Adatmozgató utasítások

MOV: A **MOV** utasításnak két operandusa van. Adatot másol át a ***forrás*** operandusból a ***cél*** operandusba. Az utasítás szintaxisa:

MOV *cél, forrás*

A ***cél*** operandus tartalma felül íródik, míg a ***forrás*** operandus értéke nem változik (A C nyelvben ismert *cél = forrás* művelet megfelelője). Fontos, hogy a két operandus mérete meg kell, hogy egyezzen.

Példák:

```
MOV AX, 0006h   ; AX = 0006h. Az EAX regiszternek csak az alsó
                ; 16 bitje íródik felül!
```

```
MOV EAX, 6h     ; ekkor a teljes EAX regiszter értéke felül
                ; íródik és a 6-os értéket fogja tartalmazni
```

```
MOV EAX, -0F10h ; EAX = FFFFF0F0h lesz (kettes komplemens)
```

```
ADAT1 DD 912, 920, 928, 936, 944 ; adatszegmens tartalma
```

```
MOV ESI, offset ADAT1  ; ESI regisztert ráállítjuk az ADAT1-re
MOV EAX, [ESI]         ; EAX regiszter értéke 912 lesz
MOV AX, [ESI]          ; Helytelen! Méretbeli különbségek!
MOV AX, 70000d         ; A fordító jelezni fogja, hogy a
                       ; regiszterben nem fér el ekkora érték
```

Az előzőleg ismertetett, címzési módok anyagrészben részletesen tárgyalva vannak a további használati lehetőségek, az operandusokat tekintve.

MOVZX: A **MOVZX** (MOVe with Zero-eXtended) utasítás átmásolja a forrás operandus tartalmát a cél operandusba és kinullázza a felső 8 vagy 24 bitet, az operandusok méretétől függően (azaz nem előjelhelyesen terjeszti ki 16 vagy 32 bitre). Ezért ezt az utasítást csak pozitív értékek esetén alkalmazhatjuk. Szintaxis:

```
MOVZX cél, forrás
```

Három különböző alkalmazási lehetőség:

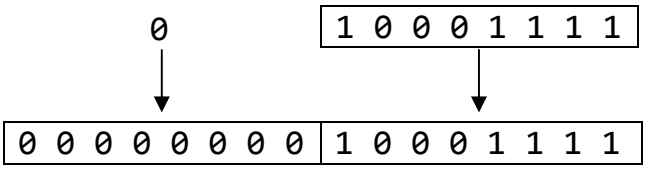
```
MOVZX reg32, reg/mem8

MOVZX reg32, reg/mem16

MOVZX reg16, reg/mem8
```

Példa:

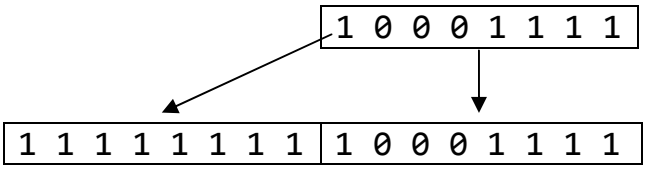
```
.data
var BYTE 10001111b
.code
MOVZX AX, var
```



MOVSX: A **MOVSX** (MOVe with Sign-eXtended) utasítás átmásolja a forrás operandus tartalmát a cél operandusba és **előjelhelyesen** kiterjeszti 16 vagy 32 bitre. Ez az utasítás már a negatív értékeket is jól kezeli. Szintaxisa és a variánsok megegyeznek a **MOVZX** utasításéval.

Példa:

```
.data
var BYTE 10001111b
.code
MOVSX AX, var
```



Hasonlóképpen működik mindkét fenti mozgató utasítás a további 2-2 variáns esetén is, azzal a különbséggel, hogy változik az operandusok mérete.

XCHG: Az **XCHG** (eXCHanGe data) utasítás megcseréli a két operandusának a tartalmát. Három különböző operandus elosztása lehet:

```
XCHG reg, reg
XCHG reg, mem
XCHG mem, reg
```

A két operandus méretének meg kell egyeznie és nem lehet közvetlen érték (konstans).

Példa:

```
MOV EAX, 10h   ; EAX = 10h
MOV EBX, 15h   ; EBX = 15h
XCHG EAX, EBX  ; EAX = 15h, EBX = 10h
```

PUSH: A **PUSH** utasítás először csökkenti az **ESP** regiszter értékét, majd bemásolja a forrás operandust a verembe. 16 bites operandus esetén az **ESP** értéke 2-vel csökken, 32 bites esetén pedig 4-gyel. Három formája van:

PUSH *reg/mem16*

PUSH *reg/mem32*

PUSH *const32*

POP: A **POP** utasítás először belemásolja a 16- vagy 32 bites cél operandusba a verem azon elemét, amelyre az **ESP** regiszter mutat, majd növeli az **ESP** értékét. Ha az operandus 16 bites, akkor az **ESP** értéke 2-vel nő, míg 32 bites operandus esetén 4-gyel. Változatai:

POP *reg/mem16*

POP *reg/mem32*

PUSHF: A **PUSHF** utasítás a **FLAGS** 16 bites regiszter tartalmát tölti be a verembe. Nincs operandusa.

POPF: A **POPF** utasítás a **FLAGS** 16 bites regisztert tölti fel az előzőleg kimentett **FLAGS** értékekkel. Nincs operandusa.

A **PUSHFD** és **POPFD** utasítások működése megegyezik a **PUSHF** és **POPF** utasítások működésével, azzal a különbséggel, hogy az **EFLAGS** 32 bites regiszter tartalma kerül kimentésre, illetve onnan visszaolvasásra.