Beágyazott és Ambiens Rendszerek Laboratórium

BMEVIMIA350

Mérési feladatok az M456 mérésekhez

2021 Benesóczky Zoltán

Bevezető

A laborok során az M123 mérésnél még teljesen hardverből megvalósított egyre bonyolultabb kalkulátorokhoz hasonlókat a MiniRISC mikroprocesszor perifériáit használva, assembly nyelven írt programmal kell megvalósítani.

Linkek a letöltendő anyagokhoz és installálandó programhoz:
MiniRISC processzor (dokumentáció)
A MiniRISC IDE fejlesztői környezet (MiniRISC IDE program)
A MiniRISC_IDE fejlesztői környezetet be kell másolni egy megfelelő könyvtárba. A programot nem kell installálni, csak az exe file-t kell elindítani.
Fontos, hogy a fejlesztői környezetet ne az operációs rendszer felhasználói adatokat tartalmazó könyvtárába (pl. User, Felhasználók, Desktop, Asztal, stb.) csomagoljuk ki, mert onnan indítva nem működik!
A MiniRISC_IDE-ben a Help-re kattintva előjön egy pdf file, amelyben tömören, táblázatosan fel vannak sorolva az egyes utasítások és hatásuk, továbbá perifériánként a periféria regiszterek címei és az egyes bitek funkciója.

A felkészüléskor el kell olvasni a MiniRISC processzor leírásának alább kijelölt részeit (MiniRISC_CPU.pdf):
1.o-5.o Bevezető, 23.o-43.o Program fejlesztői környezet (MiniRISC IDE), 44.o-63.o Utasításkészlet ismertetése, 96.o-110.o Perifériák, 124.o-132.o Példaprogramok.

Az alább ismertetett perifériákat fogjuk használni. Mindegyikhez megadtunk egyszerű programozási példákat is.

- **LED** (basic_ovr, LD): 8 db LED, az LD regiszterébe írva azok a LED-ek gyulladnak ki, amelyekhez tartozó bitbe 1 került. A kiírt érték visszaolvasható.

  Összes LED bekapcsolása:

```
DEF LD  0x80    ;LED-címe
        mov r0, #0xff
        mov LD, r0
loop:   jmp loop
```

- **DIP kapcsolók**, (basic_in, sw): A kapcsolók (8 db), a fenti állásban adnak 1-et. A regiszteréből kiolvasott adat egyes bitjei adják a 8 kapcsoló aktuális állását.

  Kapcsoló beolvasása és kiírása a LED-ekre végtelen ciklusban:

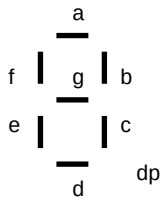
```
DEF LD  0x80    ;LED-ek címe
DEF SW 0x81     ;kapcsolók címe
loop:   mov r0, SW
        mov LD, r0
        jmp loop
```

- **Nyomógombok**, (basic_in_irq, BT): A BT regiszterből olvasva az alsó 4 biten megkapjuk az egyes nyomógombok állapotát, a lenyomott állapothoz 1 érték tartozik. A nyomógombok hardverből prellmentesítve vannak. (Részletesebben lásd. a Mini RISC leírásban.) Kapcsoló lenyomásának (lenyomási esemény) tesztelése:

```
DEF BT    0x84          ;nyomógomb regiszter
DEF BTIF 0x86           ;változás figyelő regiszter
DEF BT0 0x01            ;BT0 mask
loop:   mov r0, BT      ;nyomógombok beolvasása
        mov r1, BTIF    ;megváltozott nyomógombnál a megfelelő BTIF bit 1-lesz
        mov BTIF, r1    ;jelzés(ek) törlése (az törlődik, ahova 1-et írunk!)
        and r0, r1      ;azon bit lesz 1, amelyhez tartozó gombot lenyomták
        tst r0, #BT0    ;BT0 lenyomásának tesztelése (Z=0, ha lenyomták)
        jz  tst_BT1     ;következő BT tesztelése, ha nincs BT0 lenyomás
        jsr BT0_exe     ; a BT0 lenyomása esetén végrehajtandó szubrutin
                        ; végrehajtása
        jmp loop        ;egyszerre csak 1 gomb lenyomását vesszük figyelembe
tst_BT1:tst r0, #BT1
        ...
```

- **Időmultiplexált kijelző** (basic_display): DIG3-0 regiszterekbe kell írni a az adott sorszámú digiten megjelenítendő kijelzési képhez tartozó 7 szegmenses kódot.

