MŰEGYETEM 1782

BME

# BELÉPTETŐ RENDSZER TERVEZÉSE

*Számítógép-architektúrák*
*1. gyakorlat*

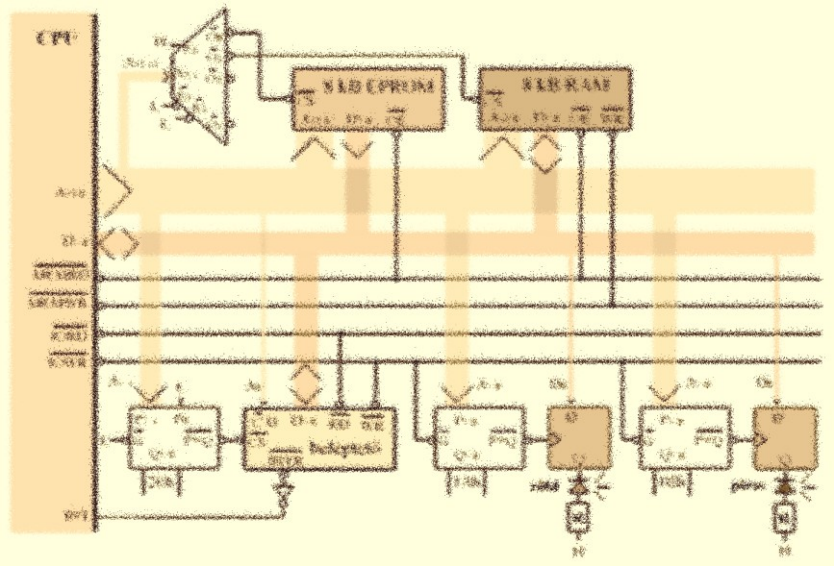

Dr. Lencse Gábor
tudományos főmunkatárs
BME Híradástechnikai Tanszék
lencse@hit.bme.hu

2012. szeptember 3.,
Budapest

# Tartalom

- A rendszer elemeinek megismerése
- A hardver feladat kitűzése
- A hardver megtervezése lépésről lépésre
- Alternatív megoldások vizsgálata
- Összefoglalás


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

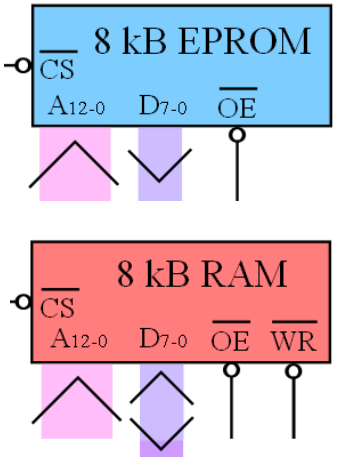


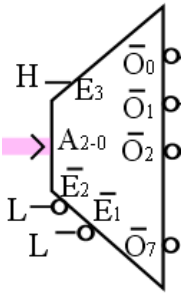
# A RENDSZER ÉPÍTŐELEMEI

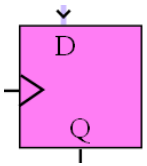

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

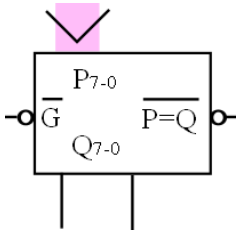
# Adottak a következő építőelemek

- hipotetikus mikroprocesszor
- beléptető periféria
- 8kB EPROM
- 8kB (statikus) RAM
- 3/8-as dekóder
- 2 db D tároló
  - Az órajel felfutó élére tárolja el a D bemenet értékét.
- 3 db 8 bites címkomparátor
  - működése: IF (/G=0)&(P=Q) THEN „/(P=Q)”=0 ELSE „/(P=Q)”=1
- 2 db 250 Ohmos ellenállás
- egy zöld és egy piros LED
  - nyitó irányban 2V feszültségeséssel
  - 5-15 mA áramigénnyel



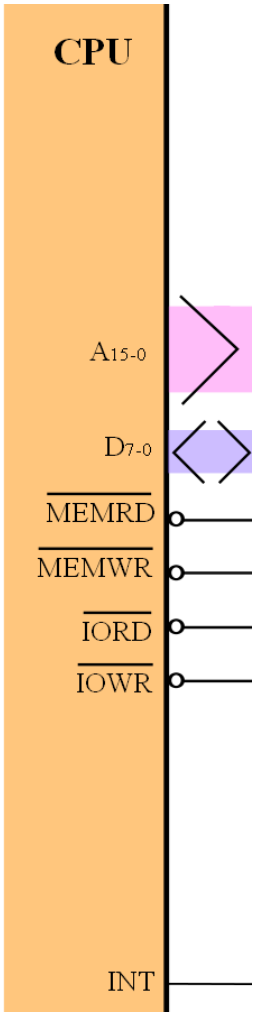



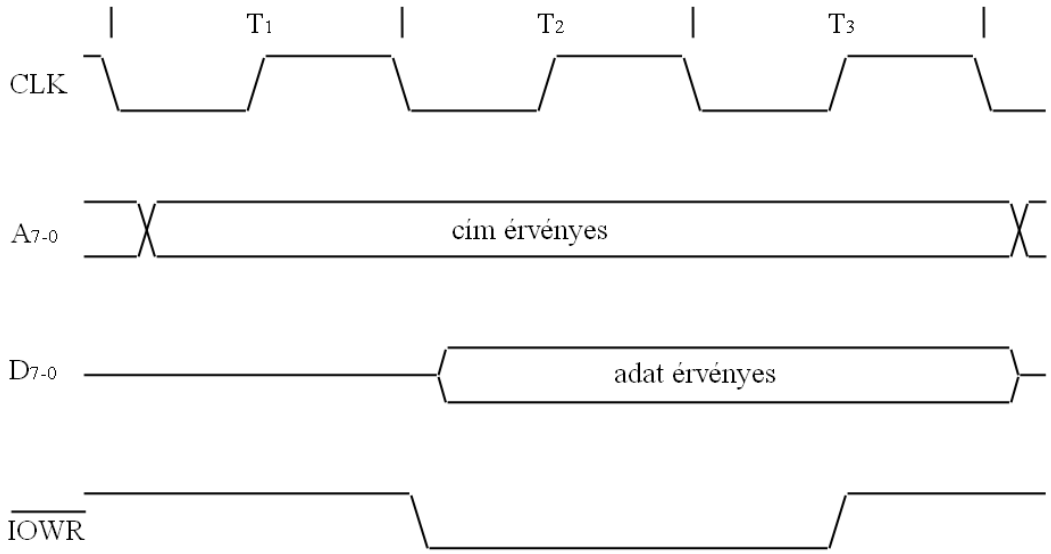
# A mikroprocesszor illesztési felülete

- $A_{15\text{-}0}$ címbusz
  - megcímezhető 64 kB memória, de csak 256 I/O port (az $A_{7\text{-}0}$ címbitekkel címezhető)
- $D_{7\text{-}0}$ adatbusz
- /MEMRD, /MEMWR, /IORD, /IOWR
  - 0 aktív kimenő vezérlő jelek
- INT: felfutó élre érzékeny IT bemenet



# A mikroprocesszor működése

- Működés
  - Kezdetben a 0000h címről indul.
  - Megszakításkor egy CALL 1000h utasítást hajt végre.
    - regisztereket nem ment, a további megszakításokat letiltja.
  - Utasításkészletét később mutatjuk be…
  - A perifériára való írás idődiagramja (az illesztéshez kell):



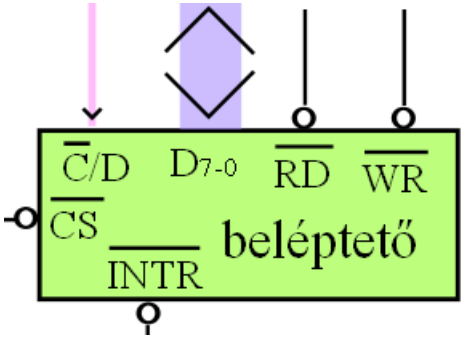
# A beléptető periféria főbb részei

- egy kártyaolvasó
  - a belépőkártyák azonosítója 0-255 egész szám
- egy 0-9 számjegyeket tartalmazó billentyűzet
  - más gomb, pl. törlés nincs
- egy relé vezérlő egység
  - nyitott vagy zárt állásba vezérelhető a zár nyelve
  - az ajtó zárt állásban is becsukódik, de csak nyitott állásban nyitható ki
- a processzor felé való illesztési felület


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A beléptető periféria illesztési felülete

- $D_{7-0}$: kétirányú adatbusz
- /CS, /RD és /WR: 0 aktív bemenő vezérlő jelek,
- (/C)/D (Command=0/Data=1): választó bemenet, hatása:
  - (/C)/D=0 érték esetén: $D_{7-0}$ olvasáskor státusz, íráskor zárvezérlés
  - (/C)/D=1 érték esetén: az adatregiszterből az utolsó eseményhez tartozó számérték (utoljára lehúzott kártya azonosítója vagy utoljára megnyomott számjegy) olvasható ki, az adatregiszter írása hatástalan.
- /INTR: alacsony logikai szinttel jelző esemény kimenet



# A beléptető periféria működése – 1

- Minden kártyához (kártyaazonosítóhoz) tartozik egy 4 decimális jegyből álló kód: a zárat akkor kell nyitni, ha a kártya lehúzása után megnyomott első 4 billentyű éppen a hozzá tartozó kódot adja. (Időzítéssel nem foglalkozunk!) A feladat megoldásához felhasználjuk a beléptető periféria alábbi funkcióit:
  - Amikor a kártyát lehúzzák, a beléptető /INTR lába alacsony szintűre vált, és a státuszregiszteréből olvasva a $D_0$ bit 1-es értékű lesz (akárhányszor is kiolvasható): ezek mindaddig fennállnak, amíg az adatregiszterből ki nem olvasták a kártya azonosítóját. Ezután adatregiszterből kiolvasható (csak egyszer!) a lehúzott kártya azonosítója, utána rögtön /INTR magasra vált, és a státuszregiszteréből olvasva a $D_0$ bit értéke 0 lesz.

(folytatjuk…)

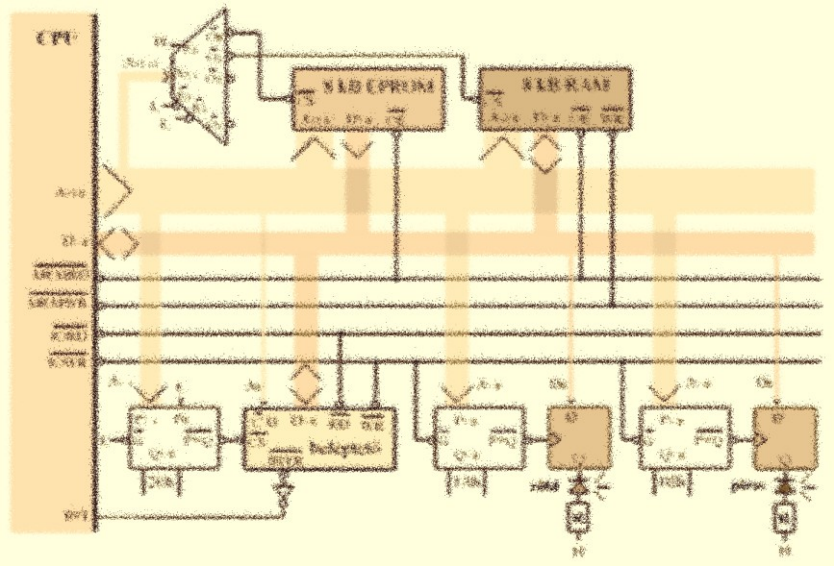

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A beléptető periféria működése – 2

(…folytatás)

- Amikor egy számjegyet beütöttek, a beléptető /INTR lába alacsony szintűre vált, és a státuszregiszteréből olvasva a $D_1$ bit 1-es értékű lesz (akárhányszor is kiolvasható): ezek mindaddig fennállnak, amíg az adatregiszterből ki nem olvasták a számjegy értékét. Ezután az adatregiszterből kiolvasható (csak egyszer!) a beütött számjegy értéke, utána rögtön /INTR magasra vált, és a státuszregiszteréből olvasva a $D_1$ bit értéke 0 lesz.
- A vezérlő regiszter $D_2$ bitjével állítható a zár állása: 0: zárás parancs, 1: nyitás parancs. A nyitás parancs után, amint az ajtót kinyitották rögtön, de legkésőbb 10s után (timeout: ha az ajtót addig nem nyitották ki) a beléptető zárnyelv vezérlője automatikusan átmegy zárt állapotba.
- A státuszregiszter $D_2$ bitjéből mindig kiolvasható a zár állása: 0: zárva, 1: nyitva


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

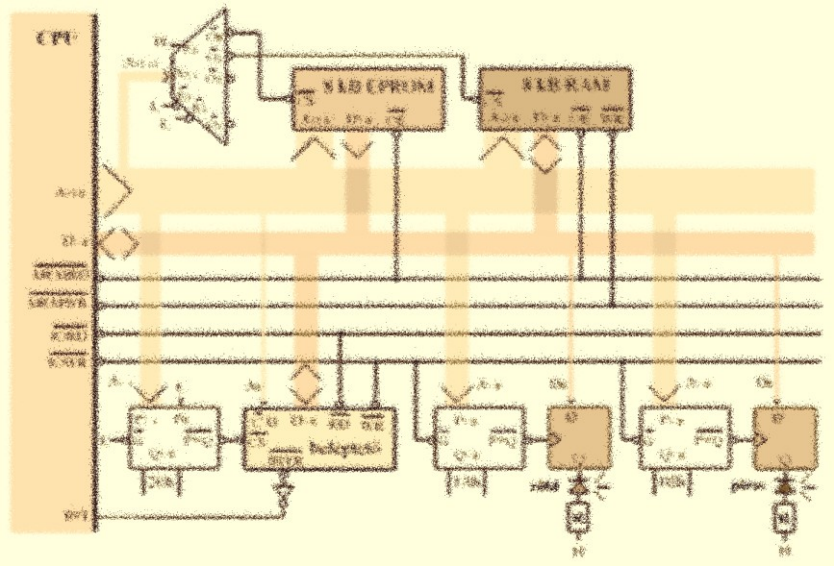



# A HARDVER SPECIFIKÁCIÓJA

# Az építendő hardver

- Illesszünk a hipotetikus mikroprocesszorhoz
  - 8kB EPROM-ot a 0000h címre
  - 8kB RAM-ot a 2000h címre
  - a beléptető perifériát a 2Eh báziscímre
    - így 2Eh/2Fh címeken érhető el
  - egy zöld és egy piros LED-et az 1Ah és az 1Bh portcímekre
    - D tárolók és ellenállások segítségével
    - az adatbusz $D_0$ bitjével lehessen az állapotukat vezérelni


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

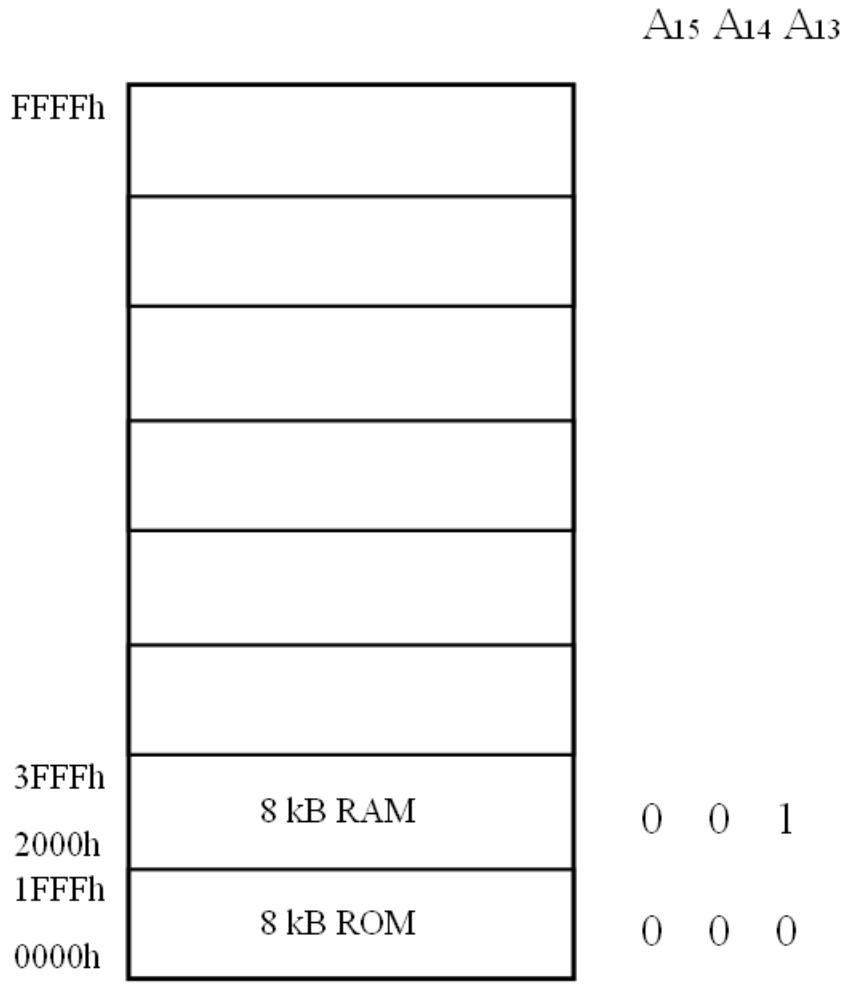


# A HARDVER TERVEZÉSÉNEK LÉPÉSEI


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

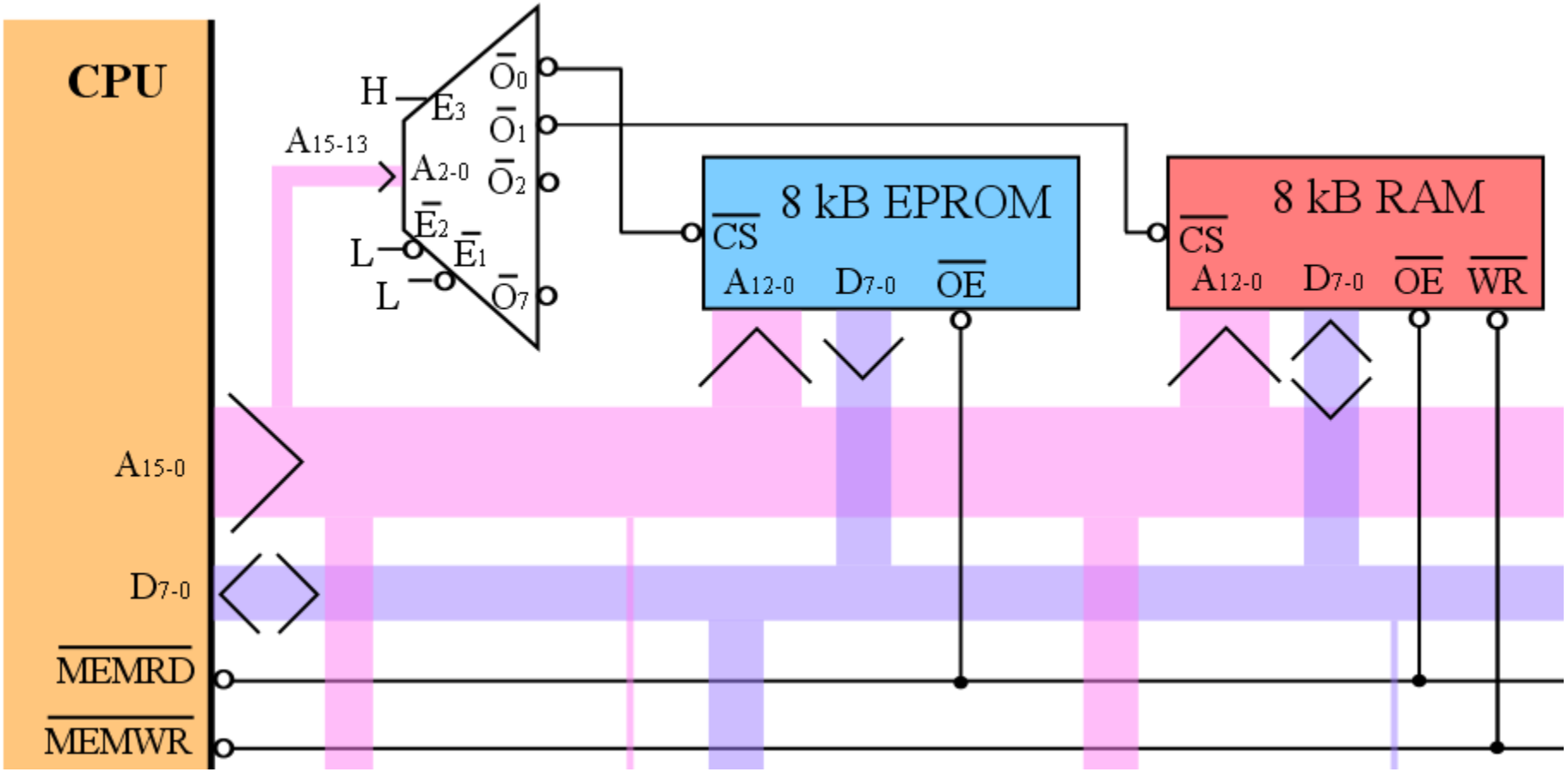
# Memóriák illesztése

- Készítsünk memóriatérképet!
  - Mellette: $A_{15\text{-}13}$ címbitek
  - egy 3/8-as dekóderrel az EPROM és a RAM illeszthető:
    - EPROM: $A_{15\text{-}13}$=000
    - RAM: $A_{15\text{-}13}$=001
    - 3/8-as dekóder $/O_0$ és $/O_1$
  - Méretük 8kB
    - 13 címbit kell: $A_{12\text{-}0}$



# Memóriák illesztése

- A CPU és a hozzá illesztett memóriák

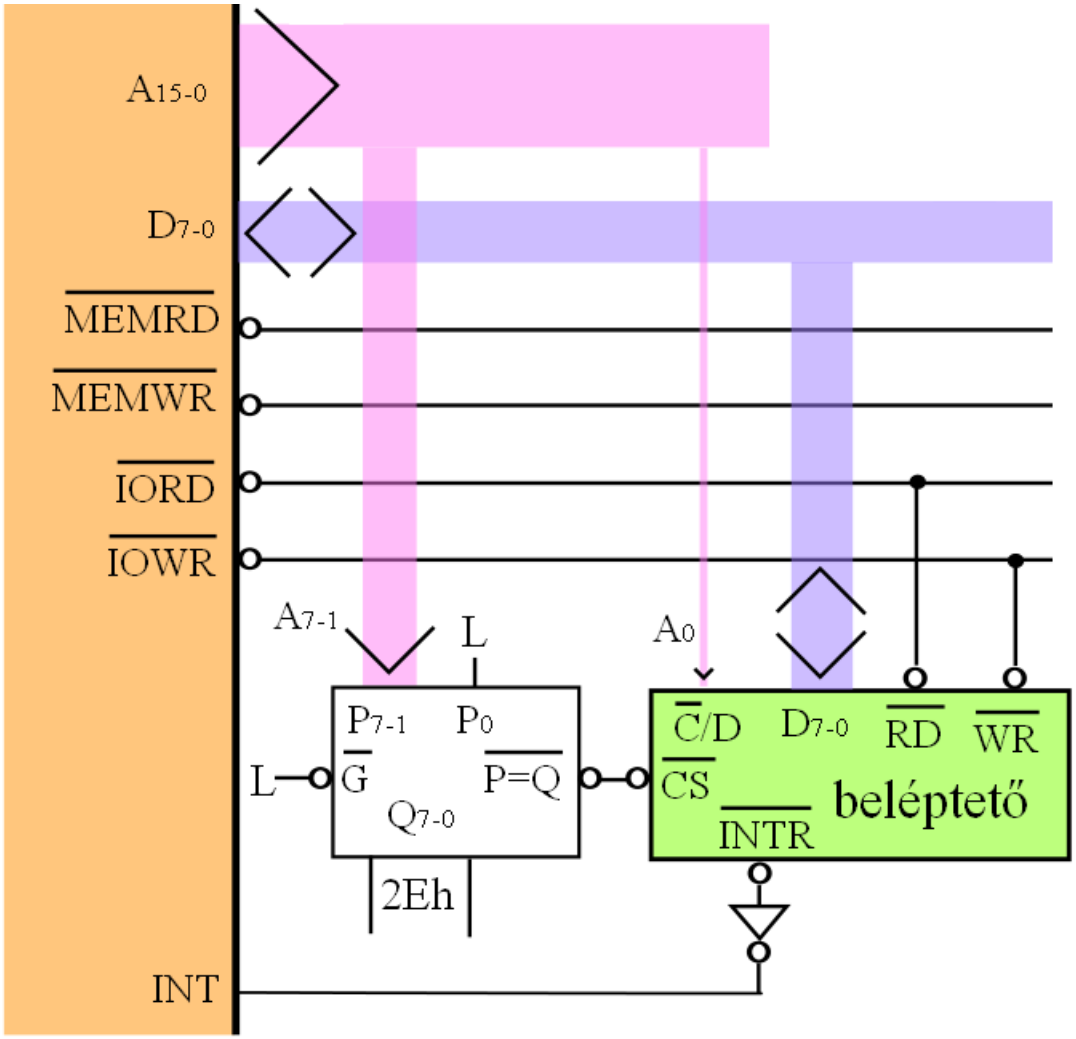


# A beléptető periféria illesztése

- A beléptető perifériát címkomparátorral a 2Eh báziscímre illesztjük:
  - a címkomparátor $P_{7-1}$ bemenetére a címbusz $A_{7-1}$ bitjeit kötjük, de az $A_0$ címbit helyett $P_0$-ra fixen logikai 0-t (L) kötünk, $Q_{7-0}$ bemenetére pedig az 2Eh értéket kapuzzuk.
  - A címkomparátor /G engedélyező bemenetét fixen logikai 0-ra (L) kötjük.
  - A címkomparátor 0 aktív /(P=Q) kimenetével engedélyezzük a beléptető perifériát annak /CS bemenetén.
- Az $A_0$ címbitet rákötjük a beléptető periféria (/C)/D bemenetére;
- az /IORD, /IOWR, és $D_{7-0}$ jeleket pedig értelemszerűen a beléptető periféria /RD, /WR, és $D_{7-0}$ lábaira kötjük.
- A periféria /INTR kimenetét inverteren keresztül kötjük a CPU felfutó élre érzékeny INT lábára.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A beléptető periféria illesztése





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# LED-ek meghajtása

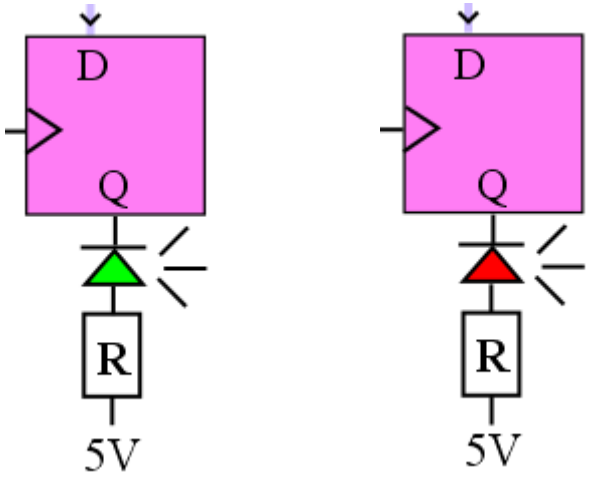
- A LED-ek meghajtásához fontoljuk meg:
  - A D tárolók TTL alkatrészek, kimeneti áram terhelhetőségük
    - logikai 1-nél: < 1mA
    - logikai 0-nál: 20mA környékén

    → csak logikai 0 esetén tudják a LED-eket meghajtani
  - Ha az egy LED-en nyitó irányban eső feszültség 2V, további veszteségekkel (kb. 1V) számolva a 250 Ohm-os áramkorlátozó ellenálláson még kb. 2V feszültségesés 8mA áramot biztosít; ez a LED-ek számára megfelelő.
  - Tehát a LED-eket a D tárolók kimenetéről 250-os ellenálláson keresztül tápfeszültségre (5V) kötjük.
  - FIGYELEM! Így a LED-ek kigyújtásához 0-t kell a D tárolókba írnunk, kioltásához pedig 1-et!


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# LED-ek meghajtása

- Ügyeljünk, hogy a LED-ek rajzjele az elméleti áramirányt kövesse!

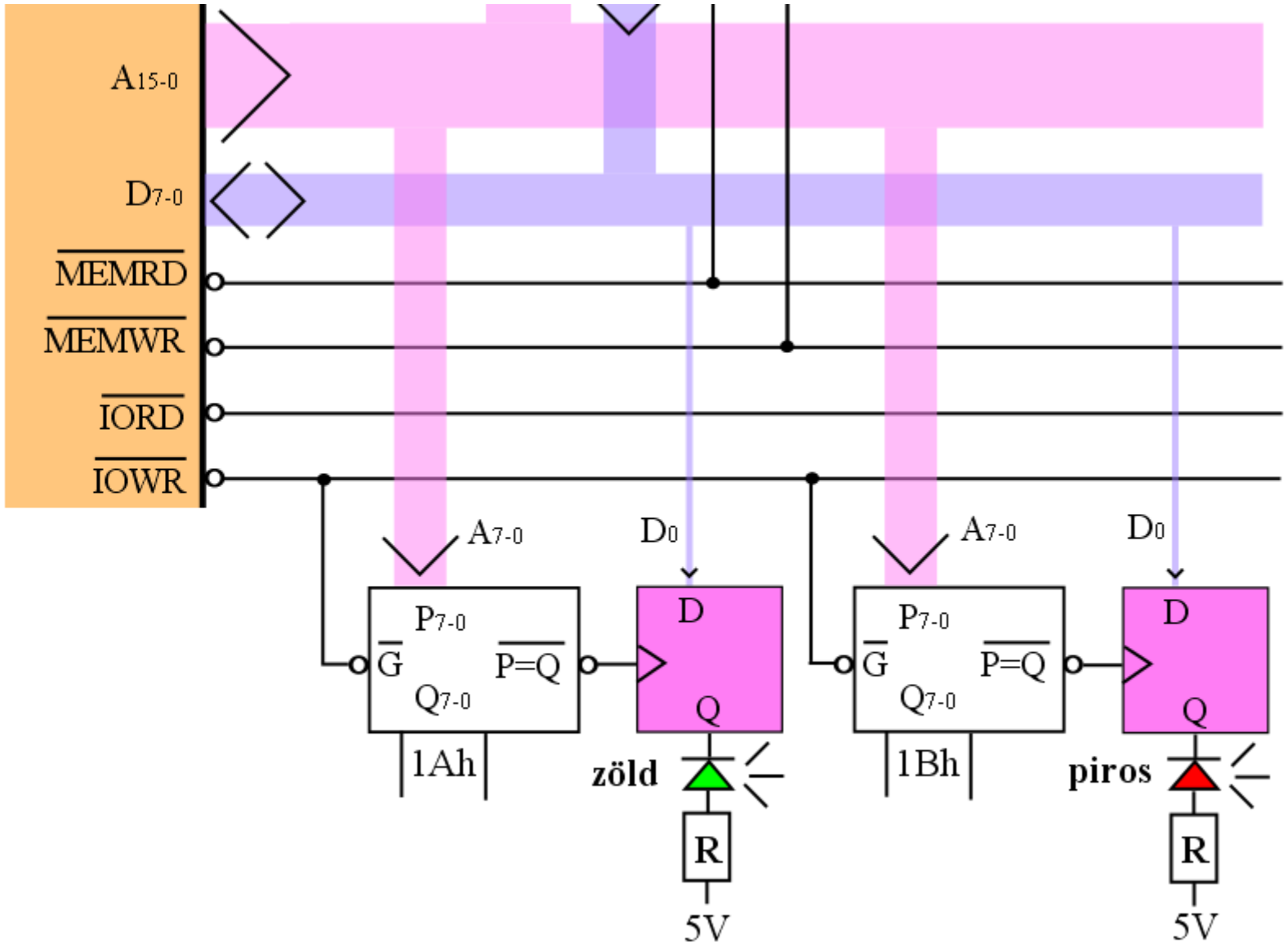


# D tárolók illesztése

- A D tárolókat az 1Ah és az 1Bh portcímekre illesztjük címkomparátorokkal.
  - A címkomparátorok $P_{7-0}$ bemenetére a címbusz $A_{7-0}$ bitjeit kötjük, $Q_{7-0}$ bemenetére pedig az 1Ah illetve 1Bh értékeket kapuzzuk.
  - A címkomparátorok engedélyezéséhez a /G engedélyező bemenetükre a CPU /IOWR kimenetét kötjük, mivel a tárolókat csak írnunk kell.
  - A címkomparátorok 0 aktív /(P=Q) kimenetét a D tároló felfutó élre érzékeny órajel bemenetére kötjük; így az /IOWR megszűnésekor a felfutó él hatására a kiválasztott D tároló eltárolja a D bemenetére kötött adatbusz $D_0$ bitjének ekkor még stabilan tartott értékét.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

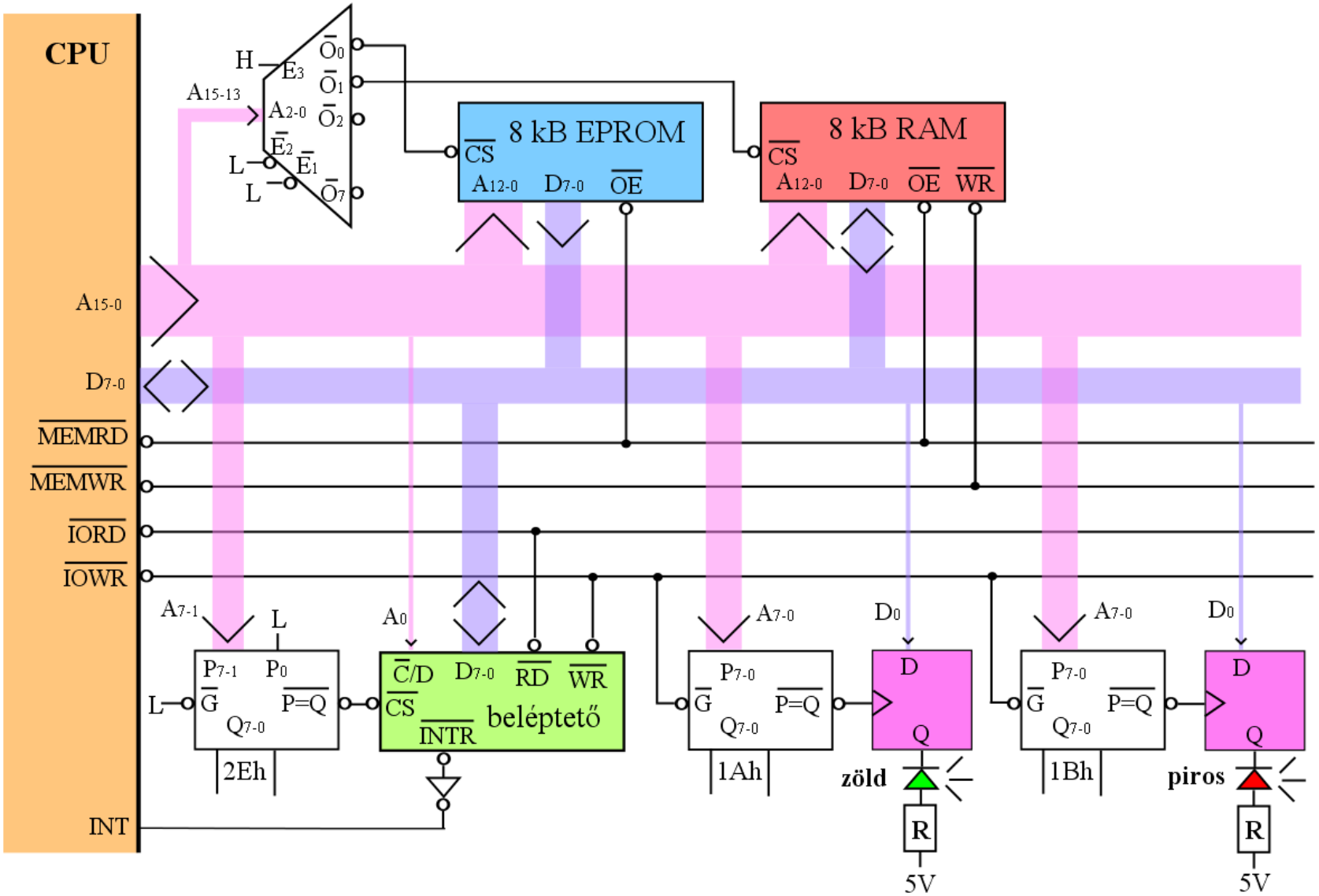
# D tárolók illesztése





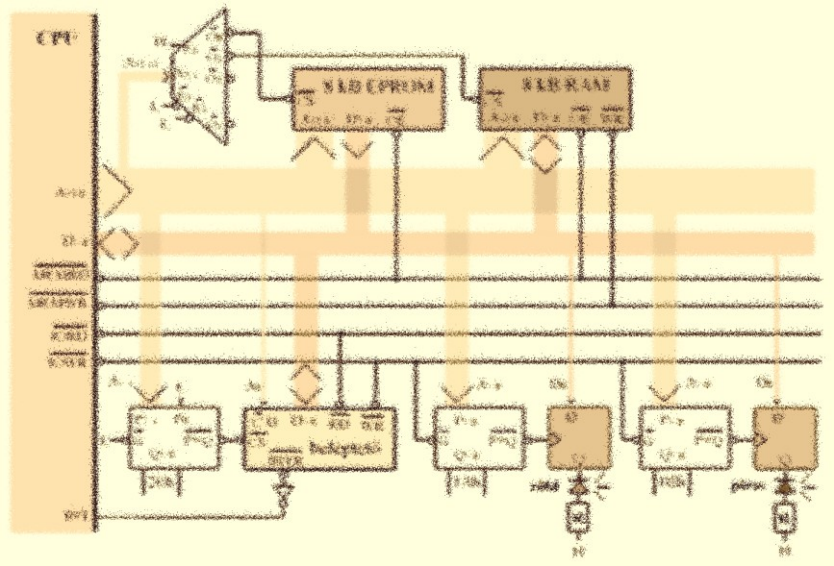
© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A teljes logikai kapcsolási rajz





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem




# ALTERNATÍV MEGOLDÁSOK VIZSGÁLATA

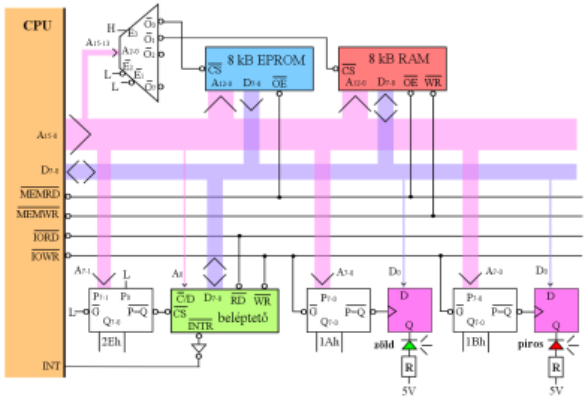
# Önálló hallgatói munka

- **Gondolkozzanak el önállóan az alábbi kérdéseken!**
  1. Hogyan módosítaná a hardvert, ha a 2000h címre nem 8 kB, hanem 16 kB RAM-ot kellene illesztenie?
  2. Milyen megoldást választana, ha a feladat úgy szólna, hogy a LED-eknek akkor kell világítani, ha vezérlésükhöz használt perifériacím 0. bitjére utoljára logikai 1 értéket írtunk és nem világítani, ha 0 értéket írtunk?
  3. Hogyan tudná egyszerűsíteni a rendszert, ha a specifikáció azt írja elő, hogy a piros és a zöld LED állapota mindig egymással ellentétes?


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Összefoglalás

- Megismerkedtünk egy beleptető rendszer építéséhez használható hipotetikus és valós építőelemekkel
- Megadtunk egy konkrét specifikációt
- Megterveztük a kívánt hardvert
  - Memóriát illesztettünk
  - Perifériát illesztettünk
  - LED-ek meghajtásáról gondoskodtunk
  - D tárolókat illesztettünk (csak írásra)




- Alternatív megoldásokat is megvizsgáltunk


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Kérdések?

## KÖSZÖNÖM A FIGYELMET!

Híradástechnikai Tanszék

Dr. Lencse Gábor
tudományos főmunkatárs
BME Híradástechnikai Tanszék
lencse@hit.bme.hu

MŰEGYETEM 1782

BME

# BELÉPTETŐ RENDSZER TERVEZÉSE

*Számítógép-architektúrák*
*2. gyakorlat*

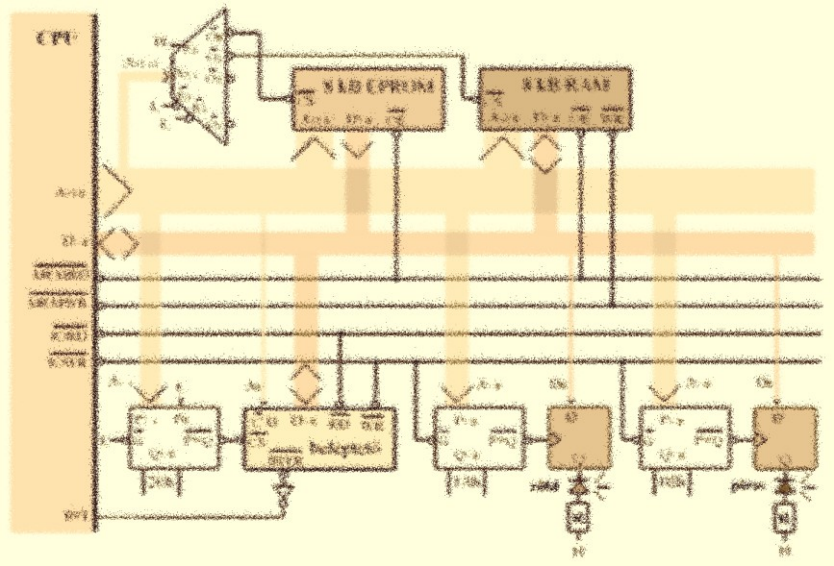

Dr. Lencse Gábor
tudományos főmunkatárs
BME Híradástechnikai Tanszék
lencse@hit.bme.hu

2012. szeptember 3.,
Budapest

# Tartalom

- Ismétlés
  - a beléptető periféria működése
  - a múlt héten elkészült hardver
- A szoftver feladat kitűzése
- A szoftver tervezésének menete (tervezési döntések)
- A hipotetikus processzor utasításkészlete
- Folyamatábrák rajzolása
- Assembly nyelvű program írása
- Alternatív megoldások vizsgálata
- Összefoglalás


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



# ISMÉTLÉS: A BELÉPTETŐ PERIFÉRIA MŰKÖDÉSE + A KÉSZ HARDVER


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A beléptető periféria működése – 1

- Minden kártyához (kártyaazonosítóhoz) tartozik egy 4 decimális jegyből álló kód: a zárat akkor kell nyitni, ha a kártya lehúzása után megnyomott első 4 billentyű éppen a hozzá tartozó kódot adja. (Időzítéssel nem foglalkozunk!) A feladat megoldásához felhasználjuk a beléptető periféria alábbi funkcióit:
  - Amikor a kártyát lehúzzák, a beléptető /INTR lába alacsony szintűre vált, és a státuszregiszteréből olvasva a $D_0$ bit 1-es értékű lesz (akárhányszor is kiolvasható): ezek mindaddig fennállnak, amíg az adatregiszterből ki nem olvasták a kártya azonosítóját. Ezután adatregiszterből kiolvasható (csak egyszer!) a lehúzott kártya azonosítója, utána rögtön /INTR magasra vált, és a státuszregiszteréből olvasva a $D_0$ bit értéke 0 lesz.

