Aritmetikai utasítások I.

Az értékadó és aritmetikai utasítások során a címzési módok különböző típusaira látunk példákat. A 8086/8088-as mikroprocesszor memóriája és regiszterei a little endian tárolást követik, vagyis az alacsonyabb címen az alacsonyabb helyiértékű bájt helyezkedik el. Az assembly nyelvben a negatív számok ábrázolására a kettes komplemens számábrázolást használják. Az egyes műveletek beállítják a `STATUS` regiszter bitjeit is:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	O	D	I	T	S	Z	-	A	-	P	-	C

Overflow	Előjeles túlcsordulás
Direction	a string műveletek iránya, 0: növekvő, 1: csökkenő
Interrupt	1: megszakítás engedélyezése, 0: tiltása
Trap	1: „single step”, 0: automatikus üzemmód
Sign	az eredmény legmagasabb helyiértékű bitje (előjel)
Zero	1, ha az eredmény 0, különben 0
Auxiliary Carry	átvitel a 3. és 4. bit között (decimális aritmetika)
Parity	az eredmény alsó 8 bitjének paritása
Carry	átvitel előjel nélküli műveletekhez

Bevezetésképpen megnézzük a `MOV` utasítás működését. A `MOV` utasítás nem aritmetikai utasítás, viszont fontos megemlíteni a további példák miatt. Az adatmozgató utasítások nem módosítják a flag-eket.

`MOV:` A `MOV` operációs kód után két operandust kell megadni. Az első operandusába képződik le az eredmény.
Példa:

```
MOV AX, 0006h          ; AX = 0006h
MOV AX, -0F10h         ; AX = F0F0h    kettes komplemens
```

Tegyük fel, hogy az adatszegmensünkben az alábbi tartalom van:

```
DS:0000 CD 20 FF 9F 00 9A F0 FE
```

`MOV AL, [0]`	`; AL = CD`	nem szokás így hivatkozni, de szabályos
`MOV AX, [0]`	`; AX= 20CD`	little endian

Összeadás

`ADD:` Az `ADD` utasítás az eslő operandushoz hozzáadja a második operandus értékét. Az `ADD` utasítás módosítja a `C`, az `O` és a `S` flag értékét a `STATUS` regiszterben (mást is, de azzal most nem foglalkozunk).
Példa:

```
MOV AX, 0005h          ; AX=0005h
ADD AL, 01h            ; AX=0006h
```

Térjünk vissza a fenti DS regiszter tartalomhoz:
`DS:0000 CD 20 FF 9F 00 9A F0 FE`

`MOV AX, 0005h`	`; AX=0005h`	
`ADD AL, [01h]`	`; AX=0025h`	DS:0001h = 20

```
MOV AX, 0005h          ; AX=0005h
MOV BX, 0006h          ; BX=0006h
ADD AX, BX             ; AX=000Bh
```

`MOV AX, 0005h`	`; AX=0005h`	
`MOV BX, 0006h`	`; BX=0006h`	
`ADD AX, [BX]`	`; AX=FEF5h`	DS:0006h = F0, de word-öt olvasunk!!

Az összeadás során ügyelni kell a túlcsordulásokra. 16 biten csak a [-32768 ; 32767] intervallumban tudunk tárolni előjeles számokat. Tekintsük az alábbi példát:

Decimális	Bináris	16 bites előjeles (decimális) eredmény
21348	0101 0011 0110 0100	21348
+ 25841	+ 0110 0100 1111 0001	+ 25841
47189	0 1011 1000 0101 0101	- 18347
	C S	

A fenti példában az előjeles számábrázolás miatt nem a valódi értéket kapjuk vissza 16 biten. A 2-es komplemenses számábrázolásban az első bit az előjel bit, amely ha 1, akkor negatív számról van szó. A fenti számítás assemblyben az alábbi kódrészlettel valósítható meg (hexadecimális értékeket használva):

```
MOV AX, 5364h          ; AX=5364h
ADD AX, 64F1h          ; AX=B855h      az AX-ben tárolt érték
                                       negatív, S:1 O:1
```

Két 16 bites számnál gondot okozhat az átvitel is. Lássuk az alábbi példát:

Decimális	Bináris	16 bites előjeles (decimális) eredmény
-28672	1001 0000 0000 0000	-28672
+ -30720	+ 1000 1000 0000 0000	+ -30720
-59392	[1] [0]001 1000 0000 0000	+ 6144
	C S	

Az eredményben a Carry értéke 1-es, az előjel pedig pozitív. Ugyanez a példa egy assembly kódrészletben:

```
MOV AX, 9000h          ; AX=9000h
ADD AX, 8800h          ; AX=1800h      az AX-ben tárolt érték
                                       pozitív, S:0 O:1 C:1
```

Ha egy 8 bites és egy 16 bites számot szeretnénk összeadni előjelhelyesen, akkor assemblyben a 8 bites operandust 16 bitesre kell konvertáltunk.

`CBW:` Az `AL` 8 bites regiszterben tárolt értéket 16 bitesre konvertálja (előjelhelyesen).

