Mikrokontroller II.

Levente VAJNA
(Mérési partner: Válik Levente Ferenc)
(Gyakorlatvezető: Tihanyi Attila Kálmán)
Pázmány Péter Katolikus Egyetem, Információs Technológiai és Bionikai Kar
Magyarország, 1083 Budapest, Práter utca 50/a
vajna.levente@hallgato.ppke.hu

***Kivonat*—A labor során MSP430 mikrokontroller szimulátorral, az IAR Visual State programmal végeztünk méréseket Assembly nyelven. Regiszterekbe írtunk ki pozitív illetve negatív egész számokat, és ezekkel végeztünk szorzást, illetve osztást. Tekintve, hogy nincs ezen műveletvégzéshez parancs létrehozva, teljes egészében magunk kellett megírjuk.**

***Keywords*-MSP430; IAR Visual State; flag; Assembly; szorzás; osztás;**

Mérés ideje: 2023.05.18.

I. Feladat: Mérés során felmerülő fogalmak

I-A. Assembly

Az Assembly a számítógépes programozás egyik legalacsonyabb szintű nyelve, amely közvetlenül kommunikál a számítógép hardverével. Az Assembly nyelv alap utasításokból áll (pl: move, add, sub), amelyeket a processzor közvetlenül értelmez és végrehajt. A programok írása assembly nyelven lehetővé teszi a maximális kontrollt a hardver felett, és lehetőséget nyújt a hatékonyság és a teljesítmény optimalizálására. Assembly nyelvű program írása bonyolult, és meglehetősen időigényes, azonban az így készült program idő és teljesítményhatékony.

I-B. MSP430

Az MSP430 [1] mikrokontroller a Texas Instruments fejlesztése, mely igen alacsonyszintű programozási ismereteket igényel, azonban ezzel együtt idő-, és energi hatékony. Programozása gyakran Assembly nyelven történik, mi is így használtuk.

I-C. Számrendszerek

A különböző számrendszerek életünk számos terén megtalálhatóak. Mindennapi életünkben, de alapvetően a matematikában is a tizes számrendszert, vagyis a decimális számrendszert használjuk. Itt a 10 az alap, tehát a számjegyek 1 és 9 közti számok.

Informatikában gyakran használatos a hexadecimális, vagyis a 16 alapú számrendszer. Itt az számjegyek lehetnek 1 és 9 közti számok, illetve betűk A-F között. (A = 10, ... , F = 15) Gyakran használt például színskáláknál, vagy memóriacímzéseknél. Elsősorban azért kedvelt számrendszer, mert a 16 egy kettő hatványa, $16 = 2^4$, tehát egy számjeggyel ábrázolhatunk 4 számjegynyi bináris számot.

És az informatika alapja a bináris, vagyis a kettes számrendszer. Olyan lényeges, hogy az 5 Neumann-elv közt is szerepel ennek használata. Könnyű használata, mivel így kettéfelé bontható a digitális jel, logikai magas, és logikai alacsony feszültségre (0, 5V). Kétféle számjeggyel kell leírni minden számot, 0 vagy 1. (false, true)

Ábrázolásuk helyiértékekkel és alaki értékekkel történik. [2] Képlet rá, ahol k alapú a számrendszer:

$$\sum_{i=0}^{n} a_i \cdot k^i$$

Elsősorban ezeket az ismereteket a mérés során rutinszerűen kell tudjuk alkalmazni, mivel a regiszterek értékét hexadecimálisan ábrázolva láthatjuk csak, a számítógép binárisan dolgozik, és azzal kell gondolkodjunk, de egyben a számmegadás pedig decimálisan történik.