A bitsorrend: dp, g, f, e, d, c, b, a



A 8-as szám kiíratása a display 0. digitjére:

```
DEF DIG0 0x90
        CODE ;kódszegmens kijelölése
        mov r0, #8
        mov r1, #sgtbl          ;szegmens táblázat címe
        add r1, r0              ;a szám szegmens képének címszámítása
        mov r1, (r1)            ;szegmenskép beolvasása
        mov  DIG0, r1           ;a szegmenskép perifériába írása

        DATA    ; adatszegmens kijelölése
; A hétszegmenses dekóder szegmensképei (0-9, A-F) az adatmemóriában.
sgtbl: DB  0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f, 0x77, 0x7c, 0x39, 0x5e, 0x79,
0x71
```

- **Timer** (basic_timer): Az időzítő programozható idő múlva jelzést ad egyszer, vagy periódikusan. (A TC parancsregiszter TREP bitjével állítható be). A parancsregiszterbe írt értékkel (TPS2-0) beállítható, hogy a timer számláló léptető jelének frekvenciája a 16MHz-es rendszer órajel hányad része legyen. A lefele számlálónak kezdőérték adható (TR kezdőérték regiszter). Az időzítés lejártakor Timer a státus regiszterében bebillen a TOUT bit, melyet a státusregiszter kiolvasása automatikusan töröl. A parancsregisszter TIE bitjével engedélyezhető, hogy a TOUT jelzés interruptot okozzon. Ezt az eseményt a státus regiszter TIT bitje jelzi. (A MiniRISC CPU minden interrupt hatására az 1-es címre ugrik (egyszerű IT rendszer). Ide a minden interruptot közösen kezelő interrupt rutinra ugró utasítást, vagy magát az interupt rutint kell elhelyezni. Több interrupt esetén az IT rutinban minden IT-sen használt periféria státusregiszterét le kell kérdezni és amelynek a státusa interrupt kérést jelez, azt ki kell szolgálni, jelzését vissza kell törölni, majd az rti utasítással visszatérni az IT rutinból, a főprogramba. Nem szabad elrontani a főprogram regisztereit. Ezt megoldja, ha a főprogram és az IT rutin más regisztereket használ. Csak a kettő közötti kommunikációra használt regiszter(ek) lehetnek közösek.)

Timer IT-vel villogtatott LED (A Timer periféria regiszterek DEF-jeit itt nem tüntettük fel.):

```
DEF TC_INI 0b11110011      ;IT en., 65536-os előosztás, ismétléses, Timer en.
DEF TIT 0b10000000
reset:  jmp main
ISR:    mov r15, TS        ;IT törlése
        tst r15, #TIT      ;Timer IT kérte?
        jz IT_END          ;visszatérés, ha nem a timer kérte (ugrás további IT kérők
                           lekérdezésére, ha vannak ilyenek)
        mov r15, LD        ;LD  beolvasása
        xor r15, #0xff     ;minden bit invertálása
        mov LD, r15        ;visszaírás a LD-be  (LED-ek villogtatása)
IT_END:rti                 ;visszatérés az IT-ből

main:   mov r0, #244
        mov TR, r0         ;16e6/(65536*244)  -> kb. 1 sec
        mov r0, #TC_INI
        mov TC, r0         ;Timer inicializálása
        mov r0, TS         ;esetleges jelzés törlése
        sti                ;globális IT engedélyezés
loop:   jmp loop           ;végtelen ciklus, most itt nincs teendő
```

