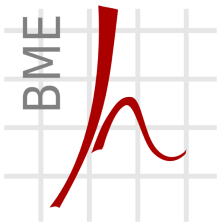
Memória védelem

Intel X86



Híradástechnikai Tanszék

Izsó Tamás

2015. október 1.

Section 1

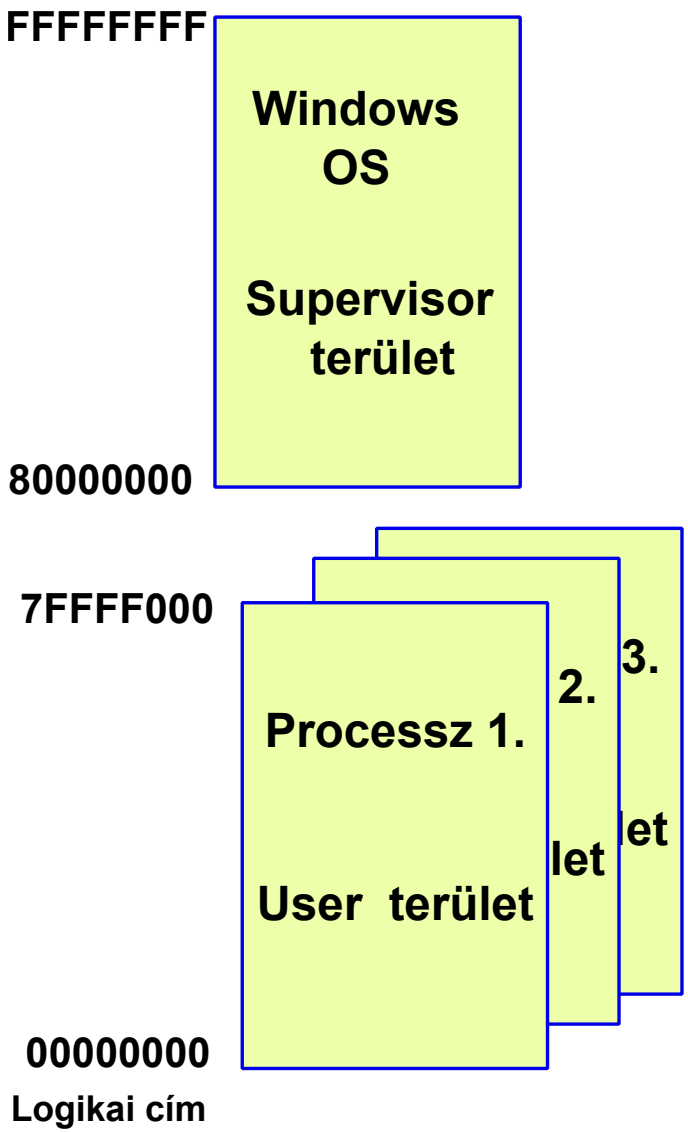
Virtuális memóriakezelés

Operációs rendszer hardver szintű támogatása

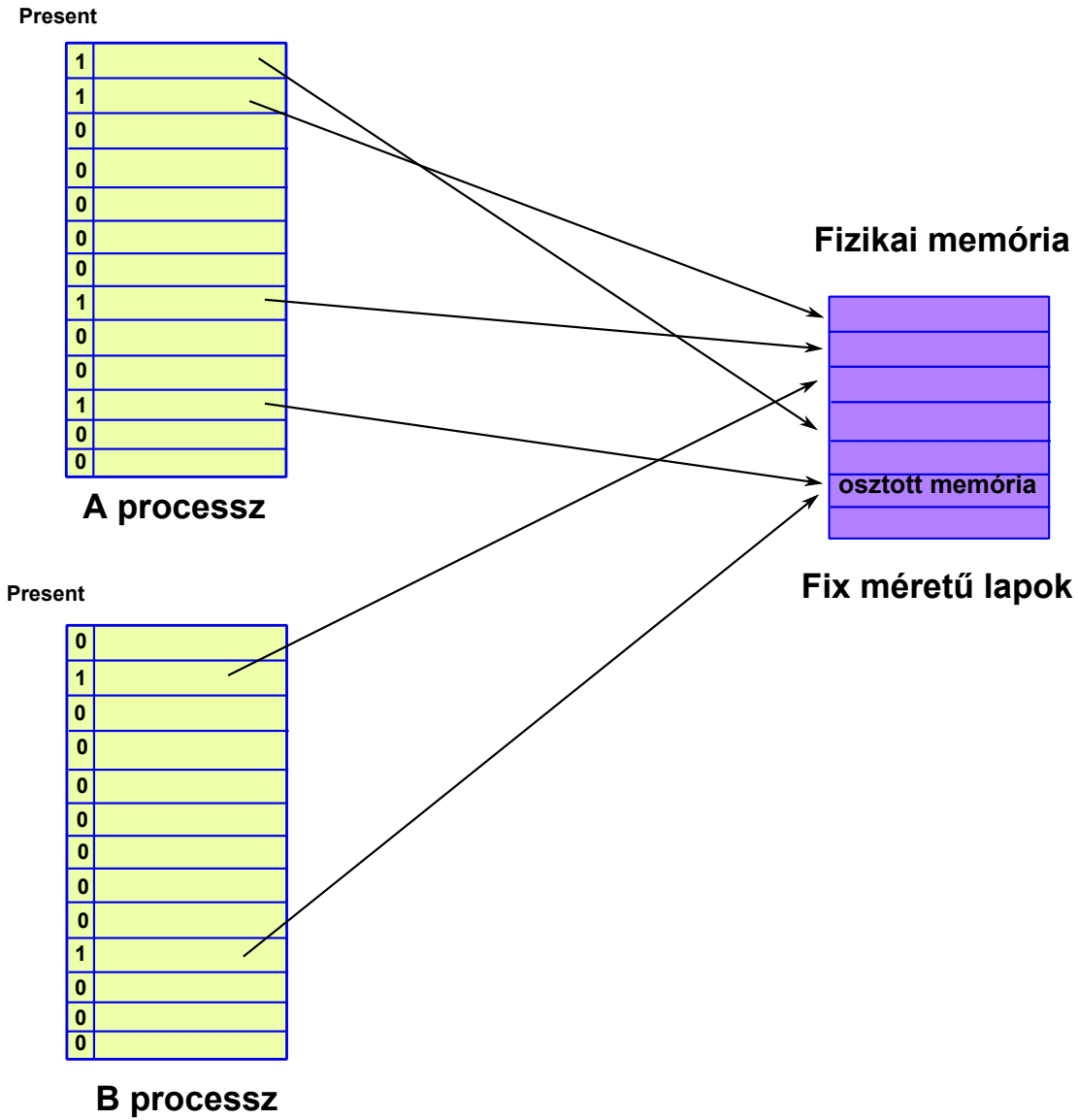
Hardver szinten támogatott operációs rendszer feladatok:

- memória kezelés;
- szoftver modul védelem;
- multitaszk szervezés;
- kivételkezelés és interrupt hívás;
- multiprocesszoros (több magos processzorok) kezelés;
- gyorsítótár alkalmazás;
- hardver erőforrások és tápfeszültség ellenőrzése;
- program futásának a nyomkövetése és teljesítményfigyelés.

32 bites Windows memória kiosztás



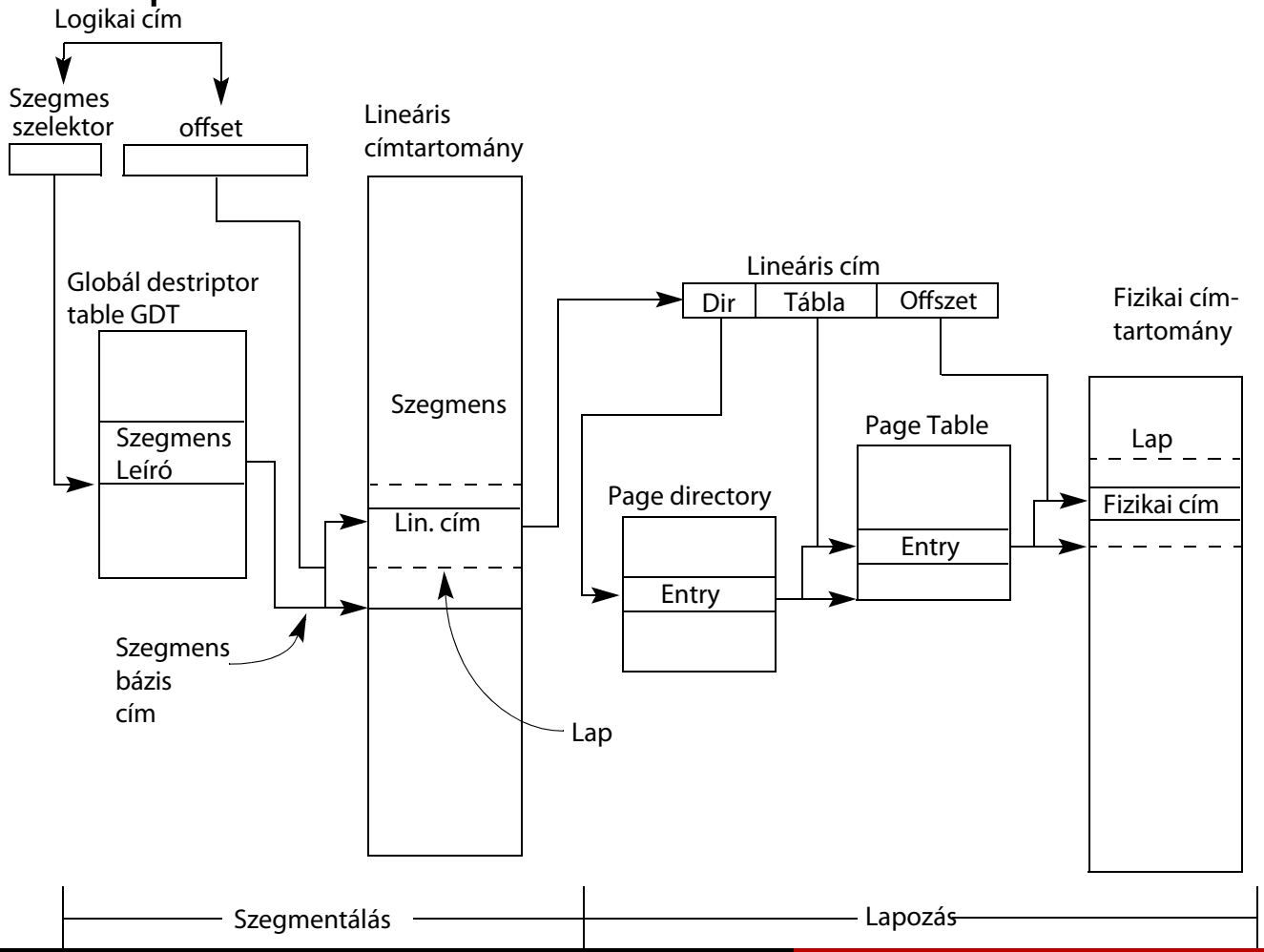
Lapozás elve (elnagyolva)



Intel x86 processzor memóriakezelése

Illusztráció!

- szegmentálás
- lapozás

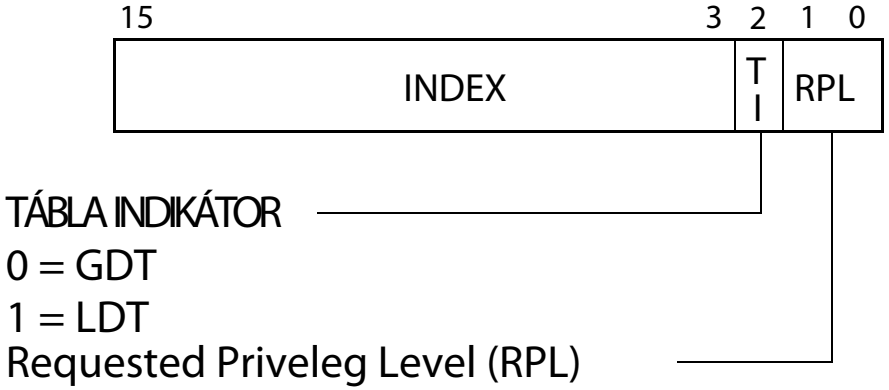


Szegmentálás feladata

- A taszkok memóriaterületének elkülönítése. A címtartományon kívül programhibából adódó hivatkozások megakadályozása. A rendszer és a felhasználói programok címtartományának a szétválasztása.
- Adat vagy kód kezdőcímének az áthelyezhetősége.
- Osztott memóriaterület szabályozott úton történő elérése.
- Osztott kódrészek hívása.
- Szegmens hozzáférések korlátozása.