(folytatjuk…)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

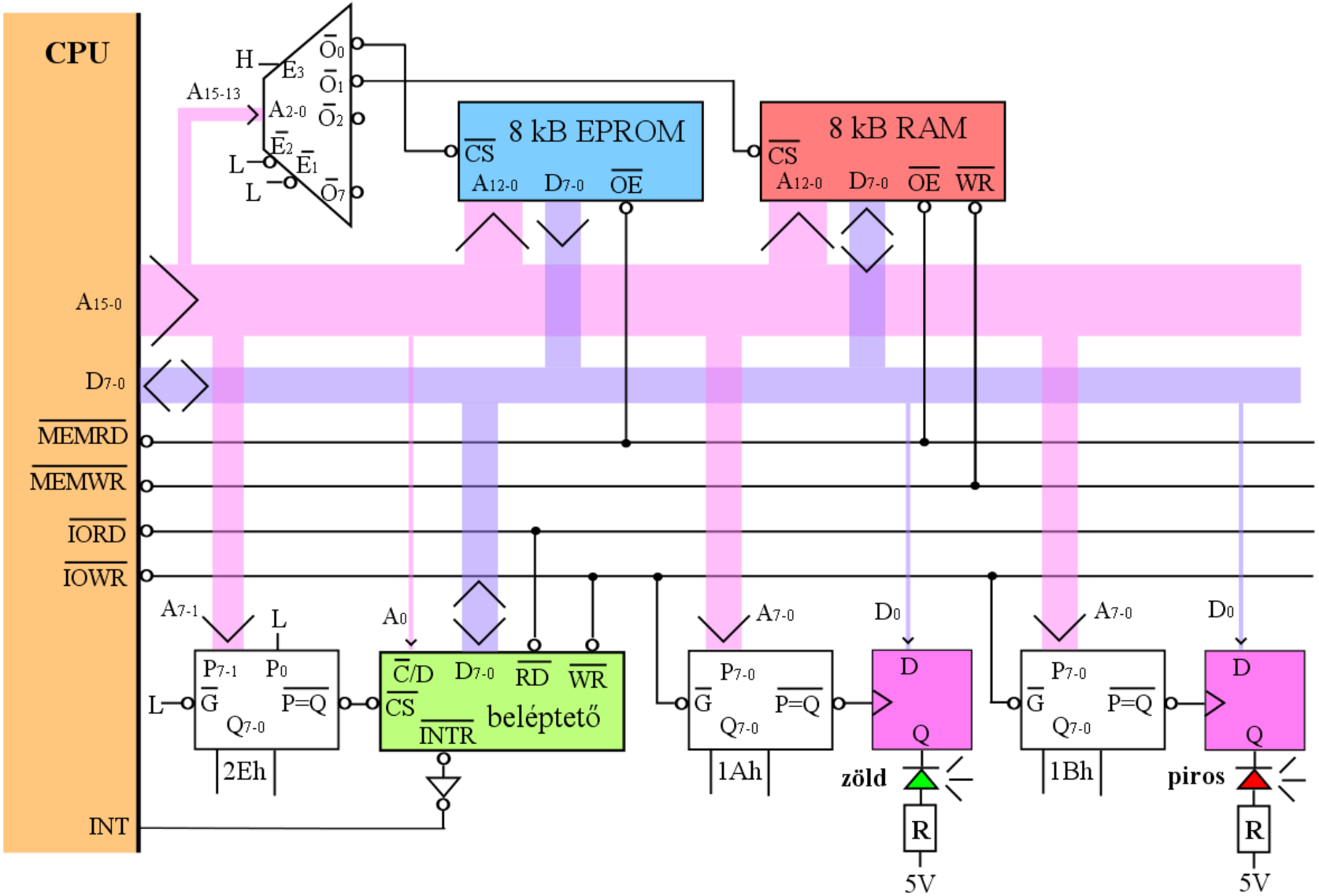
# A beléptető periféria működése – 2

(…folytatás)

- Amikor egy számjegyet beütöttek, a beléptető /INTR lába alacsony szintűre vált, és a státuszregiszteréből olvasva a $D_1$ bit 1-es értékű lesz (akárhányszor is kiolvasható): ezek mindaddig fennállnak, amíg az adatregiszterből ki nem olvasták a számjegy értékét. Ezután az adatregiszterből kiolvasható (csak egyszer!) a beütött számjegy értéke, utána rögtön /INTR magasra vált, és a státuszregiszteréből olvasva a $D_1$ bit értéke 0 lesz.
- A vezérlő regiszter $D_2$ bitjével állítható a zár állása: 0: zárás parancs, 1: nyitás parancs. A nyitás parancs után, amint az ajtót kinyitották rögtön, de legkésőbb 10s után (timeout: ha az ajtót addig nem nyitották ki) a beléptető zárnyelv vezérlője automatikusan átmegy zárt állapotba.
- A státuszregiszter $D_2$ bitjéből mindig kiolvasható a zár állása: 0: zárva, 1: nyitva

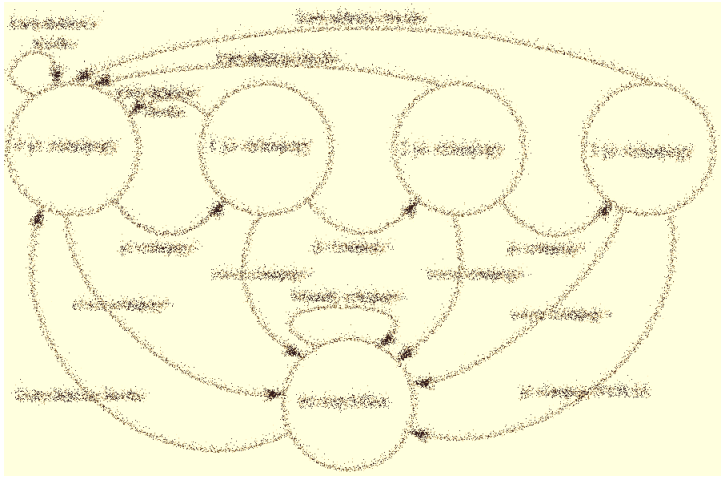

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A teljes logikai kapcsolási rajz





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



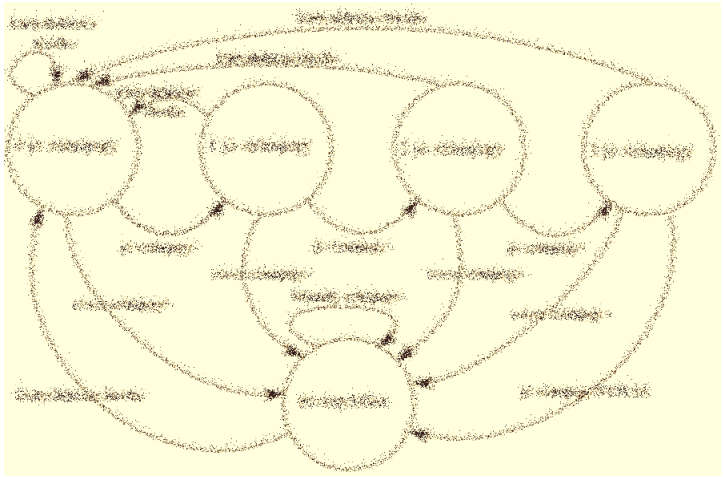
# A SZOFTVER FELADAT KITŰZÉSE


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A készítendő szoftver feladata

- A beléptető periféria zárja kezdetben zárt állásban van (így indul, ez nem a mi dolgunk).
- Követelmény, hogy:
  1. A LED-eknek mindenkor tükrözniük kell a zár állapotát (beleértve a nyitást, az automatikus zárást, valamit az ajtó fizikai nyitása által kiváltott zárást is).
  2. Kártyalehúzást követő helyes 4 jegyű kód megadása esetén a zárat nyitni kell (függetlenül attól, hogy éppen zárt vagy nyitott állásban van). Minden más bemeneti szekvencia esetén a zárat nem szabad kinyitni.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



# A szoftver tervezésének menete (tervezői döntések)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Előfeltevés, észrevétel

- Előfeltevés
  - Éljünk azzal az előfeltevéssel, hogy a kártyalehúzás és a számjegyleütés események közül egyszerre csak az egyik következik be! (A rendszer működése kellően gyors a felhasználóhoz képest.)
- Észrevétel
  - Mivel a zárásról semmiféle értesítést sem kapunk, az 1. követelményt csak úgy tudjuk kielégíteni, ha folytonosan kiolvassuk a zár állapotát és (szükség esetén)* beállítjuk a LED-eket.

* - pontosítandó, hogy csak <u>szükség esetén</u> vagy <u>mindig</u> (lásd a következő oldalon)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# További észrevételek

- Észrevételek:
  1) A zár állapotát lekérdező ciklusban a LED-ek állapotát
     a) minden lekérdezés után beállíthatjuk
     b) az előző állapotukat a programban nyilvántartva elegendő csak változás esetén állítanunk
  2) A zár állapotát lekérdező ciklusban a zár állapota alapján
     a) a LED-ek állapotát zártra és nyitottra is állíthatjuk (ekkor a kártyakód és a számjegyek alapján elegendő a zárnak nyitás parancsot adnunk és a LED-ekkel nem kell foglalkoznunk)
     b) a LED-ek állapotát elegendő csak zártra állítanunk, ha a zár nyitottra állításakor elvégezzük a LED-ek nyitottra állítását is
  3) A kártyalehúzás és a számjegyek leütésének figyelésére
     a) megszakítást használhatunk
     b) folytonos állapotolvasást végezhetünk


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Tervezői döntések (a felvetésekre)

- A következő önkényes tervezői döntéseket hoztuk
  1) A zár állapotát lekérdező ciklusban a LED-ek állapotát minden lekérdezés után beállítjuk.
     (Így az állapotukat nem kell nyilvántartanunk.)
  2) A zár állapotát lekérdező ciklusban a zár állapota alapján a LED-ek állapotát zártra és nyitottra is állítjuk.
     (Így csak itt állítunk LED-eket, a nyitás parancs kiadásakor a LED-ek állításával nem kell foglalkoznunk.)
  3) A kártyalehúzás és a számjegyek leütésének figyelésére megszakítást használunk.
     (Ezzel a döntéssel elestünk annak a lehetőségétől, hogy a programszámlálót használjuk a program állapotának a tárolására!)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# További tervezői döntések – 1

- Használjunk a program állapotának tárolására egy állapotváltozót, nevezzük AV-nek.
- Hordozza AV azt az értéket, hogy az utolsó kártyalehúzást követően hány helyes számjegy érkezett (lehetséges értékek 0-tól 4-ig) és egy másik számértékkel kódoljuk azt, ha nem volt még kártyalehúzás vagy érvénytelen számjegy jött.
- Könnyen belátható, hogy csak egy ilyen "érvénytelen" állapotra van szükség, ugyanis csak egyfajta esetben lehet folytatás: ha kártyalehúzás jön.

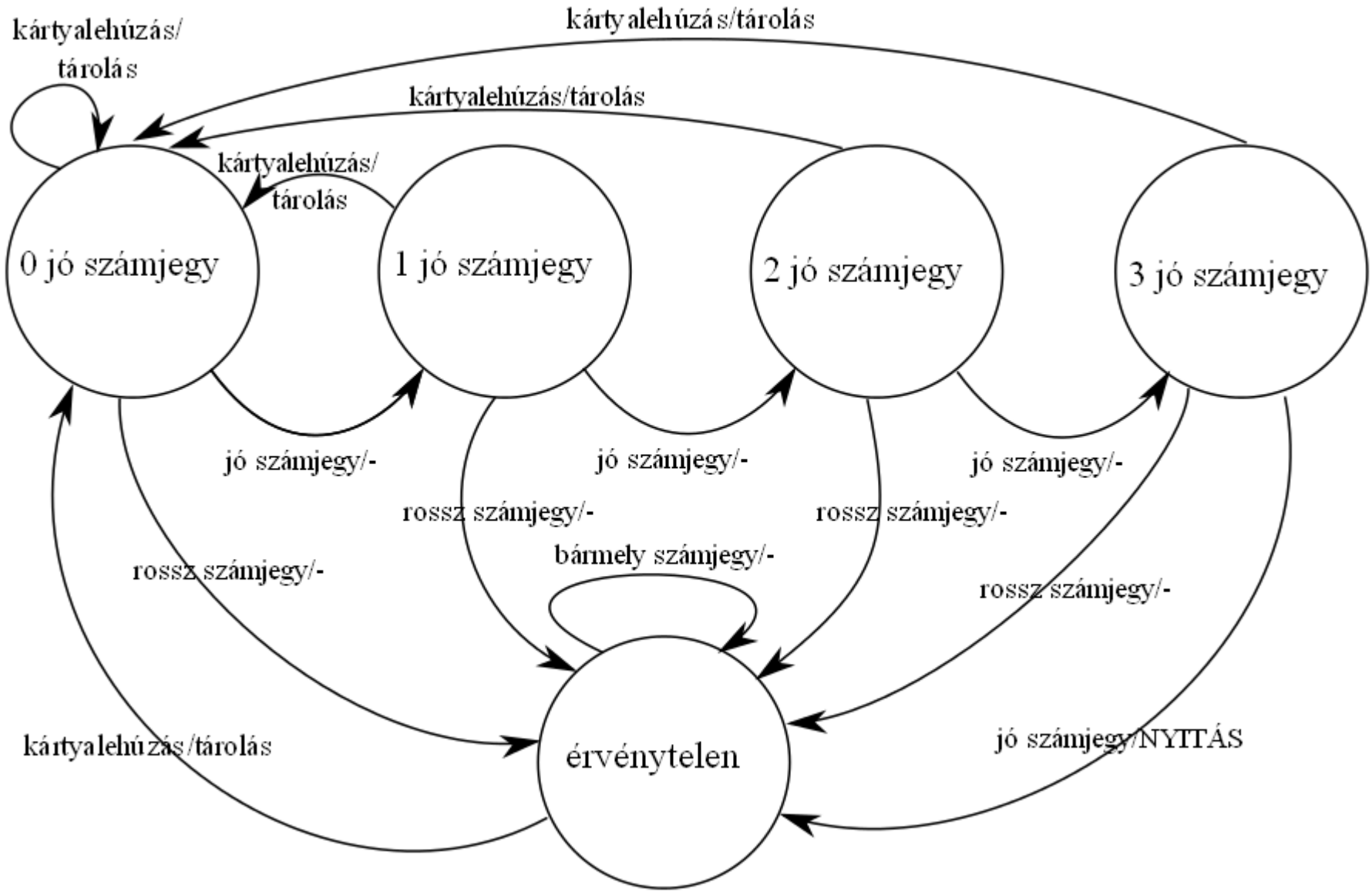

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# További tervezői döntések – 2

- Az is nyilvánvaló, hogy az AV=4 esetet sem szükséges megkülönböztetnünk az érvénytelentől, mert ha egyszer már elvégeztük a nyitást, a továbbiakban ez az állapot nem különbözik az érvénytelentől (ismét kártyalehúzásra kell várnunk).
- Tehát AV={0, 1, 2, 3, ERVTLN}
- Készítsük el az állapotgráfot!


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A beléptető rendszer állapotgráfja





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Hogyan tovább?

- Az állapotgráfnak megfelelő véges automatát formálisan is lekódolhatnánk, de helyette inkább gondolkozzunk egy kicsit, mert az megtérül a programozáskor! ;-)
- Az inputnak két fajtája lehetséges: kártyalehúzás vagy számjegy érkezése.
  - Látható, hogy a kártyalehúzás az adott állapottól függetlenül mindig AV:=0-t eredményez, és tárolni kell a kártya azonosítóját.
  - Csak számjegy érkezése esetén érdekes az AV értéke. Ekkor
    - ha az AV értéke ERVTLN, akkor nincs teendőnk
    - különben szükségképpen AV eleme {0, 1, 2, 3}, ekkor
      - ha a beérkezett számjegy megfelelő, akkor az AV értéke eggyel nő;
      - ha nem megfelelő, akkor AV:=ERVTLN lesz.

    ha ezek után AV értéke 4 lett, akkor NYITÁS parancsot adunk és AV:=ERVTLN.

# Adatszerkezetek

- KODTABLA: 4 db 256 byte-os táblázat egymás után: az i. táblázat j. karaktere a j. kártyaazonosítóhoz tartozó i. számjegy; j=0-255, i=0-3
- AV: egy darab státusz bájt, melynek jelentése:
  - i=0-3: a legutolsó kártyaolvasás után már jött i db számjegy és az jó volt
    (Vegyük észre, hogy itt i=0 éppen úgy kezelhető mint i=1-3!)
  - legyen az értéke 255 minden egyéb esetben (ERVTLN)
- K_AZON: egy bájt, ami az utolsó kártyaazonosító értékét tárolja, de csak AV=0-3 esetén érvényes


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Még egy tervezői döntés

- Igazítsuk a KODTABLA kezdetét 256 bájtos laphatárra!
  - Ez egyszerűsíti az elemeinek a címzését. A mindig használható KODTABLA_KEZDOCIME+256*AV+K_AZON cím 16 biten való kiszámítása helyett így, ha a KODTABLA AV-edik 256 bájtos lapján keressük a K_AZON-adik számjegyet, akkor elegendő AV-t a KODTABLA kezdőcímének magasabb helyiértékű bájtjához hozzáadni, K_AZON értékét pedig KODTABLA kezdőcímének alsó helyértékű (a laphatárra igazítás miatt 0 értékű) bájtja helyett használni.
  - Ez csak egy trükk, természetesen szükség esetén a 8 bites aritmetikából felépíthető a 16 bites aritmetika! ☺


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A főprogram működése

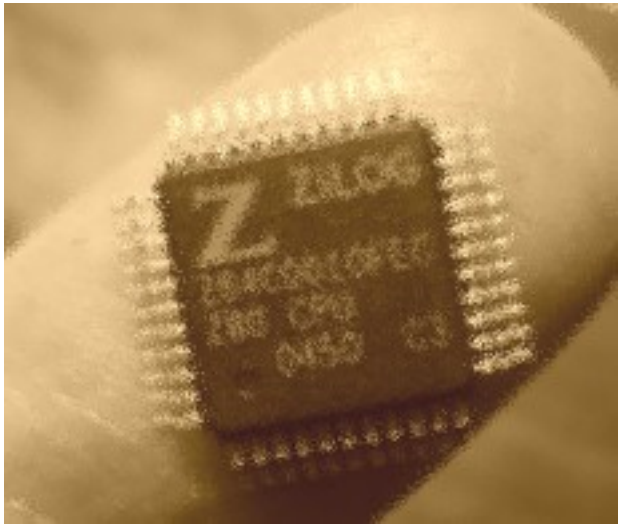
- A főprogram
  - beállítja a verem helyét
  - az állapotváltozó értékét
  - engedélyezi a megszakításokat
  - végtelen ciklusban
    - kiolvassa a zár állapotát
    - megjeleníti a LED-eken.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A megszakítási rutin működése

- Az IT rutin menti a használt regisztereket, majd:
  - Ha kártyalehúzás jött, akkor az azonosítót tárolja és AV:=0;
  - Különben számjegy jött (mert másért nem lett volna IT), tehát:
    - ha AV=255, akkor a számjegyet eldobjuk, állapot változatlan,
    - ellenkező esetben csakis 0-3 állapot lehet, így: ellenőrizzük, hogy jó-e a számjegy,
      - ha nem: AV:=255;
      - ha igen: AV++; amennyiben az állapot 4 lett, akkor: NYITÁS (csak a perifférián) és AV:=255;
- Végül az IT rutin visszatölti a mentett regiszterek értékét, engedélyezi a megszakítást és visszatér.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



# A HIPOTETIKUS PROCESSZOR UTASÍTÁSKÉSZLETE


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Regiszterkiosztás

- A hipotetikus processzornak a következő regiszterei vannak: A, F, B, C, D, E, H, L regiszterek 8 bitesek, és az SP regiszter 16 bites az alábbi értelmezésekkel:
  - A - akkumulátor: kitüntetett regiszter, az aritmetikai és logikai műveletek egyik operandusaként használjuk, és a műveletek eredménye is benne képződik; valamint az IN (perifériáról való bevitel) és OUT (perifériára való kivitel) utasítások implicit operandusa is ez.
  - F - jelzőbitek regisztere: a jelzőbitek közül csak a Z-t használjuk, értéke pontosan akkor 1, ha az utolsó aritmetikai vagy logikai művelet eredménye 0. Az A regiszterrel együtt az AF regiszterpárt alkotja, ami verem műveleteknél együtt kezelhető.
  - BC, DE, HL - regiszterpárok: együtt 16 bites regiszterként használhatók, de az egyes regiszterek külön-külön is használhatók.
  - SP - veremmutató


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Címzési módok – 1

- Akkumulátor címzés: az egyik operandus és a művelet eredménye implicit módon az akkumulátor (de nem nevezzük meg). Ezzel találkozunk az aritmetikai és logikai műveleteknél valamint az IN/OUT utasításoknál.

  AND B ; A ← A&B

  OUT 38h ; $port_{38h}$ ← A

- Regisztercímzés: operandusként regisztert vagy regiszterpárt adunk meg.

  LD A, B ; A ← B

- Közvetlen adatcímzés (immediate): közvetlenül az utasítás kódja után, az utasítás részeként szerepel az operandus

  LD B, 10 ; B ← 10


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Címzési módok – 2

- Direkt memóriacímzés: az operandus memóriabeli címét adjuk meg zárójelben (azért kell a zárójelpár, hogy a közvetlen adatcímzéstől meg tudjuk különböztetni)

  LD (3000h), D ; MEM[3000h] ← D

- Indirekt címzés: a címet tartalmazó regiszterpár zárójelben megadva szerepel (itt is azért kell a zárójelpár, hogy a regisztercímzéstől meg tudjuk különböztetni).

  LD A,(HL) ; A ← MEM[HL]


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Felhasznált mnemonikok – 1

- LD: adatmozgatás: hova, mit
- PUSH / POP: regiszter vagy regiszterpár értékének mentése a verembe / visszatöltése a veremből
- IN / OUT: a megadott portról bevitel vagy oda kivitel
- AND: bitenkénti logikai ÉS művelet a megadott operandus és az akkumulátor között
- ADD: 8 bites összeadás a megadott operandus és az akkumulátor között
- INC: a megadott regiszter értékének növelése 1-gyel
- CMP: a jelzőbitek állítása az
  "akkumulátor tartalma mínusz a megadott érték"
  művelet eredménye szerint
  (de az akkumulátor tartalma nem változik meg)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Felhasznált mnemonikok – 2

- JMP: feltétel nélküli ugrás (a megadott címre)
- JZ / JNZ: ugrás, ha az előző aritmetikai vagy logikai művelet eredménye 0 / nem 0
- CALL: szubrutin hívása
- RET: szubrutinból vagy megszakításból való visszatérés
- EI: megszakítások engedélyezése

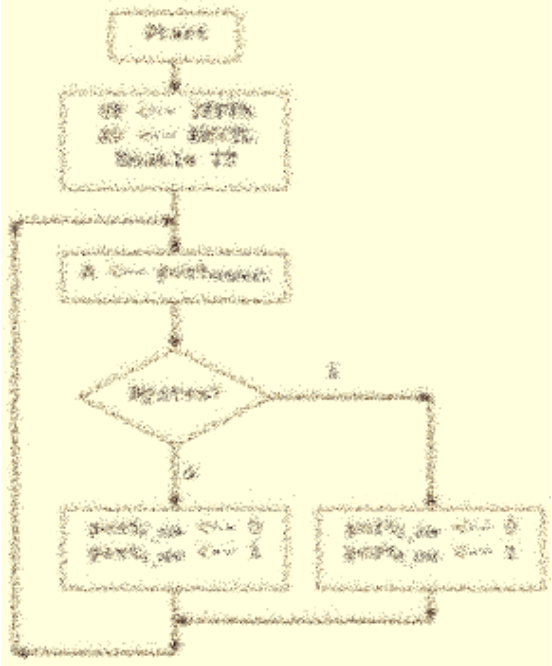

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Felhasznált fordítói direktívák

- EQU: szimbólumhoz érték rendelése
- ORG (origin): program memóriabeli kezdőcímének megadása
- $: az utasítás elhelyezési számláló aktuális értéke
- DB (define byte): adatok elhelyezése a gépi kódba az elhelyezés számláló értéke szerinti helyre
- END: a fordítás befejezése

- További elemek:
  - Címkék: sor elején kezdődnek, kettősponttal zárulnak
  - Megjegyzések: pontosvesszőtől (”;”) az adott sor végéig


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



# FOLYAMATÁBRÁK RAJZOLÁSA

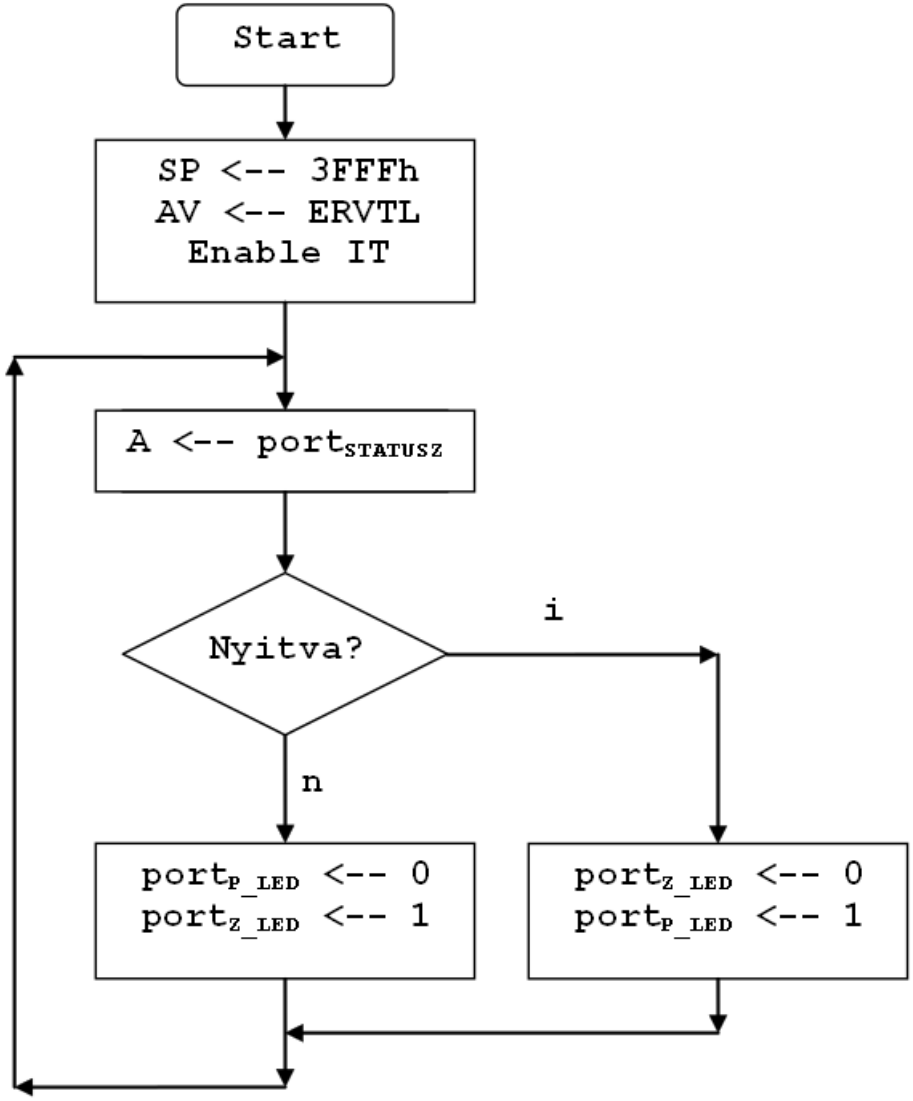

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Miért és hogyan használjuk?

- Rövid, egyszerű assembly nyelvű programok (néhányszor tíz sor) megírása előtt általában célszerű (és elegendő is) folyamatábrát készíteni.
- A folyamatábrába olyan lépéseket szoktunk írni, amit vagy közvetlenül egy gépi utasítással vagy legfeljebb 2-3 utasítással el tudunk végezni. Ezeket kifejezhetjük szavakkal, de még jobb, ha formálisan írjuk le őket (mert így egyértelmű); például az adatmozgatásokat regiszterekkel, memória/port címekkel és nyilakkal jelezzük:
  - A <-- (HL): Az A regiszterbe (akkumulátor) betöltjük a HL regiszterpár (értéke) által megcímzett memóriarekesz tartalmát.
  - $port_{5Bh}$ <-- A: Az akkumulátor értékét kiírjuk az 5Bh számú portra.
- A folyamatábra így természetesen processzorfüggő.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

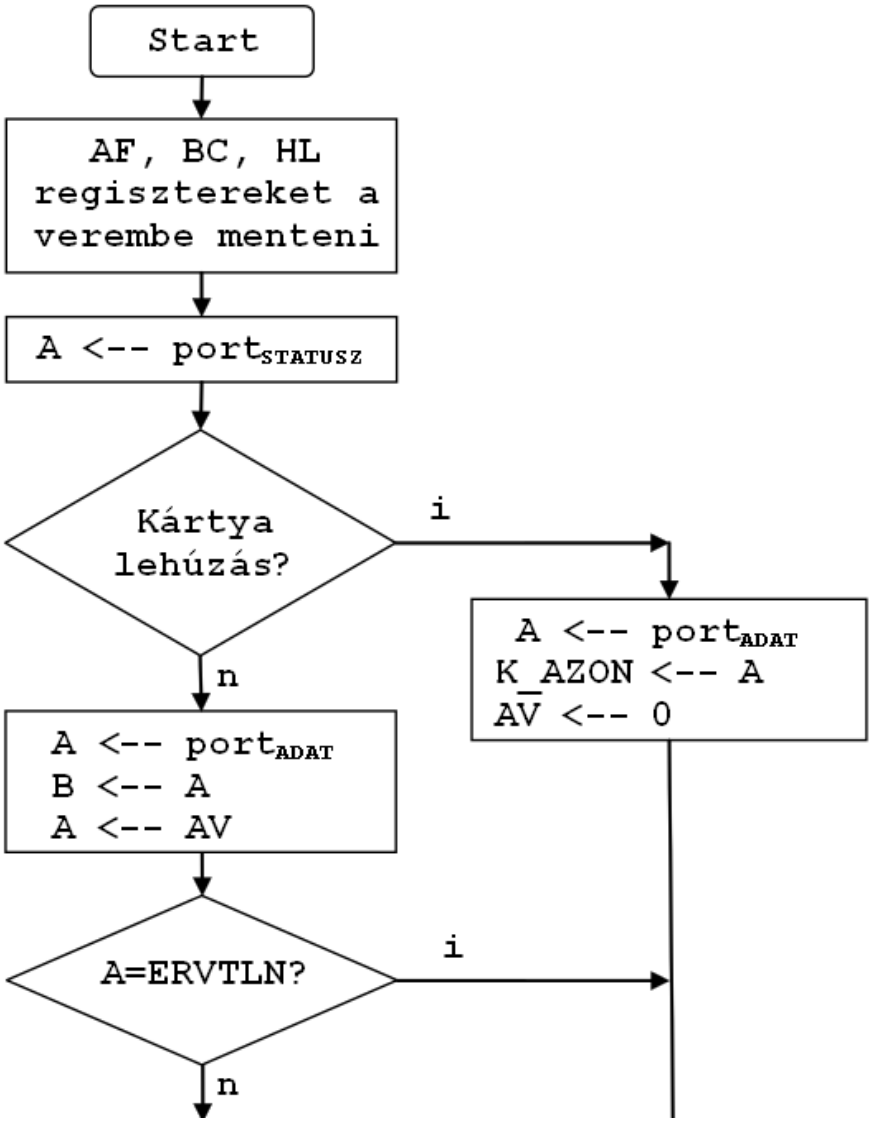
# A főprogram folyamatábrája





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

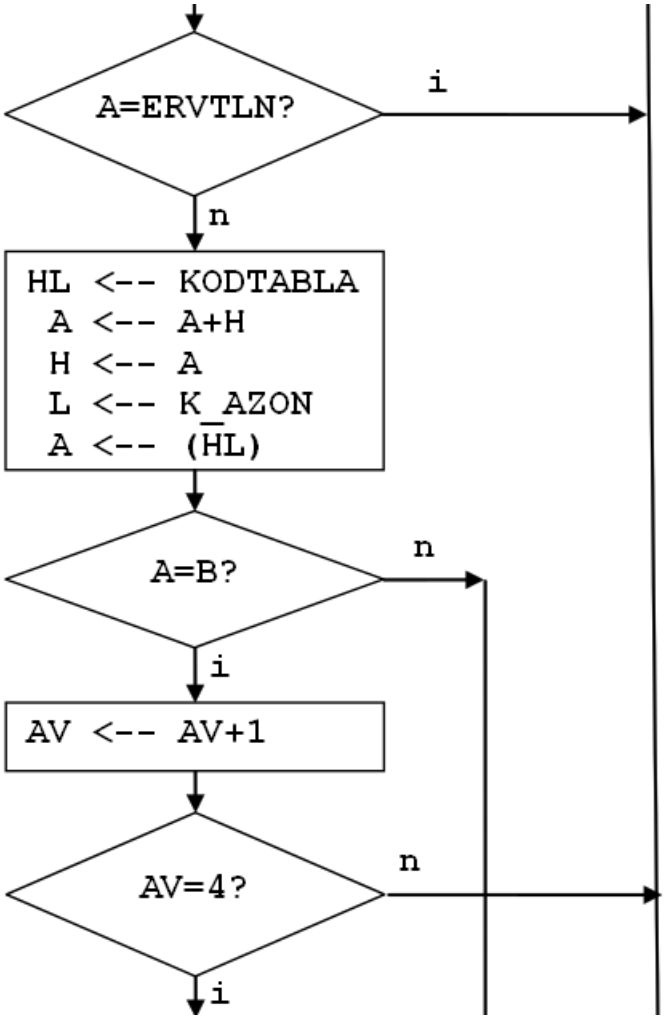
# Az IT rutin folyamatábrája – 1



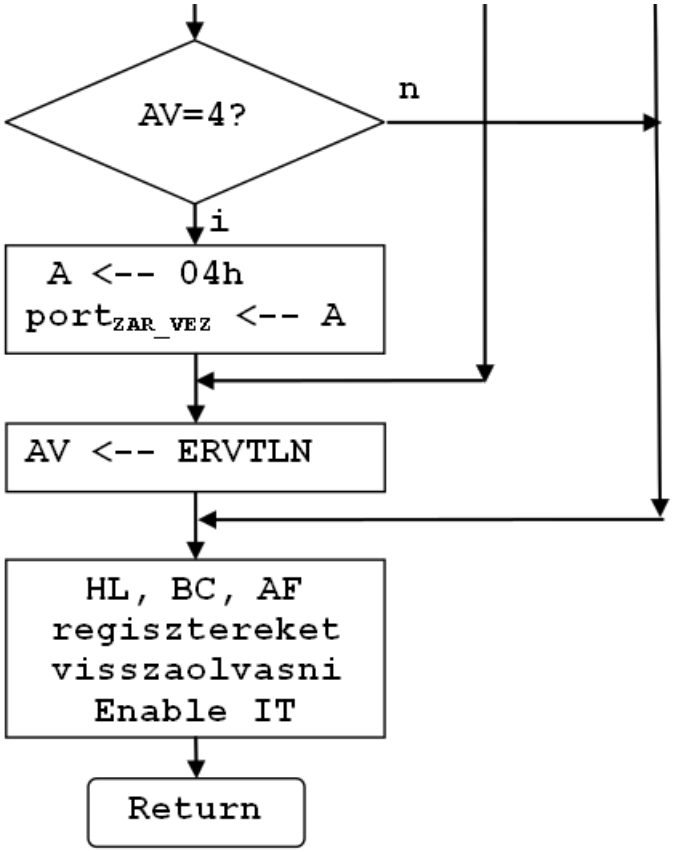


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

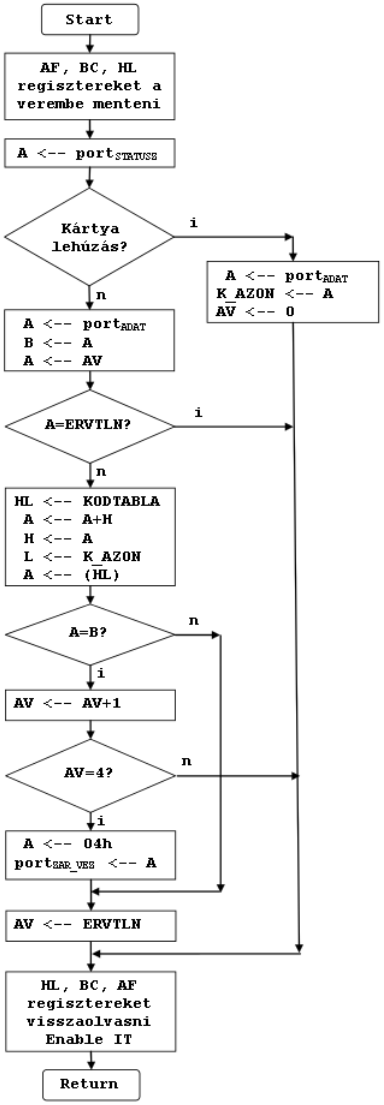
# Az IT rutin folyamatábrája – 2



# Az IT rutin folyamatábrája – 3



# Az IT rutin folyamatábrája egyben



# ASSEMBLY NYELVŰ PROGRAM ÍRÁSA


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Assembly nyelvű program – 1

```
; hardver címek
Z_LED    EQU     1Ah             ; zöld LED portcíme
P_LED    EQU     1Bh             ; piros LED portcíme
STATUSZ  EQU     2Eh             ; periféria státusz reg. portcíme
ZAR_VEZ  EQU     2Eh             ; periféria zárvez. reg. portcíme
ADAT     EQU     2Fh             ; periféria adat reg. portcíme
;
; egyéb konstansok
HOSSZ    EQU     04              ; belépőkártyákhoz kódjának hossza
KLE      EQU     00000001b       ; maszk kártyalehúzás teszteléséhez
SZJ      EQU     00000010b       ; maszk számjegy teszteléséhez
ZARALL   EQU     00000100b       ; maszk a zár állapot kiolv.-hoz
NYISD    EQU     00000100b       ; a nyitás parancs kódja
ERVTLN   EQU     255             ; az érvénytelen állapot kódja
LAP_MER  EQU     100h            ; 256 byte (trükkhöz)
;
```


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Assembly nyelvű program – 2

```
; változók címei
AV      EQU     2000h           ; állapotváltozót itt tároljuk
K_AZON  EQU     2001h           ; kártyaazonosítót itt tároljuk
; A kódtáblát majd a főprogram után az EPROM-ban helyezzük el...
;
; itt kezdődik a főprogram
        ORG     0000h           ; programkód elhelyezése a 0 címtől
        LD      SP,3FFFh        ; veremmutató beáll. a RAM tetejére
        LD      A,ERVTLN        ; az érvénytelen állapot kódja
        LD      (AV),A          ; állapotváltozó beállítása
        EI                      ; megszakítások engedélyezése
```


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Assembly nyelvű program – 3

```
; most jön a program főciklusa
FOCIKL: IN      STATUSZ         ; periféria állapotának kiolvasása
        AND     ZARALL          ; zárállapot vizsgálata
        JNZ     NYITVA          ; ha a D2 bit 1-es volt: nyitva van
; ha nem ugrott el, akkor tudjuk, hogy zárva van!
        LD      A,0             ; kigyújtás
        OUT     P_LED           ; piros LED világítson
        LD      A,1             ; kioltás
        OUT     Z_LED           ; zöld LED ne világítson
        JMP     FOCIKL          ; végtelen ciklus
NYITVA: LD      A,0             ; kigyújtás
        OUT     Z_LED           ; zöld LED világítson
        LD      A,1             ; kioltás
        OUT     P_LED           ; piros LED ne világítson
        JMP     FOCIKL          ; végtelen ciklus
;
```

# Assembly nyelvű program – 4

```
; A kódtáblát laphatárra helyezzük az alábbi apró trükkel:
; (a fenti programkód rövidsége miatt az alábbi sor nyilván
;  "ORG 0100h"-val ekvivalens)
        ORG     ($+LAP_MER-1) AND NOT (LAP_MER-1)
KODTABLA: DB    1, 4, 5, 2, 3, 4, 7, 8, 8, 0, 2, 3, 6, 2, 4, 9
        DB      4, 4, 3, 1, 7, 8, 0, 1, 3, 6, 4, 1, 4, 7, 6, 5
        DB      ...
; összesen 64 sor, soronként 16 értékkel,
; azaz 64x16=1024=4x256 db számjegy
;
```

# Assembly nyelvű program – 5

```
; most jön a megszakítási rutin
        ORG     1000h           ; az IT rutin elhelyezése
; minden olyan regiszter értékét elmentjük, amit használni fogunk
        PUSH    AF              ; az A regiszter és a flag-ek,
        PUSH    BC              ; a BC regiszterpár,
        PUSH    HL              ; a HL regiszterpár mentése
        IN      STATUSZ         ; mi volt a megszakítás oka?
        AND     KLE             ; kártyalehúzás jött?
        JNZ     LEHUZ           ; igen
; biztos, hogy számjegy volt, nem mert kártyalehúzás!
        IN      ADAT            ; számjegy kiolvasása (IT törlése)
        LD      B,A             ; a számjegy mentése
        LD      A,(AV)          ; állapotváltozó betöltése
        CMP     ERVTLN          ;
        JZ      IT_VEGE         ; ha érvénytelen, akkor kész
```

# Assembly nyelvű program – 6

```
; az AV értéke szükségképpen a [0,HOSSZ-1] intervallumba esik!
; a várt számjegy a KODTABLA+LAP_MER*AV+K_AZON címen van:
        LD      HL,KODTABLA     ; kódtábla kezdőcíme, (L=0!)
        ADD     H               ; A:=A+H
        LD      H,A             ; A KODTABLA-ban az (AV). lapon ...
        LD      L,(K_AZON)      ; ... keressük a (K_AZON). értéket
        LD      A,(HL)          ; a várt számjegy kiolvasása
        CMP     B               ; B-ben van a beérkezett számjegy
        JNZ     ROSSZJ          ; hibás a beütött számjegy
; a beütött számjegy helyes volt, ezért AV 1-gyel nő:
        LD      A,(AV)
        INC     A
        LD      (AV),A
        CMP     HOSSZ           ; elértük már a kódsorozat hosszát?
        JNZ     IT_VEGE         ; még nem
```


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Assembly nyelvű program – 7

```
; ha mar elértük: nyitunk + AV:=ERVTLN
NYITAS: LD      A,NYISD         ; nyitás parancs kódja
        OUT     ZARVEZ          ; parancs kiadása
ROSSZJ: LD      A,ERVTLN        ; az új állapot: érvénytelen
        LD      (AV),A          ; állapot eltárolása
        JMP     IT_VEGE         ; ugrás az IT rutin végére
; kártyalehúzás feldolgozása
LEHUZ:  IN      ADAT            ; belépőkártya kódjának kiolvasása
        LD      (K_AZON),A      ; kártyakód eltárolása
        LD      A,0             ; ez lesz az új állapot
        LD      (AV),A          ; állapotváltozó beállítása
; a regisztereket fordított sorrendben kell visszatölteni!
IT_VEGE: POP    HL
        POP     BC
        POP     AF
        EI
; Az IT engedélyezésének hatása egy utasítást késik!
        RET                     ; visszatérés az IT rutinból
        END                     ; vége a fordításnak
```


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# ALTERNATÍV MEGOLDÁSOK VIZSGÁLATA


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Önálló hallgatói munka – 1

- A hallgatók önállóan gondolják végig, hogy miben különbözne a program, ha nem használnánk megszakítást, hanem helyette állapotolvasással kérdeznénk le, hogy történt-e kártyalehúzás, illetve érkezett-e számjegy!
- Módosítsák ennek megfelelően a programot önállóan!

