Memória címzési módok

Jelen nayagrészben az Intel x86-os architektúrára alapuló 32 bites processzorok programozását tekintjük.

Egy program futása során (legyen szó a program vezérléséről vagy adatkezelésről) a program utasításai illetve egy utasítás argumentumai a memóriában találhatók. A memória-szervezési modell mondja meg azt, hogy miként és mekkora területhez férhetünk hozzá a memóriában tárolt adathoz. A legkisebb memória-egység, amelyet meg tudunk címezni 8 bit, vagyis 1 bájt. Gyakran azonban *szavas* címeket használunk. Egy szót (word) 2 vagy 4 bájton tudunk ábrázolni, dupla szavakat (double word vagy dword) pedig 4 vagy 8 bájton ábrázolunk.

A x86-os architektúrájú számítógépek memóriája szegmensekre van osztva. A szegmensek a memória adott méretű, összefüggő területei. A memória-szegmensek *báziscímei* (kezdőcímei) általában ismertek, és mivel a szegmensek összefüggőek, szegmensen belüli bájtokat a báziscíméhez képest tudjuk megadni. A szegmensen belüli bájtok távolságát a báziscímhez képest *offszet*nek hívjuk. A szegmensek megadásának számos módozata létezik, de az itt tárgyalt FLAT memóriamodellben a program futásához szükséges összes adat (programkódok, adatok, verem) egyetlen szegmensben tárolódnak. Ezen szegmensen belül az adatok elérésére a szegmensen belüli eltolással hivatkozhatunk.

A tárgyalt processzorcsalád regiszterei 32-bitesek, ebből kifolyólag a szegmensen belüli eltolást is 32 bites értékkel tudjuk megadni. Ez szegmensenkén 4 GB memória megcímzését teszi lhetővé.

A 32 bites x86-os architektúra regiszterei

Szegmens regiszterek (16 bites regiszterek)

- CS (Code Segment): utasítások címzéséhez
- SS (Stack Segment): verem címzéséhez

- DS (Data Segment): adat terület címzéshez
- ES (Extra Segment): másodlagos adatterület címzéshez
- FS : extra adatszegmens
- GS : extra adatszegmens

Vezérlő regiszterek (32 bites regiszterek)

- EIP (Extended Instruction Pointer): az éppen végrehajtandó utasítás logikai címét tartalmazza a CS által mutatott szegmensben
- ESP (Extended Stack Pointer): a verem tetejére beírt adat logikai címére mutat az SS által mutatott szegmensben
- EFLAGS: a processzor állapotát jelző regiszter
- EBP (Extended Base Pointer): a verem indexelt címzéséhez használatos
- ESI (Extended Source Index): a kiindulási adat terület indexelt címzéséhez használatos
- EDI (Extended Destination Index): a cél adat terület indexelt címzéséhez használatos

Minden 32 bites regiszter első Extended tagja azt jelzi, hogy a regiszter egy korábbi 16 bites regiszter kiterjesztése. A kompatibilitás megőrzésére a regiszterek alsó 16 bitje külön is nevesíthető.

Általános regiszterek 32-bites regiszterek.

A Regiszterek általánosan is felhasználhatóak, de egyes utasításokban speciális felhasználásuk/szerepük is lehet.

- EAX (Extended Accumulator): Szorcás, osztásnál van szerepe.
- EBX (Extended Base Register): Címző regiszter
- ECX (Extended Counter): Gyakran használjuk a számlálások vezérléseknél
- EDX (Extended Data Register): Szorzás, osztás, I/O műveletekben használatos

Mint a vezérlő regiszterek esetében, az általános regiszterek is a korábbi 16 bites regiszterek kiterjesztett változatai, és (az 'Extended' jelölő elhagysával) van lehetőség az alső 16 bit külön történő elérésére is. További kiegészítés, hogy az also 16 bit két fele külön külön is megnevezhető.

regiszter	felső bájt	alsó bájt	
AX	AH	AL	Accumulator (szorzás, osztás)
BX	BH	BL	Base Register (címző regiszter)
CX	CH	CL	Counter Regiszter (számláló)
DX	DH	DL	Data Register (szorzás, osztás, I/O)

