PANNON EGYETEM, Veszprém
Villamosmérnöki és Információs Rendszerek Tanszék

Digitális Rendszerek és Számítógép Architektúrák

8. előadás: Memóriák

Előadó: Vörösházi Zsolt

Jegyzetek, segédanyagok:

- Könyvfejezetek:
  - http://www.virt.vein.hu
    ⇒ Oktatás ⇒ Tantárgyak ⇒ Digitális Rendszerek és Számítógép Architektúrák
    (Nappali)
    (chapter07.pdf
    + /addtitions/ - további részek, amik a könyvben nem szerepelnek)
- Fóliák, óravázlatok .ppt (.pdf)
- Feltöltésük folyamatosan

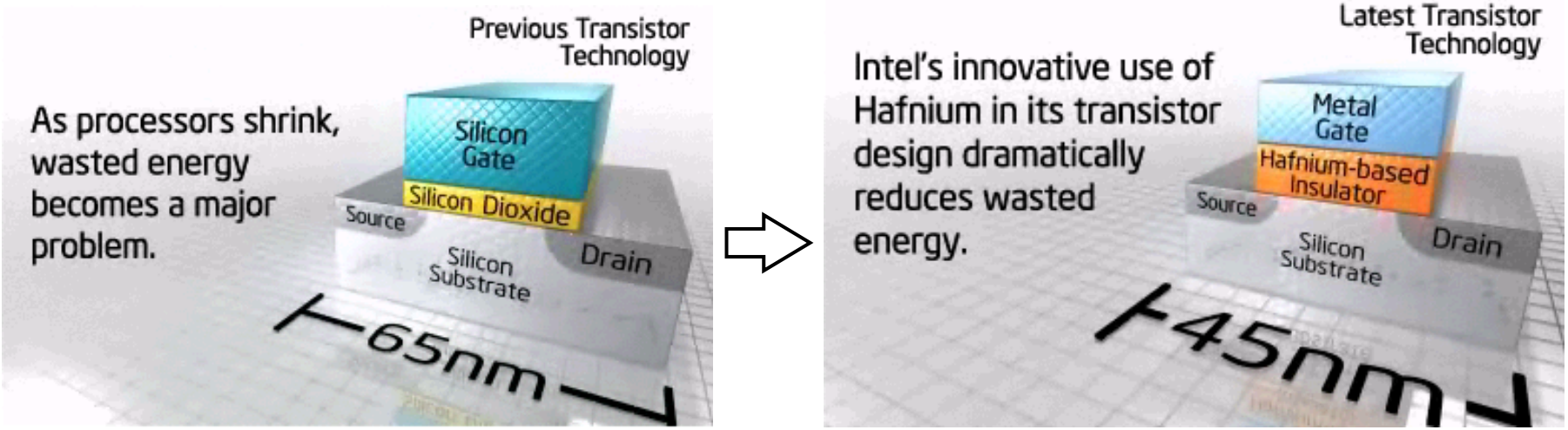
Memóriák, memória rendszerek

- RAM, ROM memóriák: nagyméretű, lineáris tároló tömb
- Virtuális memóriakezelés
  - Lapozás
  - Szegmentálás
- Cache memóriák

Mikro-minimalizálás elve:

- **Gordon Moore törvénye (1965)**: rendkívüli jelentőséggel bír a memóriák és a félvezető áramkörök méretcsökkenése esetén.
  - Tanulmány: félvezető áramkörök fejlődése (prognózis)
  - A technológia fejlődésével minden 18 hónapban az 1 felületegységre (mm^2 Si) eső tranzisztorok száma közel megduplázódik (integritási sűrűség)
  - Ezzel szemben az eszközök ára csökken, vagy stagnál.
- Jelenleg (2009. április):
  - 3D rétegszerkezet szilíciumon
  - Működő 45nm csíkszélességű tranzisztor (high K fém dielektrikum, Hafnium) pl: Intel újgenerációs processzoraiban
  - Metal gate (a PolySi –ot váltja fel)

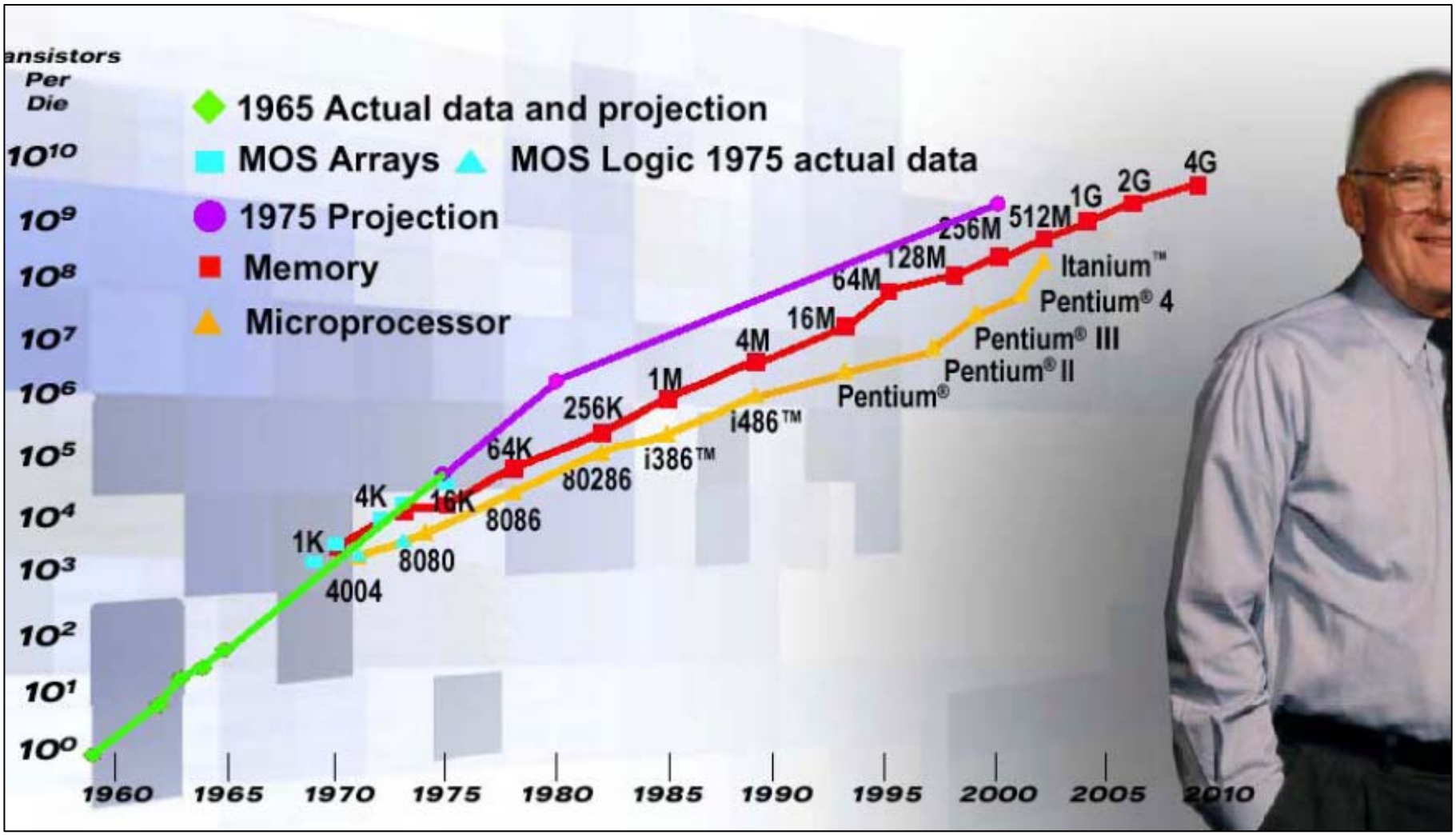
Intel 45nm High-K technológia



- Intel Core® mikro-architektúra
  - 2-, 4- magos technológia (Core2, Penryn, Xeon)
- 400-800 millió tranzisztor
- Gate szivárgási áramának (leakage current) csökkenése
- Kisebb energia fogyasztás
- Gyorsabb kapcsolási sebesség

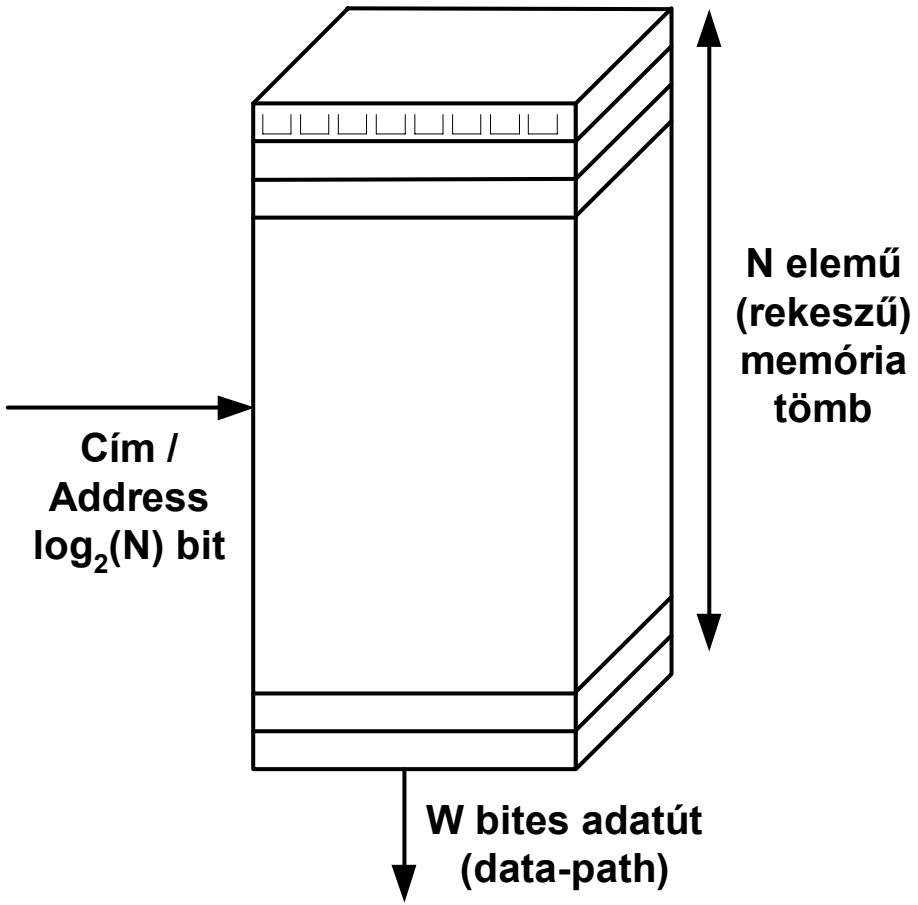
- 2009-re 32nm technológia, gyártás (~ 2 billió tranzisztor)

Moore törvénye – „Road Map”



*Die: Si felület egysége (mm²) - megvalósított tranzisztorszám

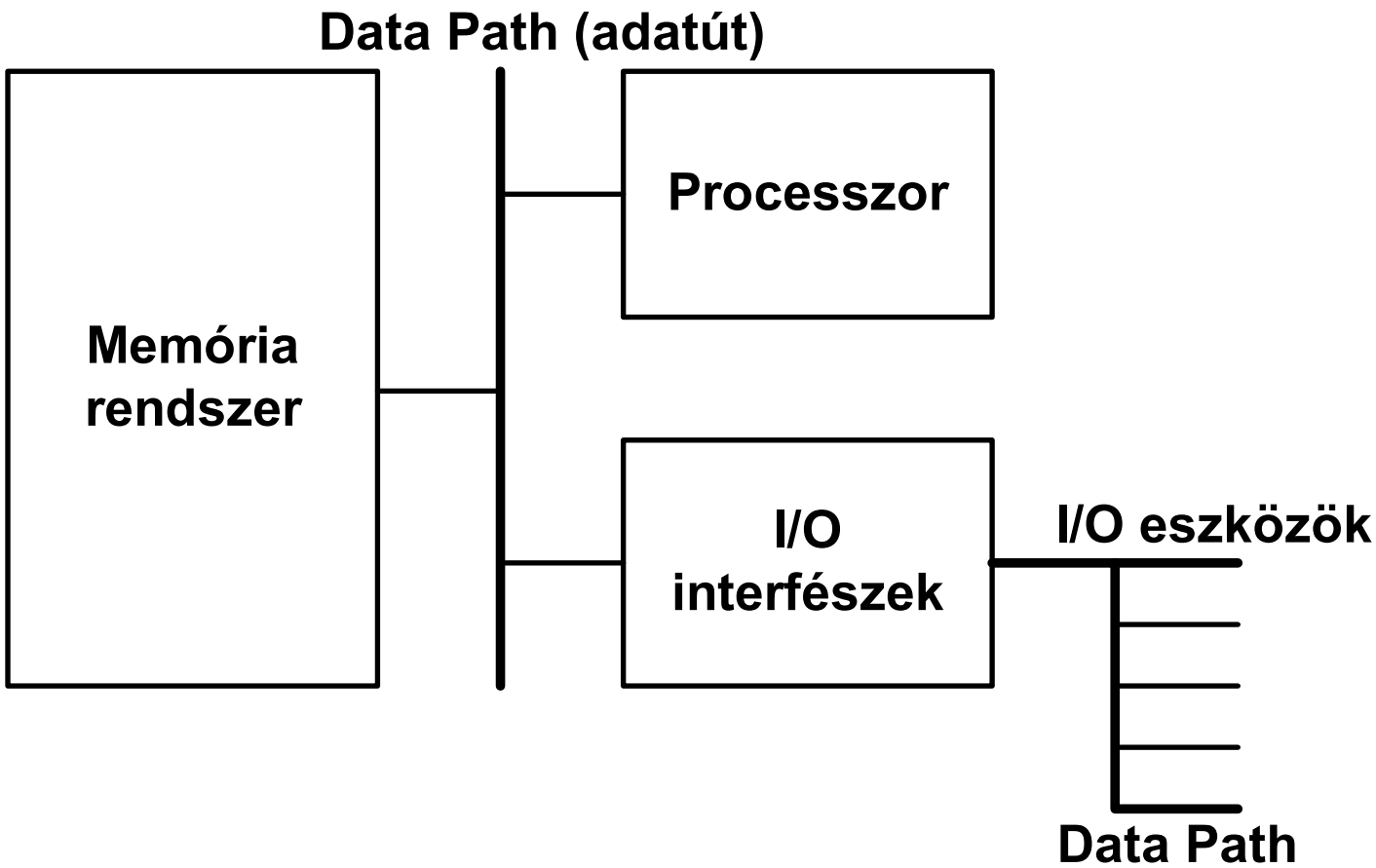
Általános memória modell:



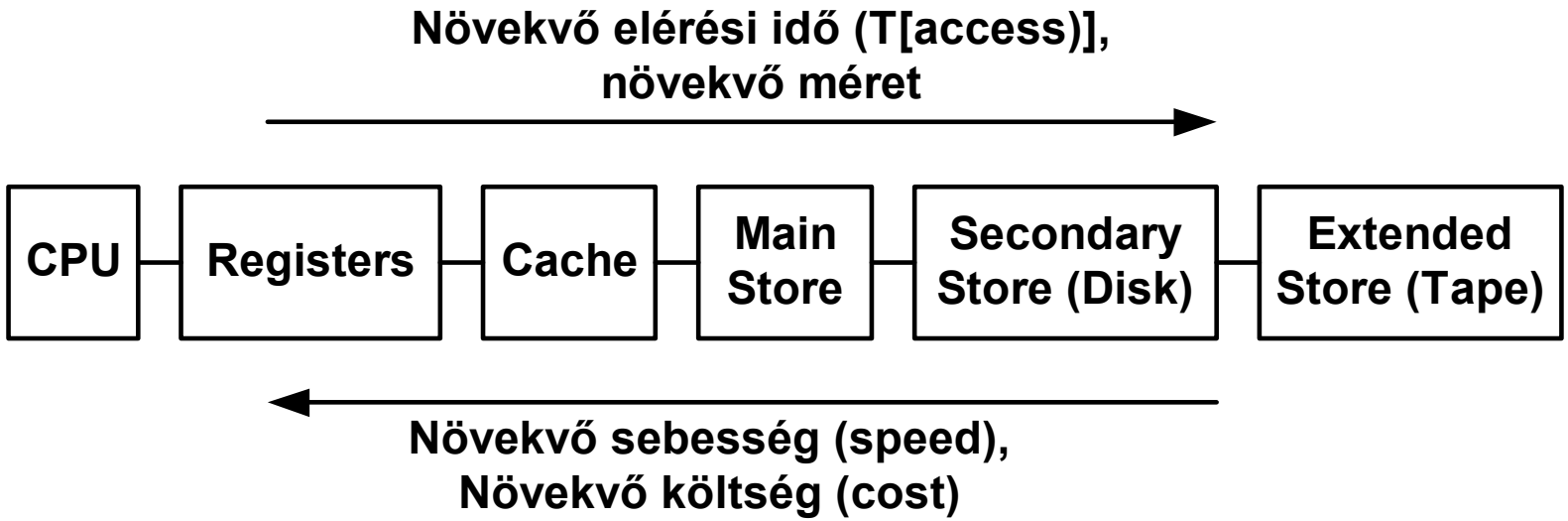
Fontos paraméterek:

- memória mérete (N elemű, rekeszű)
- címvonalak száma N függvényében: $\log_2(N)$
- w: adatvonal szélessége
- memóriaszervezési módszer
- sebesség
- T_a: minimális (access time) elérési idő: az a legrövidebb idő, ami a cím kiadásától az adat megjelenéséig tart.
- T_{ct}: memória (cycle time) ciklus idő: minimális idő két elérés között (burst módban)

Memória – CPU – I/O interfészek kapcsolatának blokkdiagramja:



Tár-hierarchia:



- **Regiszterek**: a gyorsabbak, mivel fizikailag is a legközelebb vannak a processzorhoz: kis számú, nagyon gyors regiszterbankot használunk. De nagyon drágák. (ns)
- **Cache memória:** a leggyakrabban előforduló adatokat ill. utasításokat tárolja, növeli a műveleti (végrehajtási) sebességet
- **Main Store/Memória**: programok (kód) és az adatok itt tárolódnak, együtt vagy külön (Neumann v. Harvard elvű architektúra szerint!). Elegendő mennyiség szükséges az OS (operációs rendszer) zavartalan működéséhez. (ns-os elérési idő)
- **Secondary Store/Disk**. háttértárolók, másodlagos-merevlemezes tároló elemek, nagy kapacitásúak. Fájlok, könyvtárak, swap-terület. Hosszabb időre tárolunk. (ms-os elérési idő)
- **Extended Storage/Tape**: szalagos, lemezes meghajtók, mágneslemezek. Nagyon lassúak.

Memóriafajták elérésük és hozzáférésük szerint:

- **RAM:** Random Access Memory: Véletlen hozzáférésű memória, írható és olvasható
- **ROM:** Read Only Memory: Csak olvasható memória (véletlen elérésű - RAM tulajdonság)
- SAM Volatile Memória: Serial Access Memory. Soros elérési idejű, tápfesz. kimaradásakor felejtő ún. „elpárolgó” tartalmú memória
- PROM: Programmable ROM: programozható vezetékek közötti ellenállások/biztosítékok átégetésével ill. átkötésével.
- EPROM: elektromosan programozható fel, de csak UV-fény segítségével (kis kvarc ablakon keresztül) törölhető.
- EEPROM: elektromosan programozható és törölhető
- **SRAM:** Statikus RAM: tápfesz. nélkül elveszti tartalmát, gyors, nem kell frissíteni, de nagy hegyet foglal (min. 6 tranzisztor)
- **DRAM:** Dinamikus RAM: tápfesz. alatt is frissíteni kell tartalmát, lassú, de kis helyet foglal
- RWM: Read/Write Memory: írható/olvasható memória (spec Ram)
- CAM: Content Address Memory: tartalom szerint címezhető memória (asszociatív memória) Pl :CACHE memória.
- FLASH: memóriakártyák

RAM és ROM memóriák

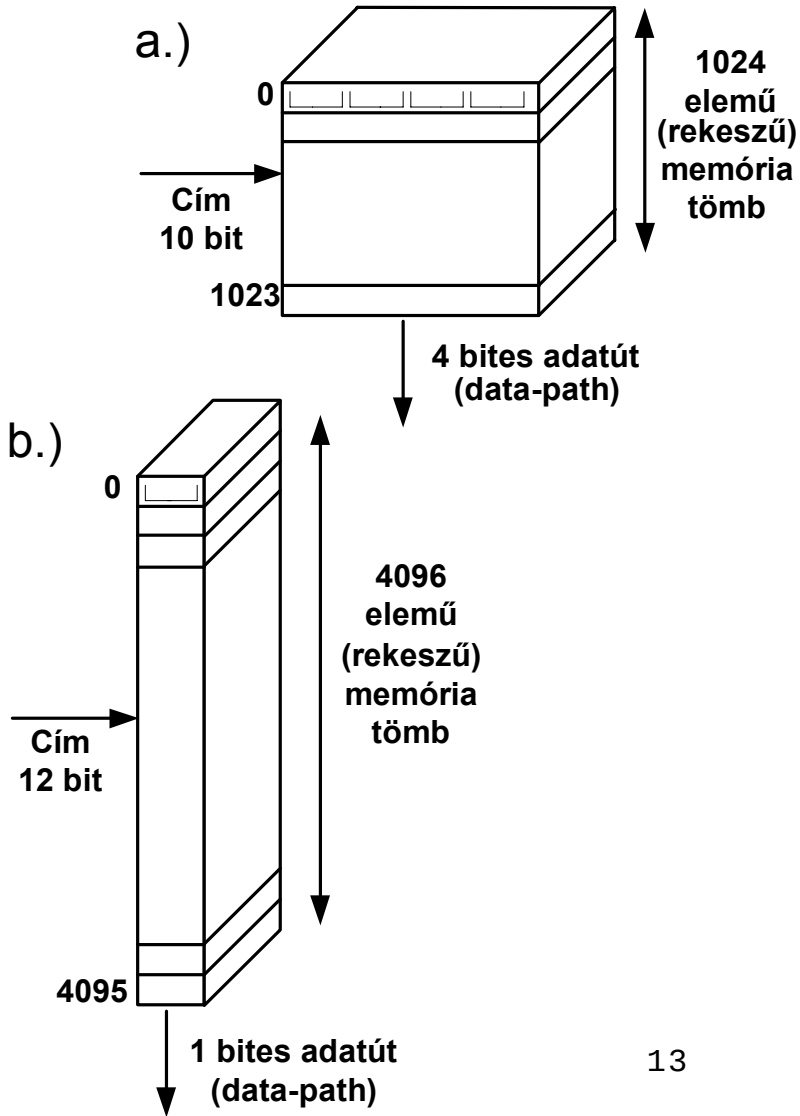
1.) ROM: Read Only Memory

- Csak egyszer írható (általában a gyártó által), utána már csak olvasható memória.
- Kikapcsoláskor megőrzik tartalmukat!
- Szervezése a RAM-hoz hasonló (de itt nem kell Write/Read vonal)
- Fontos alkalmazásai:
  - *Firmware*: elektronikai eszközök (pl. számítógép ROM-Bios funkcióinak tárolására)
  - *Kód konverter*: pl. BCD generáló 7 szegmenses kijelzőre
  - *Logikai függvény generátor*: tetszőleges logikai fgv. szintézisénél (előállításánál) ROM-ban tároljuk az igazságtáblázatot
- Programozható ROM fajtái:
  - PROM*: egyszer programozható
  - EPROM*: programozható, és UV-fénnyel törölhető
  - EEPROM*: elektromosan programozható/törölhető (Flash)

*Lásd: programozási módok

2.) RAM: Random Access Memory

- Véletlen hozzáférésű, írható és olvasható memória.
- Címzésnél: dekódoló és multiplexer áramköröket használhatunk
  - Pl. 10 lábbal → 1024 (1K) cellát tudunk megcímezni
- Jelölés: RAM celláinak száma (4096) / tároló kapacitás 4K. Példák:
  - a.) 4K= 1K x 4 RAM tartalmaz: 1024 számú 4 bites szót (word)
  - b.) 4K= 4K x 1 RAM tartalmaz: 4096 számú 1-bites szót (word) (de itt az 1 bit/szó szervezéssel lábszámot spórolunk!) 12+1



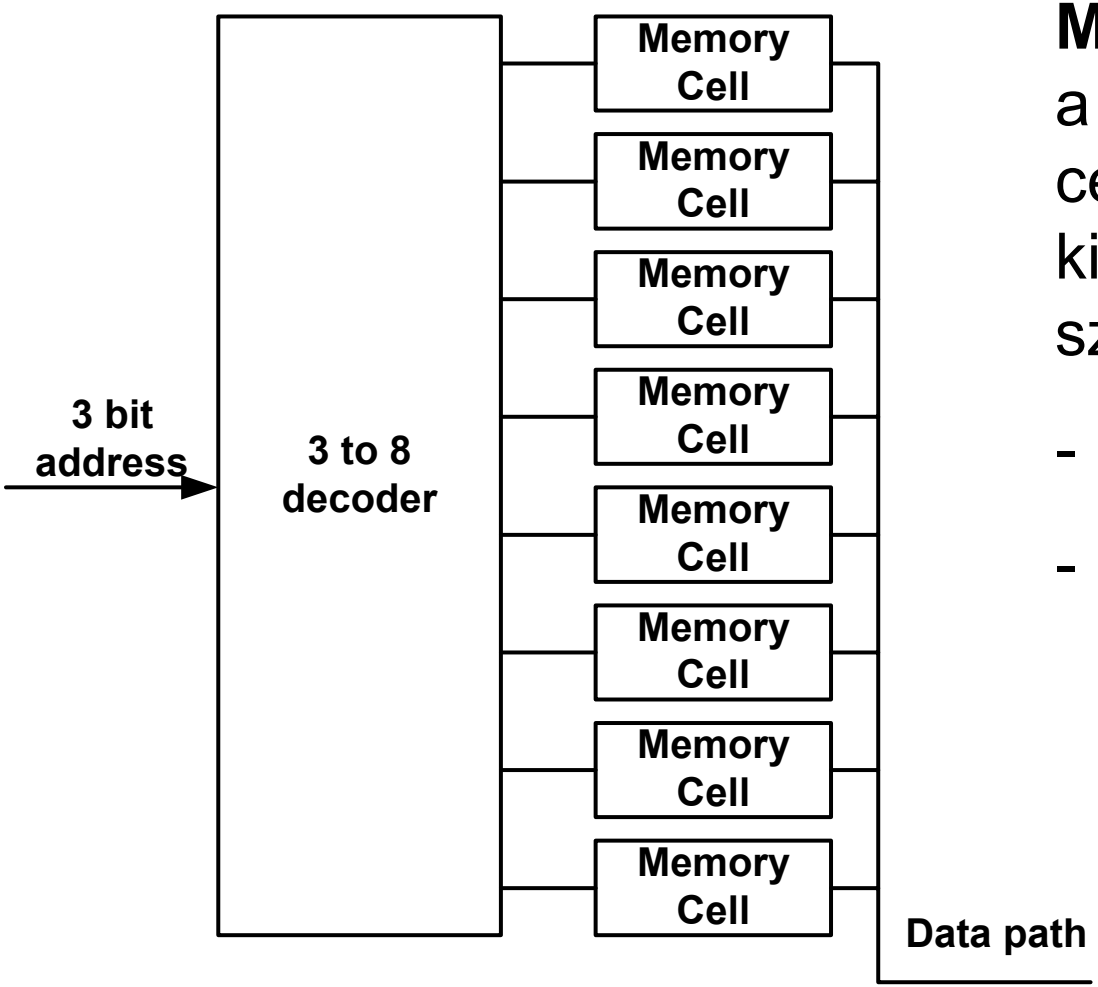
RAM fontosabb időzítési paraméterei

- *T(setup): T(access)* elérési ideje a memóriának
  - (valid) érvényes adat megjelenése előtt a címet stabil értéken kell tartani
- *T(hold): T(release):* amíg az érvényes adatot (ki)tartja az adatvonalon.
  - Ezután felszabadul a vonal és még bizonyos ideig ismét stabil állapotot kell biztosítani a következő címnek.

