {
        if( bill == billtomb[num] ){          //tömb elemek ellenőrzése
            m_button = num;                  //ha egyezés van
            while(PINC == billtomb[num]);    //megfogjuk még nyomjuk
            break;                          //folyamat befejezése
        }
        else{                                //12 ha nem nyomunk semmit
            m_button = 12;                  //következő tömbindex
            num++;
        }
        if( row == 0x40 )row = 0x08;        //első sorra állítás
        else row = row << 1;              //sorcímzés léptetése
    }

    void SEVSEG_putnumber_t()
    {
        j = (j+1)%4;                     //folyamatos kiíráshoz
        SEVSEG_putdigit(digit[j], j);
    }

    void beolvas_kiir()
    {
        matrix();                        //m.bill beolvasása

        if (m_button!=12)
        {
            switch(m_button)           /*-ra vagy #-re végünk összehasonlítást
            {
                case 10: compare(); break;
                case 11: compare(); break;
                default:      break;
            }
            if ((m_button!=10)&&(m_button!=11)) //eltároljuk a számokat
            {
                i=3;
                while (i)
                {
                    digit[i] = digit[i-1];
                    i--;
                }

                digit[0] = m_button;
            }
        }
        SEVSEG_putnumber_t();
    }
}

```

```

void compare() //összehasonlítjuk az utolsó 4 beolvasott
//számot az eltárolt értékekkel
{
    int i=0;
    seged=3;
    for (i=0; i<4; i++)
    {
        if (digit[i]!=kod[i]) //ha nem egyezik
        {
            seged=0;
            break;
        }
        else //ha egyezik
        {
            seged=1;
            break;
        }
    }
}

```

## 3.2 Önálló feladat

1. Oldja meg a kódzár feladatot úgy, hogy 2 helyes kód van: 345, 678. Az első esetében a kék LED világítson, a második esetében a zöld LED.

## 4 Gyakorló feladatok

1. Készítsen órát, melynek kezdőértékét a mátrix billentyűzetről tudja megadni.
2. Készítsen visszaszámlálót, mely 1s-onként számol és a kezdőértékét a mátrix billentyűzetről tudja megadni.
3. Egészítse ki a kódzár feladatot, hogy hibás kód esetében szólaljon meg a buzzer.
4. Készítse el az órát úgy, hogy mátrix billentyűzetről lehessen ébresztési időpontot megadni.  
Amennyiben az óra elérte azt az értéket, szólaljon meg a buzzer.

## 5. Assembly ZH-ra való felkészüléshez

1. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:

G0: Töltsé fel az r2 regiszter 12-vel, az r12 regisztert 3-mal! Abban az esetben, ha r2-ben levő érték maradék nélkül osztható r12-vel, akkor LED-ekre írjon 3 értéket, különben pedig 4 értéket!

G1: Töltsé fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően páros számokkal 0x10-től kezdődően!

G2: Adja össze az előző feladatrészben feltöltött terület első 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 2-es billentyűzetet tartalmazó oszlopának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 0. digitjére!

G4: A gomb lenyomásának idejére világítson a 3 színű LED-en a piros szín!

2. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:

G0: Töltsé fel az r3 regiszter 15-tel, az r4 regisztert 3-mal! Abban az esetben, ha r3-ban levő érték maradék nélkül osztható r4-gyel, akkor a hétszegmenses kijelző 1. digitjére írjon 3 értéket, különben pedig 4 értéket!

G1: Töltsé fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően páratlan számokkal 0x10-től kezdődően!

G2: Adja össze az előző feladatrészben feltöltött terület első 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 1-es billentyűzetet tartalmazó oszlopának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 1. digitjére!

G4: A gomb lenyomásának idejére világítson a 3 színű LED-en a zöld szín!

3. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:

G0: Töltsé fel az r10 regiszter 12-vel, az r12 regisztert 5-tel! Abban az esetben, ha r10-ben levő

érték 4-gyel osztható maradék nélkül és r12 páros, akkor LED-ekre írjon 5 értéket, különben pedig 6 értéket!

G1: Töltsé fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően páros számokkal 0xF0-től kezdődően visszafele!

G2: Adja össze az előző feladatrészben feltöltött terület első 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 3-as billentyűzetet tartalmazó oszlopának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 2. digitjére!