- **USRT (**slave USRT, szinkron soros adó/vevő**)**: A MiniRISC IDE USRT terminájába beírt érték beíródik a vételi FIFO-ba. Az US státus regiszter RXNF bitje jelzi, ha adat érkezett, az UD ímről olvasva kapjuk meg a FIFO-ból az adatot. Az USRT adatregiszterébe írt adat az adási FIFO-ba kerül, majd kiküldi az USRT. Ha nyomtatható ASCII karakter írunk ki, akkor megjelenik az USRT terminálon. Az US státus regiszter TXNF bitje jelzi, ha adat írható az adási FIFO-ba (UD-be). Interrupt kérés engedéyezése az UIE regiszter megfelelő bitjeinek 1-be írásával lehetséges. A laborban csak a vételt kell IT-sen megírni (RXNE bit 1-be írása).

  Adat fogadása a terminálról és a vettnél eggyel nagyobb adat visszaküldése programozott státus lekérdezéses periféria kezeléssel.

```
DEF UC  0x88          ; USRT kontroll regiszter     (csak írható)
DEF US  0x89          ; USRT FIFO státusz regiszter (csak olvasható)
DEF UIE 0x8A          ; USRT megszakítás eng. reg.  (írható/olvasható)
DEF UD  0x8B          ; USRT adatregiszter          (írható/olvasható)
DEF RXNE 0b00000100
DEF TXNF 0b00000010
            mov r0, #0x0f          ;adás és vételi FIFO törlése, adás és vétel engedélyezése
            mov UC, r0
wait_rec:   mov r0, US
            tst r0, #RXNE
            jz wait_rec            ;várakozás bejövő karakterre
            mov r1, UD             ;karakter beolvasása
wait_send: mov r0, US
            tst r0, #TXNF
            jz wait_send           ;várakozás amíg nincs hely az adási FIFO-ban
            add r1, #1             ;kiírandó karakter módosítása
        mov UD, r1                 ;karakter kiírása
        jmp wait_rec               ;ugrás az elejére
```

A jegyzőkönyvvel és a feltöltendő melléklettel kapcsolatos követemények

A HF portálra feltöltendő file legye zip formátumban tömörítve. A file tartalma: a jegyzőköny (pdf formátumban) és az összes program forrása, továbbá a CALC2_5-höz az lst (lista) file is. Ez utóbbiból kiderül az is, hogy mennyi kódmemóriát foglal a program. Utóbbi file a MiniRISC IDE könyvtára alatti Assembler alkönyvtárban található.
A jegyzőkönyvben a CALC2_1, CALC2_2, CALC2_3 feladatok megoldásáról elég csak röviden írni. (Le kell írni felsorolásszerűen, hogy milyen részfeladatokra lett bontva, mely részfeladatok megoldása okozott nehézséget, továbbá az esetleges tapasztalatokat.)

Részletesen a CALC2_4 és CALC2_5 programokról kell írni:

- Leírás a részfeladatokra bontásról
- Leírás az egyes részfeladatokhoz tartozó szubrutinokról, programrészekről
- A szubrutin/programrész feladata
- Bemenő paraméterek (ha vannak), kimenő paraméterek (ha vannak)
- Komplexebb részfeladatok folyamatábrája, parancs dekódoló állapotgráfja
- A működés/algoritmus rövid magyarázata (itt be kell szúrni a megfelelő program részt, de, ha lehetséges, az összetartozó program sorok mardjanak egy oldalon)
- A feladat megoldása közben mik okoztak nehézséget, milyen esetleg másoknak is hasznos tapasztalatokat szereztetek.

