Mikrokontroller I., Számábrázolás

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 összeadást, illetve kivonást. Kellett saját összeadást illetve kivonást is programozni, mivel 32 illetve 64 biten ilyen műveletvégzés nincs definiálva. Mindeközben elsajátítottuk a hexadecimális, a decimális és a bináris számrendszerek közti átváltást, valamint megfigyeltük a különböző flageket a számítások során.**

***Keywords*-MSP430; IAR Visual State; flag; Assembly; számrendszerek; átváltás;**

Mérés ideje: 2023.05.11.

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:

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

I-D. Kettes komplemens

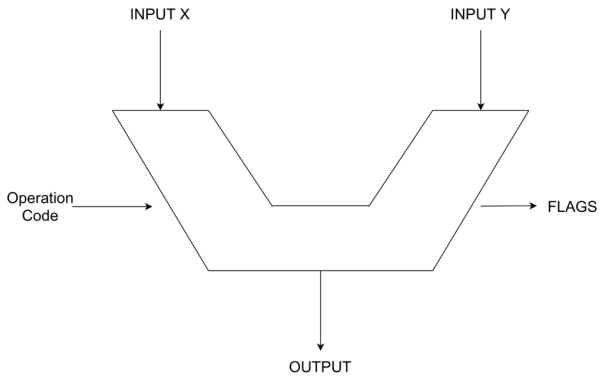
A ketes komplemens számábrázolást az előjeles egész számok minél praktikusabb ábrázolásának igénye hívta életre. Úgy alkották meg, hogy egy kivonásnál a kivonandót könnyedén, kettes komplemensű negatív számként ábrázolva a kisebbítendőhöz hozzáadva el lehessen végezni. Az alábbi algoritmussal képzünk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

Mind az összeadások esetén, mind a kivonások esetén első lépésként a regiszterekbe be kellett tölteni a literál konstansainkat. Erre szolgáltak az alábbbi utasítások:

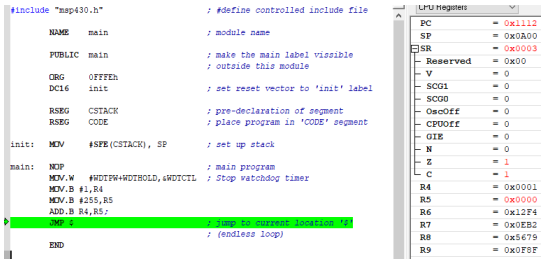
MOV.B MOV.W

Az összeadások elvégzésére kétszer kettőféle parancsunk is rendelkezésre állt:

ADD.B ADD.W ADDC.B ADDC.W

II-A. Két 8 bites előjel nélküli szám összeadása

Miután két regiszterbe egy-egy byte adatot töltöttünk a **MOVE.B** paranccsal, ezen számokat összse is kellett adni. Erre szolgált az **ADD.B** utasítás. A byte típus itt egy egy byton eltárolható számot vár. Ha nincs előjele, akkor ez lehet ugye egy 0-255 közötti egész, ha előjelesen használnánk, akkor egy -128 és +127 közötti egész értéket adhatnánk csak meg.



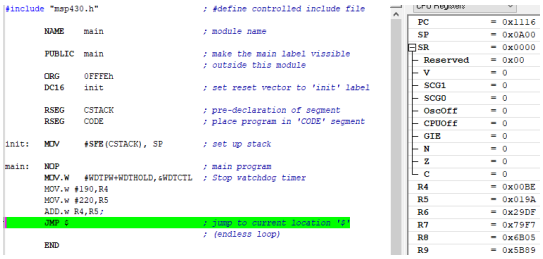
2. ábra. Két 8 bites előjelnélküli egész összeadása

A 2. ábrán látható, hogy két előjel nélküli egész számot töltök az R04 és az R05 regiszterekbe, majd ezeket adom össze. Az **ADD** utasítás szintaktikája szerint először adjuk meg az src, azaz a source értékét, és másodjára a dst, vagyis destination értékét, és az összeadás eredményét az úticél(dst) regiszterbe fogja tölteni. A 2. ábrán látható példában 255-öt, és az 1-et szeretném összeadni 1 byte-on. Ez azt jelenti, hogy mivel nem fér el a 256 8 bit-en, ezért azt várom, hogy ez R05 regiszter értéke 0 legyen. Fizika törvényeinek hála tényleg így is van, a kép jobb szélén pirossal található a regiszter értéke, és tényleg nulla lett. Ezen kívül megfigyeljük, hogy a Zero flag, valamint a Carry flag is egy értéket ad. Ez azt jelenti, hogy helyes a műveletvégzés. Mivel az $11111111b(=255d)+00000001b(=1d)=100000000b(=256)$ lenne, de a kilencedik számjegy már csak a Carry biten fog meglátszani, azaz ha még ezt össze akarnánk adni, ezt az 1-et tovább kéne vinni. (ezt később fel is használjuk)