G4: A gomb lenyomásának idejére világítson a 3 színű LED-en a kék szín!

4. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:

G0: Tölts fel az r11 regiszter 32-vel, az r12 regisztert 4-gyel! Abban az esetben, ha r11-ben levő érték 8-cal osztható maradék nélkül és r12 páros, akkor a hétszegmenses kijelző 2. digitjére írjon 7 értéket, különben pedig 8 értéket!

G1: Tölts fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően páratlan számokkal 0xF0-tól kezdődően visszafele!

G2: Adja össze az előző feladatrészben feltöltött terület első 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 7-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!

G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a piros szín!

5. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:

G0: Tölts fel az r14 regiszter 32-vel, az r15 regisztert 4-gyel! Abban az esetben, ha r14-ben levő érték 2-vel osztható maradék nélkül és r12 páros, akkor a hétszegmenses kijelző 2. digitjére írjon 7 értéket, különben pedig 8 értéket!

G1: Tölts fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően számokkal 0xF0-tól kezdődően visszafele!

G2: Adja össze az előző feladatrészben feltöltött terület utolsó 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 4-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!

G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a zöld szín!

6. Készítsen menüt, amelyben a 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot a

mátrix billentyűzetén:

G0: Tölts fel az r2 regiszter 12-vel, az r12 regisztert 3-mal! Abban az esetben, ha r2-ben levő érték maradék nélkül osztható r12-vel, akkor LED-ekre írjon 3 értéket, különben pedig 4 értéket!

G1: Tölts fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően számokkal 0xF0-tól kezdődően visszafele!

G2: Adja össze az előző feladatrészben feltöltött terület utolsó 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 4-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!

G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a zöld szín!

7. Készítsen menüt, amelyben a 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot a mátrix billentyűzetén:

G0: Tölts fel az r3 regiszter 15-tel, az r4 regisztert 3-mal! Abban az esetben, ha r3-ban levő érték maradék nélkül osztható r4-gyel, akkor a hétszegmenses kijelző 1. digitjére írjon 3 értéket, különben pedig 4 értéket!

G1: Tölts fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően számokkal 0xF0-tól kezdődően visszafele!

G2: Adja össze az előző feladatrészben feltöltött terület utolsó 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 4-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!

G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a zöld szín!

8. Készítsen menüt, amelyben a 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot a mátrix billentyűzetén:

G0: Tölts fel az r10 regiszter 12-vel, az r12 regisztert 5-tel! Abban az esetben, ha r10-ben levő érték 4-gyel osztható maradék nélkül és r12 páros, akkor LED-ekre írjon 5 értéket, különben pedig 6 értéket!

G1: Tölts fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően számokkal 0x200-tól kezdődően visszafele!

G2: Adja össze az előző feladatrészben feltöltött terület utolsó 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 4-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!

G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a kék szín!

9. Készítsen menüt, amelyben a mátrix billentyűzet 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot:

G0: Tölts fel az r11 regiszter 32-vel, az r12 regisztert 8-cal! Abban az esetben, ha r11-ben levő

érték 8-cal osztható maradék nélkül és r12 páros, akkor a hétszegmenses kijelző 2. digitjére írjon 7 értéket, különben pedig 8 értéket!

G1: Tölts fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően számokkal 0xF0-tól kezdődően visszafele!

G2: Adja össze az előző feladatrészben feltöltött terület utolsó 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 7-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 2. digitjére!

G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a zöld szín!

10. Készítsen menüt, amelyben a 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot a mátrix billentyűzetén:

G0: Tölts fel az r14 regiszter 32-vel, az r15 regisztert 4-gyel! Abban az esetben, ha r14-ben levő érték 2-vel osztható maradék nélkül és r12 páros, akkor a hétszegmenses kijelző 2. digitjére írjon 7 értéket, különben pedig 8 értéket!

G1: Tölts fel az adatmemória 0x50 db bájtját a 0x200-as címtől kezdődően számokkal 0x200-

tól kezdődően visszafele!

G2: Adja össze az előző feladatrészben feltöltött terület utolsó 10 bájtján levő értéket szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 4-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!

G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a piros szín!

11. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:

G0: Tölts fel az r2 regiszter 12-vel, az r12 regisztert 3-mal! Abban az esetben, ha r2-ben levő érték maradék nélkül osztható r12-vel, akkor LED-ekre írjon 3 értéket, különben pedig 4 értéket!