RAM memória szervezés

- **DRAM-ok** esetén az Address (cím) felbontása:
  - nagyméretű, például „1M x 1” bites memória esetén a címet (20 bites) fel kell bontani sor és oszlopcímekre (idő-multiplexált mód)
    - Row address
    - Column address
  - Ezáltal 2D-s elrendezést kaphatunk.
- **SRAM-ok** esetén általában csak egy jelet (*CS* v. *CE*) használhatunk az adott cella azonosításához (általában nem idő-multiplexál jelek), 1-D szervezés

RAM blokk diagram (1-D szervezés)

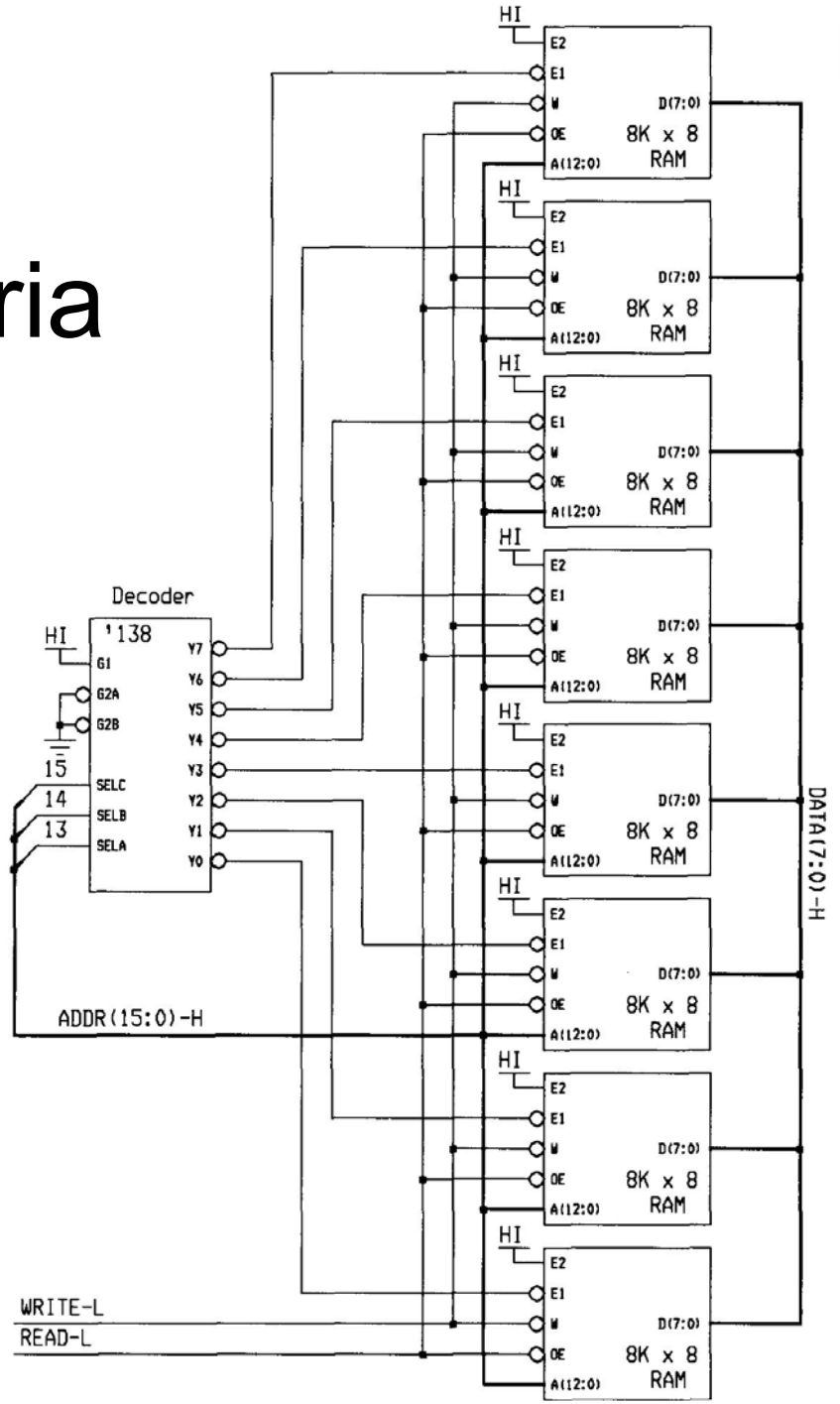


Memória szervezés célja: a nagyméretű (nagyszámú cella) memóriaelem kivezetéseinek (lábak) számának csökkentése!

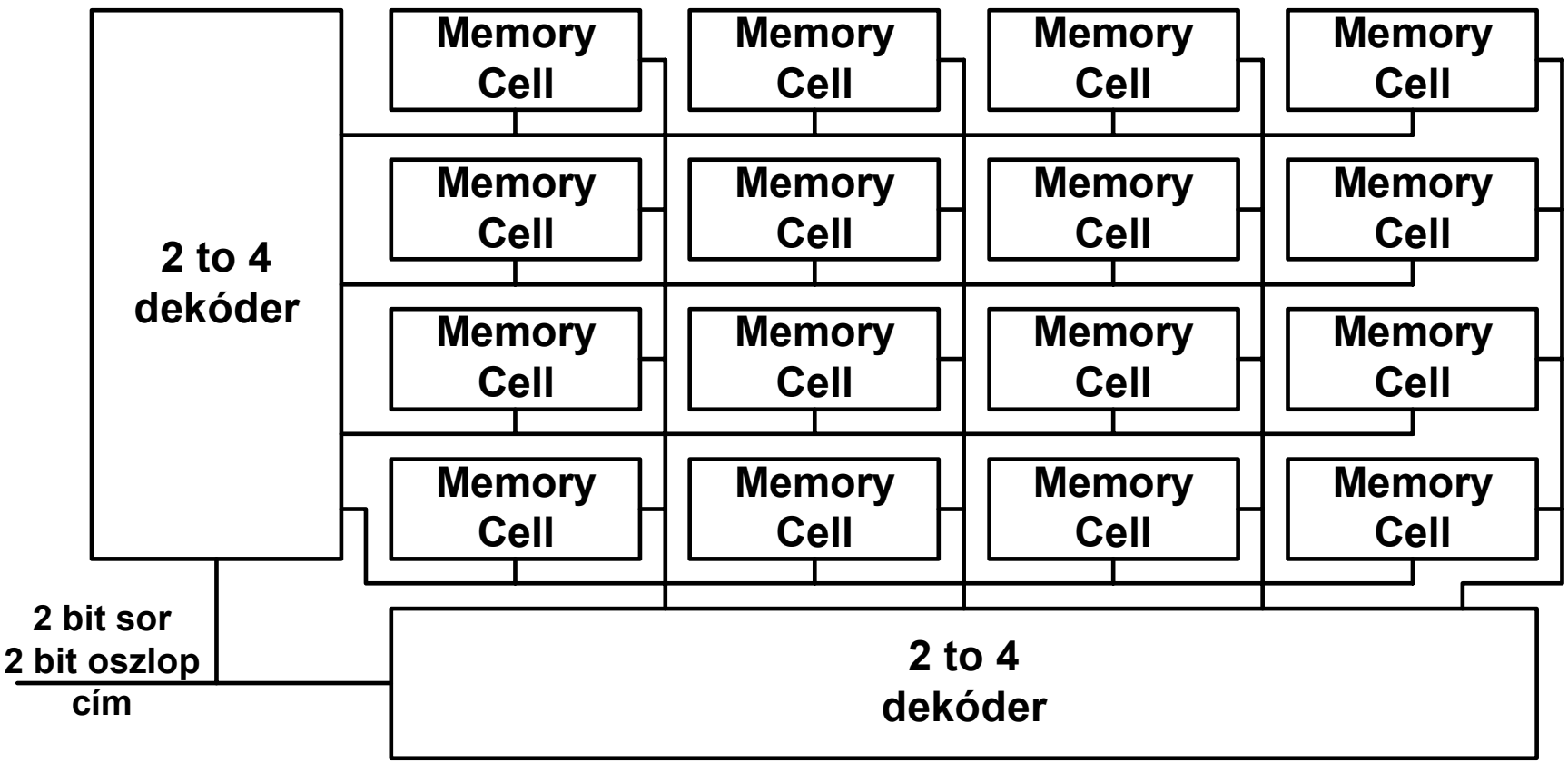
- Dekóder és
- Multiplexer áramkörökkel

Példa: 8Kx8 – 1D szervezésű memória

- 1 RAM cella: 2^16=64k
- Össz: 512 K
- w = 8-bites adatbusz
- N= 16-bites címbusz

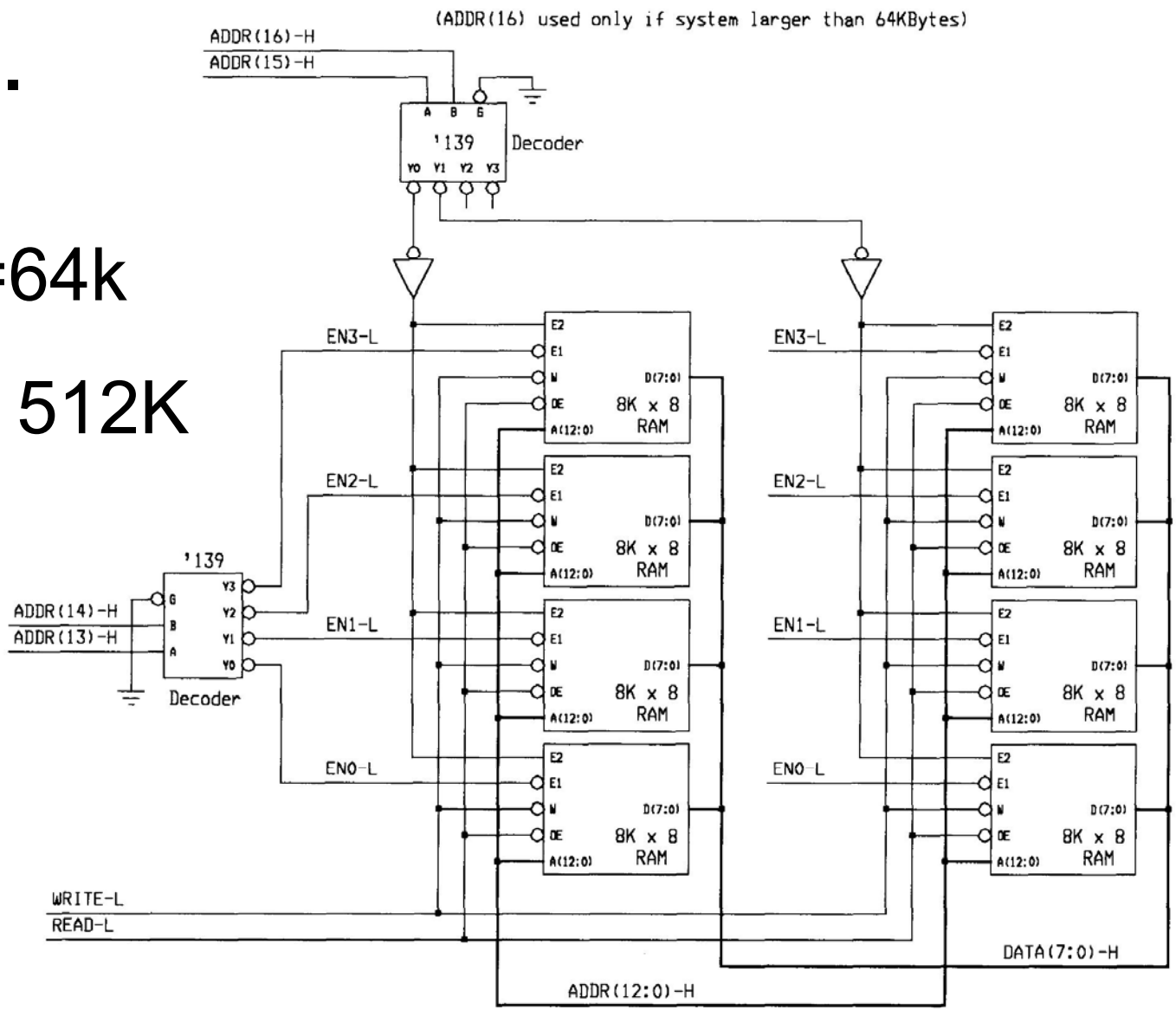


Memória cellák 2-D szervezése

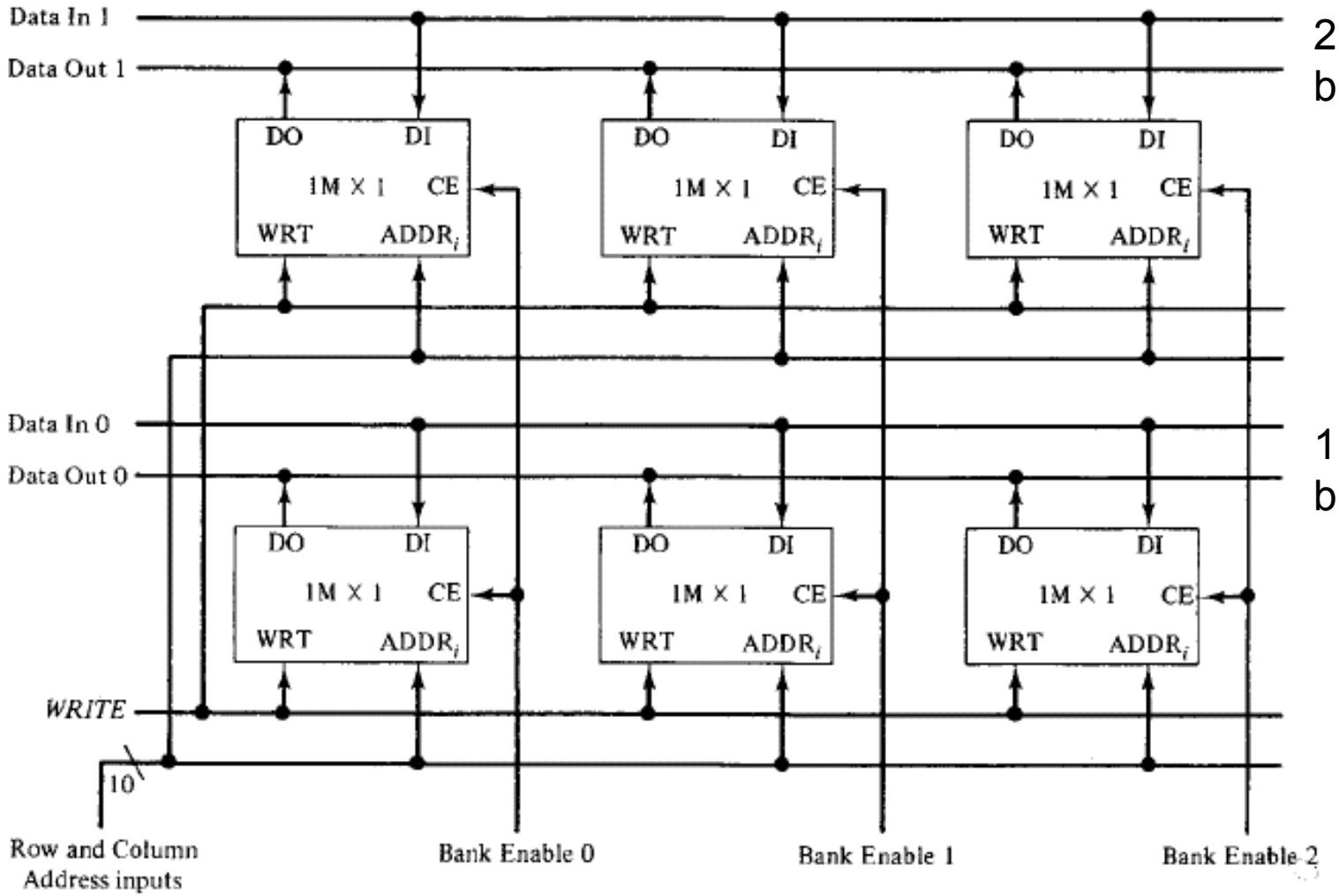


Példa: 8Kx8 2D szervezésű Mem.

- 2^16=64k
- Össz: 512K



Példa: 3Mx2 memória 1Mx1 –ből felépítve



2. bit/word

1. bit/word

Írás:
WRITE='T'

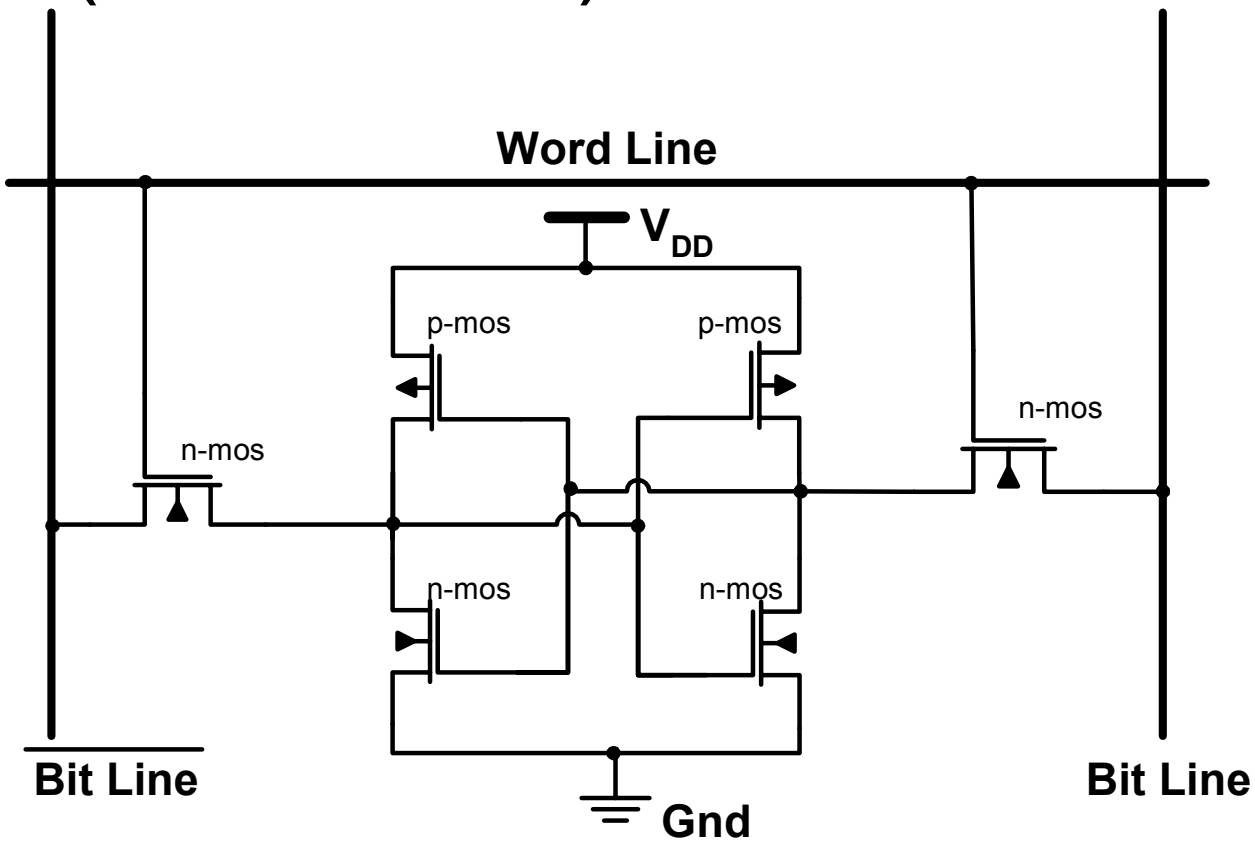
Olvasás:
WRITE='F'

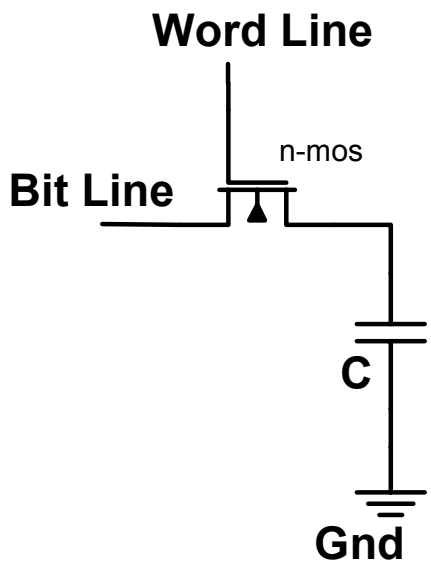
1M = 20 bites cím (10 bites sor + 10 bites oszlopcím) 2D s szervezés
+ 3 Bank: BE engedélyező jel (melyik bank kapcsolódik a buszra)

Aszinkron memória típusok:

- aszinkron SRAM
- aszinkron DRAM

RAM: **SRAM** (**Statikus**) és **DRAM** (**dinamikus**) memória cellák felépítése:





SRAM: n-mos és p-mos (n és p csatornás tranzisztorokból épül fel) 2-2 db, + 2 db áteresztő tranzisztor (össz. 6 CMOS tranzisztor)

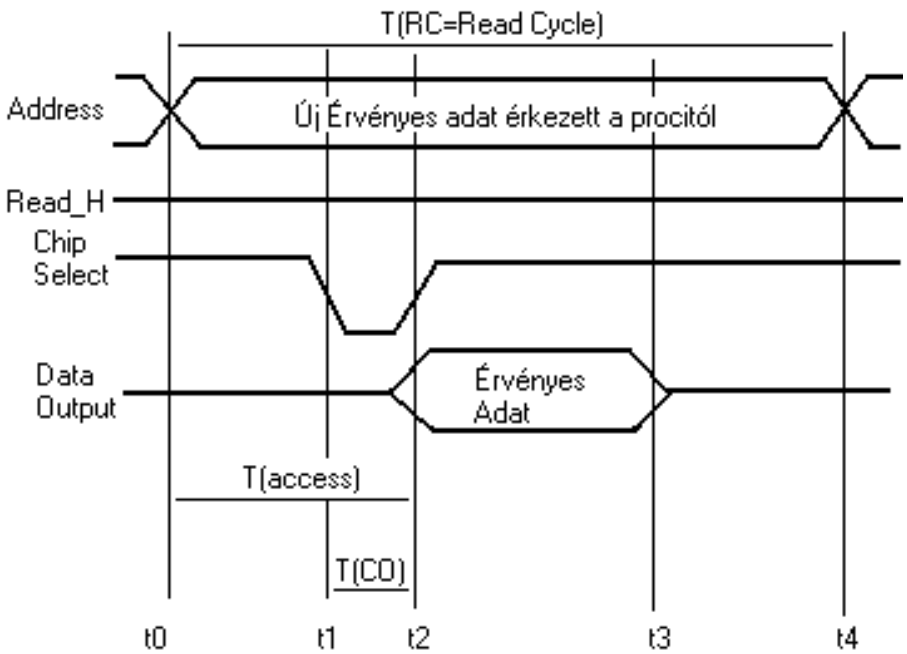
DRAM: Tárolás: egy kisméretű kondenzátoron (C), töltése és kisülése.

a.) SRAM tulajdonságai

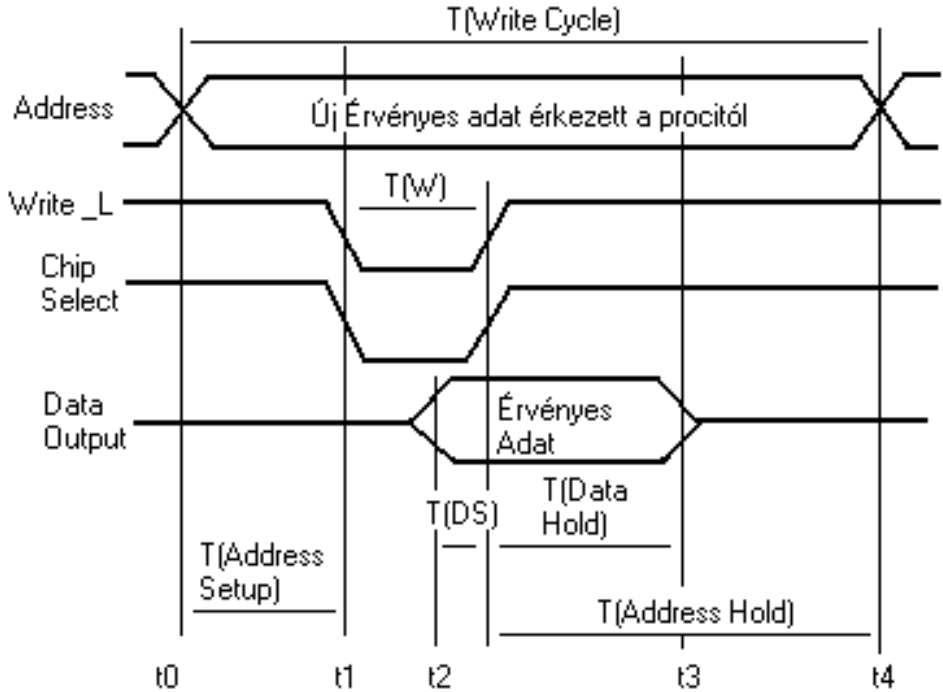
- Az információ a tápfesz. alatt is megmarad, nem kell frissíteni.
- Megvalósítható bipoláris (0,1) SRAM cellával, nMOS tranzisztorokkal, vagy CMOS tranzisztorokból (6 tranzisztor).
- Kisebb kapacitású, de gyorsabb a DRAM-nál, mivel tápfeszültség alatt sem kell frissíteni.
  - SRAM esetén a ciklusidők T(R/W Cycle)≈T(Access Time) közel azonosak.
- Nagy a fogyasztása. Integritási sűrűsége 4x kisebb. Tápfeszültség kikapcsolásával elveszti a tartalmát.
- Felhasználása:
  - Cache memóriákban, digitális oszcillátorokban, logikai analizátorokban, operatív tárakban (memória), merevlemezek gyorsító pufferében, nyomtatók memóriájában.

Aszinkron Statikus RAM idődiagramjai:

Olvasási ciklus



Írási ciklus

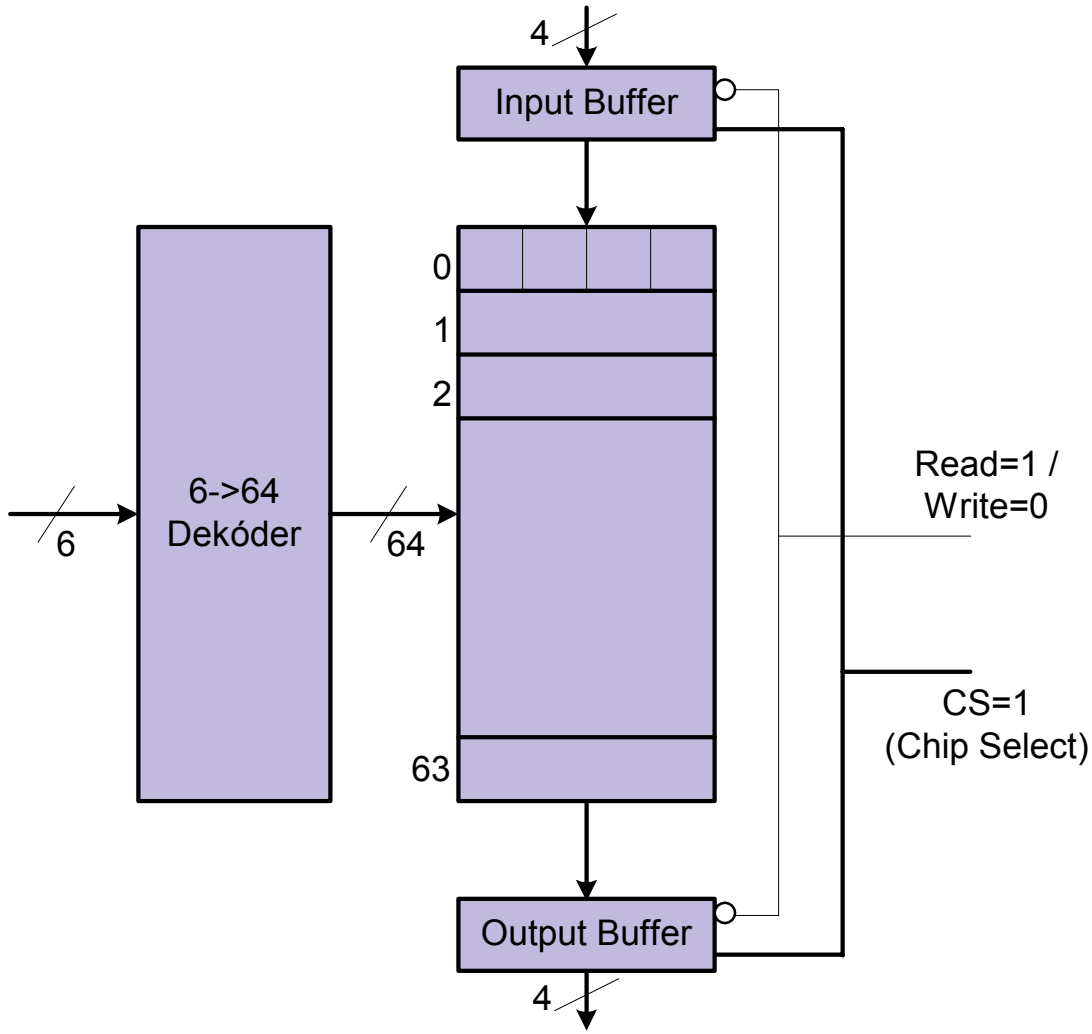


- $\overline{CS}$: Chip Select SRAM cella kiválasztó
- R/$\overline{W}$: Read-H (olvasás), Read-L (írás: Write)

Példa: 64x4-es SRAM felépítése (1-D szervezésű)

- *Feladat*: Mekkora **lábszámot** kell biztosítani az SRAM működéséhez (közös R/W, de különböző data vonalak esetén)?