Szegmensregiszter

Illusztráció!



szegmens szelektor

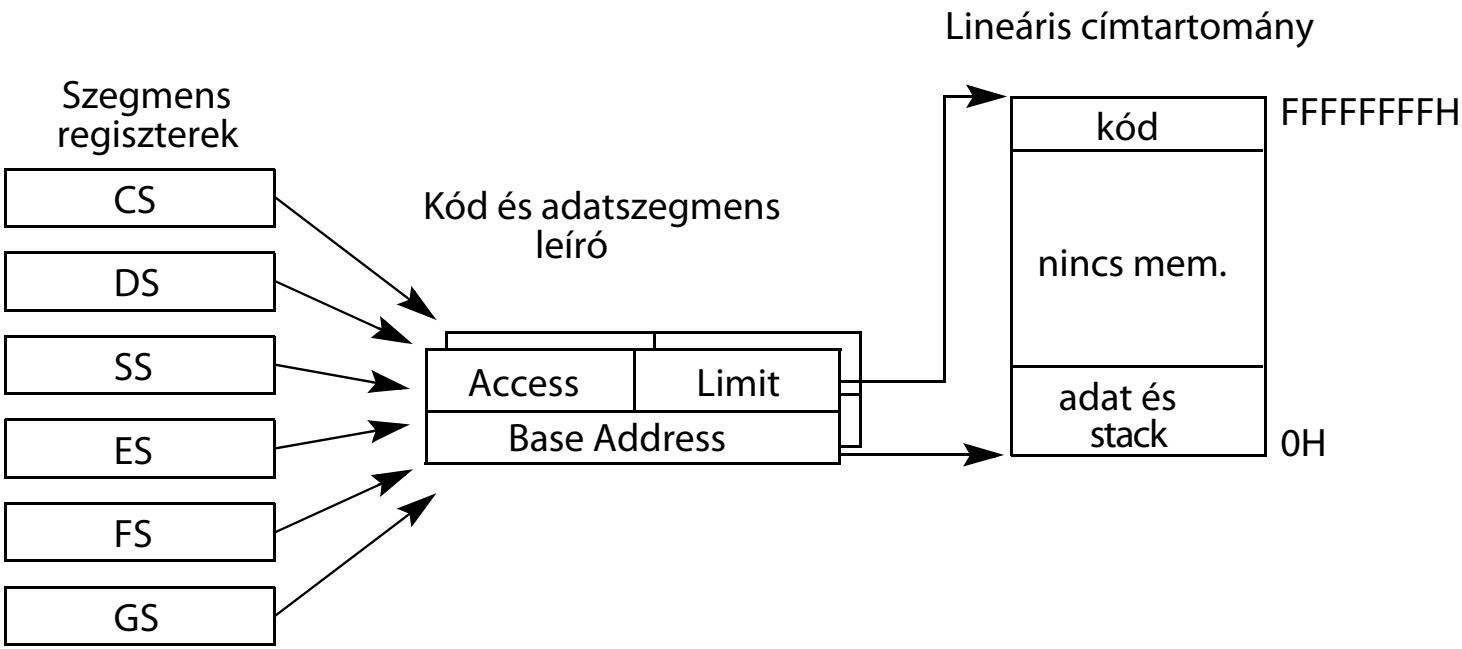
látható rész	takart rész	
szegmens szelektor	bázis cím, méret, hozzáférési információk	CS
		SS
		DS
		ES
		FS
		GS

szegmensregiszterek

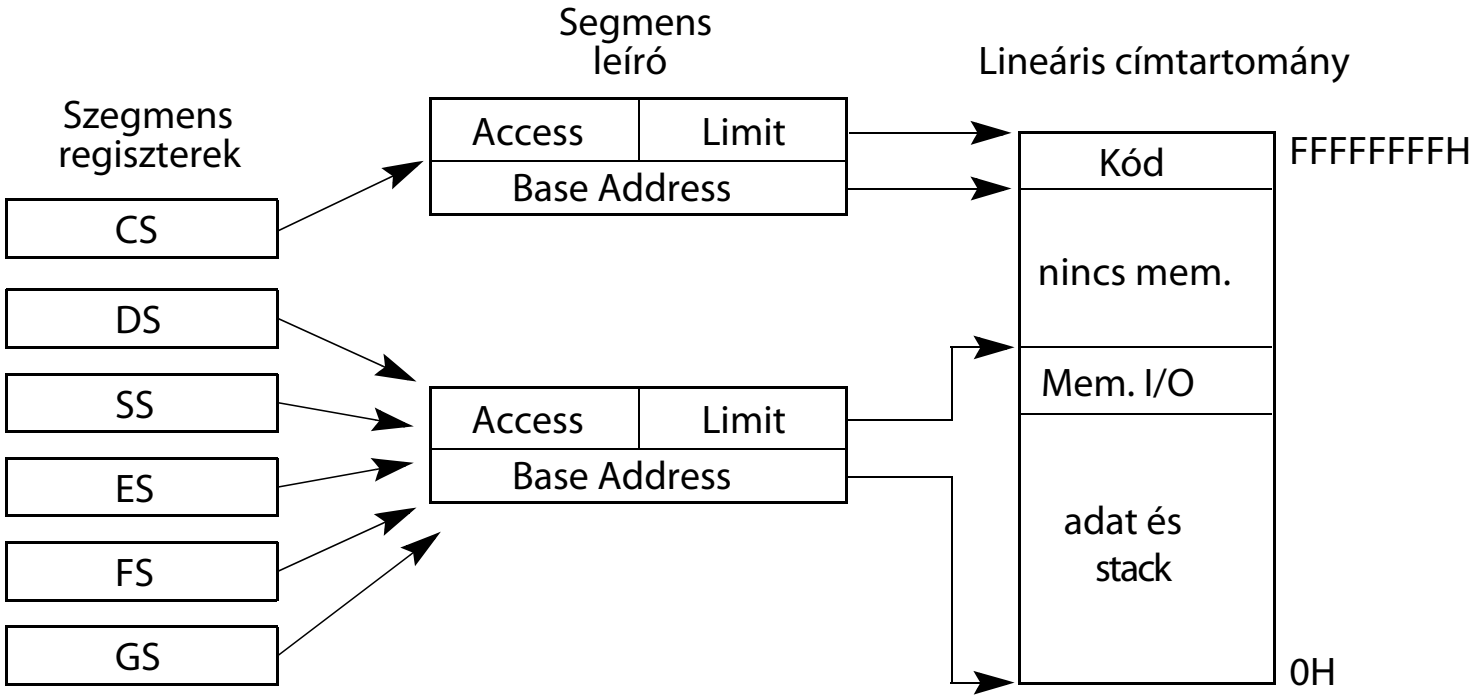
Flat modell

Szegmentálást nem lehet kikapcsolni, csak a lapozást. Ugyanakkor be lehet állítani a szegmenseket úgy, mintha nem is léteznének.

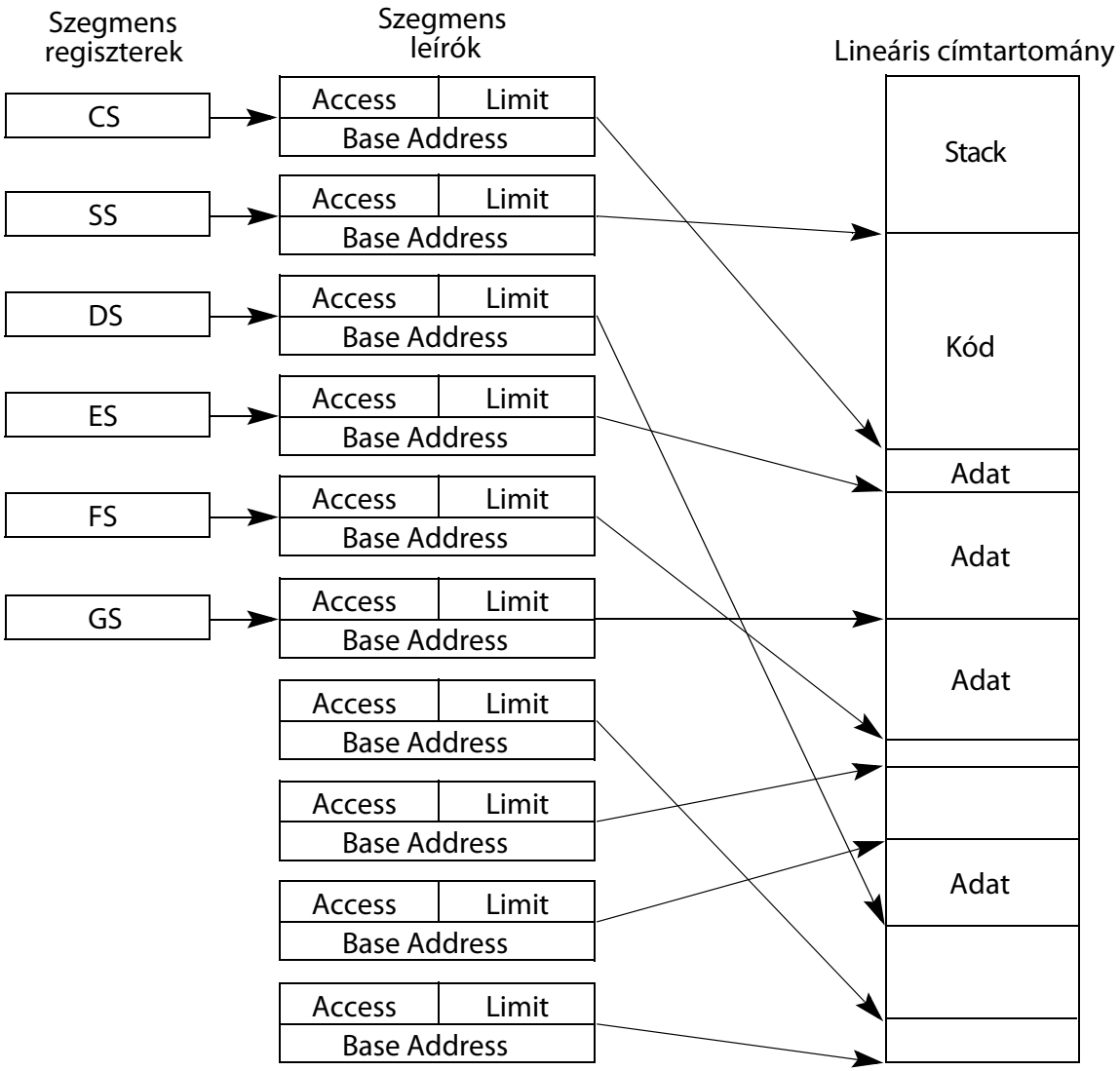
- Egy-egy szegmensleíró a kódra és adatra.
- Báziscím 0-tól.
- Méret (limit) 4GByte



Védett flat modell



Többszegmenses modell



Az x86 lapozási módok

Illusztráció!

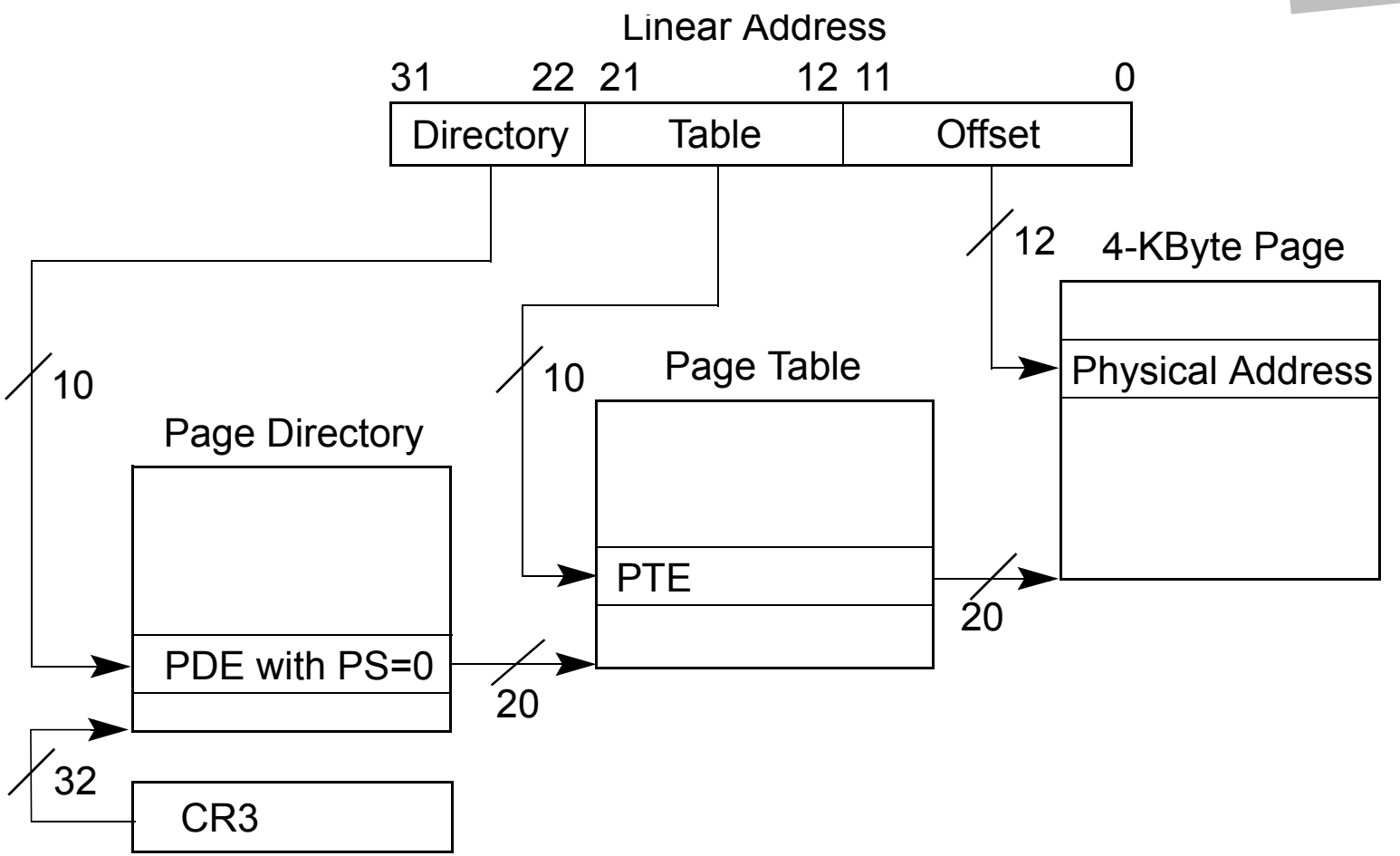
32 bites Ebben az üzemmódban a lineáris cím és a fizikai cím egyaránt 32 bites.

