A Kalkulus algebrai műveletvégzése

A négy alapművelet

Mindegyik alapművelet, és a többi művelet is, három fő részre bontható:
1. Előfeldolgozás
2. Művelet végrehajtása
3. Utófeldolgozás

Az alapműveletek végrehajtását és az elő- utó-feldolgozását az `ASMD_ROUTINES.INC` program tartalmazza.
(Az **ASMD** betűszó az **A**ddition, **S**ubtraction, **M**ultiplication és **D**ivision szavak kezdőbetűiből alakult ki.)
A program az alapműveleteket a következőképpen hajtja végre (Az eredmény mindig az X-regiszterbe kerül):
Összeadás: az Y-regiszter tartalmát adja hozzá az X-regiszter tartalmához,
Kivonás: az Y-regiszter tartalmából (kisebbítendő) kivonja az X-regiszter tartalmát (kivonandó),
Szorzás: az Y-regiszter tartalmával megszorozza az X-regiszter tartalmát,
Osztás: az Y-regiszter tartalmát (osztandó) elosztja az X-regiszter tartalmával (osztó).
Megjegyzés: A regiszterekben lévő számok ábrázolásakor az egész és tizedes-jegyek elválasztásánál vesszőt (**tizedesvessző**) használok. A pontok csak a szám tagolását jelzik a jobb olvashatóság miatt.

1. Összeadás

1.1. Előfeldolgozás

Normalizált számoknál akkor végezhetjük el az összeadást (és a kivonást is), ha azok azonos kitevőjűek. Ha ez nem teljesül, azonos kitevőre kell hozni a számokat. Mindig a nagyobb szám kitevőjéhez kell igazítani a kisebb szám kitevőjét. Azt, hogy melyik szám nagyobb a kitevőik értékei és azok előjelei határozzák meg.(A mantissza értékek és azok előjelei ennél a döntésnél nem érdekesek.)

Legyen mindkét szám kitevőjének előjele pozitív. Nagyobb az a szám, amelynek nagyobb értékű a kitevője. Ha az egyik szám kitevője pozitív, a másiké pedig negatív, akkor a pozitív kitevőjű szám a nagyobb. Amikor mindkét szám kitevőjének az előjele negatív, akkor a kisebb kitevőjű szám a nagyobb. (Pl. $10^{-2} > 10^{-4}$) Az azonos kitevőre való hozást az `ALIGN_XY_EXP` szubrutin végzi el.

Az elő-feldolgozás része még annak az eldöntése, hogy tényleg összeadást kell-e végezni, vagy kivonás lesz belőle. Ennek a döntésnek az alapja a számok (mantisszák) előjele. Az alábbi egyszerű példák segítenek a döntésben:
(A számok és előjelük zárójelben vannak, hogy ne keveredjenek a műveleti jelekkel.)

(+X) + (+Y)	->	összeadás	->	(+Y) + (+X)
(-X) + (+Y)	->	kivonás	->	(+Y) - (+X)
(+X) + (-Y)	->	kivonás	->	(+X) - (+Y)
(-X) + (-Y)	->	összeadás	->	(-X) + (-Y)

1.2. Az összeadás végrehajtása

Nagyon egyszerű és gyors művelet. Itt használható ki a PIC utasításkészletének egyetlen tagja (**DAW**), amely támogatja a BCD műveleteket, pontosabban: az összeadást. Az alábbi négysoros program végzi el két becsomagolt BCD Byte összeadását, úgy, hogy az eredmény is becsomagolt BCD legyen.

```
MOVF    X_x,W     ;   Az X-regiszter 1 byte-ja a W-regiszterbe
ADDWFC  Y_y,W     ;   A W-regiszter tartalmát hozzáadja, Carry-bit
figyelembevételével, az Y-regiszter tartalmához és az eredmény a W-regiszterbe kerül.
DAW               ;   BCD korrekció (Részletes leírása PIC gyári adatlapján.)
MOVWF   X_x       ;   Az eredmény (becsomagolt BCD) az X-regiszterbe kerül.
```

Ha túlcsordulás is volt, akkor a Carry-bit =1 lesz.