FIGYELEM! Először mindenki készítse el a saját megoldását és csak utána nézze meg a tárgy jegyzetében szereplő minta megoldást!


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Önálló hallgatói munka – 2

- Gondolkozzanak el önállóan az alábbi kérdéseken!
  1. A megszakításos és az állapotolvasásos megoldás közül melyiket biztosan nem lehet megvalósítani RAM nélkül? Miért?
  2. Hogyan tudná a másikat megvalósítani RAM nélkül? (Mit, hova helyezne el?)
  3. Hogyan módosítaná a programot, ha a KODTABLA nem illeszkedne laphatárra? (A kritikus számításhoz használjon 8 bites műveleteket, a mnemonikokat értelemszerűen állapítsa meg!)
  4. Hogyan módosulna a fenti számítás akkor, ha a KODTABLA (mint kétdimenziós tömb) indexeit felcserélnénk, azaz a 4db 256 bájt méretű tömb helyett 256 db 4 bájtos tömb lenne
  5. Hogyan valósíthatnánk meg a megalkotott állapotgráf szisztematikus lekódolását, és ez mekkora munkát jelentene!

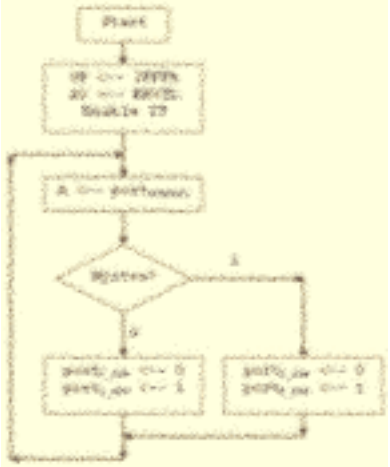
# Önálló hallgatói munka – 3

6. Hogyan tudná elkerülni a főprogramban az elágazás használatát, ha felhasználhatja még a XOR (bitenkénti kizáró vagy) utasítást, valamint:
   a) az SHR (bitenkénti eltolás jobbra) utasítást?
   b) módosíthatja a beléptető periféria státusz regiszterében a bitek specifikációját?
   c) módosíthatja a hardver specifikációját?


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Összefoglalás

- Megterveztük a beléptető szoftverét (tervezési döntések)
- Megismertük hipotetikus processzor utasításkészletét
- Folyamatábrákat rajzoltunk
- Megírtuk az assembly nyelvű programot
- Alternatív megoldásokat is megvizsgáltunk

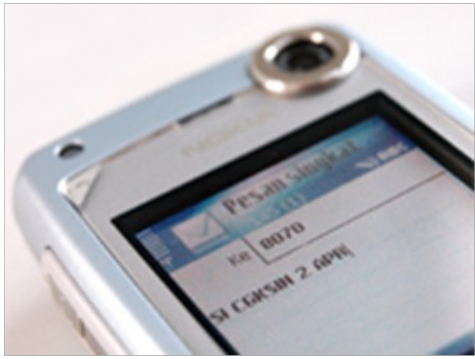


# Kérdések?

## KÖSZÖNÖM A FIGYELMET!

Híradástechnikai Tanszék

Dr. Lencse Gábor
tudományos főmunkatárs
BME Híradástechnikai Tanszék
lencse@hit.bme.hu



# Számítógép Architektúrák

*Az ajtónyitó megvalósítása Arduino-val*

Horváth Gábor
docens
BME Híradástechnikai Tanszék
ghorvath@hit.bme.hu

2012. szeptember 10.
Budapest

# Tartalom

- A gyakorlat célja:
  - Megmutatni, hogy a manapság elterjedt és olcsó mikrokontrollerekkel milyen könnyű dolgozni
- Tartalomjegyzék:
  - Az Arduino bemutatása
    - Hardver
    - Programozás
    - Szenzorok
  - Az ajtónyitó megvalósítása
    - A kijelző
    - Az RFID olvasó
    - A billentyűzet
    - A teljes hardver
    - A teljes kód

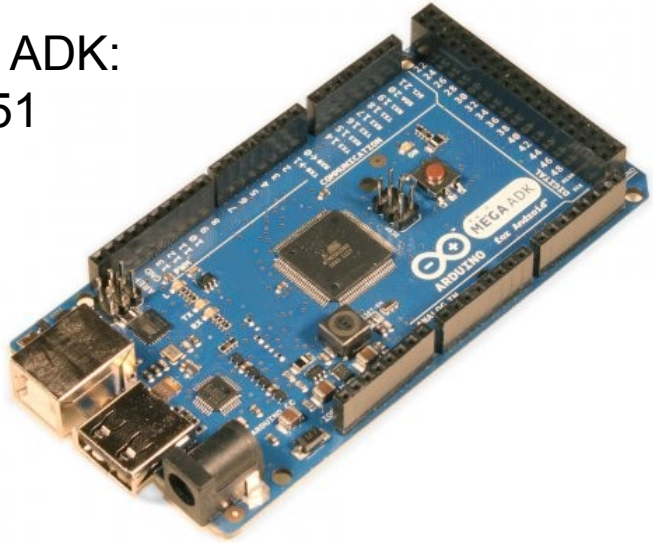
# Arduino

- 2005, Ivrea, Olaszország
- Célja: egyszerű prototípusfejlesztés
- Valójában egy teljes család (http://arduino.cc)
- Ami közös bennük:
  - Atmel AVR 8 bites processzor (**Harvard** architektúra!)
  - Beépített flash memória a programnak
  - Beépített EEPROM tartós adattárolásra
  - Beépített RAM
  - Ki/bemenetek:
    - Digitális
    - Analóg
  - USB port-on át programozható (nem mindegyik)
  - C++-szerű nyelven programozható

# Az Arduino család

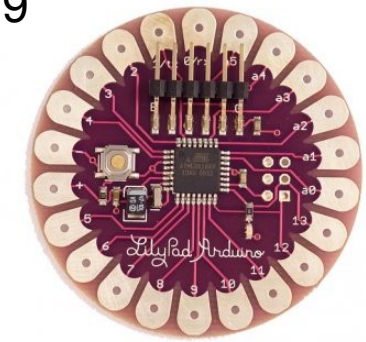
Leonardo:
€18

Mega ADK:
€51




Mini:
€15

LilyPad:
€19



# Az Arduino család

- Eltérések:
  - Bemenetek/kimenetek száma
  - Memória mérete (flash/EEPROM/RAM)
  - Néhány jellegzetesség:
    - A Mega ADK-t hozzá lehet kötni az Android eszközökhöz
    - A LilyPad ruhára varrható

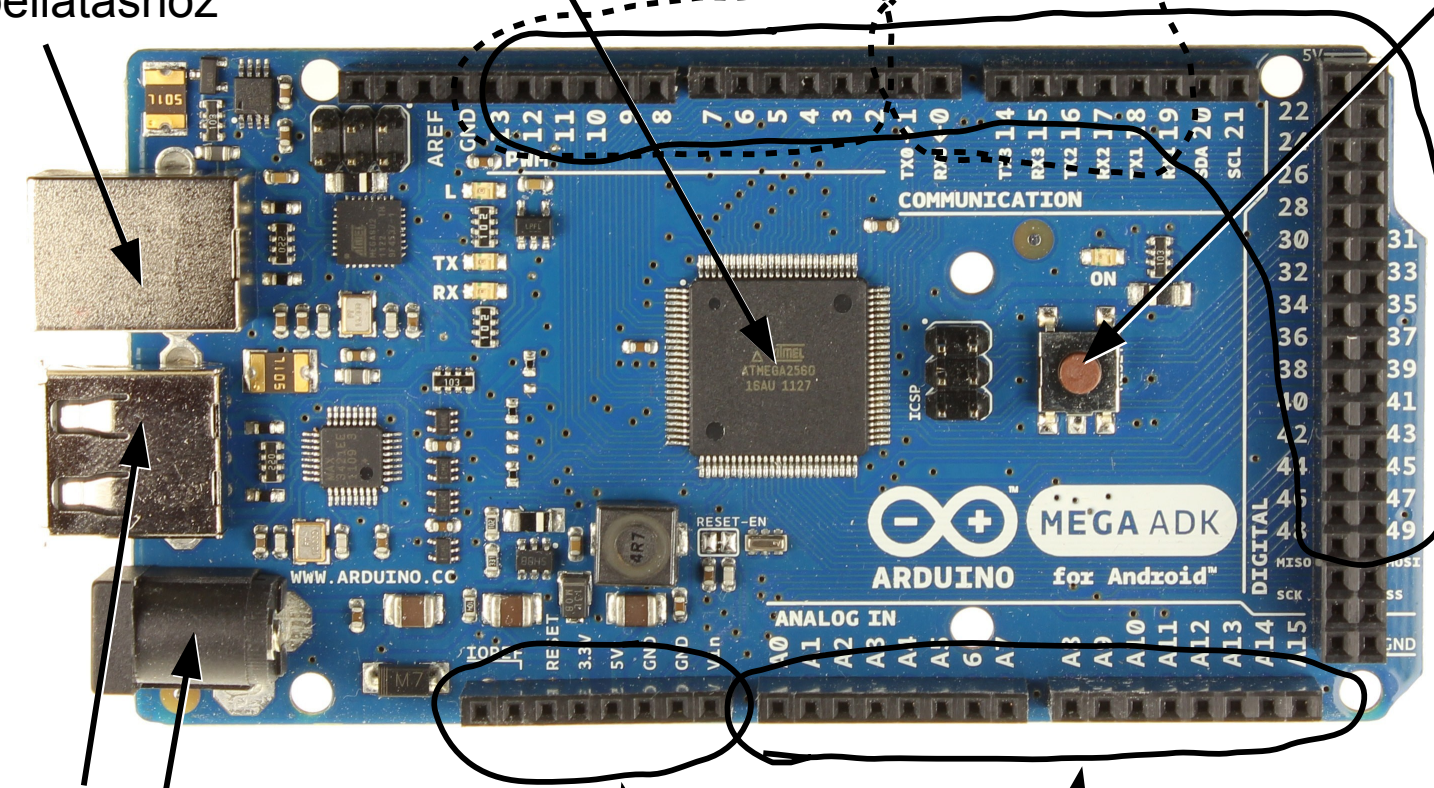
| | CPU órajel | Flash | RAM | EEPROM | Digitális I/O | Analóg I/O |
|---|---|---|---|---|---|---|
| Leonardo | 16 MHz | 32 kB | 2.5 kB | 1 kB | 20 | 12/7 |
| Mega ADK | 16 MHz | 256 kB | 8 kB | 4 kB | 54 | 16/15 |
| Mini | 16 MHz | 32 kB | 2 kB | 1 kB | 14 | 8/6 |
| LilyPad | 8 MHz | 16 kB | 1 kB | 512 byte | 14 | 6/6 |

# Átverés!

- Micsoda??
  - €51, 8 bites CPU-ért, 16 MHz-ért, pár kB memóriáért?
  - Amikor ott a Raspberry Pi 35$-ért?

    (700 MHz 32 bites ARM, 256 kB RAM, erős GPU, HD filmlejátszás, HDMI kimenet, Ethernet port, stb.)
- Más a célja:
  - Arduino:
    - Hangsúly: I/O, minden mennyiségben, pofonegyszerűen
    - Bekapcsolom, és 1 másodperc múlva már megy is
    - Az €51 a legnagyobb modell, az olcsó Leonardo a legtöbb dologra elég
  - Raspberry Pi:
    - Hangsúly: általános célú (programozás oktatás)
    - Op. rendszer (linux) kell rá! Sokáig tart, mire elindul.
    - Van programozható periféria ki/bemenete, de nehéz használni (érteni kell a linux kernelhez)

# Arduino Mega ADK



# *Programozás*

# Programozás

- Open-source cross-platform fejlesztői környezet
  (http://arduino.cc/en/Main/Software)
- Programnyelv: C++-hoz hasonló (kiterjesztés: .ino)
- Fordítás:
  - A fejlesztőkörnyezet AVR kódot fordít (cross compiler)
  - USB-n keresztül feltölti a mikrokontroller flash memóriájába
- Hibakeresés:
  - Amit az Arduino a default soros portjára ír, azt USB-n keresztül megmutatja a fejlesztőkörnyezet

# A fejlesztőkörnyezet

Szintaktikai ellenőrzés

Feltöltés az Arduino-ra

Új fájl

Fájl megnyitás

Fájl mentés

Soros port monitor

ajtonyito | Arduino 1.0.1

File Edit Sketch Tools Help

ajtonyito

```
#include <LiquidCrystal.h>
#include <Key.h>
#include <Keypad.h>

int RFIDResetPin = 2;
int LEDPin = 13;

char userID[17];
int index = 0;
enum estate { start, rfid, code};
estate state = start;

LiquidCrystal lcd(49, 47, 48, 46, 44, 42);

const byte ROWS = 4;
const byte COLS = 3;

char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
```

Done uploading.

```
Binary sketch size: 7,314 bytes (of a 258,048 byte maximum)
```

4 Arduino Mega 2560 or Mega ADK on /dev/ttyACM0

# A programozási nyelv

- Az Arduino a programot „sketch”-nek hívja
- Adattípusok:
  - int: 16 bites egész
  - long: 32 bites egész
  - boolean: logikai típus (true/false), 1 bitet foglal (ilyen C++-ban nincs)
  - float: 32 bites lebegőpontos
  - char: ASCII karakter tárolására szolgál (1 byte)
  - stb.
- Operátorok, ciklusok, elágazások: mint a C++
- class-ok használata megengedett, preprocessor direktívákat elfogad
- Két függvény van, amit kötelező megírni:
  - **void setup () { … }** - az ide írt kód induláskor fut le, egyetlenegyszer
  - **void loop () { … }** - inicializálás után ez a függvény fut le újra és újra (ha befejeződik, újraindul a függvény elejétől)

# *Ki/Bemenetek*

# Digitális ki- és bemenetek

- A digitális lábak ki és bemenetként is szolgálhatnak egyaránt
  - De egyszerre csak az egyik szerepet tölthetik be
  - Beállítás:
    - **pinMode (4, INPUT);** – a 4-es lábat bemenetre állítja
    - **pinMode (5, OUTPUT);** – az 5-ös lábat kimenetre állítja
- A digitális lábak magasba, ill. alacsonyba húzása:
  - **digitalWrite (5, HIGH);** – az 5-ös lábra logikai 1-et tesz (5V)
  - **digitalWrite (5, LOW);** – az 5-ös lábra logikai 0-át tesz (0V)
- A digitális lábak olvasása:
  - **int val;**
  - **val=digitalRead (4);** – a 4-es láb logikai értékének olvasása
  - **if (val==HIGH) …**, vagy **if (val==LOW) …**

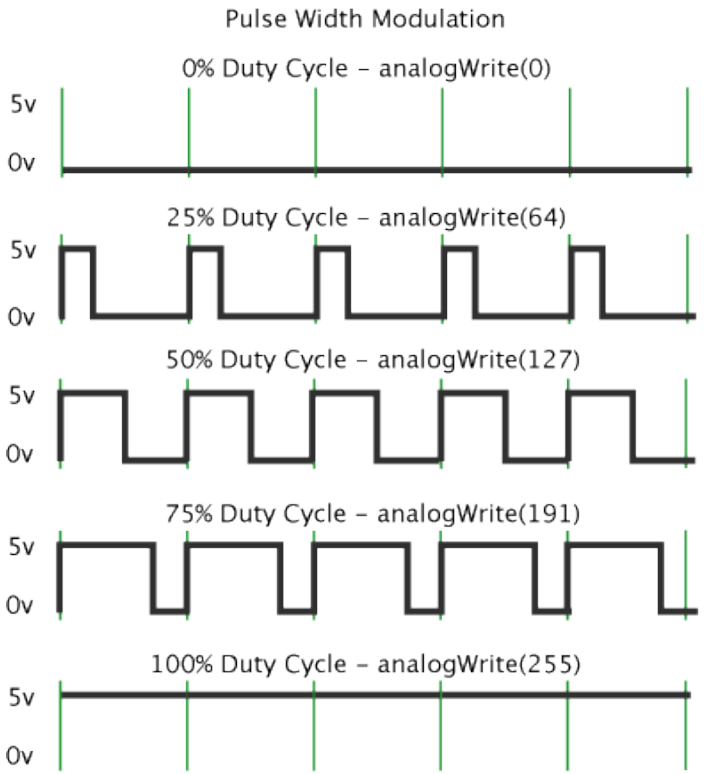
# Analóg bemenetek

- Nemcsak HIGH és LOW értékeket lehet róla olvasni
- ADC (Analog-Digital Converter) 10 bites felbontással
  → 1024 különböző feszültségszintet tud megkülönböztetni
- 0V és 5V között 1024 szint → 4.88 mV különbséget tud tenni
- Használata:
  - **int val;**
  - **val = analogRead (3);** - a 3-mas analóg bemenetről mintát vesz
  - **val** értéke: 0-tól (0V esetén) 1023-ig (5V)
- Az 5V maximum referencia változtatható az **analogReference()** függvénnyel, de 5V a maximum

# PWM kimenetek

- Folyamatos analóg kimenet (pl. 4.2V) nem tud előállítani
- Viszont a kimenet gyors ki-be kapcsolgatásával „átlagos értelemben” tetszőleges feszültséget elő tud állítani

  → **PWM: Impulzusszélesség-moduláció**
- Erre szolgál az **analogWrite ()** függvény
- Paraméterek: láb száma, kitöltöttség (Duty Cycle, 0…255 között)



# Soros ki- és bemenetek

- A kommunikációhoz max. 2 ér kell:
  - RX: a vételhez (ha szükség van vételre)
  - TX: az adáshoz (ha szükség van adásra)
- Az Arduino ADK 4 soros portot tartalmaz
  - Az elsőt használja Debug célra a fejlesztőkörnyezet USB-n keresztül
- Használata: **class Serial**-on keresztül
  - Előregyártott példányok: **Serial**, **Serial1**, **Serial2**, **Serial3**
  - *Inicializálás*: **Serial1.begin (9600);** - port megnyitása 9600 bps sebességre
  - *Írás*: **Serial1.write (…);** - elküld 1 bájtot, egy NULL terminált string-et, vagy egy tömböt
  - *Írás*: **Serial1.print (...);** - a paraméterét string-é kovertálja (ha nem az), és elküldi. A **Serial1.println (...);** még egy soremelést is utánatesz.
  - *Olvasás*: **int kaptam=Serial1.read();** - egy megérkezett bájt kiolvasása (-1, ha nincs)
  - *Érkezett-e adat*: **int mennyi = Serial1.available();** - megadja, hány bájt érkezett
  - *Lezárás*: **Serial1.end ();** - a port lezárása után a megfelelő láb digitális ki- és bemenetként használható tovább

# Soros ki- és bemenetek

- Működése:
  - Vételkor Interrupt keletkezik
  - Az Interrupt-ot lekezeli az Arduino:
    - Beleteszi az érkezett bájtot egy tárolóba (buffer)
  - A read() tulajdonképpen a tárolóból olvas
  - Az available() a tárolóban lévő bájtok számát adja vissza

# *Memóriák*

# Memóriák

- Az Arduino az alábbi memóriákkal rendelkezik:
  - **Flash memória**: ez tárolja a programot
    - Tápfeszültség nélkül is megmarad a tartalma
  - **RAM**: ebben vannak a változók és a stack is
    - Tápfeszültség kell a tartalom megőrzéséhez
  - **EEPROM**: tartósnak szánt adatokhoz
    - Tápfeszültség nélkül is megmarad a tartalma
- Az AVR processzorok Harvard architektúrát követnek
  - 2 címtér:
    - Az utasításokat és konstansokat a flash memóriából veszi
    - A változókhoz/stack-hez a RAM-ot használja
  - És az EEPROM?
    - Ahhoz külön függvénykönyvtár van

# A flash és a RAM

- Flash memória:
  - A fejlesztőkörnyezet tölti meg
  - A programból nem írható. De olvasható! Speciális függvényekkel.
- RAM:
  - Kevés van belőle, ezért spórolni kell
  - Nem változó tömböket inkább a flash-be írjuk:

```
int kodok[] = {21342, 2442, -34887, ...};
char szoveg[] = „megszentségteleníthetetlenségeiteknek”;
k = kodok[4];
c = szoveg[7];
```

    - Helyett:

```
PROGMEM prog_uint16_t kodok[] = {21342, 2442, -34887, ...};
PROGMEM prog_uchar szoveg[]  = „megszentségteleníthetetl...”;
k = pgm_read_word_near (kodok + 4);
c = pgm_read_byte_near (szoveg + 7);
```

# EEPROM

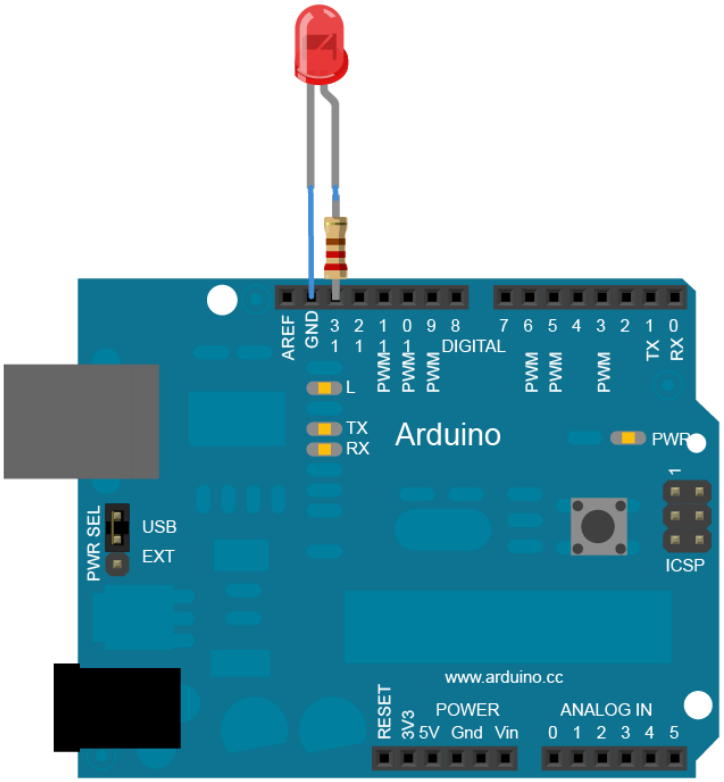
- A rendszer része az EEPROM könyvtár
- #include <EEPROM.h>
- Írása:
  - **EEPROM.write (cím, adat);**
    - A cím int típusú
    - Az adat byte típusú
- Olvasása:
  - **byte a;**
    **a = EEPROM.read (42);**
    - Kiolvassa a 42-edik byte-ot az EEPROM-ból

# *Perifériák illesztése*

# Digitális kimeneti perifériák

- **Példa**:
  LED villogtatás (Nem Arduino ADK, de ugyanaz az elv)
  Alkatrész: LED, 220 Ohmos ellenállás

  LED nélkül is megy!
  A 13-mas lábon van beépített LED.

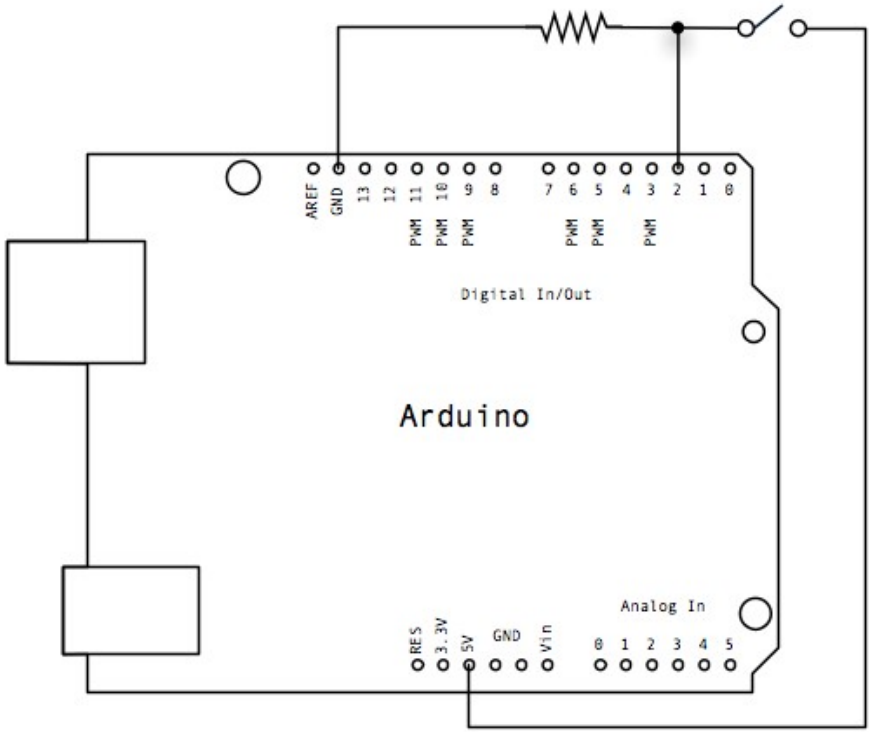



- **Kód**:

```
int ledPin = 13;
void setup () {
    pinMode (ledPin, OUTPUT);
}
void loop () {
    digitalWrite (ledPin, HIGH);
    delay (1000);
    digitalWrite (ledPin, LOW);
    delay (1000);
}
```

# Digitális bemeneti perifériák

- **Példa**:
  Nyomógomb állapotának lekérdezése, a beépített LED kigyújtása

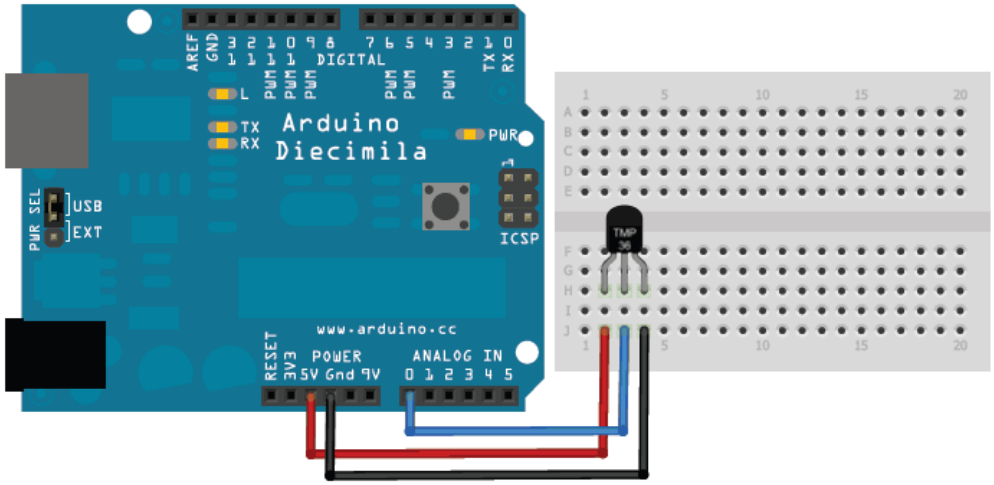



- **Kód**:

```
const int buttonPin = 2;
const int ledPin =  13;
int buttonState = 0;
void setup () {
    pinMode (ledPin, OUTPUT);
    pinMode (buttonPin, INPUT);
}
void loop () {
    buttonState = digitalRead (buttonPin);
    digitalWrite (ledPin, buttonState);
}
```

# Analóg bemeneti perifériák

- **Példa**:
  TMP36 Hőmérséklet szenzor illesztése




- **Kód**:

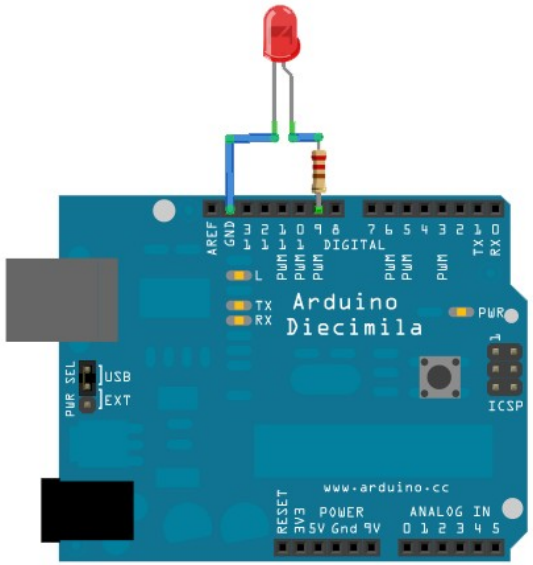
```
int sensorPin = 0;
void setup () {
    Serial.begin (9600);
}
void loop () {
    int reading = analogRead (sensorPin);
    float voltage = reading * 5.0 / 1024.0;
    float temperature = (voltage - 0.5) * 100;
    Serial.println (temperature);
    delay(1000);
}
```

# Analóg bemeneti perifériák

- A legérdekesebb perifériák az analóg bemeneti perifériák:
  - Filléres szenzorok tömkelege:
    - 3 tengelyes gyorsulásmérő (3 analóg bemenetet használ)
    - Alkohol szenzor
    - Szénmonoxid szenzor
    - Szálló por koncentráció szenzor
    - Elhajlásmérő szenzor
    - Erő (súly) mérő szenzor
    - Vibráció szenzor
    - Giroszkóp (2 tengelyes → 2 analóg bemenet kell)
    - Távolságmérő (infra fénnyel vagy ultrahanggal)
    - Hőmérő
    - Páratartalommérő
    - Stb.
- A mért fizikai mennyiséget analóg jellé alakítják

# PWM kimeneti perifériák

- **Példa**:
  LED halványítás/fényesítés




- **Kód**:

```
int ledPin = 9;
void setup ()  {
}
void loop ()  {
    for (int fadeValue = 0 ; fadeValue <= 255; fadeValue +=5) {
        analogWrite (ledPin, fadeValue);
        delay (30);
    }
    for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -=5) {
        analogWrite (ledPin, fadeValue);
        delay (30);
    }
}
```

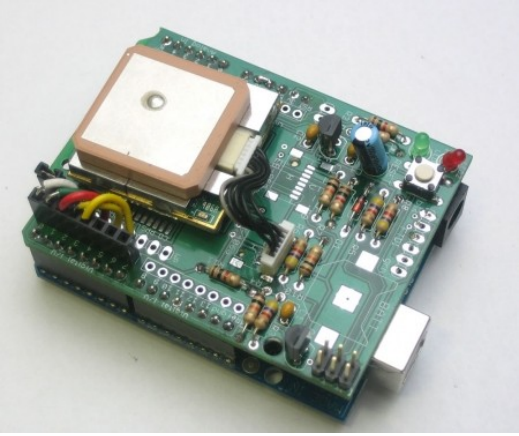
# Soros perifériák

- Példák soros portra csatlakozó perifériákra:
  - RFID modul: soros porton küldi a leolvasott kártya kódját
  - GPS modul: soros porton küldi rendszeresen a koordinátákat
  - GSM/GPRS modul: soros porton kell vezérelni, illetve az átviendő adatok is a soros porton mennek
  - Stb.

# Pajzsok (Shield)

- Az Arduino tetejére illeszthető komplett perifériák
  - Nagyon sok van:
    - GPS, LCD vezérlő, SD kártya kezelő, WIFI, Bluetooth, ZigBee, GSM, ...

GPS Shield
€18



MP3 Shield
€25

2.8” TFT és érintőképernyő
€40

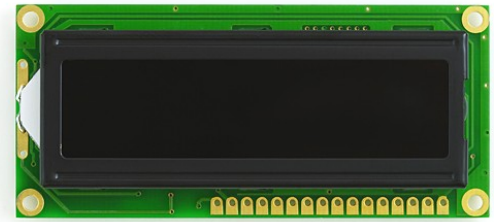
## *Az ajtónyitó megvalósítása*

# Ajtónyitó

- Milyen alkatrészek is kellenek?
  - Kellene egy kis kijelző, amire üzeneteket írhatunk a felhasználónak
  - Kellene egy numerikus billentyűzet
  - Kellene egy kártyaleolvasó – RFID jó lesz
  - Egy relé, ami az ajtót nyitja
  - Egy megfelelően választott Arduino a népes családból

# A kijelző

- Olcsón vehetünk egy 2 soros, 16 oszlopos kijelzőt (2000 Ft)



- Lábak:
  - *Adatbusz*: D0...D7, de 4 bites üzemmódban is megy, nekünk az is elég: D4...D7-et kötjük be
  - *RS*: ezzel lehet jelezni a kijelzőnek, hogy épp egy ASCII karaktert küldünk neki, vagy vezérlő parancsot
  - *EN*: engedélyezés, a kijelző ekkor olvassa el a neki küldött adatot
  - *RW*: ha nem adatot küldünk a kijelzőnek, hanem tőle kérdezünk le információt – ezt nem kötjük be
  - *Tápfeszültség* (5V)
  - *Kijelző kontraszt* (potméter)
  - *Háttérvilágítás tápellátása*
- 6 adatvezetéket használunk: RS, EN, D7, D6, D5, D4

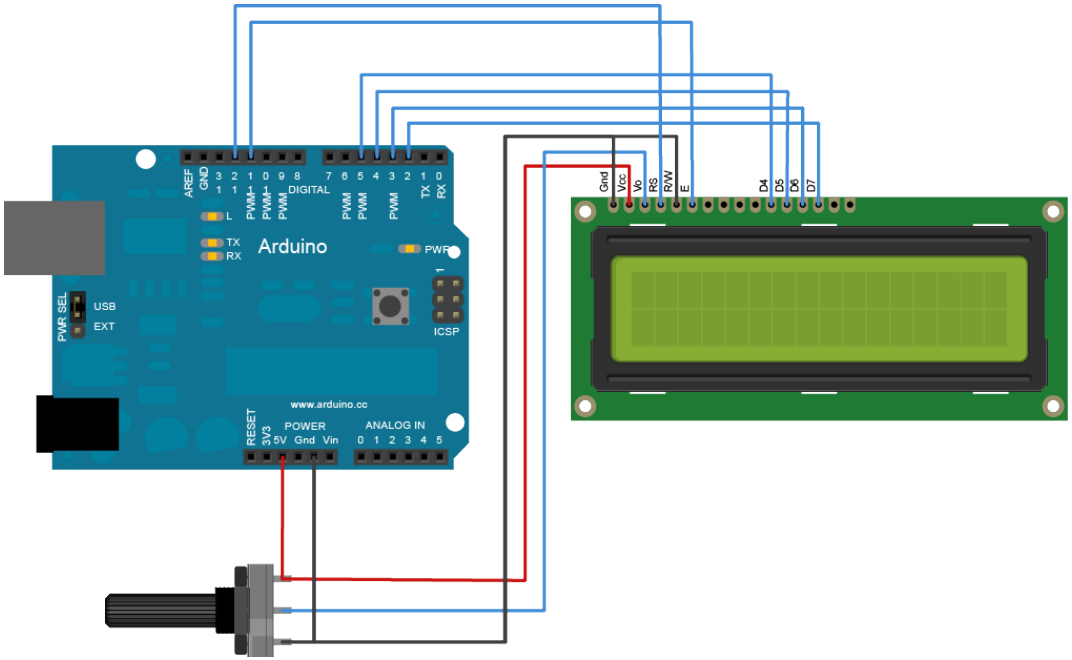
# A kijelző használata

- **Bekötés**:
  Google „arduino display”

- **Használata**:
  **LiquidCrystal** class példányosítása
  (write(), setCursor(), blink(), clear(), stb.)

- **Példa**:

```
#include <LiquidCrystal.h>
const int numRows = 2;
const int numCols = 16;
// beállítjuk, melyik lábakra kötöttük
LiquidCrystal lcd (12, 11, 5, 4, 3, 2);
void setup () {
   lcd.begin (numCols,numRows);
}
void loop () {
   for (int thisLetter = 'a'; thisLetter <= 'z'; thisLetter++) {
      for (int thisRow = 0; thisRow < numRows; thisRow++) {
         for (int thisCol = 0; thisCol < numCols; thisCol++) {
            lcd.setCursor (thisCol, thisRow);
            lcd.write (thisLetter);
            delay (200);
         }
      }
   }
}
```



# A numerikus billentyűzet

- Bárhol beszerezhető, 4x3-mas (900 Ft)

- Lábak:
  - 7 láb, a 4 sorhoz és a 3 oszlophoz
  - Ha megnyomunk egy gombot, rövidre zárja a sor és oszlopvezetékét

# A numerikus billentyűzet használata

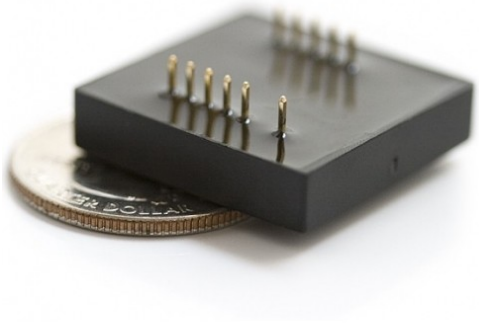
- Google „arduino keypad”
- **Használata**:
  **Keypad** class példányosítása
  (getKey(), waitForKey(), getState(), stb.)

- **Példa**:

```
#include <Keypad.h>
const byte ROWS = 4;
const byte COLS = 3;
char keys[ROWS][COLS] = {  {'1','2','3'},   {'4','5','6'},   {'7','8','9'},   {'*','0','#'}};
byte rowPins[ROWS] = {32, 22, 24, 28}; // sorok lábait hova kötöttük az Arduino-n
byte colPins[COLS] = { 30, 34, 26 };  // oszlopok lábait hova kötöttük az Arduino-n

Keypad keyPad = Keypad ( makeKeymap (keys), rowPins, colPins, ROWS, COLS );

#define ledpin 13
void setup () {
   digitalWrite (ledpin, HIGH);
}
void loop () {
   char key = keyPad.getKey();
   if(key)  {
      switch (key)  {
         case '*':
            digitalWrite(ledpin, LOW);
            break;
         case '#':
            digitalWrite(ledpin, HIGH);
            break;
      }
   }
}
```

# Az RFID olvasó

- Rádiofrekvenciás azonosítás
- Minden kártyának garantáltan egyedi, 12 hexa karakteres kódja van
- Leolvasó: drágát vettünk (7000 Ft), mert pont csak azt lehetett kapni
- Kártya: olcsó (250 Ft/db, kártya, korong, kulcstartó, stb.)