PAE Ebben az esetben a 32 bites lineáris címet 32 bitnél szélesebb vagy azzal egyenlő fizikai címre képezzük. Ez egy egzotikus címzési mód, mégis a Windows/XP az SP2-ben már ezt használja, ennek az okára később kitérünk.

IA-32e 64 bites üzemmód esetén használatos.

32 bites lapozás 4KB lapmérettel

Illusztráció!



Section 2

Védelem

Védelem

A védelem megszervezésnél a következő dolgokat kell átgondolni:

védelem tárgya Információ aminek a hozzáférését egyeseknek megengedjük, korlátozzuk, vagy megtiltjuk.

védelem alanya Aktív résztvevő, aki az információhoz a hozzáférést kezdeményezi.

tevékenység Aktív résztvevő tevékenységi sorozata az információ megszerzésére.

hozzáférési szabályok Leírják, hogy az egyes alanyok milyen tevékenységet végezhetnek az objektumokon.

Hozzáférési szabályok

DAC: Discretionary Access Control — tetszés szerinti hozzáférési jog biztosítása:

- Minden egyes felhasználó számára külön rendelkezhetünk a hozzáférési jogokról.
- Az információ birtokosa adja a jogokat.
- A DAC-ot rugalmassága miatt az operációs rendszerek előszeretettel használják

DAC hátránya

- Nehéz globális szabályokat megkövetelni.
- Nem biztos forrásból származó program a felhasználó jogosultsága alapján megváltoztathatja a hozzáférési jogokat.
- Nem biztonságosan megírt program gyengeségeit kihasználva lehetőséget kap a támadó, a DAC szabályainak a módosítására.

Hozzáférési szabályok

MAC: Mandatory Access Control előre maghatározott hozzáférési szabályok alapján történik az információk elérése.

- Rendszer szintű adathozzáférési szabályokat, a felhasználó nem tudja megváltoztatni.
- A rendszerre van bízva a jogok hozzárendelése.

Ezt használják a hardverek.

Szereplők az X86 processzorban

védelem tárgya Memória, regiszterek, I/O eszközök

védelem alanya Az azonosításhoz a felhasználó user ID vagy processz ID lenne a legalkalmasabb, de ezeket az operációs rendszer definiálja, és az alatta lévő hardver ezekről semmit sem tud. Ezért itt a futó processz kódszegmensére fogjuk a védelmet alkalmazni.

tevékenység gépi utasítások sorozata.

hozzáférési szabályok CPU-ban előre meghatározott. (Mi a játéktere akkor az operációs rendszernek?)

Védelmi szintek (gyűrűk)

MAC terminológiában **címkéket** jelent.

- 0. szint a legmagasabb szint, bár a legkisebb érték tartozik hozzá. Általában az operációs rendszer kernel programja használja, például memória lapozás, taszkok ütemezése stb.
- 1. szint a nagy prioritású készülék meghajtó programok (device driver) futnak ezen a szinten.
- 2. szint az alacsonyabb prioritású meghajtó programok részére van fenntartva.
- 3. szint a felhasználói programok futtatásához.

Miért csak 4 szint van?

- Mert a tervező így látta jónak. Vannak processzorok, ahol 8 vagy 16 szint is van.
- Két bit elegendő a tárolására.
- Mert ez lefedi a katonaságnál kidolgozott top-secret, secret, confidentil (bizalmas), unclassified szinteket.

Mandatory védelem biztosítása

- A védelem tárgyát, azaz a memóriát, regisztereket, I/O eszközöket felcímkézzük.
- A védelem alanyát azaz a kódot is felcímkézzük.
- Meg kell határozni, hogy az egyes címkével rendelkező alanyok milyen műveleteket végezhetnek a felcímkézett objektummal (angol irodalomban ez a *proteciton state*). A gyártó határozta meg, nem lehet megváltoztatni.
- Meg kell határozni, hogy milyen elvek alapján oszthatjuk ki a címkéket (angol irodalomban ez a *labeling state*).
- Meg kell határozni, hogy a védelem alanya vagy tárgya milyen szabályok alapján válthat címkét (angol irodalomban ez a *transition state*). A gyártó határozta meg, nem lehet megváltoztatni.

Adatok sértetlenségének a biztosítása

Definíció: (Információ iránya)

- ***Write*** *a védelem alanya felől halad az objektum felé.*
- ***Read*** *a védelem tárgya felől halad a védelem alanya felé.*

Kenneth J. Biba 1975-ben definiálta a róla elnevezett az adatok *integritásának* (CIA) védelmére szolgáló modellt.

Definíció: (Biba modell)

- ***No Write Up*** *A futó kód a nála magasabb privilégiummal rendelkező erőforrásokat nem módosíthatja.*
- ***No Read Down*** *A futó kód a nála alacsonyabb privilégiummal rendelkező erőforrásokat nem olvashatja.*

Adatok sértetlenségének a biztosítása

Megjegyzések:

- Az első pont triviális.
- A második meggátolja, hogy a kisebb privilégiummal rendelkező erőforrás, amit a kisebb privilégiummal rendelkező kód (user) módosított, befolyásolja a nagyobb privilégiummal működő kód működését.
- Rendszerhívás esetén biztosítani kell, hogy a nagyobb prioritással futó kód olvassa a kisebb prioritású kód által átadott paramétereket, ezért ott ellenőrizni kell azok tartalmát.

Adatok bizalmasságának a biztosítása

David Elliott Bell és Leonard J. LaPadula 1973-ban kifejlesztette az adatok *bizalmasságának* (CIA modell) a védelmére, azaz a titok kiszivárogtatásának a meggátlására szolgáló modellt.

Definíció: (Bell-LaPadula modell)

- ***No Write Down*** *A futó kód nála alacsonyabb privilégiummal rendelkező erőforrásokat nem módosíthatja.*
- ***No Read Up*** *A futó kód nála magasabb privilégiummal rendelkező erőforrásokat nem olvashatja.*

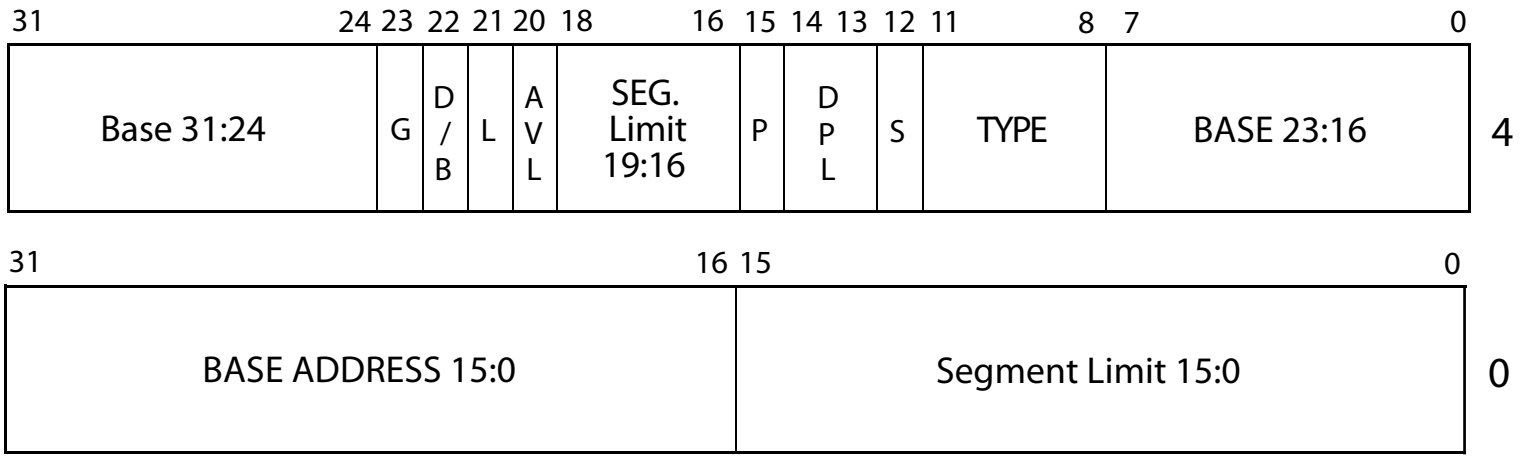
A **No Write Down** meggátolja, hogy a nagyobb privilégiummal rendelkező de vírusos kód ne szivárogtathasson ki titkos információt a kisebb privilégiummal rendelkező támadó felé. Az eredeti Bell-LaPadula modellben a kiosztott címkéket nem lehet megváltoztatni.

Biba kontra Bell-LaPadula modell

- Azonos privilégium szinttel rendelkező szereplők között a bizalmasságának és az adatok integritása biztosítható.
- A két elv alapján a különböző privilégium szinttel rendelkező szereplők között egyidejűleg a bizalmasság és az adatintegritás megtartása *automatikusan* nem teljesül.

Szegmentálás védelmi megoldások

Illusztráció!



Szegmensleíró

Szegmensleíró tartalma

Segment Limit szegmens méret (20 bit) függ G-től.

Base address Szegmens báziscíme, 4 GByte címezhető.

- **S** Hivatkozott szegmens kód, adat, vagy rendszerleíró tábla.
- **Type** Szegmens típusa, értelmezése függ S-től.
- **DPL** Szegmens privilégium szintje.
- **P** érvényes szegmensleíró van a memóriában
- **D/B** Kódnál operandus mérete, adatnál címtartomány iránya (expand down).
- **G** Szegmens méretének a mértékegysége byte/4Kbyte.
- **L** A szegmens 64 bites utasításokat tartalmaz.
- **AVL** Operációs rendszer számára fenntartott flag.

Szegmens típusa

Illusztráció!

típus értéke					szegmens típus	leírás
Decimálisan	11	10 E	9 W	8 A		
0	0	0	0	0	Data	Read-Only
1	0	0	0	1	Data	Read-Only, accessed
2	0	0	1	0	Data	Read/Write
3	0	0	1	1	Data	Read/Write, accessed
4	0	1	0	0	Data	Read-Only, expand-down
5	0	1	0	1	Data	Read-Only, expand-down, accessed
6	0	1	1	0	Data	Read/Write, expand-down
7	0	1	1	1	Data	Read/Write, expand-down, accessed
		C	R	A		
8	1	0	0	0	Code	Execute-Only
9	1	0	0	1	Code	Execute-Only, accessed
10	1	0	1	0	Code	Execute/Read
11	1	0	1	1	Code	Execute/Read, accessed
12	1	1	0	0	Code	Execute-Only, conforming
13	1	1	0	1	Code	Execute-Only, conforming, accessed
14	1	1	1	0	Code	Execute/Read, conforming
15	1	1	1	1	Code	Execute/Read, conforming, accessed

Általános védelmi eljárások

- Szegmens tartomány ellenőrzés (Base address, Limits)
- Szegmens típus ellenőrzés (*read, write, executable*, stb.). Továbbá végrehajtható segmens nem írható, és olvasni is csak akkor lehet, ha az engedélyezve van.
- Szegmensregiszterek használata:
  - CS végrehajtható,
  - SS írható szegmensre hivatkozhat.
  - DS, ES, FS, GS -be nem olvasható végrehajtható vagy rendszer szegmens nem tölthető.
- Utasítás szintű rendelkezések. Például call és jmp csak kód szegmensre, call gate, task gate, vagy TSS-re történhet. Privilegizált utasítások, stb.
- CS és SS regiszterbe nem tölthető a 0 szegmens szelektor érték, a többibe igen, de ha hivatkozunk rá kivételdobás történik.

Védelmi eljárások különböző privilégium szintek között

- CPL Current Privilege Level (aktuális privilégium szint) ami a védelem alanyához tartozó címke.
  - CPL-t a CS szegmensregiszter nem látható része tartalmazza.
  - CPL az aktuálisan futó processzhez tartozó privilégium szint.
  - Normál állapotban a CPL megegyezik az aktuális utasítást tartalmazó kódszegmens DPL-jével.
  - CPL változik, ha a program olyan kódszegmensen fut tovább, amelynek más a privilégium szintje.
- DPL Descriptor Privilege Level a védelem tárgyának a privilégium szintje. DPL-t a szegmensleíró tábla bejegyzésének a 13, és 14-ik bitje tartalmazza.

Nagyobb privilégiummal futó kód kihasználása

1. Kis privilégium szinttel rendelkező $\mathcal{U}$ kód átad egy olyan címet rendszerhívásnál $\mathcal{S}$ kódnak, amelyet meg akar változtatni, de neki nincs elég privilégiuma oda írni.
2. A meghívott $\mathcal{S}$ nagy privilégiummal (0 érték) futó kódnak joga van a megadott címre írni, így véghezviszi $\mathcal{U}$ kártékony kód kérését.

Hogyan védekezhetünk ez ellen?

