Számítógép-architektúrák

Tárolókezelési módszerek
(vázlat)

A témához ajánlott elolvasni K2 megfelelő fejezetét. Az óravázlatban hivatkozott ábrák is innen származnak és a **`Tarolokezeles.pdf`** fájlban találhatók meg. (A könyv elérhető az Egyetemi Könyvtárban, illetve anyaga elektronikusan megtalálható az Interneten is (I1), ennek szerzői jogi státusza azonban tisztázatlan.)

Tömbkapcsolás

Probléma: A megcímezhető memóriatartomány mérete kisebb a szükségesnél (például 16 biten 64kB címezhető meg, de nagyobbra van szükség).

Megoldás alapötlete: A szükséges méretű memóriát a processzor által megcímezhető méretű tömbökre bontjuk (n darab ilyen tömb lesz), a processzorhoz pedig egy perifériát illesztünk, amibe **`out`** utasítással kivisszük az általunk éppen használni kívánt tömb sorszámát. A megoldást bemutatja a 2.1. ábra.

Az átkapcsolás egyszerű, de a megoldás rugalmatlan.

Finomítás: A processzor által megcímezhető címtartomány egy része ne legyen átkapcsolható, ez alkalmas paraméterátadásra, megszakítások kezelésére, stb.

Feladat: Illesszünk 8085' processzorhoz a 0000h címre 16kB EPROM-ot, a 4000h címre 16kB RAM-ot, a 8000h címre pedig 8x 32kB RAM-ot úgy, hogy a 00h címre beírt 0-7 közötti értékkel lehessen közülük a megfelelő sorszámút engedélyezni. (Az most nem követelmény, az a 00h címről az aktuális érték visszaolvasható is legyen.)

Indexelt leképzés

Az indexelt leképzés célja is a címtartomány kiterjesztése. A tárolót rögzített méretű lapokra bontja, ezek közül egyszerre néhány érhető el. A megoldást a 2.2. ábrán mutatjuk be.

A processzor által kiadott *logikai cím*et (LA: *Logical Address*) két részre bontjuk: *index* (x bites) és *eltolás* (d bites) mezők.

Az index mező x bitjével az indexregiszter tömb 2^x regisztere közül választunk ki egyet. Ennek a regiszternek két mezője van: a z bitből álló *vezérlés* mezőben jogosultságok adhatók meg, az n bites ($n>x$) címkiterjesztés mező pedig a kiválasztott lap kezdőcímét adja meg az operatív tárban. A lapon belüli címzésre a logikai cím eltolás mezőjének értékét használjuk. A fizikai cím (PHA: *Physical Address*) mérete így $n+d$ bit.

A teljes 2^{n+d} méretű tárból egyszerre így is csak 2^{x+d} méretű részt látunk, de az indexregiszterek feltöltésével rugalmasan választhatjuk ki, hogy mely 2^x számú (2^d méretű) lapot kívánjuk látni.

Az indexregisztereket perifériaként írhatjuk.

Itt is szükséges néhány lapot (pl. megszakításrutinok, az indexregisztereket kezelő operációs rendszer funkciók tárolására) állandóan elérhetővé tenni.

Feladat: Adott egy 8085' processzor és egy 64 elemű indexregiszter tömb. Az indexregiszterek 8 bites vezérlés és 16 bites címkiterjesztés mezőből állnak. Az indexregiszter tömb 64 perifériacímet foglal el, és az indexregiszterek: vezérlés, címkiterjesztés alsó bájt, címkiterjesztés felső bájt sorrendben írhatók.

Mekkora lapokat használjunk, és így összesen legfeljebb mekkora operatív memóriánk lehet?

Virtuális tárkezelés

Probléma: a processzor által megcímezhetőnél kisebb memória áll rendelkezésünkre.