I-D. Kettes komplemens

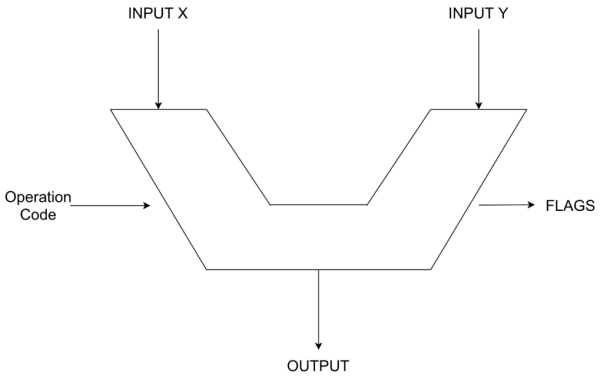
A ketes komplemens számábrázolást az előjeles egész számok minél praktikusabb ábrázolásának igénye hívta életre. Úgy alkották meg, hogy egy kivonásnál a kivonandót könnyedén, kettes komplemensű negatív számként ábrázolva a kisebbítendőhöz hozzáadva el lehessen végezni. Az alábbi algoritmussal képzünk kettes komplemens negatív számot:

$$\neg|neg.szam + 1| \tag{1}$$

Vagyis a negatív számhoz hozzáadunk egyet, majd vesszük az abszolútértékét, és elvégezzük rajta a kettes számrendszerbe átírást. Ezt követően pedig értékenként negáljuk, tehát minden 1-es 0 lesz és minden 0 1-es lesz. [3]

I-E. ALU

Az ALU (Arithmetic Logic Unit) [4] a számítógépek nélkülözhetetlen eleme, mely a CPU-n, vagyis a processzoron kap helyet. Alapvető, fundamentális számításokat végez el, összead, kivon, illetve egyes logikai műveleteket képes elvégezni, mint például AND, OR, XOR.



1. ábra. Arithmetic Logic Unit

Mint az 1. ábrán is látható, két bemeneti értékből ad ki egyet. Ezek jellemzően a regiszterekből, vagyis a műveletvégző egységhez legközelebb álló volatile memóriákból származó adatok, értékek. Ezen kívül van egy extra bemenet, ami a különböző műveletek elvégzését szabja

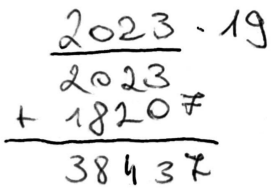
ki rá, illetve egy extra kimenet, az ún. flagek, vagy status bitek, amik néhány extra adattal szolgálnak felénk (pl.: Carry, Overflow, Zero, Negative).

II. FELADAT: SZORZÁSOK ELVÉGZÉSE

II-A. Mi is a szorzás?

A szorzás igazából ismételt összeadás. Mit jelent ez? Azt jelenti, hogy a szorzandót szorzószor adom hozzá a nullához. Papíron való szorzás egy egyszerűsített műveletvégzés, ahol már feltételezzük, hogy tudunk két egyjegyű számot összeszorozni, vagyis szorzószor nullához a szorzatot hozzáadni. (Megjegyzés: A szorzás asszociativitása miatt az is helyes, ha a szorzót adjuk hozzá nullához szorzandószor.)

Először is nézzük meg hogyan is szorzunk össze két egész számot papíron.



2. ábra. Két decimális egész szám szorzása

a 2. ábrán látható egy papíron végzett szorzás. Nem is az eredmény a lényeges, hanem a metódus. Először is a szorzó legnagyobb helyiértékével megszorzom a szorzandót, amit leírok. Következő lépésben ugyanezt megismétlem, tehát a szorzó második legnagyobb helyiértékével végigszorozzuk a számot, majd ezt is leírjuk, DE mivel tudjuk, hogy az a számjegy eredetileg a második legnagyobb helyiértéken van csak, ezért pont egy helyiértéknyivel (nagyságrenddel) kevesebbet ér. Ez technikailag azt jelenti, hogy decimális számoknál tízzel el kell osszuk, vagyis a papírra eggyel eltolva (elshiftelve) írjuk le. Ezt ismételjük meg annyiszor, ahány helyiértékes a szorzó. Ha minddel megvagyunk, nincs más dolgunk, mint összeadni (nagyságrendhelyesen) a kapott, leírt eredményeinket.