- **6+4+4+1(CS)+1(R/W)+ 1(Vdd)+1(Gnd) = 18**
- Tehát összesen 18 lábra (pin) van szükség!
- (64 rekesz, 4bit/szó)

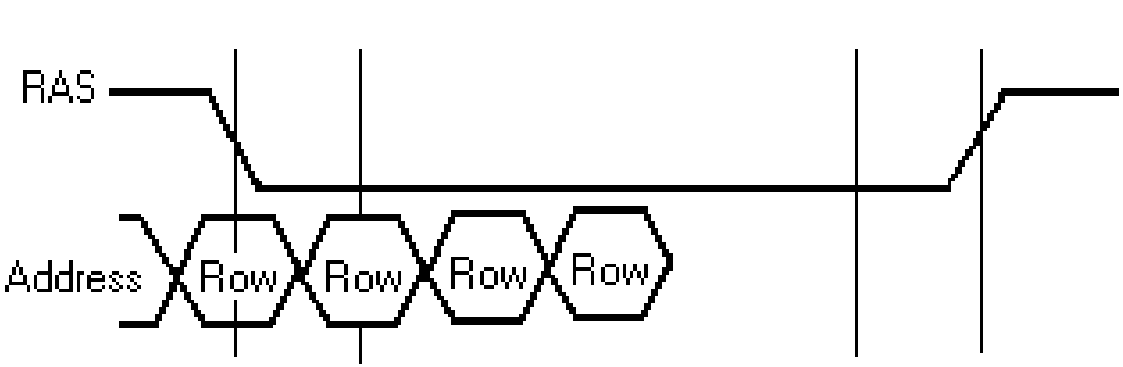


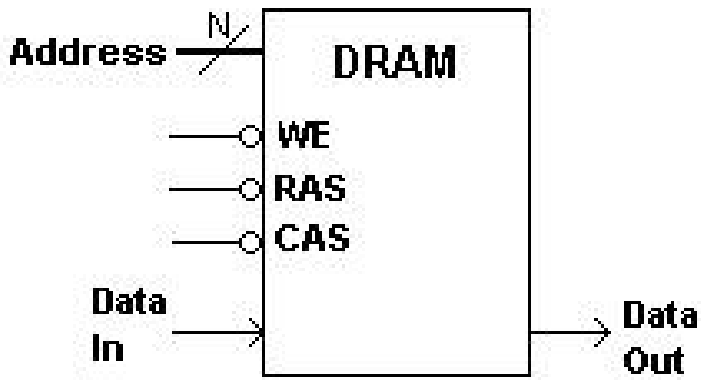
b.) DRAM tulajdonságai

- Általában CMOS technológiával készülnek.
- A tápfeszültség alatt is frissíteni kell *2-10 ms*-ként, mivel idővel elvesztik tartalmukat.
- Kicsi a fogyasztása
- Nagyobb kapacitású, mint az SRAM, de lassabb (frissítés!)
  - A hozzáférési idő kétszer nagyobb a memória R/W ciklusidejénél: 2*T(R/W Cycle)=T(Access Time).
- Egyszerűbb felépítésű (1 tranzisztor + kondenzátor), szemben az SRAM-al. Integritási sűrűsége **4x** nagyobb. (IRAM: az időzítő elektronika a DRAM-ra van integrálva).
- Itt lényegében a CS=Chip Select (korábban CE: Chip Enable!) jelet két részre osztották fel: **RAS=sorkijelölő**, és **CAS=oszlopkijelölő** komponensekre.
- Felhasználása: operatív memória (DDR-, DDR-II, DDR3-SDRAM)

DRAM frissítése

- Kb. 2-10ms-ként kell frissíteni, mivel egy kisméretű kondenzátoron tároljuk az információt, melynek feszültsége idővel exponenciálisan csökken, tehát kisül.



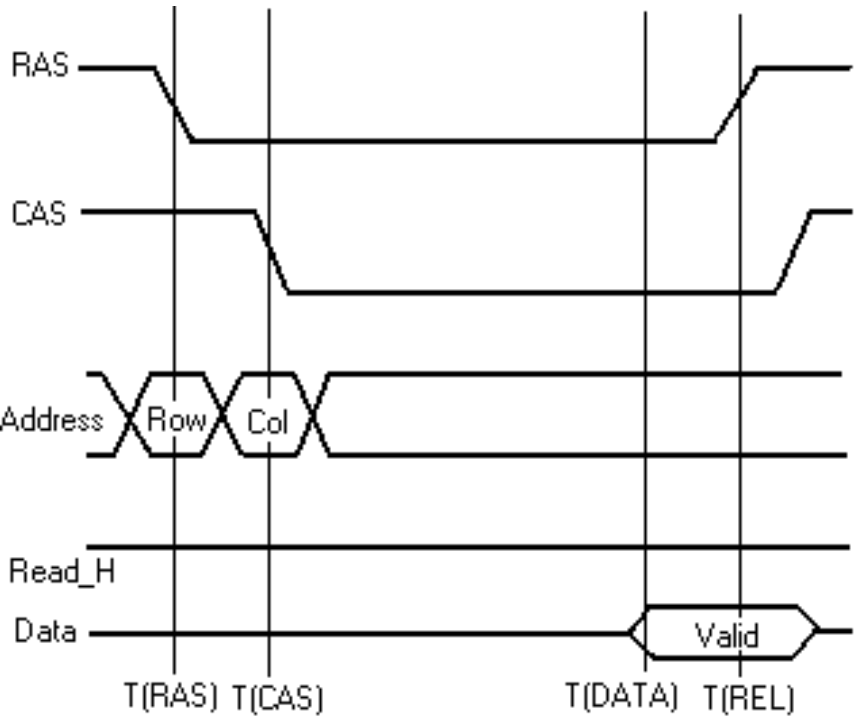


szimbóluma

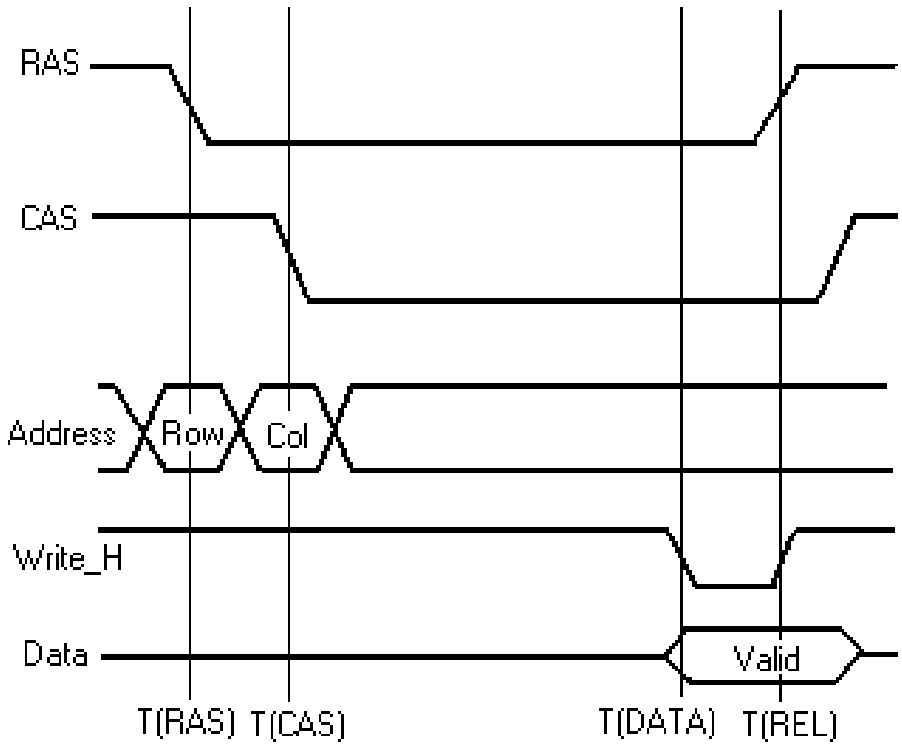
- A **frissítéskor** például először egyetlen CAS oszlopcímet adunk ki, majd a RAS-al az összes sorcímet, hogy az összes cella frissítésre kerüljön. Frissítés történhet a kiolvasási/beírási ciklus végeztével is (még a következő tranzakció előtt).

Aszinkron Dinamikus RAM időd iagramjai:

Olvasási ciklus



Írási ciklus



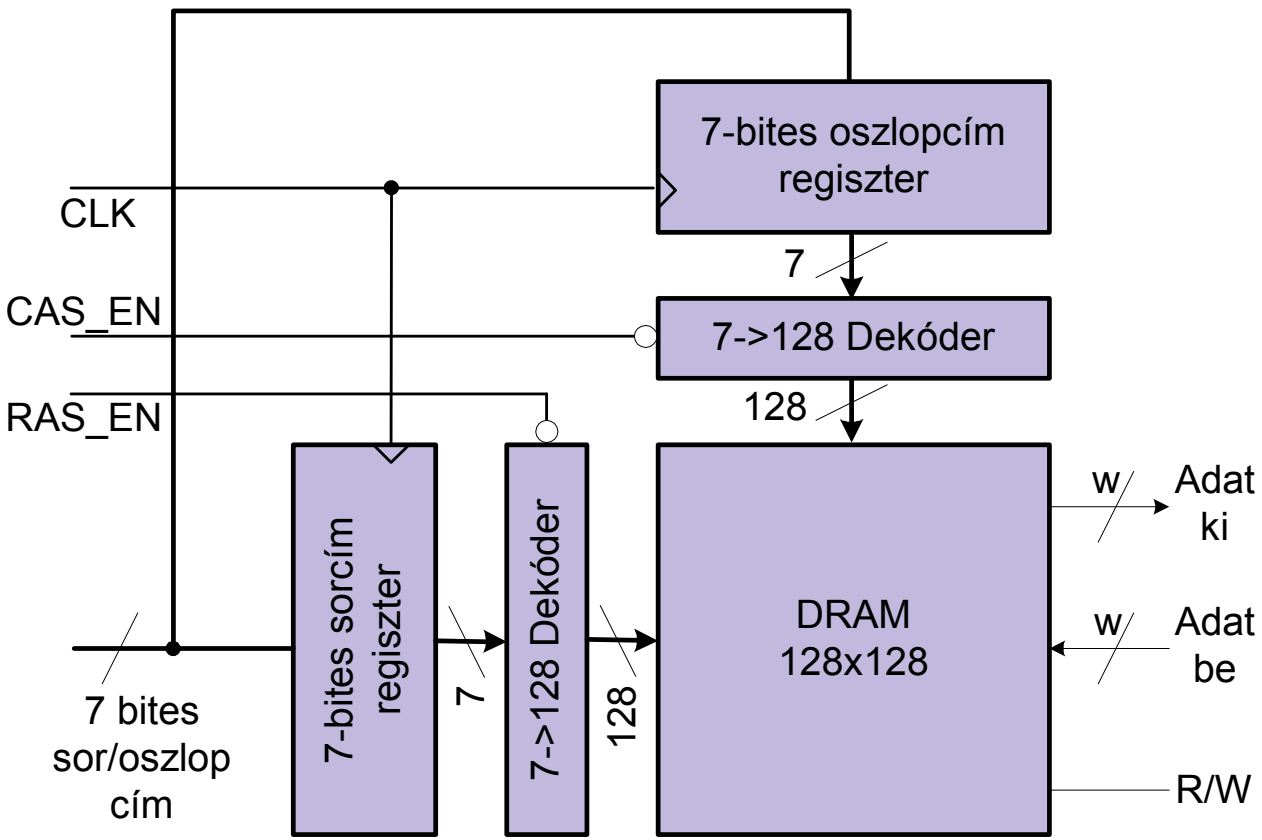
Címek: **sor-** és **oszlop-azonosítók** (RAS-CAS). A sorcím (Row), majd az oszlopcím (Col) megjelenik a címvonalon. Ezután válik elérhetővé a memória (setup time). A memória elérési idejétől (T_{acc}) függően kis idő múlva az adat érvényessé válik (Valid). A RAS felszabadul (T_{REL}) a kimeneten az adat visszatér (tri-state állapotba).

Példa: 128x128-as DRAM felépítése (2-D szervezésű)

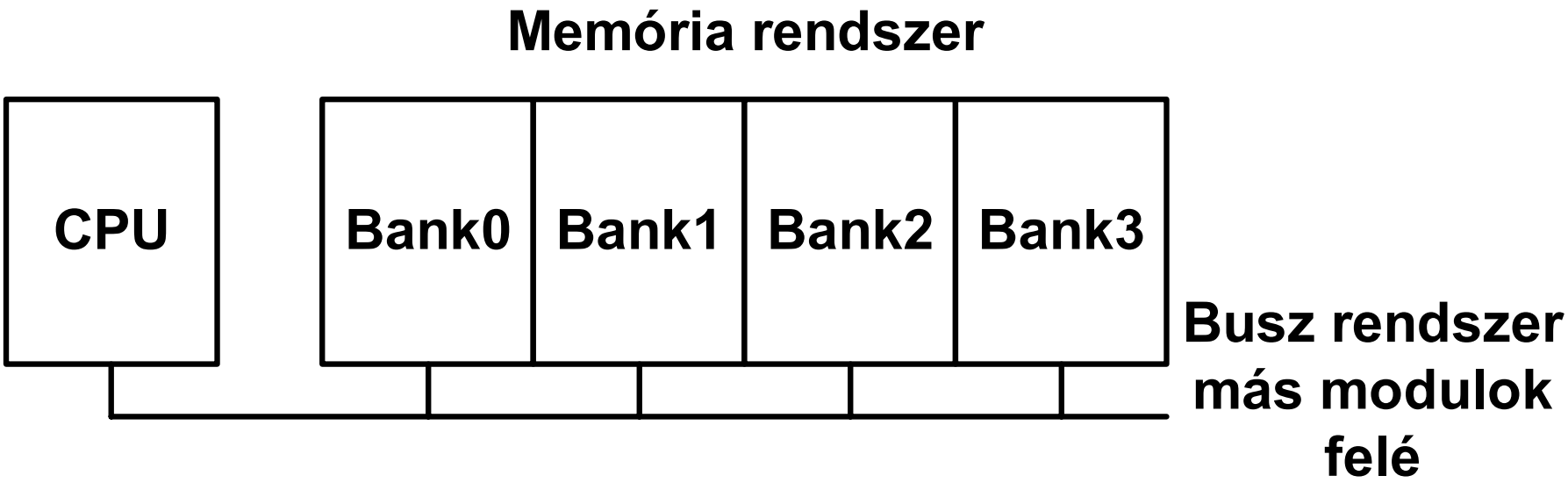
Feladat: Mekkora **lábszámot** kell biztosítani az DRAM működéséhez? Legyen w=16 bites

7+2*w+1(RAS_EN)+ 1(CAS_EN)+1(R/W) + 1(Vdd)+1(Gnd) = 44

Tehát összesen min. 44 lábra (pin) van szükség! (128*128 cella, 16bit/szó)

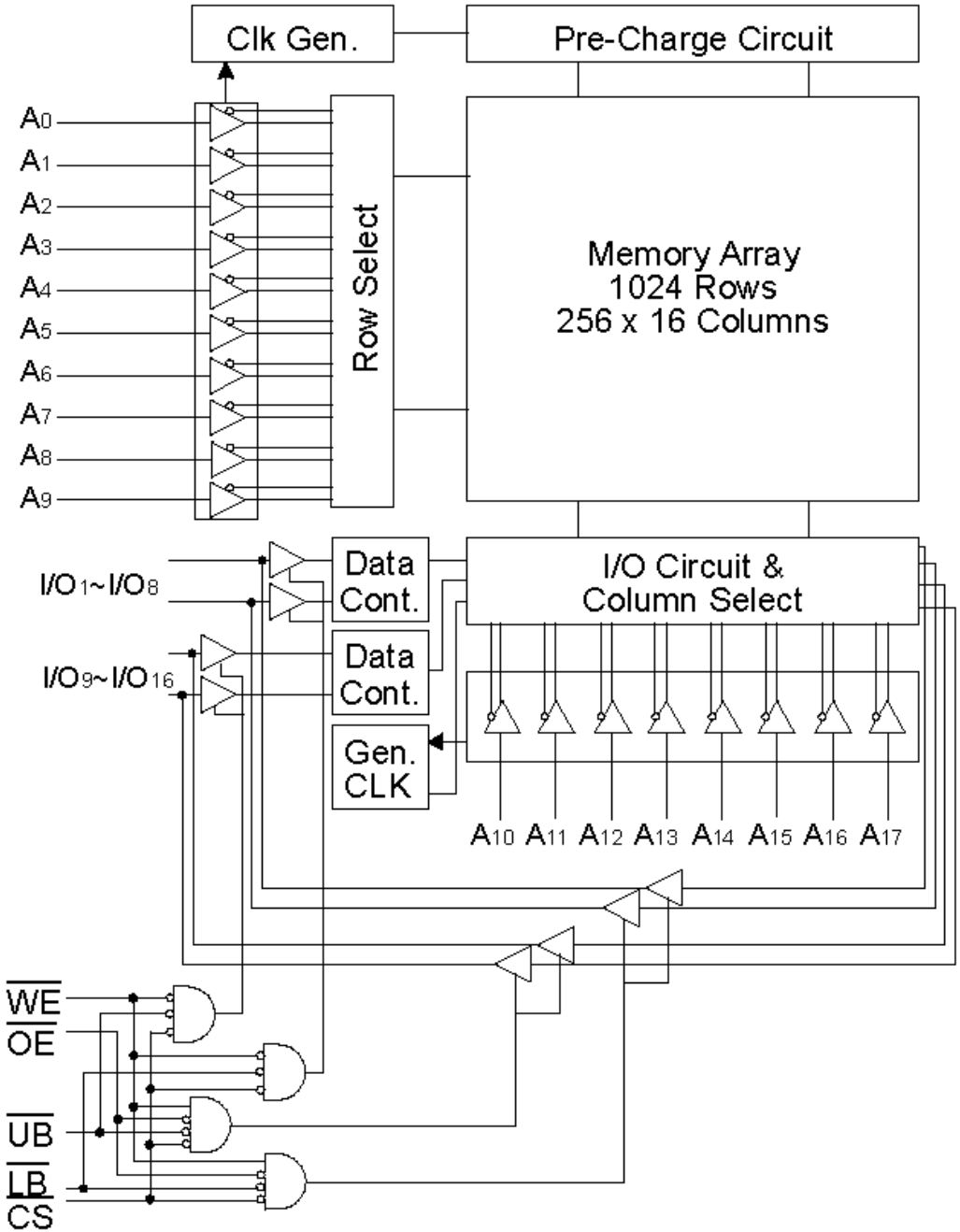


Memória „interleaving” technika



- Cél: busz sávszélesség növelése. Nagyméretű memóriák esetén sokszor gyorsabb a memóriabusz sebessége, mint amit a memória – ha egyben kezeljük – kiszolgálni képes
- Pl. Nagyméretű memóriamodult 4-bankos szervezéssel kell kialakítani. Memória igénylések egyszerre jutnak el a bankokhoz, míg a válasz – a kért adat – időszeletekben a CPU-felé.
- **Interleaving**: bankok közötti *időbeli* „átlapolódás”

Aszinkron SRAM blokkdiagramja



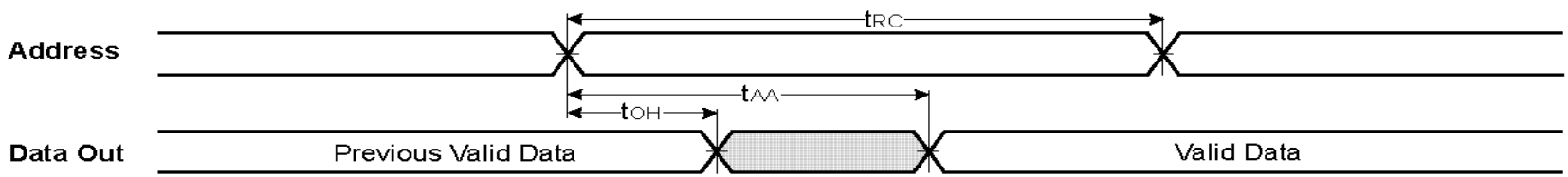
Ezekhez a részekhez a kapcsolódó segédanyagok:

Link: /additions/

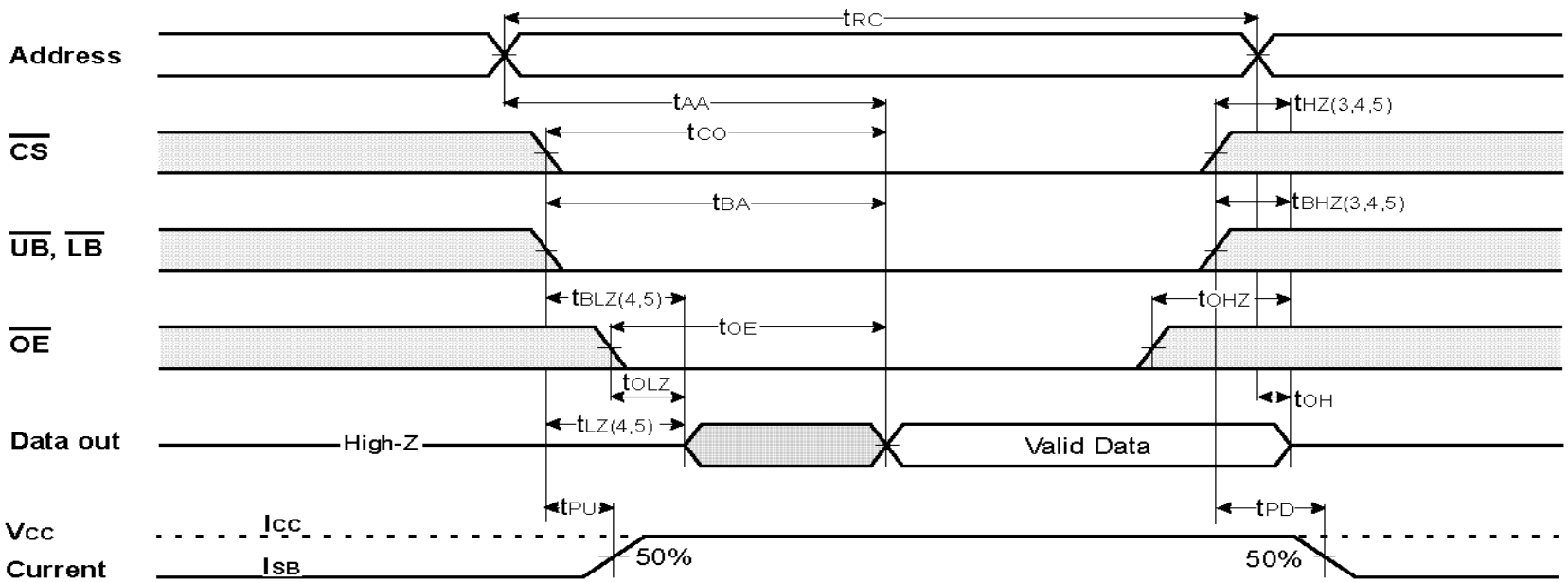
Aszinkron SRAM olvasási ciklusa

TIMING DIAGRAMS

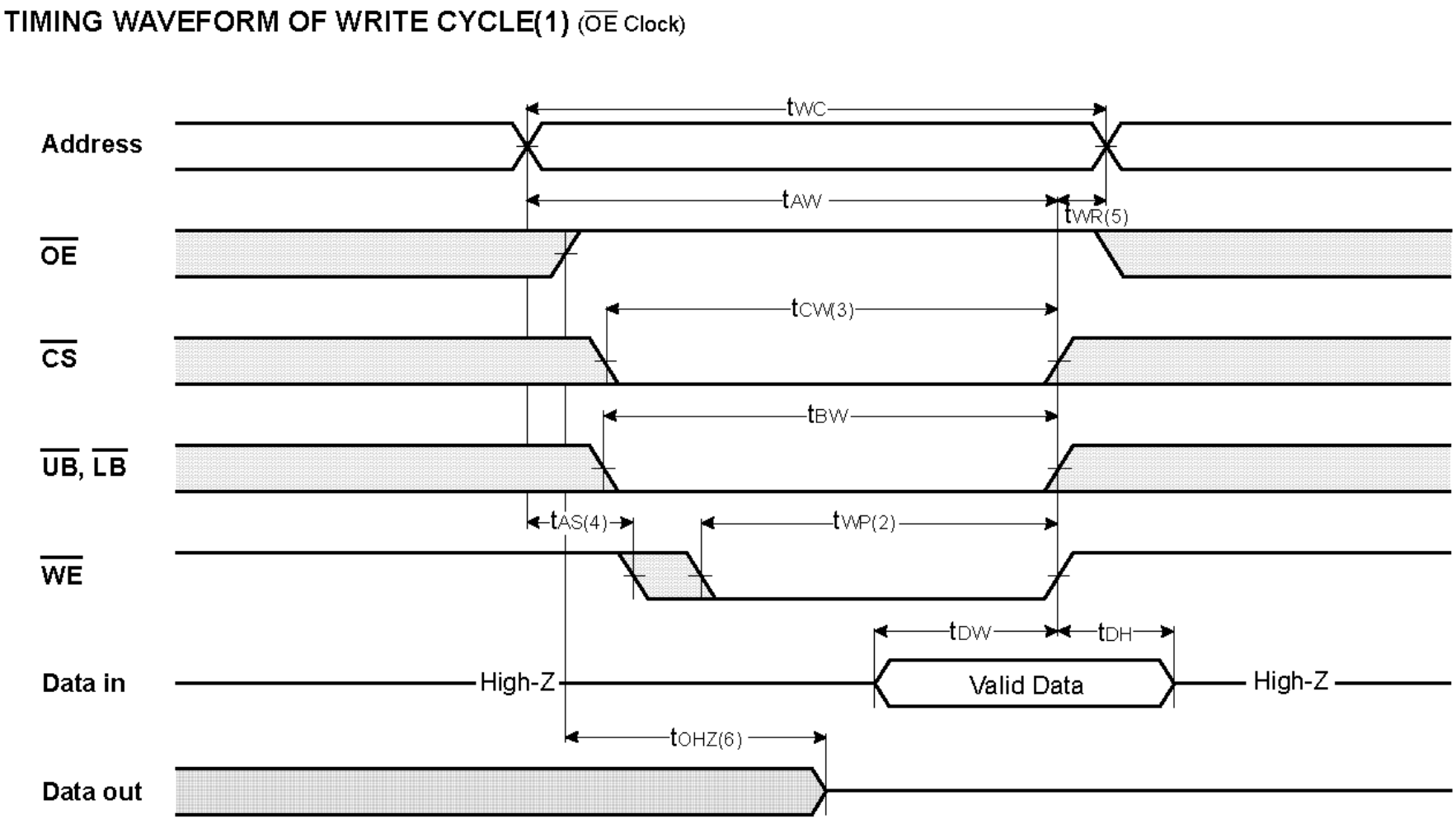
TIMING WAVEFORM OF READ CYCLE(1) (Address Controlled, $\overline{CS}$=$\overline{OE}$=V_{IL}, $\overline{WE}$=V_{IH}, $\overline{UB}$, $\overline{LB}$=V_{IL})



TIMING WAVEFORM OF READ CYCLE(2) ($\overline{WE}$=V_{IH})