Követelmények az assembly programokhoz
Az assembly programokban lehetőleg minden felhasznált konstansnak legyen neve (DEF).
A program legyen jól tabulálva. A legszélső oszlop a címkék helye. A következő oszlopban (1 TAB után) jön az utasítás.
Az utasítás sor kinézete valami hasonló legyen:
TAB vagy címke: utasítás neve, space, op2, vessző, space, op2
A program legyen megfelelő részletességgel kommentezve. Ne legyen túlkommentezve (nem kell minden utasítást magyarázni), de ne legyen alul kommentezve sem (egy hosszabb szubrutinnak csak az elején van némi magyarázat). A komment az utasítás után legalább egy TAB-al kezdődjön. Az azonos oldalon levő kommentek lehetőleg ugyanabban a vízszintes pozícióban kezdődjenek.
Minden komplexebb részfeladatot szubrutinnal kell megvalósítani. (Ettől a CALC_5-ben lehet eltérni a kódmemóriával való spórolás végett.) A szubrutinok elején kommentben mindenképpen legyen ott, hogy mit csinál, mik a bemenő paraméterei (ha vannak), mik a kimenő paraméterei (ha vannak). Az egyes programrészek, szubrutinok között legyen egy üres sor. Ügyelni kell arra, hogy szubrutint jsr utasítással kell hívni, egyszerű ugró utasítással oda ugrani nem szabad. (A jsr elmenti a stackre a jsr-t követő utasítás címét) Szubrutinból kilépni csak rts utasítással szabad, egyszerű ugró utasítással nem. (Az rts leszedi a stack tetejéről a visszatérési címet és oda ugrik.)

Mérési feladatok

Az alábbi feladatok közül azokban, ahol bemeneti operandusokat a kapcsolón (SW) állítjuk be, op1-et a kapcsoló felső 4 bitje, op2-őt a kapcsoló alsó 4 bitje alkotja. Előjel nélküli számábrázolást használunk. A program komplexebb részfeladatait szubrutinok segítségével valósítsa meg! A főprogramban az r0..r5 regisztereket használja. A szubrutinokban az r6..r12 (input r7, r6, output r7, r6). Ahol interruptos kezelés van, ott az IT rutinban az r13..r15 regisztereket és a főprogrammal közös regiszter(eke)t az r5, r4-ből válassza! Előjel nélkülis számok összehasonlítása (cmp) vagy kivonás (sub) után a negatív értéket a C flag-gel kell tesztelni, nem az N flag-gel. (Az N flaget 2-komplemens ábrázolású számok esetén kell erre használni, de ilyen a feladatok között nincs.) Az alábbi feladatokban nem specifikált részleteket saját specifikációval kell kiegészíteni a jegyzőkönyvben, a kokrét megoldásnak megfelelően.

CALC2_1

Miután a bemeneti operandusokat a kapcsolókon beállítottuk, a kiválasztott művelet típusának megfelelő nyomógomb megnyomása után az eredményt *közvetlen bináris formában a 8 db LED* diódán jelenítjük meg. Az eredmény maradjon a kijelzőn a következő nyomógomb megnyomásáig. Az egyes műveletek és a hozzájuk rendelt nyomógomb: Összeadás (op1+op2, bt0), kivonás (op1-op2, bt1), szorzás (op1*op2, bt2), ***kizáró vagy*** (XOR) (op1 XOR op2, bt3). ***Az esetleges hibás eredményt az összes LED bekapcsolásával jelezzük. Hibának számít, a kivonás negatív eredménye***. Ezt nem az N flag jelzi, hanem a C, mivel a bemenő adatoknál és az eredménynél nem 2-es komplemens számábrázolást használunk. A szorzást szubrutinnal valósítsa meg.

CALC2_2

Módosítsuk a CALC2_1 nevű egységet úgy, hogy az alábbi műveleteket támogassa: Összeadás, kivonás, szorzás, ***osztás (op1/op2)*.** Osztás esetén a LED-ek felső 4 bitje az egész részt, alsó 4 bitje a maradékot mutassa. Bővítsük a hibakezelő részt, hogy figyelje a nullával történő osztást is. Tehát ***akár negatív eredményű kivonás, akár 0 osztó*** (op2) esetén minden LED égjen a művelethez rendelt nyomógomb megnyomása után. A bonyolultabb műveleteket és amelyekhez hiba jelzés is tartozik, szubrutinnal valósítsa meg (szorzás, osztás). A szubrutin a hibát a zero flag-ban jelezze, hiba esetén a zero flag legyen 0 értékű. Az osztás algoritmusát alapvetően a bináris osztás algoritmusával várjuk. Akinek ez nem sikerül, az kivonásos módszerrel is elvégezheti, de ez kevesebbet ér (pont levonás).