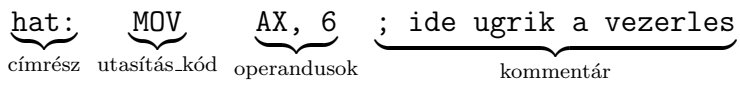
Címzési módok assembly-ben

Assembly programozási nyelvben egy utasítás a következő sémát követi:

`címrész operációs_kód operandusok kommentár`

- címrész: egyes adatok illetve utasítások szimbólikus jelölése
- operációs_kód: mnemonic, az utasítás, művelet megnevezésére szolgál
- operandusok: az utasítás paraméterei
- kommentár: A program jobb olvashatóságát és érthetőségét teszi lehetővé, de nincs hatása a program működésére

Példa:



A példában szereplő `MOV` utasítás két operandusú adat másoló művelet.

- A második operandusának értékét másolja át az első operandusba.
- Az operandusai lehetnek regiszterek, memóricímek, vagy konstansok, de memóriacím legfeljebb az egik lehet a kettő közül.
- A két operandus által jelölt adatterületek méretének megegyszerőnek kell lennie.

Adat terület címzés

A `MOV` utasítást- mint példát használva az adatokat az alábbi módokaon érhetjük el.

Kódba épített adat Az adatot közvetlenül a regiszterbe írjuk. Ennél a címzési módnál csak konstansokat tudunk megadni második operandusként.

Formátum: `regiszter, konstans`
Például:

- `MOV AX, 6`
- `MOV AX, 06F2h`

Direkt memória címzés A címrészen az operandus logikai (offszet) címét adjuk meg, nem az adatot.

Formátum: `regiszter, memóriacím`
Például:

- `MOV EAX, DSZO` *, ahol a DSZO egy 4 bytos változót jelöl*
- `MOV AX, SZO` *, ahol a SZO egy 2 byteos változót jelöl*
- `MOV AL, KAR` *, ahol a KAR egy 1 byteos változót jelöl*

Ügyelni kell arra, hogy a két operandus mérete összhangban legyen egymással. 32 bites geiszterhez csak 32 bites, 16 bites regiszterhez csak 16 bites (word), 8 bites regiszterhez csak 8 bites változókat használhatunk!

Direkt címzésnél megadhatunk második operandusként egy regisztert is. Akár a egy regisztert, akár egy logikai címet adunk meg, mindkét esetben a memóriacímen tárolt adat változhat, viszont a cím azonos marad.

Regiszter indirekt címzés Az operandusban a címet egy regiszter tartalmazza. Az ilyen módon megadott címet mutatónak hívjuk. Ebben a címzési módnál csak az ESI, EDI és EBX regiszter használható.

Formátum: `regiszter, [regiszter]`
Például:

- `EAX, [SI]` , *az ESI-ben tárolt értéket használjuk*
- `EAX, [BX]` , *a EBX-ben tárolt értéket használjuk*
- `EAX, [DI]` , *a EDI-ben tárolt értéket használjuk*

Indexelt címzés Az operandusban megadott 8 vagy 16 bites számot (eltolás) hozzáadjuk az index-regiszter (SI vagy DI) tartalmához, így alakul ki a logikai cím.

Formátum: `regiszter, szám[index-regiszter]`
Például:

- `EAX, [ESI]` , *itt csak az SI-ben tárolt értéket számítjuk*
- `EAX, 10h[ESI]` , *SI tartalmához hozzáadjuk a 10h konstanst*
- `EAX, -10h[ESI]` , *SI tartalmához hozzáadjuk a -10h konstanst*

Az indexelt címzési mód ismerősebbnek tűnhet, ha az eltolásértéket egy tömb kezdőcíme adja.
Például:

- `EAX, TOMB[ESI]` , *A TÖMB kezdőcíméhez képest az ESI értékével eltolt helyről olvasulnk.*

Bázis relatív címzés Az operandusban megadott logikai címet úgy kapjuk meg, hogy az eltolás értékét, az ESI vagy EDI valamelyikének értékét, valamint a EBX regiszterben tárolt értéket összeadjuk.