G1: Adja össze 0x50 db páros szám értéket 0x10 értéktől kezdődően szubrutin segítségével, és

az összeget írassa ki a LED-ekre!

G2: Számolja össze 0x100 és 0x200 tartományban hány darab 8-cal maradék nélkül osztható szám van szubrutin segítségével, és az összeget írassa ki a hétszegmenses kijelző 0. digitjére!

G3: Olvassa be a billentyűzet mátrix 2-es billentyűzetet tartalmazó oszlopának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 0. digitjére!

G4: A gomb lenyomásának idejére világítson a 3 színű LED-en a piros szín!

12. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:

G0: Tölts fel az r3 regiszter 15-tel, az r4 regisztert 3-mal! Abban az esetben, ha r3-ban levő érték maradék nélkül osztható r4-gyel, akkor a hétszegmenses kijelző 1. digitjére írjon 3 értéket, különben pedig 4 értéket!

G1: Számolja össze 0x120 és 0x210 tartományban hány darab 4-gyel maradék nélkül osztható szám van szubrutin segítségével, és az összeget írassa ki a hétszegmenses kijelző 1. digitjére!

G2: Adja össze 0x100 db páratlan szám értéket 0xFF értéktől kezdődően csökkenő értékekkel szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 1-es billentyűzetet tartalmazó oszlopának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 1. digitjére!

G4: A gomb lenyomásának idejére világítson a 3 színű LED-en a zöld szín!

13. Készítsen menüt, amelyben a G0-G4-ig levő gombokkal választhatja a megvalósítandó feladatot:

G0: Töltsé fel az r10 regiszter 12-vel, az r12 regisztert 5-tel! Abban az esetben, ha r10-ben levő

érték 4-gyel osztható maradék nélkül és r12 páros, akkor LED-ekre írjon 5 értéket, különben pedig 6 értéket!

G1: Számolja össze 0x210 és 0x120 tartományban hány darab 16-tal maradék nélkül osztható szám van szubrutin segítségével, és az összeget írassa ki a hétszegmenses kijelző 2. digitjére!

G2: Adja össze 0x100 db páros szám értéket 0xFF értéktől kezdődően csökkenő értékekkel szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 3-as billentyűzetet tartalmazó oszlopának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 2. digitjére!

G4: A gomb lenyomásának idejére világítson a 3 színű LED-en a kék szín!

14. Készítsen menüt, amelyben a mátrix billentyűzet 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot:

G0: Töltsé fel az r11 regiszter 32-vel, az r12 regisztert 8-cal! Abban az esetben, ha r11-ben levő

érték 8-cal osztható maradék nélkül és r12 páros, akkor a hétszegmenses kijelző 2. digitjére írjon 7 értéket, különben pedig 8 értéket!

G1: Számolja össze 0x210 és 0x120 tartományban hány darab 32-vel maradék nélkül osztható szám van szubrutin segítségével, és az összeget írassa ki a hétszegmenses kijelző 3. digitjére!

G2: Adja össze 0x100 db páratlan szám értéket 0x10 értéktől kezdődően növekvő értékekkel szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 7-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 2. digitjére!

G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a zöld szín!

15. Készítsen menüt, amelyben a 0-4-ig levő gombokkal választhatja a megvalósítandó feladatot a

mátrix billentyűzeten:

G0: Töltsé fel az r14 regiszter 32-vel, az r15 regisztert 4-gyel! Abban az esetben, ha r14-ben levő érték 2-vel osztható maradék nélkül és r12 páros, akkor a hétszegmenses kijelző 2. digitjére írjon 7 értéket, különben pedig 8 értéket!

G1: Számolja össze szubrutin segítségével a 0x210 és 0x120 tartományban hány darab olyan szám van, amelyik esetén a 2. és a 3. biten 1 érték van, és az összeget írassa ki a hétszegmenses kijelző 0. digitjére!

G2: Adja össze 0x100 db páratlan szám értéket 0x10 értéktől kezdődően csökkenő értékekkel szubrutin segítségével, és az összeget írassa ki a LED-ekre!

G3: Olvassa be a billentyűzet mátrix 4-es billentyűzetet tartalmazó sorának billentyűjének bármelyikét, és írja ki a hétszegmenses kijelző 3. digitjére!