Példa:

```
MOV AL, -10d           ; AL=0F6h
MOV BX, 2525d          ; BX=99DDh
CBW                    ; AX=FFF6h
ADD AX, BX             ; AX=09D3h
```

Szorzás

Assemblyben megkülönböztetünk előjeles és előjel nélküli szorzást.

`MUL:` Előjel nélküli szorzás. A szorzás műveletét úgy alakították ki, hogy két 8 bites, vagy pedig két 16 bites számot tudjunk szorozni. A `MUL` utasításnak egyetlen operandusa van, amely nem lehet közvetlen operandus (vagyis konstans). Két szám szorzatakor egy egyik (első) operandus 8 bites számok esetén az AL-ben tárolódik, 16 bites számok esetén pedig AX-ben. Két 8 bites vagy 16 bites szám szorzata ritkán fér el 8 illetve 16 biten, ezért 8 bites operandusok szorzata AX-ben keletkezik, a 16 bites operandusok szorzata pedig DX:AX-ben keletkezik. Azt, hogy egy számot 8 bitesnek vagy 16 bitesnek kell tekinteni, a `MUL` utasítás operandusa határozza meg. A `MUL` utasítás csak a biteket nézi, a számábrázolást nem. A `MUL` utasítás hatással van az `O` és `C` flag-ekre.

`IMUL:` Előjeles szorzás. Működése hasonló a `MUL` utasításhoz.

Példák:

Az előjel nélküli szorzásnál nem vesszük külön figyelembe az előjelet. Példaképpen szorozzuk össze a -2-t és a 3-at. Mindkét szám ábrázolható 8 biten, a -2-t kettes komplemens formában kell ábrázolni.
`-2 = FEh`, `3 = 03h`.

F	E	·	0	3
0	0			
2	F	A		
2	F	A	h	

Az előjeles szorzás úgy végzendő el, hogy a számok abszolút értékét és előjelét külön szorozzuk össze. Szorozzuk össze a -2-t és a -520 decimális számokat. A szorzást most hexadecimális alakban végezzük el (decimálisban is lehetne):
`2d = 0002h` és `520d = 0208h`

0	0	0	2	·	0	2	0	8
0	0	0	0					
	0	0	0	4				
		0	0	0	0			
			0	0	1	0		
0	0	0	0	4	1	0	h	

Az előjel pedig 0 lesz, tehát `DX:AX = 0000 0410 h`. Az eredmény decimális alakban 1040 lesz.

```
MOV AX, 0102h          ; AX=0102h
MOV CX, 0003h          ; CX=0003h
MUL CL                 ; AX=0006h      a szorzás során
                                       csak AL-et és CL-et
                                       vettük figyelembe
```

```
MOV AX, 0102h          ; AX=0102h
MOV CX, 0003h          ; CX=0003h
MUL CX                 ; DX:AX=0000 0306h
```

```
MOV AX, 0202h          ; AX=0202h
MOV SI, 0003h          ; DS:0003h = 9F
MUL byte ptr [SI]      ; AX=013Eh
```

```
MOV AX, 0602h          ; AX=0602h
MOV SI, 0003h          ; DS:0003h = 9F 00h
MUL word ptr [SI]      ; 0602h * 9F00h = DX:AX=03BB 3E00h
```

```
MOV AL, -2d            ; AL=0FEh
MOV CL, 3d             ; CL=03h
IMUL CL                ; AX=FFFAh
```

Ha `AL` egy 8 bites operandust tartalmaz, amelyet egy 16 biten tárolt értékkel szeretnénk szorozni, akkor a szorzás elvégzése előtt `AL` tartalmát előjelhelyesen 16 bitesre kell konvertálni a `CBW` (Convert Byte to Word) utasítással.

```
MOV AL, -2d            ; AL=0FEh
CBW                    ; AX=FFFEh
MOV CX, -520d          ; CX=FDF8h
IMUL CX                DX:AX=0000 0410h = 1040d
```

Feladatok

1. Adjuk össze binárisan az 53-at és a 18-at! Vizsgáljuk meg az előjelet és a túlcsordulást!

2. Adjuk össze binárisan az 53-at és a 83-at! Vizsgáljuk meg az előjelet és a túlcsordulást!

3. Adjuk össze binárisan a -53-at és a -18-at! Vizsgáljuk meg az előjelet és a túlcsordulást!

4. Adjuk össze binárisan a -53-at és a -83-at! Vizsgáljuk meg az előjelet és a túlcsordulást!

5. Adjuk össze binárisan az 53-at és a -18-at! Vizsgáljuk meg az előjelet és a túlcsordulást!

6. Adjuk össze binárisan az 18-at és a -18-at! Vizsgáljuk meg az előjelet és a túlcsordulást!

7. Szorozzuk össze a -4-et és a -3-at előjel nélküli szorzással (`MUL`)!

8. Szorozzuk össze a -14-et és a 7-et előjeles szorzással (`IMUL`)!