**Számítógép-architektúrák**

**1. Információ feldolgozó modellek**

**1. Vezérlésáramlásos modell:**   
- közös memória az utasításoknak és az adatoknak  
- vezérlő token: utasításról utasításra halad  
- lényegében egy folyamatábrával írható le  
- Fork/Join műveletekkel lehetővé teszi a párh. munkavégzést (ez a programozó feladata)  
- szekvenciális programok, egyprocesszoros rendszerek

**2.** **Adatáramlásos modell:**- nincs vezérlő token, az utasítás akkor hajtódik végre, ha minden operandusa megvan (szorgos kiértékelés)  
- nincs osztott memória  
- precedenciagráffal írható le  
- a párhuzamosság a rendszerből fakad  
- bottom-up szervezésű  
- táblázatkezelő szoftverek, sorrenden kívüli végrehajtást támogató processzorok, modern processzorok belső felépítése

**3.** **Igényvezérlet modell:**- az utasítás akkor hajtódik végre, ha arra igény van (lusta kiértékelés)  
- top-down szervezésű  
- funkcionális programozási nyelvek alapja

**2. Vezéráramlásos architektúrák  
1. Neumann-architektúra:**- Alapelemei: CPU, Memória, Be/Kimeneti perifériák  
- CPU: vezérlőegység (értelmezi az utasításokat és parancsokkal látja el az ALU-t), aritmetikai-logikai egység (ALU: tényleges számításokat végez)

**2. Harvard-architektúra:**- külön memória az utasításoknak (ez lehet csak olvasható) és az adatoknak (lehetőség, hogy a kettőt egyszerre lehessen elérni)  
- Módosított H-a: lehet írható és olvasható is az utasítás memória, de ettől még nem hasonlatos a N-a-hoz

**3. Utasításkészlet architektúrák:  
1. Utasítás készlet:**- utasítások: a processzor milyen utasításokat ismer  
- beépített adattípusok: milyen adattípusokon képes műveleteket végezni  
- regiszterek: tárolják az utasítások eredményeit és operandusait  
- címzési módok: meghatározzák, hogy az utasítások milyen módon tudnak használni egy adott objektumot  
- jelzőbitek: speciális eseményeket jeleznek (pl. zero flag)  
- perifériakezelés módja  
- megszakítás- és kivételkezelés

**2. Utasítások jellemzői:**Utasítások felépítése:  
- kód: az utasítás feladatát, annak paramétereit és operandusait tartalmazza  
- operandusok címei  
- cím, ahova az eredményt tenni kell  
- következő utasítás címe  
Címzési módok:  
- magában a gépi kódban  
- a memóriában  
- a processzor egy regiszterében

**3.** **Jellemzők:**Bájtsorrend:  
- nagy --> kicsi (big-endian)  
- kicsi --> nagy (little-endian)  
Perifériakezelés:  
- memóriára leképzett: valójában a memória egyes címei egy perifériát takarnak  
- külön I/O utasítás a processzor utasítás készletében  
Ortogonális, ha minden címzési mód használható minden címzést használó utasításban  
**CISC:** tömör (rövid utasítás), sokféle utasítás, sokféle címzési mód,  változó utasításhossz,  
**RISC**: fix utasításméret, kevés utasítás,egyszerűbb (a processzor elemi műveleteket végez → kevesebb a hibalehetőség → alacsony fogyasztás, integrálható a kis méret miatt)

**4. Perifériakezelés:  
1. Program által kezdeményezett komm.:  
A, Külön I/O címtartomány:**Összekötés a processzorral:  
- memória és perifériaműveletekre szeparált busz  
- közös multiplex busz  
**B, Memóriára leképzett perifériakezelés:  
2. Periféria által kezdeményezett komm.:  
A, Megszakítások (interrupt):**a processzor befejezi az adott feladatot, elmenti a jelenlegi állását, majd elveszi a perifériától az adatot, ezt követően folytatódik a megkezdett feladat  
**a, polling:** a processzor körbekérdez minden perifériát (prioritási sorrend)  
**b, daisy chaining:** a processzor az adatsínre ránézve, az ott talált azonosító alapján tudja meg, hogy melyik periféria jelzett  
**c, interrupt vezérlő:** el vannak benne tárolva a rá kötött perifériák azonosítói, ezek alapján azonosít, egyszerre bekapott jelzés esetén a daisy chainingnél kifinomultabb prioritási sorrendet alkalmaz  
Hátrányok kiküszöbölése:  
- legyen a lehető legrövidebb  
- kritikus esetekben a megszakítás legyen letiltva  
- az interrupt rátát érdemes maximalizálni