Aszinkron SRAM írási ciklusa



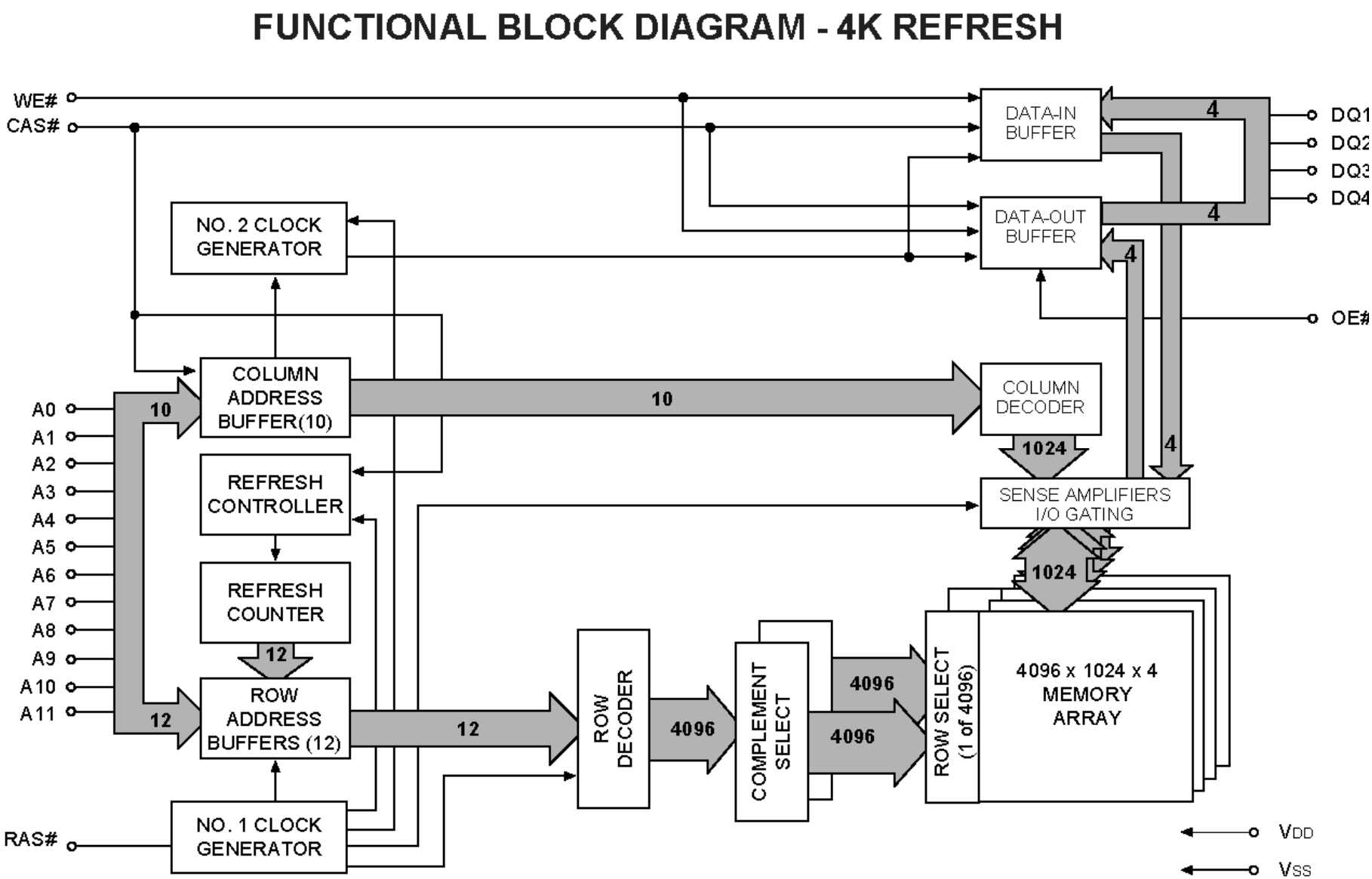
Aszinkron SRAM tipikus időzítési paraméterei

WRITE CYCLE*

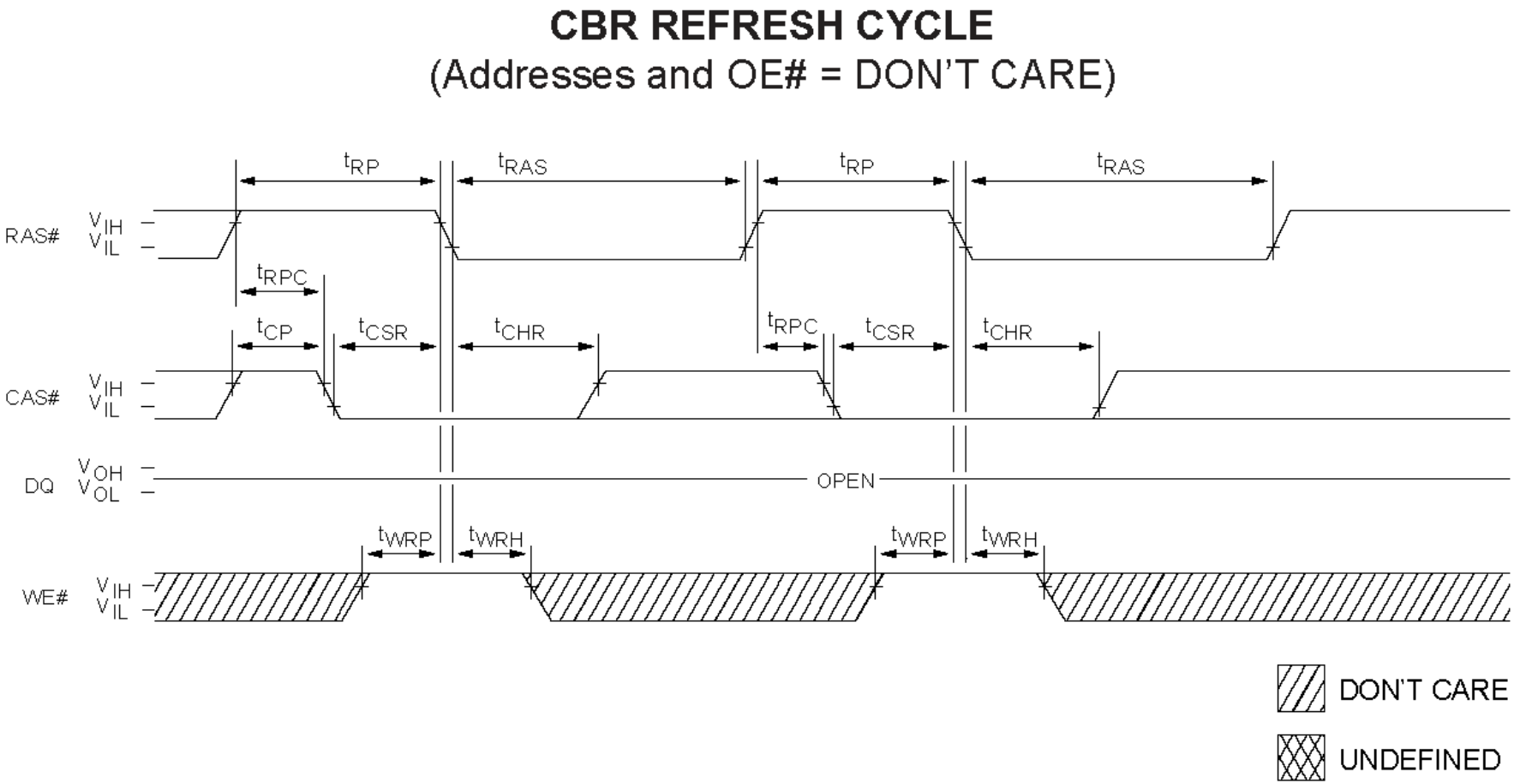
Parameter	Symbol	K6R4016V1D-08		K6R4016V1D-10		Unit
		Min	Max	Min	Max	
Write Cycle Time	t_{WC}	8	-	10	-	ns
Chip Select to End of Write	t_{CW}	6	-	7	-	ns
Address Set-up Time	t_{AS}	0	-	0	-	ns
Address Valid to End of Write	t_{AW}	6	-	7	-	ns
Write Pulse Width($\overline{OE}$ High)	t_{WP}	6	-	7	-	ns
Write Pulse Width($\overline{OE}$ Low)	t_{WP1}	8	-	10	-	ns
$\overline{UB}$, $\overline{LB}$ Valid to End of Write	t_{BW}	6	-	7	-	ns
Write Recovery Time	t_{WR}	0	-	0	-	ns
Write to Output High-Z	t_{WHZ}	0	4	0	5	ns
Data to Write Time Overlap	t_{DW}	4	-	5	-	ns
Data Hold from Write Time	t_{DH}	0	-	0	-	ns
End of Write to Output Low-Z	t_{OW}	3	-	3	-	ns

* The above parameters are also guaranteed at industrial temperature range.

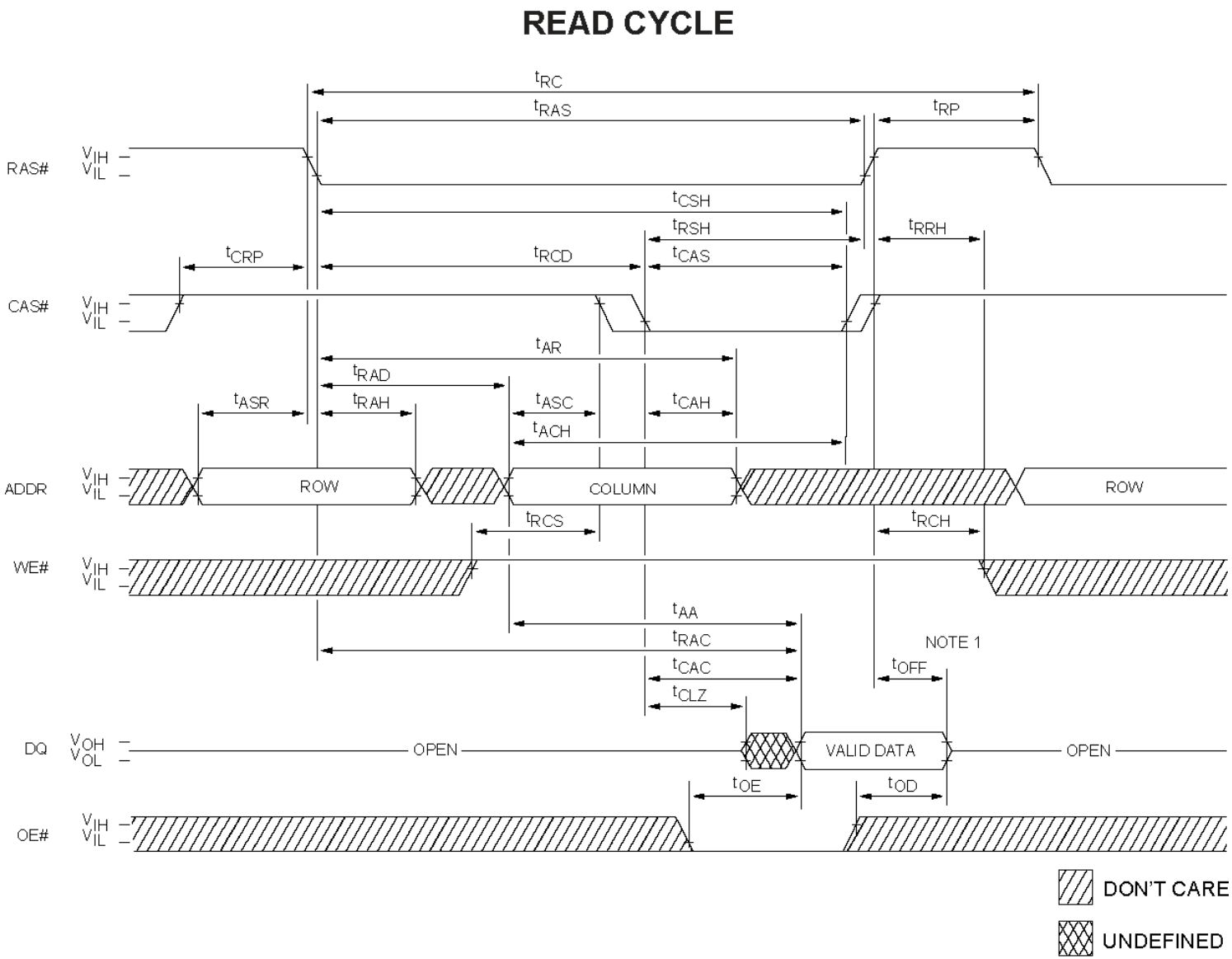
Aszinkron DRAM blokk diagramja



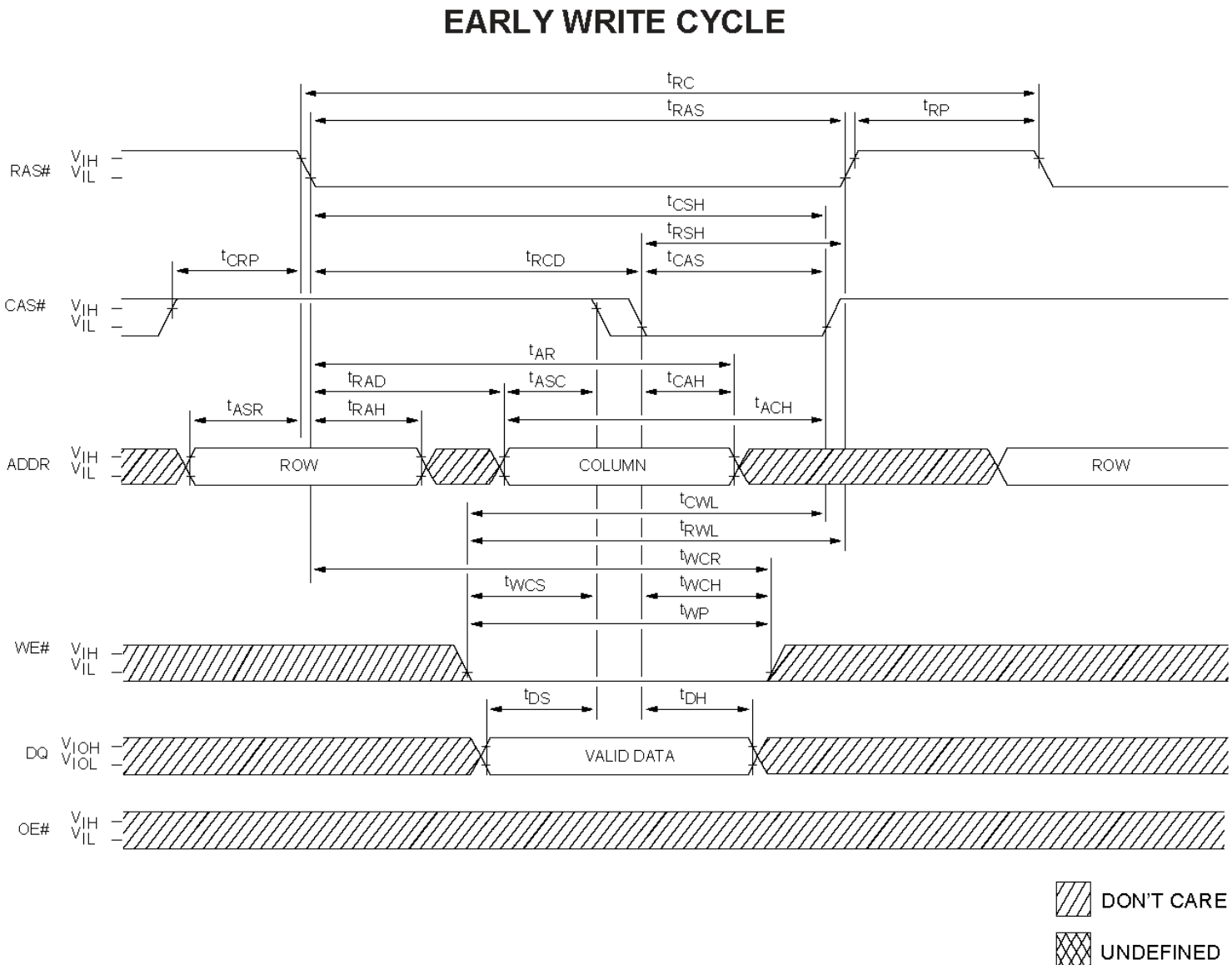
Aszinkron DRAM frissítési időd iagramja



Aszinkron DRAM olvasási ciklusa



Aszinkron DRAM írási ciklusa

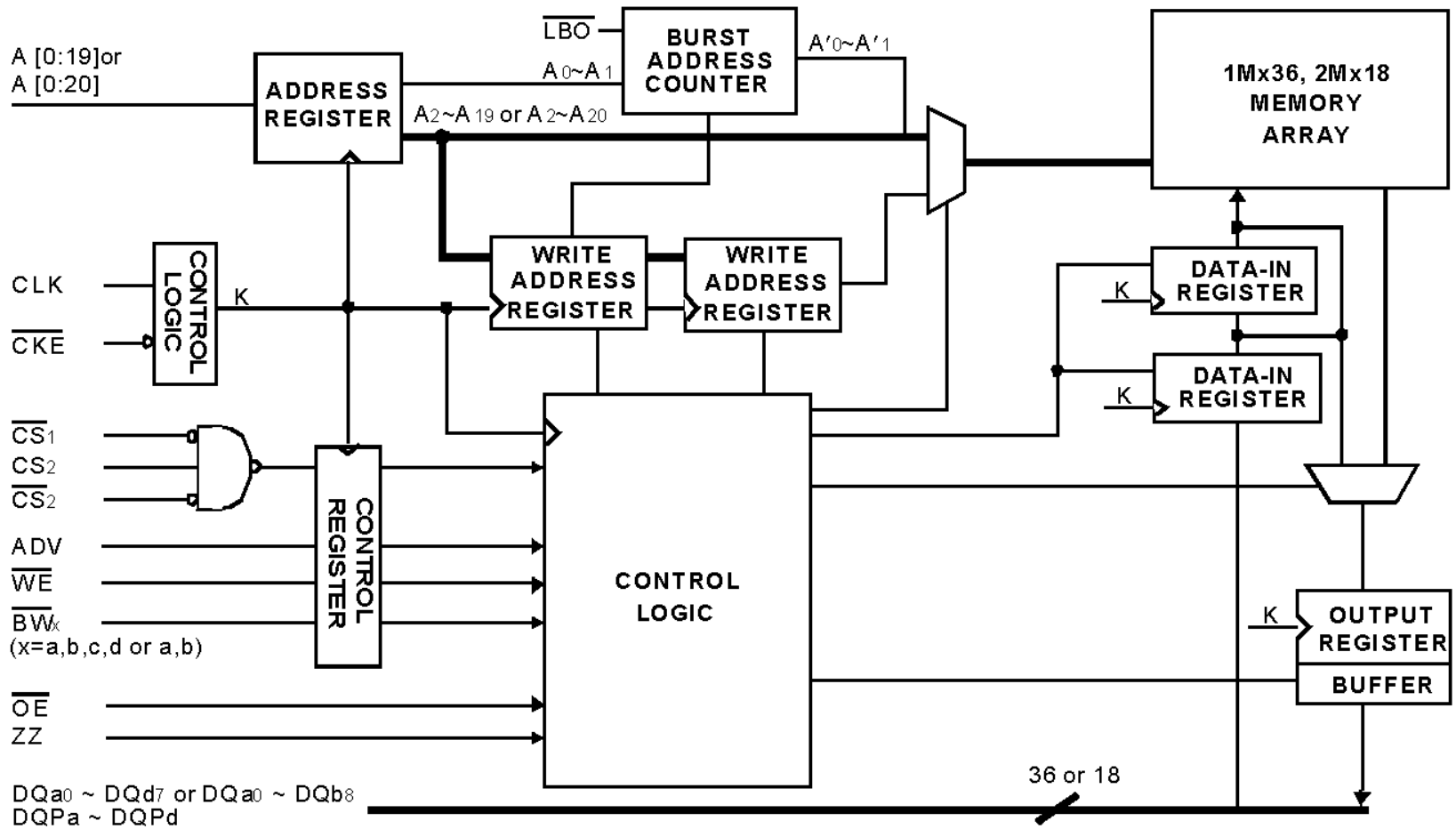


Szinkron memória típusok:

- Szinkron SRAM (ZBT, NtRAM SSRAM)
- Szinkron DRAM (DDR, DDR2 SDRAM)

