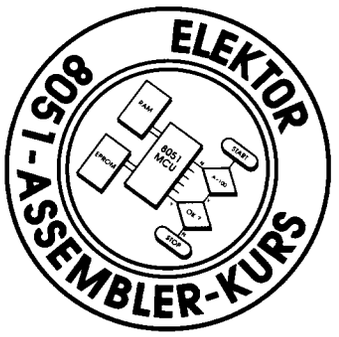
8051-es mikrokontroller és assembler-tanfolyam

4. rész: Aritmetikai utasítások



A tanfolyamnak ebben a részében végre lezárjuk a 8051-es utasításainak tárgyalását. Előttünk állnak még az aritmetikai utasítások. Velük együtt megbeszéljük a programozási technikákat is, hogy assembler-szinten egyszerű programokat tudjunk írni.

Ismereteinket egy kapacitás meghatározására, illetve idő mérésére szolgáló példaprogramban fogjuk majd hasznosítani. Sőt, már egy digitális zajgenerátor programozását is látni fogjuk.

Összeadás

Két nyolcbites szó (Byte) összeadására a 8051-es a következő összeadási utasítással rendelkezik:

ADD A,BYTE–OPERAND; Add hozzá BYTE-OPERAND-ot az akkumulátor tartalmához

Amilyen egyszerűen hangzik, olyan ravaszul rejtőzködik az ördög a részletekben, nevezetesen az utasítás hatására bekövetkező Flag-beállításokban. Az egyes Flag-ekre vonatkozó követelmények a következők: A CY átviteli bit (melyet gyakran C-nek is neveznek) akkor kerül beírásra ("1"), ha a 7. bitpozícióban átvitel keletkezik, egyébként nullázásra kerül. Ezért a C-Flag pontosan mutatja a 0–255-ig terjedő, előjel nélküli számok összeadása során keletkezett átvitelt.

A BCD összeadás AC (Auxiliary Carry) segédflag-je akkor kerül beírásra, ha a 3. bitpozícióban keletkezik átvitel, egyébként nulla értéket kap.

Ezt a Flag-et a DA A utasítás (lásd később) használja fel.

Az OV túlcsordulási (Overflow) Flag akkor kerül beírásra, ha a 6. bitpozícióban keletkezik, de a 7. pozícióban nem keletkezik átvitel. Beírásra kerül akkor is, ha a 7. pozícióban keletkezik, a 6. pozícióban viszont nem keletkezik átvitel.

Egyéb esetekben ez a Flag nullázásra kerül. Ezúttal megállapítható a –128-tól 127-ig terjedő előjeles számok összeadása során keletkezett túlcsordulás.

A Flag-ek jelentése tehát attól függ, hogyan interpretáljuk egy Byte tartalmát (előjel nélküli, előjeles vagy BCD számként). Tegyünk ezért egy rövid kirándulást a számok ábrázolásának területére.

Számok ábrázolása

1. Előjel nélküli számok

A Byte-nak előjel nélküli számként való értelmezése megfelel egy szám közismert bináris írásmódjának. Itt a bitekhez a 0. bittel kezdődően 1, 2, 4, 8, 16 stb. értéket rendelünk hozzá és az ábrázolt szám a következőképpen értelmezendő:

128 * 7-es bit + 64 * 6-os bit + ... + 4 * 2-es bit + 2 * 1-es bit + 1 * 0-ás bit

Így egyetlen Byte segítségével a 0-tól 255-ig terjedő számok ábrázolhatók.

2. BCD számok

A számok BCD (Binary Coded Decimal) ábrázolásában a Byte bal oldali és jobb oldali 4-4 bitje (az úgynevezett Nibble-ek) egy-egy binárisan kódolt egyjegyű decimális szám leírására szolgálnak.

A számok ilyen ábrázolásának az előnye az, hogy egyszerűen előállítható. Hátránya azonban, hogy az összeadáshoz és a kivonáshoz az egyszerű bináris összeadás, illetve kivonás önmagában nem használható, csupán további korrekciókkal. Ezeket a korrekciókat a DA A összeadási utasítás (lásd később) az AC Flag segítségével tudja elvégezni.

3. Előjeles számok

Több alkalmazáshoz negatív számok is szükségesek. A –128-tól 127-ig terjedő számtartomány ábrázolása céljából a következőképpen járhatunk el: Ha az ábrázolandó x szám pozitív vagy nulla, akkor a Byte-ba egyszerűen, előjel nélkül írjuk be. Ha negatív (és 128-nál kisebb abszolút értékű), akkor a 256 + x számot (amely már 128 és 255 közé esik) írjuk be előjel nélkül a Byte-ba. Ez a számábrázolás pontosan akkor ír "1"-et a 7-es bitbe, ha a szám negatív. Negatív szám ábrázolásakor tehát először az összeget kell képezni, majd az új (most már pozitív) számot negatív előjellel kell ellátni. Az előjeles számokat szorzás és osztás esetén is gyakran külön kell kezelni.

Most néhány példa kapcsán bemutatjuk, hogy hogyan jelenthet ugyanaz a Byte-on belüli bitminta különböző számokat:

A számok ábrázolása terén tett kirándulásunk alapján most már megbeszélhetjük a következő számítási példákat:

Példák:

Bináris	Hexadec.	előjel nélküli	előjeles	BCD szám
00000000	00	0	0	0
00000001	01	1	1	1
00000010	02	2	2	2
00010001	11	17	17	11
00100101	25	37	37	25
11111111	FF	255	–1	nem definiált
10000000	80	128	–128	80
011111111	7F	127	127	nem definiált

4. Nagy számok

Sok esetben a megkövetelt pontosságból adódóan szükséges számértékek ábrázolásához egyetlen Byte nem elegendő. Ilyenkor több Byte-ot szoktunk használni, például 2 Byte-ot 16 bites szóvá vagy 5 Byte-ot 32 bites szóvá fogunk össze. A memóriában való elhelyezés során az összetartozó Byte-ok egymást követő rekeszekben tárolhatók.