A teljes megcímezhető tárterület tartalmát háttértárolón tároljuk, ennek egy időben csak egy (kis) részét tudjuk leképezni az operatív tárolóra. A tárkezelő egység (MMU – *Memory Management Unit*) feladata, hogy a felhasználó számára láthatatlan módon a felhasználó által éppen elérni kívánt részeket a háttértárolóról az operatív tárolóba behozza. *Lapszervezés* esetén a memóriát fix méretű lapokra, *szegmensszervezés* esetén változó méretű szegmensekre bontva kezeljük.

Lapszervezésű virtuális tároló

A lapszervezésű virtuális tároló működésének kulcsfontosságú eleme a *laptábla*, amelyet (nagy mérete miatt) az operatív memóriában helyezünk el. Kövessük nyomon a működését a 2.3. ábrán!

A *virtuális cím* (VA: Virtual Address) alkalmas az egész (háttértárolón elhelyezkedő) virtuális memória megcímzésére. A virtuális cím két mezőből áll: *lapsorszám* (PN: Page Number) és lapon belüli *eltolás* (D: Displacement).

A laptábla operatív memóriabeli kezdőcímét a *laptáblamutató* (PTP: Page Table Pointer) tartalmazza, amihez hozzáadva egy adott virtuális cím lapsorszám részét a *lapcímmutató* (PAP: Page Address Pointer) értékét nyerjük, ami egy laptábla bejegyzés operatív tárbeli kezdőcíme. Egy ilyen bejegyzés két részből áll: *lapkezdőcím* (PA: Page Address) és *vezérlés* mezők. A vezérlés mezőben a laphoz különféle jogosultságokat adhatunk meg, és ennek a mezőnek a megfelelő bitjéből derül ki az is, hogy a kívánt lap éppen bent van-e az operatív memóriában:

- Amennyiben igen, akkor a lapkezdőcímhez értékéhez a virtuális cím eltolás mezőjének értékét hozzáadva nyerjük azt *fizikai cím*et (PHA: Physical Address) ami megadja a kívánt információ helyét az operatív memóriában.
- Ha a lap nincs bent, akkor a MMU feladata, hogy az operációs rendszerrel behozassa azt a háttértárolóról. Ehhez esetleg szükség lehet valamely lapnak a háttértárolóra való kiírására (aminek a helyére behozzuk).

Azt az esetet, amikor az elérni kívánt lap nem található az operatív memóriában, *laphibá*nak (page fault) nevezzük. A laphiba NEM kezelhető megszakítással, ugyanis megszakításnál a végrehajtás alatt álló utasításnak be kell fejeződnie, itt azonban erre nincs lehetőség. Ezt az esetet, amikor laphiba miatt az utasítás végrehajtása kényszerűen félbeszakad (felfüggesztődik), *kivétel*nek (exception) hívjuk. A processzornak képesnek kell lennie a félbeszakadt utasítás végrehajtását (megfelelő állapotmentéssel) később, a kívánt lap behozása után folytatni.

Annak a döntésnek a támogatására, hogy egy lapot melyik lap helyére hozzunk be, több információt is felhasználunk. Minden lapról nyilvántartjuk, hogy milyen régen használtuk utoljára – erre elvileg egy számlálót használhatnánk, de egyszerűségi megfontolásból inkább csak egy bitet (accessed) használunk, amit periodikusan törlünk, valamint azt is, hogy módosult-e a háttértárolóról való behozása óta (dirty bit). (Ezek a bitek is a vezérlés mezőben helyezkednek el.) Lehetőség szerint egy nem accessed és nem dirty lap helyére hozunk be másikat, amit így nem kell kiírnunk.

Amennyiben a laphibák túl gyakoriak (például az összes memóriahivatkozások több, mint 1%-a laphibát okoz), akkor a rendszer működése során az idő túl nagy részében fog a laphibák kezelésével foglalkozni, ezt vergődésnek (trashing) nevezzük (W1). Ahhoz, hogy egy task kielégítő sebességgel fusson, a végrahajtáshoz éppen szükséges lapjainak (working set: W2) nagy valószínűséggel az operatív memóriában kell lenniük.