**3. Adatátvitel:  
A, Feltétel nélküli adatátvitel:**- forgalomszabályzás hiánya  
- Adat egymásra futás (a feladó gyorsabban dolgozik, mint a fogadó)  
- Adathiány (fordítva, még nem áll készen a kiolvasandó adat)   
**B, Feltételes adatátvitel:**- adatcsere állapotot tartalmazó regiszterek: „1” ha van adat, „0” ha nincs  
- egy és kétoldali  
**C, Processzoron keresztüli:  
a, programozott adatátvitel:** a perifériák állapotát pollingok figyelik, ha kell ezek állítják meg az adott folyamatot, míg az adatátvitel végbemegy (polling ciklus)  
- ha túl gyakorik a lekérdezések, a processzor túl sok időt veszteget el  
- ha túl ritkák, adatvesztés léphet fel  
**b, megszakításra alapozott adatátvitel:** ha az interrupt értéke 0-ból 1-be megy, akkor lép működésbe  
**D, Processzor megkerülése:  
a, DMA(Direct Memory Access):  
-** DMA vezérlő: szintén egy periféria, így a processzor a periféria utasításokkal tud vele kommunikálni  
- a periféria szempontjából helyettesíti a processzort  
- egyidejűleg van a címsínen a memóriacím és az adatsínen az átadandó adat  
- az adatátvitel végét egy interrupttal jelzi a processzornak  
**b, I/O processzor:**- saját utasítás készlettel rendelkezik  
- a CPU I/O utasításokkal kommunikál az I/O processzorral, így annak a perifériákkal való utasításokat nem is kell ismernie

**4. Összekötések:  
- pont-pont összekötés:** nagyobb sebesség, nincs versenyhelyzet, drágább  
**- osztott busz:** lassúbb, olcsóbb, minél több az eszköz, annál nagyobb a torlódási lehetőség  
**Időzítés:**- szinkron: a buszon lévő órajelet minden periféria megkapja  
- aszinkron: nincs órajel, a vezérlőjelek a meghatározók  
- önidőzítő komm.: egyetlen vezeték viszi át az adatot, bitenként  
**Arbitráció a buszon:**- busz-master: képes a buszt aktívan használni, kezdeményezni, adatot átvinni  
- slave: a fentiekre nem képes, de rajta van a buszon (tehát csak akkor lép tranzakcióba, ha hívják)  
- a buszt egyszerre csak 1 master használhatja, ezt az arbitráció dönti el  
- Centralizált arbitráció: az arbiter jelet küld az első masternek, ez a jel addig megy, míg nem talál olyan perifériát, amely igényli a használatát (prioritási sorrend)(daisy chaining megoldás)  
- Elosztott arbitráció: több megoldás  
a, minden master külön vezetékkel kapcsolódik az arbiterhez + prioritási sorrend  
b, kihagyjuk az arbitrációt, a buszt előzetes lefoglalás nélkül használják (ütközésdetektáláson alapuló busz)