CALC2_3

Cseréljük le a CALC2_3 kijelzőjét a *basic display*-re! A kijelző 4 digiten balról jobbra haladva egy-egy digiten kijelzi a két bemeneti operandust és az utolsó két digiten pedig az eredmény értékét ***hexadecimálisan***. Osztás esetén a 1. digiten az eredmény egész része, az 0. digiten pedig a maradék jelenjen meg, a kettő között a pont égjen. ***Hibás vagy nem értelmezhető eredmény esetén az eredmény helyett (utolsó két digit) az „EE” hibajelzést adja.*** A kijelzéshez ***írjon olyan szubrutint***, amely a következőképpen működik: Az r7, r6 regiszterekben kapja meg a dig32 és dig10-on megjelenítendő adatot. Az r8 regiszter felső 4 bitje közül amelyik 1 értékű azon a digiten nem jelenik meg semmi (blank), az alsó 4 biten, ahol 1 van, azon digit pontját (dp) kigyújtja.

CALC2_4

A CALC2_3 egység szolgáltatása mérnöki szemmel már egészen kielégítő, azonban átlagos felhasználó számára esetleg zavaró a hexadecimális eredmény. Fejlesszük tovább a rendszert, úgy, hogy a bemeneti adatok is és az eredmények is **tízes számrendszerben** jelenjenek meg. (A többi tulajdonsága maradjon meg.)
Ennek érdekében a számológép a bemenetein beállított kapcsolóállásokból csak a 0 … 9 tartományba eső értékeket fogadja el és jelenítse meg, az ennél nagyobb (de beállítható) értékekre ***a bemeneti***

adathoz tartozó kijelző egy „E” hibajelzést ad (mindaddig, amíg a kapcsolók értéke > 9). Az így korlátozott bemeneti adattartomány esetén, minden műveletnél az eredmény <= 99 (max. 81) tehát 2 decimális digiten ábrázolható. ***Hibás vagy nem értelmezhető eredmény esetén az eredmény helyett*** (utolsó két digit) ***0.5 sec-onként villogó „EE”*** hibajelzést ad. (Az operandus hibát jelző E-t nem kell villogtatni.) 10-es számrendszerű kijelzéshez meg kell valósítani a bináris BCD átalakítást 8 bites bináris adaton. Ehhez egy gyakran használt algoritmus a „shift add 3” elnevezésű. Ugyanerre a célra fel lehet használni az osztó szubrutint is, azonban mivel az eredmény max. 81 lehet, 8 bites osztást kell megvalósítani. (A 8 bites osztó rutin a 4 bites számokhoz is jó.) A ***bináris BCD átalakítást is szubrutinnal valósítsa meg***! A hibajelzés 0.5 sec-os villogtatását a ***Timer periféria interruptos használatával*** oldja meg**!** Az IT-rutinnak az r4 regiszter 1-be állításával jelezze, ha hiba jelzést kell adni. (Ha r4 = 1, az IT rutin végezze el az eredmény digiteken az EE villogtatását, ha r4 = 0, akkor az eredmény digitek kikapcsolt állapotában hagyja abba a villogtatást.) **A kijelzéshez a CALC3-ban kért, ott megadott specifikációjú szubrutint használja!**

CALC2_5

A CALC2_5 egység a CALC2_4 egység jelentősebb módosítása az alábbiak szerint:

- A bemeneti operandusokat és a végrehajtandó műveletet a nyomógombok és kapcsolók helyett egy USRT interfészen keresztül kapja az egység. ***Mindkét operandus legfeljebb 1 decimális számjegyet tartalmazhat.*** Azaz egy bemeneti szekvencia: 1 digites decimális szám (op1:0..9); műveleti kód (+, -, *,/); 1 digites decimális szám (op2), majd lezárásként „=” vagy enter (választható). ***A lezáró karakter hatására jelenjen meg az eredmény, decimális formában***. Ekkor a kijelző alsó 2 digitje az eredményt mutassa, a felső 2 pedig az operandusokat. Osztás esetén az egész rész és a tört rész között a pont világítson.
- ***Az ESC-re kezdődjön az új adatok bevitelére várakozás***. ESC bármikor megnyomható, hatására az operandus bevitel alaphelyzetbe áll, és újra az első operandust várja. Az adatok bevitele során az adott beviteli fázisban nem értelmezhető karaktereket a program hagyja figyelmen kívül. ***Amíg nincs adat bevitel (adatra vár), addig a kijelzőn ne jelenjen meg semmi. (Ez ESC hatására történik.)***
- ***Az adatok bevitele alatt a kijelző 3. digitje az op1-et 2. digitje az op2-t mutassa decimálisan. (Az op1 bevitele után, de op2 bevitele előtt csak a 3. digit világítson. A 2. digit csak op2 bevitele után világítson, de az eredmény digitek ekkor még sötétek. Az eredmény digitek csak a lezáró karakter hatására világítanak.)***
- ***Hibás eredmény esetén az eredmény digiteken EE jelenjen meg, a felső 2 digiten továbbra is az operandusok látszanak.***
- ***Az USRT vétel kezelését interruptosan oldja meg!*** Az USRT IT az r5 regiszter 1-be állításával jelezze a főprogramnak, ha új karakter érkezett. A beérkezett adatot az r15-ben adja át a főprogramnak. A főprogram az adat felhasználása után törölje az r5-öt.
- **A kijelzéshez a CALC3-ban kért, ott megadott specifikációjú szubrutint használja!**
- Opcionális kiegészítés (nem kötelező, ha még van hely a program memóriában, akinek van kedve, megpróbálhatja bele tenni):
  a. A hibás eredmény esetén kiírt EE 0.5 sec-onként villogjon.
  b. Az eredményt USRT-on keresztül a terminál emulátorra is ki kell íratni, hasonló formában, az „=” jel után (estleg még egy soremelés legyen). A soremelés a terminál emulátoron CR és LF hatására történik meg, mindkettő kiíratása szükséges hozzá.

Megjegyzések:
A CALC2_5 bonyolultsága még a hardver verziójú feladathoz képest jelentősen leegyeszrűsített specifikáció ellenére is feszegeti a mikorokontroller kódmemóriájának határait (max. 256 utasítás). Ezért ennél a feladatnál törkedni kell arra, hogy az egyes részfeladatok lehetőleg mennél rövidebb programmal legyenek megoldva. Emiatt a mérési utasítás elején megadott regiszter megosztáshoz nem kell szigorúan ragaszkodni. Fontos, hogy a bonyolultabb funkciókat szubrutinok oldják meg. Ha az adott funkcióra több helyen is szükség van akkor ez jelentős kód memória spóroláshoz vezet és a

program szerkezete is jobban áttekinthető lesz. Ha csak egyszer használandó kódrészről van szó, akkor viszont nem szubrutinként megvalósítva további hely nyerhető. Azonban ez a megoldás csak végszükség esetén használandó.

Az parancs dekódert célszerű állapotgráfszerű programszervezéssel megoldani.