A lapméret megválasztásánál figyelembe vesszük, hogy a háttértár elérési ideje a néhány ms nagyságrendjébe esik, az átvitel viszont meglehetősen gyors, így ma kb. 4kB a szokásos lapméret.

Mivel a laptáblákat az operatív memóriában tároljuk, a virtuális címhez a fizikai cím meghatározása (címfordítás) miatt minden memóriaművelet két memória hozzáférést igényelne, ami felére csökkentené az operatív tár elérésének sebességét. Ennek kiküszöbölésére használják a *Translation*

*Lookaside Buffer*t (TLB) amelyet az MMU-ban (ami ma tipikusan a CPU része) egy néhány (néhányszor tíz) bejegyzést tárolni képes tartalom szerint címezhető memóriában (CAM Content Addressable Memory) helyeznek el: ez tárolja a néhány utoljára kiszámított PN – PA párost, és PN alapján villámgyorsan megadja PA-t.

Szegmensszervezésű virtuális tároló

A változó méretű szegmensek használata jobban igazodik egy feladat szerkezetéhez (függvények, adatszerkezetek egymástól való szétválasztása), viszont a kezelésük lényegesen bonyolultabb, ezért csak nagy gépeken szokták használni. Mi bővebben nem foglalkozunk a témával, érdeklődők K2-ben elolvashatják.

Cache tároló

Probléma: az operatív tároló nem elég gyors (a processzorhoz képest).

Ez a probléma az operatív tároló méretének növekedésével szükségszerűen előáll, hiszen a tároló kapacitásának négyzetgyökével arányos az elérési ideje. (Memóriáknál láttuk, hogy miért.)

A nagy és lassú operatív tár átlagos elérési idejét jelentősen csökkenthetjük azzal, hogy a processzor és az operatív memória közé egy kicsi és gyors *rejtett tár*at (cache) iktatunk be. A rejtett tár az operatív tár bizonyos részeinek másolatát tartalmazza, működése a felhasználó számára átlátszó, azaz a felhasználó közvetlenül nem érzékeli a létét, csupán az operatív tárat látja gyorsabbnak: amikor a keresett információ a cache-ben található, lényegesen rövidebb az elérési idő, amikor nem, akkor meg kell várni az operatív tárat.

Vegyük észre, hogy a virtuális tárkezelés és a cache alkalmazása a tárhierarchia más szintjén ugyan, de ugyanazt szolgálja: egy nagyobb méretű, lassabb tárat (majdnem) egy kisebb méretű, gyorsabb tár sebességével szeretnénk elérni!

Mivel azonban az elérési idők nagyságrendi eltérése különböző (cache és operatív memória mindegyike félvezető, operatív memóriával szemben a háttértár mechanikus eszköz), így míg a virtuális memóriánál 1% körül van az elfogadható laphiba arány, a cache esetében ez (*cache miss* rate – a találat pedig: *cache hit*) 10% körül van, és a szokásos blokkméret is jóval kisebb, 16 vagy 32 bájt körül van. (A cache blokkjait gyakran úgy hívják, hogy *cache line*.) A cache esetén a gyakorlatban több szintű hierarchiát is használnak. (Lásd DEC Alpha 21164-nél 3 szint: W4)

Több szempontból is értelmes külön utasítás és adat cache-t használni (2.5. ábra).

A külvilág szempontjából a cache vagy az operatív tár vagy processzor részének tekinthető, mindkét megoldásnak megvannak az előnyei és hátrányai (2.6. ábra).