**5. Periféria csatolófelület  
PCI:**- processzorfüggetlen  
- automata felismerés és konfigurálás  
- alacsony fogyasztás támogatás  
- 32 eszköz egy PCI buszon, 256 PCI busz egy rendszeren  
- 32 vagy 64 bites adategységek átvitele  
- burst mód: nem csak egyenként tudja átvinni a biteket, hanem egy hosszabb bitsorozatot is át tud adni  
- rejtett arbitráció: a buszért való küzdelem nyertesének a kiválasztása egy folyamatban lévő tranzakció közben is végbemehet  
**PCI rendszerek:  
Host/PC híd:** ez végzi a fordítást a processzor I/O illetve a memória kérések és a PCI eszközök között  
**PCI eszközök:** egy PCI sínen specifikált eszközök **PCI-PCI híd:** ezzel egy adott PCI sínhez egy másik PCI sín csatolható, így egy hierarchikus rendszer alakul ki  
**Arbitráció:  
-** párhuzamos arbitráció: minden periféria egy saját vezérpárral kapcsolódik az arbiterhez ( az egyikkel a periféria igényel REQ , a másikkal az arbiter visszajelez GNT )  
- rejtett arbitráció  
**Megszakítás:**- 4 csatoló interrupt  
- Üzenet (MSI): saját memóriacímet és azonosító számot kap a periféria, a tranzakció lebonyolításában a Host/PCI híd segít  
**PCI Express:**Megmaradt a PCI-ból: megszakítások kezelése ( bár itt nincs 4 vezetékes interrupt), perifériák konfigurálása (de nem 64, hanem 1024 konfigurációs regiszterrel rendelkezik, az első 64 megegyezik a PCI-éval), tranzakciókon alapuló adatátvitel  
Más lett: soros átvitel (párhuzamos helyett), pont-pont összekötés (busz helyett)  
- Root Complex (Host/PCI híd helyett), switchek  
**Adatátivtel:  
-** Tranzakciós réteg: fejléc (típus, bájtsorrend), rakomány (adat), CRC  
- Adatkapcsolási réteg: sorszám, CRC, célja az adatcsomag hibamentes eljutatása a közvetlen szomszédba  
- Fizikai réteg: kezdő, záró  
**USB:**- szabványos csatlakozás minden periféria számára  
- több periféria is ráköthető egy csatlakozóra  
- 127 eszköz egy rendszeren  
- nem terheli az I/O címteret  
- plug and play  
- olcsó  
- tápellátás  
- alacsony fogyasztás  
**Felépítés:**- USB vezérlő végzi a busz vezérlését → fa topológia  
- a fa gyökere a root hub (a rendszer egyetlen mastere)  
**USB 1.1:  
Tranzakciók:**- LS: 1.5 Mb/s  
- FS: 12 Mb/s  
- a tranzakciók keretekbe szerveződnek, ezek  1ms-ig tartanak  
Izokron átviteli mód:  
- a periféria nem lesz kiszorítva  
- nem garantálja a hibamentességet  
- csak az FS perifériák végezhetik  
- pl.: hangszóró, mikrofon, webkamera  
Bulk adatátviteli mód:  
- tranzakciós késleltetésre nincs garancia  
- minden hibadetektáló és javító eszközt bevet  
- csak FS végezheti  
- pl.: printer, szkenner, pendrive  
Control adatátviteli mód:  
- garantáltan hibamentes  
- lehet LS és FS is  
- a rendszer a következő keret 10%-áig helyet tud adni  
Interrupt adatátvitel:  
- kiolvassa, hogy milyen gyakori lekérdezést igényel a periféria  
- mindig input  
- garantáltan hibamentes  
- lehet LS (8) és FS (64) is  
- pl.: billentyűzet, egér  
**Adatátvitel:**- periféria drivere: összeállít egy adatátviteli kérést  
- USB driver: ezt akkora darabokra bontja, hogy továbbítható legyen a host controller drivernek  
- USB host controller driver: létrehoz egy láncolt listát a tényleges adatátvitel érdekében  
**Tápellátás:**- a kábel 5V feszültséget is szállít  
- a root hub min. 100mA-t, max 500mA-t tud leosztani egy csatlakozónak  
**Kábelek:**- 2 a tápellátáshoz  
- 2 a jelátvitelhez ( az FS 2xesen árnyékolt és csavart, az LS-hez ezek nem szükségesek)  
**Konfigurálás:** mindig az USB driver végzi, csatlakozáskor minden periféria egy saját egyedi címet kap  
**USB 2.0:**Változás: 1ms-es keretek helyett 125 microseces, HS (480 Mb/s)  
ÚJ micro csatlakozók (pl. telefonba)  
Izokron tranzakciók:   
-maximális 1023-ról 1024 bájt, keretenként 1 helyett 3 tranzakció  
Bulk tranzakciók:  
- fix 512 bájt  
- PING tranzakció: megkérdezi a perifériát, hogy készen áll-e  
Control tranzakciók:  
- adatmennyiség: rögzített 64 bájt  
- PING  
Interrupt tranzakciók:  
- HS-ben 1024 lett a szállítható adatmennyiség  
**USB 3.0**:  
**Változások:  
-**SS (3.8Gb/s),  
-üzenetszórás helyett forgalomszabályozás (a beérkező csomagok egy átmeneti tárolóban, bufferben tárolódnak, míg el nem kezdődik a feldolgozásuk)  
- 1 helyett 2 érpár: az egyiken befelé a másikon kifelé mennek az adatok  
- javított forgalomszabályozás: a periféria szól, ha készen áll, nem a root hubnak kell folyamatosan kérdeznie  
- jobb energiamenedzsment  
- min 150, max 900 mA  
- az adatátvitel 2 buszon történik egy a 2.0-áshoz egy pedig az SS-hez (az A típus ugyan olyan kialakítású maradt, a B típus változott)