RPL Request Privilege Level

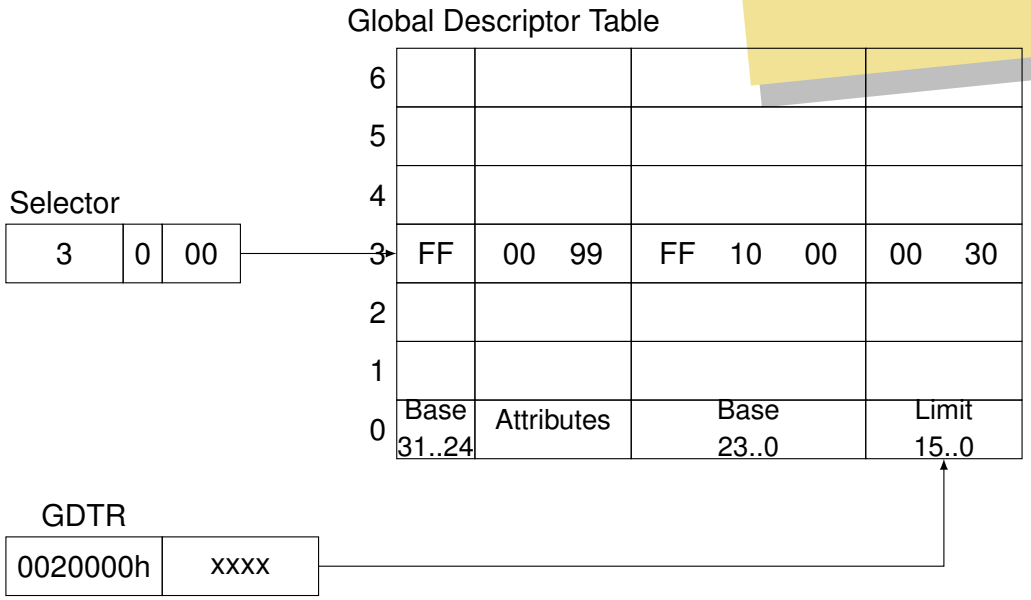
- RPL-t az operációs rendszer állítja be.
- Hozzáférés engedélyezett, ha $\max(CPL, RPL) \leq DPL$. Tehát ha a hívónak joga volt az adott területre írni, akkor ez a nagyobb privilégiummal rendelkező kódnak is engedélyezett.

Itt a Biba elv érvényesítését lehet megfigyelni.

RPL megadása és lineáris címszámítás

Illusztráció!

```
mov AX, 18h
mov DS, AX
mov ESI, 22h
mov EAX, [ESI]
```

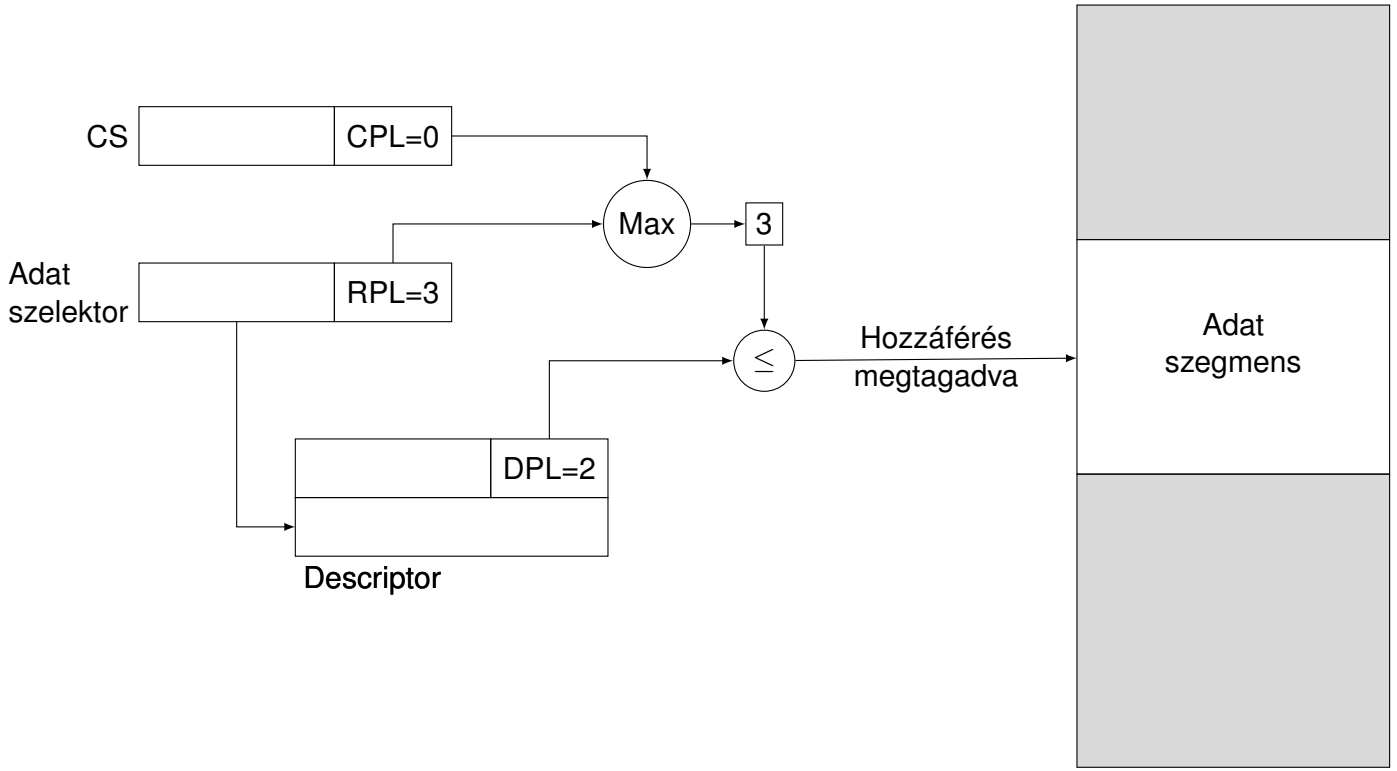


1. selector értéke: 18h = 0000000000011 0 00_b $\Rightarrow$ Index = 3, TI a GDT-t jelöli ki, RPL = 0.
2. Keressük ki az index által megadott bejegyzést a GDT-ből.
3. Bázis cím $0FFFF1000_h$.
4. Mivel ESI=22, ezért a hivatkozik cím $0FFFF1000_h + 22_h = 0FFFF1022_h$

Privilégium ellenőrzése adat hivatkozáskor

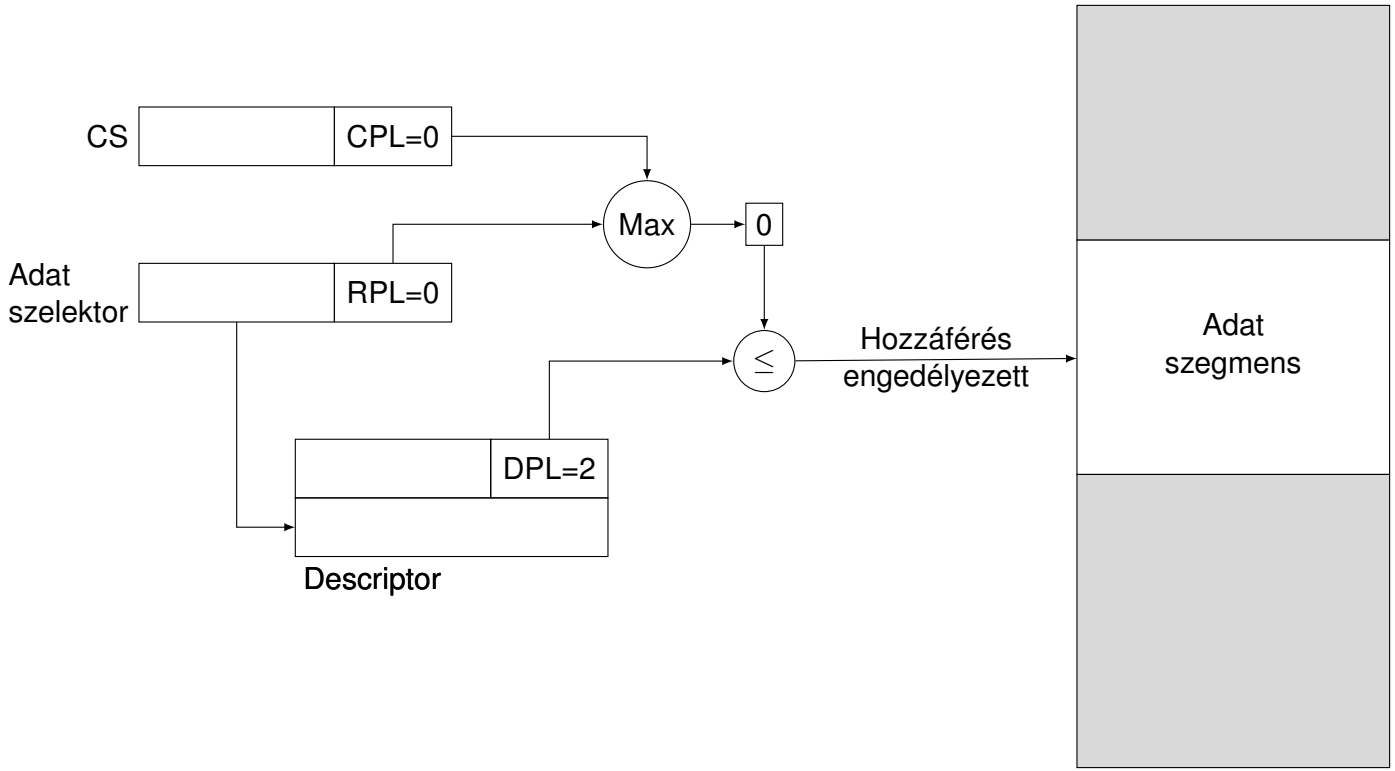
- Kisebb privilégium szintű kód (CPL) nem fér hozzá nagyobb privilégium szinttel rendelkező adathoz (DPL) (Bell-LaPadula elv).
- Csak a DS, ES, FS, GS vagy SS regiszterek megváltoztatásánál kell erre figyelni.
- Ezeket a regisztereket csak a MOV, POP, LDS, LES, LFS, LGS, LSS utasításokkal lehet módosítani.
- Szabály: $\max(CPL, RPL) \leq DPL$.

Adathozzáférés ellenőrzésére példa I



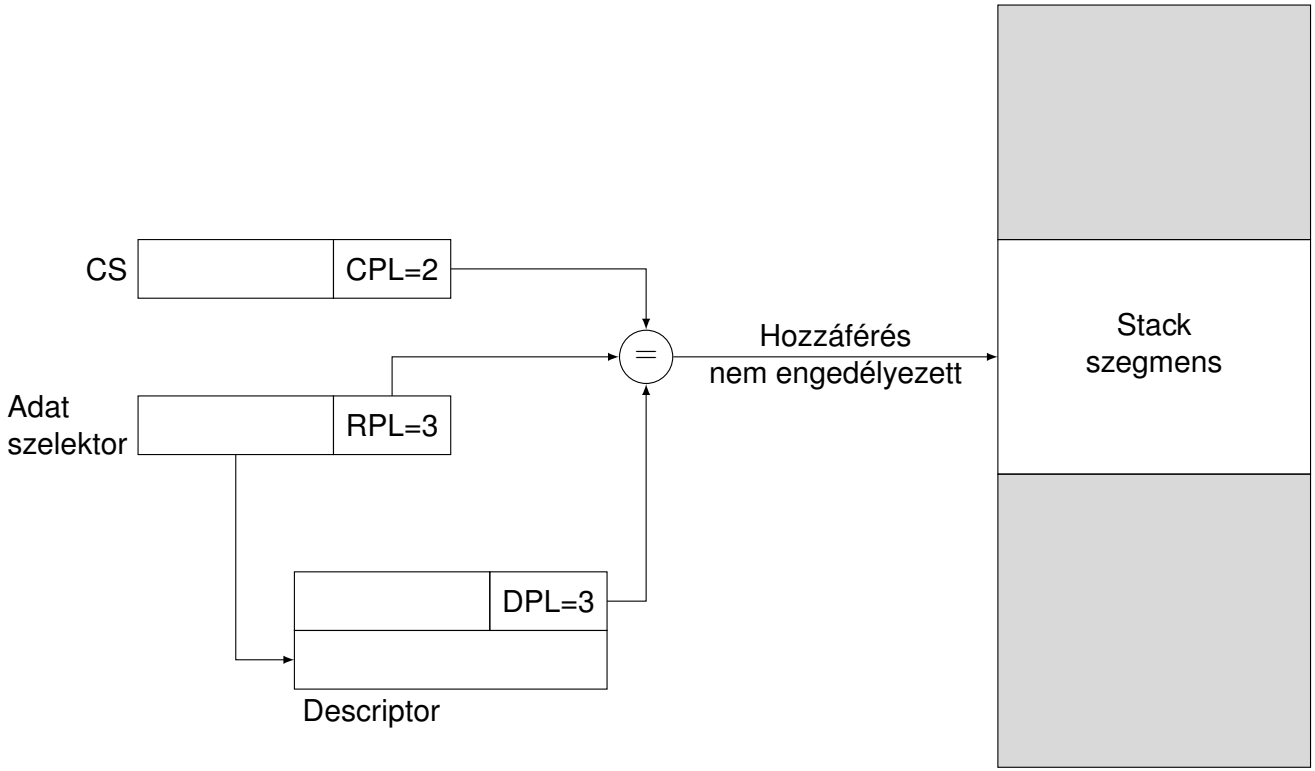
Adatszegmens hozzáférése nem megengedett