- Soros porton kommunikál
- Lábak:
  - Tápfeszültség (5V)
  - Külső antenna (nekünk a belső bőven elég)
  - Formátum kiválasztó (mi az ASCII módot választjuk)
  - 2 vonal adatátvitelre (csak az egyiket használjuk, soros portként)
  - LED/berregő a kártyaleolvasáskor (nem kötünk rá ilyet)
  - Reset
- Arduinohoz 2 vezetéket kötünk: Reset (digitális lábra), D0 (soros RX lábra)

# Az RFID olvasó használata

- Google „arduino id-12”
- **Használata**:
  soros eszközként

- **Példa**: (reset a 2-es, D0 az RX1 lábra kötve)

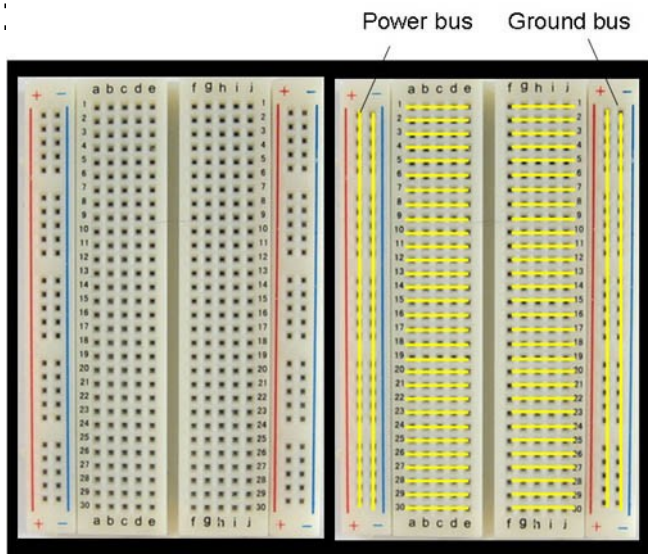
```
int RFIDResetPin = 2;                    // A 2-es digitális lábra kötöttük a leolvasó reset lábát
char userID[13];                         // Ebbe a tömbbe olvassuk majd be a kártya azonosítóját
void setup () {
   Serial.begin(9600);                   // 0-ás soros port beállítása 9600 bps-re (debug-hoz)
   Serial1.begin(9600);                  // 1-es soros port beállítása 9600 bps-re (RFID-t ide kötöttük)
   pinMode(RFIDResetPin, OUTPUT);        // RFID Reset láb beállítása „kimenet” módba
   digitalWrite(RFIDResetPin, LOW);      // RFID reset, felfutó élet csinálunk. Alacsonyba visszük...
   delay (100);                          // … kicsit várunk ...
   digitalWrite(RFIDResetPin, HIGH);     // … majd magasba emeljük.
}
void loop () {
   while (Serial1.available()) {         // Ha jött byte az RFID soros portjától, feldolgozzuk.
      int readByte = Serial1.read();     // Kiolvassuk a következő byte-ot
      if (readByte == 2)                 // Az ASCII 0x2 az „üzenet eleje” karakter
         index = 0;                      // Visszapörgetjünk az indexet, a userID-t az elejétől töltögetjük.
      else if (readByte == 3) {          // Az ASCII 0x3 az „üzenet vége” karakter
         userID[index] = 0;              // Lezárjuk, NULL terminált karakterlánccá tesszük.
         Serial.println(userID);         // Elküldjük a debug rendszernek. Ki fogja írni a PC-n.
      }
      else
         userID[index++] = readByte;     // Ha az üzenet közepén vagyunk, eltároljuk a karaktert.
   }
}
```

# A megfelelő Arduino kiválasztása

- Szempontok:
  - Hány digitális ki- és bemenet kell:
    - A kijelzőnek: 6
    - A billentyűzetnek: 7
    - Az RFID olvasónak: 2
    - Összesen: 15
    - A Leonardo 20-at kezel, de a beszerzéskor még nem létezett. Az elődje 14-et tudott, ezért az ADK-t választottuk. (54-et tud)
  - További ki- és bemenetek:
    - Analóg be, PWM ki nem kell
    - Soros port kell az RFID-nek
    - Ha debug-olni is akarunk, ahhoz kell még egy soros port (USB-n a PC serial monitor-jához)
    - A Leonardo pont 2 soros portot tud (az elődje csak 1-et tudott)
- Ideális választás: **Leonardo**

# A kapunyitó összeállítása

- Breadboard segítségével, forrasztás nélkül összehuzaloztuk
  - Breadboard: lyukrendszer szabványos lyuktávolsággal és megadott összekötöttséggel:




- De nem úsztuk meg teljesen a forrasztást:
  - A kijelzőnek és a billentyűzetnek nem volt tüskesora
    - Tüskesort vettünk és ráforrasztottuk
  - Az RFID tüskesora nem volt kompatibilis a breadboard-dal
    - Átalakítót vettünk, és ráforrasztottuk
- *Forrasztás megtanulható pár youtube tutorial-ból!*

# A kapunyitó szoftvere

- 3 állapotú állapotgépet valósítunk meg:
  - *start* állapot: kiinduló állapot, várjuk a kártyalehúzást
  - *rfid* állapot: lehúzták a kártyát, az RFID olvasó épp küldi a kártya ID karaktereit
  - *code* állapot: számjegy leütésére vár
- A 12 karakteres kártya ID-t összefűzzük a beírt 4 betűs kóddal

  → Így egy 16 karakteres string-et kapunk
- Ha ez a 16 karakter megegyezik a letároltak valamelyikével, akkor kinyitjuk az ajtót

# A kapunyitó szoftvere - 2

```
#include <LiquidCrystal.h>
#include <Key.h>
#include <Keypad.h>

int RFIDResetPin = 2;
int LEDPin = 13;

char userID[17];                                  // ide kerül a kártya ID (12 karakter), utánaírva a lenyomott gombok
int index = 0;
enum estate { start, rfid, code };                // állapotgép aktuális állapota
estate state = start;

LiquidCrystal lcd(49, 47, 48, 46, 44, 42);        // ide kötöttük a kijelzőt

const byte ROWS = 4;
const byte COLS = 3;

char keys[ROWS][COLS] = {  {'1','2','3'},   {'4','5','6'},   {'7','8','9'},   {'*','0','#'} };

byte rowPins[ROWS] = {32, 22, 24, 28};            // ide kötöttük a billentyűzet sor-vezetékeit
byte colPins[COLS] = { 30, 34, 26 };              // ide kötöttük a billentyűzet oszlop-vezetékeit

Keypad keyPad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

// Az elfogadott kártyák és kódjaik. Első 12 karakter: kártya ID, utána írt 4 karakter: a számkód
char* codes[] = {"010B4CF292261234", "010B4CED58F37899", "010B11C56FB19024", "010B1147E8B41290",
            "010B11C5F12F7085", "010B112F3F0B0963", "010B11481C4F7412", "010B1148095A3254",
            "010B1147F3AF6325", "010B114806551589", "010B1147FEA28563", "010B11C56DB33574",
            "010B4CF0D5637412", "010B4CF26DD96521", "010B4CE9B9164589", NULL};
```

# A kapunyitó szoftvere - 3

- A kötelező **setup ()** függvény:

```
void setup() {

    Serial.begin(9600);                 // 0-ás soros port beállítása 9600 bps-re (debug-hoz)
    Serial1.begin(9600);                // 1-es soros port beállítása 9600 bps-re (RFID-hoz)

    pinMode(RFIDResetPin, OUTPUT);      // RFID Reset láb beállítása „kimenet” módba
    digitalWrite (RFIDResetPin, LOW);   // RFID reset, felfutó élet csinálunk. Alacsonyba visszük...
    delay (100);                        // … kicsit várunk ...
    digitalWrite (RFIDResetPin, HIGH);  // … majd magasba emeljük.

    pinMode(LEDPin, OUTPUT);            // Beépített LED lábának beállítása „kimenet”-re

    lcd.begin(16, 2);                   // 16 oszlopos, 2 soros kijelzőnk van
    lcd.print("J\224het a k\240rtya!");  // Írjuk ki rögtön a szöveget
}
```

# A kapunyitó szoftvere - 4

- A kötelező **loop ()** függvény:

```
void loop () {
   while (Serial1.available()) {                    // Először feldolgozzuk az RFID olvasó által küldött kártya ID-t
      int readByte = Serial1.read();                // Soron következő karakter
      if (state == start && readByte == 2) {        // Ha a kiinduló állapotban megjött az „üzenet eleje” karakter (ASCII 0x2),
         state = rfid;                              // átmegyünk a kártya ID olvasó állapotba (rfid)
         index = 0;
      }
      else if (state == rfid && readByte == 3) {    // Ha megjött az „üzenet vége” karakter (ASCII 0x3),
         state = code;                              // átmegyünk a billentyűzetfigyelő állapotba (code)
         lcd.clear();                               // Töröljük a kijelzőt,
         userID[index] = 0;                         // lezáró 0-ál teszünk a kártya ID végére,
         lcd.print(userID);                         // kiírjuk a kártya ID-t a kijelzőre
         lcd.setCursor (0,1);                       // Második sorba megyünk
         lcd.print("J\224het a k\242d!");           // Kiírjuk, hogy jöhet a kód
      }
      if (state==rfid && readByte != 2 && readByte != 10 && readByte != 13 && index<12)    // ha még nincs vége az RFID kód elküldésének
         userID[index++] = readByte;                // eltároljuk a soros porton érkezett karaktert
   }

   char key = keyPad.getKey();                      // Megnézzük, volt-e gombnyomás
   if (state == code && index == 16) {              // Ha kódot olvastunk, és vége (mert megvan a 12 betűs kártya ID és a 4 betűs kód)
      userID[index] = 0;                            // Lezáró 0-át teszünk a végére
      Serial.println(userID);                       // Kiírjuk a soros terminálra debug célból
      checkCardAndCode();                           // Ellenőrizzük a helyességét, és végrehajtjuk, amit kell
      state = start;                                // Visszatérünk kiinduló állapotba
      lcd.clear ();                                 // Kijelző törlése
      lcd.setCursor (0,0);                          // Kurzor a sarokba
      lcd.print("J\224het a k\240rtya!");           // Kiírjuk, hogy jöhet a kártya
   }
   else if (state == code && key)                   // Ha jött egy számjegy, de még nem írta be mind a 4-et
      userID[index++] = key;                        // Hozzáírjuk a többihez
}
```

# A kapunyitó szoftvere - 5

- A kártya és a kód ellenőrzése:

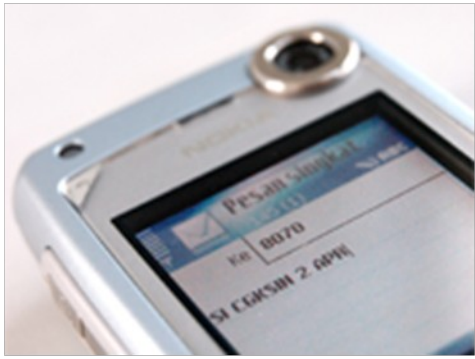
```
void checkCardAndCode () {
   lcd.clear ();                          // kijelző törlése
   lcd.setCursor (0, 0);                  // kurzor a sarokba
   int ix = 0;
   while (codes[ix]) {                    // végignézzük az összes tárolt kódot
      if (!strcmp(userID, codes[ix])) {   // ha egyezik, kész
         lcd.print ("Rendben!");          // szólunk, hogy nyílik a kapu
         digitalWrite (LEDPin, 1);        // kigyújtjuk a LED-et
         break;                           // nem keresünk tovább
      }
      ix++;
   }
   if (!codes[ix])                        // ha a lista végére értünk, és nincs meg,
      lcd.print ("Rossz k\242d!");        // közöljük a rossz hírt

   delay (1000);                          // várunk 1 másodpercet, hogy el lehessen olvasni

   digitalWrite (LEDPin, 0);              // LED lekapcsolása
   digitalWrite (RFIDResetPin, LOW);      // RFID reset, felfutó élet csinálunk. Alacsonyba visszük...
   delay (100);                           // … kicsit várunk ...
   digitalWrite (RFIDResetPin, HIGH);     // … majd magasba emeljük.
}
```

# Összegzés

- Megismertük az Arduino-t
  - Milyen kimenetei/bemenetei vannak
  - Hogyan kell ezekre perifériát illeszteni
  - Hogyan kell a hozzátartozó szoftvert megírni
- Megvalósítottuk az ajtónyitót
  - Bolti, megvásárolható eszközökből
  - Forrasztás csak az összekötögetések miatt kellett
  - Megírtuk a szoftverét

# Kitekintés

- Ezeregy Arduino klón létezik
  - Legálisan, mert open source a hardver is
- Ezeregy Arduino-szerű mikrokontroller kártya kapható
  - OLIMEXINO: AVR mikrokontrollerrel
  - DUINOMITE: PIC mikrokontrollerrel
  - MAPLE: ARM Cortex M3 processzorral
  - NETDUINO: 32 bites ARM processzorral, .NET környezetben programozható (Visual Studio-val)
  - stb...



# Számítógép Architektúrák

*Perifériakezelés a PCI-ban és a PCI Express-ben*

Horváth Gábor
docens
BME Híradástechnikai Tanszék
ghorvath@hit.bme.hu

2012. október 1.
Budapest

# Tartalom

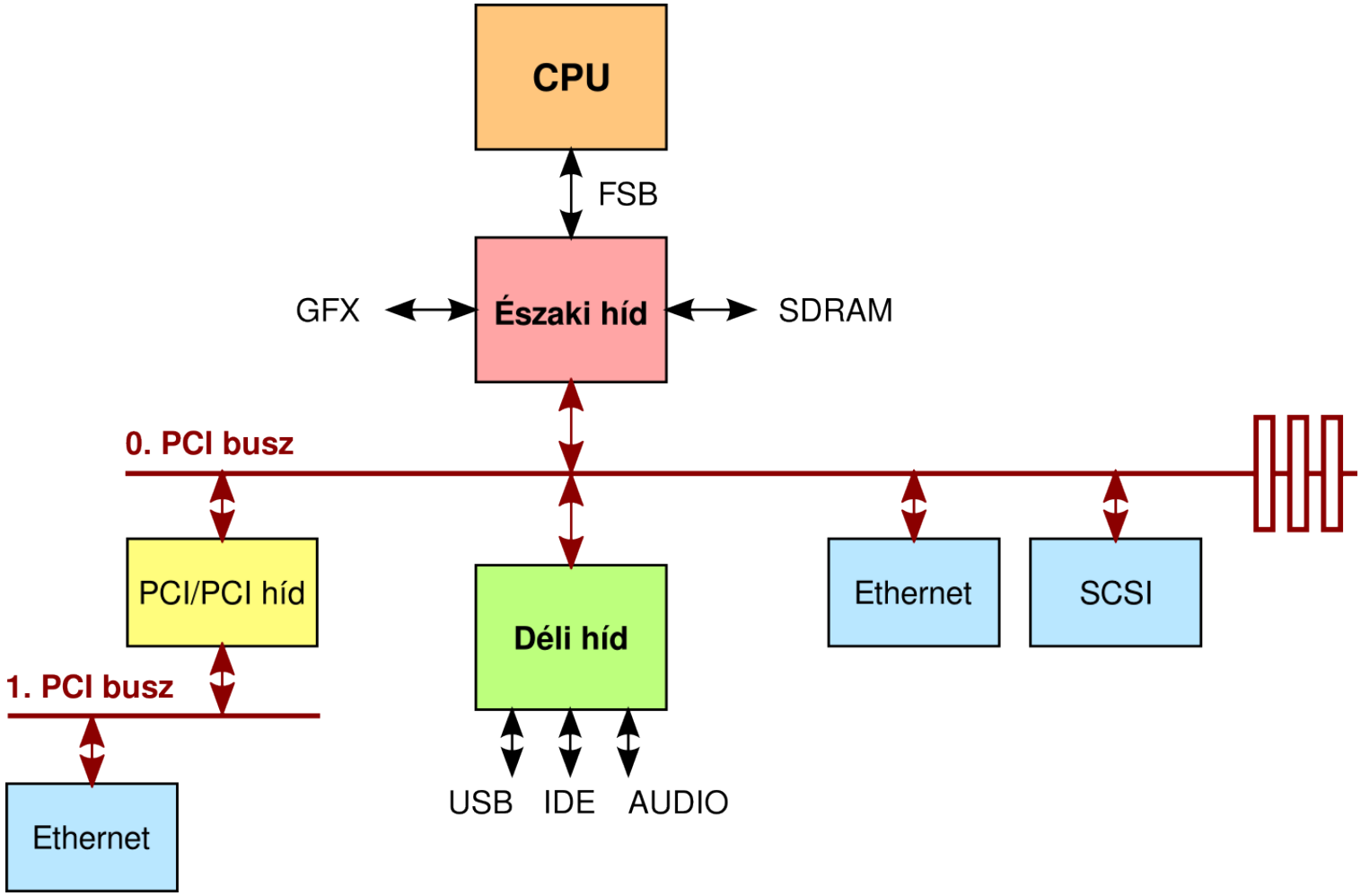
- A gyakorlat célja:
  - A PCI és PCI Express megismerése
- Tartalomjegyzék:
  - PCI alapú rendszerek
    - Egyszerű PCI adatátvitel
    - Arbitráció
    - Megszakításkezelés
    - Konfigurálás
  - PCI Express rendszerek
    - Adatátvitel a PCI Express-ben
    - Megszakításkezelés
    - Konfigurálás

# A PCI

- PCI = Peripheral Component Interfész, 1992, Intel
- Főbb tulajdonságok:
  - Processzorfüggetlen (PC, Macintosh, DEC, HP, SUN szerverek)
  - Automatikus periféria felismerés és konfiguráció támogatása
  - Átvitel: egyenként vagy burst módban
  - 32 vagy 64 bites adategységek
  - Szinkron busz, órajel: 33 MHz, 1995-től 66 MHz
  - Egy PCI buszon max 32 eszköz
  - Egy rendszerben max 256 PCI busz
  - Alacsony fogyasztás támogatása

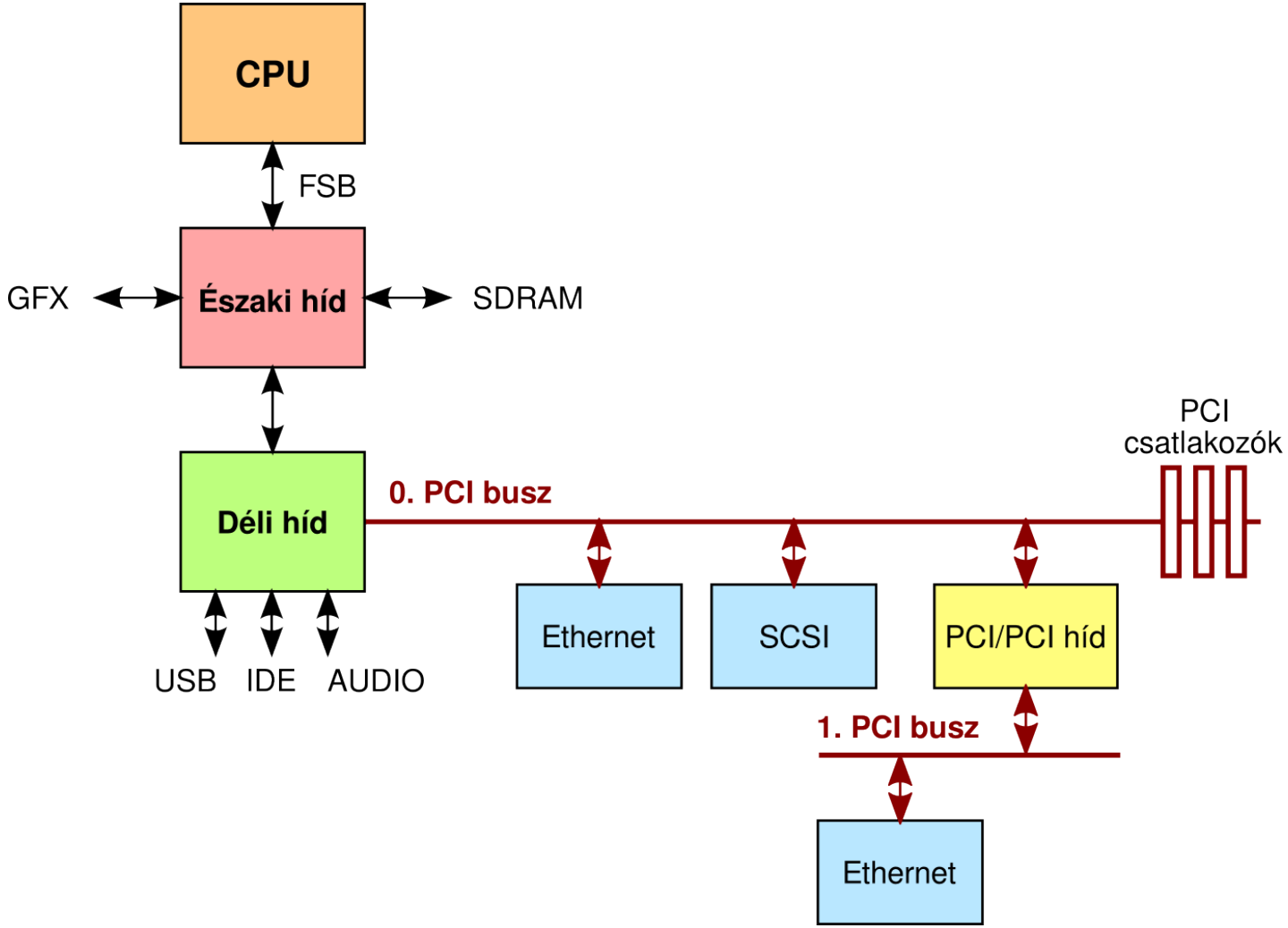
# PCI-t használó rendszerek

- PCI helye az Intel Pentium Pro-ban:



# PCI-t használó rendszerek

- PCI helye a PC-ben a Pentium 4-től kezdve:



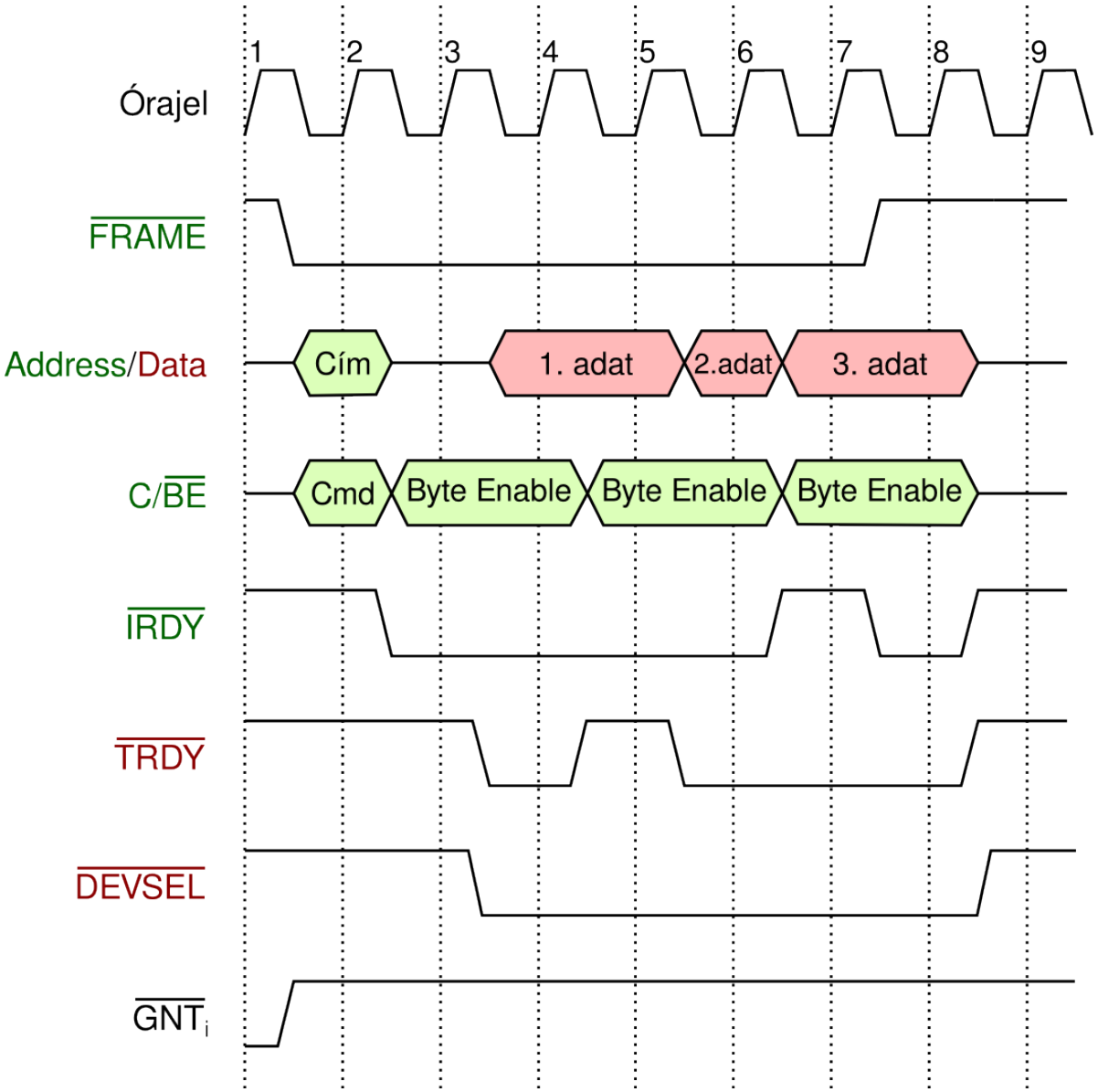
# A PCI elemei

- A PCI busz elemei:
  - Host/PCI híd
    - Processzor I/O és memória kérések ↔ PCI tranzakciók
  - PCI eszközök
    - Max 32/busz, elektromos okokból inkább max. 10
    - Egy eszközön 1-8 function (logikai periféria)
  - PCI/PCI híd
    - Egy PCI sínre egy másik sínt illeszt
- Illeszthető perifériák max száma elméletileg:
  - 8 periféria/eszköz * 32 eszköz/busz * 256 busz/rendszer = 65536 periféria/rendszer

# Adatátvitel

- Minden PCI perifériának max. 6 ablak
  - Ablak lehet:
    - A memória címtartomány egy része
    - Az I/O címtartomány egy része
  - CPU egy PCI periféria memória ablakára végez memóriaműveletet → a perifériával cserél adatot
  - CPU egy PCI periféria I/O ablakára végez perifériaműveletet → a perifériával cserél adatot
  - Az eszköz meg tudja mondani, hogy neki hány és mekkora memória és I/O ablak kell
  - A BIOS ill. az op. rendszer osztja ki az ablakokat
    - A PCI perifériákban be kell állítani az ablakok kezdetét
    - A PCI eszköz figyeli az ablakait célzó műveleteket, ha ilyet lát, válaszol

# Adatátvitel



# Adatátvitel

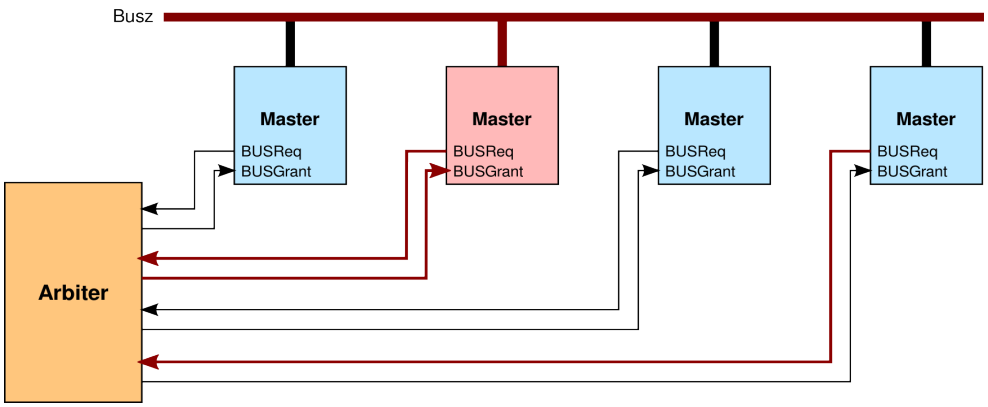
- Forgalomszabályozás:
  - Az $\overline{\text{IRDY}}$ jelzéssel jelzi a kezdeményező, hogy kész a következő adat átvitelére
  - A $\overline{\text{TRDY}}$ jelzéssel jelzi a megcímzett periféria, hogy kész a következő adat átvitelére
- Tranzakciós modellek:
  - Programozott I/O. Kezdeményező: CPU, irányultság: periféria
  - DMA. Kezdeményező: periféria, irányultság: memória
  - Peer-to-peer adatátvitel. Kezdeményező: periféria, irányultság: periféria

# PCI parancsok

| C/$\overline{BE}$ | Parancs |
|---|---|
| 0000 | Megszakítás nyugtázás (Interrupt Acknowledge) |
| 0010 | I/O olvasás (I/O Read) |
| 0011 | I/O írás (I/O Write) |
| 0110 | Memória olvasás (Memory Read) |
| 0111 | Memória írás (Memory Write) |
| 1010 | Konfiguráció olvasás (Configuration Read) |
| 1011 | Konfiguráció írás (Configuration Write) |

# Arbitráció

- Párhuzamos arbitráció:




- Rejtett arbitráció:
  - Versengés és eredményhirdetés az aktuális tranzakció ***közben***
- Algoritmus:
  - Fair
  - Figyelembe veheti a periféria késleltetés-érzékenységét
- Példa:
  - A, B késleltetésérzékeny, X, Y, Z nem az
  - Sorrend: A, B, X, A, B, Y, A, B, Z, A, B, X, A, B, Y, A, B, Z, stb.

# Megszakításkezelés

- Megszakításkérés kétféle képpen:
  - Az interrupt vezeték egyikével
  - Üzenettel jelzett megszakítással (MSI)
- Interrupt vezetékekkel:
  - Minden csatolón 4 van
  - Lehetnek osztott vagy dedikált vezetékek
  - Az, hogy a processzorban hányas megszakításként látszik, lekérdezhető az eszköztől
  - Több eszköz megszakítása is leképezhető ugyanarra a CPU megszakításra → szoftver polling deríti fel a kiváltó perifériát
- Üzenettel jelzett megszakítás:
  - A periféria egy speciális adatot ír egy speciális címre
  - Host/PCI híd figyeli a címet, az írást CPU interrupt-ra képezi le

# Konfiguráció

- Perifériánként 64 db 32 bites konfigurációs regiszter
- Konfigurációs regiszter kiolvasása (PC-n!):
  - 0CF8h I/O cím ← konfigurációs regiszter címe
  - Regiszter tartalma: kiolvasható a 0CFCh I/O címről
- Konfigurációs regiszter megváltoztatása (PC-n!):
  - 0CF8h I/O cím ← konfigurációs regiszter címe
  - 0CFCh I/O cím ← konfigurációs regiszter új értéke
- Konfigurációs regiszterek címzése:

| 31 | 30...24 | 23...16 | 15...11 | 10...8 | 7...2 | 1...0 |
|---|---|---|---|---|---|---|
| 1 | Fenntartott | Busz szám | Eszköz | Function | Reg. szám | 00 |

# Konfiguráció

- Konfigurációs regiszter írása/olvasása:
  - Host/PCI híd I/O műveletet lát a 0CF8h és a 0CFCh címen
  - Látja, melyik eszköz van megcímezve

    → beállítja az $\overline{\text{IDSEL}}$ vezetékét 0-ba, a többiekét 1-be
  - Konfiguráció írás/olvasás tranzakciót indít

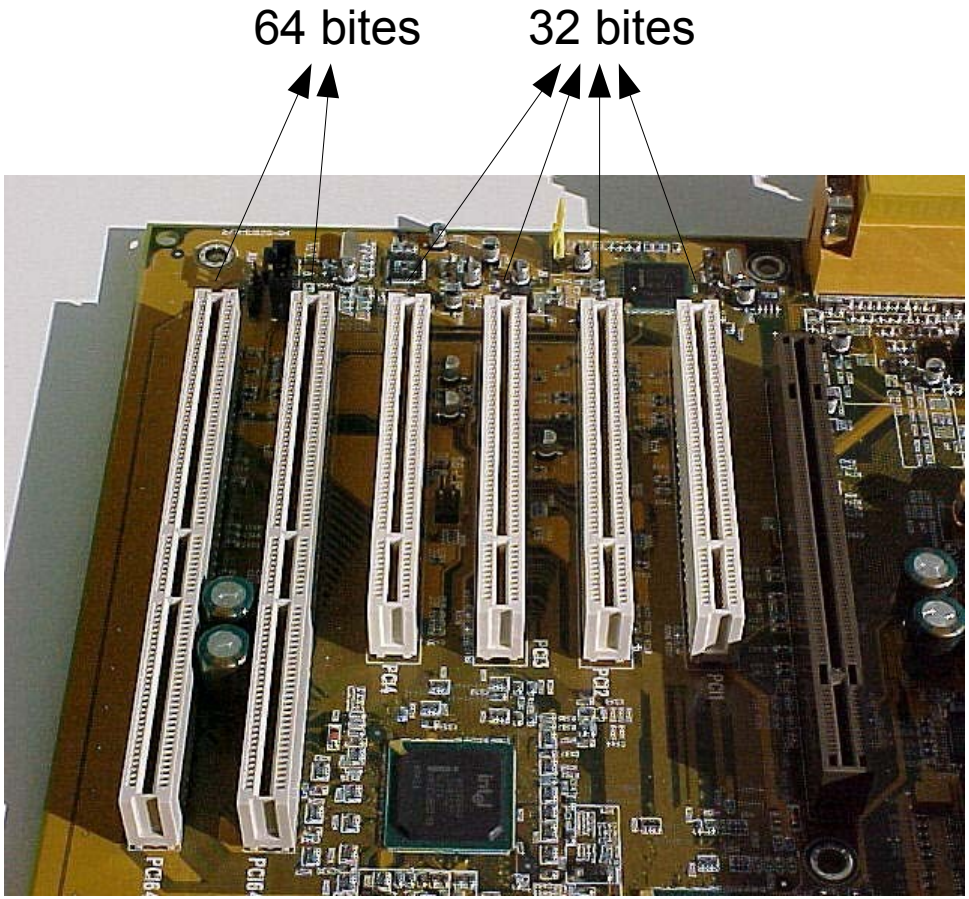
    → csak a kijelölt periféria figyel!
    - C/BE vonalak: konfiguráció írás/olvasás kódja
    - A/D vonalakra:
      – Konfiguráció típusa
      – A megcímzett function száma
      – Kért regiszter száma
    - Regiszter tartalmának átvitele: mint az egyszerű adatátvitelnél

# Konfiguráció

- Regiszterek tartalma:
  - A 64 regiszterből 16 kötött:
    - VendorID, DeviceID, Revision
    - Class Code
    - Interrupt Pin
    - Interrupt Line
    - Base Address Register 0...5
- Rendszerindítás:
  - Eszközök szisztematikus lekérdezése
  - Ablakok kiosztása
  - Device driver betöltése
  - További eszközfüggő konfiguráció
  - ...

# PCI csatoló

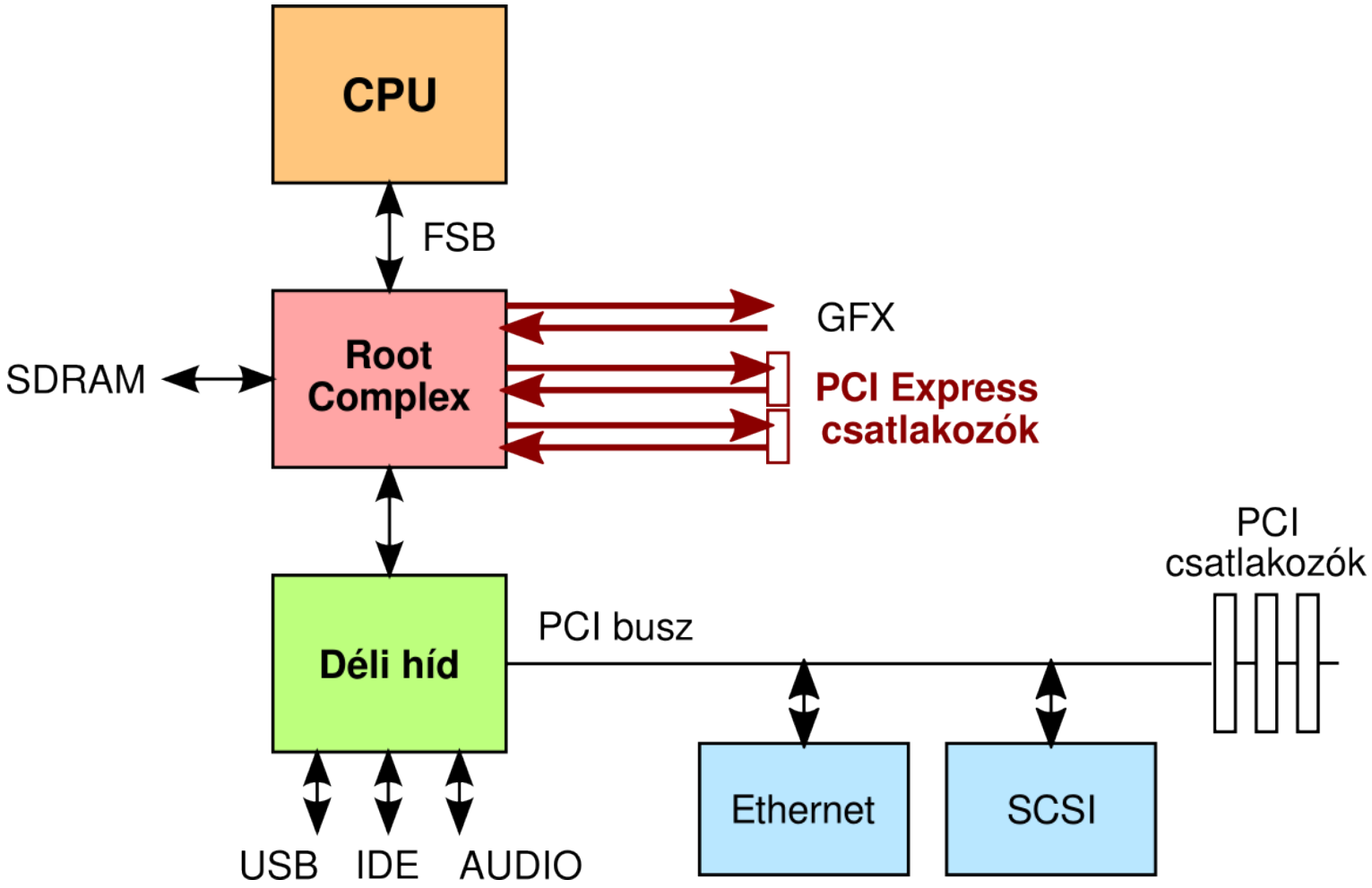
- Kötelezően használandó:
  - Órajel, Reset
  - A/D[0...31]
  - C/$\overline{\text{BE}}$[0...3]
  - $\overline{\text{FRAME}}$
  - $\overline{\text{IRDY}}$, $\overline{\text{TRDY}}$, $\overline{\text{STOP}}$
  - $\overline{\text{DEVSEL}}$, $\overline{\text{IDSEL}}$
  - Paritás, $\overline{\text{PERR}}$, $\overline{\text{SERR}}$
  - $\overline{\text{REQ}}$, $\overline{\text{GNT}}$
- Opcionális:
  - A/D[32...63]
  - C/$\overline{\text{BE}}$[4...7]
  - $\overline{\text{INTA}}$, $\overline{\text{INTB}}$, $\overline{\text{INTC}}$, $\overline{\text{INTD}}$
  - $\overline{\text{REQ64}}$, $\overline{\text{ACK64}}$
  - $\overline{\text{CLKRUN}}$



# PCI Express

- Célok:
  - PCI-nál nagyobb adatátviteli sebesség
  - Lehető legteljesebb szoftveres PCI kompatibilitás
- Leglátványosabb eltérések:
  - Soros átvitel
    - Így kiküszöbölhető a párhuzamos esetben tapasztalható jelek közötti csúszás
  - Pont-pont összeköttetések
    - Nem osztott a közeg → nincs versengés, várakozás, éhezés, arbitráció

# Egyszerű PCI Express rendszer



# Egyszerű PCI Express rendszer

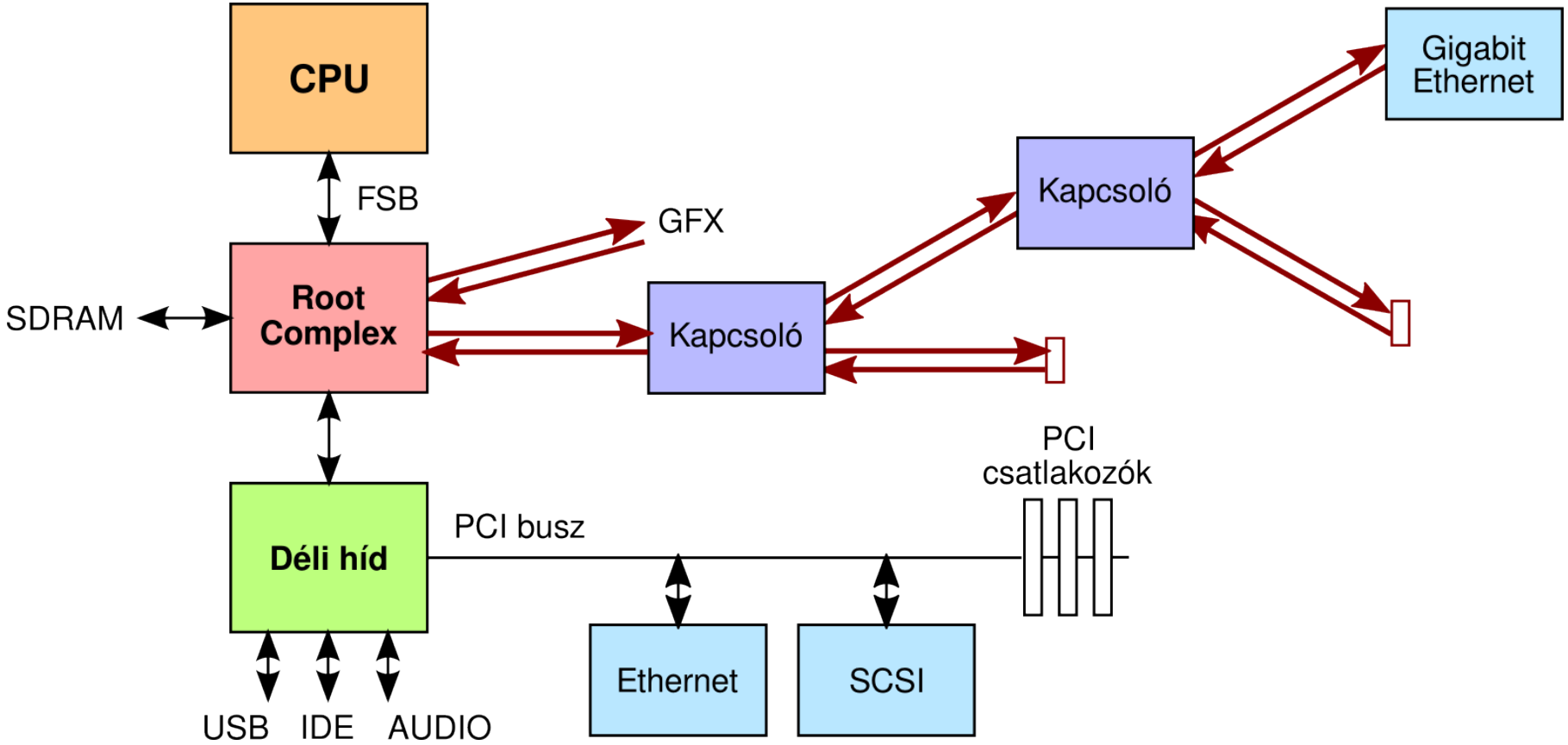
- Adatátvitel sorosan:
  - Órajel nincs (automatikus, a 0/1 átmenetekből, PLL-lel)
  - Egy érpár: 500 MB/s
  - Két eszköz között full duplex (kétirányú) kapcsolat:

    → 2 soros érpár
  - Két eszköz több full duplex soros kapcsolattal is össze lehet kötve
    - 1 full duplex soros kapcsolat neve: ***lane*** (pálya)
    - Szabvány: 1x, 4x, 8x, 16x, 32x lane
    - Több lane esetén a bitek párhuzamosan vihetők át