A cache vezérlésnek többféle módja van (és ezeknek más-más előnye és hátránya van, ami alapján a gyakorlatban adott célra egyiket vagy másikat választhatjuk), most mi azzal foglalkozunk, amelyik tartalom szerint címezhető tárolót (CAM) alkalmaz. Működésének egyik lényeges lépését mutatja be a 2.7. ábra. Itt látható, hogy egy cache blokk memóriabeli címét (BAM) a CAM egyszerre összehasonlítja az összes benne tárolt bejegyzéssel és találat esetén azonnal megadja, hogy az adott blokk hol helyezkedik el a cache-ben (BAC). Az ábra nem tünteti fel, hogy mi történik, ha nem volt egyezés. Ilyenkor célszerű a legrégebben használt blokk helyére betölteni az újat. Ennek egy jó megvalósítása a CAM shiftregiszterként való használatán alapul: egyezés esetén az egyező sort kiveszik, a fölötte levő összes sort eggyel lefele léptetik és az egyező sort a legfelső sor helyére teszik. Ha nincs egyezés, akkor az összes sor eggyel lejjebb lép, és a legfelső sorba kerül az éppen a cache-be behozott blokkot leíró BAM-BAC páros. Így a legrégebben használt (LRU: Least Recently Used) blokk helyére hoztuk be az újat.

Megjegyzés: a 2.7. ábra úgy tiszta és érthető, ha a VA helyére PHA kerül: az operatív tárbeli fizikai cím! Az egy másik kérdés, hogy a virtuális tárkezeléssel való együttes használat érdekében a címfordítás kiküszöbölésére VA is tárolható. Ekkor azonban oda kell figyelni, hogy mi történik, ha

az adott cache blokkot tartalmazó lap (vagy szegmens) kikerül az operatív memóriából!

A fenti megoldást *teljesen asszociatív* (fully associative) leképzésnek nevezzük. A másik véglet a *direkt leképzés* (direct mapping), amikor az operatív memóriát a cache memória méretével megegyező méretű részekre bontjuk, és ezek cache line méretű blokkjainak fixen adódó helye van a cache-ben. Ilyenkor az operatív memória két különböző cache memória méretű részének azonos sorszámú cache line méretű blokkja ugyanarra a cache line-ra képeződik le, így az egyiknek a cache-be történő behozatala szükségszerűen kiüti a másikat! Ilyenkor csak azt kell nyilvántartanunk, hogy az adott cache line az operatív memóriának éppen melyik cache méretű részéből származik. A megoldás egyszerűsége miatt olcsóbb, de kevésbé hatékony (kisebb a cache hit rate).

A fenti megoldás javítása az *N-utas asszociatív* (N-way set associative) leképzés: ilyenkor a direkt leképzés hátrányait úgy csökkentjük, hogy N darab cache memóriát használunk, így az egymást kiütő blokkokból legfeljebb N példány még egyidejűleg bent lehet az N darab cache memória valamelyikében (a neki megfelelő rögzített helyen). Ekkor minden egyes cache sorhoz kell egy N méretű tartalom szerint címezhető memória, aminek a segítségével mind az N darab cache memóriára nézve egyszerre meg tudjuk mondani, hogy egy adott operatív tárbeli címen található blokk benne van-e. Tipikusan 2 vagy 4 utas megoldást szoktak használni.

További kérdés, hogy mi történjen íráskor (write policy). Erre több megoldást is használnak:

- write through: az írási művelet mind a cache-ben, mind az operatív tárban megtörténik
- write back: íráskor a cache blokk dirty jelölést kap, és később kerül csak átírásra az operatív tárban – legkésőbb a cache blokk felülírásakor meg kell történnie!

A cache téma iránt mélyebben érdeklődőknek ajánljuk W3 tanulmányozását.

Források:

K2: Németh Gábor, Horváth L.: Számítógép architektúrák, 2. kiadás, Akadémiai Kiadó, 1993. (2.1. Tárolókezelési módszerek 10-25. oldal)

I1: http://www.doksi.hu/get.php?lid=166

W1: http://en.wikipedia.org/wiki/Thrash_(computer_science)

W2: http://en.wikipedia.org/wiki/Working_set

W3: http://en.wikipedia.org/wiki/CPU_cache

W4: http://en.wikipedia.org/wiki/Alpha_21164