G4: A gomb lenyomásának idejére villogjon a 3 színű LED-en a piros szín!

## 6. AVR C ZH-ra való felkészüléshez

1. Készítsen két műveletes számológépet, amely 1, 4, 7 számokkal végez összeadást vagy szorzást, ahol az összeadás műveletét # jel, a szorzás műveletét a \* szolgálja. Az operandusok bevitelére használja a mátrix billentyűzetet. Az első operandus a 3. digiten a műveleti jel a LED-eken, a második operandus a 2. digiten, az eredmény a 0. és 1. digiteken jelenjen meg a hétszegmenses kijelzőn:

G0: Első operandus bevitеле (bevitelre csak 1,4,7 billentyű használható).

G1: Műveleti jel bevitеле (bevitelre csak a \*, # billentyűk használhatók).

G2: Második operandus bevitеле (bevitelre csak 1,4,7 billentyű használható).

G3: Eredmény megjelenítése vezető nulla kikapcsolásával.

G4: A gomb lenyomásának idejére villogjon 1 másodperces ütemben a 3 színű LED-en a piros szín!

2. Készítsen két műveletes számológépet, amely 2, 5, 8 számokkal végez kivonást vagy szorzást,

ahol a kivonás műveletét # jel, a szorzás műveletét a \* szolgálja. Az operanduszok bevitelére használja a mátrix billentyűzetet. Az első operandus a 3. digiten a műveleti jel a LED-eken, a második operandus a 2. digiten, az eredmény a 0. és 1. digiteken jelenjen meg a hétszegmenses kijelzőn:

G0: Első operandus bevitеле (bevitelre csak 2,5,8 billentyű használható).

G1: Műveleti jel bevitеле (bevitelre csak a \*, # billentyűk használhatók).

G2: Második operandus bevitеле (bevitelre csak 2,5,8 billentyű használható).

G3: Eredmény megjelenítése vezető nulla kikapcsolásával.

G4: A gomb lenyomásának idejére villogjon 1 másodperces ütemben a 3 színű LED-en a zöld szín!

3. Készítsen két műveletes számológépet, amely 3, 6, 9 számokkal végez összeadást vagy kivonást, ahol az összeadás műveletét # jel, a kivonás műveletét a \* szolgálja. Az operanduszok bevitelére használja a mátrix billentyűzetet. Az első operandus a 3. digiten a műveleti jel a LED-eken, a második operandus a 2. digiten, az eredmény a 0. és 1. digiteken jelenjen meg a hétszegmenses kijelzőn:

G0: Első operandus bevitеле (bevitelre csak 3,6,9 billentyű használható).

G1: Műveleti jel bevitеле (bevitelre csak a \*, # billentyűk használhatók).

G2: Második operandus bevitеле (bevitelre csak 3,6,9 billentyű használható).

G3: Eredmény megjelenítése vezető nulla kikapcsolásával.

G4: A gomb lenyomásának idejére villogjon 1 másodperces ütemben a 3 színű LED-en a kék szín!

4. Készítsen összeadót vagy szorzót, amely csak a 1, 4, 7 és a 3, 6, 9 számjegyekből álló 3 jegyű

számmal végez összeadást. Az operanduszok bevitelére használja a mátrix billentyűzetet. Az első operandus, a második operandus és az eredmény jelenjen meg a hétszegmenses kijelzőn: G0: Első 2 jegyű operandus bevitеле (bevitelre csak 3, 6, 9, 1, 4, 7 billentyűk használható).

G1: Műveleti jel bevitеле (bevitelre csak a \*, # billentyűk használhatók).

G2: Második 2 jegyű operandus bevitеле (bevitelre csak 3, 6, 9, 1, 4, 7 billentyűk használható).

G3: Eredmény megjelenítése vezető nulla kikapcsolásával.

G4: A gomb lenyomásának idejére villogjon 1 másodperces ütemben a 3 színű LED-en a kék szín!

5. Készítsen összeadót vagy kivonót, amely csak a 1, 4, 7 és a 3, 6, 9 számjegyekből álló 3 jegyű

számmal végez összeadást. Az operanduszok bevitelére használja a mátrix billentyűzetet. Az első operandus, a második operandus és az eredmény jelenjen meg a hétszegmenses kijelzőn: G0: Első 3 jegyű operandus bevitеле (bevitelre csak 3, 6, 9, 1, 4, 7 billentyűk használható).