      → **De nem szinkronban !!!**

      – 32x pálya esetén 32x500 MB/s = 16 GB/s

# PCI Express rendszer kapcsolókkal

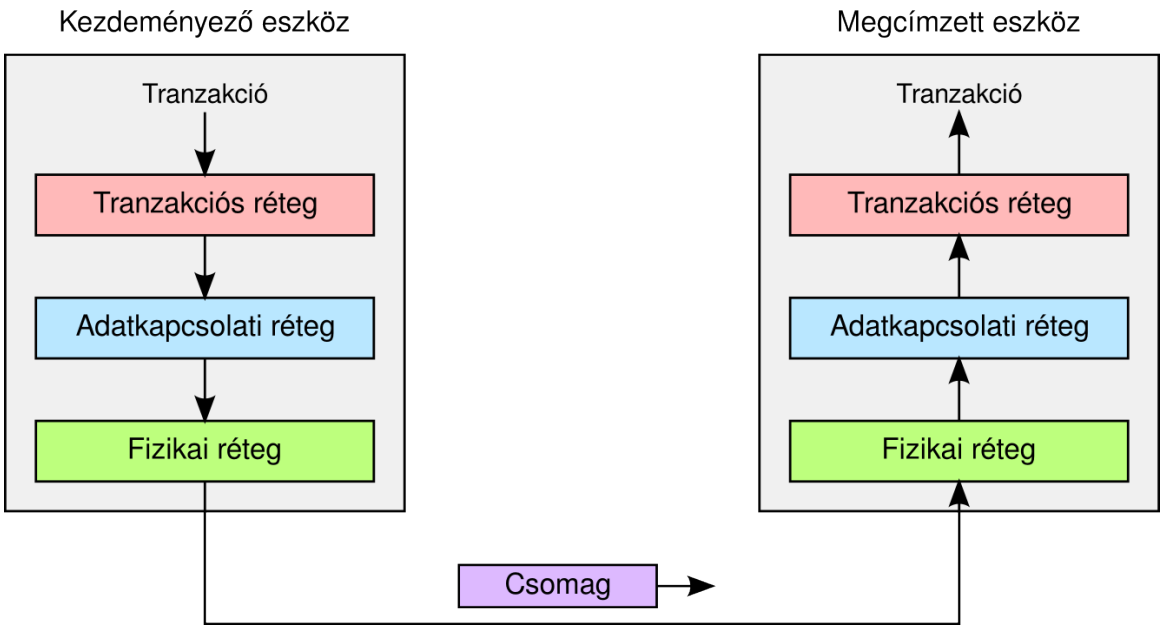


# PCI Express rendszer kapcsolókkal

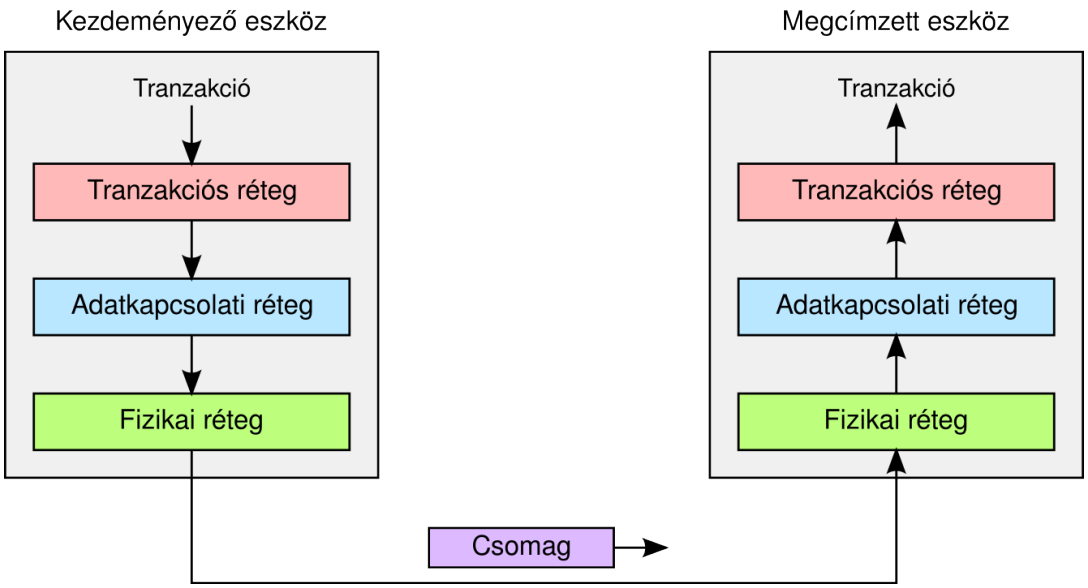
- Csomagkapcsolt hálózat:
  - Eszközök kapcsolókon keresztül kötődnek a Root Complexhez
  - Tranzakciók csomagként utaznak

# Tranzakciók továbbítása

- Tranzakció 3 rétegen megy keresztül, mielőtt a soros érpárra kerül:

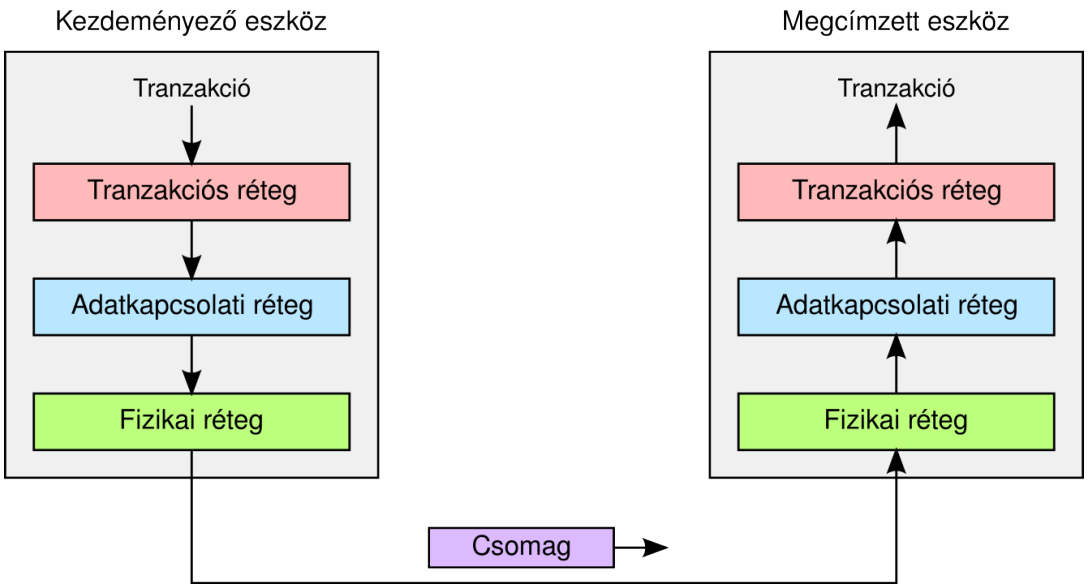


# Tranzakciók továbbítása



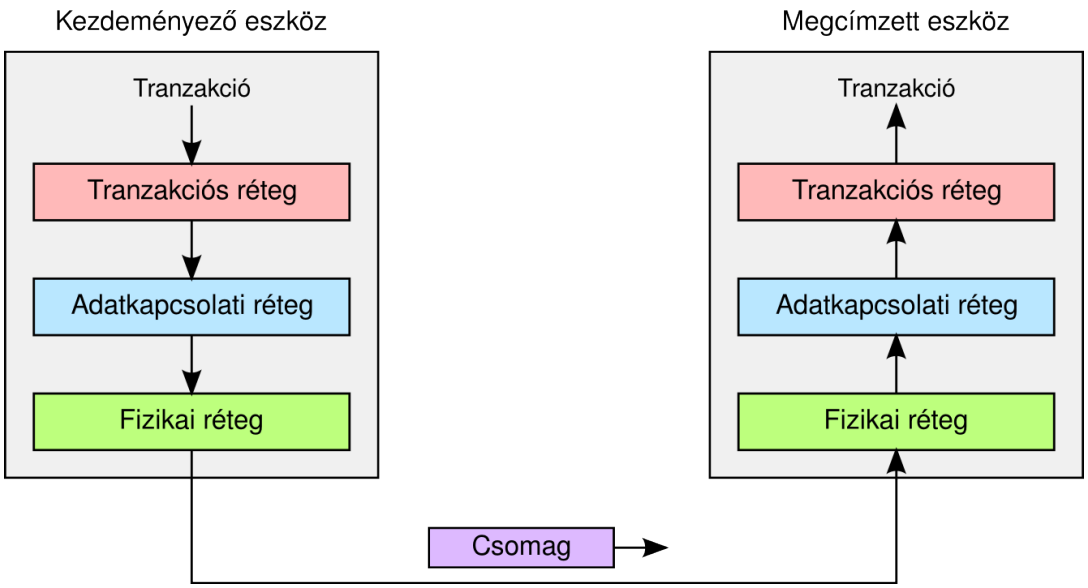

- Tranzakciós réteg: tranzakció becsomagolása
  - Fejléc:
    - Command, bájtsorrend (PCI: C/$\overline{BE}$)
    - Cím (PCI: Address)
  - Rakomány:
    - Átvinni kívánt adat
  - CRC

# Tranzakciók továbbítása



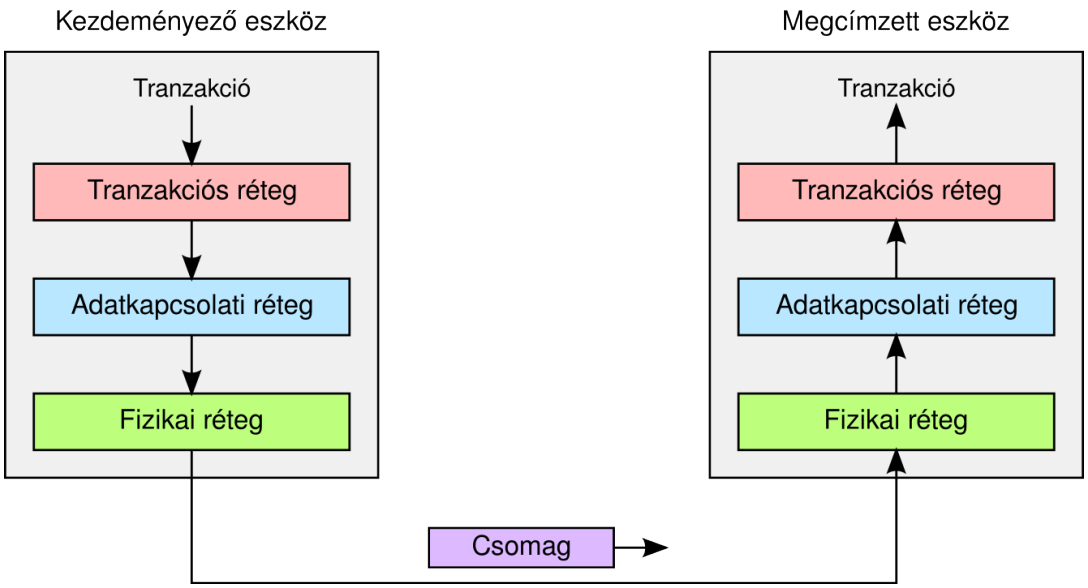

- Adatkapcsolati réteg: csomag hibamentes eljuttatása a szomszédba
  - Újabb fejléc: sorszám (auto-inkrementált)
  - Újabb CRC (sorszámot is védi)

# Tranzakciók továbbítása

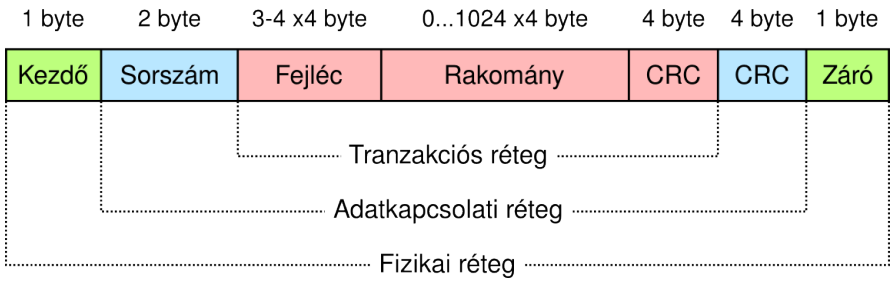



- Fizikai réteg: csomag átvitele a soros érpáron
  - Csomaghatárok kijelölése speciális bájtokkal
  - Megfelelő számú 0/1 átmenet biztosítása:
    - Bitek összekeverésével
    - 8 bitet 10 biten kódolással

# Tranzakciók továbbítása



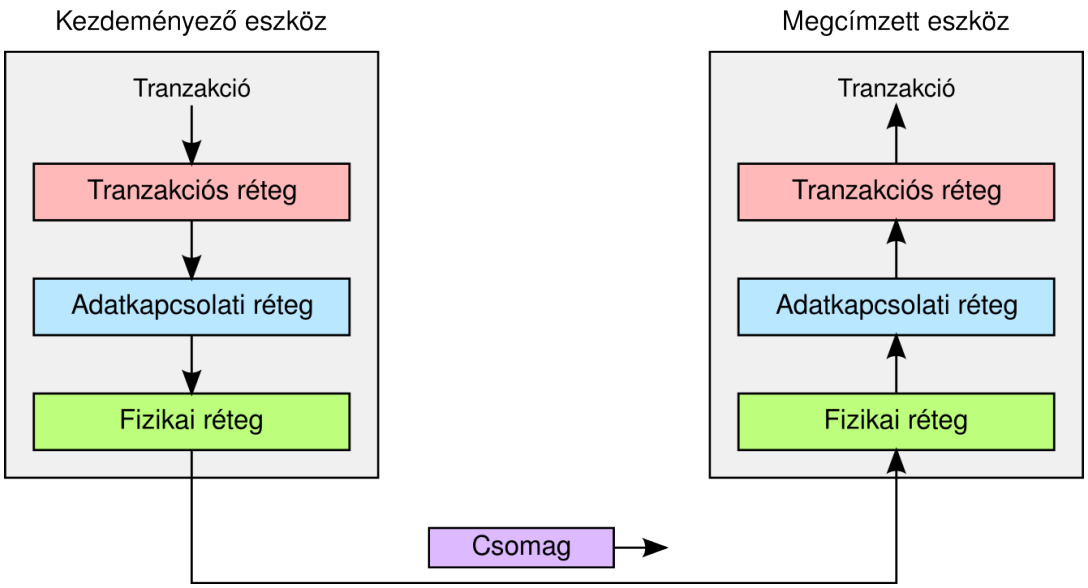

- Csomag formátuma:




- 1 db 32 bites adat átvitele:

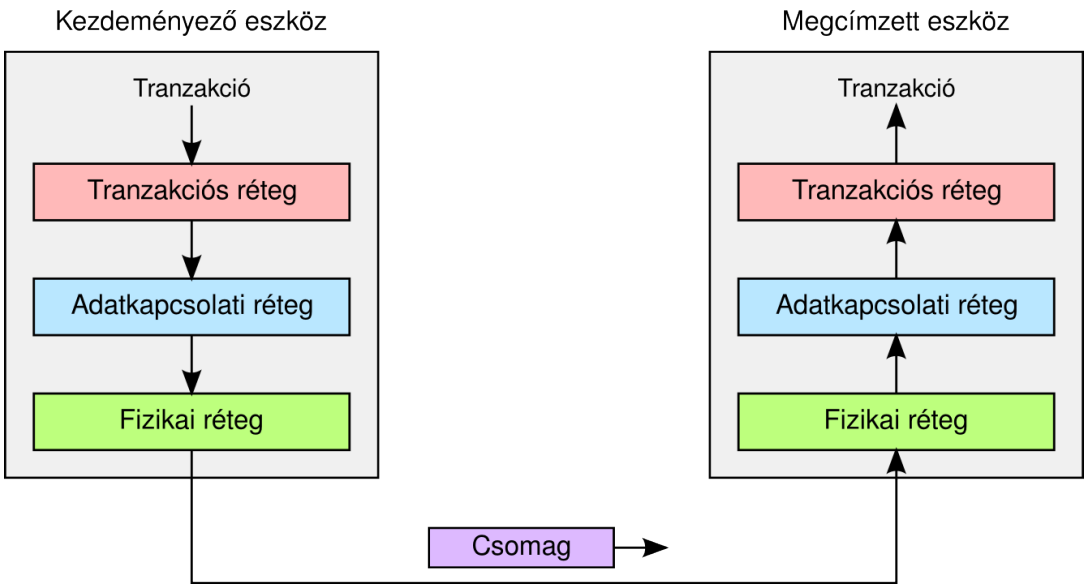
  1+2+3*4+1*4+4+4+1 = 28 bájt

# Tranzakciók továbbítása



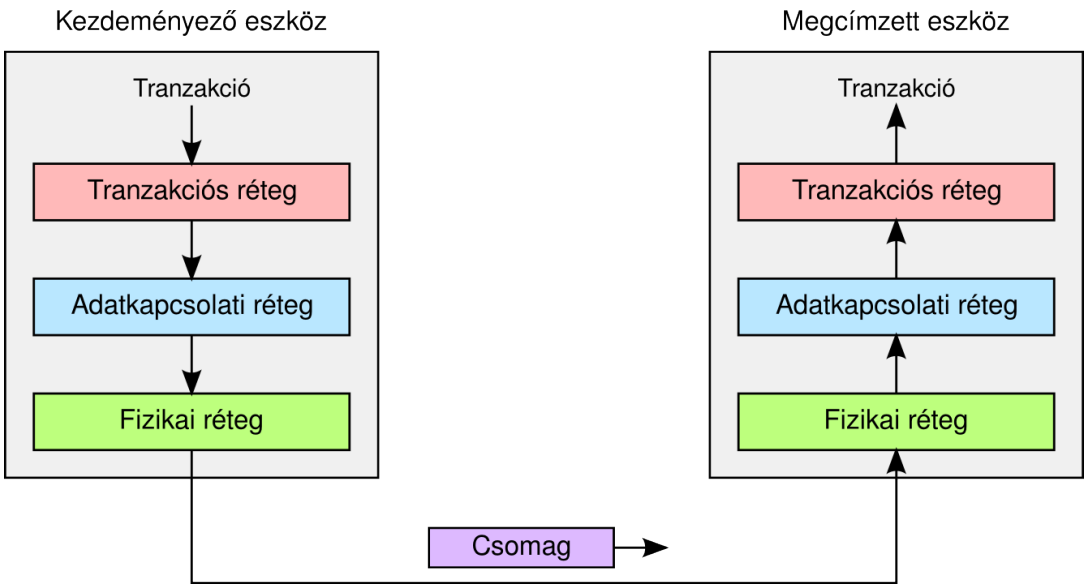

- Vevő oldal, fizikai réteg:
  - Csomaghatárok azonosítása
  - 8-ból 10 bites kódolás visszaállítása
  - Bitösszekeverés visszaállítása

# Tranzakciók továbbítása




- Vevő oldal, adatkapcoslati réteg:
  - CRC és sorszám ellenőrzése
  - Ellenőrzés függvényében negatív vagy pozitív nyugta küldése
  - Negatív nyugta esetén a feladó újraküldi a csomagot

# Tranzakciók továbbítása




- Vevő oldal, tranzakciós réteg:
  - CRC ellenőrzése
  - Tranzakció végrehajtása

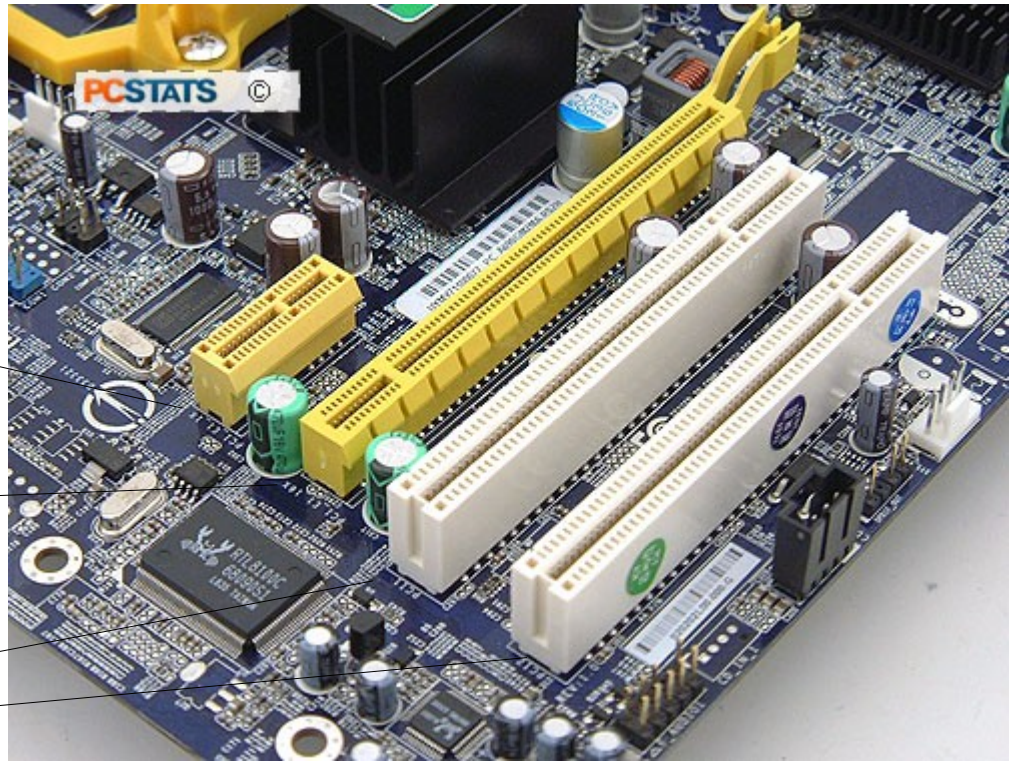
# Tranzakciók továbbítása

- Posted vs. Non-posted tranzakciók:
  - Posted: válasz nem szükséges, pl. írási tranzakciók
  - Non-posted: válasz szükséges, pl. olvasási tranzakciók
    - A tranzakció csak egy olvasási kérés
    - A megcímzett periféria egy újabb tranzakcióban válaszol a kért adatokkal a rakományban
- QoS: fejlécben „forgalmi osztály” mező
  - Sürgősség jelölése
  - A sürgős tranzakció a továbbítás során megelőzheti a nem sürgőset
- Forgalomszabályozás
  - Beérkező csomagok feldolgozásig átmeneti bufferben tárolódnak
  - Fogadó fél rendszeresen jelzi a küldőnek a szabad buffer méretét
  - Ha a küldő nem lát elég buffert a fogadónál, vár a küldéssel

# Megszakításkezelés és konfigurálás

- Megszakításkezelés
  - Üzenet alapú (MSI) → mint a PCI-ban
  - Jelvezeték alapú:
    - A periféria két speciális üzenetet küldhet a Root Complex-nek:
      - Vegye úgy, hogy 0-ba húzta az egyik interrupt vonalat
      - Vegye úgy, hogy 1-be emelte az egyik interrupt vonalat
    - Az így jelzett megszakítás kezelése mint PCI-ban
- Konfiguráció:
  - Nem csak 64 regisztere lehet egy perifériának, hanem 1024
  - Első 64 írása/olvasása: mint PCI-ban (teljes kompatibilitás)
  - Továbbiak: másféle címzés (64 bites, nem részletezzük)

# PCI Express csatoló



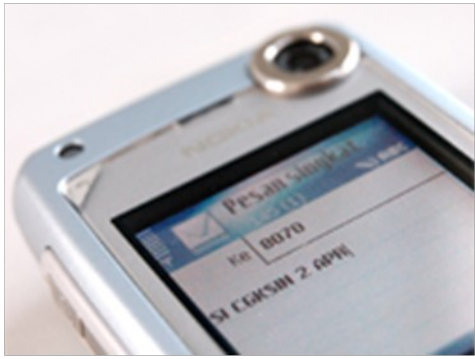

PCIe x1

PCIe x16

PCI

# Összegzés

- Mit tanultunk?
  - PCI: busz, párhuzamos
  - PCI Express: pont-pont, soros
  - Hogy juttatja el a PCI és a PCI Express a CPU kéréseit a perifériáknak?
  - Hogy oldották meg a gyakorlatban az arbitrációt a PCI-ban?
  - Hogy oldották meg a gyakorlatban a megszakításkezelést?
  - Hogy oldották meg a gyakorlatban a perifériák konfigurálását?



# Számítógép Architektúrák

*Az USB csatolófelület*

Horváth Gábor
docens
BME Híradástechnikai Tanszék
ghorvath@hit.bme.hu

2012. október 12.
Budapest

# Tartalom

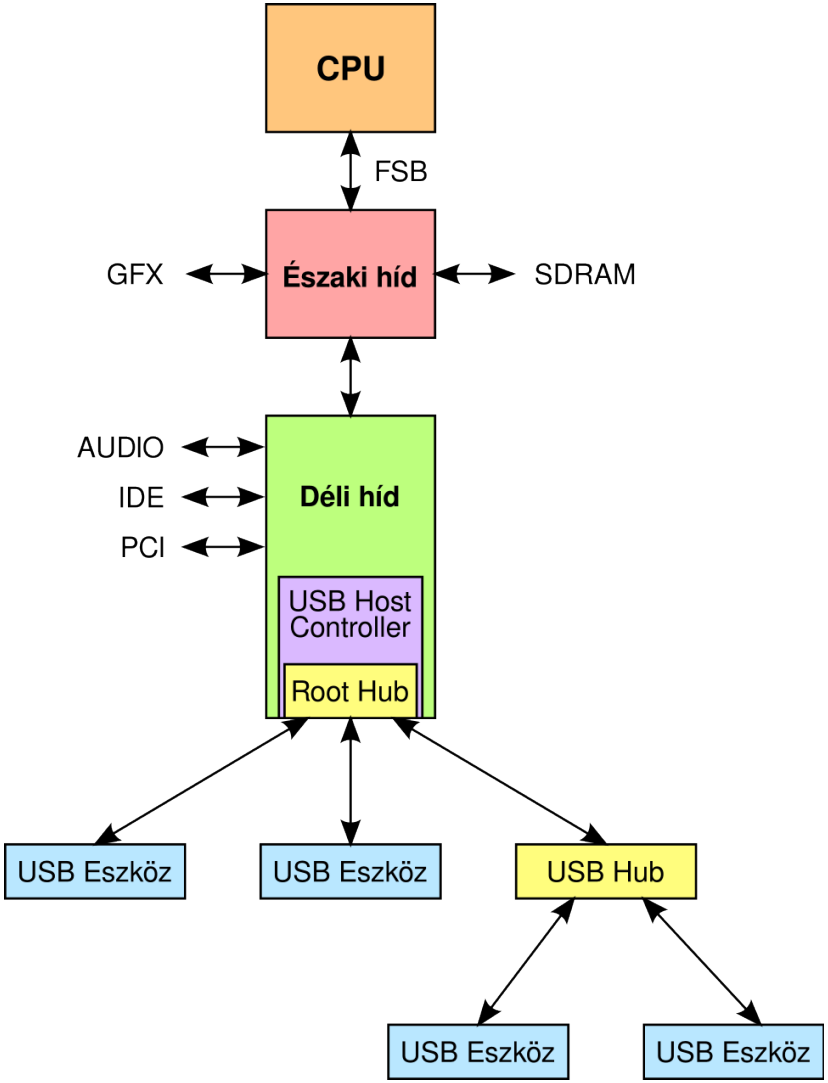
- A gyakorlat célja:
  - Az USB csatolófelület megismerése
- Tartalomjegyzék:
  - Motiváció
  - Az USB felépítése
  - Az USB 1.1
    - Adatátvitel
    - Interrupt-ok
    - Csatlakozók, tápellátás
    - Konfigurálás
  - Az USB 2.0
    - Mi változott?
  - Az USB 3.0
    - Mi változott?

# Motiváció

- 90-es évek: a PC-s perifériák csatlakozása körül káosz
  - Ezernyi gyártóspecifikus csatlakozó
  - Ha nem volt az alaplapon, illesztőkártya kellett
  - A PC-be tehető illesztőkártyák száma korlátos
  - Az ezernyi csatlakozóhoz különböző kábelek, hosszabbítók kellenek
- USB megjelenése
  - 1994, Compaq, DEC, Intel, Microsoft, NEC, Nortel
  - Elsődleges célpont a PC, de azóta minden architektúra támogatja
- Főbb tulajdonságok:
  - Szabványos fizikai csatlakozók
  - Több periféria köthető 1 csatlakozóra
  - Max. 127 eszköz vezérlőnként
  - Perifériák nem terhelik a címteret és nem kell interrupt vonal sem
  - Automatikus periféria felismerés és konfiguráció támogatása
  - A számítógép kikapcsolása nélkül illeszthető/eltávolítható perifériák
  - Olcsó vezérlőáramkörök, alkatrészek
  - A tápellátás az USB kábelen keresztül megoldható (általában)
  - Alacsony fogyasztás támogatása

# Az USB felépítése

- Fa topológia
  - Levelek: a perifériák
  - Csomópontok: USB hub-ok
  - Gyökér: root hub
    (a host controller része)
- Periféria lehet:
  - Egyszerű eszköz
  - Összetett eszköz:
    - Fizikailag egy eszköz
    - Logikai több eszköz



# Mitől busz

- Megszoktuk, hogy a busz osztott közeg
  - Minden szereplő mindent hall
- De ez egy fa!
- Mégis busz:
  - Egyetlen master van, a root hub
  - Amit a master beszél, azt mindenki hallja (terjed a fában)
  - Akire vonatkozik, az odafigyel, a többi nem
  - Amit a periféria mond, az a fában felfelé haladva éri el a mastert
- Nem kell arbitráció!
  - Mivel csak 1 master van
  - Senki nem szólalhat meg, csak ha a master erre utasítja
- Ha busz, akkor megvan a hátránya is:
  - Minél többen csatlakoznak rá, annál lassabb lesz

# USB 1.1

# Az USB 1.1

- 1998, az első igazán elterjedt változat
- Adatsebességek
  - 1.5 Mb/s (Low Speed, LS)
  - 12 Mb/s (Full Speed, FS)

# Tranzakciók és keretek

- A perifériákkal **tranzakciók** segítségével kommunikálunk
- LS és FS sebességű tranzakciók
  - Az LS tranzakciókat az LS és az FS eszközök is látják
  - Az FS tranzakciókat csak az FS eszközök látják
- Mindig a master, a root hub indítja
- Tartalma:
  - Megcélzott periféria azonosítója
  - Ki- vagy bemeneti művelet
  - Átvinni kívánt adatok
    - Kimeneti irány esetén a root hub teszi a buszra
    - Bemeneti irány esetén a megcélzott periféria
- A tranzakciók **keretekben** utaznak
  - Minden keret 1 ms ideig tart (FS sebességgel: 1500 bájt/keret)
  - Minden keret több tranzakciót is szállíthat
  - Csak *teljes* tranzakciókat szállíthat (lehet, hogy nem lesz kitöltve)

# Adatátviteli módok

- A perifériák sokfélék
  - Van, akinek a kis késleltetés a legfontosabb
  - Van, akinek a hibamentes átvitel
  - Stb.
- **Adatátviteli módok**: hogyan vigyük át az adatot
- USB-ben:
  - Izokron
  - Bulk
  - Control
  - Interrupt

# Izokron adatátviteli mód

- Késleltetésérzékenyeknek
- Cél: folyamatos, ingadozásmentes adatfolyam
- Hangszóró, kamera, stb.
- Jellemzők:
  - Csak FS periféria jöhet szóba
  - A periféria minden keretben garantáltan kap lehetőséget tranzakcióra
    → Senki nem szoríthatja ki onnan
  - A rendszer **nem** garantálja a hibamentességet
    → Úgy sincs idő újraküldeni, ha hibás az adat
  - Rakomány: max. 1023 bájt, overhead: 9 bájt

# Bulk adatátviteli mód

- Nem számít a késleltetés, csak a hibamentes átvitel
- Printer, szkenner, pendrive, stb.
- Elviselik, ha néha kimaradnak 1-1 keretből
- Jellemzők:
  - Csak FS periféria jöhet szóba
  - A késleltetésre nincs garancia
  - Minden hibadetektáló és javító eszközt bevet
  - Adatméret: 8, 16, 32 vagy 64 bájt, overhead: 13 bájt

    → Óriási az overhead!!!

# Control adatátviteli mód

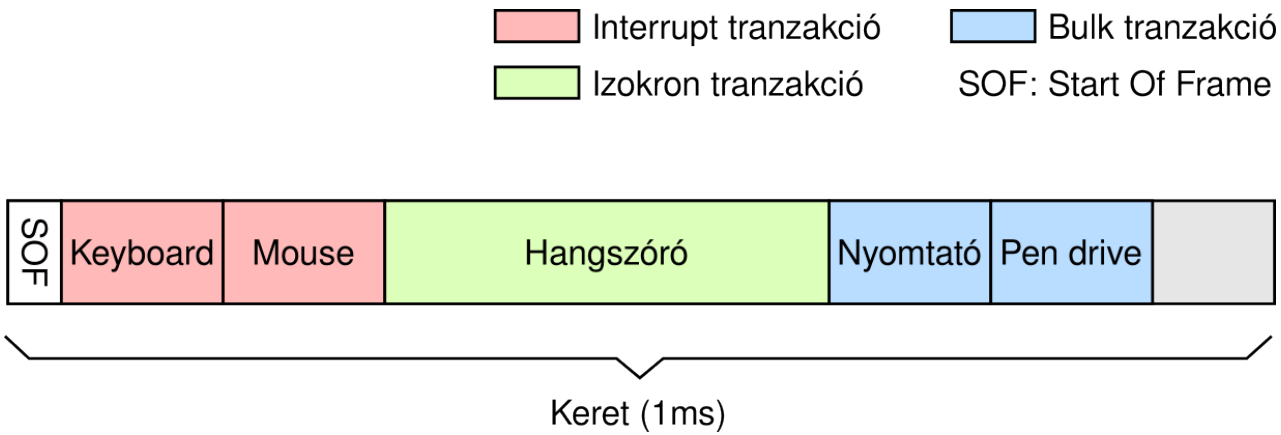
- A perifériák konfigurálását, beállítását szolgálják
- Szabványos és gyártóspecifikus parancsok küldésére
- Jellemzők:
  - LS és FS eszközök is támogatják
  - Ha van átvinni kívánt control tranzakció, a rendszer helyet szorít neki
    - De csak a keretméret 10%-a erejéig
  - Garantált hibamentes átvitel
  - Adatméret:
    - LS esetben: max. 8 bájt, overhead: 63 bájt
    - FS esetben: max. 64 bájt, overhead: 45 bájt

# Interrupt adatátviteli mód

- A periféria nem kezdeményezhet! Interruptot sem!
- **Polling**-al kell kérdezgetni, van-e közlendője

  → Erre szolgál az interrupt tranzakció
- Jellemzők:
  - LS és FS eszközök is támogatják
  - Irányultság: mindig beviteli
  - Az eszköz meg tudja mondani, milyen sűrűn szeretné, hogy kérdezgessék
    - 1 és 255 keretidő között állítható
  - Garantált hibamentes átvitel
    - Hiba esetén ismétlés a következő keretben
  - Adatméret:
    - LS esetben: max. 8 bájt, overhead: 19 bájt
    - FS esetben: max. 64 bájt, overhead: 13 bájt

# A keretek kitöltése



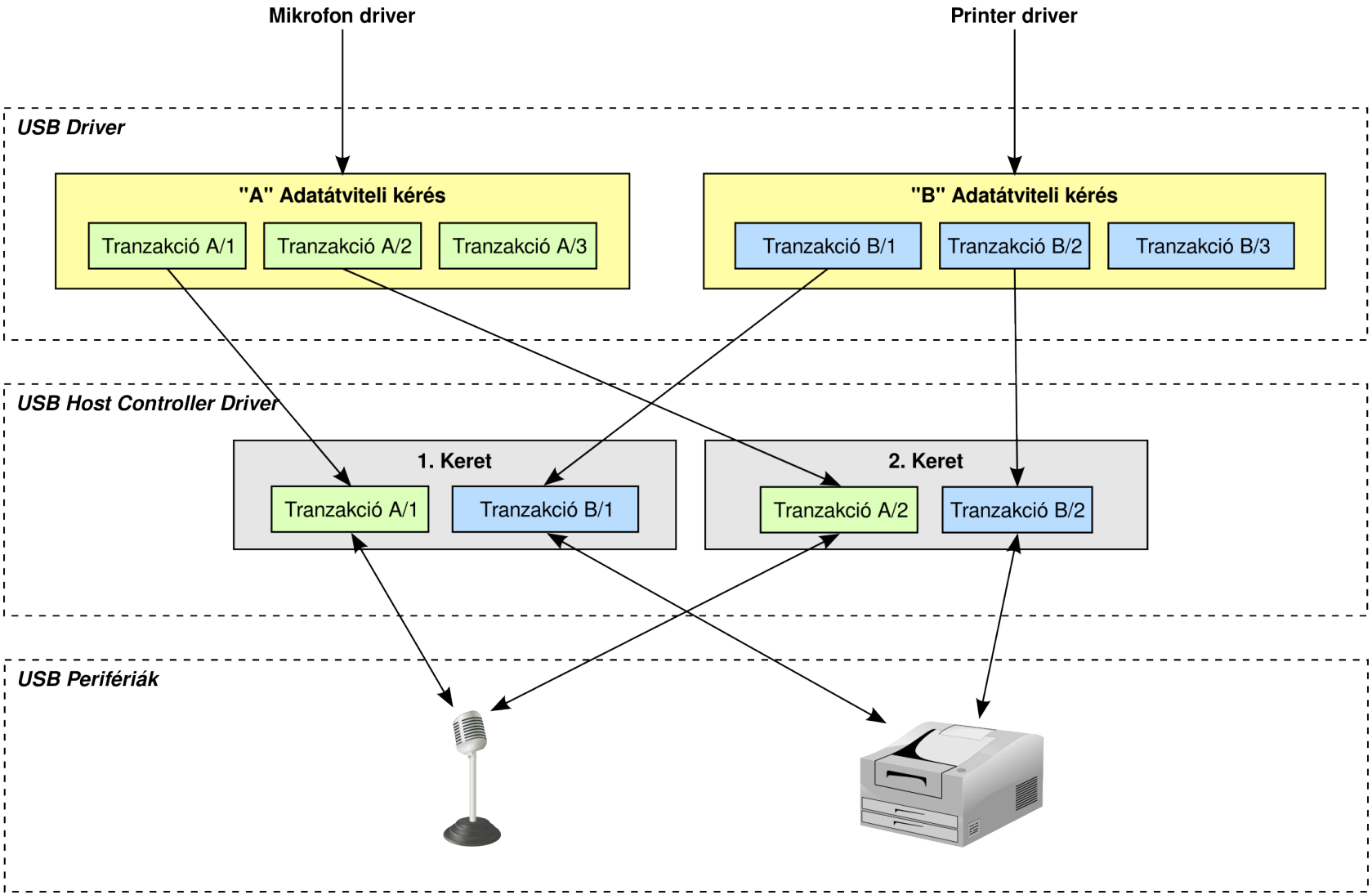

- Feltöltés:
  - Az izokron és interrupt tranzakcióknak elsőbbsége van
    - De nem foglalhatnak többet a keret 90%-ánál
    - Ha csatlakozik egy új periféria, amivel több lenne 90%-nál

      → nem engedik belépni
  - A fennmaradó 10%-ban elsőbbsége van a control tranzakcióknak
  - Maradék: bulk. Ha marad.

# Az adatátvitel folyamata

- **Hardver szereplők:**
  - Host controller a root hub-bal
  - Hub-ok
  - Perifériák
- **Szoftver szereplők:**
  - A periféria driver
  - Az USB driver
  - Az USB Host Controller driver
- **Az adatátvitel:**
  - A periféria driver kezdeményezi
    - Összeállítja az adatátviteli kérést (IRP: I/O Request Packet)
    - Továbbítja az USB driver-nek
  - Az USB driver
    - Az IRP-t feldarabolja a tranzakció max. méretének megfelelő darabokra
    - Továbbítja az USB Host Controller driver-nek
  - A Host Controller Driver
    - Tranzakciókat képez
    - A tranzakciókat keretekhez rendeli
    - 1 ms-onként kibocsát egy keretet

# Az adatátvitel folyamata



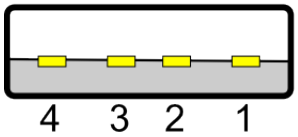
# Tápellátás

- A jelvezetékek mellett „föld” és „+5V” vezeték is van

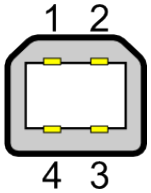
  → Perifériák ebből is megoldhatják a tápellátásukat!
- A root hub minden csatlakozójából max. 500mA vehető fel
  - Ebből gazdálkodhat minden rá kötött eszköz
- Szabvány előírás: minden hub porton legyen min. 100mA
  - Vagyis USB-ről táplálkozó hub-nak max. 4 port-ja lehet (500mA-t kap, 4x100mA kell a kimenő port-okon, +saját tápellátás)
- Vannak saját tápellátással rendelkező hub-ok
  - Minden kimenő port-jukra biztosíthatják a max. 500mA keretet
- USB perifériák (saját táp nélkül):
  - A konfigurációs fázisban max. 100mA-t vehetnek fel
  - A konfigurációs fázisban közlik, mennyi áramot igényelnek
  - Ha nincs ennyi, nem léphetnek be a rendszerbe

# Kábelek, csatlakozók

- 4 vezeték
  - Táp: 0V, 5V
  - Jelátvitel: D+, D–, differenciális, soros átvitel
- Kábelek
  - LS: egyszeresen árnyékolt (vékony, hajlékony, olcsó)
  - FS: kétszeresen árnyékolt, az adatvezetékek megcsavarva
- Csatlakozók
  - Type A: periféria a hubhoz
  - Type B: hub a perifériához (ha külön van)




Type A




Type B

# Konfigurálás

- Rendszerindításkor, új eszköz csatlakozásakor
  - A hub érzékeli, hogy új eszköz csatlakozott
    (a periféria magasba húzza FS esetben a D+-t, LS esetben a D– -t)
  - Az USB driver időnként végigkérdezi a hub-okat, jött-e új periféria az egyik portra. Ha jött, konfigurálja őket:

    1) Szól a hub-nak, hogy reset-elje a portot
    Reset hatására a periféria a 0-s címre hallgat

    2) Kiolvassa a periféria eszközleíró táblázatait
    (0-s címre küldött control tranzakciókkal)
    - Benne vannak a gyártó, típus, eszköz jellege, paraméterei, stb.