Formátum: `regiszter, szám[regiszter][regiszter]`
Például:

- `EAX, 10h[ESI][EBX]`
- `EAX, [EBX][EDI]`
- `AX, [EBX + ESI + 10h]`

Verem terület címzés

A verem (stack) terület címzése bázis relatív címzéssel történik, azzal a különbséggel, hogy EBX helyett EBP-t használjuk. A fizikai cím meghatározásához nem DS-t, hanem SS lesz a szegmens regiszter (vagyis a verembl olvasunk indexelten).

Program terület címzés

Egy-egy utasítás során az IP értéke az utasítás hosszával növekszik és a soron következő utasításra mutat a CS-ben. Bizonyos vezérlési szerkezetekben (pl. ciklusok, feltételek) az IP értékének megadásával a kívánt helyen folytathatjuk a program futását.

IP relatív címzés Az EIP (már módosult) pillanatnyi értékéhez hozzáadódik egy 32 bites előjeles operandus. A programkódokban bizonyos kódrészletek kezdő utasítását címkével láthatjuk el, ha azt szeretnénk, hogy egy adott feltétel esetén ott folytatódjon a program futása. A feltételes vezérlés átadás és a ciklus utasítás mindig ilyen címzési móddal valósul meg.

Például:
`JMP CIKLUS` , *ahol CIKLUS egy címke az assembly forráskódban*

Direkt utasítás címzés Ez a címzési mód közeli (NEAR) vagy távoli (FAR) vezérlés átadásoknál játszik szerepet. A vezérlés átadás közeli, ha a programkód ugyanabban a szegmensben folytatódik tovább és távoli szegmensváltás esetén. Közeli vezérlésátadás esetén a 16 bites operandus lesz az EIP új tartalma, távoli vezérlésátadásnál pedig az IP és a CS tartalma is megváltozik. Ilyen vezérlés lehet például az eljáráshívás.

Például:
`CALL ELJARAS` , *az ELJARAS nevű eljárás hívása*

Indirekt utasítás címzés Bármilyen címzési móddal megadott szóban vagy dupla szóban tárolt címre történő vezérlés átadás.

Például:

- `JMP EAX`
- `JMP [EBX]`

Feladatok

Adott az alábbi módon definiált adat szegmens.

```
.data

; Bájt, szó és duplaszó területfoglalás, 5 kezdoértékkel (decimális)
SZAM1 DB 5
SZAM2 DW 5
SZAM3 DD 5

; Ugyanaz, mint fent, csak újabb kulcsszavakkal
SZAM4 BYTE 5
SZAM5 WORD 5
SZAM6 DWORD 5

; Negatív számértékek
SZAM7 BYTE -5
SZAM8 WORD -5
SZAM9 DWORD -5

; További példák (hexadecimális)
SZAM10 BYTE 12h
SZAM11 WORD 34h
SZAM12 DWORD 56h

; Adatterület foglalása (tömb-szeru)
ADAT1 DW 912, 920, 928, 936, 944
ADAT2 DB 11, 15, 20
ADAT3 DW 1912, 1920, 1928, 1936

; 15 elemből álló 1D tömb.
; 3x5 méretu 2D mátrixként értelmezzük majd.
MATRIX1 DW  0,  0,  0,  0,  0
Matrix1RowSize = ( $ - MATRIX1 )
DW  0, 48, 92, 48,  0
DW 0,  0,  0,  0,  0
```

1. Mi a szegmensen belüli eltolása az alábbi értékeknek, feltéve, hogy az adatok a szegmens elején (a 0 címen) kezdődnek, a sorrendjük a memóriában megegyezik a definiálásuk sorrendjével, és közöttük szabad terület nincs.

   - SZAM1
   - SZAM4
   - SZAM5
   - SZAM9
   - ADAT1 3. eleme-ja
   - ADAT3 utolsó eleme
   - MATRIX1 2. sorának 4. eleme.

2. Írj kódrészletet, ami az alábbi elvégzi az alábbi adatmozgatásokat. A tömb indexek [sorindex, oszlopindex] formában vannak megadva, és 0-tól indulnak.

   - AL ←SZAM1
   - AX ←SZAM2
   - EAX ←SZAM3
   - AX ←SZAM10
   - EAX ←SZAM12
   - EAX ←SZAM12
   - AX ←ADAT1[3]
   - AX ←MATRIX1[2,1]
   - EAX ←MATRIX1[0,2]