Tanfolyamunk során mindig a legkisebb helyértékű (tehát a legkisebb című) Byte-ot tároljuk elsőként. Az egymást követő Byte-ok itt is előjel nélküli, előjeles vagy BCD számokként foghatók fel. Ha a számok ábrázolására 2 Byte-ot használunk (ami a mikrokontroller alkalmazásoknál gyakran elegendő), akkor a következő számtartományok ábrázolhatók:

A 16-bites szavak előjel nélkül: 0-tól 65535-ig előjellel: – 32768-tól 32767-ig BCD: 0-tól 9999-ig terjedő számtartományt fednek le.

Néhány példa:

A HEX Byte címe m + 1 m		HEX	előjel előjeles BCD nélküli		BCD
00	00	0000	0	0	0
12	34	1234	4660	4660	1234
0A	BC	0ABC	2748	2748	nem definiált
80	00	8000	32768	-2768	8000
FF	FF	FFFF	65535	-1	nem definiált

Összeadás átvitellel

A 16 vagy 32 bites számok összeadása során az a probléma, hogy az egyik Byte-pozícióból keletkező átvitelt a következő Byte hozzáadása során tekintetbe kell venni. Erre a következő utasítás szolgál:

ADD A,BYTE-OPERAND; Add össze OPERAND + CY-t A-val

A gép a megadott operandus mellett a CY Flag-et is hozzáadja az akkumulátor tartalmához. Pl. A belső RAM-ban ZAHL1 cím alatt álló 16 bites számnak a ZAHL2 számhoz való hozzáadását a következők szerint kell programozni (mindkét szám előjel nélküli):

```
MOV  A,ZAHL1;     vidd a ZAHL1 kisebb helyértékű
                  Byte-ját az akkumulátorba
ADD  A,ZAHL2;     add hozzá a ZAHL2 kisebb
                  helyértékű Byte-ját
MOV  ZAHL1,A;     tárold az eredményt ZAHL1
                  kisebb helyiértékű Byte-ján
MOV  A,ZAHL1+1;   vidd a szám következő Byte-ját
                  az akkumulátorba
MOV A,ZAHL2+1;    add hozzá a következő Byte-ot
                  az előző átvitellel együtt
MOV  ZAHL1+1,A;   eredmény tárolása a ZAHL1
                  nagyobb helyiértékű Byte-ján.
```

BCD korrekció

Két BCD szám összeadása után, hogy az akkumulátorban megint BCD számot kapjunk, közvetlenül az összeadás után a

```
DA A ; BCD korrekció
```

utasítást kell végrehajtani. Az utasítás a következőket váltja ki:

Ha 4 kisebb helyértékű bit (alsó Nybble) az akkumulátorban 9-nél nagyobb, vagy az AC (Auxiliary Carry Flag) be van írva, akkor az akkumulátorhoz 6 kerül hozzáadásra. Ezáltal a kis helyértékű BCD pozícióban már a helyes szám adódik ki. Ha ez a korrekciós összeadás olyan átvitelt eredményez, mely valamennyi nagyobb helyértékű bitpozíción végigtolódik, akkor sor kerül a CY Flag beírására.

Ezután annak tesztelése következik, hogy a CY Flag be van-e írva, vagy a nagyobb helyértékű 4 bit (felső Nybble) értéke a 9-et meghaladja-e. Ha igen, akkor most a 6-os érték a nagyobb helyértékű bitekhez kerül hozzáadásra. Ez adja ki a felső Nybble-hez tartozó BCD számot. Amennyiben itt átvitel lép fel, úgy a CY átviteli Flag beírásra kerül. Ezáltal a CY Flag a DA utasítás után azt jelzi, hogy a megelőző ADD, illetve ADDC utasítás hatására végzett összeadás során 99-nél nagyobb eredmény keletkezett.

Ennek működését a következő példával világítjuk meg:

Egy DA A utasítás után a CY Flag itt is az átvitelt jelzi, BCD-ben kifejezve ez a 100-as értéknek felel meg.

Kivonás

Kivonásra csak egy utasítás létezik, mégpedig a

```
SUBB A,BYTE-OPERAND;
vond ki az OPERANDUSt
és az átvitelt az AKKU-ból
```

Ennek az utasításnak az eredményeként a megadott operandus levonódik az akkumulátor tartalmából. Ezt követően még az átvitel-bit (Carry, azaz CY, illetve C) kivonására is sor kerül. Ha ezt meg akarjuk akadályozni, akkor előbb a C flag-et a CLR C utasítás útján explicite nullázni kell.

Összehasonlítások

A 8051-es kivonási utasításának van még egy fontos alkalmazása, ugyanis két érték egymással való összehasonlítására is felhasználható. Erre a következő tény nyújt lehetőséget: Ha egy x számból egy y számot kivonunk, akkor átvitel pontosan abban az esetben keletkezik, ha y nagyobb x-nél. Annak tesztelése céljából tehát, hogy az y szám nagyobbe az x számnál, egyszerűen ki kell vonni x-ből az y-t, majd meg kell vizsgálni a CY Flag-et. A programban ez a következőképpen alakul (mindkét számot előjel nélkülinek tekintve):

```
; 99 és 11 összeadása
  MOV  A,#11H ; 11-es BCD szám az
                akkuba              00010001B 11
  ADD  A,#99H ; +99-es BCD szám:  + 10011001B 99
;             ; nem BCD számot
                eredményez        = 10101010B AA
  DA   A      ; eredménye           00010000B +
                                    átvitel
;             ; ennek BCD meg-
                felelője          1 100010000=110
```

```
MOV  A,x          ; hozd elő a belső RAM-ból
                    az x számot
CLR  C            ; nullázd az átvitelt
SUBB A, y         ; képezd az x-y különbséget
JC   NAGYOBB      ; Ha y > x, ugorj a NAGYOBB-ra
```

Az előjeles számok összehasonlítása kissé körülményesebben végezhető csak el.