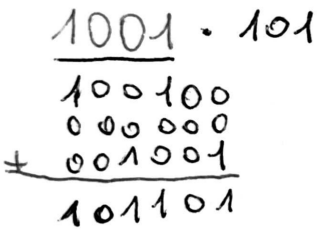
Mi is áll emögött? A szorzás disztributív (2. azonosság) tulajdonságát használjuk ki. Mi ez?

$$a \cdot (b + c) = a \cdot b + a \cdot c \tag{2}$$

Ennek értelmében $2023 \cdot 19 = 2023 \cdot (1 \cdot 10^1 + 9 \cdot 10^0) = 2023 \cdot 1 \cdot 10^1 + 2023 \cdot 9 \cdot 10^0$. Azért írtam így fel, hogy egyértelműen lássuk a tízes számrendszer helyiértékeit. És tudjuk még továbbá, hogy egy S számrendszerbeli szám S^X -nel való szorzása esetén van a legkönnyebb dolgunk, mivel ez X -szer való balraeltolását jelenti a számnak úgy, hogy a szám végére 0-kat írunk.

Ezt az előző tulajdonságot használjuk fel a kettes számrendszer-beli szorzás esetén is, illetve azt, hogy az eggyel való szorzása a számnak önmagát jelenti, nullával való szorzása a számnak pedig nullát jelent, és nullát hozzáadni a számhoz továbbra is változatlanul az eredeti számot jelenti. (Megjegyzés: Fontos észrevenni, hogy egy A meg egy B számjegyű szám szorzása egy $A+B$ számjegyű szorzatot fog jelenteni.)

Nézzük is meg, hogy néz ki egy hasonló szorzás bináris számokkal, azaz a kettes számrendszerben.



3. ábra. Két bináris egész szám szorzása

Nagyon hasonlóképp végzünk szorzást bináris számokkal, itt a 3. ábrán látható konkrét példában a $9 \cdot 5 = 45$ szorzást végeztem el binárisan, és ha visszaváltjuk a számot decimális számrendszerbe láthatjuk, hogy tényleg 45 lesz az eredmény. A megfontolás mögötte ugyanaz.

II-B. Nézzük meg ezt a gyakorlatban!

Az MSP430 szimulátorral Assembly nyelven először megpróbálkozunk egy bináris számot kettővel megszorozni. Ehhez felhasználjuk a II-A. részben található ismereteket.

```
MOV.B   #19, R4
RLA.B   R4
```

Ezután ha megnézzük mi van a négyes regiszterben, 38-at fogunk látni hexadecimálisan, azaz $0x0026$.

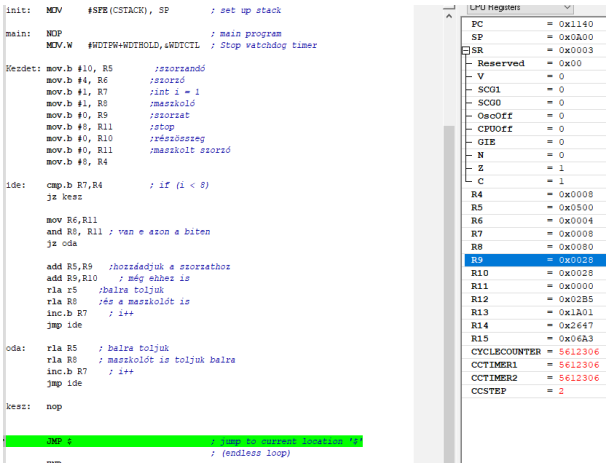
Most próbáljuk meg a 19-et 10-zel megszorozni. Tudjuk, hogy ez azt jelenti, hogy $19 \cdot (1 \cdot 2^3 + 1 \cdot 2^1)$, vagyis binárisan úgy néz ki, hogy $10011b \cdot 1010b = 10111110b$. Ezt úgy végezzük el, hogy:

```
MOV     #19, R4
MOV     #0, R5
RLA     R4
ADD     R4, R5
RLA     R4
RLA     R4
ADD     R4, R5
```

Ha most megnézzük az R5 regiszter értékét $0x00BE$ értéket láthatnánk, ami pont a 190. Itt pont azt csináltam, amit a II-A. részben leírtam, aképpen, ahogy itt fentebb leírtam, vagyis megszoroztam kettővel (balra rotate), azt hozzáadtam R5 regiszter kezdeti 0 értékéhez, majd megszoroztam kétszer egymás után kettővel (balra rotate), és ezt a számot is hozzáadtam R5-höz.