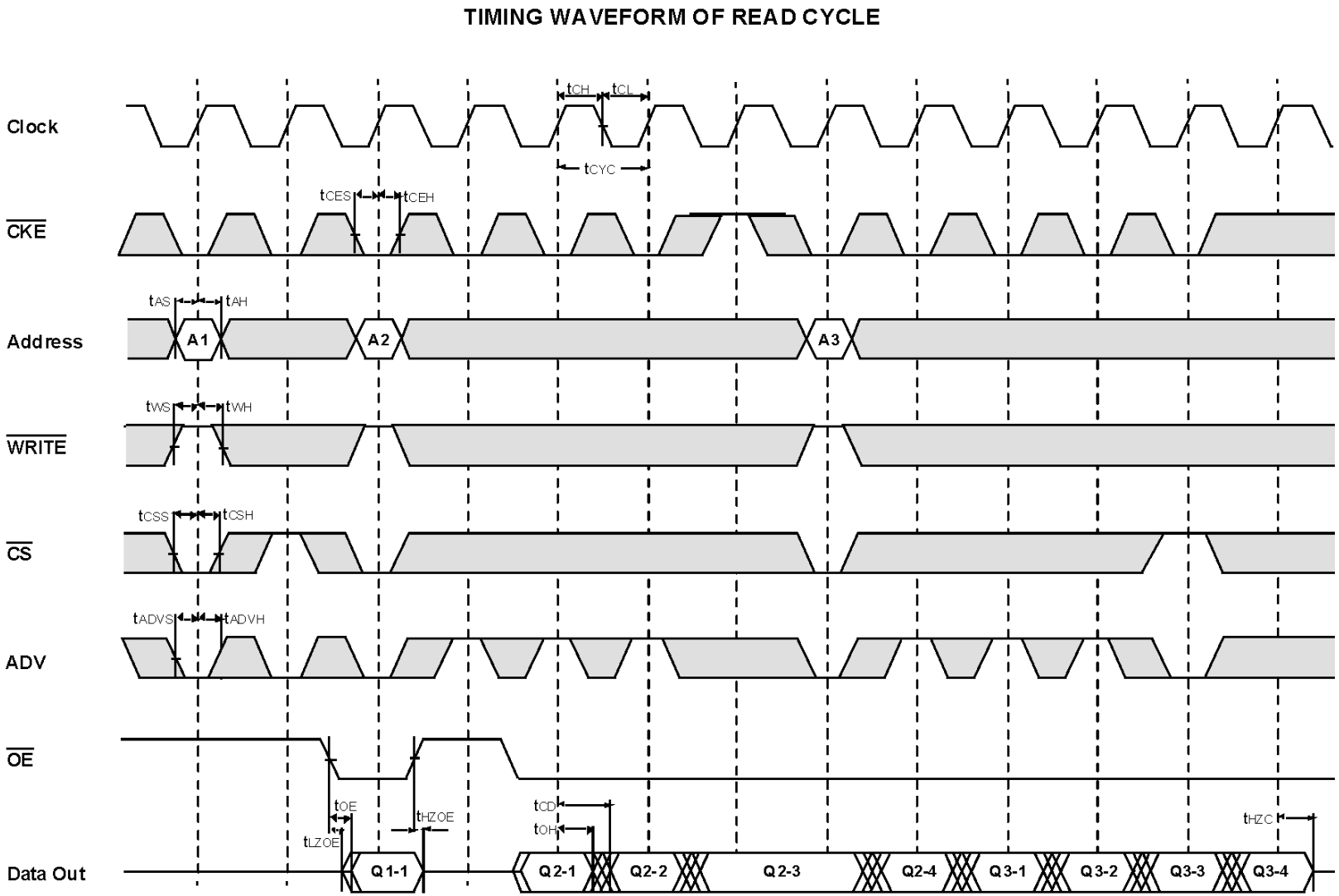
ZBT (NtRAM) SSRAM blokk diagramja

- /additions/ds_k7n323645m.pdf

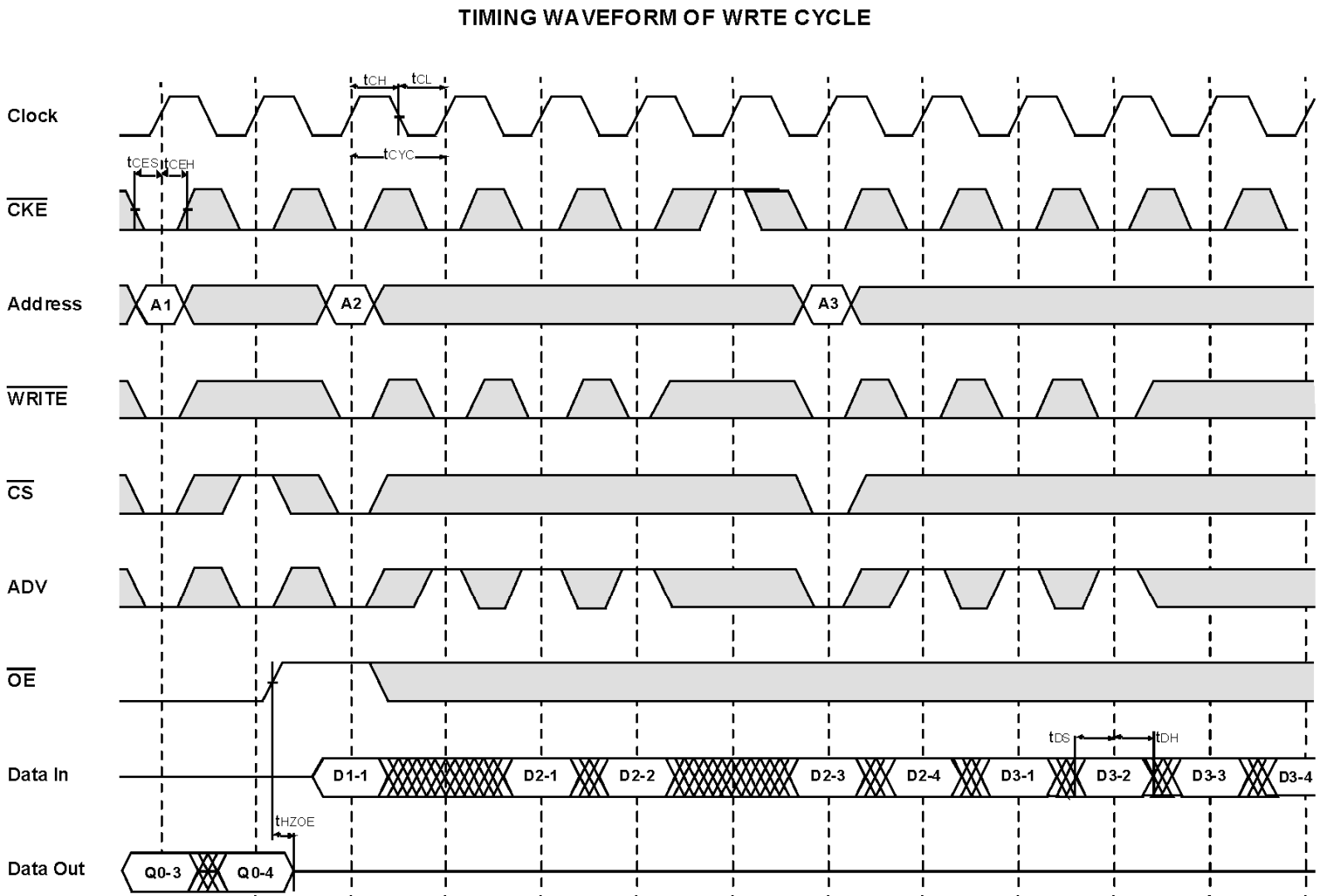


- ZBT: Zero Bus Turnaround = No turnaround

ZBT SSRAM olvasási ciklus



ZBT SSRAM írási ciklus



DDR SDRAM blokk diagramja

- /additions/512MBDDRx4x8**x16**.pdf

- ∑512M = 32M x 16

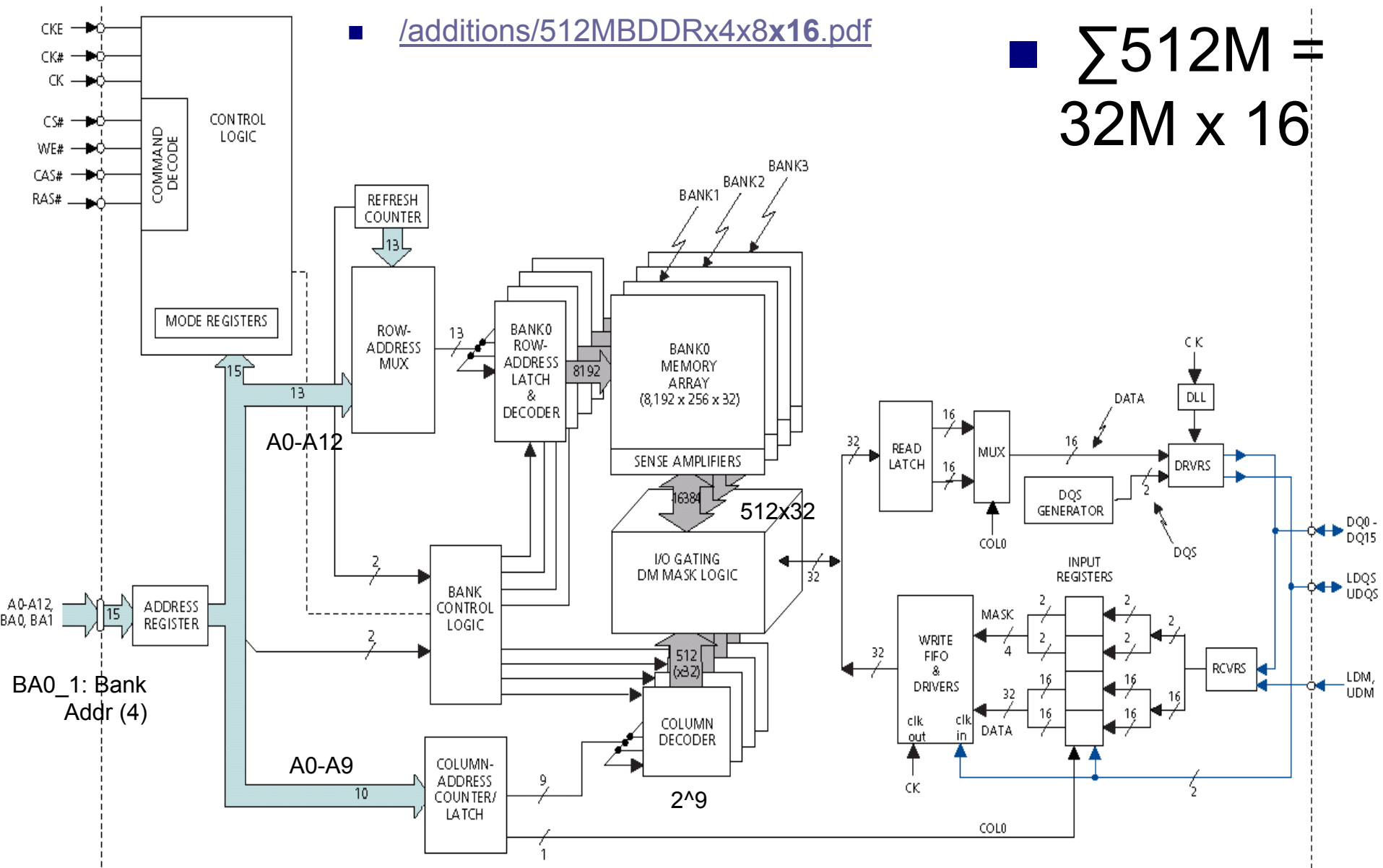
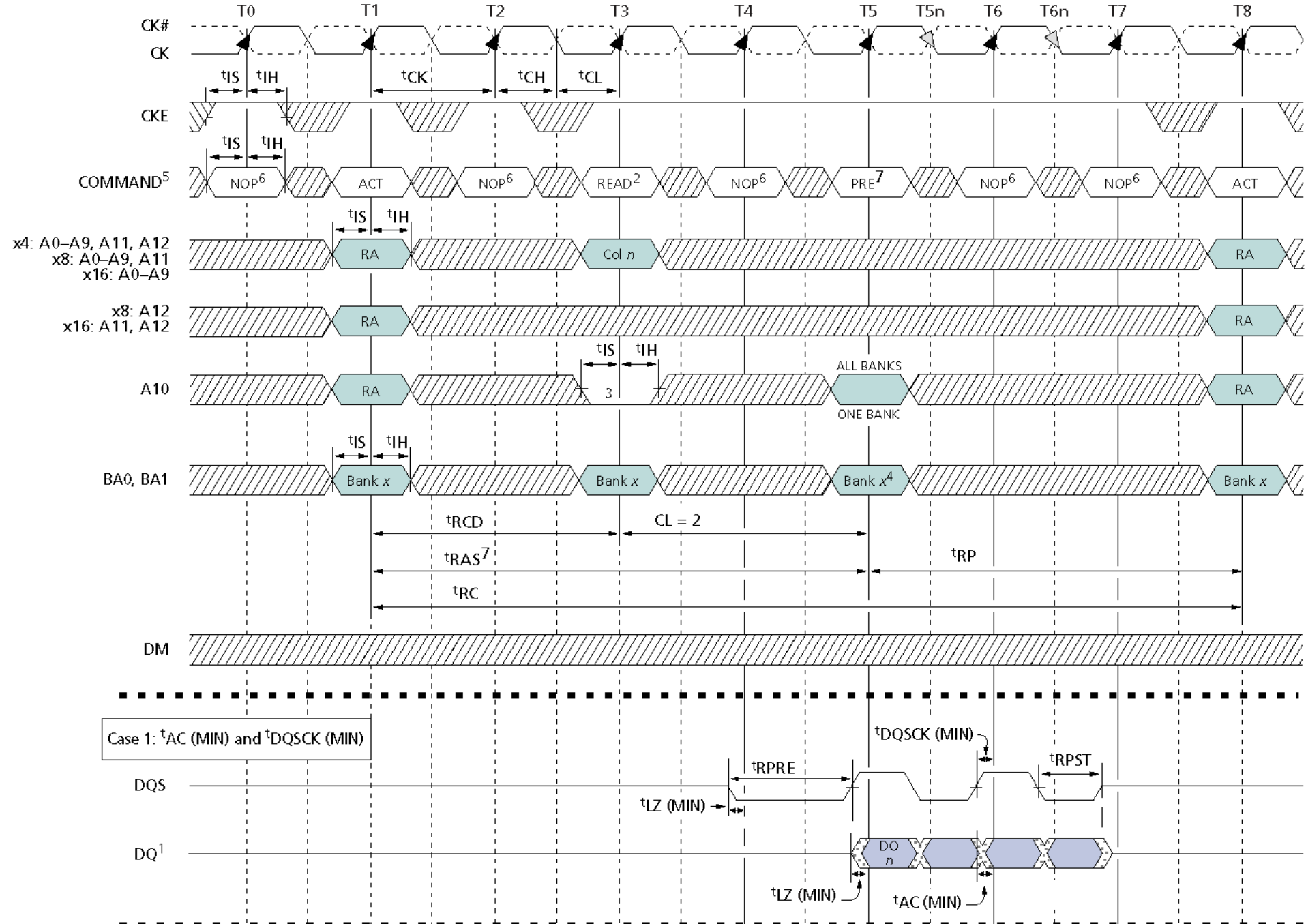
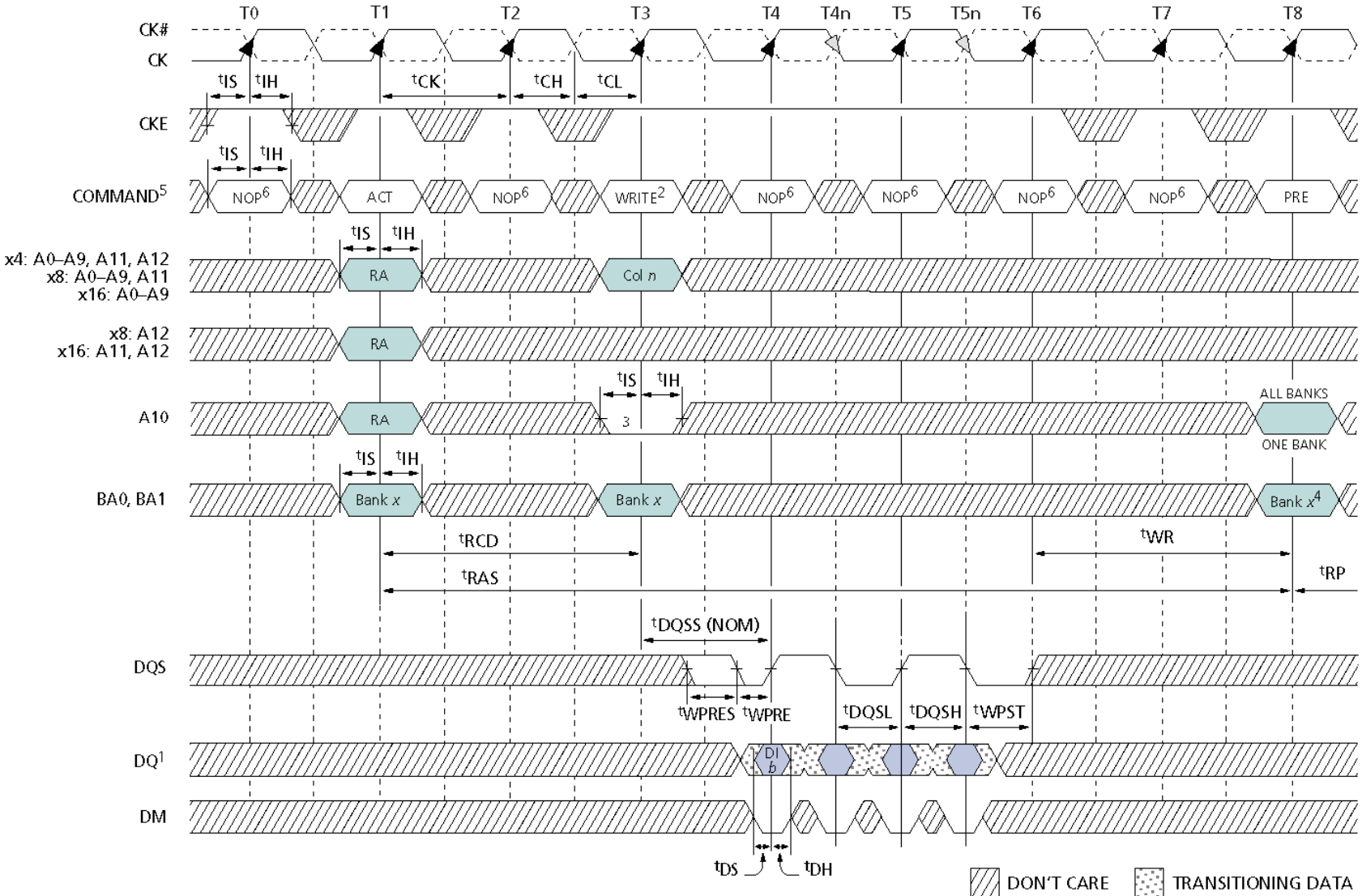


Figure 49: Bank Read - Without Auto Precharge



DDR SDRAM írási ciklus

Figure 51: Bank Write - Without Auto Precharge



DDR-SDRAM

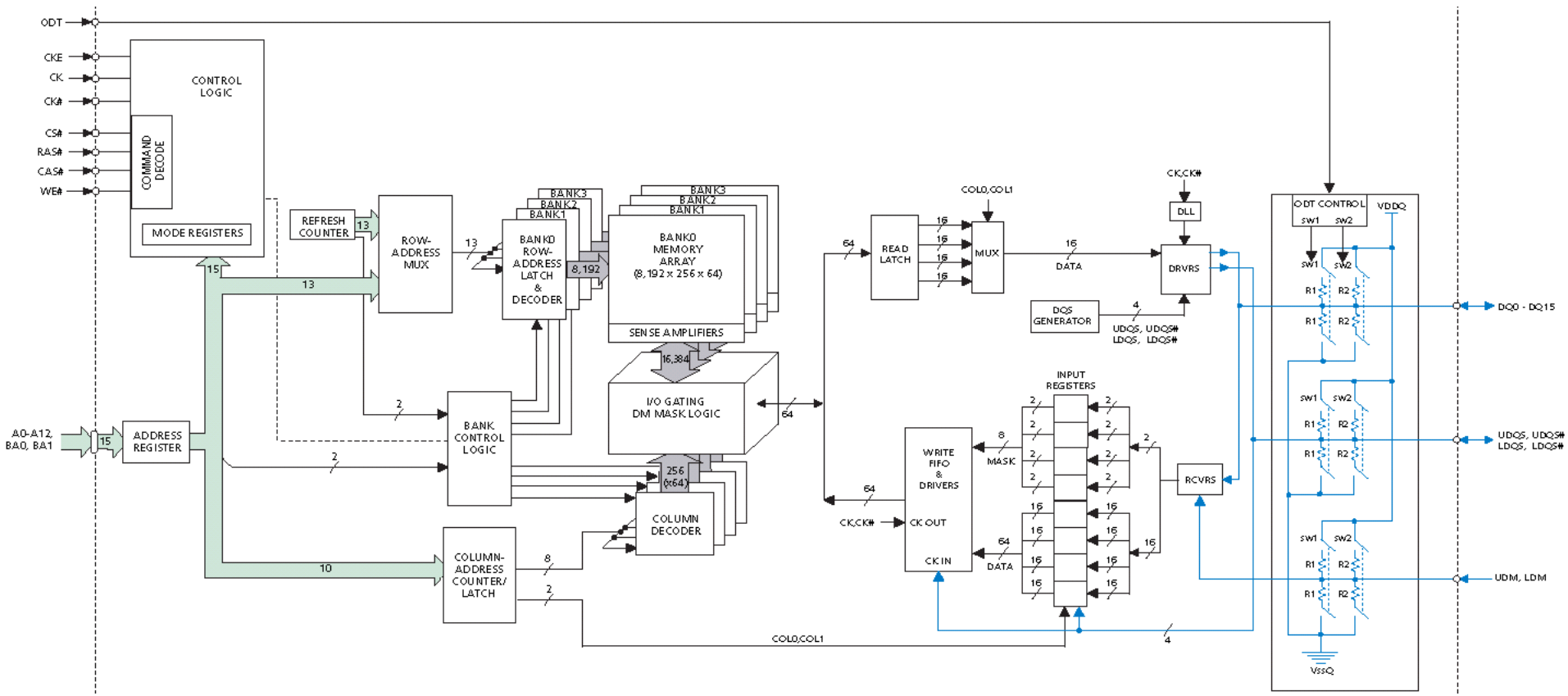
Standard name	Memory clock	Cycle time	I/O Bus clock	Data transfers per second	Module name	Peak transfer rate
DDR-200	100 MHz	10 ns	100 MHz	200 Million	PC-1600	1600 MB/s
DDR-266	133 MHz	7.5 ns	133 MHz	266 Million	PC-2100	2100 MB/s
DDR-300	150 MHz	6.67 ns	150 MHz	300 Million	PC-2400	2400 MB/s
DDR-333	166 MHz	6 ns	166 MHz	333 Million	PC-2700	2700 MB/s
DDR-400	200 MHz	5 ns	200 MHz	400 Million	PC-3200	3200 MB/s

With 64-bit data at a time:

Peak Transfer Rate = transfer rate of (memory bus clock rate) × 2 (for dual rate) × 64 (number of bits transferred) / 8 (number of bits/byte)

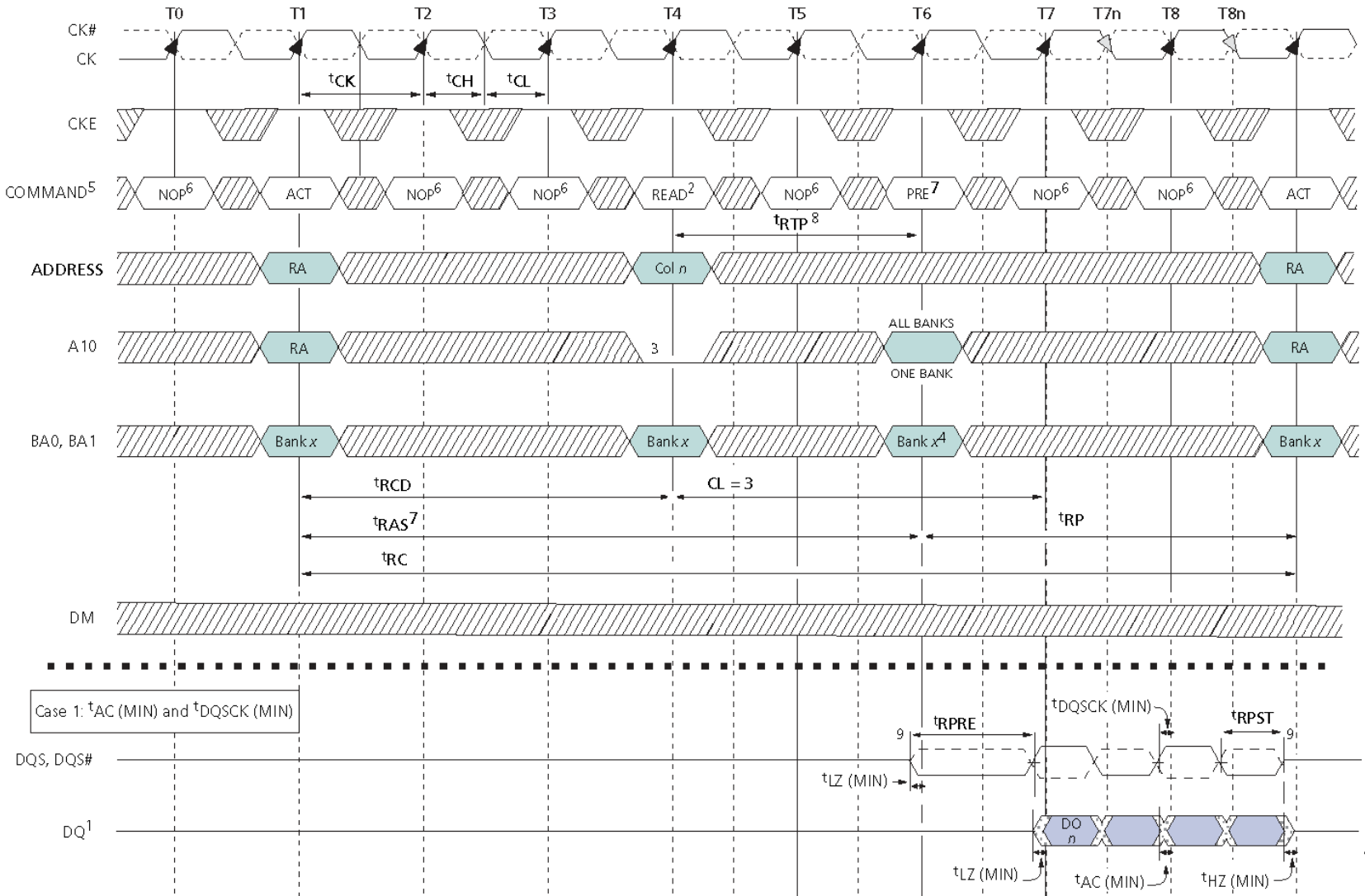
DDR-II SDRAM blokk diagram

- /additions/512MbDDR2.pdf



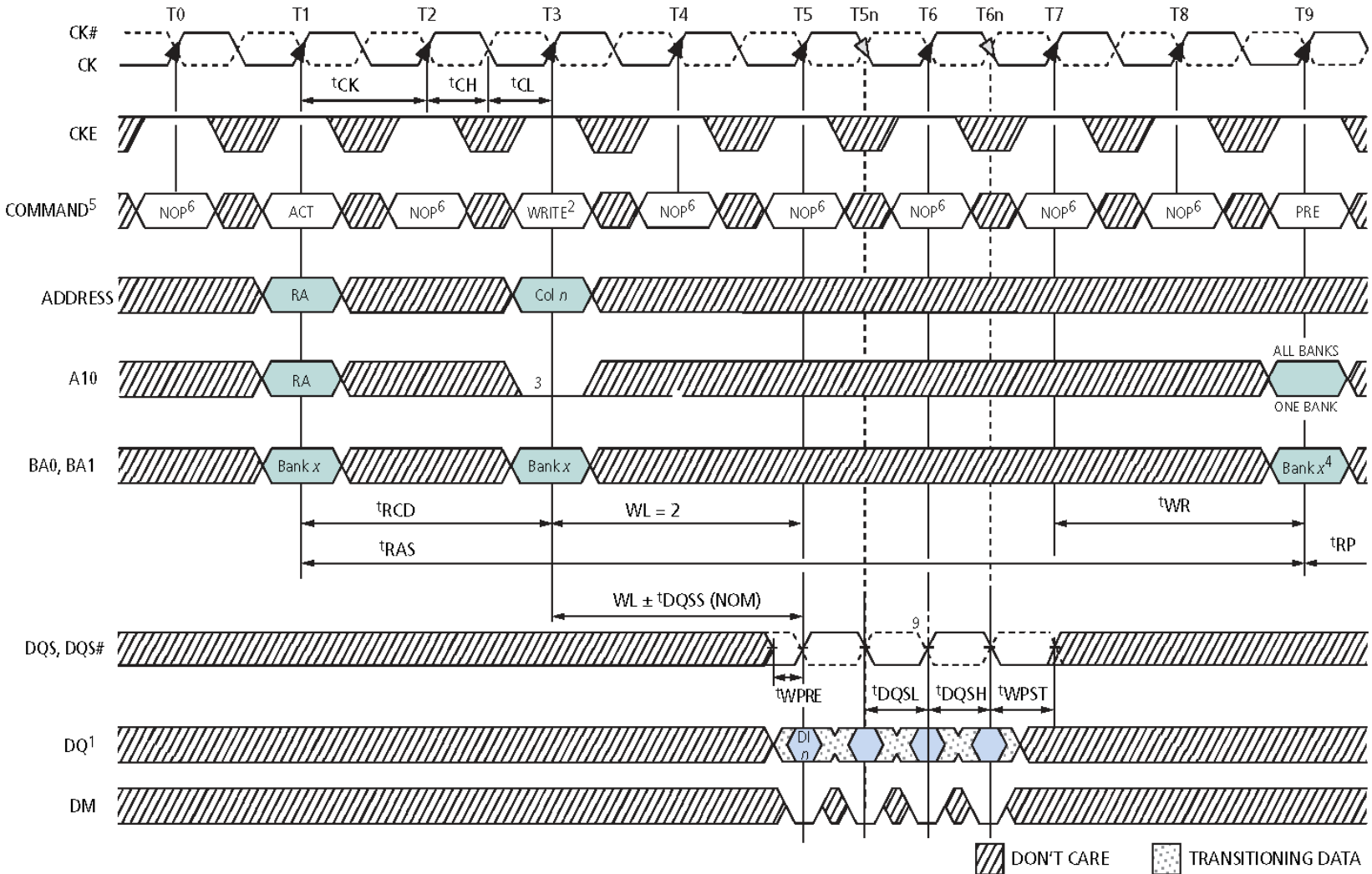
DDR-II SDRAM olvasási ciklus

Figure 28: Bank Read – Without Auto Precharge



DDR-II SDRAM írási ciklus

Figure 41: Bank Write–Without Auto Precharge



DDR2-SDRAM

Standard name	Memory clock	Cycle time	I/O Bus clock	Data transfers per second	Module name	Peak transfer rate
DDR2-400	100 MHz	10 ns	200 MHz	400 Million	PC2-3200	3200 MB/s
DDR2-533	133 MHz	7.5 ns	266 MHz	533 Million	PC2-4200	4266 MB/s
DDR2-667	166 MHz	6 ns	333 MHz	667 Million	PC2-5300	5333 MB/s
DDR2-800	200 MHz	5 ns	400 MHz	800 Million	PC2-6400	6400 MB/s
DDR2-1066	266 MHz	3.75 ns	533 MHz	1066 Million	PC2-8500	8533 MB/s

With 64-bit data at a time:

Peak Transfer Rate = transfer rate of (memory bus clock rate) × 2 (for dual rate) × 64 (number of bits transferred) / 8 (number of bits/byte)

SDRAM-ok összehasonlító táblázata

		SD	DDR	DDR-II
I/O tápfeszültég	V_{DD}	3.3	2.5	1.8
CAS várakozási idő (latency)	CL	1-3	2-3	3-5
Órajel ciklus ideje (Clock cycle time)	t_{CK}	6ns	5ns	3ns
ACTIVE to PRECHARGE parancs	t_{RAS}	42ns	40ns	40ns
ACTIVE to READ or WRITE késleltetés	t_{RCD}	18ns	15ns	12ns
ACTIVE to ACTIVE/AUTO REFRESH parancs peridódus	t_{RC}	60ns	55ns	54ns
ACTIVE Bank(a) to ACTIVE Bank(b) parancs	t_{RRD}	12ns	10ns	7.5ns
Írás utáni “visszaállási idő”	t_{WR}	12ns	15ns	15ns
PRECHARGE parancs periódus	t_{RP}	18ns	15ns	12ns
REFRESH to REFRESH parancs intervallum	t_{REFC}	64ms	73us	105ns-70us
Átlagos, periódikus frissítés ideje	t_{REFI}	15us	7.8us	7.8us

Virtuális memória kezelés

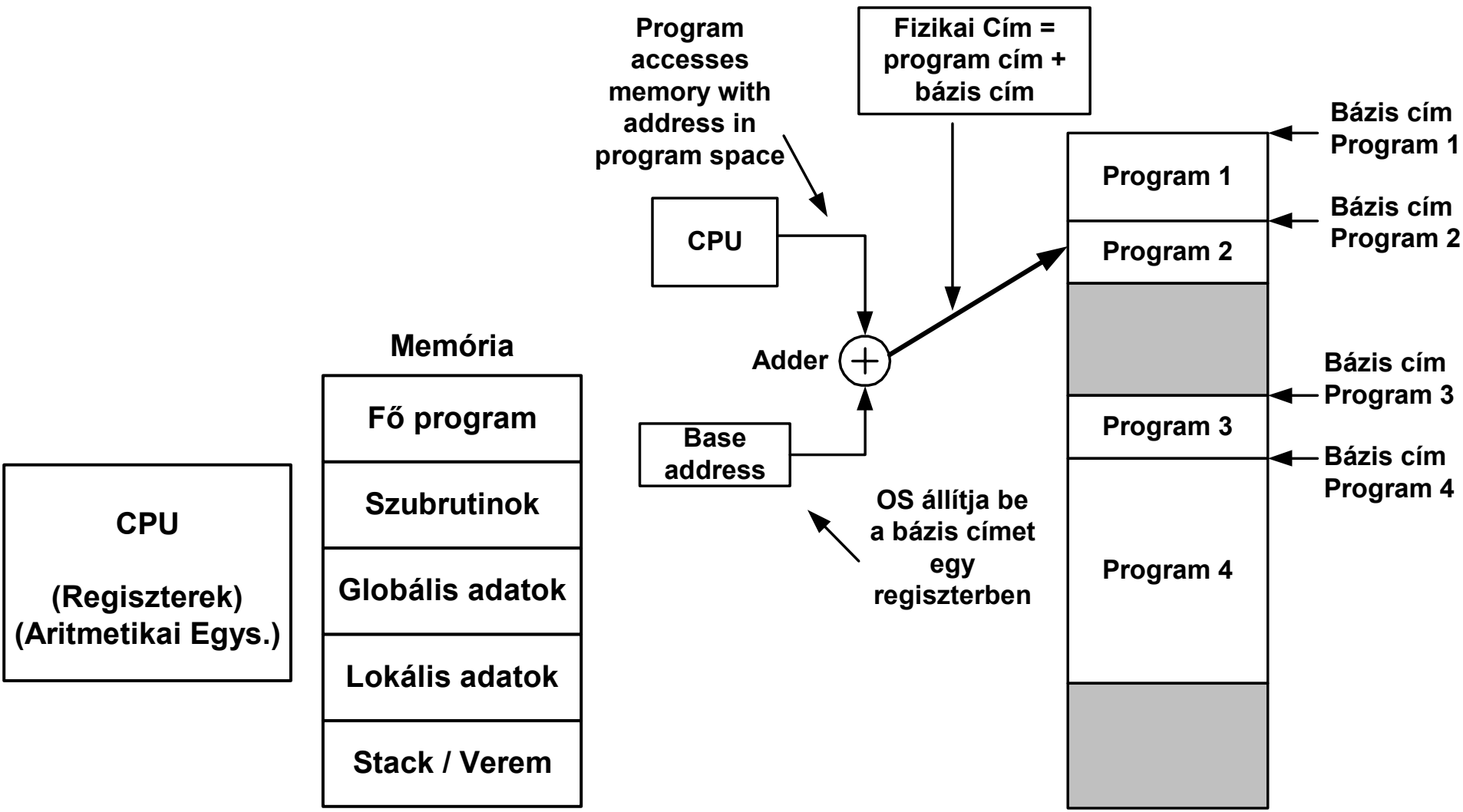
Virtuális memória kezelés

- Miért van szükség rá?
  - Ha a rendelkezésre álló fizikai memóriaterület kevesebb, mint amit a program(ok) igényelnek, akkor virtuális memóriára van szükségünk.
- Virtuális memóriakezelés: kapcsolatot teremt a rendelkezésre álló fizikai, ill. a program által igényelt logikai memória megfelelő szervezése között.
- Feladat: a program által igényelt *virtuális* cím $\Rightarrow$ *valós* memóriacímre konvertálása. Ezt nevezzük **címfordítás**-nak.