Szorzás és osztás

A 8051-es egy B-nek nevezett segédakkumulátorral rendelkezik. Ennek az SFR címe OFOH és előjel nélküli 8 bites számok szorzására és osztására használatos.

Erre szolgál a következő két utasítás:

```
MUL  AB   ; képezd A és B szorzatát,
            A lesz a kis helyértékű rész
DIV  AB   ; képezd az A/B osztást;
            A lesz a hányados, B pedig a
            maradék
```

Az első utasítás az A és B akkumulátorok tartalmát szorozza össze és az eredményként kiadódó 16 bites számot az A és a B akkumulátorban tárolja. A szorzat kis helyértékű Byte-ja az A akkumulátorban a nagy helyértékű Byte a B akkumulátorban kerül tárolásra. Ha az eredmény 255-nél nagyobb, akkor sor kerül az OV Flag beírására.

Az osztási utasítás A-t B-vel osztja. A hányados egész része az osztás után az A akkumulátorba kerül. A keletkezett maradék tárolása a B akkumulátorban történik.

A szorzási és az osztási utasítás nem nagy teljesítőképességű, mert sem 16 bites számok szorzatát, sem 16 bites szám 8 bites számmal való osztását nem lehet vele elvégezni. A 8051-est követő modellek (pl. a Siemens 80537-ese) már hatékonyabb utasításokkal rendelkeznek.

A 8051 aritmetikai gyengéinek némi enyhítése céljából az EMON51-be olyan segédprogramok vannak beépítve, melyek 16x16 bites szorzat 32 bites eredményének meghatározására alkalmasak. A tanfolyam diszkettjén található listában (EMON51.LST) utána lehet nézni, hogy hogyan is megy végbe egy ilyen szorzás.

A monitor 32 bites szám 16 bites számmal való osztásának elvégzésére is képes. Ebből a célból a 16 bites és 32 bites értékekkel úgyszólván „írásban" történik a számolás. Akit érdekel, a listában ennek is utánanézhet.

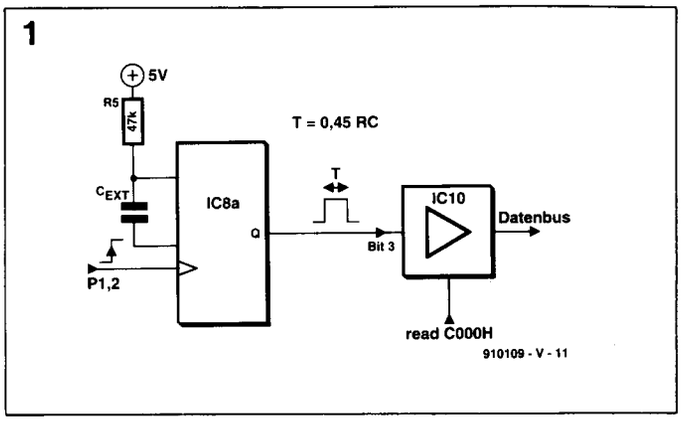
Kapacitásmérés V24-es kimenettel

A számítási lehetőségek imént lezárt megbeszélése után ezeket a lehetőségeket rögtön fel is kívánjuk használni egy olyan egyszerű kapacitásmérő építéséhez, mely a kapacitás értékét a V24-es interfész útján adja ki. Ily módon természetesen nemcsak kapacitások mérhetők, hanem azonos elv szerint ellenállás- és kapacitásmérés is végezhető.

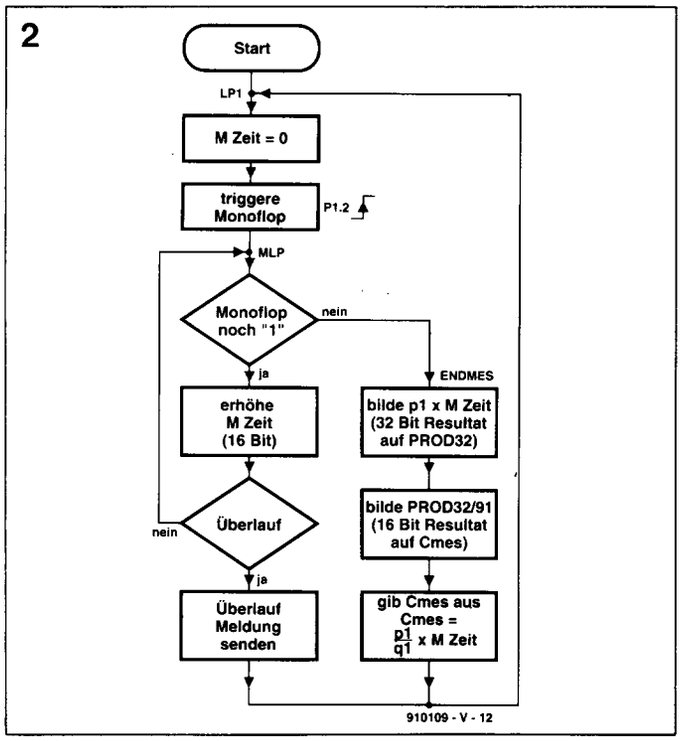
A méréshez használt (a bővítőkártyán elhelyezett) hardvert sematikusan az *1. ábrán* mutatjuk be. A program folyamatábrája a *2. ábrán* található. Tulajdonképpen az egész igen egyszerű: Először az IC8A monoflop kerül szoftver úton triggerelésre. Ezután a szoftver a monoflop kimenetén megméri az impulzus időtartamát. Ebből az időtartamból történik a kapacitásérték kiszámítása, ami végül a kimenetre kerül. Ám mint mindig, az ördög most is a részletekben búvik meg!