Az alábbi megoldásban a szubrutin csak akkor tér vissza, ha a teljes adatfeldolgozás megtörtént (elérte a végállapotot). Ekkor a megfelelő regiszterekben visszaadja az adatokat (op1, op2, elvégzendő művelet kódja) és kezdő állapotba kerül. Ez csak egy általános mintapélda az állapotgráfszerű megvalósításra, a konkrét feladatban 4-nél több állapotra lehet szükség...

```
DEF State1      1
DEF State2      2
DEF StateEnd    3
DEF StateDefault   4

;state reg: r12
state_loop:
        ;új adat jött tesztelése
        ;ugrás vissza, ha nem jött, állapotgépes feldolgozás, ha jött
State_1:
        cmp r12, State1
        jnz State_2
        ;tevékenység State1-ben
        ;feltétel vizsgálat
        mov r12, #új_állapot_kódja      ;ha a megfelelő feltétel teljesül, a köv. állapotba lép
        jmp state_loop
State_2:
        cmp r12, State2
        jnz State_End
        ;tevékenység Statek2-ben
        ;feltétel vizsgálat
        mov r12, #új_állapot_kódja      ;ha a megfelelő feltétel teljesül, a köv. állapotba lép
        jmp state_loop
State_End:
        cmp r12, StateEnd
        jnz State_Default               ;hibás állapotkód, ilyen csak program hiba miatt lehet!
        ;tevékenység StateEnd-ben
        ;feltétel vizsgálat
        mov r12, #State1                ;ha a feltétel teljesül, a kezdőállapotba lép
        rts                             ;visszatérés, minden adatot feldolgoztunk
State_Default:                          ;egyik állapottal sem egyezik, program hiba, ha ide ér!
        mov r12, #State1                ;legközelebb a kezdő állapotból indul
        jmp state_loop
```

Ha egy állapotból többfele lehet ágazni, akkor abban az állapotban ez a rész többször is szerepel:

```
        ;feltétel vizsgálat
        mov r12, #új_állapot_kódja      ;ha a megfelelő feltétel teljesül, a köv. állapotba lép
```

Melléklet

Bináris osztás algoritmusa (4 bites számokon bemutatva)Mintapélda: 14 : 5 = 2 maradék 4

(1110 : 0101 = 0010 maradék 0100)
n bites szám esetén i kezdőértéke n-1, a ciklus lépésszáma: n

i	osztandó (maradék)	maradék-osztó*2^i	maradék-osztó*$2^i < 0$?	osztó*2^i	osztó	h3	h2	h1	h0
inicializálás	1110	1110		01010000	0101	0	0	0	0
3	1110	1110 -0101000	**igen** (a hányasdos i-edik bitje 0)	0101000	0101	**0**			
2	1110	1110 -010100	**igen** (a hányasdos i-edik bitje 0)	010100		0	**0**		
1	1110	1110 - 1010 --------- 0100	**nem**, (a hányados i-edik bitje 1) elvégezzük a kivonást és az eddigi maradékot az eredménnyel felülírjuk	01010		0	0	**1**	
0	**0100**	0100 -0101	**igen**	0101		**0**	**0**	**1**	**0**

Javaslatok egy 8 bites osztást végző progamhoz:
A ciklus számláló kezdőértéke 8, mivel 8-as ciklussal végezhető el az algoritmus.
Az osztó*2^i előállításához az osztó regisztert és egy 0 kezdőértékű másik regisztert úgy használunk, hogy azok együtt egy 16 bites regisztert alkotnak, melynek kezdetben a felső 8 bitje az osztó. Ezt a regiszter párost minden lépésben 1-szer jobbra shiftelve megkapjuk az osztó*2^i-t.

Amig e regiszter páros felső 8 bitje nem 0, addig biztosan teljesül, hogy maradék- osztó*$2^i < 0$, csak ezután kell kivonással ellenőrizni. Ha egy kivonás során a C (carry flag) 1, akkor az eredmény negatív. A hányados kezdőértéke 0. Ha a hányadosba írandó bit negáltját a C flagban állítjuk elő (a kivonás után pont ez a helyzet), akkor az i-edik lépésben a C flag-et be tudjuk shiftelni az eredmény regiszter legkisebb bitjébe, majd ezt megnegálva előállítjuk a megfelelő értékű bitet. A shiftelések során a teljes ciklus végére minden bit a megfelelő helyre kerül. (A fenti táblázat ezt nem így mutatja, hanem h(i)-ben piros jelöli az aktuálisan előállított bitet.) A labor feladatokban elég 4 bites osztást megvalósítani, ami kevesebb utasítással megoldható. Ez az utolsó feladatnál lehet fontos, a kódmemóriával való sprórolás miatt.

Utoljára módosította: Benesóczky Zoltán 2021.04.09.