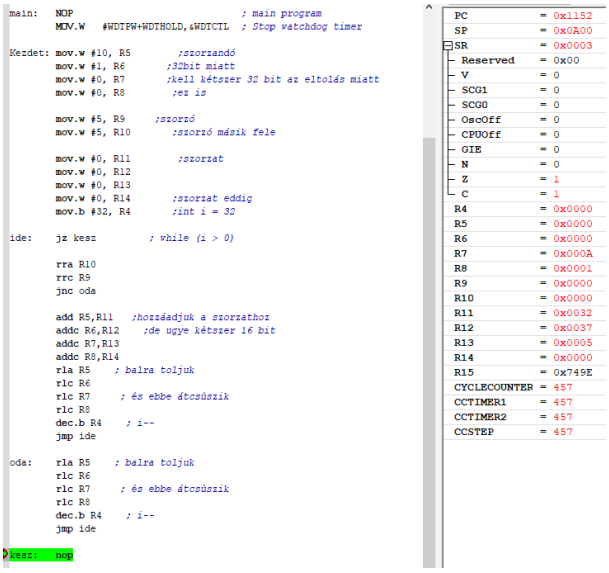
Nézzük meg ezt két tetszőleges 8bites (8 számjegyű) számmal.

A 4. ábrán látható példában összeszorzom a tizet a néggyel. Amire figyelni kell, hogy itt nem tudjuk előre, hogy mi lesz a szám, szóval nem írhatjuk bele csak így, hogy mikor kell összeadni és eltolni. Így automatizáljuk, és elágazásokat teszünk bele. Ebben az esetben maszkolással megvizsgálom, hogy az éppen soronkövetkező biten egyes van-e. Ha igen, akkor hozzá is adja az eredményregiszterhez, shifteli a maszkolót, és a szorzandót is, valamint növeli a lépésszámlálót. Ha nulla a maszkolás eredménye, akkor átugorja az összeadást, és csak shifteli a maszkolót meg a szorzandót, illetve növeli a lépésszámlálót. Ha a lépésszámláló elérte azt a számot, ahány biten végzünk szorzást (R4), akkor leáll.



4. ábra. Két tetszőleges 8bites egész szorzása

16 biten és 32 biten az elv ugyanez, csak már kifutunk a környezet által számukra biztosított határok közül, ezért még a múltórai laboron elsajátítottak alapján külön össze is kell adni, illetve a shiftelés során is több regiszternyi adatot tolunk el. Hogy is néz ez ki.



5. ábra. Két tetszőleges 32bites egész szorzása

Mivel az előző maszkolós megoldás a 32 bites példában túl sok regisztert elhasználna, többet, mint amennyink erre van, ezért más megvalósítást kell eszközölnünk, de a háttér továbbra is változatlan. Most az 5. ábrán látható, hogy nem maszkolót léptetünk balra, hanem magát a szorzót jobbra. Ez lényegét tekintve ugyanazt eredményezi, mindig a soron következő LSB (less significant bit) -t vizsgáljuk, illetve a shiftelés végett ez a bit kicsúszik a Carry bitbe, tehát egész pontosan a Carry értékétől függően fogjuk végrehajtani, avagy átugrani az összeadást. Az összeadás tényleg változatlanul megmaradt, annyi különbséggel, hogy ahogy múlt laboron csináltuk, a Carry bit értékét hozzáadjuk a magasabb word értékéhez. Továbbá arra is figyelni kell, hogy mint ahogy azt a II-A. részben kiemeltem, számítani kell arra, hogy megduplázódik az eredmény számjegyeinek száma.