    3) Egyedi címet ad a perifériának
    (0-s címre küldött control tranzakcióval)

    4) Ellenőrzi, beléphet-e:
    - Kielégíthető-e a tápellátási igénye
    - Izokron és interrupt adatátviteli igénye kielégíthető-e

    5) Ha minden rendben, az új eszköz beléphet az adatforgalomba

# USB 2.0

# Az USB 2.0

- 2000-ben jelent meg
- Legfontosabb újdonság:
  - Nagy sebességű adatátvitel: 480 Mb/s (high speed, HS)
- Ami nem változott:
  - LS és FS eszközök változatlanul működnek
  - Kábelek, csatlakozók maradtak (kiegészültek)
  - Ugyanazok az adatátviteli módok
  - Tápellátás, konfigurálás ugyanúgy
- Ami változott:
  - Keretidő: 1ms-os keretek helyett 125μs-os **mikró keretek**
  - HS sebességgel 1500 helyett 7500 bájt rakomány / mikró keret
  - Egy vadonatúj **forgalomszabályozás**i eljárás

# Nagy sebességű tranzakciók

- **Izokron adatátviteli mód:**
  - 1 helyett 3 izokron tranzakciót enged meg ugyanazon perifériához
  - A max. 1023 bájt helyett max. 1024 vihető át
- **Bulk adatátviteli mód:**
  - Tranzakciónként 512 bájt lett (fix)
  - Új forgalomszabályozás!
    - USB 1.1:
      - Root hub nem tudja, hogy az eszköz kész-e a fogadásra
      - Leküldte az adatot, és a negatív nyugtából tudta meg
      - Feleslegesen leküldött adat foglalja a kereteket!
    - USB 2.0:
      - PING tranzakció
      - Megkérdezi a perifériát, mehet-e az adat
      - Periféria nem csak negatív vagy pozitív nyugtát küldhet
      - Új nyugtatípus: „vettem, de egy kis időt kérek a következőig”

# Nagy sebességű tranzakciók

- **Control tranzakciók:**
  - Rögzített 64 bájtos méret
  - PING forgalomszabályozás
- **Interrupt tranzakciók**
  - Adatmennyiség max. 1024-re nőtt (8, ill. 64 bájtról)
  - Egy mikró-keretben 3 intrerrupt tranzakció is mehet ugyanazon perifériához

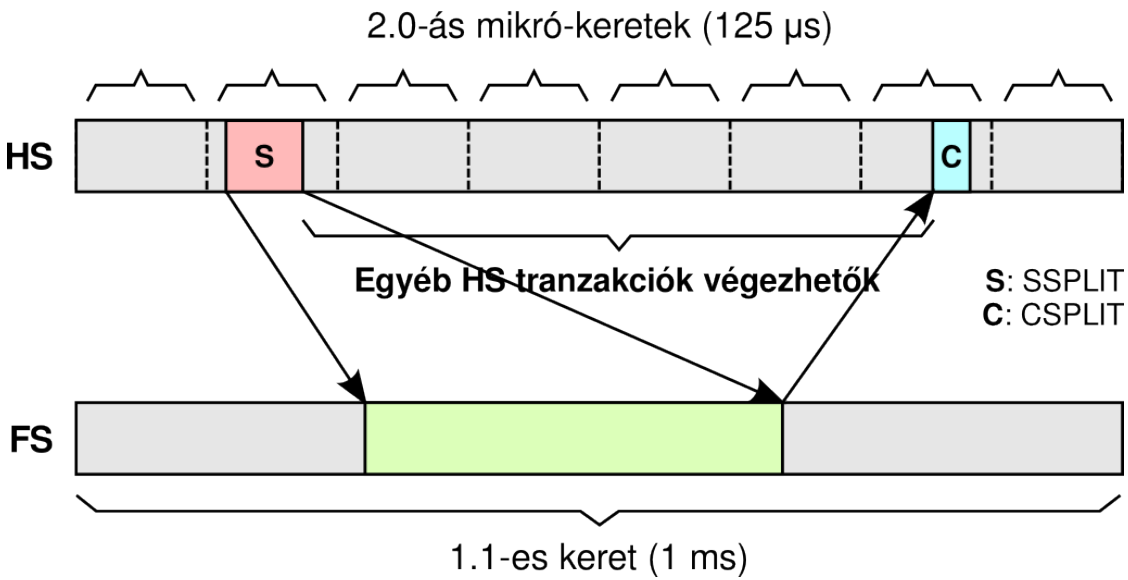
# LS és FS tranzakciók USB 2.0-ban

- USB 2-ben más a keretidő és a bitsebesség

  → LS és FS eszközök alapból nem tudnak részt venni benne!
- A kompatibilitást a hub-ok biztosítják:
  - A rájuk kötött LS és FS eszközöknek azt a látszatot keltik, hogy USB 1.1-re csatlakoznak
- A megoldás részletei:
  - A root hub nagy sebességgel leküldi a hub-nak a tranzakciókat
  - A hub szép lassan LS/FS sebességgel lebeszéli az eszközzel
  - Amikor a root hub szerint már vége kellett, hogy legyen, újra ránéz a végeredményre
  - **Osztott tranzakció**!

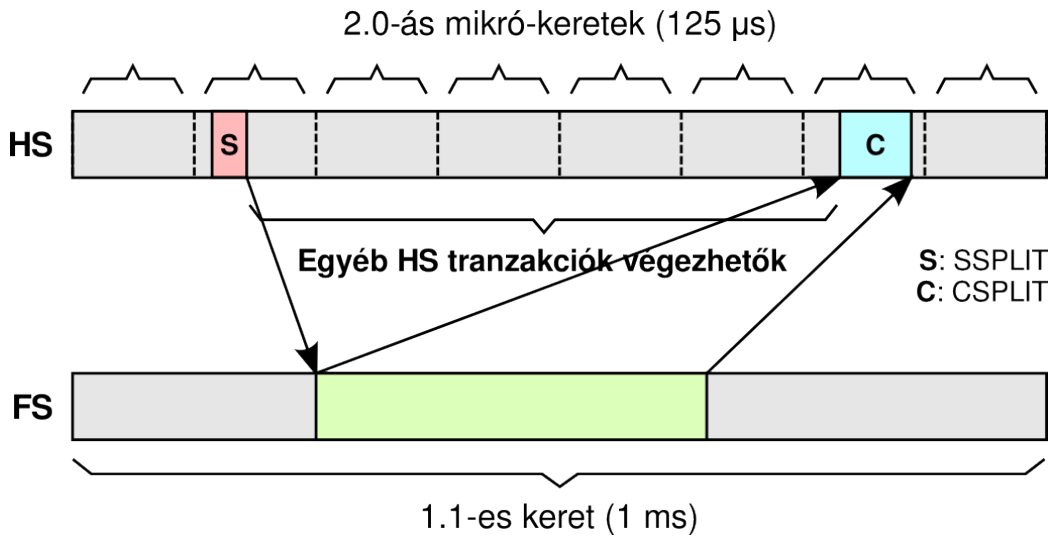
# Osztott kiviteli tranzakció

- A root hub egy SSPLIT tranzakcióban elküldi:
  - A célperiféria címét
  - A neki szánt adatot
- Később a root hub visszanéz: CSPLIT tranzakció
  - Bekéri a hub-tól a nyugtát
- A CSPLIT megérkezéséig a nyugtát a hub tárolja



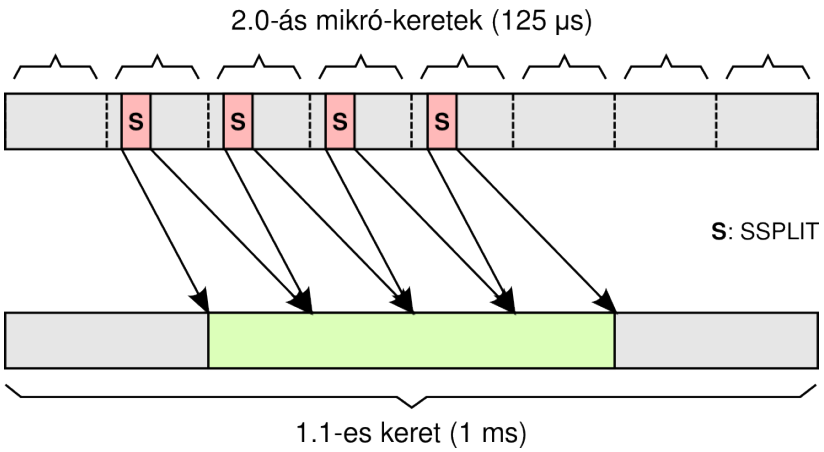
# Osztott bemeneti tranzakció

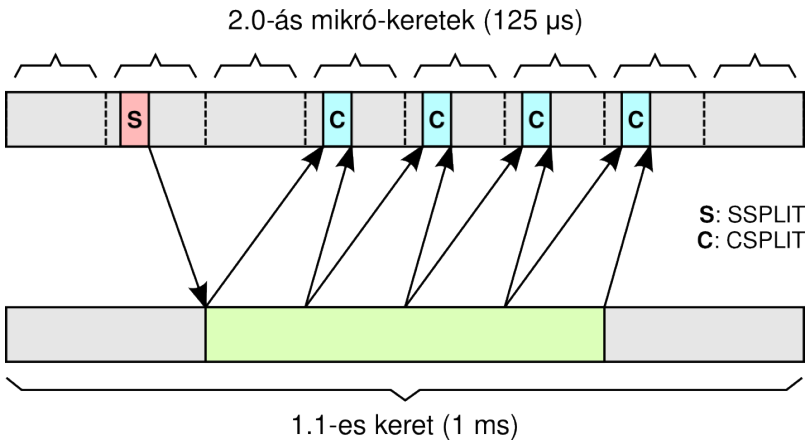
- A root hub egy SSPLIT tranzakcióban elküldi:
  - A célperiféria címét
  - A kérést
- Később a root hub visszanéz: CSPLIT tranzakció
  - Bekéri a hub-tól a periféria által küldött adatokat
  - És a nyugtát
- A CSPLIT megérkezéséig az adatokat és a nyugtát a hub tárolja



# Osztott izokron tranzakciók

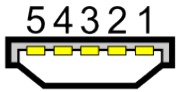
- Cél: egyenletes, állandó adatátviteli ráta
- Az FS izokron tranzakciókat feldarabolva küldi az FS eszköz hub-jához
  - Minden mikró keretben egy darab
  - Így egyenletesebb lesz az átviteli sebesség
  - Így egyenletesebb lesz a mikró keretek terhelése



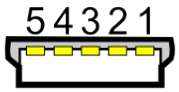



# Új csatlakozók

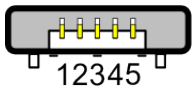
- Mobil eszközök igényeinek megfelelően:
  - Mini
  - Mikró
- Mobil eszközök sajátossága:
  - Hoszt és periféria is lehet
  - Kár lenne ehhez 2 csatlakozó
- Kombinált foglalatok:
  - Van aljzat, amibe az „A” és „B” csatlakozó is belemegy
  - „A”-t beledugva host-ként viselkedik
  - „B”-t beledugva egyszerű periféria
- Lábak:
  - 0V, +5V
  - D+, D–
  - „Szerep” láb: földelve host, nyitva periféria



Mini-A



Mini-B



Micro-A

Micro-B

# USB 3.0

# Az USB 3.0

- 2008-ben jelent meg
- Legfontosabb újdonság:
  - Szuper sebességű adatátvitel: 3.8 Gb/s (super speed, SS)
- Ami nem változott:
  - Fa topológia, hub-okkal a csomópontokban
  - A 4 adatátviteli mód megmaradt
  - Ugyanúgy kell az eszközöket konfigurálni
  - Továbbra is a root hub az egyetlen master

  → Szoftver szinten nincs változás
- Változások:
  - Fizikai szinten szinte minden. Teljesen új rendszer.
  - Kompatibilitás?
    - Megoldás: az USB 3.0 tulajdonképpen két külön busz egymás mellett
      - Egy USB 2.0-ás busz az LS, FS és HS tranzakcióknak
      - Egy külön busz az SS tranzakcióknak

# Szupergyors adatátvitel

- Változások részletesen (SS tranzakciók)
  - **Megszűnt az üzenetszórás**
    - Forgalomirányítással érnek célba az adatok
    - A hub-ok csak arra továbbítják a tranzakciót, amerre a címzett van
  - **Store and forward csomagtovábbítás**
    - A hub megvárja, amíg teljesen megjön hozzá a tranzakció
    - Csak ezután küldi tovább
  - **Egy helyett két érpáron zajlik a kommunikáció**
    - Oda-vissza irányba párhuzamosan
    - A PCI Express-nél látott módon (bitkeverés, 8-ból 10-be kódolás, stb.)
  - **Jobb forgalomszabályozás**
    - Nem kell mindent egyenként nyugtázni (végén közös nyugta)
  - **Jobb energia menedzsment**
    - Finomabb felbontásban szabályozható a perifériák ébrenléte
    - Mivel nincs üzenetszórás, a perifériák többet alhatnak
  - **Áramfelvétel:**
    - A port-onkénti minimum 100 mA-ről 150 mA-re nőtt
    - A maximum 500 mA-ről 900 mA-re nőtt

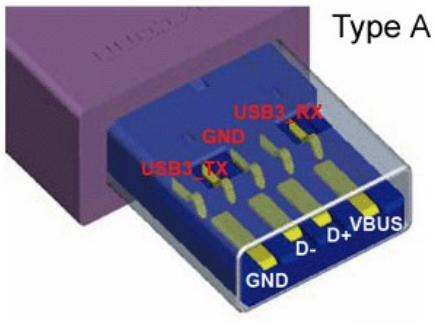
# A tranzakciók célba juttatása

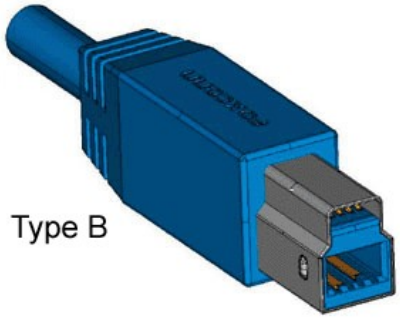
- A tranzakciók útvonalát a root hub határozza meg
  - Ismeri az USB fát
  - Minden tranzakciót kiegészít egy listával
    - Benne van, hogy melyik útba eső hub merre továbbítsa
- Részletek:
  - Az USB fa max. mélysége: 6

    → Egy tranzakció max. 5 hub-on haladhat át
  - Egy USB hub port-jainak max. száma: 15
  - A tranzakciókhoz egy **route string**-et ragaszt
    - Ez 5 db 4 bites szám
    - Az i. köztes hub kiolvassa az i. 4 bites számot
    - Ha 0, akkor maga a hub a tranzakció célpontja
    - Ha nem, akkor a 4 bites szám határozza meg a portot, amerre küldenie kell

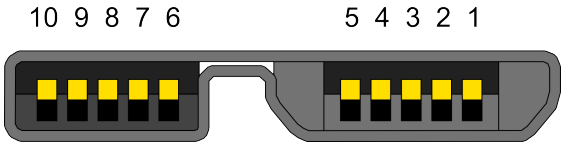
# Új kábelek és csatlakozók

- Új kábel kell: benne fut az USB 2.0 csavart érpárja, és az USB 3.0 két csavart érpárja is → jóval vastagabb (drágább)
- Csatlakozók:
  - 2 részre bontották: USB 2.0 és USB 3.0 részre



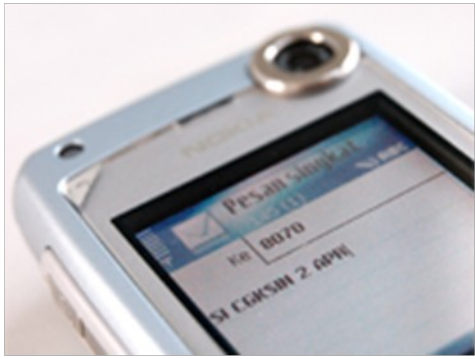




- Érdekesség: Type „B”-ben van még egy táp
  - Ezt a periféria táplálja
  - ??? nem pont őt kell táplálni ???
  - Ez teszi lehetővé a vezeték nélküli USB átvitel használatát!
- Micro is van:



# Összegzés

- Mit **ne** tanuljunk meg
  - A számértékeket (mi hány bájt)
  - A kábelek, csatlakozók lábkiosztását
  - Mi hány amper és mi hány volt
- Mit tanuljunk meg feltétlenül?
  - Az adatátviteli módokat
    - Hogy hogyan igazodik egy csatlakozófelület a sokféle periféria sokféle igényéhez
  - Hogy hogyan tud működni egy busz egyetlen master-rel
  - Hogy hogyan osztozkodnak az eszközök a busz sávszélességén
  - Hogy az USB-ben nincs igazi interrupt (csak polling)
  - Hogy hogyan oldották meg, hogy az USB 2.0 kompatibilis maradjon a régi eszközökkel
  - Hogy mi kellett ahhoz, hogy az USB 3.0 olyan gyors lehessen
  - Hogy hogyan oldották meg, hogy az USB 3.0 kompatibilis maradjon a régi eszközökkel
  - Hogy miként oldották meg az eszközök konfigurálását



# Számítógép Architektúrák

*1. ZH megoldása*

Horváth Gábor
docens
BME Híradástechnikai Tanszék
ghorvath@hit.bme.hu

2012. október 12.
Budapest

# „B” csoport

# Tesztek

- A mágneses adathordozó egy blokkjának állapota a következő: ←←←→→→←→←

  A váltáshoz 1-es bitet, a váltás hiányához 0-ás bitet rendelünk.
- Milyen bitsorozat tartozik a megadott mágneses mintázathoz:
  - **00100111**
- Változtassa meg a harmadik bitet (1-től számolva)!
- Milyen mintázatot kapunk?
  - ←←←←←←→←→

# Tesztek

- Sorolja fel, milyen összetevőkből áll egy szektor olvasási kérés teljes kiszolgálási ideje!
- Melyeket lehet ezek közül átlapolni?
  - **Parancsfeldolgozási késleltetés**
  - **Seek idő**
  - **Forgási idő**
  - **Adatleolvasási idő**
  - **Adatok átküldése az interfészen**
  - **Az utóbbi kettő átlapolható**

# Tesztek

- Ha egy merevlemez minden sávjában ugyanannyi, egyforma méretű szektor található, akkor
  - A külső sávokban nagyobb az adatsűrűség
  - A belső sávokban nagyobb az adatsűrűség
  - Minden sávban egyforma az adatsűrűség

# Tesztek

- Ha egy merevlemez minden sávjában ugyanannyi, egyforma méretű szektor található, akkor
  - A külső sávokban nagyobb az adatsűrűség
  - **A belső sávokban nagyobb az adatsűrűség**
  - Minden sávban egyforma az adatsűrűség

# Tesztek

- Mely 3 koordinátával azonosít egy szektort fizikailag a merevlemez?
  - **Cilinder**
  - **Fej**
  - **Szektorszám**

# Tesztek

- Mi a PCI és a PCI Express viszonya kompatibilitás szempontjából?
  - Szoftver szinten kompatibilisek
  - Hardver szinten kompatibilisek
  - Egyáltalán nem kompatibilisek
  - Minden szempontból kompatibilisek

# Tesztek

- Mi a PCI és a PCI Express viszonya kompatibilitás szempontjából?
  - **Szoftver szinten kompatibilisek**
  - Hardver szinten kompatibilisek
  - Egyáltalán nem kompatibilisek
  - Minden szempontból kompatibilisek

# Tesztek

- A PCI ablakok milyen címtartományra vonatkoznak?
  - Csak a memória címtartományára
  - Csak a periféria címtartományra
  - Mindkettőre vonatkozhatnak, az op. rendszer határozza meg a rendszer indításakor
  - Mindkettőre vonatkozhatnak, a periféria határozza meg, hogy melyik ablak melyikre vonatkozzon

# Tesztek

- A PCI ablakok milyen címtartományra vonatkoznak?
  - Csak a memória címtartományára
  - Csak a periféria címtartományra
  - Mindkettőre vonatkozhatnak, az op. rendszer határozza meg a rendszer indításakor
  - **Mindkettőre vonatkozhatnak, a periféria határozza meg, hogy melyik ablak melyikre vonatkozzon**

# Tesztek

- Milyen arbitrációt használ a PCI?
  - Semmilyet, mert csak 1 master lehet a buszon
  - Daisy chain alapú arbitrációt
  - Párhuzamos arbitrációt

# Tesztek

- Milyen arbitrációt használ a PCI?
  - Semmilyet, mert csak 1 master lehet a buszon
  - Daisy chain alapú arbitrációt
  - **Párhuzamos arbitrációt**

# Tesztek

- Mit jelent a PWM kimenet az Arduino-n?
  - Impulzusszélesség-modulációt
  - Impulzusfrekvencia-modulációt
  - Impulzusszám-modulációt

# Tesztek

- Mit jelent a PWM kimenet az Arduino-n?
  - **Impulzusszélesség-modulációt**
  - Impulzusfrekvencia-modulációt
  - Impulzusszám-modulációt

# Tesztek

- Mely állítások igazak a Daisy chain alapú és a PIC alapú interrupt kezelésre?

| | |
|---|---|
| Daisy chain alapú interrupt kezelés | Körbenforgó interrupt kiszolgálás megvalósítható |
| | Az eszközök prioritása nem változtatható |
| PIC alapú interrupt kezelés | Tetszőlegesen bővíthető |
| | Nem működik, ha egyidejűleg több eszköz is jelez interruptot |

# Tesztek

- Mely állítások igazak a Daisy chain alapú és a PIC alapú interrupt kezelésre?

Daisy chain alapú interrupt kezelés

PIC alapú interrupt kezelés

Körbenforgó interrupt kiszolgálás megvalósítható

Az eszközök prioritása nem változtatható

Tetszőlegesen bővíthető

Nem működik, ha egyidejűleg több eszköz is jelez interruptot

# Tesztek

- Miért jelet gondot, ha a polling intervallum
  - Túl kicsi:
    - **Nagy terhelés a processzornak**
  - Túl nagy:
    - **Lemaradunk egy eseményről (adatvesztés)**

# Tesztek

- Jelölje meg az igaz állításokat!
  - A DMA vezérlőt a periféria programozza fel
  - A DMA vezérlő alkalmazása csökkenti a CPU interrupt terhelését
  - Több DMA vezérlő is lehet egy buszon
  - A DMA vezérlő minden adategység sikeres átvitelét interrupt-tal jelzi a processzornak

# Tesztek

- Jelölje meg az igaz állításokat!
  - A DMA vezérlőt a periféria programozza fel
  - **A DMA vezérlő alkalmazása csökkenti a CPU interrupt terhelését**
  - **Több DMA vezérlő is lehet egy buszon**
  - A DMA vezérlő minden adategység sikeres átvitelét interrupt-tal jelzi a processzornak

# Tesztek

- Forgalomszabályozás hiányában mikor és milyen hibák fordulhatnak elő?

| | |
|---|---|
| Adat egymásrafutás | Küldő gyorsabb, mint a fogadó |
| Adathiány | Fogadó gyorsabb, mint a küldő |

# Tesztek

- Forgalomszabályozás hiányában mikor és milyen hibák fordulhatnak elő?

Adat egymásrafutás → Küldő gyorsabb, mint a fogadó

Adathiány → Fogadó gyorsabb, mint a küldő

# Tesztek

- Tegye ki a "<", a ">", vagy az "=" relációs jelet a RISC és CISC viszonyával kapcsolatban:
  - Utasítások száma:
    - RISC < CISC
  - Regiszterek száma:
    - RISC > CISC
  - Címzési módok száma:
    - RISC < CISC
  - Program mérete:
    - RISC > CISC

# Tesztek

- Egy utasításkészletet ortogonálisnak nevezünk, ha
  - Ha egy adott célra csak egy utasítást tartalmaz
  - Ha egy adott célra sokféle, kényelmesen használható utasítás is rendelkezésre áll
  - Ha minden címzést használó utasítás minden címzési módot használhat
  - Ha minden címzést használó utasítás csak egyetlen címzési módot használhat

# Tesztek

- Egy utasításkészletet ortogonálisnak nevezünk, ha
  - Ha egy adott célra csak egy utasítást tartalmaz
  - Ha egy adott célra sokféle, kényelmesen használható utasítás is rendelkezésre áll
  - **Ha minden címzést használó utasítás minden címzési módot használhat**
  - Ha minden címzést használó utasítás csak egyetlen címzési módot használhat

# Tesztek

- Adat és utasítás egyidejűleg (párhuzamosan) olvasható ki
  - A Neumann architektúrában
  - A Harvard architektúrában
  - A módosított Harvard architektúrában

# Tesztek

- Adat és utasítás egyidejűleg (párhuzamosan) olvasható ki
  - A Neumann architektúrában
  - **A Harvard architektúrában**
  - **A módosított Harvard architektúrában**

# Tesztek

- Mely tulajdonság mely architektúrá(k)ra jellemző?

| | |
|---|---|
| Neumann architektúra | Az adatok és az utasítások külön memóriában vannak |
| Harvard architektúra | Az adatok a programból módosíthatók |
| Módosított Harvard architektúra | Önmódosító program írása lehetséges |
| | Az op. rendszer bármikor betölthet a diszkről egy programot |

# Tesztek

- Mely tulajdonság mely architektúrá(k)ra jellemző?

Neumann architektúra

Harvard architektúra

Módosított Harvard architektúra

Az adatok és az utasítások külön memóriában vannak

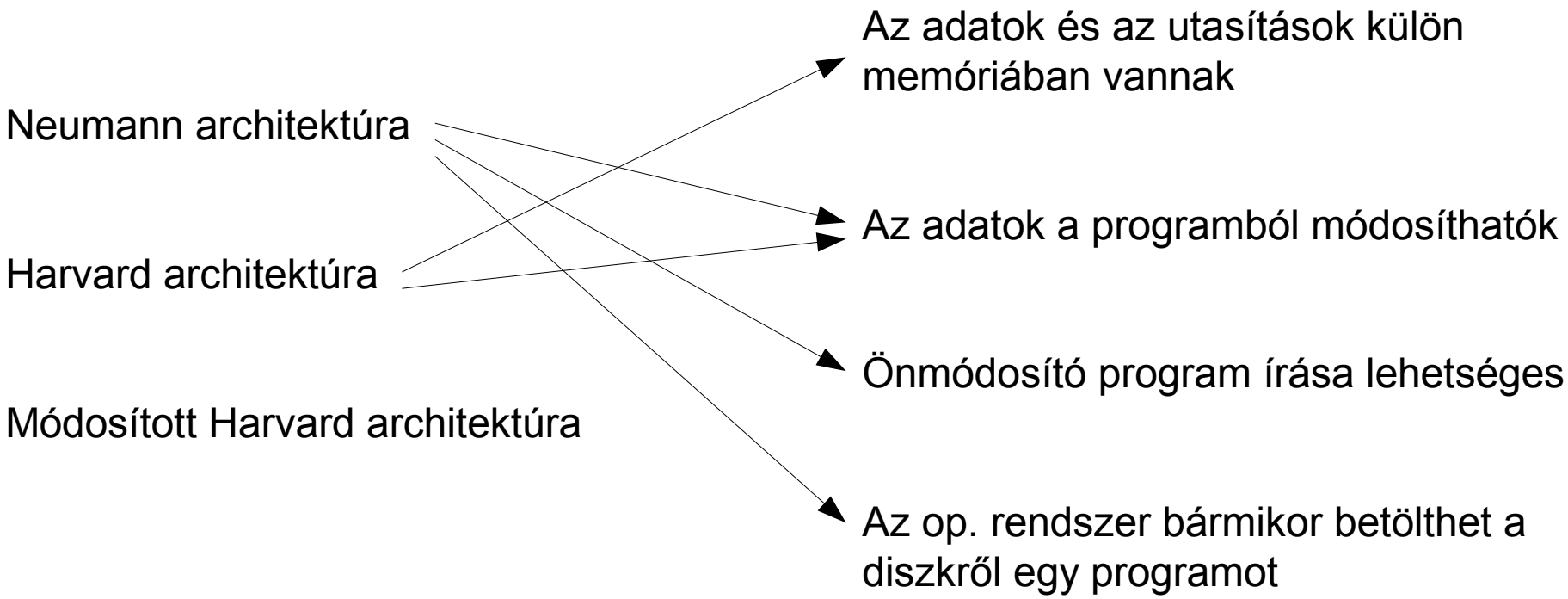
Az adatok a programból módosíthatók

Önmódosító program írása lehetséges

Az op. rendszer bármikor betölthet a diszkről egy programot

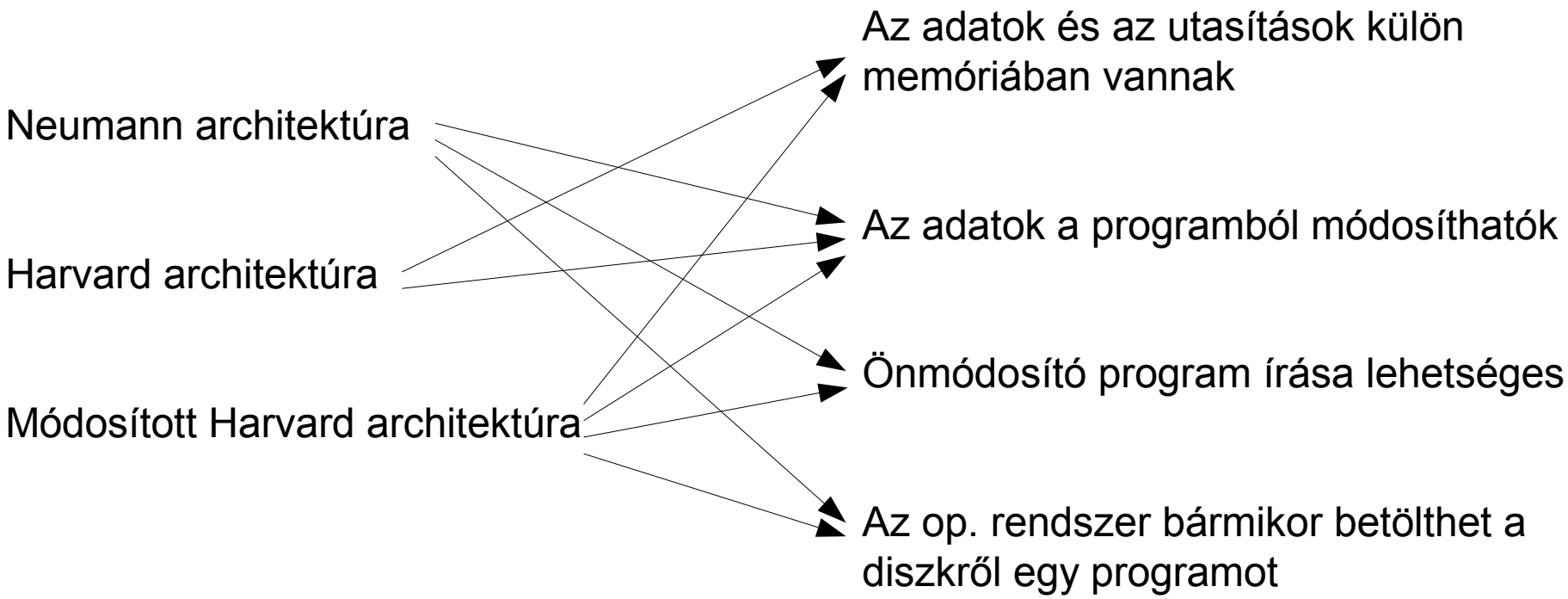
# Tesztek

- Mely tulajdonság mely architektúrá(k)ra jellemző?



# Tesztek

- Mely tulajdonság mely architektúrá(k)ra jellemző?



# Tesztek

- Jelölje be az igaz állítást eredményező lehetőségeket!
- Az igényvezérelt architektúrában az igénygráf irányított élei megadják,
  - hogy egy operandus kiszámolásához mely műveletet kell elvégezni
  - hogy mi a soron következő végrehajtható utasítás
  - hogy egy művelet eredménye mely más műveletek operandusa

# Tesztek

- Jelölje be az igaz állítást eredményező lehetőségeket!
- Az igényvezérelt architektúrában az igénygráf irányított élei megadják,
  - **hogy egy operandus kiszámolásához mely műveletet kell elvégezni**
  - hogy mi a soron következő végrehajtható utasítás
  - hogy egy művelet eredménye mely más műveletek operandusa

# Tesztek

- Milyen információfeldolgozási modell szerint működnek az alábbiak?

Neumann elvű számítógép

Vezérlésáramlásos

Harvard architektúrájú számítógép

Adatáramlásos

Táblázatkezelő program

Igényvezérelt

Funkcionális programozási nyelv

# Tesztek

- Milyen információfeldolgozási modell szerint működnek az alábbiak?

Neumann elvű számítógép → Vezérlésáramlásos

Harvard architektúrájú számítógép → Vezérlésáramlásos

Táblázatkezelő program → Adatáramlásos

Funkcionális programozási nyelv → Igényvezérelt

# Tesztek

- Milyen formális eszközzel írunk le egy programot
  - A vezérlésáramlásos modellben:
    - **Folyamatábra**
  - Az adatáramlásos modellben:
    - **Precedenciagráf / függőségi gráf**

# 1. Számpélda

- Processzorunk órajel frekvenciája 1GHz. A számítógéphez egy 100 Mbit/s segességű hálózati interfészt kötünk, melyen 1500 bájtos (=12000 bit) csomagok közlekednek. A periféria állapotának lekérdezése (mely nem tartalmazza a hálózati csomag tartalmát) 600 órajelet igényel. Az interrupt feldolgozási idő 400 órajel. Ha jött csomag, annak átvitele 5000 órajelet vesz igénybe. Jelenleg épp egy 24 Mbit/s sebességű letöltés van folyamatban a hálózaton.

# 1. Számpélda

- Processzorunk órajel frekvenciája 1GHz. A számítógéphez egy 100 Mbit/s segességű hálózati interfészt kötünk, melyen 1500 bájtos (=12000 bit) csomagok közlekednek. A periféria állapotának lekérdezése (mely nem tartalmazza a hálózati csomag tartalmát) 600 órajelet igényel. Az interrupt feldolgozási idő 400 órajel. Ha jött csomag, annak átvitele 5000 órajelet vesz igénybe. Jelenleg épp egy 24 Mbit/s sebességű letöltés van folyamatban a hálózaton.
- Hány ms-onként kell lekérdezni a perifériát, hogy biztosan ne maradjunk le egy csomag érkezéséről?
  - Pont annyi időnként, amennyi időnként ***elméletileg*** jöhet csomag:
  - (12000 bit / csomag) / ($100*10^6$ bit / s)
    = $0.12*10^{-3}$ s / csomag = **0.12 ms**

# 1. Számpélda

- Processzorunk órajel frekvenciája 1GHz. A számítógéphez egy 100 Mbit/s segességű hálózati interfészt kötünk, melyen 1500 bájtos (=12000 bit) csomagok közlekednek. A periféria állapotának lekérdezése (mely nem tartalmazza a hálózati csomag tartalmát) 600 órajelet igényel. Az interrupt feldolgozási idő 400 órajel. Ha jött csomag, annak átvitele 5000 órajelet vesz igénybe. Jelenleg épp egy 24 Mbit/s sebességű letöltés van folyamatban a hálózaton.
- Mekkora terhelést jelent a periféria kezelése a processzor számára, ha a csomagok érkezését polling-al figyeljük?
  - $0.12*10^{-3}$ s / lekérdezés → 1 / ($0.12*10^{-3}$) lekérdezés / s
    - Minden lekérdezés 600 órajelig tart:
      1 / ($0.12*10^{-3}$) lekérdezés / s * 600 órajel / lek. = $5*10^{6}$ órajel / s
  - Ha ténylegesen van csomag, azt is el kell venni!
    - 24 Mbit/s mellett milyen gyakran jön csomag:
      ($24*10^{6}$ bit / s) / (12000 bit / csomag) = 2000 csomag / s
    - 2000 csomag / s * 5000 órajel / csomag = $10*10^{6}$ órajel / s
  - Teljes terhelés:
    - 1GHz = $10^{9}$ órajel / s
    - ($15*10^{6}$ órajel / s) / ($10^{9}$ órajel / s) = 0.015 = **1.5%**

# 1. Számpélda

- Processzorunk órajel frekvenciája 1GHz. A számítógéphez egy 100 Mbit/s segességű hálózati interfészt kötünk, melyen 1500 bájtos (=12000 bit) csomagok közlekednek. A periféria állapotának lekérdezése (mely nem tartalmazza a hálózati csomag tartalmát) 600 órajelet igényel. Az interrupt feldolgozási idő 400 órajel. Ha jött csomag, annak átvitele 5000 órajelet vesz igénybe. Jelenleg épp egy 24 Mbit/s sebességű letöltés van folyamatban a hálózaton.
- Mekkora terhelést jelent a processzor számára, ha interrupt-ot használunk a csomag érkezések jelzésére?
  - Előbb kiszámoltuk, hogy 24 Mbit/s mellett milyen gyakran jön ténylegesen csomag
  - Minden csomag 1 interrupt, és el is kell venni a perifériától:
    - $(24*10^6$ bit / s) / (12000 bit / csomag) = 2000 csomag / s
    - 2000 csomag / s * (400 órajel / csomag + 5000 órajel / csomag) = $10.8*10^6$ órajel / s
  - Teljes terhelés:
    - 1GHz = $10^9$ órajel / s
    - ($10.8*10^6$ órajel / s) / ($10^9$ órajel / s) = 0.0108 = **1.08%**

# 2. Számpélda

- Egy merevlemez 3 db kétoldalas lemezt tartalmaz, melyek mindegyikén 20000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs. Az átlagos seek idő 4 ms. Mérésekkel megállapítottuk, hogy az egy szektorra vonatkozó olvasási kérések kiszolgálási ideje 10 ms.

# 2. Számpélda

- Egy merevlemez 3 db kétoldalas lemezt tartalmaz, melyek mindegyikén 20000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs. Az átlagos seek idő 4 ms. Mérésekkel megállapítottuk, hogy az egy szektorra vonatkozó olvasási kérések kiszolgálási ideje 10 ms.
- Ha a parancsfeldolgozási késleltetéstől és az interfészen való átviteli időtől eltekintünk, milyen gyorsan forog a lemez? (fordulat/perc-ben megadva)
  - Olvasási kérések (az eltekintések mellett): seek idő + forgási késleltetés
  - → forgási késleltetés = 10 ms - 4 ms = 6 ms
  - Egy véletlen igény forgási késleltetése a teljes körülfordulási idő fele → körülfordulási idő: 12 ms
  - (60000 ms / perc) / (12 ms / fordulat) = **5000 fordulat / perc**

# 2. Számpélda

- Egy merevlemez 3 db kétoldalas lemezt tartalmaz, melyek mindegyikén 20000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs. Az átlagos seek idő 4 ms. Mérésekkel megállapítottuk, hogy az egy szektorra vonatkozó olvasási kérések kiszolgálási ideje 10 ms.
- Adjuk meg CHS koordináta rendszerben a lemez kapacitását!
  - **Cilinderek: 20000** (megegyezik a sávok számával)
  - **Fejek: 6** (mivel ennyi adathordozó réteg van)
  - **Szektorok: 1000** (sávonként értendő)

# 2. Számpélda

- Egy merevlemez 3 db kétoldalas lemezt tartalmaz, melyek mindegyikén 20000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs. Az átlagos seek idő 4 ms. Mérésekkel megállapítottuk, hogy az egy szektorra vonatkozó olvasási kérések kiszolgálási ideje 10 ms.
- A lemez kapacitása bájtokban mérve:
  - Sávok száma összesen: 20000 * 6
  - Szektorok száma: sávok száma * szektorok/sáv = 20000*6*1000
  - Merevlemez kapacitása: szektorok száma * egy szektor mérete = 20000 * 6 * 1000 * 500 = **$6 * 10^{10}$**

# 2. Számpélda

- Egy merevlemez 3 db kétoldalas lemezt tartalmaz, melyek mindegyikén 20000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs. Az átlagos seek idő 4 ms. Mérésekkel megállapítottuk, hogy az egy szektorra vonatkozó olvasási kérések kiszolgálási ideje 10 ms.
- Meddig tart egy szektor leolvasása az adathordozóról? (Ha már ott van a fej)
  - Pont addig, amíg egy szektornyit elfordul a lemez a fej alatt
  - Seek, forgási idő nem számít! A feladat szerint már ott a fej!
  - 12 ms / fordulat → 12 ms / 1000 szektor = **0.012 ms / szektor**

# 2. Számpélda

- Egy merevlemez 3 db kétoldalas lemezt tartalmaz, melyek mindegyikén 20000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs. Az átlagos seek idő 4 ms. Mérésekkel megállapítottuk, hogy az egy szektorra vonatkozó olvasási kérések kiszolgálási ideje 10 ms.
- Ha a parancsfeldolgozási idő 0.1 ms, az adatátviteli interfész sebessége pedig $50*10^6$ bájt/s, akkor átlagosan mennyi ideig tart egy 2000 bájtos kérés teljes kiszolgálása?
  - Parancsfeldolgozás: 0.1 ms
  - Seek: 4 ms
  - Forgási dő: 6 ms (véletlen pozícióba átlag egy fél fordulat kell)
  - Leolvasási idő: 0.012 ms / szektor, de 2000 bájt az 4 szektor, vagyis 4 * 0.012 ms = 0.048 ms
  - Adatátviteli idő: átlapolódás miatt csak az utolsó szektor átviteli ideje kell: (500 bájt / szektor) / ($50*10^6$ bájt / s) = $10^{-5}$ s / szektor = 0.01 ms
  - Összesen: 0.1 + 4 + 6 + 0.048 + 0.01 = **10.158 ms**

# „A” csoport

# Tesztek

- Mikor hajtható végre egy művelet a tanult 3 információfeldolgozási modellben?

| | |
|---|---|
| Vezérlésáramlásos modell | Amikor szükség van az eredményére |
| Adatáramlásos modell | Amikor a vezérlő token odaér |
| Igényvezérelt modell | Amikor minden operandusa rendelkezésre áll |

# Tesztek

- Mikor hajtható végre egy művelet a tanult 3 információfeldolgozási modellben?

| | |
|---|---|
| Vezérlésáramlásos modell | Amikor szükség van az eredményére |
| Adatáramlásos modell | Amikor a vezérlő token odaér |
| Igényvezérelt modell | Amikor minden operandusa rendelkezésre áll |

# Tesztek

- Hogyan kell jelezni a program leírásában a párhuzamosítható részeket
  - A vezérlésáramlásos modellben:
    - **Fork/Join primitívekkel**
  - Az adatáramlásos modellben:
    - **Sehogy**
  - Az igényvezérelt modellben:
    - **Sehogy**

# Tesztek

- Jelölje be az igaz állítást eredményező lehetőségeket!
- A precedenciagráf irányított élei megadják,
  - hogy egy operandus kiszámolásához mely műveletet kell elvégezni
  - hogy mi a soron következő végrehajtható utasítás
  - hogy egy művelet eredménye mely más műveletek operandusa

# Tesztek

- Jelölje be az igaz állítást eredményező lehetőségeket!
- A precedenciagráf irányított élei megadják,
  - hogy egy operandus kiszámolásához mely műveletet kell elvégezni
  - hogy mi a soron következő végrehajtható utasítás
  - **hogy egy művelet eredménye mely más műveletek operandusa**

# Tesztek

- Melyek a Neumann architektúra főbb komponensei?
  - Processzor
  - Cache
  - Perifériák
  - Memória
  - Rendszerbusz
  - Háttértár

# Tesztek

- Melyek a Neumann architektúra főbb komponensei?
  - **Processzor**
  - Cache
  - **Perifériák**
  - **Memória**
  - Rendszerbusz
  - Háttértár

# Tesztek

- Honnan tudja egy Neumann elvű számítógép processzora, hogy egy memóriaeléréskor a meghivatkozott memóriabeli objektumot milyen típusúnak tekintse?
  - Onnan, hogy a memóriában az objektumok típusa is le van tárolva
  - Az objektumok típusa nincs ugyan letárolva, de tudja, milyen típusú, hiszen ő tette bele
  - Onnan, hogy az utasítás, amivel a program a memóriához nyúl, milyen típusú adatra vonatkozik

# Tesztek

- Honnan tudja egy Neumann elvű számítógép processzora, hogy egy memóriaeléréskor a meghivatkozott memóriabeli objektumot milyen típusúnak tekintse?
  - Onnan, hogy a memóriában az objektumok típusa is le van tárolva
  - Az objektumok típusa nincs ugyan letárolva, de tudja, milyen típusú, hiszen ő tette bele
  - **Onnan, hogy az utasítás, amivel a program a memóriához nyúl, milyen típusú adatra vonatkozik**

# Tesztek

- Az összetartozó párok összekötésével jelezze a fix és a változó utasításkódolás főbb tulajdonságait!

| | |
|---|---|
| Fix hosszúságú utasításkódolás | Helytakarékosabb, rövidebb programhoz vezet |
| Változó hosszúságú utasításkódolás | A processzor gyorsabban tudja dekódolni |

# Tesztek

- Az összetartozó párok összekötésével jelezze a fix és a változó utasításkódolás főbb tulajdonságait!