Virtuális memóriakezelés #2

- Az OS elrejti a program (felhasználó számára is) a fizikai memóriakorlátot, azzal hogy virtuális memóriát használ.
- „Multi-programming”: egyszerre több program futhat, minden programnak saját logikai/bázis címtartománya van, amellyel hivatkozni lehet rá.
- **Fizikai cím = program memóriacíme + program báziscíme**
- **Virtuális memóriakezelés két fő típusa:**
  - **1.) szegmentálás**
  - **2.) lapozás**

Virtuális Memória Rendszer



Program részei: kód / adat (lapokkal)

multiprogramozás

1.) **Szegmentálás** mechanizmusa: virtuális memória címzés

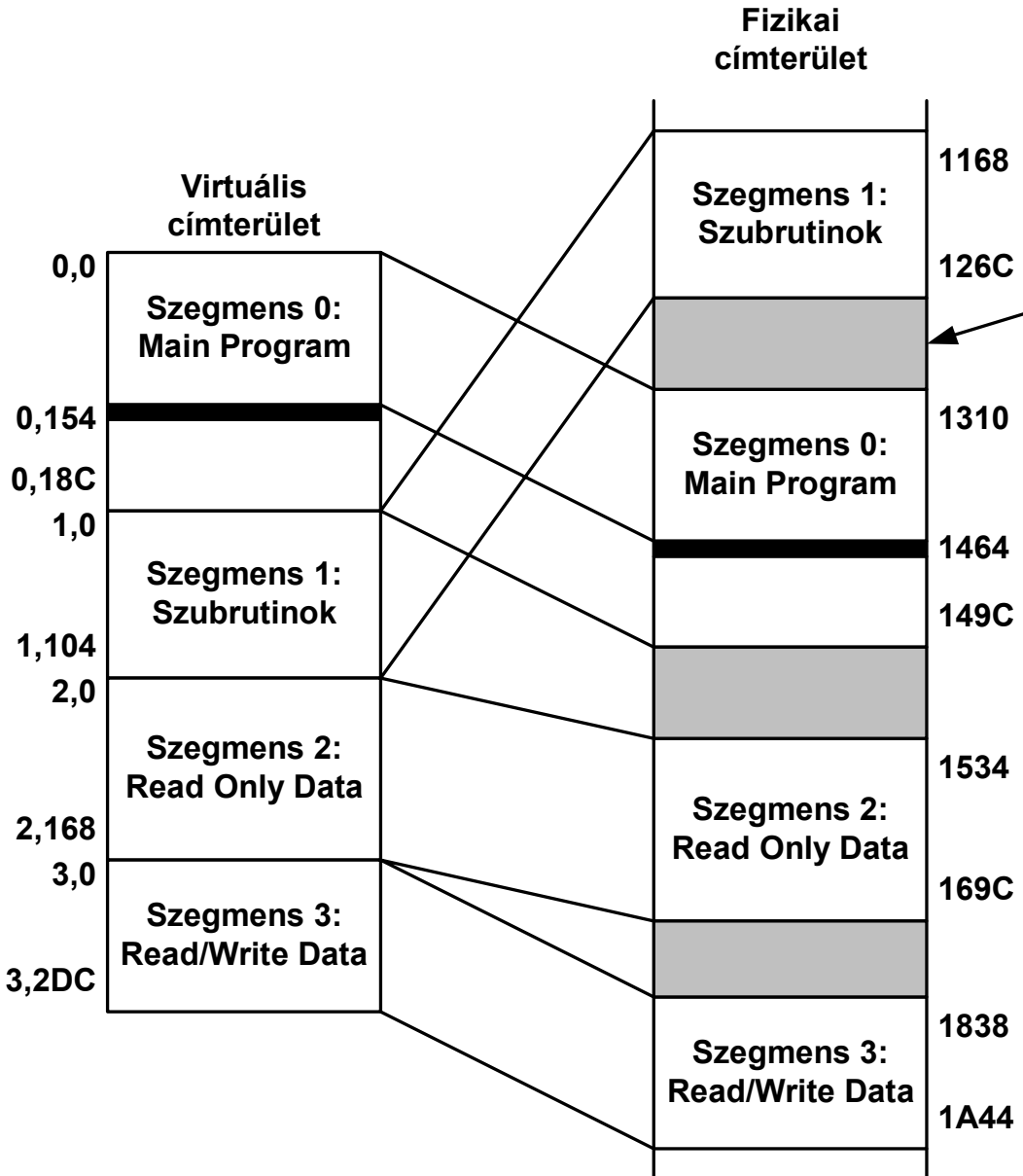
- **Szegmens**: a program és a memória kevés számú, nagyméretű, és változó méretű logikai egységekre van feldarabolva.
- A program négy fő szegmense funkciójuk szerint:
  - 0. szegmens: főprogramok (olvasható/végrehajtható),
  - 1. szegmens: szubrutinok,
  - 2. szegmens: csak olvasható adatok,
  - 3. szegmens: olvasható / írható adatok.
- Az egyes szegmensek között *átlapolódás* lehetséges!
- Hátránya: ha sokszor cserélgetjük a szegmenseket, akkor a változó blokkmérete miatt üres helyek, hézagok keletkeznek, ami a memória rossz kihasználásához vezet, mivel a kisméretű üres területekre már nem lehet behívni újabb szegmenset. Ezt nevezzük *külső tördelődésnek*.

Címfordítás szegmentálás esetén

- **FIZIKAI CÍM= Szegmens bázis címe + Szegmensen belüli eltolás (offset) címe**
- A „+” itt ténylegesen *összeadást* jelent! HW-es megvalósítás kell az összeadó miatt, és kell egy összehasonlítás, azért hogy nem címeztünk-e ki a szegmens címtartományán kívülre! Hiba esetén megszakítást (interrupt) küld.
- Szegmenstábla: A szegmenstáblából olvassuk ki a szegmens számát (hexa érték) követően a hozzá tartozó szegmens hosszát, szegmens címet (hexa), továbbá az egyes szegmensek elérési információit: R/W/X: olvasható /írható /végrehajtható.
- **Példa**: Logikai cím a virtuális memóriában: 0,154 = 0 szegmens (főprogram) száma, és a szegmens 154 offszetcíme a virtuális memóriában. Legyen a 0.szegmens hexa címe 1310. Ezt kell „összeadni” a hexa 154-el.

1 3 1 0 + 1 5 4 = 1 4 6 4 (Fizikai Cím)

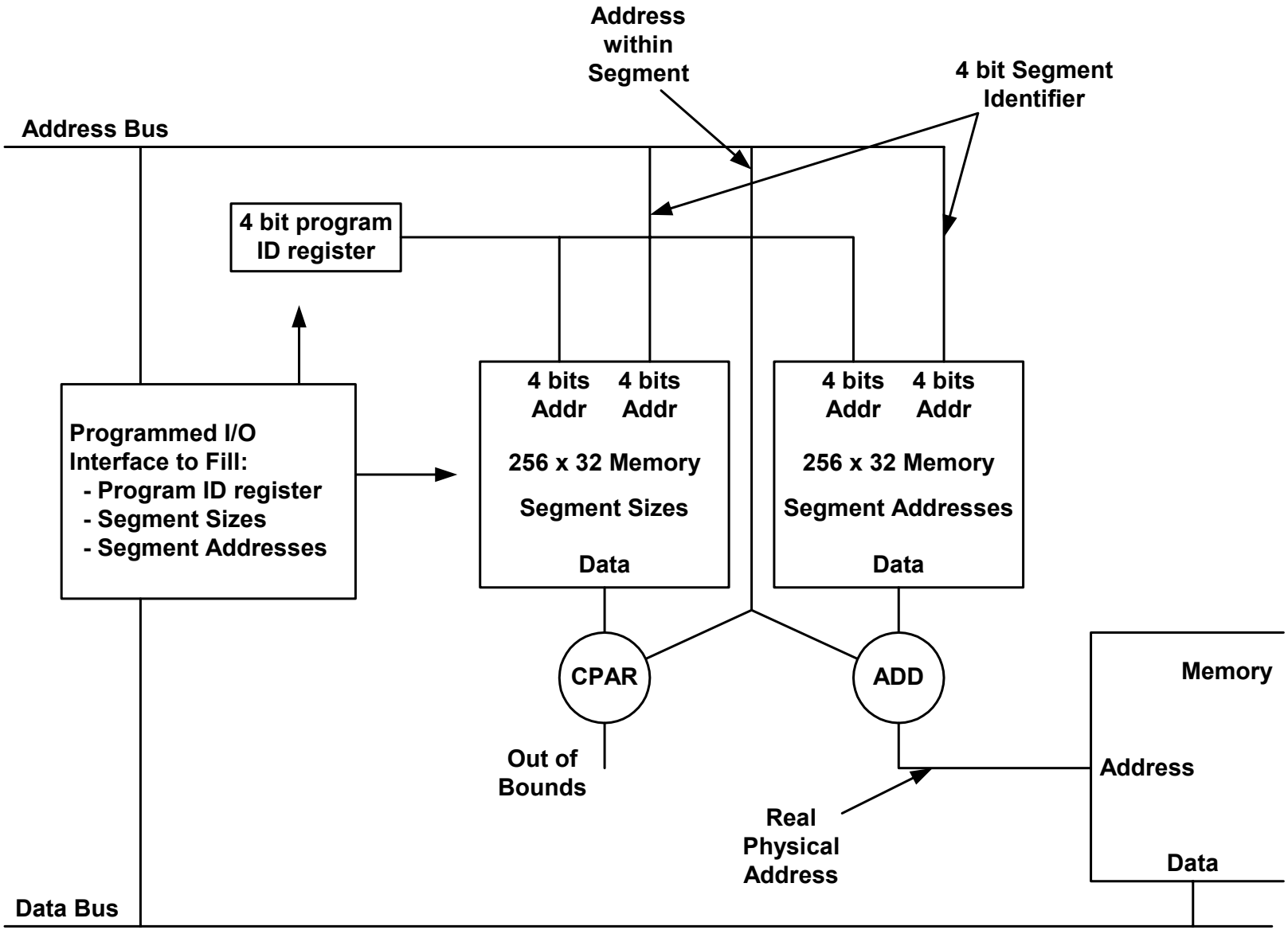
Példa: Szegmentálás mechanizmusa



Program Szegmens Tábla

Segment	Length	Address	Access
0	190	1310	Read, Execute
1	108	1168	Read, Execute
2	16C	1534	Read
3	210	1838	Read, Write

Szegmentálás címfordító áramköre



Szegmentálás címfordító áramkör

- Az áramkör **16 programot** tud kezelni, amelyek egyenként ***16 szegmensből*** állnak (16x16=256 location). Az áramkör a címgeneráló és a memória között helyezkedik el. **32 bites** címvonalat használ. A szegmenscímet párban generálja: áll egyrészt egy szegmensszámból és egy szegmensen belüli eltolásból (offset).
- Program futása előtt a CPU betölti a megfelelő szegmenset a memóriába, majd beállítja a *szegmens hosszát és címét* a két **256x32-**es memóriába (az egyik a szegmens méretét, a másik a szegmens címét tárolja).
- Ezután az OS beállítja a megfelelő mintát a ProgramID regiszterben, és inicializálja/elindítja a programot. A ProgramID egy 4 bites regiszter - a 16 különböző program egyidejű tárolásakor - bizonyos minták alapján kapcsol a programok között.
- A memóriában a szegmenset a „báziscímének” és a „szegmensen belüli eltolás címének” *összeadásával* kapjuk meg, mindig összehasonlítjuk a szegmensen belüli címet a maximális címmel. Ha túl nagy címet kapunk, akkor egy „out of bounds” (határon kívül vagyunk) megszakítást küld.

2.) **Lapozás** mechanizmusa: virtuális memória címzés

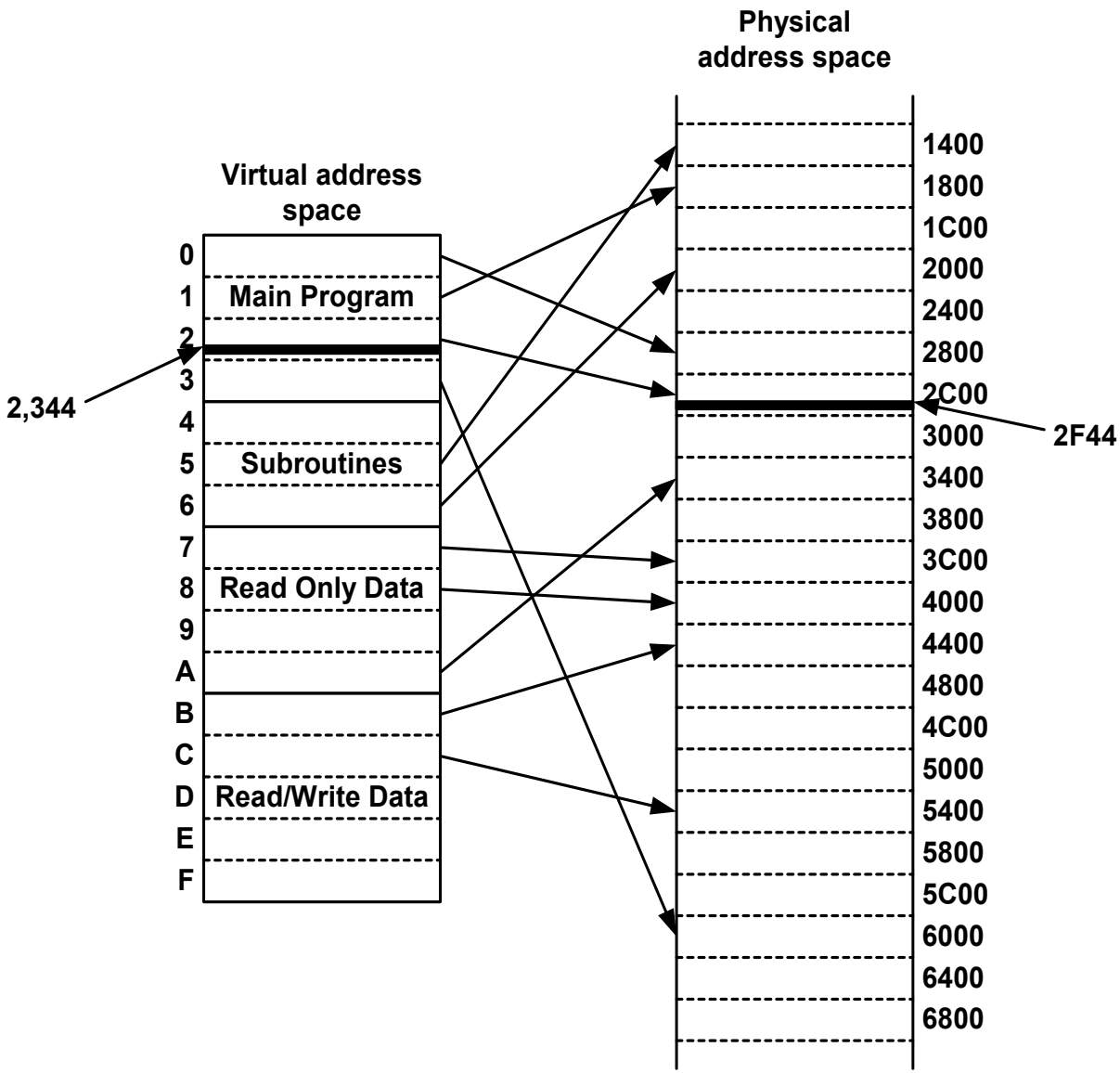
- **Lap**: a program és a memória is fizikailag sok, kisméretű, egyenlő méretű darabokra van osztva (fizikai határok).
- A főprogram, szubrutinok, globális adatok, lokális adatok, verem mind-mind egy-egy azonos méretű lapnak felelnek meg a memóriában.
- A lapozás gazdaságtalan, mert a kis méretű adatot is egy nagy méretű lapba kell tölteni, nem használja ki a rendelkezésre álló lapméretet (belső tördelődés).

Címfordítás lapozás esetén

- **Fizikai Cím=Lap Báziscíme + Lap eltolás (offset) címe**.
- A „+„ itt *konkatenációt / összefűzést*, nem pedig direkt összeadást jelent. Ezáltal sokkal gyorsabb lesz a fizikai cím leképezése, mivel nem kell összeadót használni, a konkatenációt egyszerű huzalozással megoldhatjuk. A hexadecimális címek egy-egy számjegyét 4 bites bináris számmá kódoljuk, és összefűzzük.
- **Példa**: Logikai cím a virtuális memóriában: 2,344= 2. lap száma és a lap 344. rekeszcíme a virtuális memóriában. Legyen a 2. lap címe 2C00. Ezt kell összefűzni a 344-el.

												(Fizikai Cím)
2	C	0	0	+	3	4	4	= 2	F	4	4	
0010	1100	0000	0000	+	0011	0100	0100	= 0010	1111	0100	0100	

Példa: Lapozás mechanizmusa



Page Table

Page	Address	Access
0	2800	Read, Execute
1	1800	Read, Execute
2	2C00	Read, Execute
3	6000	Read, Execute
4	Not in memory	Read, Execute
5	1400	Read, Execute
6	2000	Read, Execute
7	3C00	Read, Execute
8	4000	Read
9	Not in memory	Read
A	3400	Read
B	4400	Read
C	5400	Read, Write
D	Not in memory	Read, Write
E	Not in memory	Read, Write
F	Not in memory	Read, Write

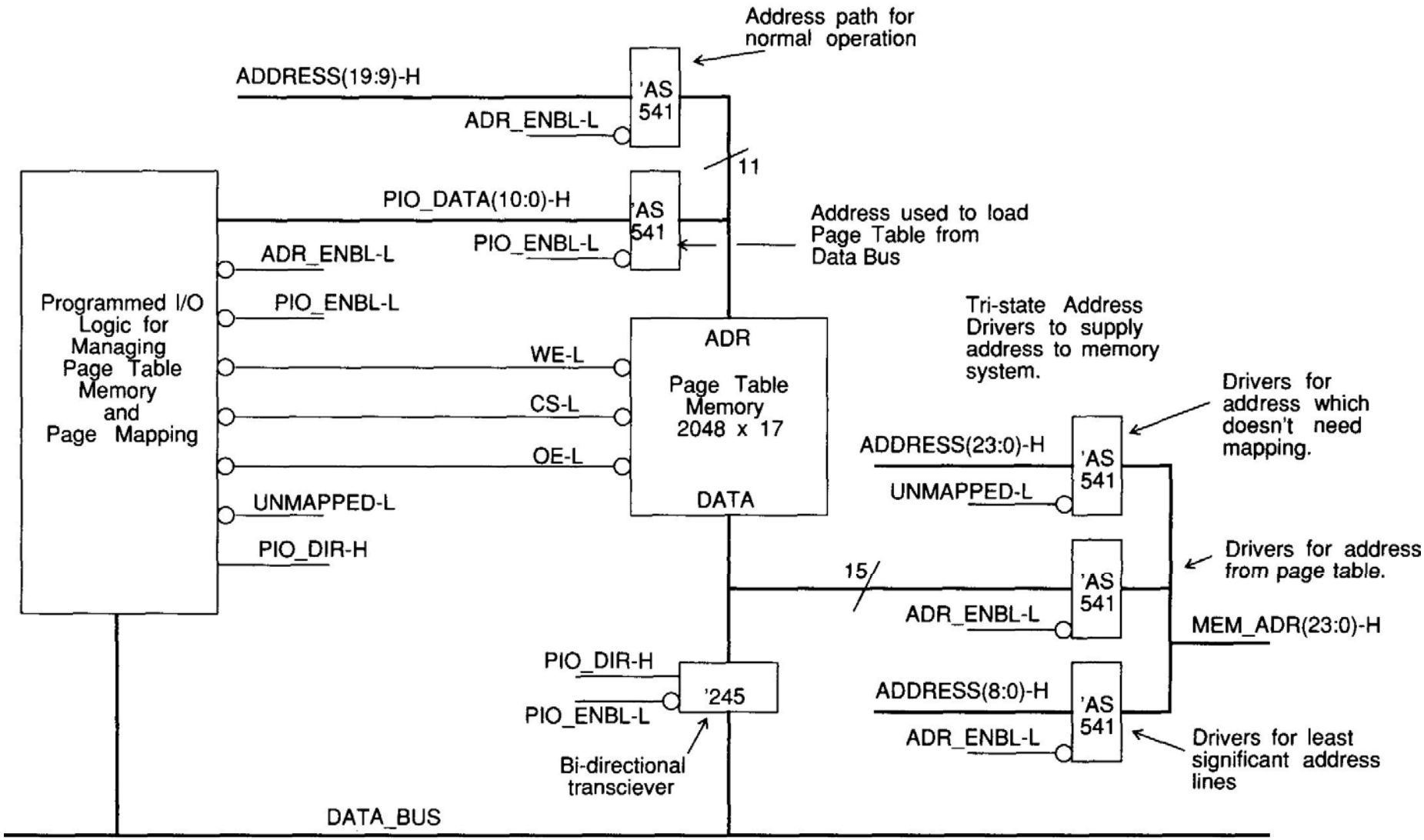
Laptábla bejegyzései

- Az OS minden futó folyamathoz hozzárendel egy laptáblát. A laptáblából olvassuk ki a lap számát (hexa érték), majd a hozzá tartozó lapcímet (hexa), továbbá az egyes lapok hozzáférési információit: R/W/X: olvasható /írható /végrehajtható. A fizikai címet a CPU határozza meg. Mivel a lapok kis egységek, nagy laptábla kell, célszerű őket a memóriában tartani.
- Itt a lapcímeken az összeadás helyett konkatenációt használunk: időt nyerünk (a fenti lapcímben a legkisebb helyiértékű bitek csupa nullák, azokat nem kell hozzáadni; amikben pedig nem csupa nulla szerepel, azokat egyszerűen egymás után fűzzük). A lapok mérete általában mindig kisebb, mint a szegmensek mérete, ezért a kisebb méretű lapok száma több kell, hogy legyen (pl. 1024)

TLB – hit, miss

- **TLB (Translation Lookaside Buffer):** A leggyakrabban használt lapok (LRU) lapcímfordításhoz szükséges adatait tartalmazó átmeneti cache-tár, amely a processzor és a Cache-memória között helyezkedik el.
  - **TLB hit:** Lapozásos virtuális tárkezelésnél azaz eset, amikor a virtuális címnek megfelelő lap a főtárban van, és róla bejegyzés található a TLB-ben (jele: h).
  - **TLB miss:** Lapozásos virtuális tárkezelésnél azaz eset, amikor a virtuális címnek megfelelő lapról nem található bejegyzés a TLB-ben. (jele: m=1-h). Ez akkor fordul elő,
    - - ha a lap a főtárban van, de nincs bejegyzés a TLB-ben (ez a "tiszta" TLB miss);
    - - ha a lap nincs a főtárban, tehát laphiba keletkezett.

Lapozásos címfordító áramkör



Lapozásos címfordító áramkör

- A lapok mérete 512 byte-os, a laptábla *2.048* bejegyzést (lapot) tartalmaz. (Fizikai címet 24-bitesnek tekintjük)
- A megcímezhető max memória 512x2048= *2*^20. (1Mbyte) 20 bit a virtuális címtartomány.
- A cím lapcímből és lapon belüli eltolási címből tevődik össze. Egy lap 512 (2^9) byteos $\Rightarrow$ *9 bitet* használunk (ADDRESS (0:8)) a *lapon belüli hely azonosítására*. További „fennmaradó” (2048 = 2^11) *11 bitet* a megfelelő *lap címének* azonosítására (ADDRESS (9:19)).
- A laptábla tehát 2.048x17 nagyságú, 17 bittel címezhető (17 = 15 bit a címzésre + *2 státuszbit*). A címfordítás csak 15 bites (=24-9). Az egyik státuszbit jelzi, hogy a lap a főmemóriában található-e (hit), ill. a másik jelzi, hogy a betöltés óta módosult-e a lap (dirty bit).
  - **laptábla feltöltése**: a vezérlőjelek: CS:ChipSelect, WE: WriteEnable hatására PIO_DATA-n keresztül adatokkal töltjük fel a laptáblát (11 bites)
  - **normál mód**: A 9 legkisebb helyiértékű címvonalat (ADDR:8:0) kapjuk közvetlenül a címbuszról. Mivel összesen 24 fizikai címvonalunk van a maradék 15 vonal a laptáblából jön. 11 bites címvezetéken keresztül azonosítjuk a megfelelő lap helyét a laptáblában. A laptábla kiválasztja a kívánt lap báziscímét és az OE: Output Enable vezetékre rakja. *Konkatenációval* megkapjuk a fizikai címet. (A 11 magasabb helyiértékűt a 9 alacsonyabb helyiértékűvel összefűzve)

Virtuális (másodlagos) tároló:

$$R_{VM} = \frac{T_{access}}{T_{MS}} = \frac{8ms}{60ns} = 133333$$

Arányossági tényező

- T_{access}: virtuális memória elérési ideje (háttértárolón!) [ms]

$$T_{access} = T_{Seek} + T_{Rotational\_Latency} + T_{Transfer}$$

- T_{MS}: főtár (Main Store elérési ideje) [ns]
  - Ezen peridódus alatt egy 3 GHz processzor, 1 utasítás/órajel ciklus (IPC=Instruction Per Clock cycle)

$3GIPS \cdot 8ms = 24 \cdot 10^6$ utasítást hajt végre

További fontosabb fogalmak:

- OS szinten kezelendő:
  - Lapcsere stratégiák
  - Trashing (vergődés)
  - Térbeli/Időbeli lokalitás elve

Cache memóriák

Kapcsolódó segédanyag: cachecoherence2.pdf

Cache (gyorsító) tárolók

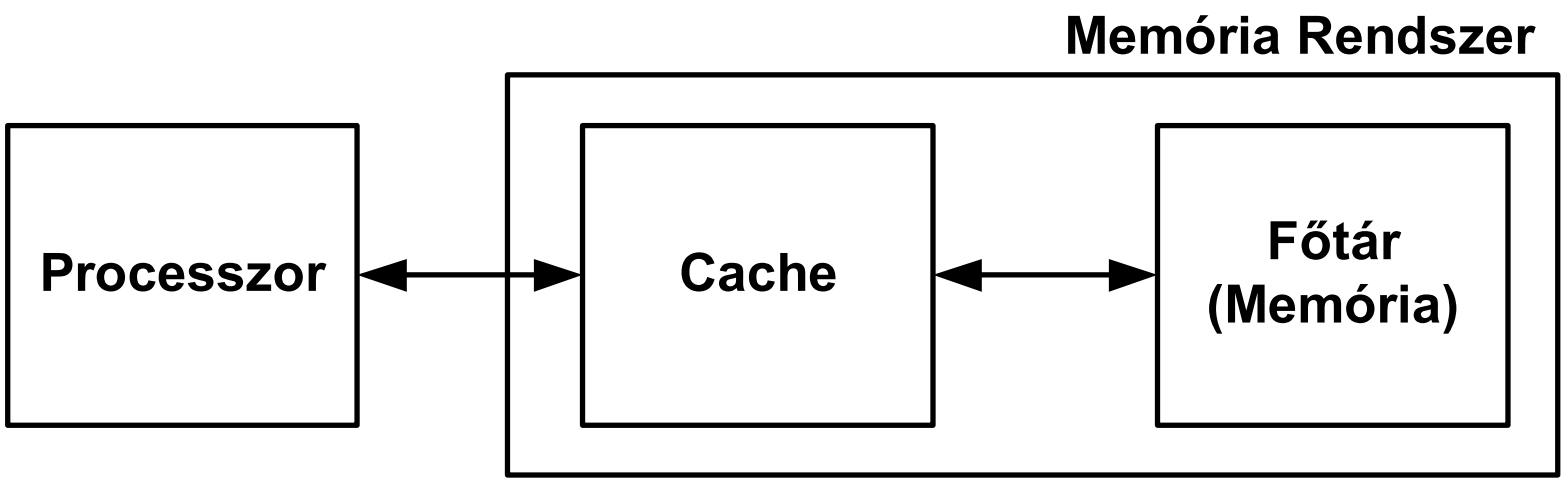
- CACHE: kisméretű, de nagy sebességű memória, amely a processzor és a főmemória között helyezkedik el. Célja a műveletek nagysebességű végrehajtásához tárolást biztosítson a processzor felé. Működése hasonlít a virtuális memóriáéhoz, csak az aktuálisan használt adatokat (*LRU* technika) tárolja. A cache a program (felhasználó) számára rejtett (transparent), nem tudja, hogy használja, csak azt érzékeli, hogy gyorsabb a végrehajtás.
- Cache típusok – elérésük szerint:
  - *közvetlen leképezésű (direct mapping)*
  - *teljesen asszociatív (fully associative)*
  - *n-utas csoport asszociatív (n-way set assoc.)*

Cache memória használatával

$$R_{CA} = \frac{T_{A\_Mainstore}}{T_{A\_Cache}} = \frac{60ns}{1.5ns} = 40$$

- 3 GHz processzor, 1 Instruction/Clock cycle (IPC)

  $3GIPS \cdot 60ns = 180$ utasítás



Egy cache tároló legfontosabb jellemzői a következők:

- a cache-tár *mérete*, a blokk mérete (az adatcsere a főtár és a cache között mindig *blokkos* formában történik),
- egy blokk *kikeresésének* eljárása a cache tárban,
- *aktualizálási* eljárás, amely szerint a processzor által módosított adatot a cache-tárba és a főtárba írjuk,
- a megfelelő *helyettesítési* stratégia (replacement policy), amivel eldöntjük, hogy a cache-ben melyik blokkot lehet felülírni, ha új blokkot kell bemásolni,
- a főtár és a cache-tár adategyezőségének biztosítása (*koherencia*).