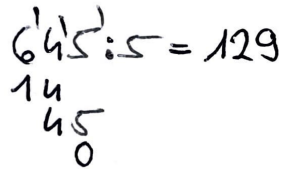
Vegyük számba, mi történik, ha előjeles a szám. Nem történik semmi egetrengető dolog, elején meg kell vizsgálnunk az előjelbiteket, például egy nullát hozzáadással, így a Negative bitben egy egyes lesz, és ezekkel végezhetünk elágazásokat a **JZ** utasítás segítségével. Vagy végezhetünk velük **XOR** műveletet is. Ezt pedig el kell tárolni, hogy ennek függvényében legyen a végén az eredmény, valamint figyelnünk kell arra, hogy eggyel kevesebb bitet vehetünk figyelembe, hiszen az előjel bit nem vehető figyelembe az összeadásokkor.

III. Osztások elvégzése

III-A. Mi is az osztás?

Ha végiggondoljuk az egészosztás elvégzése tulajdonképpen egy ismételt kivonás. A kérdés az, hogy hányszor tudjuk ”teljesen” kivonni az osztandóból az osztót, ez lesz a hányados, amennyi pedig marad, mert nem tudjuk ”teljesen” kivonni a legvégén, az lesz a maradék. [5]

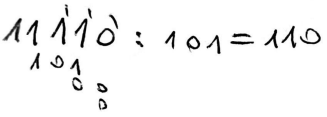
Hogyan is végezzük ezt el papíron osztást?



6. ábra. Két decimális egész szám osztása

Látható a 6. ábrán, hogy szerencsére nem ilyen hosszadalmas a dolgunk, van erre algoritmus, amit még alsóban jóeséllyel megtanultunk. Hogy is működik ez? Vesszük az első számjegyet (ha abban egyértelműen látjuk hogy nincs meg, mert mondjuk több számjegyű az osztó, vagy mert tudjuk, hogy kisebb az a szám pl.: ha 6 helyett a 6. ábrán 4 lenne, akkor vehetünk rögtön több számjegyet is papíron, de ezt a gép nyilván nem fogja hasraütésszerűen megmondani), és megnézzük, hogy abban hányszor van meg. Az eredményt leírjuk az egyenlőségjel mögé, a maradékot pedig leírjuk alá. Majd vesszük a következő helyiértéket, leírjuk mellé (ez azt jelenti, hogy az előző maradékot megszorozzuk tízzel, vagyis shifteljük eggyel balra úgy, hogy melléhelyezzük a jelenlegi utolsó számjegyet, amit még nem vettünk. Ebben az újonnan képzett maradékban nézzük most meg, hogy hányszor van meg az osztó, ennek eredményét is leírjuk az egyenlőségjel mögé, majd a maradékot ugyanúgy leírjuk. Ezt ismételjük annyiszor, ahány számjegyünk van. Látható, hogy ez minden bizonnyal kevesebb ciklussal jár, mint az eredeti eljárás, de persze mindegyik helyes.

Binárisan is nézzük meg.



7. ábra. Két bináris egész szám osztása

Látható, hogy teljesen ugyanez történik, még annyival könnyebb dolgunk is van, hogy csak el kell döntenünk a kettő számról, hogy megvan-e benne (≡ nagyobb vagy egyenlő a maradék szám, mint az osztó), tehát ez is egy nagyon könnyen programozható eljárás.

III-B. Lássuk a gyakorlatban!

Tulajdonképpen meg is tudánk valósítani az osztást, hogyha az osztandóból kivonjuk az osztót, és egy számláló regiszterben számolnánk mennyiszer tudtuk ezt megtenni, és a ciklus tart mindaddig, amíg nagyobb a kisebbített osztandó az osztónál (első módszer a III-A. részből):

```
        MOV     #255, R4
        MOV     #5, R5
        MOV     #0, R6

start:  CMP     R5, R4
        JL      fine

        SUB     R5, R4
        INC     R6
        JMP     start

fine:   NOP
```

A kódban R4 az osztandó, R5 az osztó, R6 a ciklusváltozó. A program végén az R4 regiszterben lesz a maradék, és R6 regiszterben a hányados. Ez is egy jó megoldás.