1.3. Utófeldolgozás

A fenti rövid programrész hétszer ismétlődik: x és y = 0 ....6. Az összeadás végén az X_6,4 bit értéke lehet 1 is. Ez akkor fordul elő, ha túlcsordulás volt művelet végrehajtása során. Ebben az esetben normalizálni kell az X-regisztert, ami úgy történik, hogy az X-regiszter tartalmát 4-gyel el kell tolni jobbra, azaz 10-zel kell osztani.(`SHIFT_X_R1`)

Ha, például, az eredmény: 12,345.678.901.234, akkor az eltolás után: 1,234.567.890.123 lesz. A jobb szélen álló **4**-es elveszik. (A pontok csak a szám tagolását segítik.) Természetesen az exponenst is módosítani kell, azaz 1-gyel növelni kell, ami a teljes szám tekintetében 10-zel való szorzást jelent. Ha a kitevő előjele pozitív, akkor az exponens növelése +1 hozzáadásával megoldható. Ha az exponens előjele negatív, akkor 1-et ki kell vonni a kitevő abszolút értékéből. Ez jelenti a 10-zel való szorzást.

2. Kivonás

2.1. Előfeldolgozás

Ha szükséges, az Y-regisztert el kell menteni.
A kitevőket azonos értékűre kell hozni.

2.2. A kivonás végrehajtása

A kivonás műveletnél is összeadást végzünk úgy, hogy a kisebbítendőhöz hozzáadjuk a kivonandó szám tízes komplemensét. Egy szám kilences komplemensének kiszámítása az alapja a tízes komplemens kiszámításának. (A kilences komplemenshez 1-et adunk és máris megkapjuk a tízes komplemenst.)

Példa: számoljuk ki az ***y – x*** értéket (mindkét szám 2 számjegy hosszú) Legyen **y=17** és **x=11.**

A kivonást így is felírhatjuk: ***y – x = y + (100 – x) - 100*** **17 - 11 = 17 + (100 - 11) - 100**

A zárójelben lévő szám ***(100-x)*** az ***x*** tízes komplemense. **100 - 11 = 89**

A tízes komplemenst felírhatjuk így is: ***100-x = (99 – x) + 1.*** **(99 - 11) + 1 = 89**
A zárójelben lévő szám ***(99-x)*** az ***x*** kilences komplemense. Egy szám kilences komplemense maguknak a számjegyeknek 9-re való kiegészítése: 0-9, 1-8, 2-7, 3-6, 4-5, 5-4, 6-3, 7-2, 8-1, 9-0. Ha ehhez 1-gyet hozzáadunk, akkor kapjuk a tízes komplemenst.

A példa alapján: **17 - 11 = 17 + 89 - 100 = 106 - 100 = 6**

A program a kivonandó becsomagolt-BCD számjegyeinek tízes komplemensét a `TENS_COMPLEMENT` szubrutinnal állítja elő. A 10-es komplemensű számhoz hozzáadja az Y-regiszter tartalmát (`ADD_Y_2_X`) és majdnem kész a kivonás.

2.3. Utófeldolgozás

Egy kivonás eredménye lehet pozitív vagy negatív, a kisebbítendő és a kivonandó abszolút értékétől függően. Ha az előző tízes-komplemenses összeadást követően az X_6 regiszter felső négy bit-je 9-es értékű, akkor a kivonás eredménye negatív. (X_6 = 1 0 0 1 X X X X) Ekkor az eredménynek is a 10-es komplemensét kell venni.

Az előző példa módosításával, ha **y=17** és **x=19**:
17 - 19 = 17 + (99 - 19 + 1) - 100 = 17 + 81 - 100 = 98 - 100 = -2

Az utolsó lépés: ha szükséges, az Y-regisztert vissza kell állítani.

3. Szorzás

3.1. Előfeldolgozás

Ha szükséges, az Y-regisztert el kell menteni.

3.2. A szorzás végrehajtása

A szorzás végrehajtása egyszerűbb, mint a kivonásé. Nem kell közös kitevőre hozni a szorzótényezőket. Csak össze kell szorozni a mantisszákat majd a kitevők előjeleitől függően összeadni azokat egymáshoz vagy kivonni azokat egymásból. Ilyen egyszerű.

A Kalkulus programja a mantisszák szorzását összeadásokra vezeti vissza. Most a mantisszák abszolút értékeivel számolunk! Ezt 7*7 ciklusban, azaz 49 lépésben végzi el. A szorzás elvégzéséhez a program mindhárom indirekt-regisztert használja:

FSR0 az R-regiszter mutatója (átmeneti tároló)
FSR1 az X-regiszter mutatója
FSR2 az Y-regiszter mutatója

Az átmeneti eredmény a P-regiszterbe kerül.