1. Időmérés

Az idő mérésére azt a 16 bites számot (MZEIT) használjuk, mellyel a monoflop impulzusának időtartamát számláljuk. A mérés kezdetén (Ip1 címre) ezt az értéket 0-ra állítjuk (20. és 21. sor a 3. ábrán). Ezután a P1.2 port-bit útján indítóimpulzus (pozitív homlok) kerül a monoflopra. (Az Ipp címkéjű várakozóhurok ezt a startimpulzust meghosszabbítja, hogy a monoflopnak újabb triggerelés előtt némi ideje maradjon a regenerálódásra). Ezután kezdődik az időmérőhurok (MLP címke). A OCOO-OH RAM-cím 3. bitje alapján kerül megállapításra, hogy a monoflop kap-e még triggerelést. Ha igen, akkor az MZEIT-ben található érték eggyel megnövekszik. Ha nem, akkor elértünk a mérés végére és ugrás következik az ENDMES címkére. Az MZEIT 16 bites érték növelése a 29-estől 34-esig terjedő sorokban történik és a 16 (illetve 32) bites változókkal való számolás szoká-



1. ábra. A kapacitásmérő sematikus felépítése



2. ábra. A kapacitásmérő program folyamatábrája

1. ábra.
- Datenbus = adatbusz

- **2. ábra.**
- triggere Monoflop = triggerezd a monoflopot
- Monoflop noch „1" = monoflop még „1"
- nein = nem
- ja = igen
- erhöhe M Zeit = növeld az M időt
- Überlauf = túlcsordulás
- Meldung senden = jelzést küldeni
- bilde p1 x M Zeit (32 Bit Resultat auf PROD32) = képezd a p1 x M időt (a 32 bites eredményt rakd PROD32-re)
- bilde PROD32/91 (16 Bit Resultat auf Cmes) = képezd PROD32/91-et (a 16 bites eredményt rakd Cmes-ra)
- gib Cmes aus = add ki Cmes-t
- Cmes = $\frac{p1}{q1}$ x M Zeit = Cmes = $\frac{p1}{q1}$ x M idő

sos módszerét szemlélteti. A kis helyértékű Byte-tal kezdve a számítás „írásban" történik és minden esetben figyelembe veszi az átvitelt. Az MZEIT érték növelése után megint a hurok kezdetére térünk vissza.

2. Túlcsordulás felismerése

Előfordulhat, hogy az időmérő hurkon belüli számlálása során a 16 bites számra megengedett értéktartományt túllépjük. Ez az az eset, amikor a műszerre túl nagy kapacitású kondenzátort kapcsolunk.

A túlcsordulás a keletkezett átvitel alapján a 35-ös sorban szereplő utasítással állapítható meg. Túlcsordulás esetén a V24 útján megfelelő jelzés kerül kiadásra (36-os és 37-es sorok).

3. Átszámítás

A kondenzátor mérése során kapott értékeket természetesen a szokásos mennyiségben kívánjuk kiadni. Ehhez a mérőhurkon történő átfutások számából meg kell határozni a monoflop impulzus időtartamát. Mivel a hurok lefutása minden alkalommal 12 mikroszekundum hosszúságú, az impulzus időtartama a hurok végén éppen T=12*MWERT mikroszekundum. (Ez az adat nem egészen pontos, mert a hurokba való belépésig már 6 mikroszekundum eltelik a hiba azonban a mi céljaink esetében elhanyagolható.)

A gyártó adatai szerint 74HC123 használata esetén a monoflopnál az impulzus tartama T=0,45 · Rext · Cext (10 nF feletti Cext esetén; az értékek ohmban, szekundumban, farádban).

A bővítőkártyán megépített kapcsolásunkban R értéke 47k és így:
C [mikrofarádban]· = 0,5673 MWERT

Hogyan tudjuk ezt az értéket kiszámítani? A 8051-es alkalmazásakor lebegőpontos aritmetika sajnos nem áll rendelkezésre. Minden Forth-programozó tudja azonban, hogy ilyenkor valódi törtekkel kell számolni, azaz a 0,5673=5673/10000 alakot használjuk.

E törtnek a (programunkban p1-gyel és q1-gyel jelölt) számlálóját és nevezőjét 16 bites számokként kezeljük. A 0,5673-mal való szorzás céljából az MWERT-et először a számlálóval szorozzuk (39-től 52-ig terjedő sorok), és a kapott (a PROD32-nél kezdődő 4 Byte-ban tárolt) 32 bites számot ezután a nevezővel osztjuk (53-tól 63-ig terjedő sorok) és íme: az eredmény 16 bites (a Cmes-ben tárolt) érték formájában áll outputra készen. A decimális output a 66-os sorban található EMON51 rutin útján valósul meg.

el. Ebből a célból a programban p1 és q1 értékét 1000-re állítjuk be és indítjuk a programot. Ilyenkor az outputot nem a kondenzátor értéke, hanem közvetlenül az MWERT szám képezi. A mérőpontokra 1 mF-os kondenzátort kötünk. (Ha pontosan szeretnénk mérni, akkor a lehető legkisebb tűrésű kondenzátort kell használnunk.) A mérés során kapott számot (pl. 1540-et) megjegyezzük és a végleges programban q1-ként használjuk, miközben p1 helyén a p1=1000 értéket véglegesítjük. Ezzel a kiegyenlítés már meg is történt, mert: természetesen az 1540-es MWERT fog adódni megint, ha a mérőpontra pontosan 1µF értékű kondenzátort kötünk. A kiadódott értéket p1=1000-rel szorozva és aztán q1=1540-nel osztva pontosan az 1000 nF-os, helyes kijelzés jelenik meg. A monoflop átbillenési ideje megközelítőleg egyenesen arányos a mért kapacitással, tehát a kijelzés kalibrációja helyes.

Már ez az egyszerű mérési program is megmutatta, hogy a mért értékek kiértékeléséhez egy sor részletekbe menő ismeret szükséges. Aki ezen a téren gyakorlatot akar szerezni, az tanulmányozza a monitorbeli aritmetikai rutinokat és kis programok formájában próbálja ki azokat.