**6. Háttértárak:**  
**Adattárolási merevlemez (HDD):  
1.Forgólemezes:**- kör alakú lemez (adattárolás)  
- fej: a hordozón tárolt mechanikai, optikai vagy mágneses adatot elektromos jellé alakítja  
- a fejt csak leolvasáskor vagy íráskor kell mozgatni, a lemezek folyamatos mozgásban vannak  
- Adat helye: radiális távolság, elfordulási szög, dimenzió  
**2. Mágneses adattárolás:  
-** ferromágneses anyag (nikkel, kobalt, vas)  
- kis cellákból, mágneses doménekből áll (ezek ellenkező spinűek, de külső mágneses tér hatására egy irányba fognak mutatni), ez a mágneses tér eltávolítása után is megmarad  
- az író/olvasó fej szakaszról szakaszra halad, ezzel változtatva mágneses tér irányát → 0, ha nem változik, 1 ha igen → csak blokkokban lehet írási/olvasási műveletet végezni, nem szakaszokban  
**Felépítése:**- jól záró alumínium ház (belső szennyeződés elkerülése miatt)  
- lemezek: vékony, sima, nem rezonáló, több rétegből állnak  
- lemezforgató motor: minden lemez közös középpontja, alacsony vibráció, tartós, gazdaságos  
- író/olvasó fej (head)  
- fejszerkezet + fejszerkezet mozgató motor  
**Adattárolás:**- szektor: fix hosszú blokk (ált. 512 bájt)  
Szektor részei:  
- rés (2 szektor között)  
- előtag  
- adat kezdete  
- CRC (hibadetektálásra szolgáló hibaösszeg)  
- ECC (hibajavító)  
- töltelék terület  
**Adatazonosítás:  
-** spirális elrendezés (optikai meghajtó)  
- sávos elrendezés (merevlemez)  
Sávos elrendezés:- sáv: ugyanazon a rétegen, ugyanolyan radiális távolság  
- cilinder: a rétegek ugyanazon radiális távolságra lévő sávjainak halmaza (kívülről befelé számozva)  
- fej (head): adathordozó réteg sorszáma  
- CHS (cylinder-head-sector)  
Logikai elrendezés (LBA):  
- a szektorok saját azonosítót kapnak  
- cilinder stratégia  
- szerpentin stratégia (rétegenként)  
**Zóna rendszerű adattárolás (ZRB):** a lemezeket zónákra osztják fel, ezeken belül minden sáv ugyanannyi szektort tartalmaz  
**CLV:** fix adatsebesség, változó forgási sebesség  
**Szervó vezérlés:**- segít a sávváltásban (seek) → sávazonosító  
- segít a sávtartásban → sakktábla szerű mintázat  
**Teljesítmény jellemzők:**- válaszidő: parancs beérkezés → adatátvitel befejezése  
- kiszolgálási idő: parancs feldolgozás → adatátvitel befejezése  
- átviteli sebesség: kiszolgált kérések száma/sec (IOPS), átvitt adatok/sec (pl. Mb/s)  
**Várakozási sor:**- kizárólag seek időre összpontosított: SSTF (a fej aktuális pozíciójához legközelebbihez tart), LOOK(belülről kifelé olvas, ha elért a peremet visszafordul)  
- teljes hozzáférési időre optimalizált: SATF (a forgatási késleltetést és a seek időt is beleveszi)  
**Kiszolgálási idő tényezői:**- parancs feldolgozási késleltetés  
- seek idő  
- forgatási késleltetés  
- leolvasás a hordozóról  
- adatátviteli idő

**Félvezető alapú háttértárak (SSD):  
Adattárolás:**- vezérlő gate, lebegő gate, drain, source  
**Adatok reprezentálása:** egy tranzisztor 1 bitet tárol, ha a lebegő gate fel van töltve elektronnal akkor 0-s, ha nincs akkor 1-es a bit  
**Adatok kiolvasása:** kis feszültséget adunk a vezérlő gate-re és megnézzük, hogy folyik-e áram a source és a drain között: ha igen, akkor 1-es a bit, ha nem akkor 0-s  
**Váltás 1-ből 0-ba (programozás):** nagyfeszültség a drain-en és a vezérlő gate-n + leföldeljük a source-t → az elektronok bejutnak és benn is maradnak a lebegő gate-ben  
**Váltás 0-ból 1-be (törlés):** a source-ra nagy feszültség kerül + a vezérlő gate szigetelődik + a drain-t szabadon hagyjuk → kiáramlanak az eletronok  
**Ezen műveletek miatt a szigetelő rétegek között beszorulnak az elektronok, így annak szigetelési képessége romlik → ÖREGEDÉS  
SLC:** a levegő gate töltöttségét figyelembe véve a tranzisztorok csak 1 bitet tárolnak.  
**MLC:** n bitet tárolnak → gyorsabb öregedés  
**NOR arch.:** a vezérlő gatek-et soronként egy WL-re (word line), a draineket oszloponként egy BL-re (bit line), a sourceokat egy SL-re (source line) kötik  
Bájtonként címezhető, gyorsabb olvasás  
**NAND arch.:**  vezérlő gatek ugyan úgy, a drainek és a sourcok oszloponként sorba vannak kötve, kiindulásuk egy BL vezeték, végük egy közös SL vezeték  
Lap alapú szervezése miatt kitűnő háttértár, gyorsan programozható és törölhető, 2x akkora tranzisztorsűrűség  
**Adattárolás szervese:** bitek → tranzisztorok → lap (a programozás és az olvasás alapegysége) → blokk (a törlés alapegysége) → tárolósík (plane) → szilíciumlap → tok  
**Műveletek:**- Használt: a lap érvényes adatot tartalmaz  
- Érvénytelen: a lap tartalma megváltozott, de még nem írható rá  
- Törölt: a lapra a blokk törlése óta nem írtak, így írásra kész  
**Szemétgyűjtő algoritmus:**1. Keres egy több érvénytelen lapból álló blokkot, majd azokat használt elemeit átteszi az írási frontra  
2. Amint a blokk minden lapja érvénytelen lett, kitörli azt, így írásképes lesz a blokk  
**SSD vezérlő:  
Többletírás:** az SSD saját maga által végzett műveletek  
**Túlméretezés:** az SSD nem teljes tárolókapacitását teszi elérhetővé, annak egy részét saját feladataira fordítja (pl. szemétgyűjtés)  
**Kopás-kiegyenlítés:** egy táblázatban nyilvántartja a blokkok kopását, ez alapján dolgozik  
**Adattömörítés:** egyes vezérlők a duplikációra is figyelnek  
**Több írási kérés bevárása:** ha több írási kérés ugyanarra a lapra vonatkozik  
**Belső RAID szervezés:** a biztonságos tárolás miatt (adatduplikáció)