Az első 7 lépésben az X_0 file-regiszter tartalmát szorozza meg az Y_0 … Y_6 regiszter értékeivel. Ezt a műveletet az INNER_LOOP („belső_hurok”) nevű szubrutin végzi el. Az egyes szorzások elvégzése helyett az eredményt szorzótáblából veszi elő. Ehhez szükséges a Kalkulus program legnagyobb méretű táblázata: TBL_MULTIPLICATION.INC

A „szorzás” eredményei egy hurkon belül az R-regiszterben gyűlnek, majd a ciklus végén hozzáadódnak a P-regiszterhez.

A következő 7 lépésben az X_1 file-regiszter értéke kerül összeszorzásra az Y_0 … Y_6 értékeivel. És így tovább.

A 49. szorzás, vagy a 7. hurok, után már majdnem kész az eredmény. Még egy fontos lépés van hátra. A P-regiszter tartalmazza az eddigi eredményt, de nem helyérték szerint! A P-regiszter tartalmának az X-regiszterbe való áttöltésekor lehet beállítani a helyértékeket:

```
MOVFF  P_C,X_6
MOVFF  P_B,X_5
MOVFF  P_A,X_4
MOVFF  P_9,X_3
MOVFF  P_8,X_2
MOVFF  P_7,X_1
MOVFF  P_6,X_0
```

(A szorzótábla elvileg 100*100 elemet tartalmaz. (00-tól 99-ig!) A gyakorlatban ez csak 100*98 elemű táblázat. Először is, kihasználjuk azt a tényt, hogy bármely szám nullával szorozva nulla eredményt ad. Másodszor, bármely szám 1-gyel szorozva: önmaga. Ezt a két speciális esetet a software figyeli. Ezek nagyon fontos kivételek, mert így a táblázat nem a H’00000’ címen kezdődik, hanem a H’00400’ címen. Különben gond lenne a program H’00000’ címről való indulásával. A H’00400’-as címig „szabad” a program-memória, így egy rövid bootloader használata is megoldható.

A táblázat címzéséhez a szorzótényezőket használjuk. (Ezek becsomagolt BCD Byte-ok!) Az egyik szorzótényező adja a TBLPTRL értékét, míg a másik a TBLPTRH értékét. Az így kapott 16 bit-es számot meg kell szorozni 2-vel, mert az eredmény 2 Byte-os. Jó, hogy tízes számrendszert használunk, mert így jókora rések vannak a táblázatban, ahová elfér a program többi része. Tizenhatos számrendszernél a teljes program-memóriát kitöltené a táblázat.)

Megjegyzés:

A szorzás gyors elvégzéséhez nagyon hasznos a szorzótábla, de nem feltétlenül szükséges, hogy a PIC-ben legyen. Lehet az a külső memóriában is. Könnyen belátható, hogy ez viszont lelassítja a szorzás elvégzését.

Nem is feltétlenül szükséges a szorzótábla. A PIC utasításkészlete tartalmaz egy MULWF utasítást, amely a W-regiszterben és a kiválasztott f-regiszterben lévő két 8-bit-es bináris szám összeszorzását végzi el egy utasítás alatt. Az eredmény 16 bit-es. Ha ezt az utat választjuk, először a becsomagolt BCD számokat kell binárissá alakítani, majd a szorzás után az eredményt visszaalakítani becsomagolt BCD számmá. Ez a megoldás is jelentősen lassítja a művelet elvégzését.

A szorzótábla segítségével végrehajtott szorzás, ha a szorzótényezők nem tartalmaznak 1-est és 0-át, 570µsec-ig tart, 32MHz-es órajelnél. A teszt-számok: 5,555.555.5 * 10^5 * 7,777.777.7 * 10^7 = 4,320987567901* 10^{13}. Ugyanez a számítás a BCD-bináris és bináris-BCD konverzió használatával több, mint 4,5msec-ig tart. Az osztásnál ez az érték 19,1msec-ről 146msec-re nő. Persze, még ez sem túl sok egy négy alapműveletes számológépnél.