Eltolás és forgatás

Az akkumulátorban található bitminta balra vagy jobbra forgatást (rotációját) a következő utasítások segítségével végezhetjük el:

```
RL   A   ; rotáld az akkumulátort balra
RLC  A   ; rotáld az akkut a C bit bevonásával
           balra
RR   A   ; rotáld az akkumulátort jobbra
RRC  A   ; rotáld az akkut a C bit bevonásával
           jobbra
```

Elmélet és gyakorlat

Sajnos a dolog az előbb említett kifejezésekkel a gyakorlatban csak igen mérsékelten válik be. Egyrészt a 47 kΩ-os ellenállásunknak is van bizonyos tűrése. Még súlyosabban esnek azonban a latba a 74HC123 tűrései. Kapacitásmérőnk kalibrálása tehát elengedhetetlen. Ahelyett azonban, hogy az ellenállást egyenlítenénk ki, a kiegyenlítést a programban végezzük

A balra forgató utasítások hatását a *4. ábrán* mutatjuk be.

Az RLC, illetve RRC utasítás előtt a C Flag-et explicite nullázva az akkumulátor el is tolható (shiftelés), ami 0-val való feltöltést tesz lehetővé lépésről lépésre.

Az eltolási művelettel léptetőregiszter funkció valósítható meg (shift-regiszter, lásd később), vagy aritmetikai műveletek céljából számok tolhatók el (lásd pl. a DIV rutin megvalósítását az EMON51-ben).

3

```
****** LISTING of EASM51 (BSP8) ******
LINE LOC  OBJ        T       SOURCE
   1 0000             ; ****** DATEI BSP8.A51 ****************************************
   2 0000             ;
   3 0000             P1      EQU 090H        ; SFR Adressen wie ueblich
   4 0000             ACC     EQU 0E0H
   5 0000             DPL     EQU 082H
   6 0000             DPH     EQU 083H
   7 0000             ;                       ; Eichkonstanten
   8 0000             p1      EQU   1000      ; Zaehler
   9 0000             q1      EQU   1540      ; Nenner des Eichfaktors
  10 0000             ;
  11 0000             MONTOP  EQU   050H      ; benutze Assembler zur RAM Verwaltung
  12 0000                     ORG   MONTOP    ; ueber MONITOR RAM
  13 0050             MZEIT   DS    2         ; 16 Bit gemessene ZEIT
  14 0052             p       DS    2         ; p=p1 : Multiplikator (16 Bit Wert)
  15 0054             q       DS    2         ; q=q1 : Divisor ( 16 Bit Wert)
  16 0056             PROD32  DS    4         ; Produkt MZEIT*p (32 Bit Wert)
  17 005A             Cmes    DS    2         ; Resultat in nF ( 16 Bit Wert)
  18 005C             ;
  19 005C                     ORG   4100H         ; Anfangsadresse des Programms
  20 4100 75 50 00 [2] lp1     MOV   MZEIT+0,#0    ; 16 Bit-Wert MZEIT zuruecksetzen
  21 4103 75 51 00 [2]         MOV   MZEIT+1,#0
  22 4106 C2 92    [1]         CLR   P1.2          ; Trigger:=0
  23 4108 E4       [1]         CLR   A
  24 4109 D5 E0 FD [2] lpp     DJNZ  ACC,lpp       ; Warteschleife Triggerpuls
  25 410C D2 92    [1]         SETB  P1.2          ; Monoflop triggern
  26 410E 90 C0 00 [2]         MOV   DPTR,#0C000H  ; Adresse fuer Monoflop-Ausgang
  27 4111 E0       [2] MLP     MOVX  A,@DPTR       ; Monoflop Ausgang lesen
  28 4112 30 E3 15 [2]         JNB   ACC.3,ENDMES  ; BIT 3 = 0 heisst ENDE
  29 4115 E5 50    [1]         MOV   A,MZEIT       ; sonst 16 Bit Wert erhoehen
  30 4117 24 01    [1]         ADD   A,#1          ; d.h. 1 addieren
  31 4119 F5 50    [1]         MOV   MZEIT,A       ; speichern (LSB)
  32 411B E5 51    [1]         MOV   A,MZEIT+1     ; MSB holen
  33 411D 34 00    [1]         ADDC  A,#0          ; 0 + Uebertrag addieren
  34 411F F5 51    [1]         MOV   MZEIT+1,A     ; MSB speichern
  35 4121 50 EE    [2]         JNC   MLP           ; Uebertrag -> Ueberlauf,sonst weiter
  36 4123 90 41 74 [2]         MOV   DPTR,#OVTXT   ; Ueberlaufmeldung senden
  37 4126 31 88    [2]         ACALL STXT
  38 4128 80 D6    [2]         SJMP  lp1
  39 412A 90 03 E8 [2] ENDMES  MOV   DPTR,#p1      ; Ende der Zeitmessung
  40 412D 85 82 52 [2]         MOV   p+0,DPL       ; p:=p1 ( 16 Bit Wert)
  41 4130 85 83 53 [2]         MOV   p+1,DPH
  42 4133 78 52    [1]         MOV   R0,#p
  43 4135 79 50    [1]         MOV   R1,#MZEIT
  44 4137 75 30 52 [2]         MOV   COMMAND,#ccMUL ; berechne MZEIT*p
  45 413A 12 02 00 [2]         LCALL MON
  46 413D 86 56    [2]         MOV   PROD32+0,@R0  ; speichere nach PROD32 (32 Bit Wert)
  47 413F 08       [1]         INC   R0
  48 4140 86 57    [2]         MOV   PROD32+1,@R0
  49 4142 08       [1]         INC   R0
  50 4143 86 58    [2]         MOV   PROD32+2,@R0
  51 4145 08       [1]         INC   R0
  52 4146 86 59    [2]         MOV   PROD32+3,@R0
  53 4148 90 06 04 [2]         MOV   DPTR,#q1      ; q:=q1
  54 414B 85 82 54 [2]         MOV   q+0,DPL
  55 414E 85 83 55 [2]         MOV   q+1,DPH
  56 4151 78 56    [1]         MOV   R0,#PROD32
  57 4153 79 54    [1]         MOV   R1,#q
  58 4155 75 30 53 [2]         MOV   COMMAND,#ccDIV ; berechne (MZEIT*p)/q (16 Bit)
  59 4158 12 02 00 [2]         LCALL MON
  60 415B 86 5A    [2]         MOV   Cmes+0,@R0    ; speichere nach Cmes
  61 415D 08       [1]         INC   R0
  62 415E 86 5B    [2]         MOV   Cmes+1,@R0
  63 4160 90 41 7F [2]         MOV   DPTR,#TXT1    ; sende ersten Text
  64 4163 31 88    [2]         ACALL STXT
  65 4165 78 5A    [1]         MOV   R0,#Cmes
  66 4167 75 30 05 [2]         MOV   COMMAND,#ccdR016 ; gib Cmes dezimal aus
  67 416A 12 02 00 [2]         LCALL MON
  68 416D 90 41 82 [2]         MOV   DPTR,#TXT2    ; sende zweiten Text
  69 4170 31 88    [2]         ACALL STXT
  70 4172 21 00    [2]         AJMP  lp1
  71 4174                  ;
  72 4174 0D 0A 4F      OVTXT  DB    13,10,'Overflow',0
          76 65 72
          66 6C 6F
          77 00
  73 417F 43 3D 00      TXT1   DB    'C=',0
  74 4182 20 6E 46      TXT2   DB    ' nF',13,10,0
          0D 0A 00
  75 4188                  ;
  76 4188                  ccSTXT  EQU  002H  ; Monitor Kommandos und Adressen
  77 4188                  ccdR016 EQU  005H
  78 4188                  ccMUL   EQU  052H
  79 4188                  ccDIV   EQU  053H
  80 4188                  COMMAND EQU  030H
  81 4188                  MON     EQU 0200H
  82 4188                  ;
  83 4188 75 30 02 [2] STXT    MOV COMMAND,#ccSTXT
  84 418B 02 02 00 [2]         LJMP MON
  85 418E                      END
********** SYMBOLTABLE (26 symbols) **********
       P1 :0090      ACC :00E0      DPL :0082      DPH :0083
       p1 :03E8       q1 :0604   MONTOP :0050     MZEIT :0050
        p :0052        q :0054   PROD32 :0056      Cmes :005A
      lp1 :4100      lpp :4109      MLP :4111    ENDMES :412A
    OVTXT :4174     TXT1 :417F     TXT2 :4182    ccSTXT :0002
  ccdR016 :0005    ccMUL :0052    ccDIV :0053   COMMAND :0030
      MON :0200     STXT :4188
```