II-B. Két 16 bites előjel nélküli szám összeadása

Az előző feladatohoz képest ez nem sokban tért el. Szerencsénkre a szimuláció képes kezelni a két byte-os integereket, amiket a mikrokontroller ”word”-nek nevez. Így itt a különbség hogy az utasítás után nem .B a szintaktika, hanem .W a használatos. Tehát miután a **MOV.W** paranccsal betöltöttük továbbra is az R04, és R05 regiszterekbe a két tetszőleges számunkat, az **ADD.W** instrukcióval összeadatjuk a két egészet.

Azért, hogy ezúttal látható legyen, hogy képes a nagyobb számokat is kezelni, 190-et és a 220-at adtuk össze. $190+220=410$, tehát nem várunk semmi problémát, és azt szeretnénk látni, hogy sikeres, és minden flag 0 legyen.

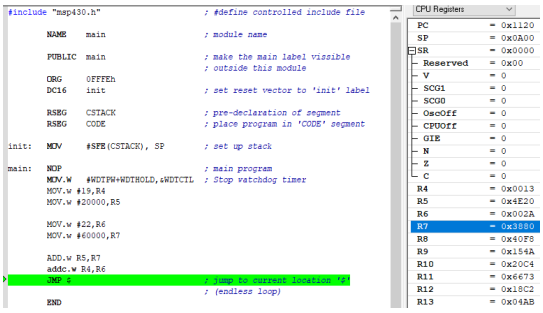


3. ábra. Két 16 bites előjelnélküli egész összeadása

A 3. ábrán látható jobb szélen az R05 regiszterben az összeadás eredménye, ami $0x019A$ hexadecimális szám, vagyis decimálisan $1\cdot16^2+9\cdot16^1+10\cdot16^0=410$, tehát helyes eredményt kaptunk, és a status biteket megtekintve valóban az elvárt viselkedést tanúsította, és mind a négy zérus.

II-C. Két 32 bites előjel nélküli szám összeadása

Ez a feladat már igényelt egy kis végiggondolást. Mivel a szimuláció és a mikrokontroller csak 8 és 16 bites számokat tud kezelni alapból, ezért magunk kellett megírni, hogy össze tudja ezeket adni. Az alapkoncepció, hogy egy 32 bites számot két 16 bitesen akarunk eltárolni, első 2 byte-on a szám első felét, (magasabb helyiértékek), második két byte-on a szám kisebb helyiértékű felét.

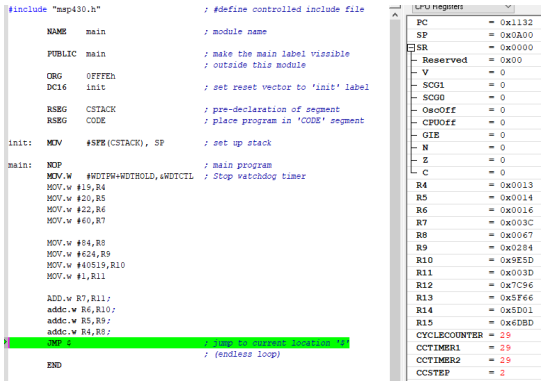


4. ábra. Két 32 bites előjelnélküli egész összeadása

A 4. ábrán az látható, hogy négy regiszterbe töltünk értékeket, és utána két lépésben adunk össze. Először a kisebb helyiértékű word-ket adjuk össze, majd a második összeadást már egy **ADDC.W** utasítással hajtjuk végre. Ez azért szükséges, mert ha akkora két számot akarnánk összeadni, hogy tovább kéne vinni értéket, akkor azt a magasabb helyiértékű word-ben lévő számnak át kell adni. Vegyük a példát, amit megcsináltunk. Először a 20000-et és a 60000-et szeretnénk összeadni. Erről biztosan tudjuk, hogy fel kell villants a Carry flaget, mivel két byte-on maximálisan ábrázolható szám a $2^{16}=65536$, ami pedig kisebb mint 80000. A következő összeadás hozzásadja az előző Carry értéket, és úgy összegzi a nagyobb helyiértékű word-t, amiben a 19 és a 22 szerepel. Tehát a tényleges számok, amiket így beírtam az $19\cdot2^{16}+20000=1265184$, illetve $22\cdot2^{16}+60000=1501792$. Ezen két szá összege $1265184+1501792=2766976$, ami hexadecimálisan ábrázolva $0x2A3880$. Ha megnézzük az R06 és R07 regiszterek értékeit, éppen ez a szám található benne. Nem csalás, nem ámítás, tényleg jól működik.