Nézzük meg a III-A. részben ismertetett második megoldással, ami olyan, mint a papíron osztás.

```
        MOV     #255, R4    ;osztando
        MOV     #0, R5      ;maradek
        MOV     #5, R6      ;oszto
        MOV     #8, R7      ;szamlala
        MOV     #0, R8      ;hanyados
        CLRZ
start:  JZ      fine

        RLA     R4
        RLC     R5
        CMP     R6, R5
        JL      next

        SUB     R6, R5
        SETC
        RLC     R8
        DEC     R7
        JMP     start

next:   CLRC
        RLC     R8
        DEC     R7
        JMP     start

fine:   NOP
```

Ez a kód valószínűleg gyorsabb, de ugyanúgy működik mindkettő. Természetesen mindkettő esetben ellenőrizni is kéne, hogy nem nulla az osztó, mert első esetben végtelen ciklusba jutnánk, mivel mindig kivonná a nullát, de így sosem lesz az osztandó kisebb nullánál. Második esetben pedig csupa egyes lenne az eredmény, mivel bármilyen számot raknánk a maradék regiszterbe, az nem lenne kisebb nullánál, tehát a maradékba kerülne az osztandó, és mivel nagyobb, ezért kivonja belőle a nullát, tehát marad önmaga, és egyest beír a hányados regiszterébe. De feltételezzük most, hogy ismeri mindenki, hogy nullával nem osztunk.

Ha ennél több bites számokat szeretnénk osztani egymással, hasonlóképp a szorzáshoz, annyi csupán a teendőnk, hogy több regiszternyi adatot shiftelünk a maradék esetén is, az osztandó esetén is, és a hányados esetén is. Az összehasonlításkor pedig word-nként ellenőrizzük, hogy nagyobb-e, illetve ha ugyanakkora, akkor megnézzük a következő regiszter értékét, ha pedig utolsó regiszter, akkor az alapján döntünk.

Mi történik, ha negatív számok közt végezzük? Ugyanúgy járunk el, mint szorzás esetén, az első előjelbitetkülön kezeljük, nem vesszük figyelembe a műveletvégzés közben, csak az elején **XOR** művelettel eldöntjük, hogy negatív vagy pozitív lesz az eredmény, mert ugye a XOR művelet az, ami éppen azt vizsgálja, hogy mindkettő megegyezik-e, vagy mindkettő különböző. Ezt az előjelet az elején a megoldásba beleshifteljük. Előny, hogy tudunk így negatív számokat is szorozni/osztani, hátrány, hogy eggyel kevesebb biten ábrázolhatunk számot.

Lezárás

Összegzésképpen az órán nehéz volt, és nem volt elég idő átgondolni, de otthon volt idő szépen átgondolni, és lejátszani, hogy mi is történik. Tanulságosnak mindnképp mondható, mert így mégjobb betekintést nyertem mind a számítógép számolásába, mind az Assembly nyelvbe.

Hivatkozások

[1] TexasInstruments, „Msp430 user's guide," 2006. [Online]. Available: https://www.ti.com/lit/ug/slau049f/slau049f.pdf?ts=1649510678917&ref_url=https%253A%252F%252Fwww.ti.com%252Fsitesearch%252Fen-us%252Fdocs%252Funiversalsearch.tsp%253FlangPref%253Den-US%2526searchTerm%253Dslau049%2526nr%253D160

[2] K. András, „Digitális rendszerek számábrázolás, mikrokontrollerek," 05 2023. [Online]. Available: https://moodle.ppke.hu/pluginfile.php/74654/mod_resource/content/1/Bev_Meres_2022_uC.pdf

[3] M. B. Naszlady, „Adatábrázolás és logikai áramkörök," p. 12, 09 2022.

[4] Y.-Y. Chuang, „Arithmetic logic unit (alu) introduction to computer," 09 2017. [Online]. Available: https://www.csie.ntu.edu.tw/~cyy/courses/introCS/17fall/lectures/handouts/lec04_ALU.pdf

[5] TexasInstruments, „Efficient multiplication and division using msp430," 07 2018. [Online]. Available: https://www.ti.com/lit/an/slaa329a/slaa329a.pdf?ts=1684531083159&ref_url=https%253A%252F%252Fwww.ti.com%252Fsitesearch%252Fen-us%252Fdocs%252Funiversalsearch.tsp%253FlangPref%253Den-US%2526searchTerm%253Dslaa329a%2526nr%253D2