Ez a megoldás előnyökkel is jár. Mivel kisebb program-memóriájú PIC áramkörök is használhatók, jelentősen megnő az alkalmazható eszközök száma. (A ***hp*** Woodstock család új alapjánál a PIC18F46K22-es IC-t használom.)

3.3. Utófeldolgozás

Először meg kell vizsgálni, hogy a mantisszák összeszorzása után van-e túlcsordulás. (Mivel csak normalizált számokat használunk egy szorzás legnagyobb eredménye:

9,999.999.999.999 * 9,999.999.999.999 = 99,999.999.999.998 lehet.

Azaz, az X-regiszter X_6 Byte-jában a felső négy bit lehet nullától eltérő. Ekkor van túlcsordulás. Ebben az esetben, a normalizáláshoz az eredményt el kell osztani tízzel és az X-regiszter kitevőjét meg kell növelni. (Ha a kitevő előjele pozitív, akkor az exponens növelése +1 hozzáadásával megoldható. Ha az exponens előjele negatív, akkor 1-et ki kell vonni a kitevő abszolút értékéből.)

Az utófeldolgozás következő lépése az exponensek összeadása. Természetesen, itt sem az exponensek abszolút értékeinek összeadásáról van szó, hanem az előjeles összeadásról, ami az exponensek előjelétől, valamint az exponensek abszolút értékétől függ.

Most már a szorzás eredménye az X-regiszterben van. A normalizálás után csak az X-mantissza előjelét kell beállítani, ami egyszerű dolog. Ha mindkét szorzótényező azonos előjelű: az eredmény pozitív. Ha a szorzótényezők ellentétes előjelűek, akkor az eredmény előjele: negatív.

Az utolsó lépés: ha szükséges, az Y-regisztert vissza kell állítani.

4. Osztás

4.1. Előfeldolgozás

Ha szükséges, az Y-regisztert el kell menteni.
Az X-regiszter ellenőrzése: ha nulla, akkor hibajelzés, mert a nullával való osztás nem értelmezhető.
Az Y-regiszter ellenőrzése: ha nulla, akkor az eredmény is nulla. Nem kell számolni.

4.2. Az osztás végrehajtása

Sokáig az osztás egyszerű és gyors végrehajtása lehetetlennek tűnt. Miután az ember rájön az algoritmusra, már nagyon egyszerű az egész. Itt is a fokozatos közelítést lehet alkalmazni. (Először csak a mantisszákkal foglalkozunk!)

A feladat, a hányados kiszámítása: osztandó / osztó = hányados (+ maradék)

Fordítsuk meg a dolgot! Közelítsük meg az osztandót a következő módon:
osztandó = „hányados” * osztó.
Ahol a „hányados” egy ismert szám, vagy inkább, ismert számok összege:

osztandó = Σ m* N * osztó, ahol m = 0 vagy 1, $N=2^n*10^{-12}$, ahol n = 0….43.

Lássuk ezt egy kicsit részletesebben:
Az osztandó és az osztó maximális értéke: 9,999.999.999.999 lehet.
Az ismert számok értékei legyenek a 2 egész-számú hatványai megszorozva 10^{-12}-nel.
(A pontok itt is csak a tagolást segítik.)

(Úgy is felírhatjuk ezt, hogy az osztót és az osztandót is megszorozzuk 10^{12}-nel. (Azért ennyivel, mert 12 tizedes jeggyel számolunk. Így egész számokat kapunk, melyek maximális értéke: 9.999.999.999.999. Az ismert számok értékei ebben az estben legyenek a 2 egész-számú hatványai.)

Példák: $2^{43}*10^{-12}$ = 8,796.093.022.208

.

$2^{30}*10^{-12}$ = 0,001.073.741.824

.

$2^{11}*10^{-12}$ = 0,000.000.002.048

.

$2^{5}*10^{-12}$ = 0,000.000.000.032

.

$2^{0}*10^{-12}$ = 0,000.000.000.001

Szorozzuk meg az osztót az első próbaszámmal, a $2^{43}*10^{-12}$-nel (8,796.093.022.208), majd vonjuk ki az osztandóból. Ha az eredmény negatív, akkor ezt a számot, mármint a $2^{43}*10^{-12}$-t, elvetjük. Ha az eredmény pozitív, akkor megtartjuk, mert része lesz a hányadosnak. Pozitív értéknél a kivonás eredményével számolunk tovább. Addig folytatjuk ezt, amíg el nem fogynak a 2 hatványai. A 2 hatványait, természetesen, táblázatból lehet kiolvasni. A táblázat kiolvasását a `LOAD_PWR2_2_Y` szubrutin végzi el.