II-D. Két 64 bites előjel nélküli szám összeadása

Ez az előző feladathoz képest ha már megértettük, hogy mit is csinálunk, gyerekjáték. Ugyanúgy először Carry nélkül adjuk össze, majd utána viszont Carry értékeit elkérve.

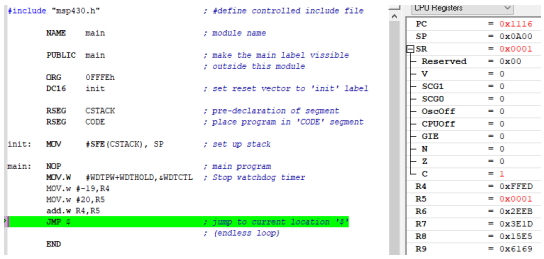


5. ábra. Két 64 bites előjelnélküli egész összeadása

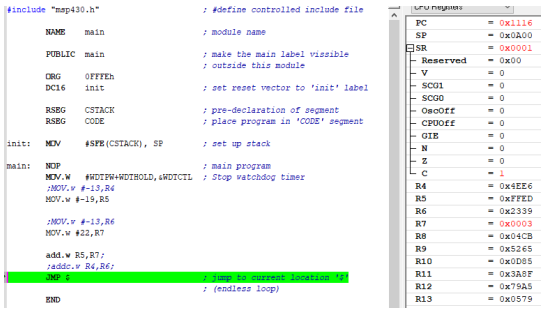
Éppen ezt tesszük az 5. ábrán látható összeadásban. Lényeges különbség, hogy most nem 2, hanem 4 word-ből áll egy 64 bites szám, de az összeadás menete változatlan, leszámítva, hogy kétszer annyi összeadást végzünk el. Nézzük is meg, hogy jó eredményt kapunk-e. $(19 \cdot 2^{48} + 20 \cdot 2^{32} + 22 \cdot 2^{16} + 60) + (84 \cdot 2^{48} + 624 \cdot 2^{32} + 40519 \cdot 2^{16} + 1) = 28994691217031229d = 0x6702849E5D003D$

Ha megnézzük az R08, R09, R10, R11 regisztereket, és így sorban kiolvassuk, pont ezt az eredményt kapjuk, tehát megint nem csalódtunk.

II-E. Összeadások előjeles egészekkel



6. ábra. Két 8 bites előjeles egész összeadása



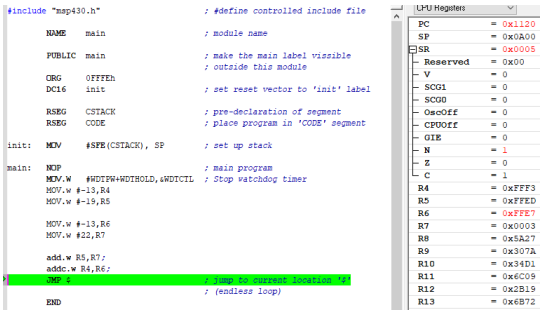
7. ábra. Két 16 bites előjeles egész összeadása

Azt bettük észre, hogy minden próbát áll, és azt végzi el, amit mondunk neki, ha meg nem úgy, ahogy mi gondoltunk, akkor sem a gép a hibás, hanem a gondolatmenetünk, de annál izgalmasabb volt kitalálni, hogy mi a hiba a végiggondolásban.