**Illesztő felületek:**   
**ATA:**- 40 eres szalagkábel párhuzamos adatátvitellel → UDMA miatt +40 föld kábel **- PIO Mode:** az adatátvitelt a processzor végzi el, bájtonként → nem multitaxing képes  
**- DMA Mode:** az adat ki/bevitelt a DMA vezérlő végzi, a processzor feladata csak a ki/beviteli kérés eljuttatása  
**- UDMA Mode:** az adatátvitel az órajel fel és lefutó élein történik → 2x sebesség  
**SATA:**- maradt: parancsok, paraméterek, adatátviteli mód  
- változott: sodort, árnyékolt érpáron soros átvitel történik, pont-pont összekötés a SATA vezérlővel  
- SATA: 150 Mb/s, SATA-2: 300 Mb/s, SATA-3: 600 Mb/s  
- várakozási sor: nem kell megvárni egy parancsot, az adatátvitel egyből indulhat, amint az előtte lévő véget ért  
**SCSI:**- PATA-hoz képest: nagyobb funkcionalitás, megbízhatóbb → drágább  
- párhuzamos adatátvitel 68 eres kábelen  
- lényegében egy busz, amelyre több meghajtó csatlakozik, mint vezérlő → prioritási sorrend  
- max 320 Mb/s  
**Soros SCSI, SAS:**- az SCSI parancsokat ültette át SATA interfészen  
- SATA illesztőkön használható  
- ATA parancsokat küldhetünk egy SAS rendszerrel összekötött SATA meghajtóra  
- 2 csatlakozó: 2 vezérlő (akár 2 külön számítógépre való csatlakozás)  
- max 600 Mb/s  
**FC:**- optikai kábel: nagy sebesség, nagy hatótáv → drága → rézkábelre váltás (csökken a sebesség és a hatótáv is)  
- csomagkapcsolt hálózat, melynek csomópontjai: a vezérlő, a meghajtók, a köztes kapcsolók  
- az SCSI kérést csomag formájában, egymásnak adják tovább míg el nem éri a célt

**7. Memória technológiák:  
SRAM (Statikus RAM):**- egyetlen bit tárolására alkalmas, ehhez 6 tranzisztor szükséges: 6T  
- olvasás: a ponált és negált bitvezetéket is 1-re húzzák, majd a szóvezetékre logikai 1 kerül: ha a bit értéke 1, akkor a ponélt 1, a negált 0, 0 esetén fordítva → ez nem változtatja meg az SRAM állapotát  
- írás: ha 1re akarjuk akkor ponált 1, negált 0, ha 0-ra akkor fordítva  
- nagy sebesség, processzorhoz hasonló gyártás (félvezető lapkára integrálható tranzisztorok) → cache memória  
**DRAM (Dinamikus RAM):**- felépítése: 1T1C (1 tranziszot, 1 kondenzátor)  
- kiolvasás: a bitvetekéket 0 és 1 közötti félútra tesszük, a szóvezeték logikai 1et kap  
destruktív, mert a kiolvasás után a kondenzátor töltése távozik a bitvezetéken, így ezt követően azt egyből vissza kell helyezni  
- írás: kijelöli a cellát és a megfelelő töltést a bitvezetéken át a kondenzátorba kényszeríti  
- frissítést igényel, mivel a töltés magától képes elszivárogni  
- olcsó, nagy adatsűrűség → rendszermemória

