Shortcut kettes komplemens ábrázolású hexa számok előjelének megállapítására: akkor és csak akkor negatív, ha a legnagyobb helyiértékű számjegy >= 8. (Mert $8_{16} = 1000_2$.) Figyeljünk az ábrázolás szélességére, 16 biten például B7Fh-t 0B7Fh-nak kell kezelni.

Horner elrendezés: Leírom a legmagasabb helyiértékű számjegyet. Ezután minden lépésben amit eddig leírtam, beszorzom a számrendszer alapjával, és hozzáadom a következő számjegyet. Ezt addig csinálom, amíg hozzá nem adtam a legkisebb helyiértékű számjegyet.
Például a 754_8 Horner elrendezésben: (7 * 8 + 5) * 8 + 4

Regiszterek
Általános célú (néhány speciális esetet kivéve): AX (accumulator), BX (base), CX (counter), DX (data)
Vezérlő: SI (source index), DI (destination index), BP (base pointer), SP (stack pointer), IP (instruction pointer), FLAGS (STATUS)
Szegmens: CS (code), DS (data), ES (extra), SS (stack)
Szegmens_reg : offset → fizikai cím számítás: szegmens_reg * 16 + offset
(Az eddigi anyaghoz) **fontosabb FLAGS bitek:** O (overflow, előjeles túlcsordulás), C (carry, átvitel a legfelső bitről), Z (zero, az eredmény 0), S (sign, az eredmény legmagasabb helyiértékű bitje, ami többnyire az előjele is kettes komplemens ábrázolásban)

Címzési módok (példákkal)
Adat terület: alapértelmezett szegmens a DS

- Közvetlen operandus = immediate = kódba épített adat = konstans.
  Pl. `MOV AX, 9 ;` AX értéke 9 lesz
- Regiszter címzés: regiszter tartalmára hivatkozunk.
  Pl. `MOV AX, BX ;` AX értéke BX értéke lesz
- Direkt memóriacímzés: a kódterületen közvetlenül megadott memóriaterületre hivatkozunk. Többnyire címkével adjuk meg, de közvetlenül a kódba is „bedrótozhatjuk”. Pl.: címkével `MOV AX, SZO` ; AX értéke a SZO változót jelentő memóriaterületen lévő szó lesz, számmal megadva `MOV AX, DS:[01F4h]` ; AX értéke a DS szegmensen belüli 1F4h eltoláson lévő szó lesz.
- Indexelt címzés: egy eltolás (displacement) és egy indexregiszter (SI vagy DI, esetleg BX) összegéből előáll egy offset. A memóriában ezen az offseten található értékre hivatkozunk. A displacement lehet negatív.
  Pl. `MOV AX, 122h[SI]` ; AX értéke a DS szegmensen belüli SI+122h eltoláson lévő szó lesz.
- Regiszter-indirekt címzés: indexelt címzés eltolás (displacement) nélkül. A regiszter lehet SI, DI, vagy BX.
  Pl. `MOV AX, [SI]` ; AX értéke a DS:SI által mutatott memóriaterületen lévő szó lesz.
- Bázis-relatív (bázisindex) címzés: indexelt címzés + bázis. Az offsetet egy displacement, a BX regiszter, és egy indexregiszter (SI vagy DI) összegéből kapjuk. Displacement nélkül is használható.
  Pl.: `MOV AX, 8[BX][SI]` ; AX értéke a 8+BX+SI offseten lévő szó lesz.

Stack (verem) terület: alapértelmezett szegmens az SS

- Bázis-relatív címzést használunk, BX helyett BP-vel. Index nélkül is használható.

Kód terület: alapértelmezett szegmens a CS. A következő utasítás címe: CS:IP. Az utasítások végrehajtásának elején kap értéket. Ugrás = IP (és ha kell, CS) módosítása.