- **Cache hit:** Ha processzor olyan adatot igényel, mely a cache-ben megtalálható, akkor találatról, vagy cache hit-ről beszélünk. A találatot a cache-vezérlő azzal állapítja meg, hogy a bemásolt *blokkokhoz* tartozó címrészek (toldalék, vagy „tag”) alapján valamelyik cache blokkban benne van-e a processzor által igényelt adat főtárbeli címe. *Jele: h*
- **Cache miss:** Ha a processzor által igényelt adat nincs meg a cache-ben, ezt tévesztésnek vagy cache miss-nek nevezzük. Jele: *m=1-h*

Cache memória további jellemzői

- T_{CA} : cache memória elérési idő
- T_{MS} : főtár (main store) elérési ideje
- h: cache memória „hit rate”-je

$$T_{EFF} = h \times T_{CA} + (1-h) \times (T_{CA} + T_{MS}) \Rightarrow$$

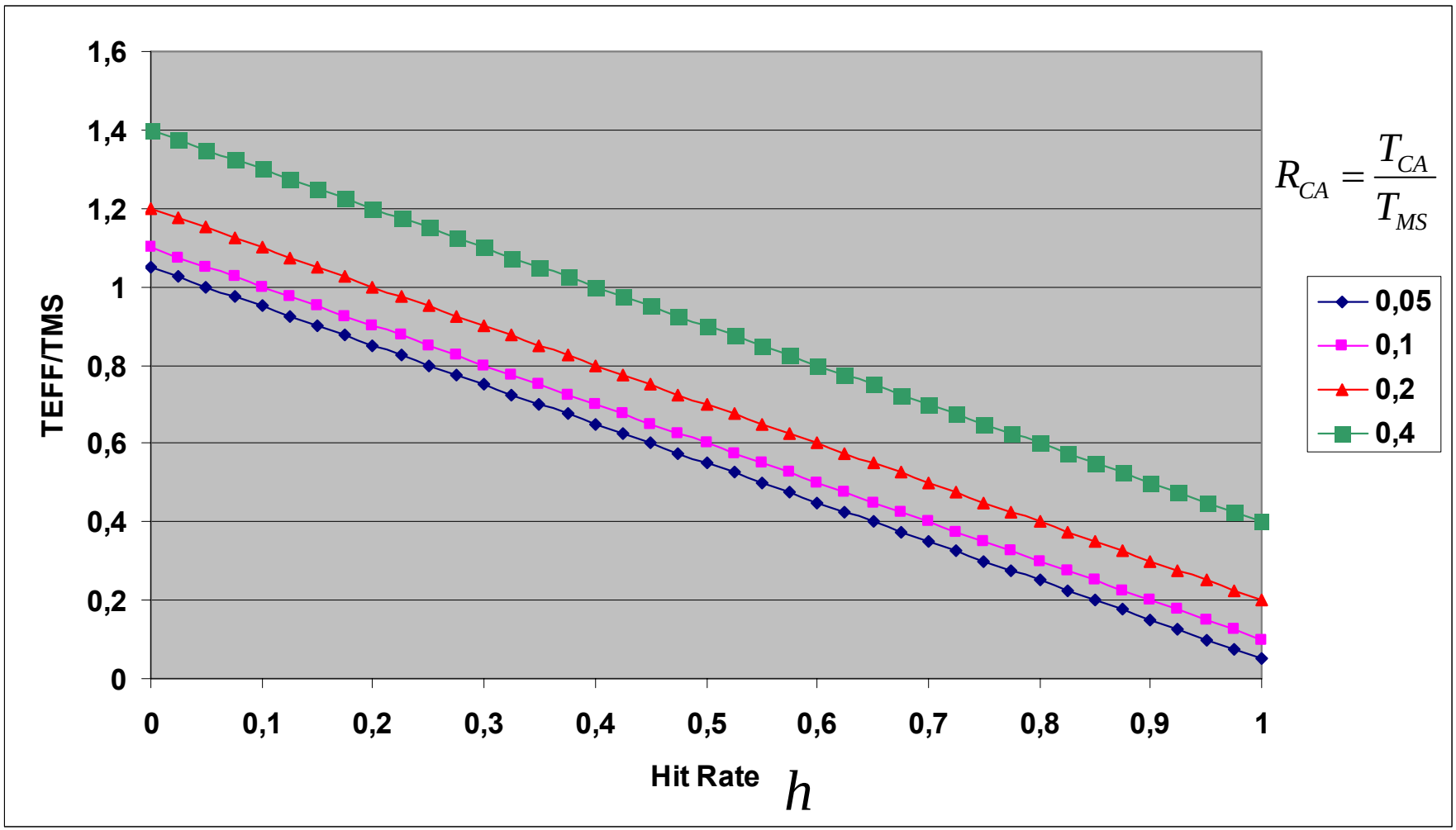
$$T_{EFF} = T_{CA} + (1-h) \times T_{MS}$$

- Effektív memória elérés

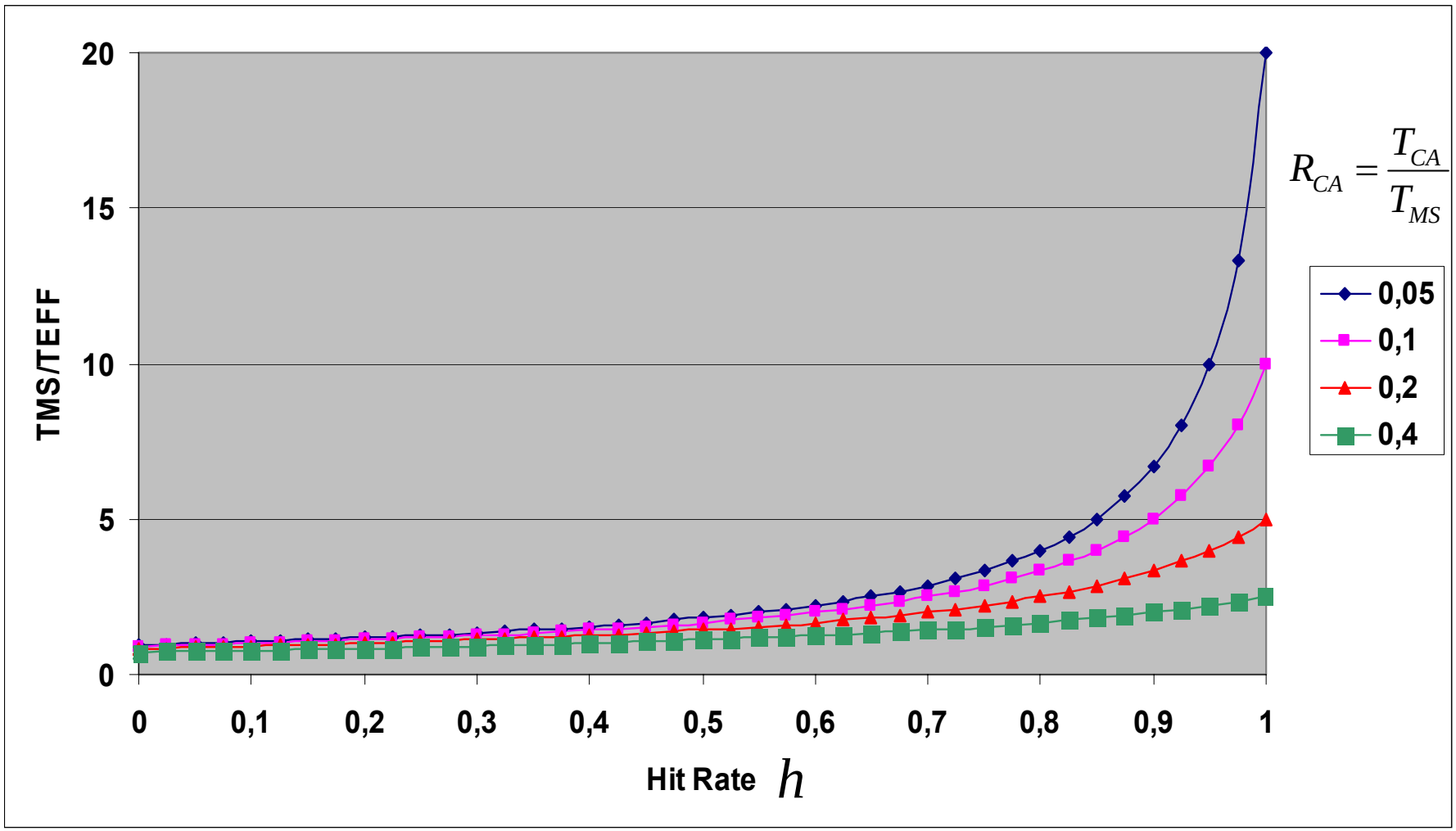
$$S = \frac{T_{MS}}{T_{EFF}}$$

- Speed-up

Effektív memóriaelérés (T_{EFF})



Effektív gyorsulás (speed-up)

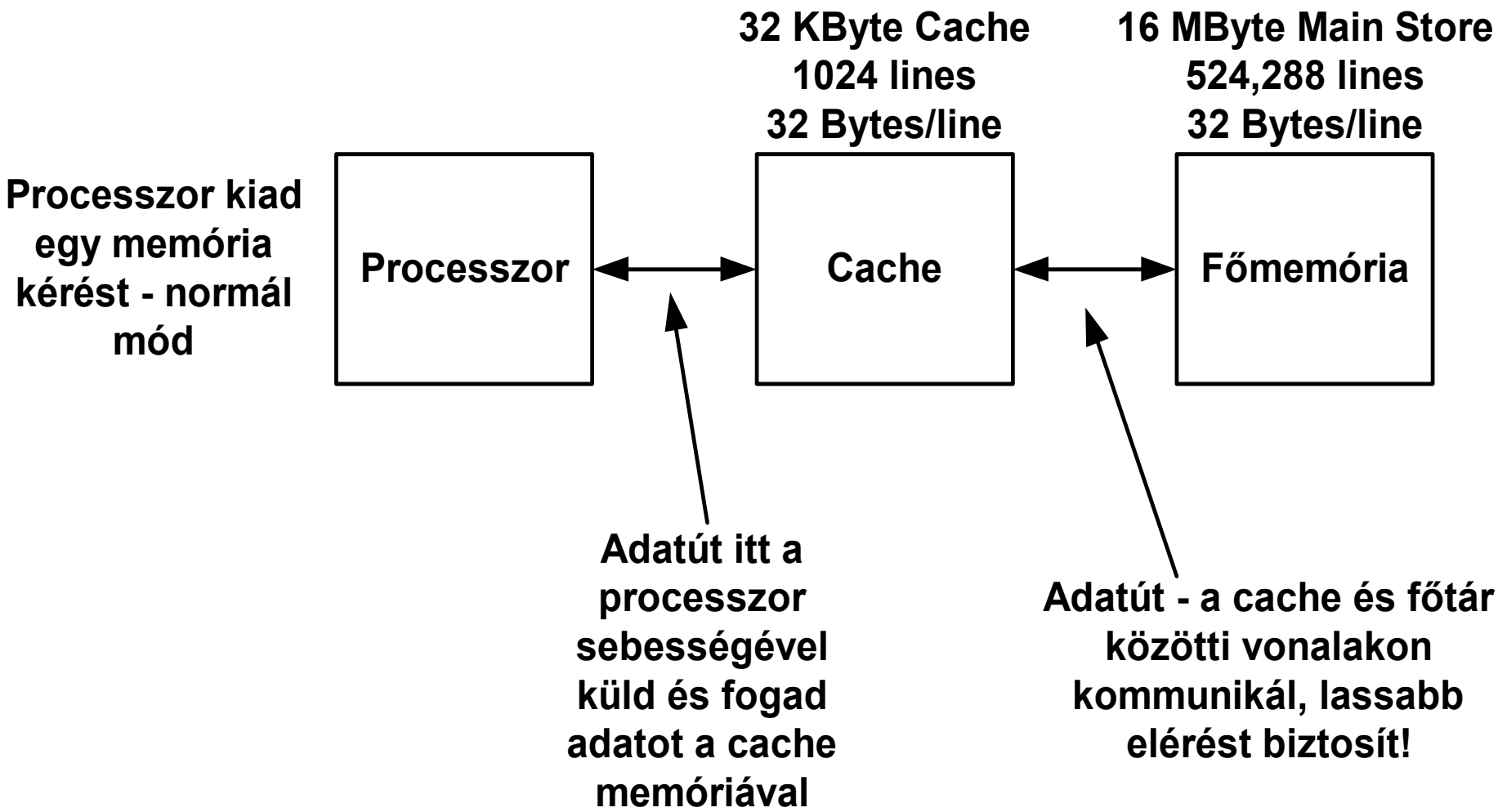


Asszociatív cache típusok

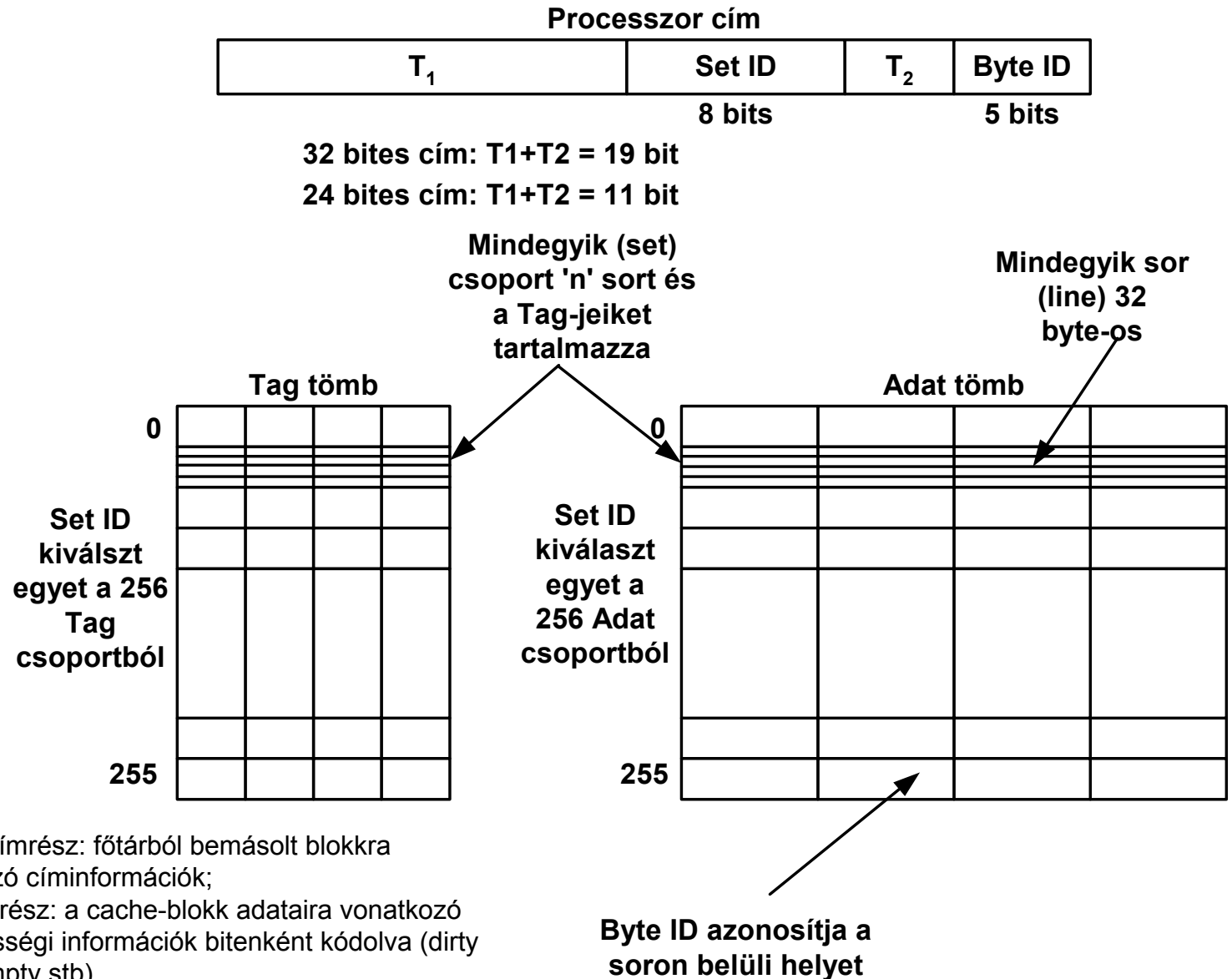
- ***közvetlen leképezésű***: a központi memória minden egyes blokkja csak egy adott sorban (rögzített / direct sorindex szerint) szerepelhet a cache-ben.
  - *Alacsony hit rate*
- ***teljesen asszociatív***: a központi tár bármelyik blokkja a cache-be bárhova betölthető, a blokk címe pedig bekerül a cache toldalék részébe (azonosításhoz).
  - *Gyors és Drága hw-t igényel*
- ***n-utas csoport asszociatív***: Olyan cache-tároló, amely több, "n" sorból álló csoportokra van osztva, és az egy csoporthoz tartozó cache-tárolórész önmagában *teljesen asszociatív* tárolóként működik. Annak megállapítása, hogy egy cache-be írandó blokk melyik *csoporthoz* tartozik, a *közvetlen leképezésű* cache-hez hasonlóan, a memóriacímből képzett indexszel kerül meghatározásra (az előző két módszer ötvözete)
  - Valós cache szervezés

Megj: „***blokk***” alatt a főtár rekeszeinek olyan egymás utáni sorozatát értjük, melynek bemásolása a cache-be egy lépésben megtörténhet.

A cache- és főmemória szervezése



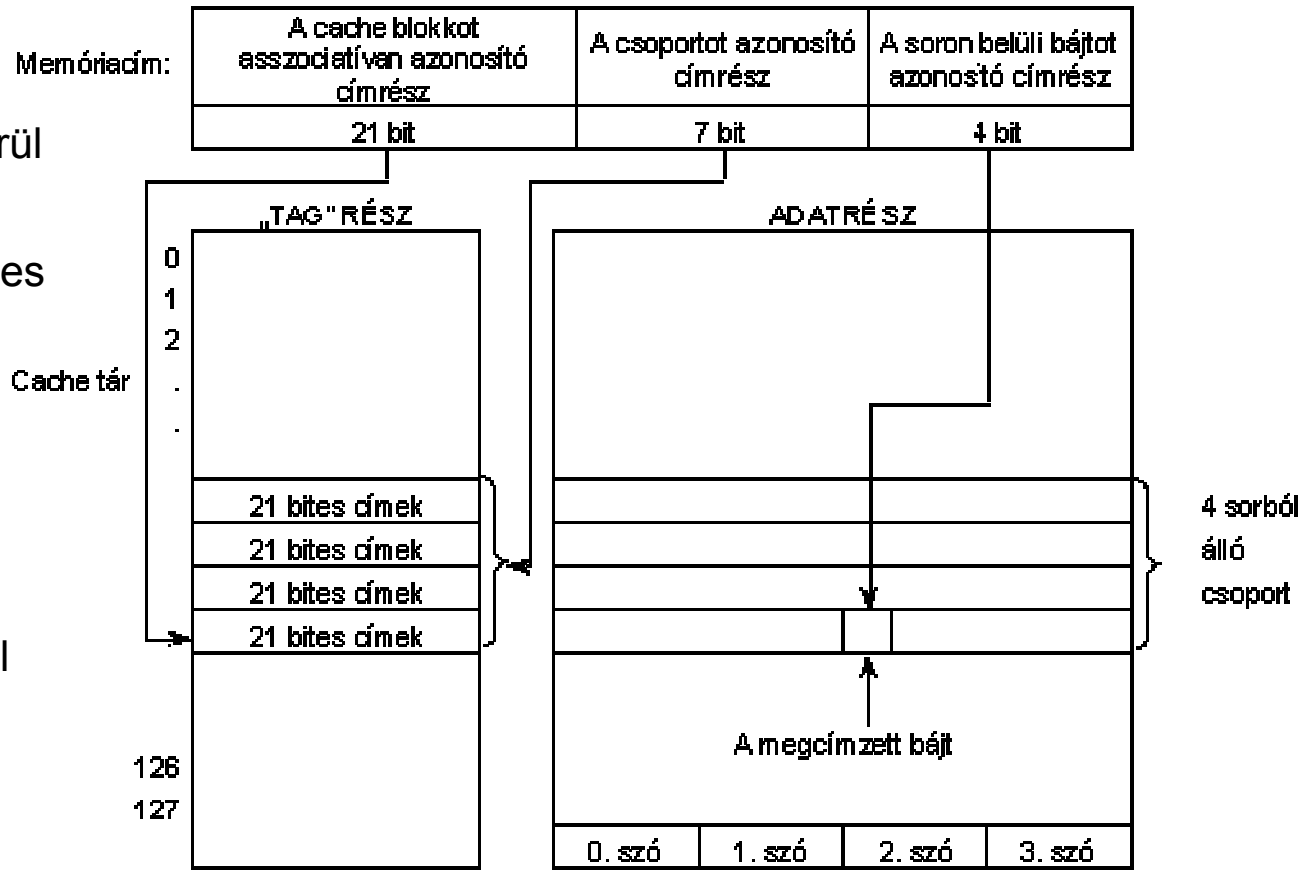
4-utas csoport asszociatív cache



- **TAG**: - címrész: főtárból bemásolt blokkra vonatkozó címinformációk;
  - vezérlőrész: a cache-blokk adataira vonatkozó érvényességi információk bitenként kódolva (dirty bitek, empty stb)

Példa: 4-utas csoport asszociatív cache memória

- Első lépésben a csoportot azonosító, itt 7-bites index alapján a konkrét csoport kerül kiválasztásra.
- Ezt követően a letárolt 21 bites címek asszociatív azonosításával meghatározásra kerül a címnek megfelelő sor.
- Ezen belül egy konkrét bájt megcímzése a 32-bites processzorcím alsó 4 bitjével történhet meg.



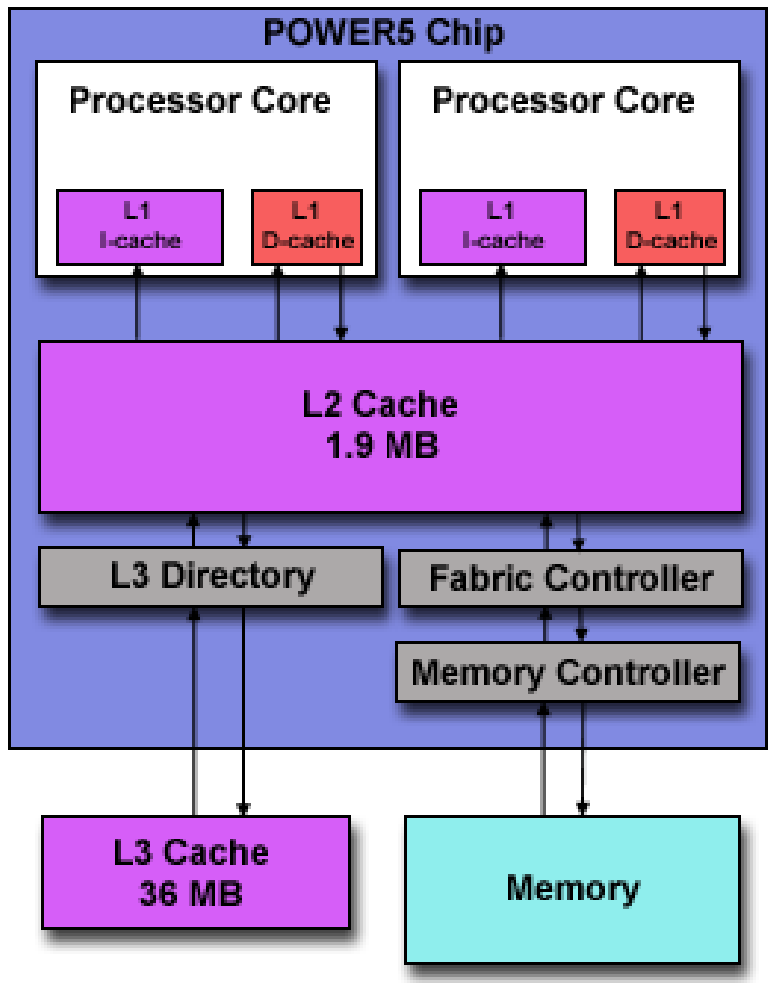
Cache visszaírási (aktualizálási) stratégiák

- Write-through
  - Adat megváltozásakor mind a cache, mind pedig a főmemória egyszerre aktualizálódik
- Write-back
  - Az adat csak igény esetén aktualizálódik, frissül a főtárban (‘dirty bit’ használata)

PI: Cache-hierarchia szintek

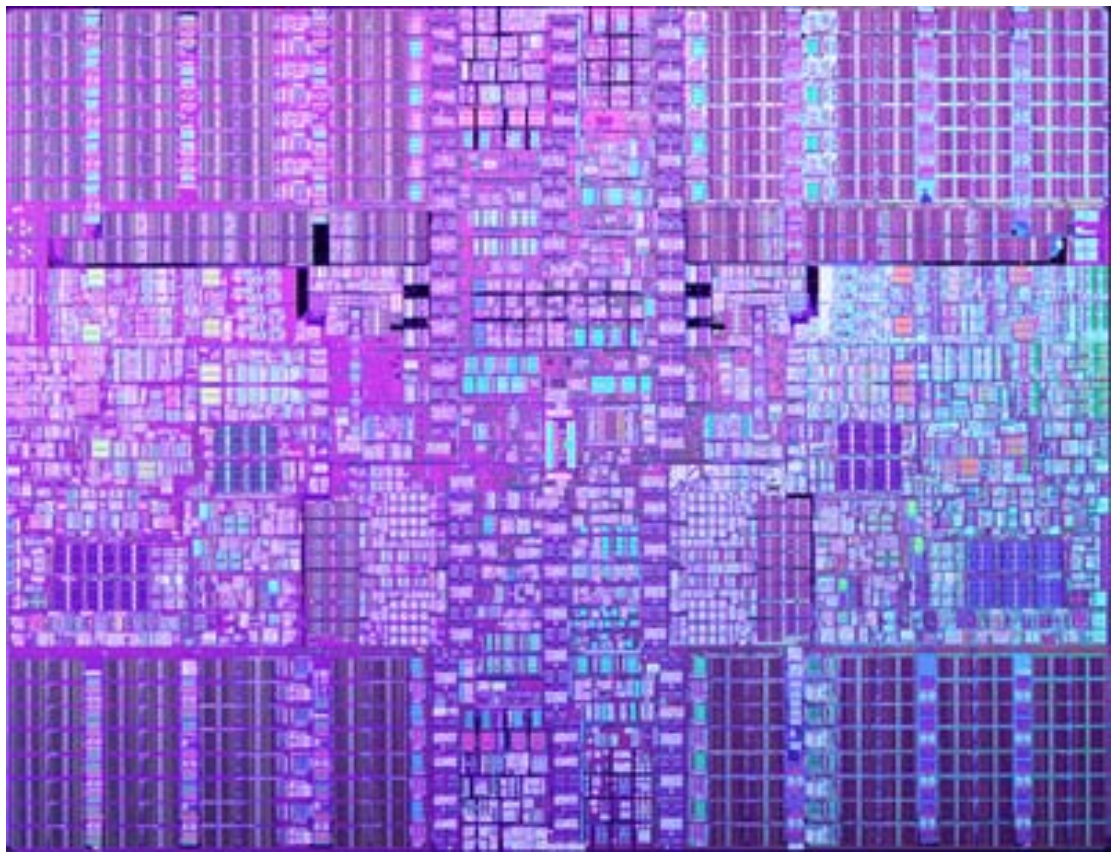
- Egy nagyméretű cache
  - Nagy asszociativitás szükséges
  - lassú
  - kicsit „hit rate”
- Hierarchia típus
  - Inclusive (minden szinten azonos tartalom lehet)
  - Exclusive (kizárólagos)
- Hierarchia szintek:
  - L1
  - L2
  - L3

Példa: IBM Power5



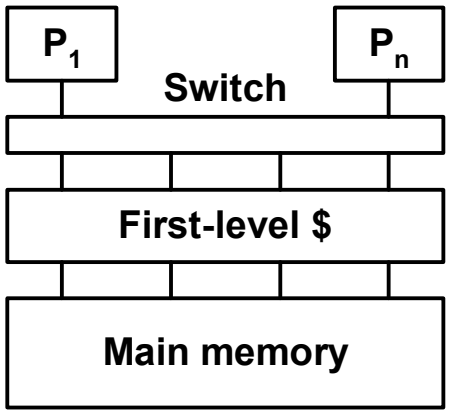
PI: IBM Power6 (2007. május 22.)

- 65nm
- 341mm2
- ~790 millió tr.
- ~100 W
- 2 mag
- 2 thread / mag
  - *in order*
- L2: 4-4 MByte
- L3: 32 MByte



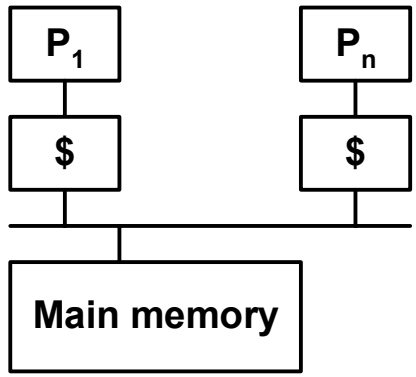
Multiprocesszoros (SMP) rendszerek különböző memória hierarchiái

- a.)