G1: Műveleti jel bevitеле (bevitelre csak a \*, # billentyűk használhatók).

G2: Második 3 jegyű operandus bevitеле (bevitelre csak 3, 6, 9, 1, 4, 7 billentyűk használható).

G3: Eredmény megjelenítése vezető nulla kikapcsolásával.

G4: A gomb lenyomásának idejére villogjon 1 másodperces ütemben a 3 színű LED-en a piros szín!

6. Készítsen összeadót vagy kivonót, amely csak a 1, 7 és a 6, számjegyekből álló 3 jegyű számmal

végez összeadást. Az operanduszok bevitelére használja a mátrix billentyűzetet. Az első operandus, a második operandus és az eredmény jelenjen meg a hétszegmenses kijelzőn:

G0: Első 3 jegyű operandus bevitеле (bevitelre csak 1, 6, 7 billentyűk használható).

G1: Műveleti jel bevitеле (bevitelre csak a \*, # billentyűk használhatók).

G2: Második 3 jegyű operandus bevitеле (bevitelre csak 1, 6, 7 billentyűk használható).

G3: Eredmény megjelenítése vezető nulla kikapcsolásával.

G4: A gomb lenyomásának idejére villogjon 1 másodperces ütemben a 3 színű LED-en a zöld szín!

7. Készítsen órát, amely a hétszegmenses kijelzőn a digit 0-án és digit 1-en a perct, a digit 3-on

és digit 2-ön az órát mutatja. A másodperc mutatása a digit 2 és digit 1 közötti két LED-en történjen:

2-es osztályzat: Óra és perc kijelzése a hétszegmenses kijelzőn, az óra időzítéséhez késleltető függvényt használhat.

3-as osztályzat: Az előző feladatrész mellett a másodperc kijelzés megjelenítése.

4-es osztályzat: Óra és perc kijelzése a hétszegmenses kijelzőn, az óra időzítéséhez timer interruptot használjon!

5-ös osztályzat: Az előző feladatrész mellett a másodperc kijelzés megjelenítése.

8. Készítsen órát, amely a hétszegmenses kijelzőn a digit 0-án és digit 1-en a másodpercet, a digit 3-on és digit 2-ön a percet mutatja. A másodperc mutatása a digit 2 és digit 1 közötti két LED-en  
is történjen:

2-es osztályzat: Másodperc és perc kijelzése a hétszegmenses kijelzőn, az óra időzítéséhez késleltető függvényt használhat.

3-as osztályzat: Az előző feladatrész mellett a másodperc kijelzés megjelenítése a digitek közötti LED-eken is.

4-es osztályzat: Másodperc és perc kijelzése a hétszegmenses kijelzőn, az óra időzítéséhez timer interruptot használjon!

5-ös osztályzat: Az előző feladatrész mellett a másodperc kijelzés megjelenítése a digitek közötti LED-eken.

9. Készítsen számrendszerek közötti átszámítót, amely 2, 4, 8 és 10 számrendszerek között átszámítja a billentyűzet mátrixon bevitt és a hétszegmenses kijelzőn megjelenített decimális számértéket:

G0: 2 digites decimális szám bevitelle a mátrix billentyűn és megjelenítése a digit 0 és digit 1 kijelzőn.

G1: a bevitt érték oktális megjelenítése a hétszegmenses kijelzőn.

G2: a bevitt érték 4-es számrendszerbeli megjelenítése a hétszegmenses kijelzőn.

G3: a bevitt érték felső bájtjának bináris megjelenítése a LED-eken.

G4: a bevitt érték alsó bájtjának bináris megjelenítése a LED-eken.

10. Készítsen számrendszerek közötti átszámítót, amely 2, 4, 8 és 10 számrendszerek között átszámítja a billentyűzet mátrixon bevitt és a hétszegmenses kijelzőn megjelenített oktális számértéket:

G0: 2 digites oktális szám bevitelle a mátrix billentyűn és megjelenítése a digit 0 és digit 1 kijelzőn.

G1: a bevitt érték decimális megjelenítése a hétszegmenses kijelzőn.

G2: a bevitt érték 4-es számrendszerbeli megjelenítése a hétszegmenses kijelzőn.