3. ábra. Mérőműszerünk programlistája

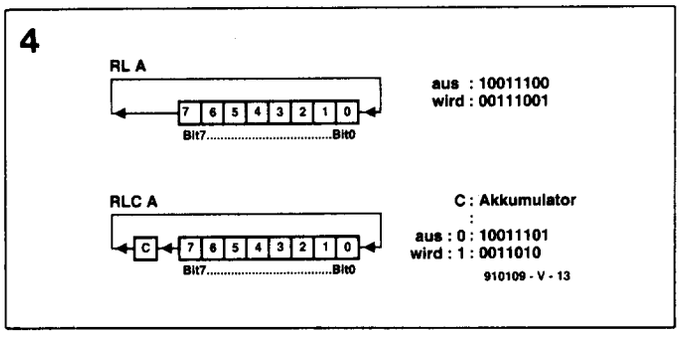
3. ábra

- SFR Adressen wie ueblich = SFR címek szokás szerint
- Eichkonstanten = kalibrációs konstansok
- Zaehler = számláló
- Nenner des Eichfaktors = kalibrációs tényező nevezője
- benutze Assembler zur RAM Verwaltung = RAM kezeléshez assembler használata
- ueber Monitor RAM = Monitor RAM felett
- 16 Bit gemessene Zeit = 16 bites mért idő
- p=p1: Multiplikator (16 Bit Wert) = p=p1: szorzó (16 bites érték)
- q=1: Divisor (16 Bit Wert) = q=q1: osztó (16 bites érték)
- Produkt MZEIT*p (32 Bit Wert) = MZEIT*p szorzat (32 bites érték)
- Resultat in nF (16 Bit Wert) = eredmény nF-ban (16 bites érték)
- Anfangsadresse des Programms = a program kezdőcíme
- 16 Bit-Wert MZEIT zurücksetzen = 16 bites értéket nullázni
- Warteschleife Triggerimpuls = triggerimpulzus várakozóhurok
- Monoflop triggern = monoflop triggerelés
- Adresse für Monoflop-Ausgang = monoflop-kimenet címe
- Monoflop Ausgang lesen = monoflop-kimenet olvasása
- BIT 3=0 heisst ENDE = BIT 3=0 azt jelenti: VÉGE
- sonst 16 Bit Wert erhöhen = egyébként a 16 bites értéket növelni
- d.h. 1 addieren = azaz 1-et hozzáadni
- speichern (LSB) = tárolás (LSB)
- MSB holen= MSB behozása
- 0 + Uebertrag addieren = 0 és átvitel összeadása
- MSB speichern = MSB tárolása
- Uebertrag > Ueberlauf, sonst weiter = átvitel > túlcsordulás, egyébként tovább
- Ueberlaufmeldung senden = túlcsordulási jelentés küldése
- Ende der Zeitmessung = vége az időmérésnek
- p:=p1 (16 Bit Wert) = p:=p1 (16 bites érték)
- berechne MZEIT*p = MZEIT*p számítása
- speichere nach PROD32 (32 Bit Wert) = tárolás PROD32 után (32 bites érték)
- berechne (MZEIT*p)/q (16 Bit) = (MZEIT*p)/q számítással (16 Bit)
- speichere nach Cmes = tárolás Cmes után
- sende ersten Text = első szöveg küldése
- gib Cmes dezimal aus = Cmes decimális output
- sende zweiten Text = második szöveg küldése
- Monitor Kommandos und Adressen = monitor parancsok és címek