Adathozzáférés ellenőrzésére példa II



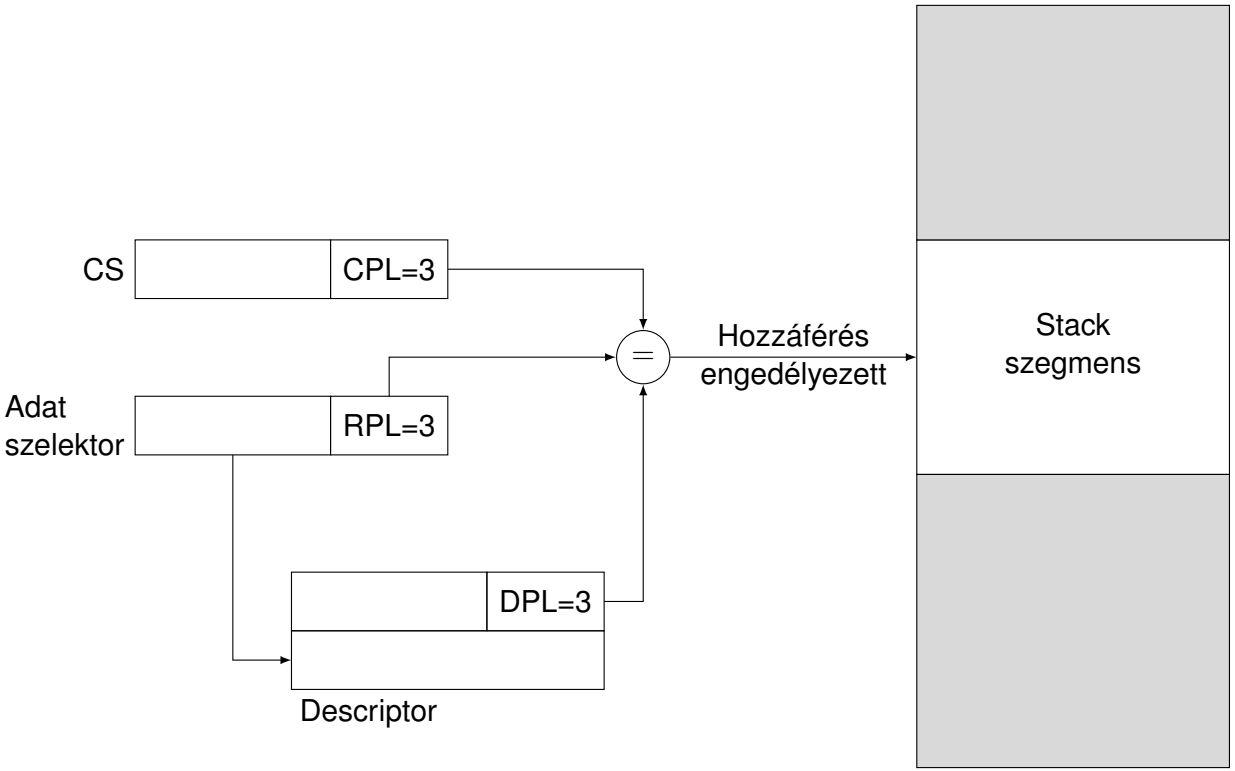
Adatszegmens hozzáférése engedélyezett

Stack hozzáférés ellenőrzésére példa I



Stack szegmens hozzáférése nem megengedett

Stack hozzáférés ellenőrzésére példa II



Stack szegmens hozzáférése engedélyezett

Miért nem hívunk meg kisebb privilégiummal rendelkező kódot

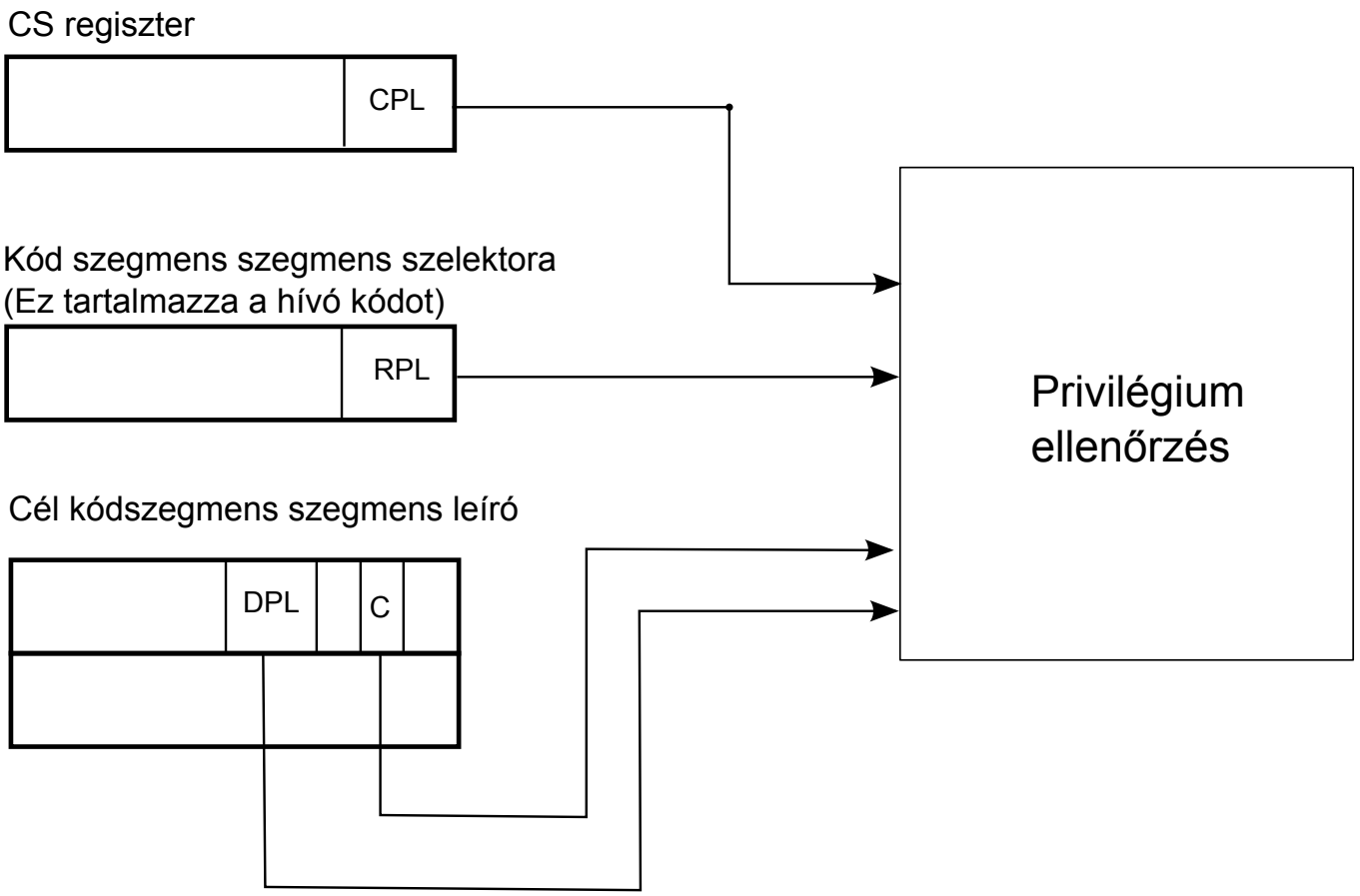
- Könnyű átmenni az alacsonyabb privilégiummal rendelkező kódba, de nehéz onnan visszatérni.
- Áttérés után a kód nem érheti el a saját adatait, mivel a megváltozott CPL numerikus értéke nagyobb, mind a kód DPL értéke.
- Valóban szükség van erre?

Miért nem ugorhatunk nagyobb privilégiummal rendelkező kódba

- Biztonsági okokból nem tehetjük meg.
- Valóban szükség van rá? Igen, a device drivereknél, például hogy az USB eszközt lássa a felhasználói program.
- Megoldás *call gate*-tel, amit a következő fejezetben tárgyalunk.

Kódszegmens váltás

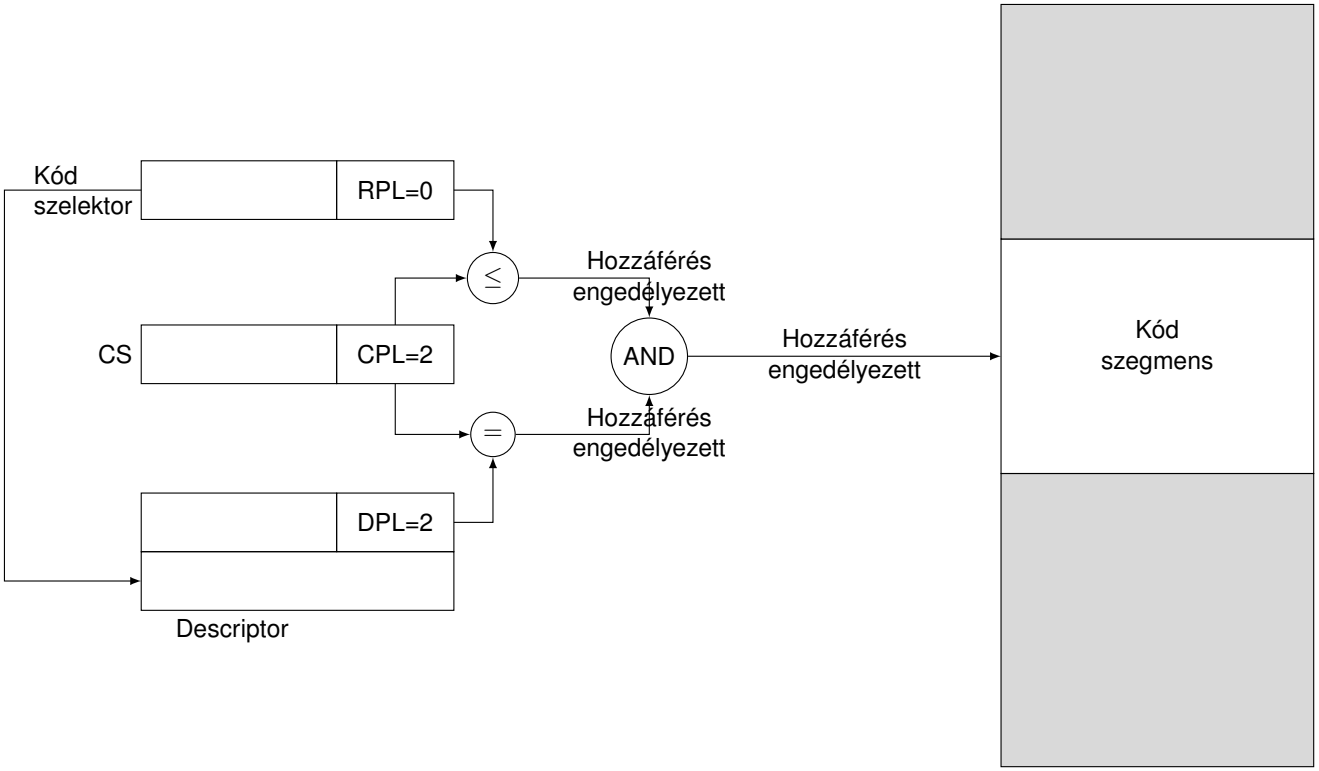
A privilégium ellenőrzésénél felhasznált adatok.



Szegmens conform típusának a felhasználása

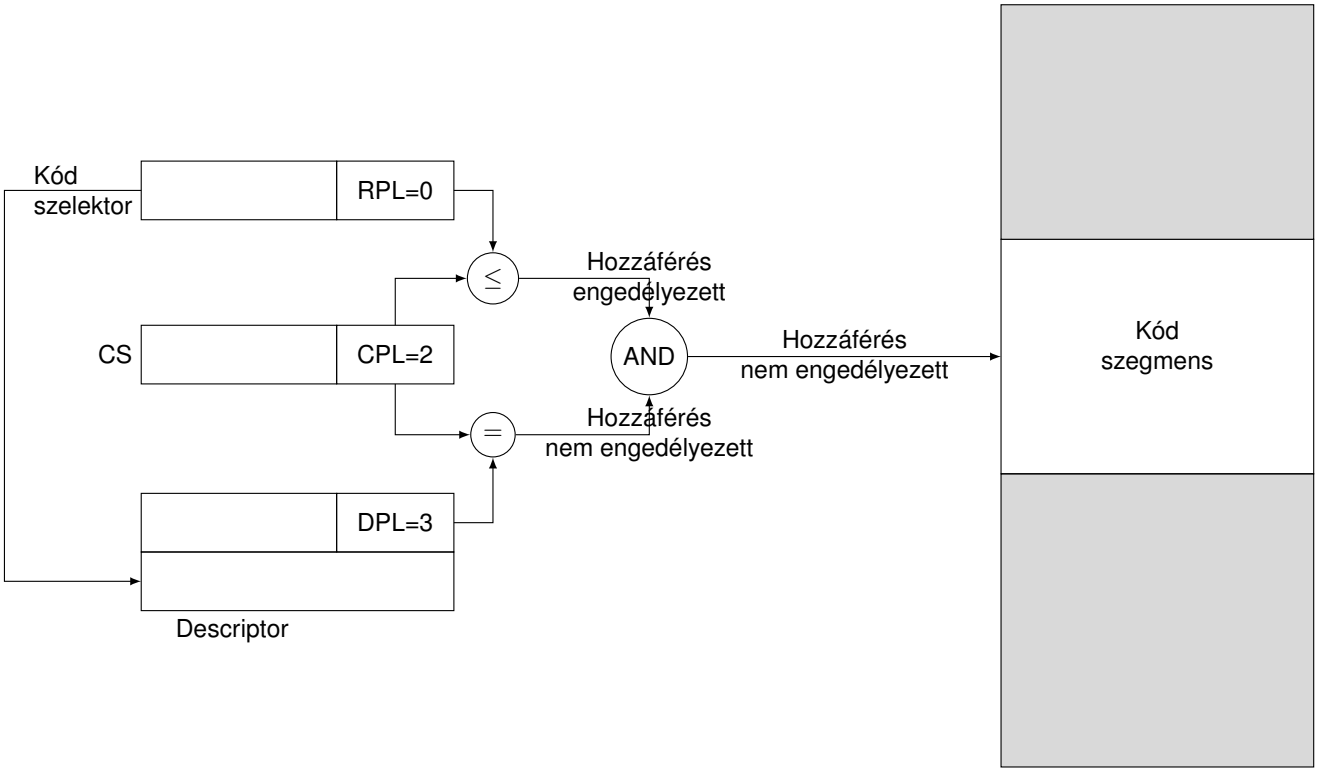
- **Nem conform**(nem alkalmazkodó) kódszegmenst csak akkor lehet meghívni, ha $CPL = DPL$. Ebben az esetben *RPL*-nek kisebb szerepe van, csak arra kell figyelni, hogy az értéke kisebb vagy egyenlő legyen a CPL-lel.
- **Conform** tulajdonságú kódszegmens $CPL \geq DPL$ feltétel mellett meghívható. Ilyenkor az RPL nem számít. A függvény meghívása után a CPL értéke nem változik, ez az egyetlen olyan eset, hogy a CPL és az utasításokat tartalmazó szegmens DPL-je nem biztos, hogy azonos értékű. Nem jön létre stack váltás. Felhasználási lehetőség például a matematikai könyvtári függvények céljára.