Fix hosszúságú utasításkódolás

Változó hosszúságú utasításkódolás

Helytakarékosabb, rövidebb programhoz vezet

A processzor gyorsabban tudja dekódolni

# Tesztek

- A hexadecimális 0x0A0B0C0D számot adja meg
  - Little endian bájtsorrenddel tárolva:
    - **0D0C0B0A**
  - Big endian bájtsorrenddel tárolva:
    - **0A0B0C0D**

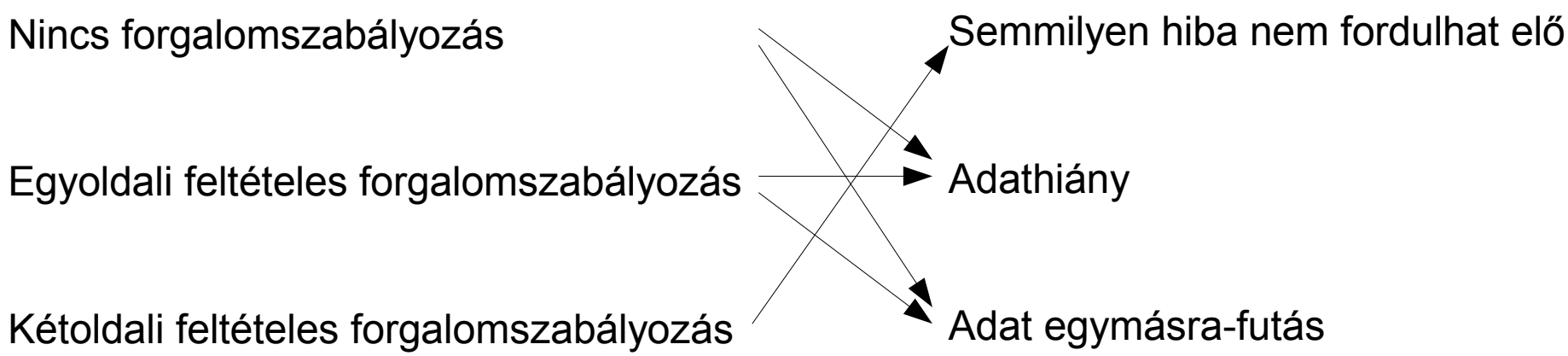
# Tesztek

- Melyik forgalomszabályozás mellett milyen hibák fordulhatnak elő az adatátvitel során:

| | |
|---|---|
| Nincs forgalomszabályozás | Semmilyen hiba nem fordulhat elő |
| Egyoldali feltételes forgalomszabályozás | Adathiány |
| Kétoldali feltételes forgalomszabályozás | Adat egymásra-futás |

# Tesztek

- Melyik forgalomszabályozás mellett milyen hibák fordulhatnak elő az adatátvitel során:



# Tesztek

- Tegyük fel, hogy a CPU támogatja az interruptok használatát, és ez engedélyezve is van. Mely állítások igazak?
  - A lassú perifériákat célszerű polling-gal kezelni, mert úgy kisebb a processzor terhelése
  - A lassú perifériákat célszerű polling-gal kezelni, mert úgy gyorsabban értesülünk a periféria megváltozott állapotáról
  - A gyors perifériát célszerű interrupt-tal kezelni, mert úgy kisebb a processzor terhelése

# Tesztek

- Tegyük fel, hogy a CPU támogatja az interruptok használatát, és ez engedélyezve is van. Mely állítások igazak?
  - A lassú perifériákat célszerű polling-gal kezelni, mert úgy kisebb a processzor terhelése
  - A lassú perifériákat célszerű polling-gal kezelni, mert úgy gyorsabban értesülünk a periféria megváltozott állapotáról
  - **A gyors perifériát célszerű interrupt-tal kezelni, mert úgy kisebb a processzor terhelése**

# Tesztek

- Van egy bemeneti perifériánk, és egy processzorunk. Milyen forgalomszabályozást célszerű használni az adatátviteli hibák elkerülésére, ha
  - A periféria sebessége befolyásolható:
    - **Kétoldali feltételes forgalomszabályozás**
  - A periféria sebessége nem befolyásolható:
    - **Egyoldali feltételes forgalomszabályozás**

# Tesztek

- Mely állítások igazak a Daisy chain alapú és a párhuzamos arbitrációra?

Tud úgy elromlani egy periféria, hogy másokat elzárjon a busztól

Daisy chain alapú arbitráció

Ha elromlik az arbiter, leáll a rendszer

Párhuzamos arbitráció

Igazságosabb kiszolgálást is lehetővé tesz

Könnyen bővíthető új perifériával

Nem működik, ha egyidejűleg több eszköz is bejelenti igényét a buszra

# Tesztek

- Mely állítások igazak a Daisy chain alapú és a párhuzamos arbitrációra?

Daisy chain alapú arbitráció

Párhuzamos arbitráció

Tud úgy elromlani egy periféria, hogy másokat elzárjon a busztól

Ha elromlik az arbiter, leáll a rendszer

Igazságosabb kiszolgálást is lehetővé tesz

Könnyen bővíthető új perifériával

Nem működik, ha egyidejűleg több eszköz is bejelenti igényét a buszra

# Tesztek

- Milyen eszköz található az Arduino analóg bemenetein?
  - Analóg-digitális átalakító (ADC)
  - Digitális-analóg átalakító (DAC)
  - PWM generátor

# Tesztek

- Milyen eszköz található az Arduino analóg bemenetein?
  - **Analóg-digitális átalakító (ADC)**
  - Digitális-analóg átalakító (DAC)
  - PWM generátor

# Tesztek

- Milyen forgalomszabályozást használ a PCI?
  - Semmilyet
  - Az egyik fél jelezni tudja, ha kész az adatátvitelre
  - Mindkét fél jelezni tudja, ha kész az adatátvitelre

# Tesztek

- Milyen forgalomszabályozást használ a PCI?
  - Semmilyet
  - Az egyik fél jelezni tudja, ha kész az adatátvitelre
  - **Mindkét fél jelezni tudja, ha kész az adatátvitelre**

# Tesztek

- Hogyan tudja használni egy PCI eszköz a rendszerszintű DMA vezérlőt?
  - A csatolófelületen erre szolgáló lábak segítségével
  - Speciális tranzakciók segítségével
  - Sehogy

# Tesztek

- Hogyan tudja használni egy PCI eszköz a rendszerszintű DMA vezérlőt?
  - A csatolófelületen erre szolgáló lábak segítségével
  - Speciális tranzakciók segítségével
  - **Sehogy**

# Tesztek

- Mely adatátviteli lehetőségeket támogatja a PCI és a PCI Express? (Kezdeményező → megszólított)
  - CPU → PCI periféria
  - PCI periféria → CPU
  - PCI periféria → memória
  - memória → PCI periféria
  - PCI periféria → PCI periféria

# Tesztek

- Mely adatátviteli lehetőségeket támogatja a PCI és a PCI Express? (Kezdeményező → megszólított)
  - **CPU → PCI periféria**
  - PCI periféria → CPU
  - **PCI periféria → memória**
  - memória → PCI periféria
  - **PCI periféria → PCI periféria**

# Tesztek

- Mely 3 koordinátával lehet azonosítani egy pontot egy több adathordozó réteggel ellátott forgólemezes adattárolón?
  - **Radiális távolság**
  - **Egy fix referenciához képesti szög**
  - **Az adathordozó réteg száma**

# Tesztek

- Ha egy merevlemez minden sávjában ugyanannyi, egyforma méretű szektor található, akkor
  - A külső sáv szektorait gyorsabban kell leolvasni
  - A belső sáv szektorait gyorsabban kell leolvasni
  - Minden szektort ugyanannyi ideig tart leolvasni

# Tesztek

- Ha egy merevlemez minden sávjában ugyanannyi, egyforma méretű szektor található, akkor
  - A külső sáv szektorait gyorsabban kell leolvasni
  - A belső sáv szektorait gyorsabban kell leolvasni
  - **Minden szektort ugyanannyi ideig tart leolvasni**

# Tesztek

- Rajzolja fel a merevlemez egy szektorának felépítését!
  - Rés
  - Előtag
  - „Adat kezdődik” jel
  - Adatok
  - Hibadetektáló kód (CRC)
  - Hibajavító kód (ECC)
  - Töltelék terület

# Tesztek

- A mágneses adathordozó egy blokkjának állapota a következő: →←←←→→←→←

  A váltáshoz 1-es bitet, a váltás hiányához 0-ás bitet rendelünk.
- Milyen bitsorozat tartozik a megadott mágneses mintázathoz:
  - **10010111**
- Változtassa meg a harmadik bitet (1-től számolva)!
- Milyen mintázatot kapunk?
  - →←←→←←→←→

# 1. Számpélda

- 1. Processzorunk órajel frekvenciája 100MHz. A számítógéphez egy billentyűzetet kötünk, melyen átlagosan 10 karaktert ütnek le másodpercenként, de két leütés akár 50 ms-onként is követheti egymást. A billentyűzet állapotának lekérdezése (mely tartalmazza a lenyomott gomb kódját is) 500 órajelet igényel. Az interrupt feldolgozási idő ezen felül még 100 órajel.

# 1. Számpélda

- 1. Processzorunk órajel frekvenciája 100MHz. A számítógéphez egy billentyűzetet kötünk, melyen átlagosan 10 karaktert ütnek le másodpercenként, de két leütés akár 50 ms-onként is követheti egymást. A billentyűzet állapotának lekérdezése (mely tartalmazza a lenyomott gomb kódját is) 500 órajelet igényel. Az interrupt feldolgozási idő ezen felül még 100 órajel.
- Hányszor kell másodpercenként lekérdezni a billentyűzetet, hogy ne maradjunk le semmiről?
  - 1 lekérdezés / 50 ms = 1 lekérdezés / 0.05 s =
    = **20 lekérdezés / s**

# 1. Számpélda

- 1. Processzorunk órajel frekvenciája 100MHz. A számítógéphez egy billentyűzetet kötünk, melyen átlagosan 10 karaktert ütnek le másodpercenként, de két leütés akár 50 ms-onként is követheti egymást. A billentyűzet állapotának lekérdezése (mely tartalmazza a lenyomott gomb kódját is) 500 órajelet igényel. Az interrupt feldolgozási idő ezen felül még 100 órajel.
- Mekkora terhelést jelent a processzor számára, ha a billentyűzet kezelésére polling-ot használ?
  - 20 lekérdezés / s * 500 órajel / lekérdezés = $10^4$ órajel / s
  - 100MHz = $10^8$ órajel/s
  - Terhelés: ($10^4$ órajel / s) / ($10^8$ órajel/s) = $10^{-4}$ = **0.01%**

# 1. Számpélda

- 1. Processzorunk órajel frekvenciája 100MHz. A számítógéphez egy billentyűzetet kötünk, melyen átlagosan 10 karaktert ütnek le másodpercenként, de két leütés akár 50 ms-onként is követheti egymást. A billentyűzet állapotának lekérdezése (mely tartalmazza a lenyomott gomb kódját is) 500 órajelet igényel. Az interrupt feldolgozási idő ezen felül még 100 órajel.
- Mekkora terhelést jelent a processzor számára, ha a billentyűzet kezelésére interrupt-ot használ?
  - 10 esemény / s * (500 órajel / esemény + 100 órajel / esemény) = $6*10^3$ órajel / s
  - 100MHz = $10^8$ órajel/s
  - Terhelés: ($6*10^3$ órajel / s) / ($10^8$ órajel/s) = $6*10^{-5}$ = **0.006%**

# 2. Számpélda

- Egy merevlemez 2 db kétoldalas lemezt tartalmaz, melyek mindegyikén 30000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs, a lemezek állandó 10000 fordulat/perc-el forognak.

# 2. Számpélda

- Egy merevlemez 2 db kétoldalas lemezt tartalmaz, melyek mindegyikén 30000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs, a lemezek állandó 10000 fordulat/perc-el forognak.
- Adjuk meg CHS koordináta rendszerben a lemez kapacitását!
  - **Cilinderek: 30000** (megegyezik a sávok számával)
  - **Fejek: 4** (mivel ennyi adathordozó réteg van)
  - **Szektorok: 1000** (sávonként értendő)

# 2. Számpélda

- Egy merevlemez 2 db kétoldalas lemezt tartalmaz, melyek mindegyikén 30000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs, a lemezek állandó 10000 fordulat/perc-el forognak.
- A lemez kapacitása bájtokban mérve:
  - Sávok száma összesen: 30000 * 4
  - Szektorok száma: sávok száma * szektorok/sáv = 30000*4*1000
  - Merevlemez kapacitása: szektorok száma * egy szektor mérete = 30000 * 4 * 1000 * 500 = **$6 * 10^{10}$**

# 2. Számpélda

- Egy merevlemez 2 db kétoldalas lemezt tartalmaz, melyek mindegyikén 30000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs, a lemezek állandó 10000 fordulat/perc-el forognak.
- Mennyi ideig tart, míg a lemez megtesz egy fordulatot?
  - (60 sec / perc) / (10000 fordulat / perc) =

    $6*10^{-3}$ sec/fordulat = **6 ms / fordulat**

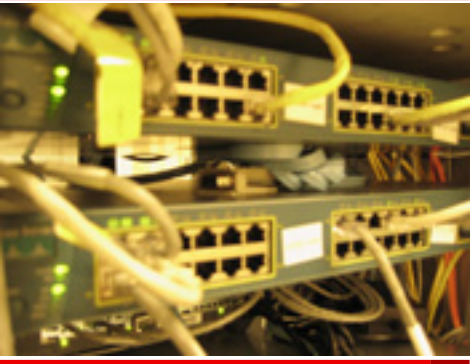
# 2. Számpélda

- Egy merevlemez 2 db kétoldalas lemezt tartalmaz, melyek mindegyikén 30000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs, a lemezek állandó 10000 fordulat/perc-el forognak.
- Meddig tart egy szektor leolvasása az adathordozóról? (Ha már ott van a fej)
  - Pont addig, amíg egy szektornyit elfordul a lemez a fej alatt
  - Seek, forgási idő nem számít! A feladat szerint már ott a fej!
  - 6 ms / fordulat → 6 ms / 1000 szektor = **0.006 ms / szektor**

# 2. Számpélda

- Egy merevlemez 2 db kétoldalas lemezt tartalmaz, melyek mindegyikén 30000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs, a lemezek állandó 10000 fordulat/perc-el forognak.
- Ha a parancsfeldolgozási idő 0.1 ms, az átlagos seek idő pedig 5 ms, az adatátviteli interfész sebessége pedig $50*10^6$ bájt/s, akkor átlagosan mennyi ideig tart egy 2000 bájtos kérés teljes kiszolgálása?
  - Parancsfeldolgozás: 0.1 ms
  - Seek: 5 ms
  - Forgási dő: 3 ms (véletlen pozícióba átlag egy fél fordulat kell)
  - Leolvasási idő: 0.006 ms / szektor, de 2000 bájt az 4 szektor, vagyis 4 * 0.006 ms = 0.024 ms
  - Adatátviteli idő: Kiszámoljuk egy szektor átviteli idejét az interfészen: (500 bájt / szektor) / ($50*10^6$ bájt / s) = $10^{-5}$ s / szektor = 0.01 ms
    Átlapoljuk a leolvasási idővel. Vegyük észre, hogy az interfész lassabb, mint a leolvasás. Emiatt az adatátvitel+leolvasás összesen = Első szektor leolvasása + mind a 4 szektor átvitele = 0.024 ms + 0.04 ms = 0.044 ms
  - Összesen: 0.1 + 5 + 3 + 0.044 = **8.144 ms**

MŰEGYETEM 1782



BME

# MEMÓRIA TECHNOLÓGIÁK

*Számítógép-architektúrák*
*4. gyakorlat*


Dr. Lencse Gábor
tudományos főmunkatárs
BME Híradástechnikai Tanszék
lencse@hit.bme.hu

2011. október 3.,
Budapest

# Tartalom

- Emlékeztető: mit kell már tudnom a memóriákról?
- SRAM és DRAM felépítése és ennek következményei
  - SRAM felépítése és működése
  - DRAM felépítése és működése
  - SRAM és DRAM összehasonlítása
- DRAM fejlődése
  - Aszinkron DRAM típusok
  - Szinkron DRAM típusok
  - Legfontosabb időzítések szinkron DRAM-oknál
- Memória modulok
- Feladatok
- Összefoglalás

# EMLÉKEZTETŐ: MIT KELL MÁR TUDNOM A MEMÓRIÁKRÓL?

# Emlékezető – ellenőrzés céljából

- Ismertnek tételezzük fel a alapvető memóriatípusokat, illesztési felületüket, műveleiket és azok idő diagramjait.
  - ROM (maszk programozott, PROM, EPROM): olvasás
  - SRAM: írás, olvasás
  - DRAM: írás, olvasás, frissítés
  - NVRAM: írás, olvasás, store, recall

*Akinek hiányosságai vannak, olvassa el a róluk szóló részt a*

***Memória technológiák***

*anyag elején a weben található jegyzetben!*

( `http://sites.google.com/site/szgarch` )

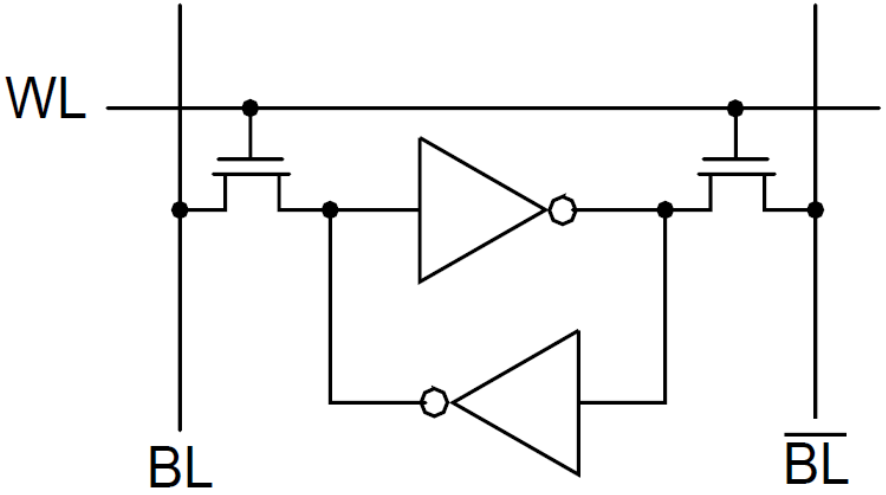

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SRAM ÉS DRAM FELÉPÍTÉSE ÉS ENNEK KÖVETKEZMÉNYEI


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM egy cellájának felépítése – 1

- A statikus RAM egy cellája (ami egy bit tárolására képes) – logikai áramkörként tekintve – két keresztbe kötött inverterből áll:
  - az egyik inverter bemenete a bitet,
  - a másiké annak inverzét tárolja.

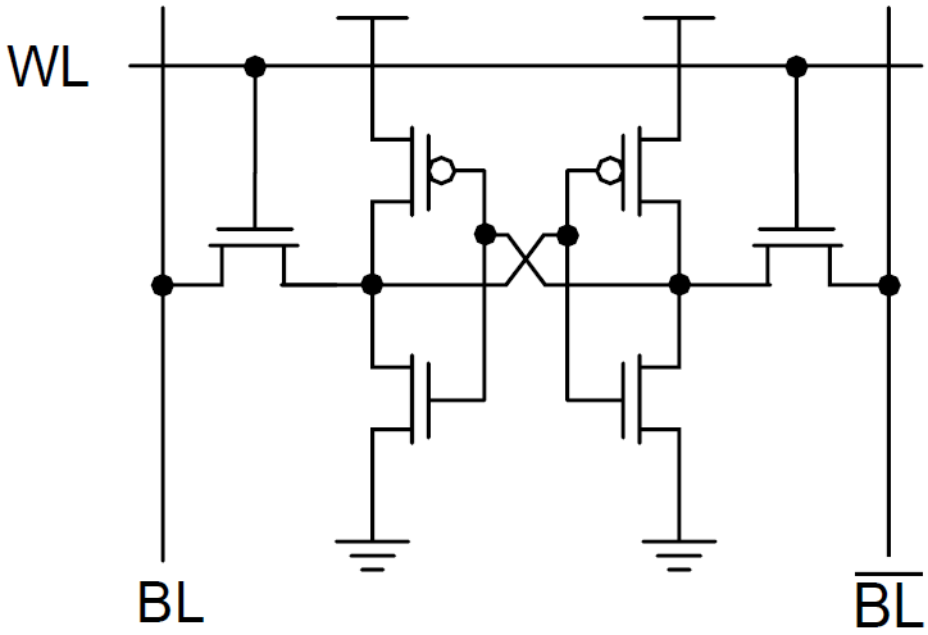



- Ez az áramkör mindaddig képes megőrizni a bit értékét, amíg tápfeszültséggel ellátjuk.
- A cella kiolvasását és írását további két tranzisztorral valósítják meg, ezeket *hozzáférési* (access) tranzisztoroknak hívjuk.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM egy cellájának felépítése – 2

- Mivel az inverterek megvalósításához is 2-2 tranzisztor szükséges, így cellánként összesen 6 tranzisztort használnak fel.




- Az ábrán WL (**W**ord **L**ine) a *szóvezetéket*, BL (**B**it **L**ine) a *bitvezetéket* jelenti.

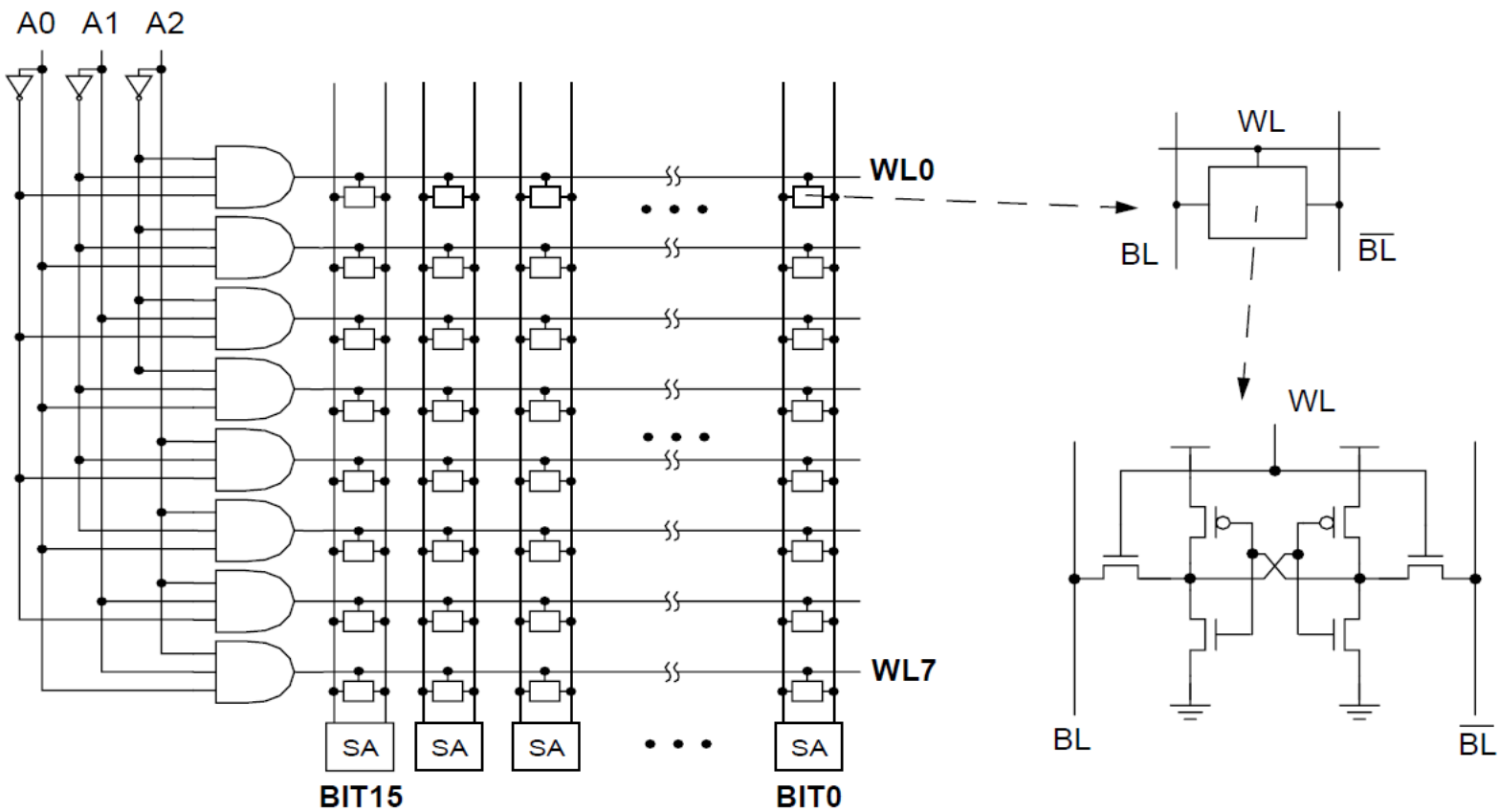
# Az SRAM felépítése – 1

- A cellákat mátrixba rendezik. Az egyszerűség kedvéért most a *szószervezésű* megoldással foglalkozunk; legyen a *címbitek* száma $n$, így a *szavak száma* $k=2^n$, a *szavak szélessége* pedig $m$ bit, ekkor a mátrix $k$ sorból és $m$ oszlopból áll.
- A következő dián egyszerű példaként egy nagyon kis méretű memóriát látunk a következő értékekkel: címbitek száma $n$=3, dekódolt címvezetékek (az ábrán $WL_0$-$WL_7$ szóvezetékek) száma $k$=8, a szószélesség $m$=16.
  - A bitmezőtől balra egy inverterekkel és ÉS kapukkal megvalósított 3/8-as dekódert látunk; a cellák tartalmát pedig a jobb oldalon kinagyítva és kirészletezve találjuk.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM felépítése – 2

- Az *érzékelő erősítőket* SA-val (sense amplifier) jelöltük.
- A rajzon nem szerepelnek, de minden bitvezetékhez tartoznak még a *előfeszítő* (precharge) áramkörök is.



# Az SRAM olvasása – 1

1. Az előfeszítő áramkörök a ponált és negált bitvezetékeket egyaránt logikai 1 szintre húzzák fel, majd a bitvezetékekről lekapcsolódnak. A bitvezetékek kapacitása miatt a bitvezetékek feszültségszintje megmarad.

   (Ez azért hasznos, mert a celláknak könnyebb/gyorsabb egy bitvezetéket 0-ra lehúzni, mint 1-re felhúzni.)

1. Ha az *i*. memóriaszót szeretnénk kiolvasni ($0<=i<2^n$), akkor a $WL_i$ szóvezetékre logikai 1 értéket adunk.

   (Ezt az ábrán a bitmezőtől balra található dekóderrel valósítjuk meg.)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM olvasása – 2

3. Az *i*. memóriaszó összes bitjének hozzáférési tranzisztorain át a bitek értéke megjelenik a bitvezetékeken:
   - amelyik bit értéke Q=1, ott BL értéke 1 marad, /BL-t pedig 0-ra húzza le a bit inverze (/Q);
   - a Q=0 értékű bitnél pedig a helyzet éppen fordított.
3. Az érzékelő erősítők érzékelik BL és /BL különbségét, és a kimenetükön rendelkezésünkre áll az *i*. memóriaszó bitjeinek értéke.

   (Minél érzékenyebbek, annál gyorsabb a kiolvasás.)

- Az olvasás a kiolvasott memóriaszó értékét nem változtatja meg.
- A bitek felismerését segíti (és gyorsítja), hogy nem abszolút jelszinteket, hanem BL és /BL különbségét kell felismerni.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM írása – 1

1. A ponált (BL) és negált (/BL) bitvezetékekre ráadjuk a beírandó bitek logikai értékének megfelelő feszültséget:
   - ha a *j*. bit értéke 1, akkor $BL_j$=1, $/BL_j$=0;
   - ha a *j*. bit értéke 0, akkor $BL_j$=0, $/BL_j$=1.
1. Ha az *i*. memóriaszót szeretnénk írni ($0<=i<2^n$), akkor a $WL_i$ szóvezetékre logikai 1 értéket adunk.
3. Az *i*. memóriaszó összes bitjének értéke a hozzáférési tranzisztorain keresztül felveszi a bitvezetékek által rákényszerített értéket, mivel a bitvezetékek meghajtó tranzisztorai erősebbek, mint a cellák tranzisztorai.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

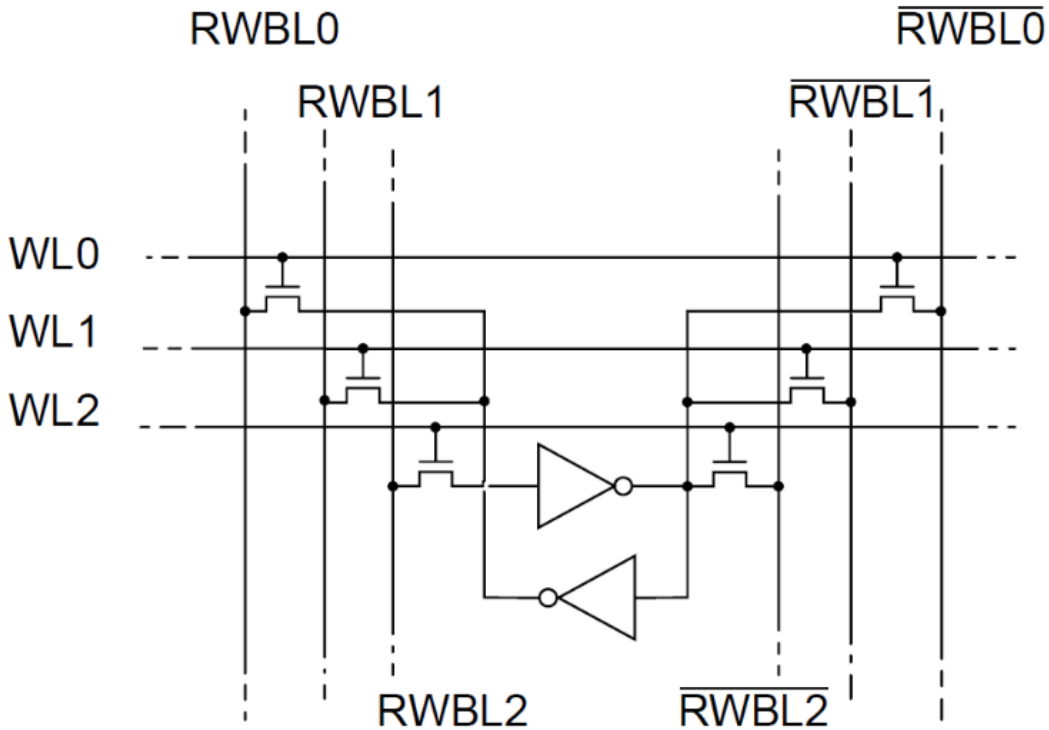
# Az SRAM írása – 2

- A megfelelő működéshez természetesen a tranzisztorok megfelelő méretezésére van szükség.
- Az írásnál szintén fontos szerepe van a két bitvezetéknek: mindkét inverter bemenetét megfelelő szintre beállítva billentjük át a bistabil multivibrátort.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Több portos SRAM megvalósítása

- Amennyiben több portos memóriát szeretnénk (ha egyidejűleg több eszköznek hozzá kell férnie), akkor a hozzáférési tranzisztorok száma nő: portonként egy pár tranzisztorra van szükség.
  - Tehát például 3 db írásra és olvasásra egyaránt használható port esetén 4+3x2=10 tranzisztorra van szükség, és egy cella a így néz ki:



# Az SRAM makrostruktúrája – 1

- Az SRAM struktúrájának bemutatásakor szószervezést használtunk. Ez komoly probléma, hiszen például egy kifejezetten kis kapacitásúnak számító 8kB-os SRAM esetén a címbitek száma $n$=13, a szóvezetékek (más néven dekódolt címvezetékek) száma $k=2^{13}=8192$, míg a szószélesség $m$=8; tehát a bitmező 8192 sorból és 8 oszlopból áll, ami meglehetősen aránytalan.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM makrostruktúrája – 2

- Erre a problémára viszonylag kis méret esetén elégséges megoldás lehet a *bitszervezés*, ahol a címbitek felét sorcímként, a másik felét oszlopcímként használjuk és a dekódolt címvezetékek metszéspontjában egy-egy ÉS kapuval engedélyezzük a cellát, de mivel ez bitenként egy-egy ÉS kaput igényelne, ezért a gyakorlatban a *módosított bitszervezés*t szokták használni, ahol *sor dekódert* és *oszlop multiplexert* alkalmaznak.
- Ezt a megoldást a DRAM struktúrájánál fogjuk bemutatni.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM makrostruktúrája – 3

- Bizonyos méret felett azonban nem egyetlen bitmezővel, hanem hierarchikus felépítésű struktúrával valósítják meg az SRAM-ot (és a DRAM-ot is).


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM interfész lehetőségei – 1

- Szinkronizációs szempontból az SRAM-nak kétféle interfésze lehet:
  - Aszinkron interfész
  - Szinkron interfész

- Aszinkron interfész
  - Az első gyakorlaton bemutatott illesztési felület aszinkron: órajeltől függetlenül, a /CE, /OE, /WE jelekkel vezérelve, a lehető legrövidebb időn belül teljesíti a kapott parancsot.
  - Amennyiben az SRAM-ot mikroprocesszorhoz (vagy mikrokontrollerhez) illesztve operatív tárként használjuk, ez az interfész a szokásos megoldás.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

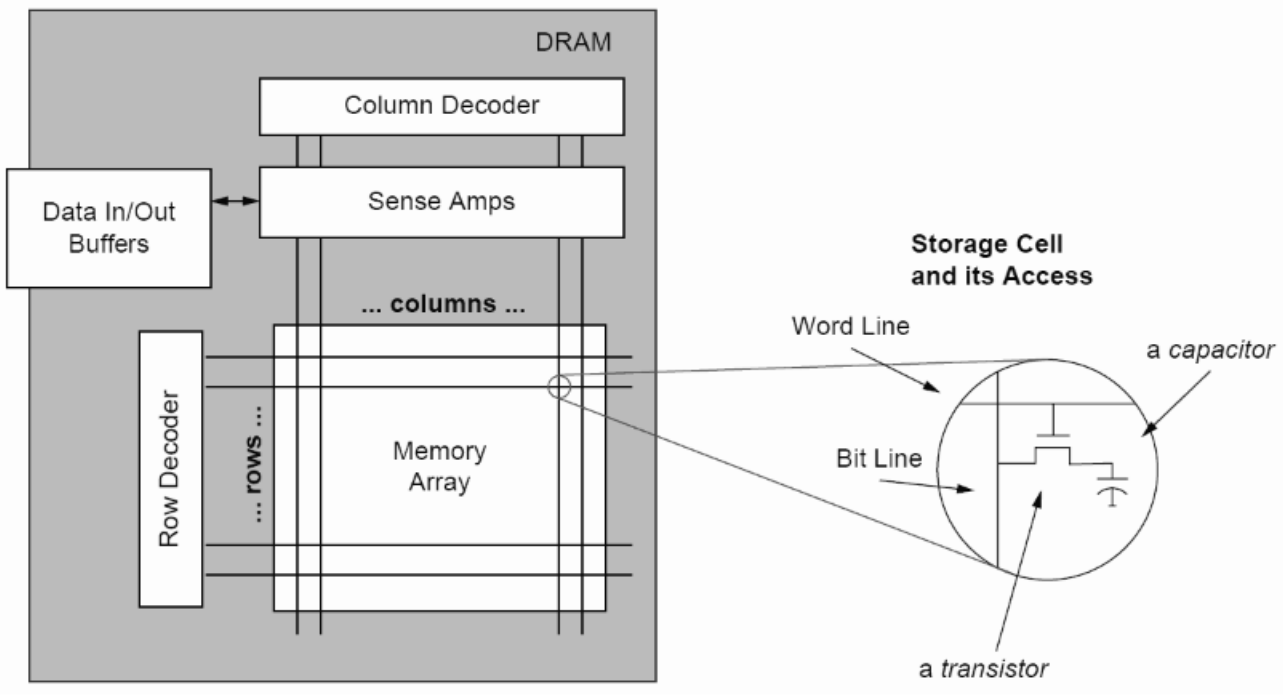
# Az SRAM interfész lehetőségei – 2

- Szinkron interfész
  - A szinkron interfésznél a korábban bemutatott illesztési felület kiegészül még egy (esetleg két) órajel bemenettel (CLK).
  - Ez a megoldás lehetővé teszi, hogy az átvitel *pipelined burst mode*ban történjen: például olvasáskor nemcsak a megadott címről, hanem a rákövetkező bizonyos számú címről is megtörténik a kiolvasás, és az egymást követő memóriarekeszekből való olvasás és a CPU felé való átvitel egyes fázisai egymással átlapolódnak.
  - Ezt a megoldást cache esetén szokták használni, hiszen így egy teljes cache sort igen hatékonyan át lehet vinni.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

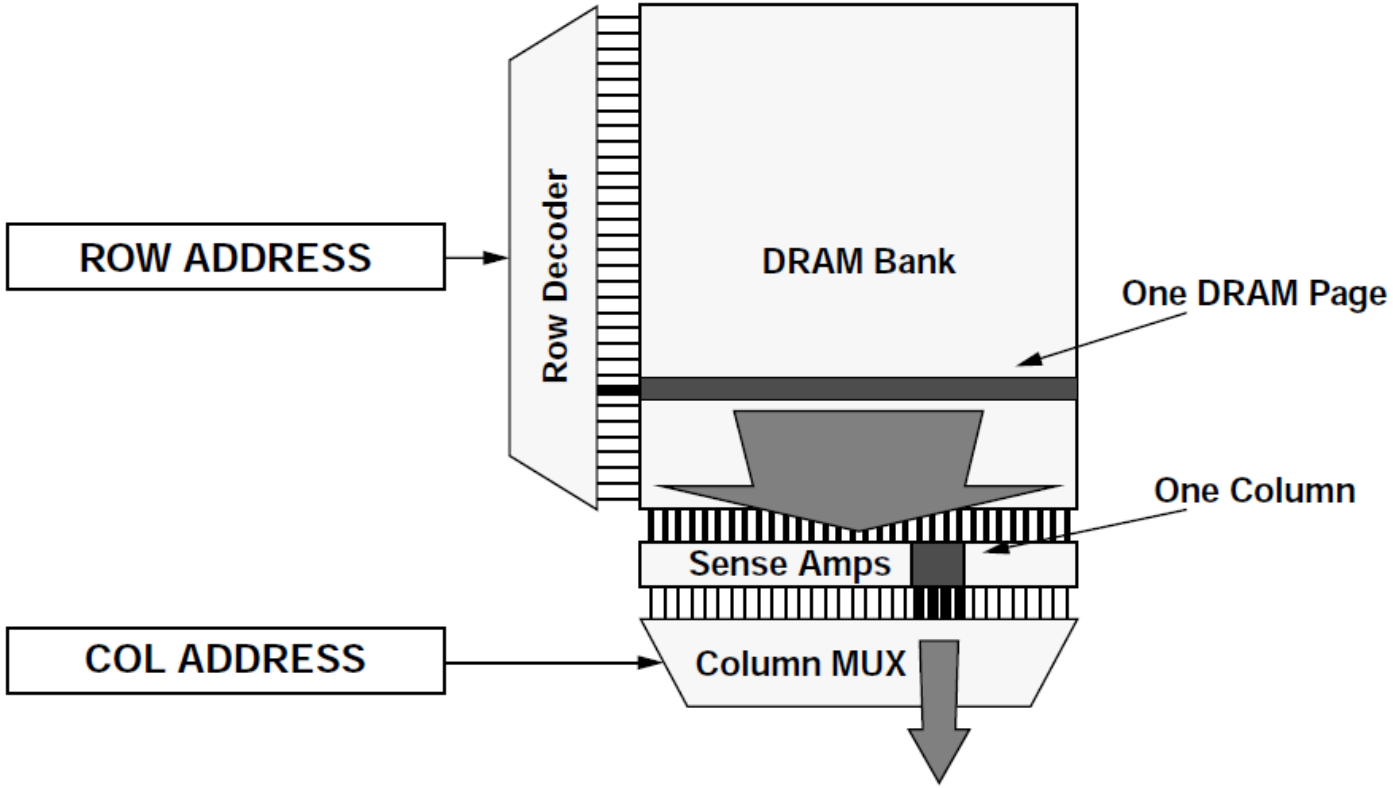
# A DRAM felépítése – 1

- A dinamikus RAM egy cellája (ami egy bit tárolására képes) logikailag egy tranzisztorból és egy kondenzátorból áll.
- A cellákat mátrixba rendezik; egy adott bitet a sor- és oszlopszáma azonosít.



# A DRAM felépítése – 2

- A cellákból egyszerre mindig egy sort lehet kezelni, ezt a sorcím alapján a sor dekóder választja ki; a sor bitjei közül pedig szükség esetén az oszlop multiplexerrel választunk. (Ezt módosított bit szervezésnek hívják.)





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM olvasása – 1

1. Az SRAM-hoz hasonlóan a bitvezetékeket itt is előfeszítjük, de az SRAM-mal ellentétben nem logikai 1-re, hanem csak a logikai 0 és 1 szintje közé "félútra".
2. A sorcím által kijelölt dekódolt sorvezetékre (a bevezető ábrán Word Line néven szerepel) logikai 1-et adunk, ennek hatására a kiválasztott sor tranzisztorai összekapcsolják a kondenzátorokat a hozzájuk tartozó bitvezetékkel, így a kondenzátor töltése módosítja a bitvezeték feszültségszintjét.
   - Az olvasás az SRAM-mal ellentétben destruktív!