III. FELADAT: KIVONÁSOK ELVÉGZÉSE

A kivonások elvégzésére kétszer kettőféle parancsunk is rendelkezésre állt:

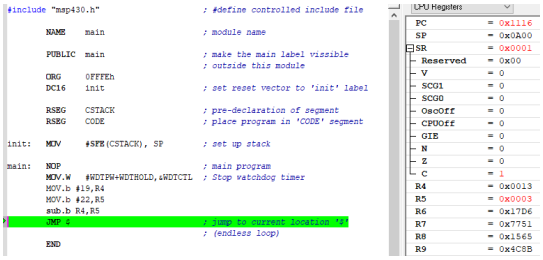
SUB.B SUB.W SUBC.B SUBC.W



8. ábra. Két 32 bites előjeles egész összeadása

III-A. Két 8 bites előjel nélküli szám kivonása

A szintaktika nagyon hasonló az összeadáshoz, a különbség csupán annyi, hogy **ADD** helyett a **SUB** parancsot használjuk. A megfontolás mögötte azonban már egészen más. A kivonás is valójában egy összeadás. Azért, hogy a kivonás összeadásként is elvégezhető legyen, megalkották a kettes komplemens előjeles számábrázolást (1 egyenlet). Itt az első bit-et ”beáldozzuk” előjelbitnek, így az ábrázolási tartományunkat eltoltuk -128 és 127 közé. Ami még érdekes, hogy pont visszafelé növekednek a számok, tehát ha pl 8 biten ábrázoljuk a -1 -et, az 11111111 lesz, de cserébe ha ehhez hozzáadjuk a 00000001 -et, akkor tényleg nullát kapunk, meg egy Carry flaget (és egy Zero flaget is).

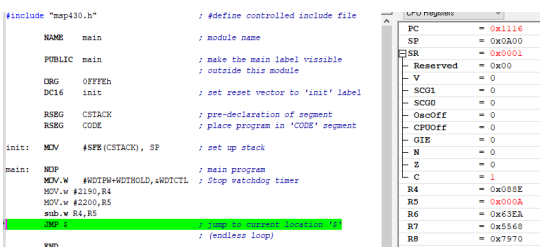


9. ábra. Két 8 bites előjelnélküli egész kivonása

A 9. ábrán látható példában egy ennél kézzelfoghatóbb kivonást végeztünk el, 22-ből kivontuk a 19-et, aminek 3-nak kell lenni, és ha megnézzük, tényleg 3-mat kaptunk. Ami még érdekes, hogy a Carry flag jelez. A Borrow bit jelenti, hogy kivonáskor az előtte levő számjegytől kölcsön kéne kérni egyet, ami persze nála mínusz egyet jelentene. Szóval itt a Borrow 0, vagyis nincs baj, és tényleg, mivel nem mentünk a kivonással negatívba.

III-B. Két 16 bites előjel nélküli szám kivonása

Szintén nagyon hasonló az előzőhöz, itt is a különbség csupán, hogy .B helyett .W -t használunk.



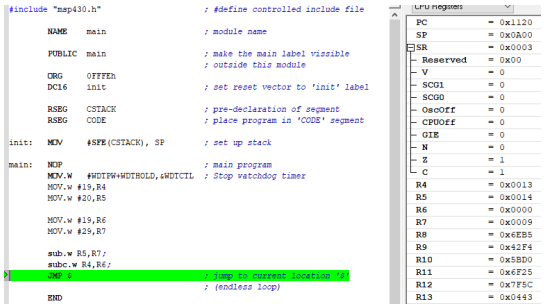
10. ábra. Két 16 bites előjelnélküli egész kivonása

Amint a 10. ábrán megfigyelhető, itt is egy könnyebben értelmezhető példát vettünk, $2200 - 2190 = 10d = 0xA$,

és lás csodát, tényleg az van a regiszterben, és a Carry továbbra is 1, ami helyes, mivel most sem mentünk negatív tartományba, tehát nem kell kölcsönkérni helyiértéket.

III-C. Két 32 bites előjel nélküli szám kivonása

Ennél a feladatnál már több átgondolásra van szükség. Igaz, pont ugyanúgy történik a számolása, mint az **ADDC.W** esetén, csak most **SUBC.W** -t kell a helyére írni.

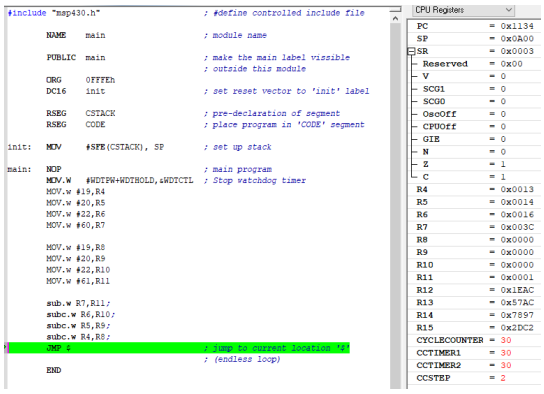


11. ábra. Két 32 bites előjelnélküli egész kivonása

Itt is egy könnyű példát hoztam, és az eredmény tényleg 9, mint ahogyan elvártuk tőle, még a Carry is jelez, úgyhogy tudjuk, hogy jól csináltuk.