**DRAM alapú rendszerek:**- bank: tárolómező, DRAM tárolócellák mátrixba rendezett halmaza  
- chip: független bankokból áll  
- rank: független memória eszköz, több chip-ből áll  
- a memóriamodulok írását/olvasását a memóriavezérlő végzi  
**Két-fázisú hozzáférési protokoll:**1. a sorcímet tesszük rá a sordekóderre  
2. az oszlopcímet tesszük rá az oszlop-multiplexerre  
**Parancsok:  
- ACTIVATE:** adott sor megnyitása **- READ:** megnyitott sorból adott oszlop kiolvasása **- WRITE:** egy nyitott sor adott oszlopát változtatja meg **- PRECHARGE:** nyitott sor bezárása **- REFRESH:** megnyitja és be is zárja a sort → kondenzátorok frissítése  
**Időzítés:  
- TRCD :** addig tart a sor megnyitása, míg annak a tartalma megjelenik az érzékelő erősítőkben **- CL vagy TCAS :** egy nyitott sor egy oszlopának a kiolvasása **- TRP :** a PRECHARGE parancs végrehajtásához kellő idő **- TRAS :** amíg egy sor nyitva lehet (kondenzátor töltés regenerálásához kellő idő)  
**Burst adatátvitel:** a READ/WRITE parancsra nem csak 1, hanem több oszlop átvitele is megtörténik  
Min. érték: DDR=2, DDR2= 4, DDR3/DDR4 = 8

dominik <3 vitatigris roman catapult

**DRAM memóriamodul:  
DRAM chip:**- több egymástól független bankból áll (mindegyik rendelkezik saját sordekóderrel és saját oszlopmultiplexerrel) → egyszerre több sor is nyitva lehet, így kisebb a memóriakésleltetés  
**Bankok száma:** DDR-SDRAM/DDR2 = 4, DDR3 = 8, DDR4 = 16 → az írási olvasási műveletek nem 1-1 bitre, hanem minimális értékűre vonatkoznak  
Ezek sebessége nem sok, így egy memóriamodul több chipet használ, ezek szinkronban dolgoznak  
**Interfészek:**- parancsvezetékek: ezen jelzi a chip a kívánt műveletet  
- bank kiválasztó vezeték: a művelet melyik bankot érinti  
- címvezeték: parancshoz tartozó cím (ACTIVATE esetén sorcím, READ/WRITE esetén oszlopcím)  
- adatvezeték: READ/WRITE esetén ide kerülnek az adatok

**Több DRAM rank használata:**Egymástól független egységek, melyek a parancs, bank kiválasztó, címvezetéken és adatvezetéken is osztoznak → több rank nem növeli a memória sáv szélességét (mivel ugyanazon a buszon vannak)

**Több csatornás rendszerek:**- Szinkron csatorna: csak akkor működik, ha a memóriamodulok egyező mérettel és időzítéssel bírnak  → 1 memóriavezérlőre több memóriamodul csatlakozik (szétoszlanak az adategységek)  
- Független csatornák: 1 memóriavezérlőre 1 memóriamodul

**Memóriavezérlő:  
Címleképzés:**Ha a memóriavezérlőhöz olvasási/írási kérés jön, annak feladata, hogy megkeresse a kéréshez tartozó memóriacsatornát, rankot, bankot, sort  
- FCFS: a kérések beérkezési sorrendben hajtódnak végre  
- FR-FCFS: a kérések átrendeződnek, a gyorsan végre hajthatók kerülnek előre  
**Sorokkal való gazdálkodás:**- nyitva hagyja  
- lezárja  
- APM: megtippeli az előző sorminta alapján, hogy lezárja-e vagy nem  
**Frissités:** a chipekben található időzítők segítségével időnként lefrissít 1-1 sort (hogy ne álljon sokáig a munka)

**DRAM technológiák:  
SDR SDRAM:**- 2 vagy 4 bank  
- burst adatátvitel támogatás (1-8 közötti hossz) **DDR SDRAM:**- sebessége 2x-ese az SDR SDRAM-hoz képest  
- le és felfutó órajelen is végez adatátvitelt  
- min burst = 2 **DDR2 SDRAM:**- belső órajel: parancsok átvitele, belső működés vezérlése  
- külső órajel: a belső 2x-ese, adatok átvitele → egy órajelciklusban 2 adategységet visz át (1-et a felfutón, 1-et a lefutón)  
- min. burst = 4  
- 4 vagy 8 bank **DDR3 SDRAM:**- külső órajel 4x-ese a belsőének  
- burst-öt rögzítették 8-ra **DDR4 SDRAM:**- nincs változás a külső órajelben, se a burstben  
- bankok száma 16 lett  
- nagyobb órajel frekvencia  
- pont-pont topológia: minden egyes DIMM dedikált sínnel csatlakozik a memóriavezérlőhöz **GDDR5 SDRAM:**- nincsenek benne se rankok, se modulok  
- a chipek közvetlenül a vezérlőre vannak kötve  
- 32 bitesek  
- 16 bank, egyszerre 2 sor is nyitva lehet (írás/olvasás egyszerre)  
- 3 féle órajel: cellák órajele, parancs órajel (ez lehet a cella órajel 4x-ese is), adatátvitel órajele (külső memória, a cellák órajelének 16x-osa)  
- drágább, nagyobb fogyasztású