- IP relatív címzés: IP-hez hozzáadódik az előjeles 8 (vagy adott esetben 16) bites előjeles érték.
- Direkt utasítás címzés: közvetlenül megadott, konkrét cím kerül CS:IP-be.
- Indirekt utasítás címzés: bármilyen címzési móddal megadott szóban vagy dupla szóban tárolt címre történő ugrás.

Ugrástípusok:

- SHORT: 8 bites közvetlen IP relatív értéket adunk meg
- NEAR: közvetlen megadás esetén 16 bites IP relatív értéket adunk meg, indirekt címzés esetén az érték IP régi tartalmától függetlenül beíródik IP-be.
- FAR: másik szegmensbe ugrás. CS:IP ← 32 bites operandus.

A több érték összegén alapuló címzési módokban a tagok sorrendje felcserélhető, és [] helyett + is írható, például `9[BX][SI]` ugyanaz, mint `[9+SI+BX]`. Displacementeknél a mínusz jel is használható, például `-9[SI]` ugyanaz, mint `[SI-9]`.
Szegmens override (explicit szegmens megadás): a címzés elé írjuk, `DS:`, `SS:`, `ES:`, vagy `CS:`. Bedrótozott direkt memóriacímzésnél akkor is ki kell írni, ha az alapértelmezett szegmenst szeretnénk, lásd a példát.

Összeadás: **ADD** op1, op2
Működése: op1 ← op1 + op2
Hatással van a Z, C, O, S flagekre:

- Z ← 1, ha az eredmény 0. Z ← 0, egyébként.
- C ← a legnagyobb helyiértékű bitről keletkezett átvitel
- O ← 1, ha túlmentünk az előjeles ábrázolás határain, azaz látszólag két pozitív szám összege negatív, vagy két negatív szám összege pozitív lett.
- S ← az eredmény legfelső bitje

Előjelkiterjesztés: **CBW**
0 operandusú.
Működése:

- Ha AL előjele (legnagyobb helyiértékű bitje) 1: AH ← FFh
- Ha AL előjele (legnagyobb helyiértékű bitje) 0: AH ← 00h

A mnemonik a **C**onvert **B**yte to **W**ord rövidítése.

Előjel nélküli szorzás: **MUL** op
Működése:

- Ha op 8 bites: AX ← AL * op
  C ← 0 és O ← 0, ha AH == 0, egyébként C ← 1, és O ← 1
- Ha op 16 bites: DX:AX ← AX * op
  C ← 0 és O ← 0, ha DX == 0, egyébként C ← 1, és O ← 1

C és O tehát akkor lesz 0, ha az eredmény „felső fele” 0, azaz az eredmény elfért az operandus eredeti méretén.
op nem lehet közvetlen operandus (konstans)

Előjeles szorzás: **IMUL** op
Működése (ugyanaz, mint a MUL-nál, csak előjelesen):

- Ha op 8 bites: AX ← AL * op
  C ← 0 és O ← 0, ha AH == 0, egyébként C ← 1, és O ← 1
- Ha op 16 bites: DX:AX ← AX * op
  C ← 0 és O ← 0, ha DX == 0, egyébként C ← 1, és O ← 1

C és O tehát akkor lesz 0, ha az eredmény „felső fele” 0, azaz az eredmény elfért az operandus eredeti méretén.
op nem lehet közvetlen operandus (konstans)

Adatmozgatás: **MOV** cél, forrás
Működése: cél ← forrás
Flageket nem változtat.

BYTE PTR, WORD PTR: típuskényszerítés. Hasonlít a C-s castolásra. Legtöbbször akkor kell, ha nem derül ki a kódból az operandus mérete. Például:

```
MUL [SI] ; Ez 8 vagy 16 bites szorzás? Mekkora értéket olvassunk?
MUL BYTE PTR [SI] ; Így már látszik, hogy bájtos (8 bites) művelet
MUL WORD PTR [SI] ; Így pedig wordös (16 bites)
```