Kódszegmens hozzáférés ellenőrzésére példa I



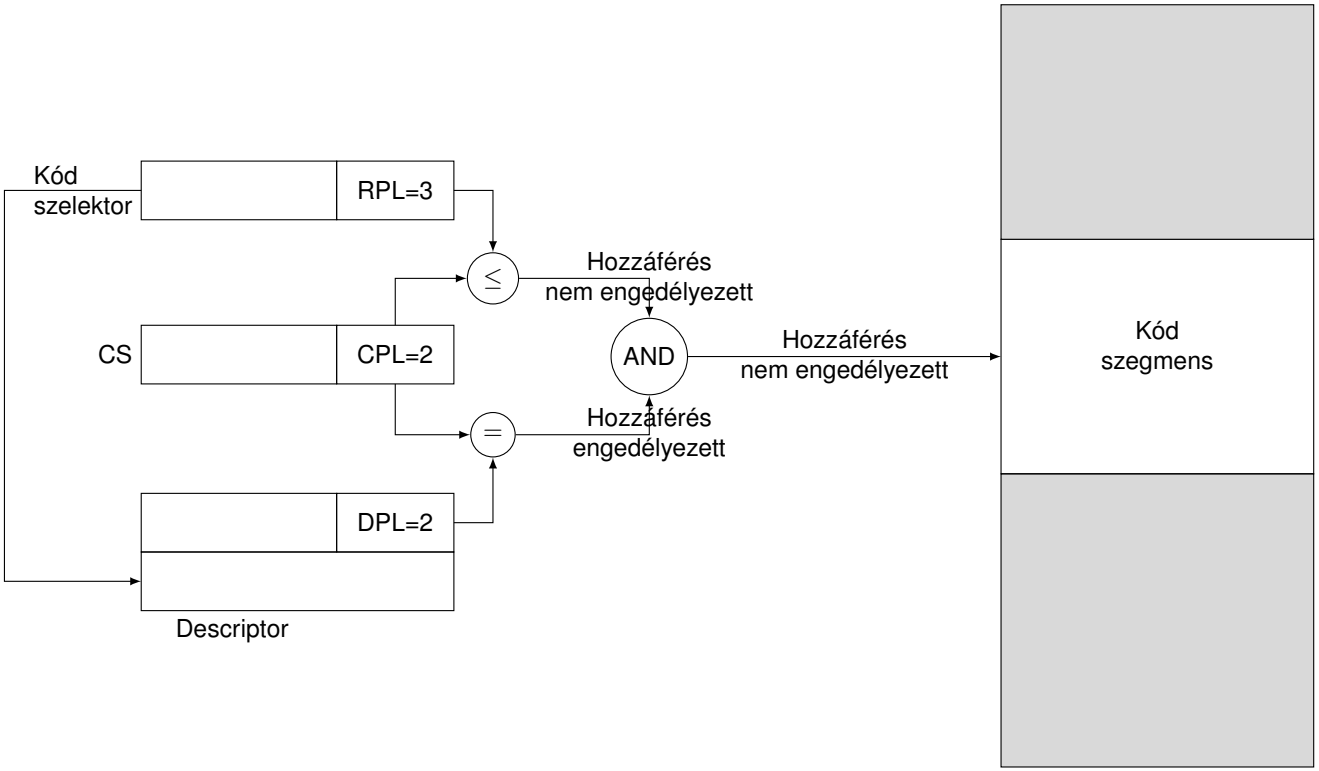
Nem conform kódszegmens hozzáférése engedélyezett

Kódszegmens hozzáférés ellenőrzésére példa II



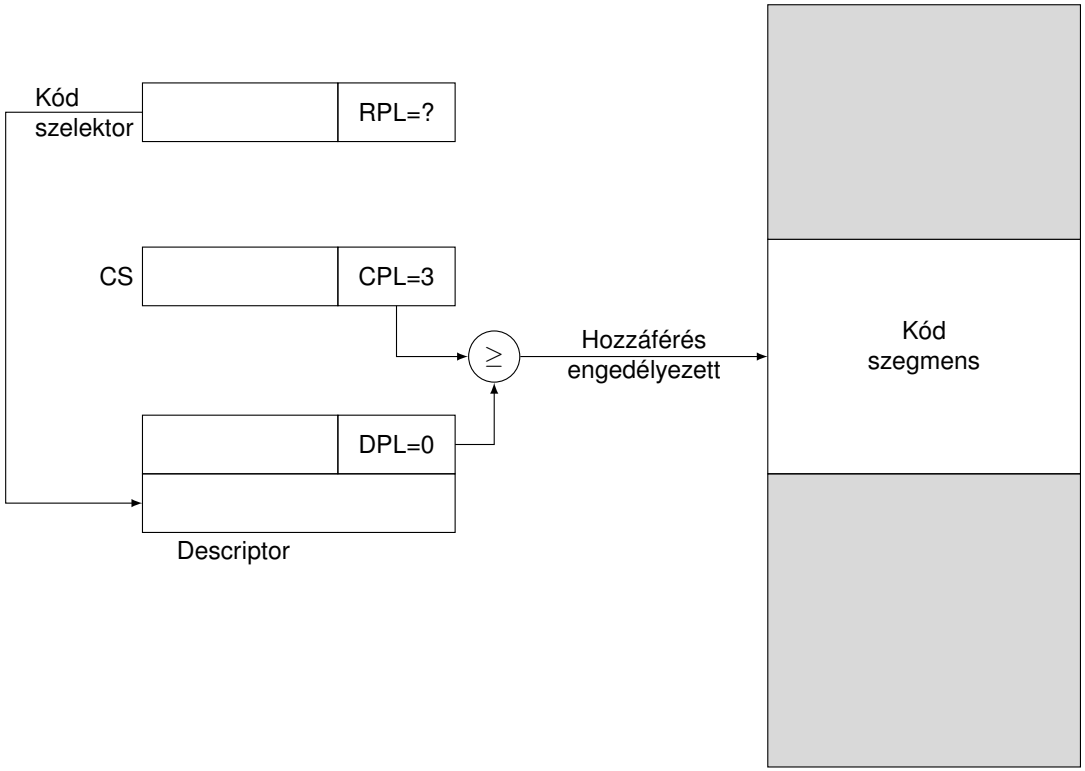
Nem conform kódszegmens hozzáférése nem engedélyezett

Kódszegmens hozzáférés ellenőrzésére példa III



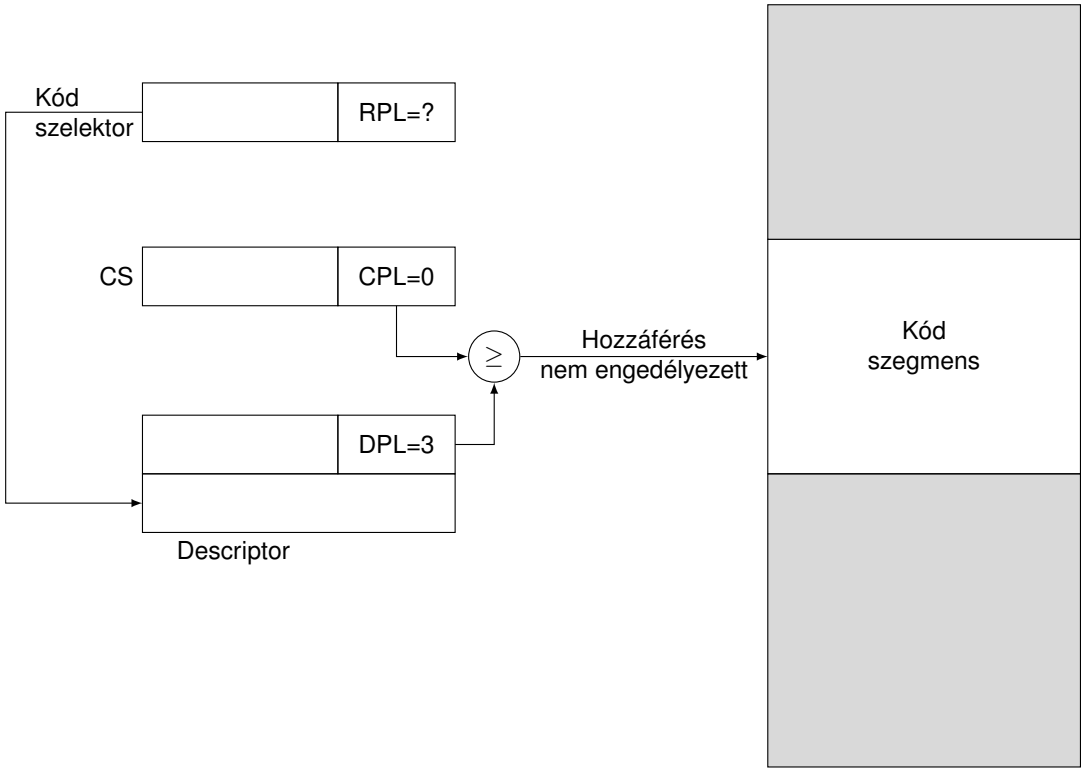
Nem conform kódszegmens hozzáférése nem engedélyezett

Kódszegmens hozzáférés ellenőrzésére példa IV



Conform kódszegmens hozzáférése engedélyezett

Kódszegmens hozzáférés ellenőrzésére példa V



Conform kódszegmens hozzáférése nem engedélyezett

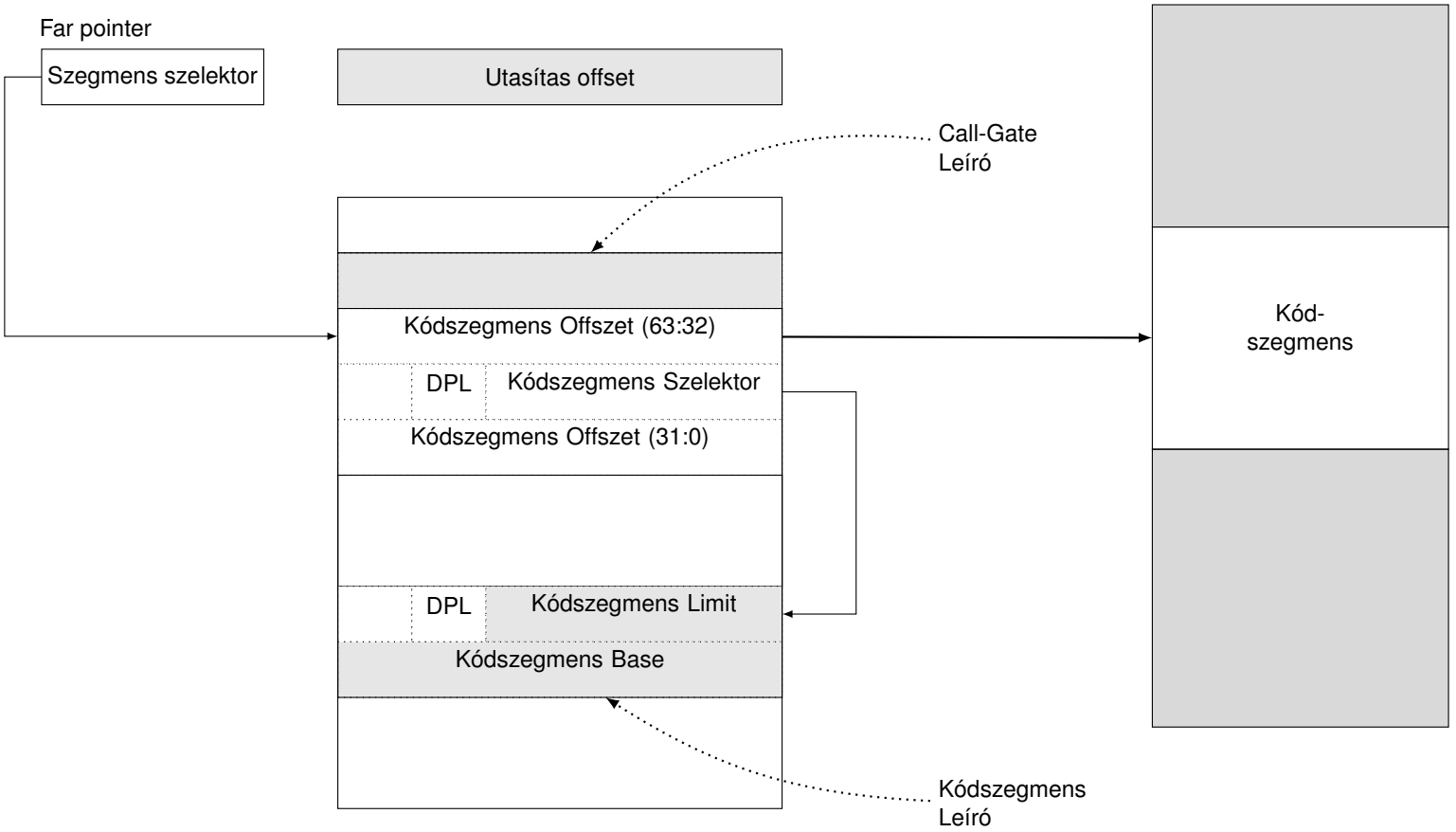
OS meghívása

Nagyobb privilégiummal rendelkező kódszegmenst, egyszerű call utasítással biztonsági okokból nem hívhatunk meg. Különböző interface-k léteznek, hogy a hívások minél ellenőrzöttebben történjenek.