4.3. Utófeldolgozás

Hasonló, mint a szorzásnál. Először az X-regiszterben lévő eredményt kell normalizálni, majd az eredmény kitevőjét kell kiszámítani. A mantissza előjelének kiszámítása azonos a szorzásnál használt eljárással.

Ha szükséges, az Y-regisztert vissza kell állítani.

5. Négyzetre emelés (x^2)

Ez a művelet csak megszokásból vagy kényelmi okokból került a számológépbe. Mindig az X-regiszterben lévő számot emeli négyzetre.

5.1. Előfeldolgozás: Az Y-regiszter elmentése.
5.2. A négyzetre emelés végrehajtása: az X-regisztert átmásolja az Y-regiszterbe, majd végrehajtja a szorzást.
5.3. Utófeldolgozás: az Y-regiszter visszaállítása.

6. Reciprok ($1/x$)

Az előzőhöz hasonlóan ez is kényelmi művelet. Mindig az X-regiszterben lévő szám reciprokát számolja ki.

6.1. Előfeldolgozás: Az Y-regiszter elmentése. A művelet végrehajtása előtt ellenőrzi az X-regiszterben lévő szám értékét. Ha az nulla, akkor hibajelzést ad, mivel 0-val nem értelmezhető az osztás.
6.2. A reciprok érték kiszámítása: 1-gyet ír az Y-regiszterbe, majd végrehajtja az osztást.
6.3. Utófeldolgozás: az Y-regiszter visszaállítása

7. Négyzetgyök ($\sqrt{x}$)

Ez már egy kicsit bonyolultabb művelet. A Newton-Raphson iterációs, azaz közelítéses, módszert használom.

$R_{n+1} = [R_n + N/R_n] / 2$
;N = Az a szám, melynek a négyzetgyökét szeretnénk kiszámolni.
;R_n = kezdőérték, ahol n=0.
A gyorsabb műveletvégzés érdekében a kezdőértéket táblázatból olvassa ki a program. Nem kellett túl sok kezdőértéket eltárolni. Mindösszesen 20 érték van a **TBL_SQUARE_ROOT.INC** táblázatban. Ennek az, az oka, hogy két-dekádonként a számok gyökei a mantissza számjegyekben megegyezők.
Példák:

$\sqrt{0,02} = 1,414213562373 * 10^{-1}$
$\sqrt{2} = 1,414213562373 * 10^{0}$
$\sqrt{200} = 1,414213562373 * 10^{1}$

$\sqrt{0,2} = 4,472135954999 * 10^{-1}$
$\sqrt{20} = 4,472135954999 * 10^{0}$
$\sqrt{2000} = 4,472135954999 * 10^{1}$

7.1. Előfeldolgozás: Az Y-regiszter elmentése. A művelet végrehajtása előtt ellenőrzi az X-regiszterben lévő szám értékét. Ha, az, negatív, akkor hibajelzést ad.
7.2. A négyzetgyök érték kiszámítása.
7.3. Utófeldolgozás: az Y-regiszter visszaállítása

8. Pi (π)

A π értékének beírása nem művelet, csak kényelmi funkció. A π értéke az x-regiszterbe kerül: $3,141.592.535.897*10^0$. A π „beírásakor” a stack megemelkedik, kivéve, ha azt ENTER előzte meg.

9. Faktoriális ($n!$)

A faktoriális érték „kiszámításakor” nem használok aritmetikai műveleteket. A PIC program-memóriájában van a n=0 …n=69 értékeket tartalmazó táblázat: **TBL_FACTORIAL.INC**. Így egyformán rövid idő bármelyik faktoriális érték kiolvasása.

9.1. Előfeldolgozás: Az Y-regiszter elmentése. Az X-regiszter tartalmának ellenőrzése, hogy az, 0 és 69 közötti természetes szám-e. Ha ez nem teljesül, a művelet végrehajtása hibajelzéssel leáll.
9.2. A faktoriális táblázat kiolvasása és beírása az X-regiszterbe.
9.3. Az Y-regiszter visszaállítása

Utolsó módosítás: 2021. 01. 02.