III-D. Két 64 bites előjel nélküli szám kivonása

Itt a megfontolás semmiben sem tér el az előző feladattól, a megvalósítás is csupán annyiban, hogy megduplázzuk a felhasznált regiszterek számát, illetve a kivonások számát.

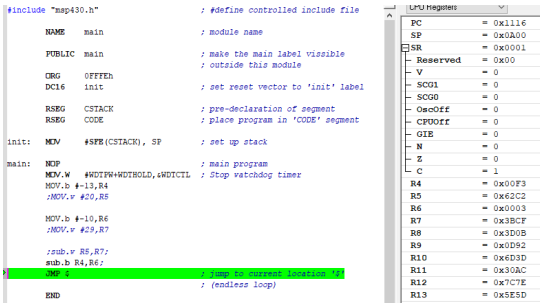


12. ábra. Két 64 bites előjelnélküli egész kivonása

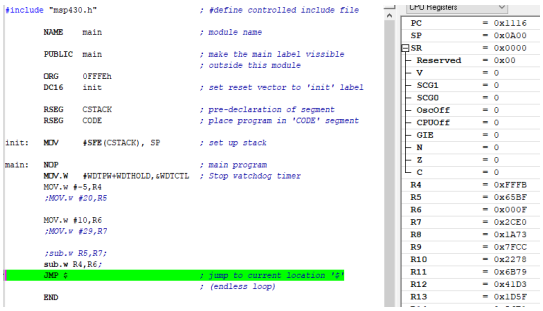
A 12. ábrán egy még egyszerűbb példával demonstráltuk a működését, mivel az előző esetben sikerült akkora számot összeadni, hogy még a Matlab se akarta pontosan kiszámolni. Jól látható, hogy 1 a végeredmény, és ezt is vártuk, illetve 1 a Carry, úgyhogy tényleg rendesen működik. (illetve a Zero flag is 1, mivel utoljára a 19-et vontam ki a 19-ből, ami pedig nulla)

III-E. Kivonások előjeles egészekkel

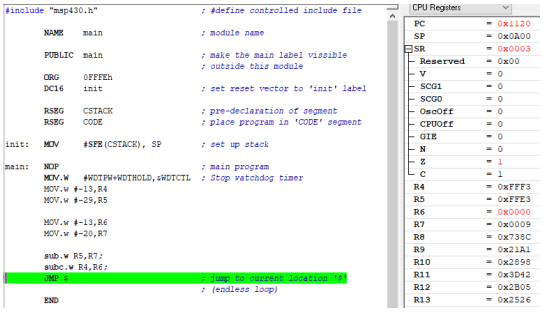
A tapasztalat, hogy ezeknél a számoknál is teljesen úgy viselkedik, ahogy elvárt. Mivel minden kivonást összeadásként (kettes komplementer) végez el, és egy negatív szám kivonása az ugyanúgy csak az a művelet, hogy visszanegálja pozitív számmá, és ha van Carry, azt hozzáadja (illetve kivonásnál ugye Borrownak nevezzük, így ha nincs Borrow, akkor rendes kettes komplementert képez).



13. ábra. Két 8 bites előjeles egész kivonása



14. ábra. Két 16 bites előjeles egész kivonása



15. ábra. Két 32 bites előjeles egész kivonása

LEZÁRÁS

Összességében nem is feltétlenül az összeadogatás, kivonás fogott meg, hanem az, hogy amit tavaly Naszynál vettünk Bevezetés a számítástechnikába órán, azt most itt élesbe is megnézhettük, ráadásul tényleg nagyon hasonló módon a Little Man Computerhez. Ezen kívül jó, hogy a labor és egyben az Assembly végett kicsit jobban értem az ALU működését. Élveztem látni, és utánagondolni vagy olykor számolni is, hogy miért kaptuk azt a flaget, amit. Egyedül azt sajnálom, hogy az Overflow flagre nem néztünk meg példát, de ami késik, nem múlik.

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