- call gate (többszegmens modell esetén, rugalmas)
- szoftver interrupt (többszegmens és flat modell esetén)
- SYSENTER/SYSEXIT (többszegmens és flat modell esetén, gyorsabb mint az interrupt)

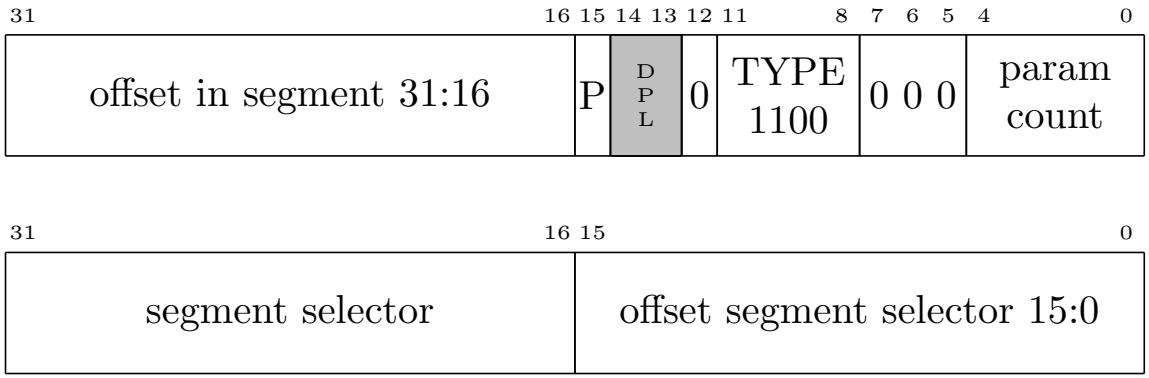
Call gate

A *call gate* leíró is a GDT vagy a LDT-ban található.

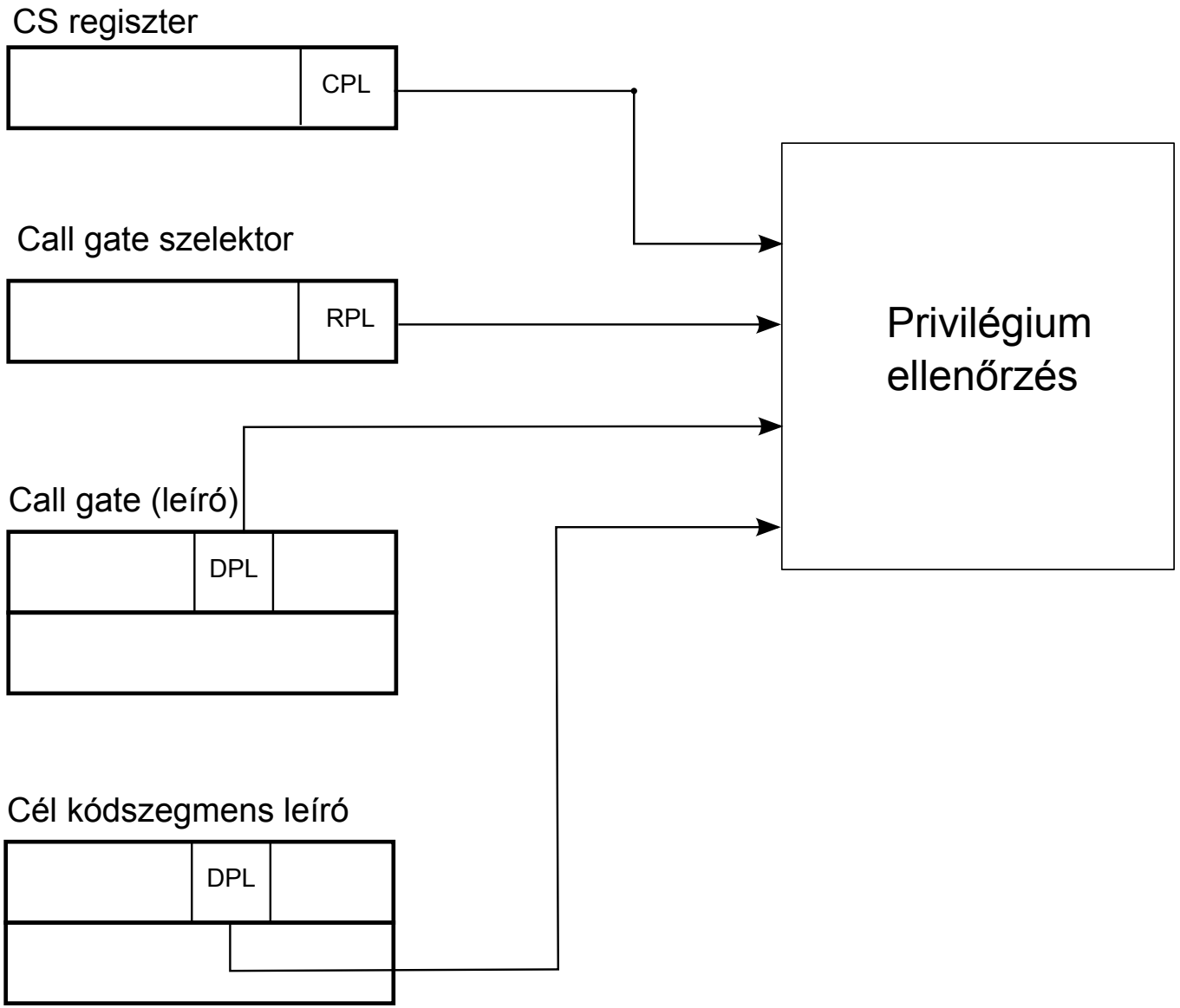


call gate leíró adatai

- Meghívandó kódszegmens szelektora.
- Szegmensen belül az eljárás belépési címe.
- A hívó kódtól megkövetelt DPL privilégium szint.
- Stack váltás esetén:
  - átadandó paraméterek száma,
  - stack adatának a mérete. Lehetővé teszi, hogy 32 bites alkalmazás 16 bites programot hívjon meg.



Call gate ellenőrzés

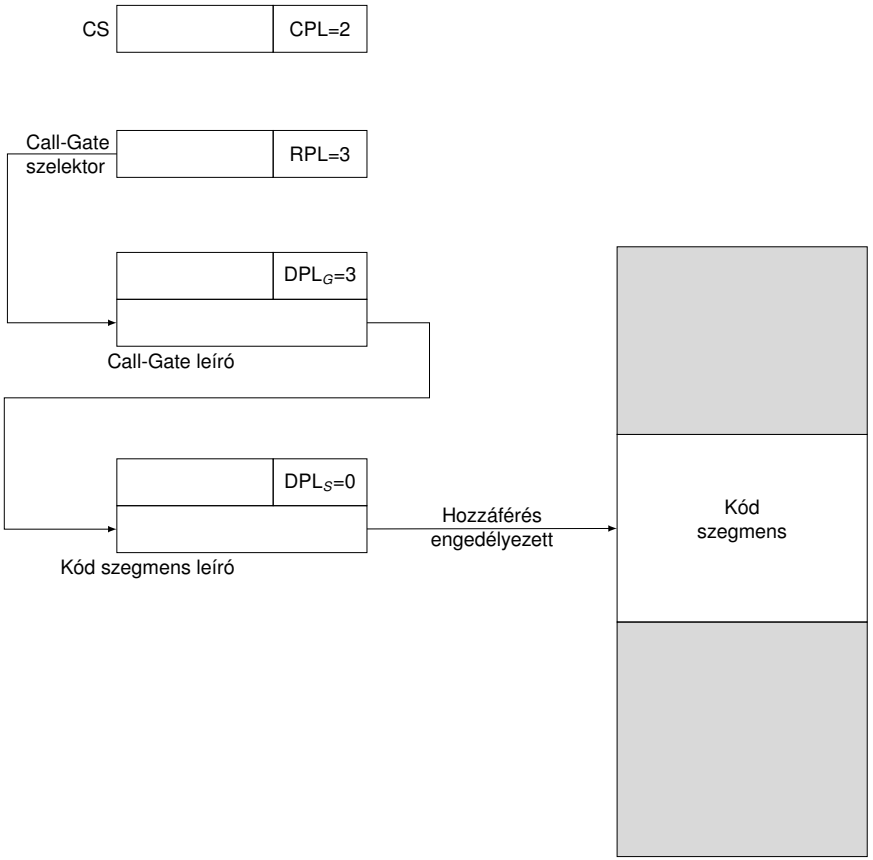


Call gate hozzáférési szabályok

- $CPL \leq DPL_{GATE}$
- $RPL \leq DPL_{GATE}$
- CALL utasítás végrehajtása csak a $DPL_{cél} \leq CPL$ szabály esetén lehetséges,
- Nem conform szegmensbe ugrás JMP utasítással csak akkor lehetséges, ha $DPL_{cél} = CPL$.
- Conform szegmensbe ugrás JMP utasítással csak akkor lehetséges, ha $DPL_{cél} \leq CPL$.

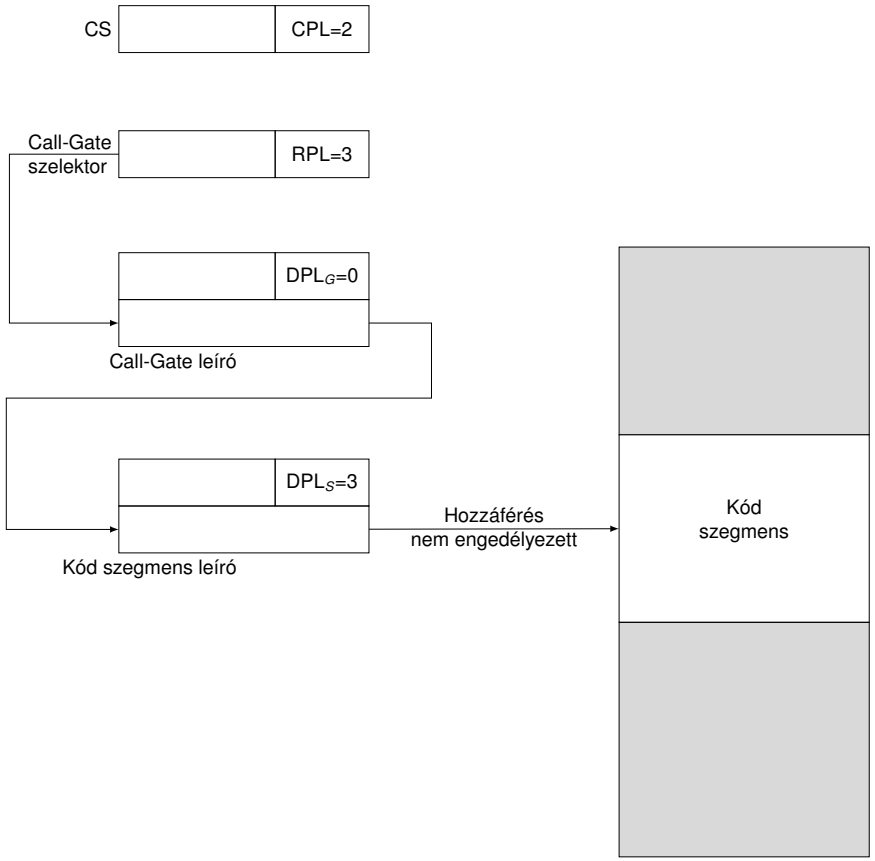
Miért nem megengedett a kisebb privilégiummal rendelkező függvény hívása? Azért mert akkor a visszatérésnél lennének problémák.

Kódszegmens hozzáférés ellenőrzésére példa I



Kódszegmens hívás Call-Gate-n keresztül, hozzáférése engedélyezett

Kódszegmens hozzáférés ellenőrzésére példa II



Kódszegmens hívás Call-Gate-n keresztül, hozzáférése nem engedélyezett

Stack gyengeség kiküszöbölése

A támadó kihasználhatná, ha a nagyobb privilégium szinttel futó kód a kisebb privilegizált szinttel rendelkező taszk stack-jét használná. (Ott maradt adatok kilesése, szándékos stack overflow-val vagy underflow-val való trükközés.)