4. ábra

- aus: 10011100 = 10011100-ból
- wird: 00111001 = 0011101 lesz

4. ábra. A rotációs utasítások működési módja

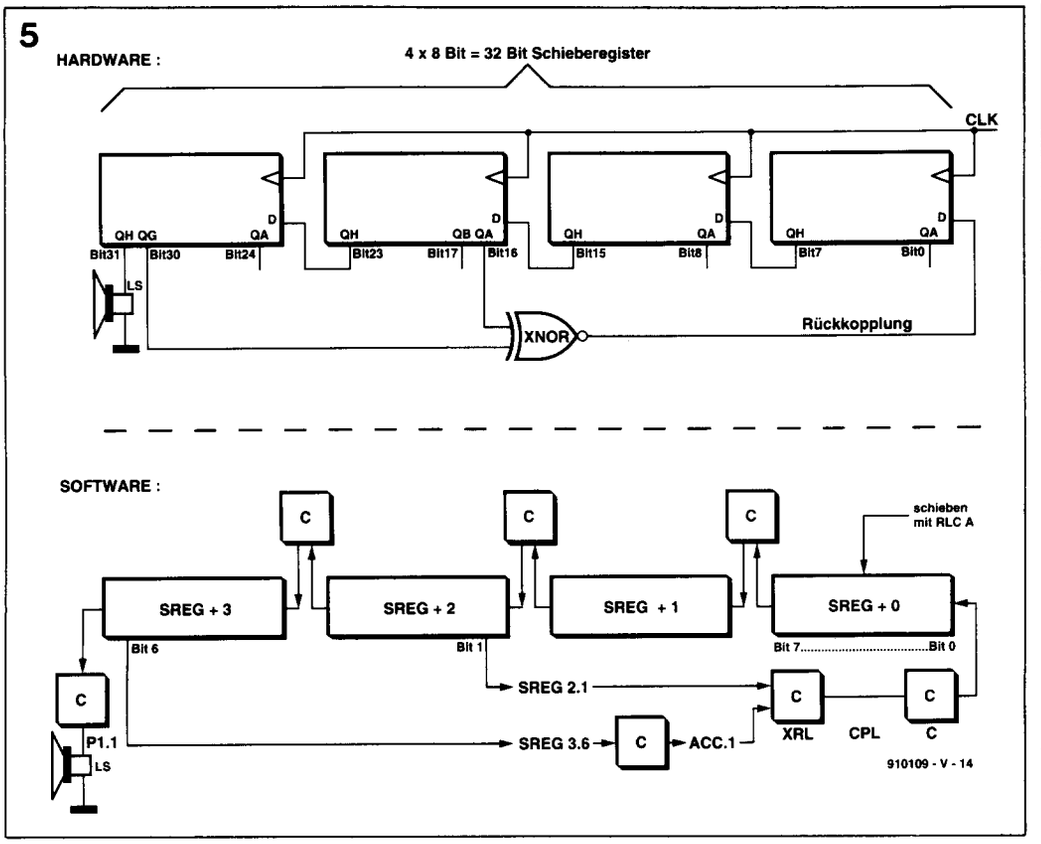


Zajgenerátor

Az [1]-ben olyan zajgenerátor leírása található, melynek lelkét az *5. ábrán* bemutatott léptetőregiszter képezi. Ezt a kapcsolást szimuláljuk most a Compuboardon és így digitális zajgenerátort programozunk.

Outputként természetesen megint a bővítőkártyán rendelkezésre álló hangszórót használjuk, melynek vezérlése P1.1 útján történik.

Az *5. ábrából* állapíthatjuk meg azt is, hogyan alakítható át programmá a kapcsolás. A léptetőregiszter négy egymást követő Byte-ban kerül tárolásra a belső RAM-ban, a jobb szélső bit a legalacsonyabb című Byte-ban található. A tárolás ilyen módja pontosan a 32 bites számok ábrázolásának felel meg. Ezt most balra kell eltolni és ennek során az XNOR



5. ábra. Zajgenerátorként alkalmazott léptetőregiszter: a hardver és a szoftver megoldás összehasonlítása

6

```
****** LISTING of EASM51 (BSP9) ******
LINE LOC  OBJ          T       SOURCE
   1 0000                ; ****** DATEI BSP9.A51 ****************************************
   2 0000                ;
   3 0000                P1      EQU 090H        ; SFR Adressen wie immer
   4 0000                ACC     EQU 0E0H
   5 0000                ;
   6 0000                MONTOP  EQU   050H      ; benutze Assembler zur RAM Verwaltung
   7 0000                        ORG   MONTOP    ; ueber MONITOR RAM
   8 0050                SREG    DS    4         ; 32 Bit Schieberegister
   9 0054                ;
  10 0054                        ORG   4100H           ; Anfangsadresse des Programms
  11 4100 75 50 00 [2]          MOV   SREG+0,#0       ; Schieberegister (32 Bit) loeschen
  12 4103 75 51 00 [2]          MOV   SREG+1,#0
  13 4106 75 52 00 [2]          MOV   SREG+2,#0
  14 4109 75 53 00 [2]          MOV   SREG+3,#0
  15 410C E5 53     [1] NEU     MOV   A,SREG+3        ; bilde EXNOR verknuepfung
  16 410E A2 E6     [1]         MOV   C,ACC.6         ; Bit 6 von SREG Byte 3
  17 4110 92 E1     [2]         MOV   ACC.1,C         ; nach Bit 1 im Akku
  18 4112 65 52     [1]         XRL   A,SREG+2        ; mit Position 1 von SREG+2 XOR
  19 4114 A2 E1     [1]         MOV   C,ACC.1         ; ist XOR von Bit 17 und 30
  20 4116 B3        [1]         CPL   C               ; C ist NOT-XOR von Bits 17 und 30
  21 4117                ;
  22 4117 E5 50     [1] SCHIEBE MOV   A,SREG+0        ; 32 Bit Schiebergister
  23 4119 33        [1]         RLC   A               ; rechtestes Byte schieben
  24 411A F5 50     [1]         MOV   SREG+0,A        ; und abspeichern
  25 411C E5 51     [1]         MOV   A,SREG+1        ; naechstes Byte holen
  26 411E 33        [1]         RLC   A               ; Bit wird in C gespeichert
  27 411F F5 51     [1]         MOV   SREG+1,A
  28 4121 E5 52     [1]         MOV   A,SREG+2        ; usw.
  29 4123 33        [1]         RLC   A
  30 4124 F5 52     [1]         MOV   SREG+2,A
  31 4126 E5 53     [1]         MOV   A,SREG+3
  32 4128 33        [1]         RLC   A
  33 4129 F5 53     [1]         MOV   SREG+3,A        ; C enthaelt nun neue Ausgabe
  34 412B 92 91     [2]         MOV   P1.1,C          ; an Lautsprecher raus
  35 412D 80 DD     [2]         SJMP  NEU             ; und von vorne weitermachen
  36 412F                       END
********** SYMBOLTABLE (6 symbols) **********
       P1 :0090      ACC :00E0    MONTOP :0050     SREG :0050
      NEU :410C   SCHIEBE :4117
```