**8. Virtuális memória:**- a processzoron futó programok virtuális címeket használnak, a processzor lábain és a buszon viszont már fizikai címek jelennek meg → címfordítás szükséges,  ezt az MMU végzi  
- fizikai memória bájtjai = nagyobb egységek a virtuális memóriában, ezek a lapok (nem kell külön-külön az adatokat megcímezni, elég az azokat összekötő kapcsolatot)  
**Laptábla (összerendelés, megadja melyik lap, melyik keretre lett elhelyezve):**- lehet globális, de lehet minden programnak külön-külön  
- laptábla bejegyzésekből áll: minden bejegyzés egy lap-keret összerendelést ír le  
EZEK:  
- lap sorszáma  
- keret sorszáma  
- lapot birtokló folyamat azonosítója  
- a lapon végezhető műveletek  
-  Vezérlő bitek:  
         - Valid: 1, ha érvényes az összerendelés és 0, ha a lap nincs benne a fizikai memóriában  
         - Dirty: 1, ha lapra történt írási művelet és 0, ha a lapokat nem kell a diszkre kiírni  
         - Accessed: 1, ha volt mostanában hivatkozás erre a lapra. 0 esetén ezekből a lapokból fog                                                   . válogatni az op. rendszer, hogy melyeket dobja ki.

**Címfordítás menete:**1. a processzor a hivatkozott címet virtuális címnek tekinti → leválasztja a címről a lapszámot és megkeresi a hozzá tartozó laptábla bejegyzést  
2. ha valid=1, akkor keret szám kiolvasását követően megkapja a fizikai címet  
3. ha valid=0, akkor megkeresi a lapot a háttértáron, beolvassa és elhelyezi a fizikai memóriában  
4. ha nincs hely a fizikai memóriában, akkor olyan keretet választ ki melynek bitjei accessed=0 és dirty=0, ezeket áthelyezi a háttértárba

**TLB:** gyakran használt lapok lapszám ←→ keretszám összerendeléseket tárolja  
- a gyors cache memóriában helyezkedik el  
- először hozzá fordul a processzor, ha nem találja meg benne a keresett kerethez való elérést, kénytelen a memóriához fordulni  
- minél nagyobba a TLB lefedettsége, annál ritkábban kell a memóriában lévő laptáblához fordulni  
- nagy a felületi igénye a félvezetőn, nagy a fogyasztása  
  
**Egyszintű tömb:**- a fizikai memóriában folytonosan tárolt tömb, i. eleme az i. laphoz tartó laptábla bejegyzés  
- gyors → egy műveletbe kerül a bejegyzés kiolvasása, hiszen a CPU pontosan tudja, hogy mely címen van a bejegyzés  
- kicsi laptábla bejegyzés (összességében csak a vezérlő biteket és egyetlen cím jellegű információt kell tárolnia)  
- benne kell lennie a memóriában, mérete nagyon nagy is lehet

**Hierarchikus laptábla:**- a labtáblákat kis részekre szedik szét és ezeket lapokon tárolják el. Ezek címeire mutatók mutatnak (ezeket is lapokon tárolják), ezekre további mutatók mutatnak, így egy fa szerkezet alakul ki.- a laptáblának csak a gyakran használt részeit tárolja el a memóriában → nagy ár: keresés esetén az egész fa struktúrát be kell járni, ehhez sok lassú memóriaművelet szükséges

**Virtualizált laptábla:**- a tényleges címfordítási információt tartalmazó laptábla bejegyzéseket folytonosan helyezzük el a virtuális címtartományban (mintha egyszintű tömb lenne), ennek ellenére továbbbra is hierarchikus laptáblánk van, melynek legalsó szintje a virtuális címtartomány  
- az 1. szint a fizikai kezdőcímet és a tényleges címfordítási információt tartalmazó, legalsó szintű laptábla a virtuális kezdőcímet és egy speciális regisztert tárol  
- ebben a megvalósításban a hatékony TLB fontosabb mint valaha, mivel TLB-hiba esetén is a TLB-re támaszkodva keressük meg a laptábla bejegyzést a fizikai memóriában

**Inverz laptábla:**- azért inverz, mert a laptáblákat nem a lap sorszámával, hanem a keret sorszámával indexeljük  
- hash tábla

**Méretezés:**- nagyobb lapok:  
     \* nagyobb TLB lefedettség  
     \* diszk alapú háttértár esetén a nagy a kis lapok között elenyésző a betöltési idő  
- kisebb lapok:  
     \* kisebb az esélye, hogy soha nem használt adatokat is “drágán” eltárolunk- általában 4-8 kB méretűek a lapok