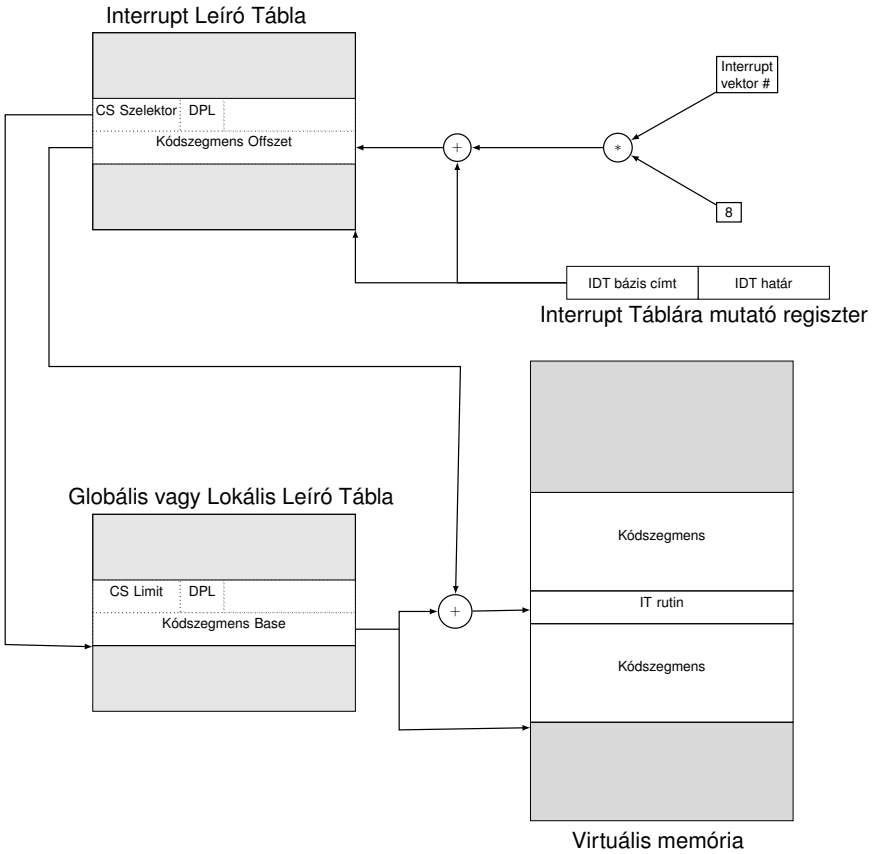
Stack switch

Privilégium szint váltás esetén a processzor stack-et is vált!

0, 1, 2 privilegizált szintekhez a rendszer task-state-segmens (TSS) területén vannak tárolva az egyes gyűrűkhöz tartozó stack-ek (SS, ESP) címei. Stack váltás lépései:

1. Régi stack (SS és ESP) értékének az elmentése.
2. DPL alapján a megfelelő SS, ESP értékek betöltése (stack váltás).
3. Stackre letett paraméterek átmásolása. Call-Gate leíró 5 bitje tartalmazza a stack-re letett adatok méretét.
4. Visszatérési cím elmentése.

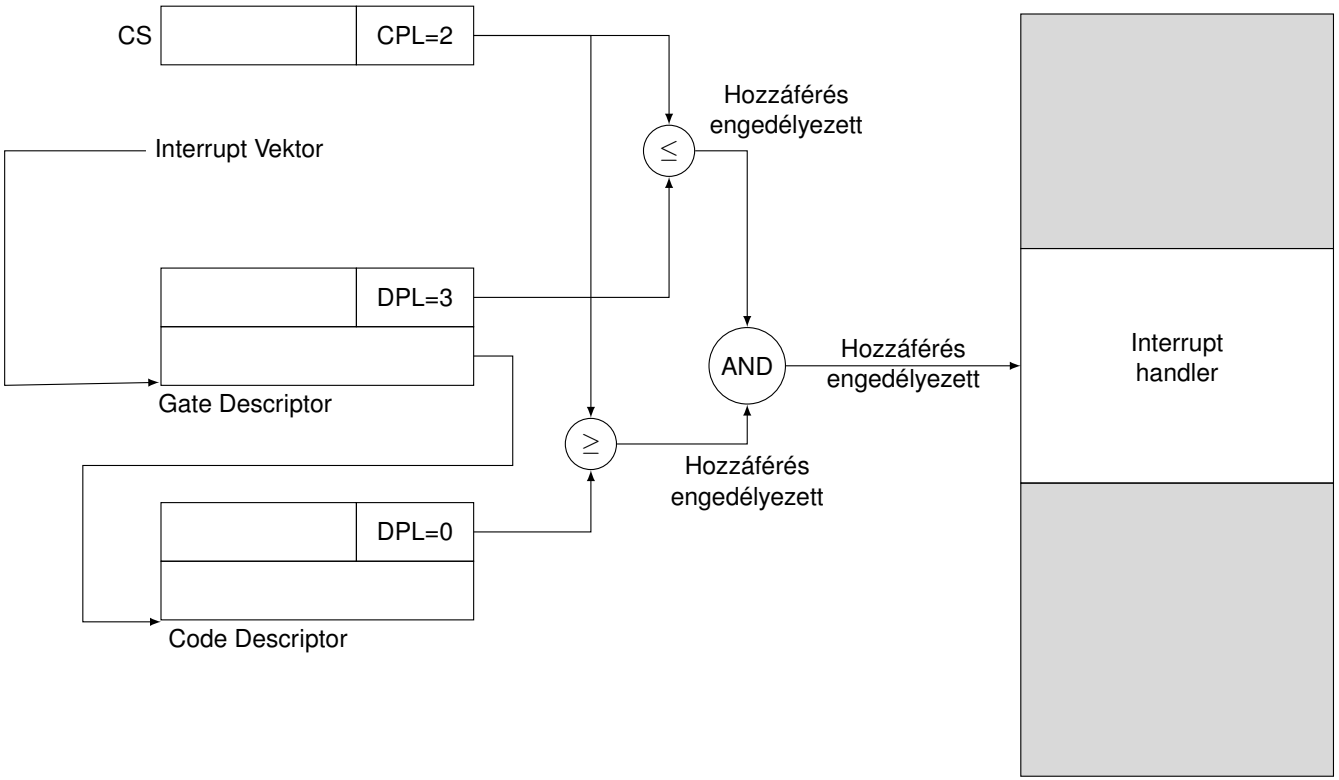
Protected IT rutin hívás



Iterrupt vektor gate leírása

Az interrupt vektor hívás a Call-Gate hívással hasonlatos módon történik.

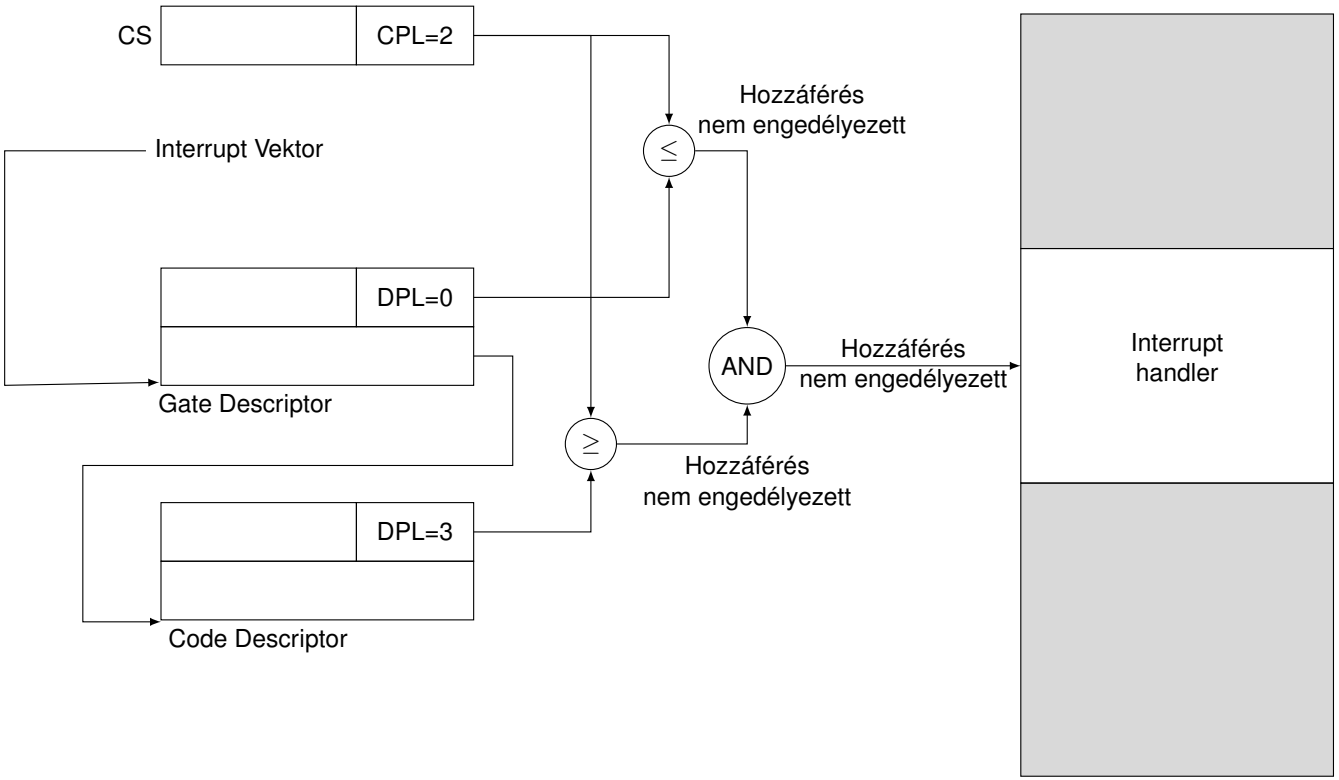
Protected IT rutin hívás ellenőrzése



Iterrupt vektor privilégium ellenőrzés

Az interrupt hívás engedélyezett.

Protected IT rutin hívás ellenőrzése



Iterrupt vektor privilégium ellenőrzés

Az interrupt hívás nem engedélyezett.

Operációs rendszer szolgáltatásának a hívása

A Pentium II processzorokban jelent meg az sysenter és sysexit utasítások. Régebbi X86 rendszerekben, illetve az operációs rendszerekben amíg át nem tértek az újabb gyorsabb utasításra, interruptot használtak. Linux az int 0x80, Windows int 0x2e interruptot használta erre a célra.

```
mov   eax, 5            ; Op. rendszer fv
lea   edx,[esp+0x8]     ; Paraméter
int   2e
```

Operációs rendszer szolgáltatásának a hívása

A sysenter és sysexit utasítások végrehajtásának az ideje azonos az egyszerű call és ret utasításokkal.

A rendszer 3 modell specifikus regiszterben beállítja a :

- SYSENTER cél CS - hívott eljárás kódszegmense.
- SYSENTER cél ESP - hívott eljárás stack kerete.
- SYSENTER cél EIP - hívott eljárás címe.

```
mov eax, 0x1234
mov edx, address of ntdll!KiFastSystemCall
call edx
retn 8
...
ntdll!KiFastSystemCall :
mov edx, esp
sysenter
retn
```

Lapszintű hozzáférési szabályok

A 4 privilegium szint a következőképpen képződik le a lap U/S bitjére:

- DPL = 0,1,2 érték supervisor mód.
- DPL = 3 felhasználó mód

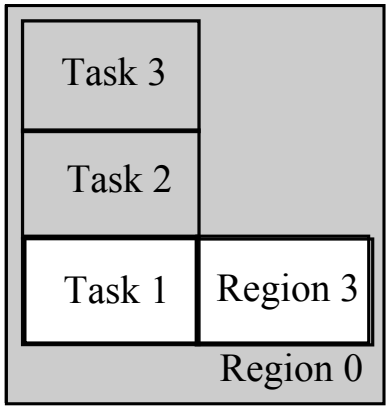
Típus védelem lap esetén csak írásra/olvasásra terjed ki. PAE és 64 bites lapozásnál végrehajtás letíltás (execute disable NX) is lehetséges.

- Csak olvasás van megengedve, ha $R/W = 0$
- Írás és olvasás is engedélyezett, ha $R/W = 1$
- Inicializálás miatt a supervisor minden lapot írhat és olvashat, de csak akkor, ha a CR0 regiszter WP bitje 0. Különben neki is be kell tartani a játékszabályokat.
- Supervisor módú lapokat a user módú program sem írni sem olvasni nem tudja.
- Page Directory Table, és Page Table esetén az R/W és a U/S bitek használata azonos a fent leírtakkal.

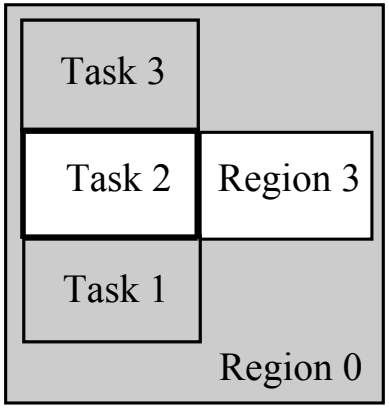
ARM Memory Protection Unit

- 8 vagy 16 zóna, hasonló feladatot lát el mint a 80x86 szegmense, de előnye, hogy nem vesz részt a címszámításban.
- tartomány ellenőrzés (kezdőcím és méret)
- védelmi szintek megkülönböztetése
- írás, olvasás, futtatás engedélyezése, letíltása
- cache vezérlés
- a zóna átlapolódhat, ilyenkor a nagyobb prioritás lesz mérvadó. A prioritás merőleges a védelmi szintre.

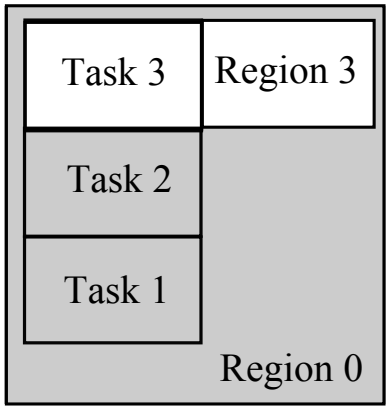
ARM Memory Protection Unit



Task 1 running



Task 2 running



Task 3 running

(User access)

(Privileged access)

ARM Memory Management Unit

- része az MPU;
- a lapleíró bejegyzésben szerepelnek hozzáférést szabályozó információk, hasonlóan mint ahogy az X86 processzornál láttuk.