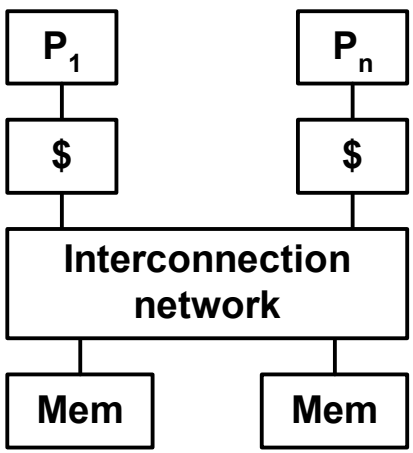
Shared cache = közös

- b.)



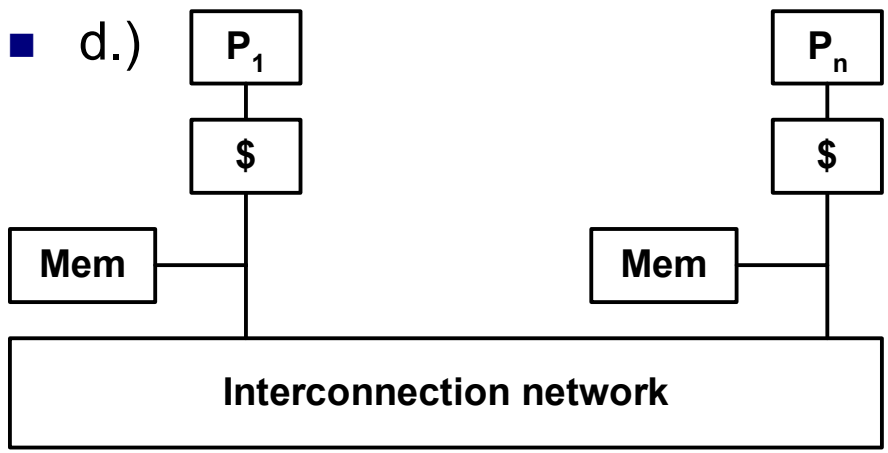
Bus based shared memory

- c.)



"Dancehall"

- d.)

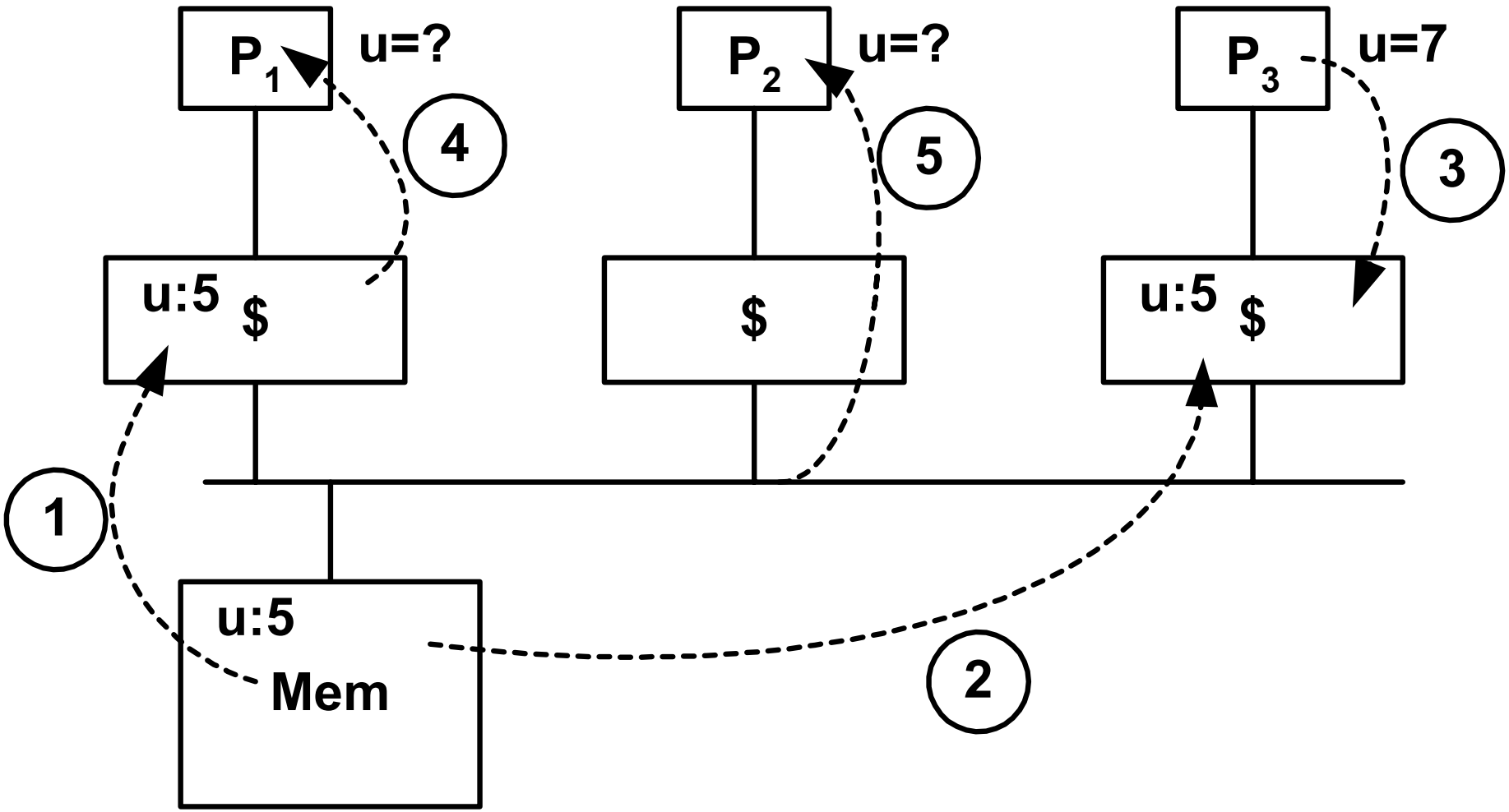


Distributed memory = elosztott

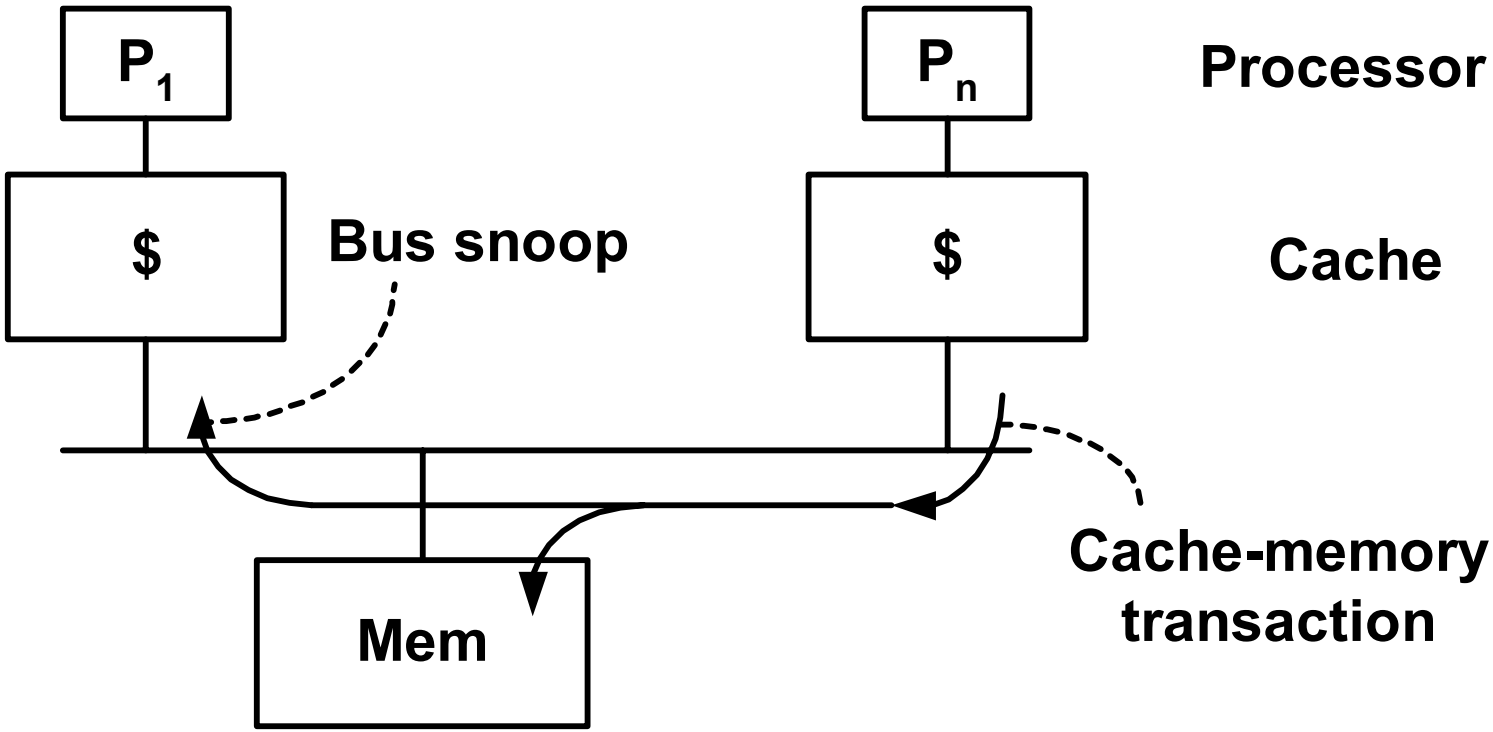
Cache koherencia probléma

- Osztott memóriahasználat esetén lép fel, amikor a **P** processzorok saját **$** cache memóriával rendelkeznek.
- Probléma abból adódhat, amikor ugyanazon osztott memória blokk tartalma (**u** location) megjelenik egy vagy több processzor saját cache memóriájában (read), és pl. történik egy írási tranzakció (egyik processzor módosítja a főmemória tartalmát), a többi processzor pedig még a régi cache-beli tartalommal (másolat) dolgozik, tehát nem aktualizálódnak a cache memóriák értékei.
- Megoldás:
  - Snooping (busz figyelő protokoll)
  - Invalidation / update protokoll

Példa: Cache-koherencia probléma!

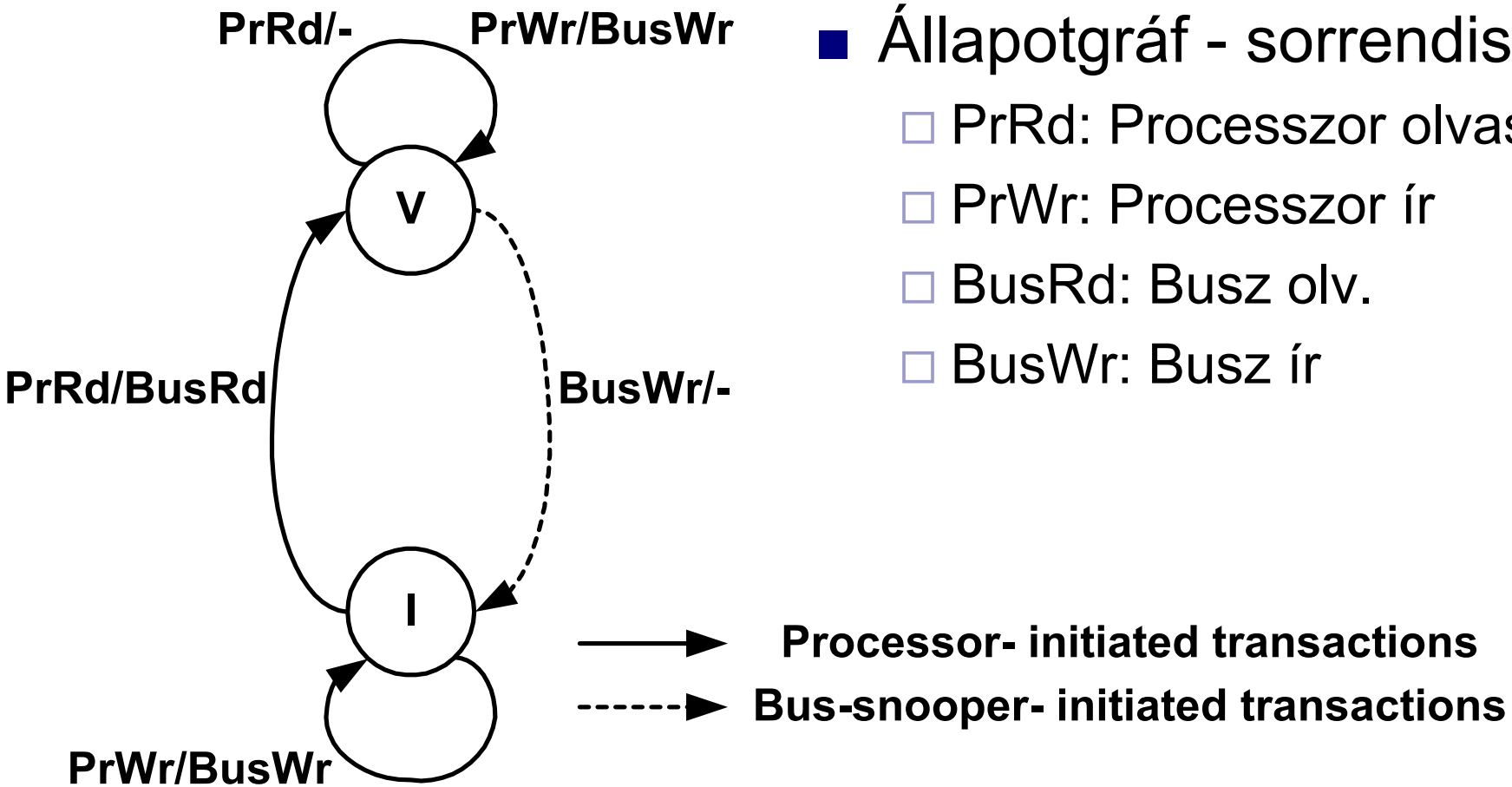


Busz figyelő (snoopy) protokoll



- Busz figyelése (snoop): cache koherencia probléma megoldására (lokális cache vezérlő feladata)
  - minden tranzakció (R/W) és
  - tranzakciók sorrendjének figyelése

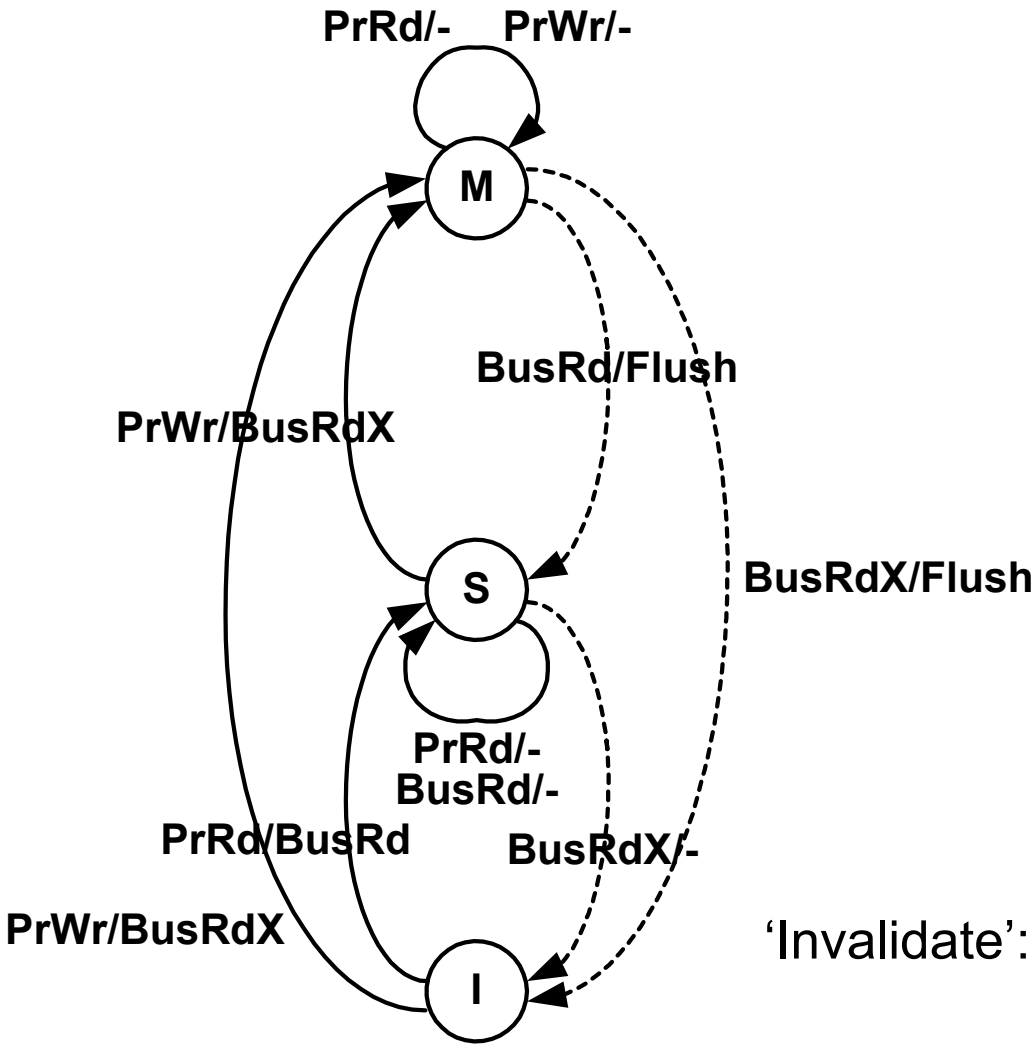
a.) Buszfigyelő (snoopy) protokoll egy *Write-through* cache esetén



- Állapotgráf - sorrendiség
  - PrRd: Processzor olvas
  - PrWr: Processzor ír
  - BusRd: Busz olv.
  - BusWr: Busz ír

- Cache blokkok két érvényes állapota: Valid (V), Invalid (I)

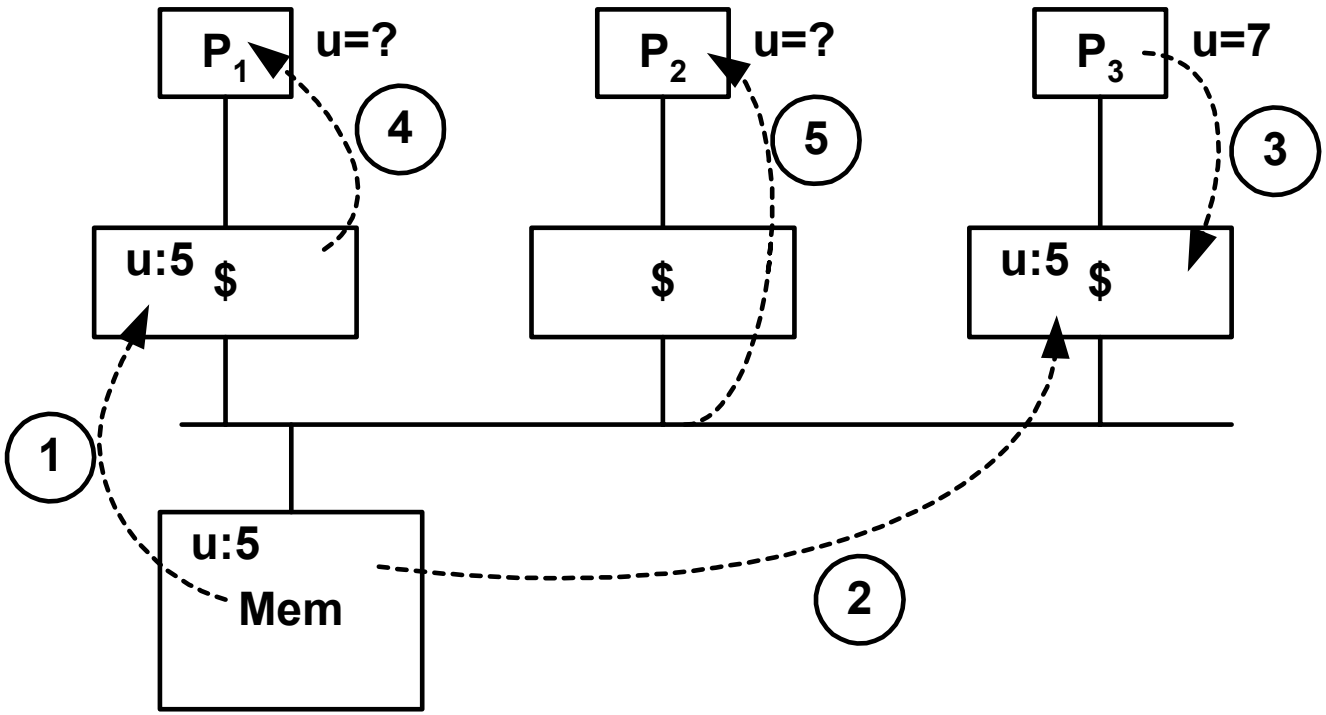
b.) 3-állapotú (**MSI**) *Write-Back* Invalidation protokoll



- M – modified / módosított, érvényes másolat
- S – shared / osztott (nem módosított
- I – invalid / érvénytelen (not present = invalidate)
- Flush – adatot a cache szolgáltatja
- BusRdX - Read exclusive (kizárólagosan olvasható)

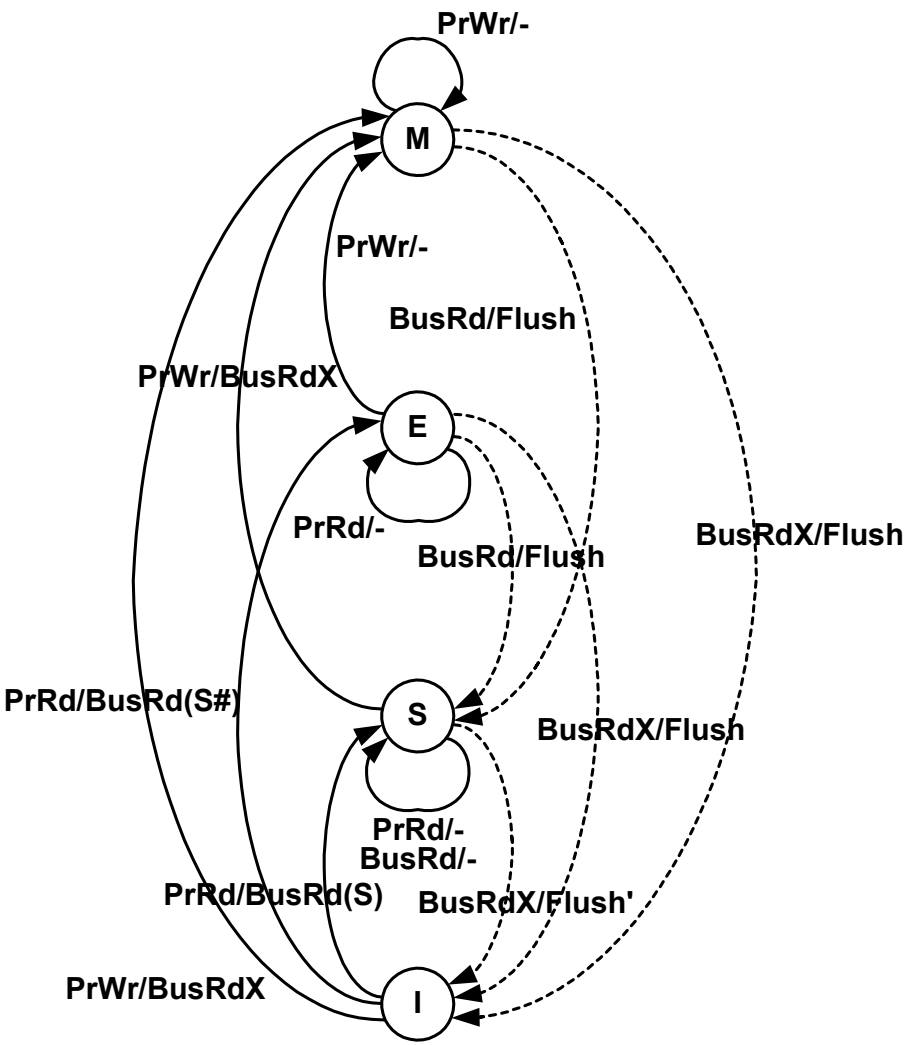
‘Invalidate’: cache másolat érvénytelenítése

Példa: MSI protokoll működése



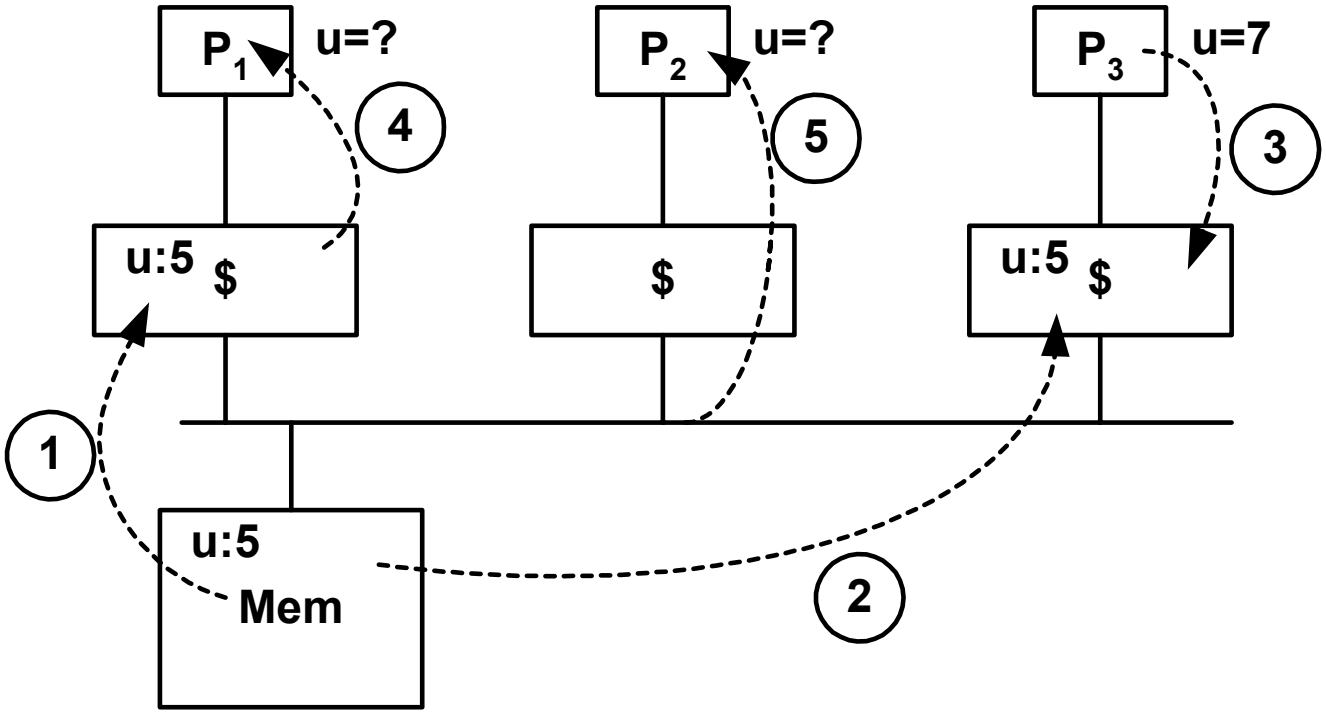
	Processor Action	State in P_1	State in P_2	State in P_3	Bus Action	Data Supplyed By
1	P_1 reads u	S	-	-	BusRD	Memory
2	P_3 reads u	S	-	S	BusRD	Memory
3	P_3 writes u	I	-	M	BusRDX	Memory
4	P_1 reads u	S	-	S	BusRD	P_3 cache
5	P_2 reads u	S	S	S	BusRD	Memory

c.) 4-állapotú (MESI) *Write-Back* Invalidation protokoll



- M – modified / módosított - dirty
- E – exclusive-clean / kizárólagos (csak egyetlen cache tárolja, még nem módosított),
- S – shared / osztott (nem módosított) több CPU is tárolja
- I – invalid / érvénytelen
- BusRd(S) – Busz olvasáskor egy másik cache tárolja az adatot
- BusRd(S#) – Busz olvasáskor nem egy másik cache tárolja az adatot
- Flush – többszörös osztott másolat, így csak az egyik processzornak kell végrehajtania

Példa: MESI (Dragon) protokoll működése – Write Back Update



	Processor Action	State in P_1	State in P_2	State in P_3	Bus Action	Data Supplyed By
1	P_1 reads u	E	-	-	BusRD	Memory
2	P_3 reads u	Sc	-	Sc	BusRD	Memory
3	P_3 writes u	Sc	-	Sm	BusUpd	P_3 cache
4	P_1 reads u	Sc	-	Sm	null	-
5	P_2 reads u	Sc	Sc	Sm	BusRD	P_3 cache