**Cache memória**

* A programok nem véletlenszerűen nyúlkálnak bele memóriába, hanem egyfajta mintázatot követnek, ezek a lokalitási elvek:  
  \*    Időbeli lokalitás: ha egy memóriában tárolt adaton valamilyen műveletet végzünk, akkor valószínű hogy ez újra be fog következni (pl.: ciklikus algoritmusok)  
  \*    Térbeli lokalitás: -||-, hogy a közelében lévő adatokon is műveletet fogunk végezni (pl.: médiatartalom lejátszása)  
  \*    Algoritmikus lokalitás: egyes programok dinamikus adatszerkezetet használnak (pl. láncolt listák), ezek nem időbeli és térbeli összefüggések, mégis szabályos viselkedést mutatnak → gyorsabb elérés
* a lokalitási elvekből adódik: vigyük a lehető legközelebb a processzorhoz a gyakran használt adatokat → cache, SRAM típusú (gyors, rengeteg tranzisztorból áll, költséges, gyorsan melegszik, kicsi)

**Kétféle cache megvalósítás:**- Címzési mód szerint:  
     \* Transzparens: a cache a fizikai és a virtuális memória címeivel címezhető, először a cache-ben            keresünk, ha itt nincs találat akkor fordulunk az operatív memóriához (a programnak fogalma sincs, hogy a cache-hez fordul-e)  
     \* Nem transzparens: a cachenek és az operatív memórának más-más címtartománya van (a program tudja, ha a cache-hez fordul)  
- Menedzsment szerint:  
     \* Implicit: a cache maga dönti el, hogy mely adatok kerüljenek bele és melyek csak a lassúbb DRAM-ba  
     \* Explicit: a futó alkalmazás vagy az op. rendszer felügyeli, hogy mi kerüljön a cache-be (load és store műveletek)  
**Ezek alapján 4 eset:  
-** Transzparens cache: *TRANSZPARENS     IMPLICIT*- Szoftver-menedzselt cache: *TRANSZPARENS     EXPLICIT*- Önszervező scratch-pad: *NEM TRANSZPARENS     IMPLICIT*- Scratch-pad memória: *NEM TRANSZPARENS     EXPLICIT*

**Transzparens cache szervezés:**- alapegysége a blokk (akár a virtuális memóriánál a alap)  
- felső bitjei a blokkszám, alsók a blokkon belüli eltolás  
- járulékos információk:  
  \* cache tag: információ arra nézve, hogy az adott cache blokk az operatív memóra mely részét tárolja   
  \* valid bit: =1, ha érvényes adatot tartalmaz  
  \* dirty bit: =1, ha az adott blokk tartalma módosult, mióta a cache-ben van  
**Teljesen asszociatív szervezés:** az operatív memória bármely blokkja belekerülhet egy cache blokkba, a cache tag ebben az esetben maga a blokk memóriabeli sorszáma  
**Direkt leképzés:** a memóriablokk száma alapján egyértelműen eldől, hogy melyik melyik cache blokkban található, azonban ebben az esetben versenyhelyzet alakulhat ki a cache blokkjaira  
**N-utas asszociatív szervezés:** a fenti 2 előnyeit próbálja ötvözni, a blokkszám alsó bitjei indexelik a megfelelő blokk halmazt (n db lehetőség), és az ezekhez tartozó tag-ek összehasonlításával jutunk eredményre

**Nem transzparens cache szervezés:**Ilyenkor a címtér egy része van az SRAM-ban megvalósítva, továbbá nem kellenek vezérlő bitek, tag-ek, mivel a program pontosan tudja, hogy mit és hol akar keresni a cache-ben, így nem kell semmiféle keresést használni

**Cache menedzsment:  
Betöltés:  
   \*** soha  
   \* amikor a futó program meghívja  
   \* idő előtt  
**Szemét:** az a blokk, ami a be és kikerülés között egyszer sem lett meghívva  
**Idő előtti betöltés:**- explicit menedzsment esetén, az alkalmazás idő előtt kérheti egy blokk cache-be töltését a rendelkezésre álló utasításokkal  
- implicit menedzsment esetén a processzor cache prefetch egysége spekultatív módon próbálja kitalálni, hogy mire lehet szükség a közeljövőben

**Cserestratégia:**- random  
- round robin, körben forgó elv: először az 1-es potenciális áldozatot választjuk, majd a 2-est, majd a 3-ast…  
- legrégebben használt (LRU, ez a legnépszerűbb)  
- nem a legutóbb használt (NMRU): a legutoljára használt kivételével random választunk 1et  
- legritkábban használt (LFU)

**Írási műveletek:**- write-through: ha egy cache-blokkra írási művelet történik, azt rögtön átvezetjük az operatív memóriába  
- write-back: blokk frissülés után nem írjuk ki azonnal, csak ha már kikerült a cache-ből  
- write buffer: komótos tempóban, a háttérben végzi az operatív memóriába való átírást, a cache-ből kikerülő adatok (vagy a write-throught esetén blokkfrissüléskor is) ide kerülnek, hogy majd be legyenek írva az operatív memóriába