G3: a bevitt érték felső digitjének bináris megjelenítése a LED-eken.

G4: a bevitt érték alsó digitjének bináris megjelenítése a LED-eken.

11. Készítsen reakció időmérőt, amely G0-ás gomb hatására bekapcsolja a LED0-át, és elindítja a hétszegmenses kijelzőn a századmásodperc futását, majd a G1 gomb hatására ezt leállítja a számlálást:

2-es osztályzat: az időmérő megoldása késleltető függvény segítségével.

3-as osztályzat: az előző feladat megoldása, 5-ször ismételt mérés átlagát kijelezve a hétszegmenses kijelzőn.

4-es osztályzat: az időmérő megoldása timer interrupt segítségével történjen.

5-ös osztályzat: az előző feladat megoldása, 5-ször ismételt mérés átlagát kijelezve a hétszegmenses kijelzőn.

12. Készítsen gyakoriság számlálót, amely a mátrix billentyűn bevitt 10 db számjegy közül kiírja a leggyakrabban bevitt számjegyet, illetve annak a gyakoriságát:
- 2-es osztályzat: 10 db számjegy bevitelre a mátrix billentyűn, majd G0 hatására a digit 2-ön a leggyakoribb szám megjelenítése.
- 3-as osztályzat: az előző feladat megoldása, és digit 0-án és digit 1-en a leggyakrabban előforduló számjegy gyakoriságának kiíratása.
- 4-es osztályzat: az előző feladat megoldása, a bevitt számjegyek átlagát kijelezve a hétszegmenses kijelzőn.
- 5-ös osztályzat: az előző feladat megoldása, de az időzítések megoldása timer interrupt segítségével történjen.

## Megszakítás vektortábla

Vector No.	Program Address <sup>(2)</sup>	Source	Interrupt Definition
1	\$0000 <sup>(1)</sup>	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$0002	INT0	External Interrupt Request 0
3	\$0004	INT1	External Interrupt Request 1
4	\$0006	INT2	External Interrupt Request 2
5	\$0008	INT3	External Interrupt Request 3
6	\$000A	INT4	External Interrupt Request 4
7	\$000C	INT5	External Interrupt Request 5
8	\$000E	INT6	External Interrupt Request 6
9	\$0010	INT7	External Interrupt Request 7
10	\$0012	TIMER2 COMP	Timer/Counter2 Compare Match
11	\$0014	TIMER2 OVF	Timer/Counter2 Overflow
12	\$0016	TIMER1 CAPT	Timer/Counter1 Capture Event
13	\$0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
14	\$001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
15	\$001C	TIMER1 OVF	Timer/Counter1 Overflow
16	\$001E	TIMER0 COMP	Timer/Counter0 Compare Match
17	\$0020	TIMER0 OVF	Timer/Counter0 Overflow
18	\$0022	SPI, STC	SPI Serial Transfer Complete
19	\$0024	USART0, RX	USART0, Rx Complete
20	\$0026	USART0, UDRE	USART0 Data Register Empty
21	\$0028	USART0, TX	USART0, Tx Complete
22	\$002A	ADC	ADC Conversion Complete
23	\$002C	EE READY	EEPROM Ready
24	\$002E	ANALOG COMP	Analog Comparator
25	\$0030 <sup>(3)</sup>	TIMER1 COMPC	Timer/Counter1 Compare Match C
26	\$0032 <sup>(3)</sup>	TIMER3 CAPT	Timer/Counter3 Capture Event
27	\$0034 <sup>(3)</sup>	TIMER3 COMPA	Timer/Counter3 Compare Match A
28	\$0036 <sup>(3)</sup>	TIMER3 COMPB	Timer/Counter3 Compare Match B
29	\$0038 <sup>(3)</sup>	TIMER3 COMPC	Timer/Counter3 Compare Match C
30	\$003A <sup>(3)</sup>	TIMER3 OVF	Timer/Counter3 Overflow
31	\$003C <sup>(3)</sup>	USART1, RX	USART1, Rx Complete
32	\$003E <sup>(3)</sup>	USART1, UDRE	USART1 Data Register Empty
33	\$0040 <sup>(3)</sup>	USART1, TX	USART1, Tx Complete
34	\$0042 <sup>(3)</sup>	TWI	Two-wire Serial Interface
35	\$0044 <sup>(3)</sup>	SPM READY	Store Program Memory Ready