6. ábra. A digitális zajgenerátor programlistája

(kizárólagos NOR) kapu kimenete írja elő, hogy milyen bittel kell a léptetőregisztert jobbról utántölteni. A hangszórókimenet a legmagasabb pozícióban (31-es bit) helyezkedik el.

A program listája a *6. ábrán* látható. Ez az *5. ábra* közvetlen átalakítása, az egyetlen trükk az XNOR kapu megvalósításában rejlik (15-östől 20-asig terjedő sorok), ahol az akkumulátor 1-es bitje kerül felhasználásra az XNOR kapu kimenetének megfelelő új bit, SREG3.6 és SREG2.1 bitek alapján történő előállítása során. A visszacsatolt léptetőregiszter olyan véletlenszerű mintát (pszeudovéletlen, azaz álvéletlen számsorozatot) hoz létre, mely csak igen hosszú idő múlva ismétlődik. Pontosabban:

A hurok egyszer végigfutása 23 mikroszekundumot vesz igénybe; a léptetőregiszter az ismétlődés bekövetkeztéig mintegy 2 milliárd állapoton megy keresztül. Így tehát olyan „véletlenjel" generálása történik, melynek letapogatási sebessége kb. 43 kHz és amely kb. 13 óra eltelte után ismétlődik. A spektrum formálása céljából természetesen még egy aluláteresztő szűrő hozzákapcsolása is szükséges volna ahhoz, hogy a jelet mérő zajgenerátorként lehessen használni. Hangszórónk azonban aluláteresztő szűrő nélkül is elég erősen zajong (20 kHz-ig megközelítően fehér zajnak megfelelő spektrum).

Feladat

Az Olvasó a következő feladatokra vállalkozhat: A kapacitásmérő méréshatárának kibővítése úgy, hogy az kisebb és nagyobb kapacitások mérésére is alkalmassá váljék. Esetleg automatikus méréshatár átkapcsolás is beprogramozható. A kapacitásmérő úgy is átprogramozható, hogy segítségével meg lehessen állapítani a rákötött kondenzátorról, hogy az az előre megadott értékkel (előírt toleranciasávon belül) megegyezik-e. Mindez lehetőséget nyújt az aritmetikai utasítások kiadós megismerésére. Aki akarja, megkísérelheti a zajgenerátor egyszerű ütemadóként való használatát is.

Kilátások és előrejelzés

Az eddigiekben a 8051-es csaknem valamennyi utasítását megbeszéltük és az utasítások hatását is bemutattuk. Ezzel szoftver szempontjából a 8051-es már a kezünkben van. A következő folytatások ezért inkább hardverorientált tartalmúak: a 8051-es Timerje, LC kijelző csatlakoztatása, a 8051-es soros interfésze és a D/A, illetve A/D átalakítás. ■

5. ábra
- 4x8 Bit = 32 Bit Schiebregister = 4x8 = 32 bites léptetőregiszter
- Rückkopplung = visszacsatolás
- schieben mit RLC A = RLC A-val

6. ábra
- SFR Adressen wie immer = SFR címek szokás szerint
- benutze Assembler zur RAM Verwaltung = RAM kezeléshez assembler használata
- ueber MONITOR RAM = MONITOR RAM felett
- 32 Bit Schiebregister = 32 bites léptetőregiszter
- Anfangsadresse des Programms = a program kezdőcíme
- Schiebregister (32 Bit) loeschen = (32 bites) léptetőregiszter törlése
- bilde EXNOR verknuepfung = EXNOR kapcsolat képzése
- Bit 6 von SREG Byte 3 = SREG 3. Byte-jának 6. bitje
- nach Bit 1 im Akku = az akku 1. bitjében
- mit Position 1 von SREG+2 XOR = XOR kapcsolatban a SREG 2. Byte 1. bitjével
- ist XOR von Bit 17 und 30 = éppen a 17. és 30. bitből képzett XOR
- C ist NOT-XOR von Bit 17. und 30 = C a 17. és 30. bitből képzett NOT-XOR
- 32 Bit Schiebregister = 32 bites léptetőregiszter
- rechtestes Byte schieben = jobb szélső Byte léptetése
- und abspeichern = és tárolása
- naechstes Byte holen = következő Byte behozatala
- Bit wird in C gespeichert = a bit a C-ben kerül tárolásra
- usw. = stb.
- C enthaelt nun neue Ausgabe = C most az új outputot tartalmazza
- an Lautsprecher raus = output a hangszóróra
- und von vorne weitermachen = és folytatás elölről