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM olvasása – 2

3. Az érzékelő erősítők érzékelik a feszültségnek az előfeszített szinthez képest való eltérését, majd előállítják a korrekt logikai szintet, ami a kondenzátorokba vissza is íródik.
   - Így regeneráljuk a tönkretett tartalmat.
   - Amíg a /RAS jel alacsony szintje fennáll, a kiválasztott sor *aktív* állapota fennmarad. Angolul úgy fejezik ki, hogy "the page is open".
3. Az oszlop multiplexer az oszlopcím alapján kiválasztja a kért bitet és az megjelenik a DRAM kimenetén.
4. A /RAS jel felfutó élének hatására a sor aktív állapota megszűnik.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM adatszélessége

- A kívánt adatszélesség elérésére többféle módunk van.
  - Lehet tokonként csak egy bitet kezelni és egy memória modulra a kívánt számú DRAM tokot rátenni.
  - Lehet DRAM tokonként több bitet is kezelni;
    - akár úgy, hogy egymással párhuzamosan működtetünk több bitmezőt,
    - de úgy is, hogy egy oszlopcím hatására egymás mellett több bitet (pl. 4-et) választunk ki.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM írása

1. A sorcímmel kiválasztott sor aktiválása ugyanúgy történik, mint olvasás esetén (lásd ott: 1-3. lépések).
2. Az oszlopcímmel kiválasztott bit értékét az érzékelő erősítőre az adatbemeneten kapott értéknek megfelelően "rákényszerítjük".
3. Az egész sor visszaíródik (frissül), benne az oszlopcímmel kiválasztott bit az új értéket veszi fel.
4. A /RAS jel felfutó élének hatására a sor aktív állapota megszűnik.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM frissítése – klasszikus

- Az eredeti megoldás szerint (amikor a DRAM lényegében semmi támogatást sem ad a frissítéshez) a frissítés az olvasás 1-3. és 5. lépésével történik.
- Az olvasáshoz képest a 4. lépés azért marad el, mert a /RAS=0 tartama alatt nincs /CAS=0.
- Ehhez a megoldáshoz szükséges, hogy a DRAM kívülről megkapja a frissítendő sor címét.
  - A címet az egyszerű, 8 bites gépek idején tipikusan a CPU-nak kellett előállítania; ehhez egy számlálót használt, amiben nyilvántartotta, hogy éppen melyik sor frissítése következik. Adott időközönként egy megszakítási rutin elvégezte a következő esedékes sor frissítését.
  - Később a frissítés memória vezérlő feladata lett.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM frissítése – korszerű – 1

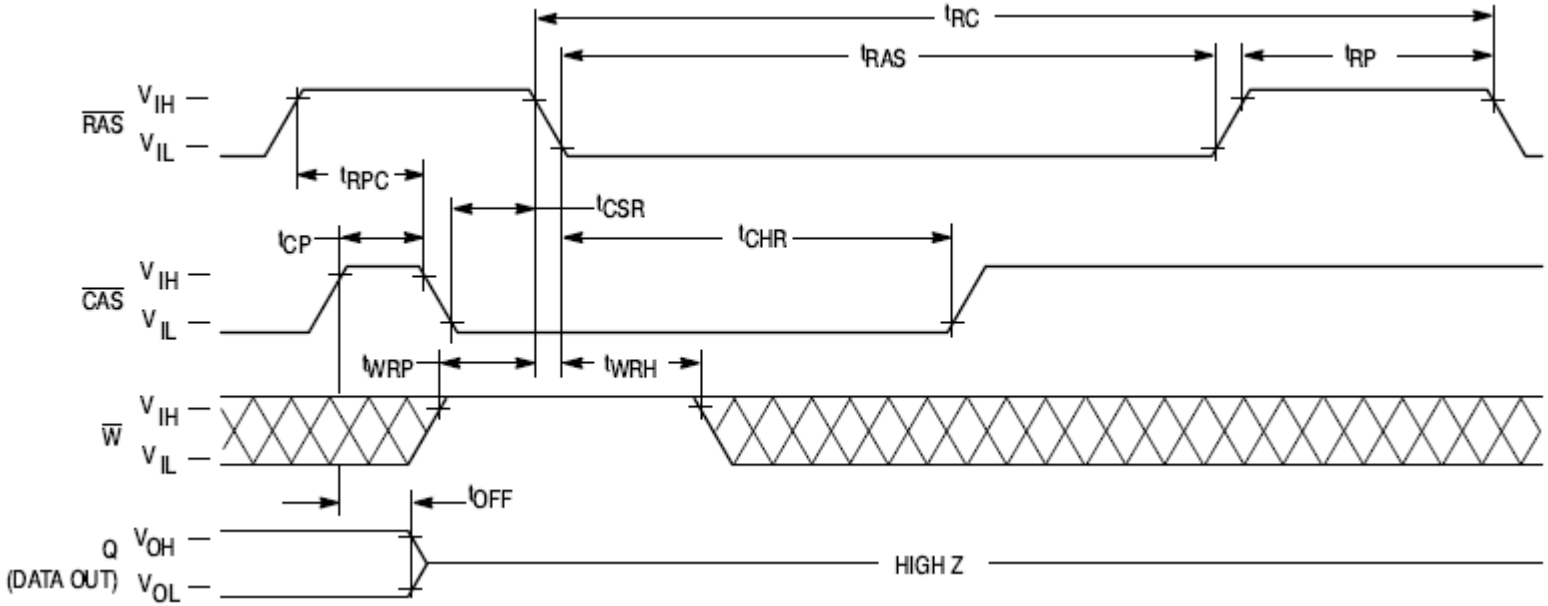
- DRAM chipeket kiegészítették egy számlálóval, ami nyilvántartja, hogy melyik sort kell frissíteni, és a frissítés protokollját is módosították:
  - Amennyiben a /CAS jel szintje 0 lesz még a /RAS jel 0 szintje előtt, majd ezt követi a /RAS=0, akkor ez (a korábban illegális eset) azt jelenti, hogy a kapott címet figyelmen kívül hagyva, a DRAM-nak a belső számlálója szerinti sort kell frissítenie.
  - Ilyenkor természetesen növeli is a számláló értékét.

Ezt *CBR frissítés*nek (CAS-before-RAS refresh) nevezik.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM frissítése – korszerű – 2

- Az alábbi ábra egy katalógusból származik, pusztán a CBR frissítés illusztrációjának szánjuk, az egyes időzítésekkel most nem foglalkozunk.





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SRAM és DRAM összehasonlítása – 1

- Költség szempontjából
  - Mivel egy SRAM cella 6 tranzisztorból áll, egy DRAM cella pedig egy tranzisztorból és egy kondenzátorból, ráadásul az SRAM-nál bitenként két bitvezeték van, a DRAM-nál pedig csak egy, ezért adott méretű lapkán DRAM esetén több (6-8x annyi) bit fér el, következésképpen olcsóbb.
  - Szintén az olcsóságot segíti elő, hogy DRAM-nál a címet két részletben viszik át, ami csökkenti a tok lábszámát, de a DRAM fent megismert működése miatt további lassulást nem okoz.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SRAM és DRAM összehasonlítása – 2

- Sebesség szempontjából
  - Működési sebességet tekintve azonban az SRAM nyer, ennek oka szintén a belső felépítésében keresendő: a bistabil multivibrátor állapota gyorsabban érzékelhető (néhány ns), mint a kondenzátor csekélyke töltésének állapota (néhányszor 10ns).
- Felhasználás egyszerűsége szempontjából
  - Az illesztés is SRAM esetén kényelmesebb: a cím egyben kezelhető; ráadásul frissítésre sincs szükség.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SRAM és DRAM összehasonlítása – 3

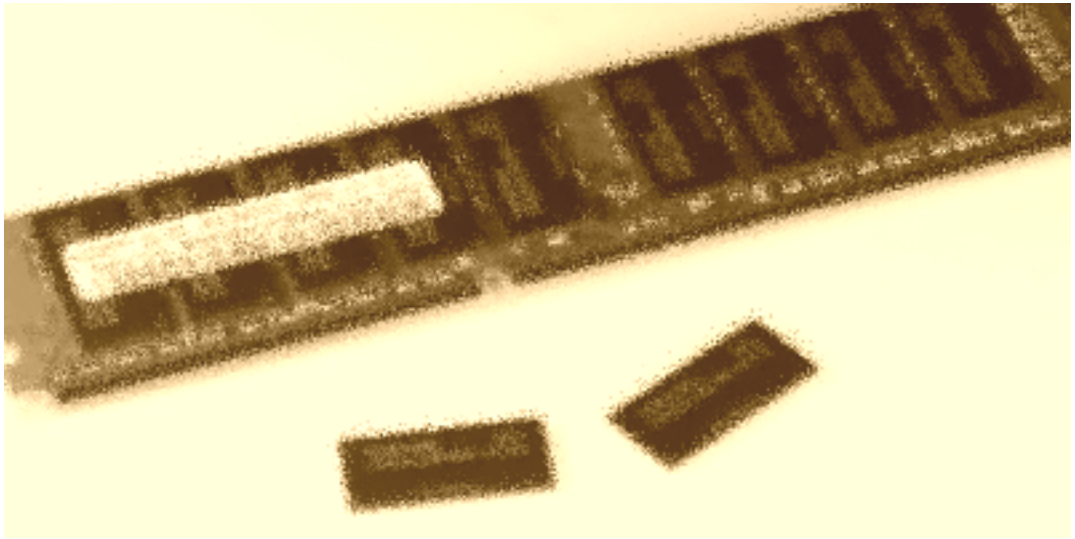
- Célszerű felhasználási terület szempontjából
  - A fentiekből következően számítógépek operatív memóriája céljára DRAM-ot használnak.
  - Az SRAM-ot cache céljára és (cache nélküli) kis rendszerek operatív memóriájaként is alkalmazzák.
  - A CPU és az SRAM azonos technológiával gyártható, ezért azokat könnyű egy lapkára integrálni (belső cache), míg a DRAM gyártási technológiája eltérő.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Elvi korlát az elérési időre

- Felmerül a kérdés, hogy ha az SRAM és DRAM közötti sebességkülönbség oka a technológiában keresendő, akkor miért van értelme több szintű cache alkalmazásának - különösen akkor, ha ezek mindegyikét a CPU-val azonos lapkára integrálják.
- A válasz nagyon egyszerű: a jelterjedési késleltetés miatt. A biteket tároló cellákat (akár egy, akár több bitmezőt használunk is) lényegében mindenképpen két dimenzióban kell elrendeznünk, így a befoglaló négyzet oldala a bitszám (azaz a memóriakapacitás) négyzetgyökével arányos: így ha például a cache kapacitását két nagyságrenddel növeljük, akkor a jelterjedési késleltetés egy nagyságrenddel nő.

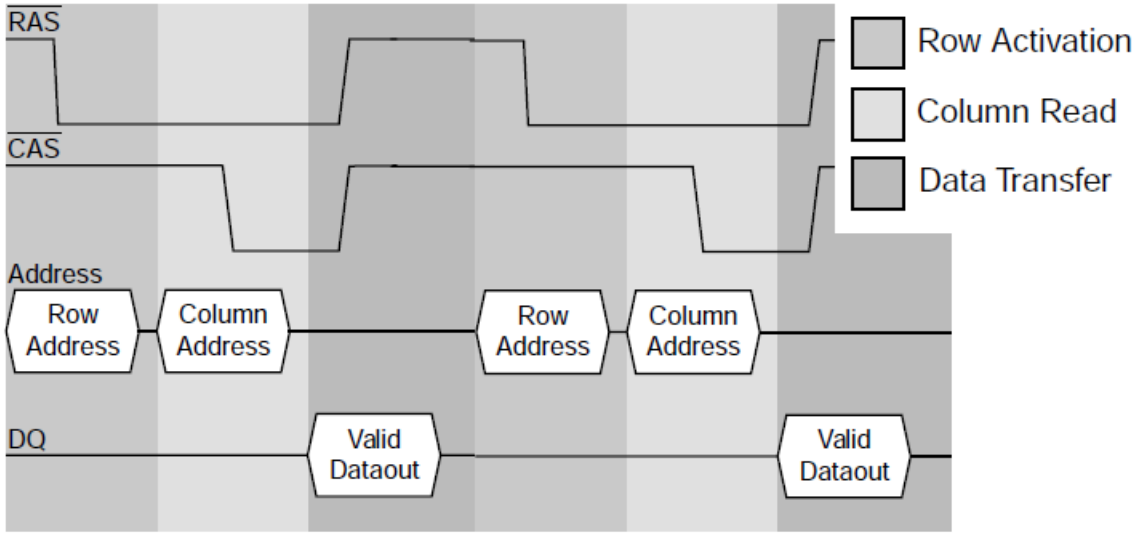

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



# A DRAM FEJLŐDÉSE

# A klasszikus aszinkron DRAM

- Teljesítményének fokozására több, általában egymásra épülő megoldás is született, a továbbiakban ezeket vesszük sorra. Összehasonlítási alapként tekintsük az alábbi ábrát; itt minden olvasásánál újra és újra meg kell adni először a sorcímet, majd az oszlopcímet.



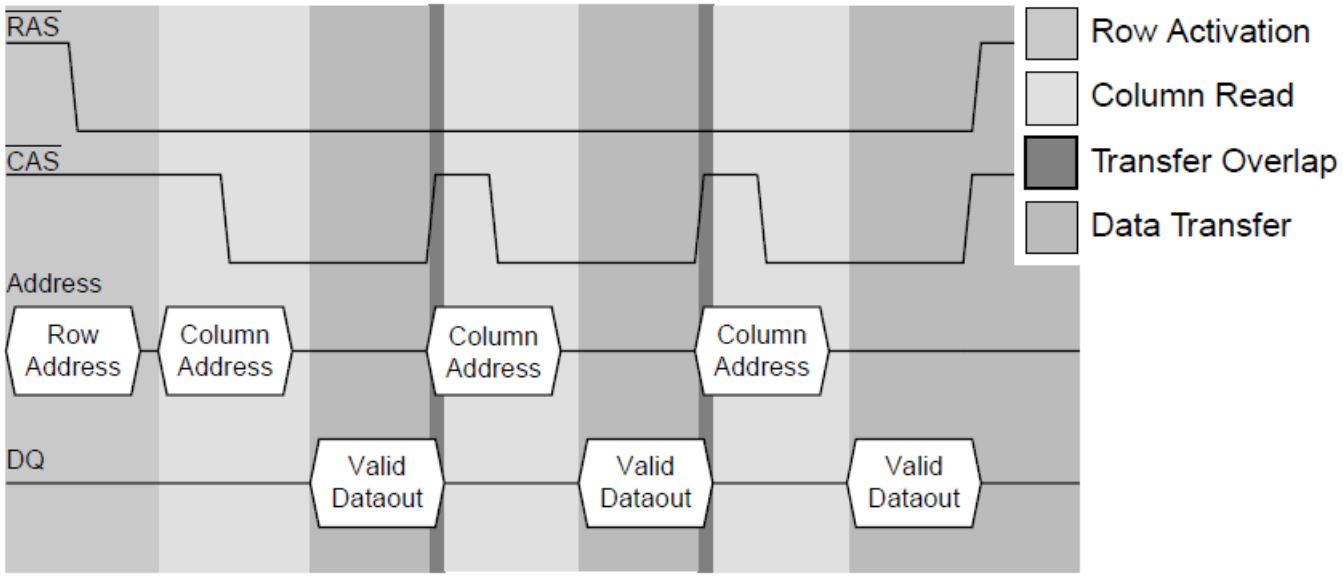
# FPM DRAM – 1

- A DRAM olvasásának átlagos ideje lerövidíthető, ha megengedjük, hogy azonos sorcímről történő olvasás esetén a /RAS jel alacsonyan tartása mellett, csupán a CAS jellel vezérelve egymás után több oszlopcímről is olvashassunk.
- Láttuk, hogy amíg a /RAS=0 fennáll, a sorcím által kiválasztott sor összes bitje elérhető az érzékelő erősítőkben, tehát csak ki kell választani közülük a számunkra még szükséges további biteket.
- Ezt a megoldást (gyors) *lapolvasás*nak (**F**ast **P**age **M**ode) nevezzük; az azonos sorcímhez tartozó oszlopokat egy memória "lap"-nak tekintjük.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# FPM DRAM – 2

- Figyeljük meg a megoldás nyereségét az alábbi ábrán!
  - A sorcímet laponként egyszer elegendő átvinni.
  - Apró átlapolás is megfigyelhető az érvényes adat és az új oszlopcím átvitelénél, de ennek mértéke nem jelentős.



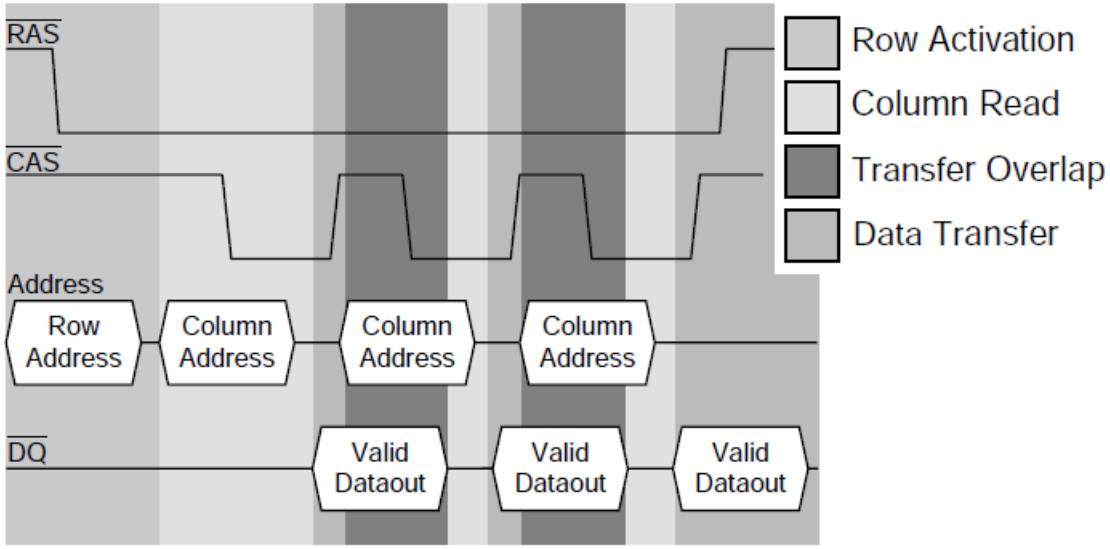
# EDO DRAM – 1

- A következő lépés az EDO (**E**xtended **D**ata **O**ut) DRAM, ami az FPM DRAM kiegészítése egy kimeneti latch-csel; így az adat kívülről még elérhető marad, miközben már új oszlopcímet adunk meg. Ezzel igen jelentős átlapolás érhető el a bitmezőből kiolvasott adat átvitele és az új oszlopcím megadása között.
- Megjegyzés: Az EDO DRAM-nál bevezették az /OE bemenetet, a CPU (vagy memória vezérlő) ezzel tudja jelezni, hogy meddig van szüksége az adatra.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# EDO DRAM – 2

- Figyeljük meg az átlapolást az érvényes adat és az új oszlopcím átvitele között!



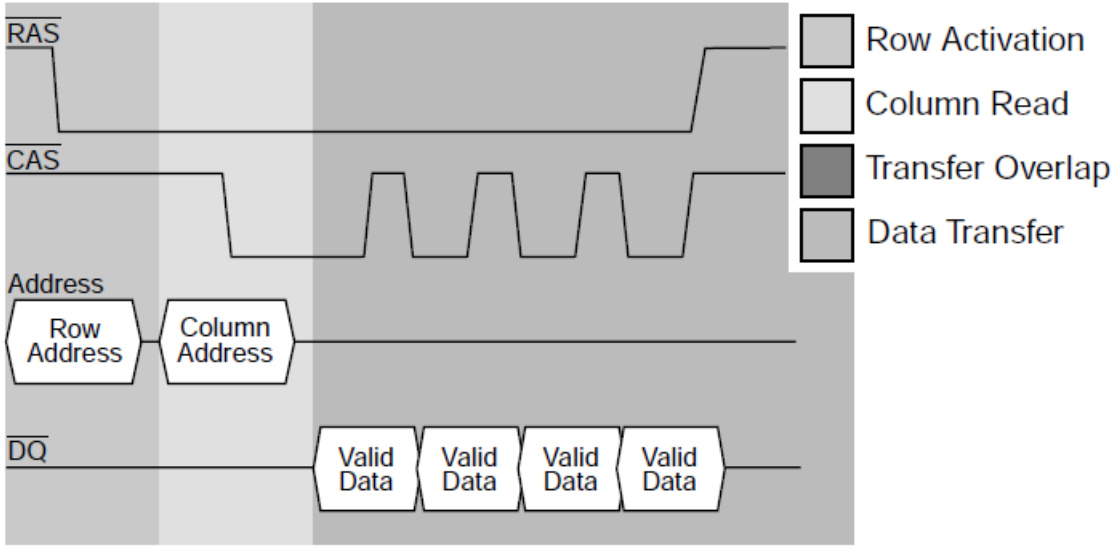
# BEDO DRAM – 1

- Ismét egy újabb lépés a BEDO (**B**urst-Mode **EDO**) DRAM, ahol már nem is kell megadni az új oszlopcímet, hanem az oszlopcímet egy belső számláló egyesével növeli a kezdeti értékről.
- Így természetesen csak egymást követő címek érhetők el, de a gyakorlatban úgyis ez a szokásos eljárás.
- (Ez a megoldás is kifejezetten hasznos, de kevésbé terjedt el.)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# BEDO DRAM – 2

- Figyeljük meg, hogy az időegységenként átvitt adatok mennyisége tovább nő!



# SDRAM – 1

- Az SDRAM (**S**ynchronous **DRAM**) egy igen jelentős fejlődési lépés. Míg korábban a /CAS és /RAS bemenetek értéke tetszőleges időpontban változhatott, és azok közvetlenül vezérelték a DRAM belső működését, addig az SDRAM-ban minden bemeneti változás egy külső órajellel szinkronizáltan történik.
- Ez azért előnyös, mert az SDRAM tervezői így kihasználhatják, hogy az IC nem akármikor, hanem csak előre meghatározott időpontokban kaphat valamilyen parancsot.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

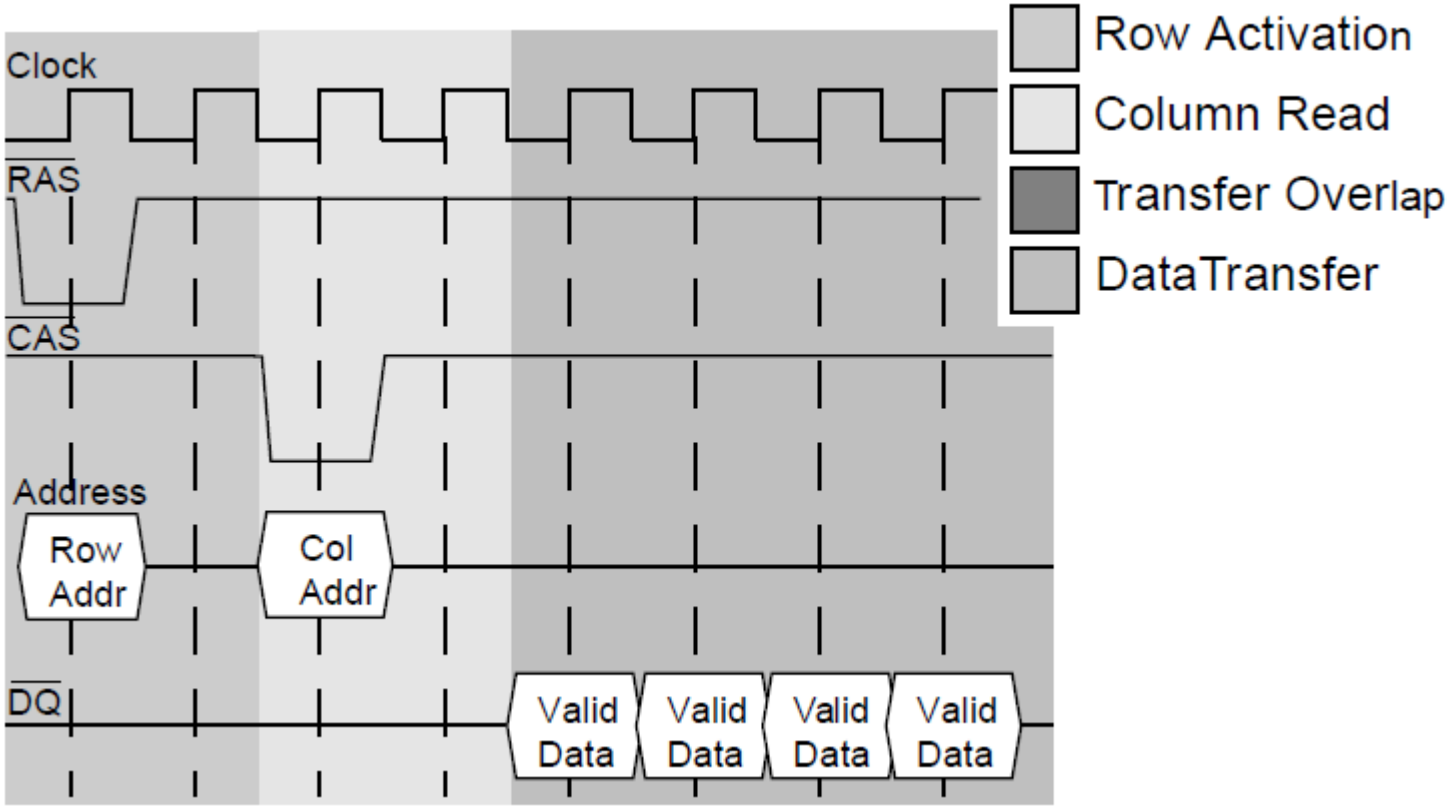
# SDRAM – 1

- A BEDO DRAM-hoz hasonlóan az SDRAM is képes automatikusan növekvő oszlopcímekről egymás után több adat átvitelére (burst), de attól eltérően itt már ezt nem kell a /CAS jellel újra és újra igényelni, hanem az adatok megjelenítése az órajel ütemére történik, és az SDRAM-ban van egy programozható regiszter, amiben az egymás után átviendő adatok számát be lehet állítani (például a cache sor hosszának függvényében).
- Bár az SDRAM megjelenésekor teljesítményében nem volt jobb a BEDO DRAM-nál, hosszú távon nyertes lett, mert a szinkronizált működés egyre nagyobb működési frekvenciát tett lehetővé.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SDRAM – 2

- Figyeljük meg, hogy az órajellel ütemezetten az újabb adatok megjelennek a kimeneten – minden további kérés nélkül!



# SDRAM – 3

- A további, ma is általánosan elterjedt típusok mind az SDRAM továbbfejlesztett változatai, ezeket röviden áttekintjük, de előbb megismerjük a legfontosabb időzítéseket.
- Megjegyzés: A későbbi, DDRx SDRAM-októl való megkülönböztetés érdekében az eredeti SDRAM-ot SDR (**S**ingle **D**ata **R**ate) SDRAM-nak is szokták nevezni.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SDRAM időzítések

- Mivel minden órajellel szinkronizáltan történik, ezért az időzítéseket nem ns-ban, hanem a szükséges órajel ciklusok számában szokták megadni. Természetesen ezek az órajel frekvenciájának függvényében értelmezhetők, és szükség esetén ns-ra is átszámíthatók. (Aszinkron DRAM esetében pedig természetesen ns-ban adják meg őket.)
- A következőkben megadjuk azt a négy időzítést, amivel az egyes SDRAM memória modulokat jellemezni szokták.
- Az időzítések értékét a következőkben használt sorrendben, a számértékeket egymástól kötőjellel elválasztva szokták feltüntetni, például: 3-4-4-8.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# CAS Latency

- Definíció
  - A /CAS=0-tól az adat megjelenéséig eltelt idő. (Természetesen egy már aktív sor esetén.)
- *Magyarázat*
  - *Eddig tart, amíg az érzékelő erősítőben már jelen levő bitek közül az oszlopcím által kiválasztott bit a kimeneten megjelenik.*
- *Jelölések*
  - $t_{CL}$, $t_{CAS}$, ($t_{ACT}$)
- Alternatív megnevezés
  - **A**ccess **C**olumn **T**ime


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# RAS to CAS Delay

- Definíció
  - A /RAS=0-tól legalább ennyi időnek el kell telnie a /CAS=0 megjelenéséig.
- *Magyarázat*
  - *Eddig tart a sor "megnyitása": a sor kiválasztásától addig eltelő idő, amíg a kiválasztott sor tartalma megjelenik az érzékelő erősítőben. Ez után lehet olvasni vagy írni.*
- *Jelölés*
  - $t_{RCD}$


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Row Precharge time

- Definíció
  - A /RAS jelnek legalább ennyi ideig magas szintűnek kell lennie, azaz egy aktív sor lezárásától kezdve ennyi időnek kell eltelnie, mire egy másikat meg lehet nyitni.
- *Magyarázat*
  - *Ennyi idő szükséges a bitvezetékek előfeszítéséhez.*
- *Jelölés*
  - $t_{RP}$


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Row Address Strobe time

- Definíció
  - A /RAS jelnek legalább ennyi ideig alacsony szintűnek kell lennie.
- *Magyarázat*
  - *Ez az idő ahhoz szükséges, hogy a sor kiolvasása és visszaírása rendben megtörténjen.*
- *Jelölés*
  - $t_{RAS}$
- Alternatív megnevezések
  - **R**ow **A**ctive **T**ime
  - Active to Precharge (time)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# DDR SDRAM

- A DDR (**D**ouble **D**ata **R**ate) SDRAM azonos órajel esetén az SDR SDRAM-hoz képest (közel) kétszer akkora átviteli sebességre képes.
- Ezt úgy oldja meg, hogy az órajelnek mind a felfutó, mind a lefutó élénél végez adatátvitelt. Míg az SDR SDRAM minimális burst hossza 1, a DDR SDRAM-é 2.
- A szabványos nevében már a fizikai órajel frekvenciájának a dupláját használja, így a DDR-200 SDRAM valójában 100MHz-en működik.
  - A moduljai 64 bit szélesek, és a nevükben figyelembe veszik azt is, hogy hány byte információt visznek át párhuzamosan. Például egy 200MHz-en működő 64 bit széles DDR-400-at PC-3200 néven lehet megvásárolni, és maximum 3200MB/s átviteli sebességre képes.

# DDR2 SDRAM

- A DDR2 SDRAM belső órajele csak fele a külsőnek, így adott belső órajel mellett a külső kétszer akkora lehet, és a külső órajelciklusonként visz át két bitet, tehát a belső órajel minden periódusa alatt 4-et.
- Míg a DDR-nél 2 bites volt, itt már 4 bit-es a prefetch puffer (a DDR3-nál pedig 8 bites).
- Nézzünk itt is egy tipikus számpéldát.
  - Egy DDR2-800 SDRAM-nak a belső órajele 200MHz, a külső 400MHz, a "800" úgy jön ki, hogy a külső órajel fel- és lefutó élénél is van adatátvitel; a 64 bites szélesség miatt pedig PC2-6400 modulként adják el, és maximum 6400 MB/s átviteli sebességre képes.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# DDR3 SDRAM

- A DDR3 SDRAM belső órajele már csak negyede a külsőnek, így adott belső órajel mellett a külső négyszer akkora lehet, és a külső órajelciklusonként visz át két bitet, tehát a belső órajel minden periódusa alatt 8-at.
- Nézzünk itt is egy tipikus számpéldát.
  - Egy DDR3-1600 SDRAM-nak a belső órajele 200MHz, a külső 800MHz, az "1600" úgy jön ki, hogy a külső órajel fel- és lefutó élénél is van adatátvitel; a 64 bites szélesség miatt pedig PC3-12800 modulként adják el, és maximum 12800 MB/s átviteli sebességre képes.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# DDRx SDRAM

- A DDR sorozat tagjai egymással sem mechanikailag sem feszültségszint tekintetében nem kompatibilisek.
  - A disszipáció csökkentése érdekében egyre kisebb feszültségszinteket használnak.
- A DDR4 piaci megjelenése 2012-re várható.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

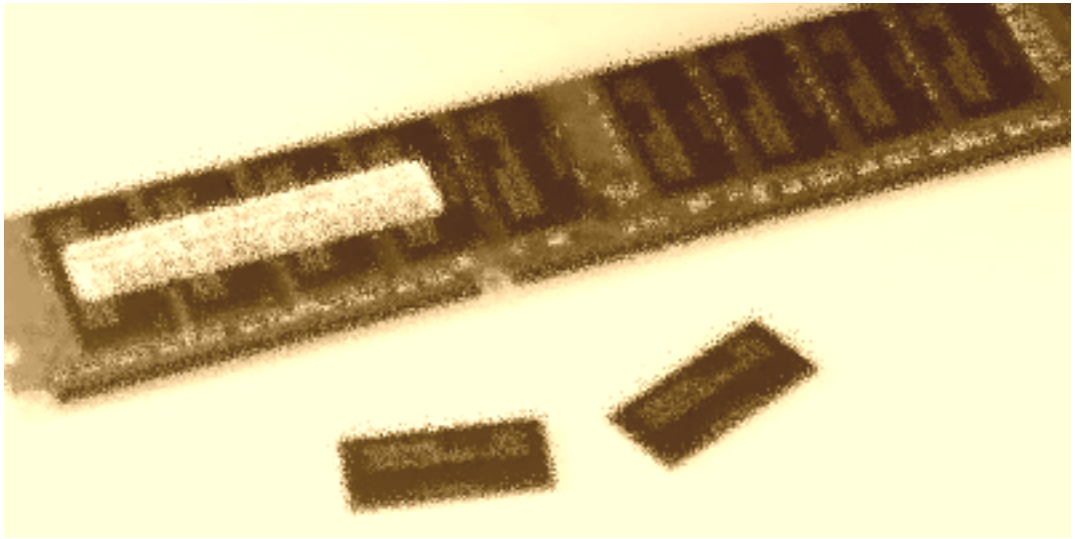
# A GDDR sorozat

- A GDDR (**G**raphics **DDR**) sorozat videokártyákhoz készült.
- A számozás 1-gyel (vagy kettővel) előbbre tart a DDR sorozatnál, így például a GDDR3 közelítőleg a DDR2-nek felel meg. Ahhoz képest van néhány előnye, ebből rendszertechnikailag számunkra az az érdekes, hogy egyidejűleg írható és olvasható.
- A GDDR4 a DDR3-ra épül és a GDDR3 utódjának szánták, de nem igazán váltotta be a hozzá fűzött reményeket.
- A GDDR5 is DDR3-ra épül, és 2008 óta kapható.

# A Rambus irányzat

- A Rambus cég nem gyárt memóriát csak fejleszt és licencel. Első termékével az **RDRAM**-mal (**R**ambus **DRAM**) forradalmi újításokba fogott (például lényegesen keskenyebb buszt használ; a parancsait nem 1 bites vonalak jelei, hanem több bites üzenetek hordozzák), de a teljesítményhez képest az ár túl magas volt, így bár egy ideig terjedt, aztán az olcsóbb DDR sorozat kiszorította.
- A következő terméke az **XDR** (e**X**treme **D**ata **R**ate) DRAM jelenleg a DDR2 és a GDDR4 vetélytársa. Olyan esetekben van esélye, ahol a kis lábszám előnyt jelent.
- 2008-ban elkészült az **XDR2 DRAM** specifikációja is, de azóta sem talált gyártóra.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



# MEMÓRIA MODULOK


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

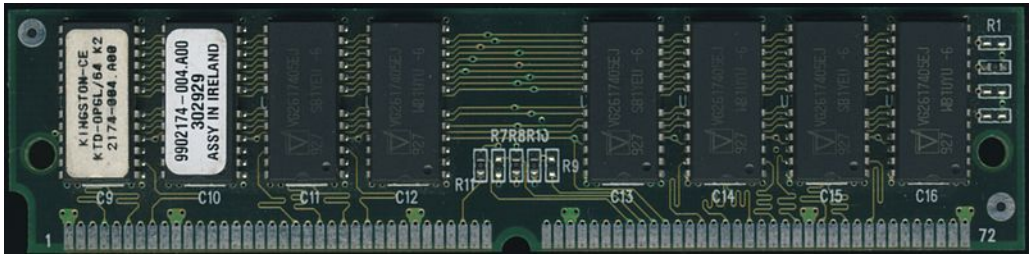
# SIMM – 1

- A SIMM (**S**ingle **I**n-line **M**emory **M**odule) memória modulok csak az egyik oldalon tartalmaznak memória IC-ket és bár a csatlakozási felületnél mindkét oldalon vannak érintkezőik, ezek redundánsak.
- Főleg az 1980-as és 90-es években használták őket, 30 és 72 érintkezős kivitelben készültek, az előbbiek 8/9 bitesek, az utóbbiak 32/36 bitesek voltak - attól függően, hogy tartalmaztak-e hibajavításra (ECC) 8 bitenként +1 bitet. Ezek általában még aszinkron DRAM-ok voltak, de készült még DDR SDRAM is SIMM kivitelben.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SIMM – 2

- Felül 8/9 bites aszinkron DRAM, alatta 32/36 bites FPM DRAM, legalul 32/36 bites  EDO DRAM



# DIMM – 1

- A DIMM (**D**ual **I**n-line **M**emory **M**odule) memória modulok mindkét oldalon tartalmaznak memória IC-ket, adatszélességük is 64 bit, és kihasználják mindkét oldalon az érintkezőket.
  - Az Intel Pentiummal terjedtek el, így már nem kellett őket párban használni, mint a SIMM modulokat.
- Bár készültek FPM és EDO DRAM-ok is DIMM kivitelben, de a DIMM-ek már főleg szinkron DRAM-ok, azon belül bejárták a technológiai fejlődést. Az érintkezők számában is változatosak.

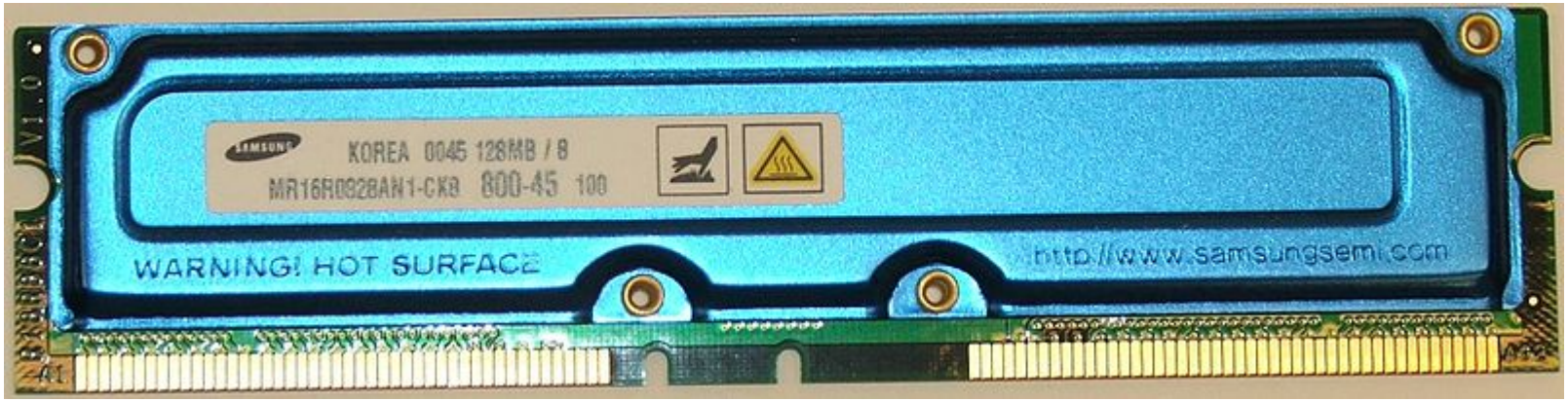

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# DIMM – 2

- Az alábbiak közül a felső egy 168 érintkezős SDRAM, az alsó egy 184 érintkezős DDR SDRAM:

# RIMM

- Bár az **RDRAM** (**R**ambus **DRAM**) vagy **DRDRAM** (**Direct Rambus DRAM**) modulok is SIMM vagy DIMM kivitelűek, megkülönböztetésül mégis **RIMM**-nek (**R**ambus **I**n-line **M**emory **M**odule) szokták őket nevezni.
  - Jól látható jellemzőjük az integrált hűtő - de integrált hűtő lehet más SIMM/DIMM modulon is!



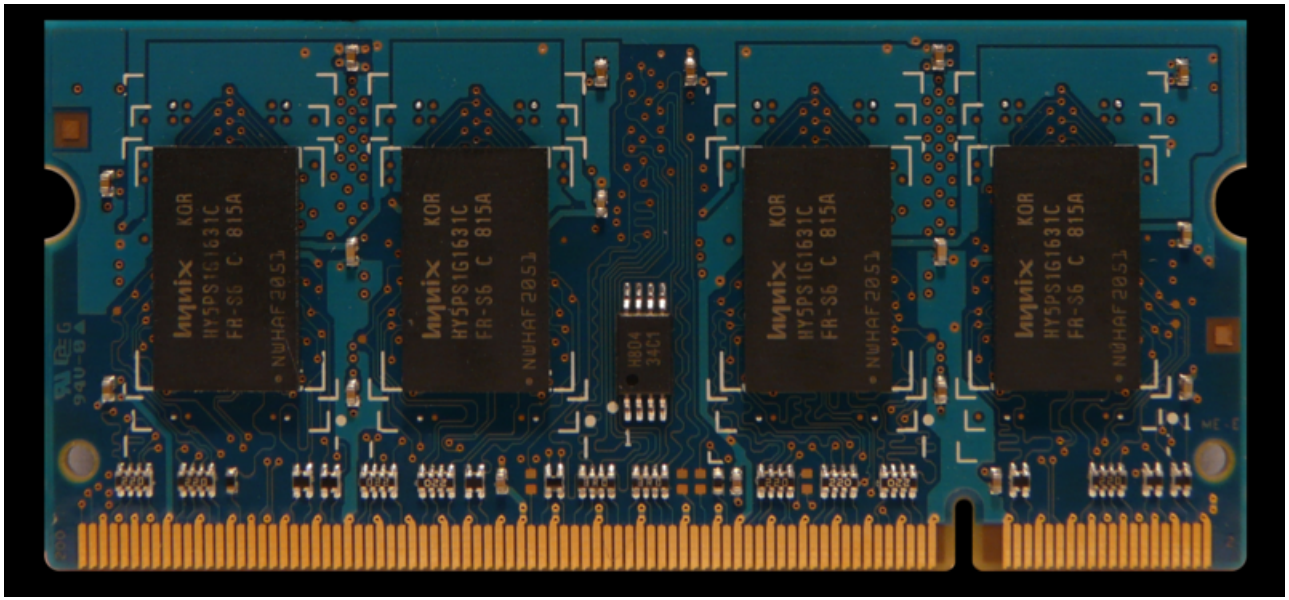
# XDR DRAM

- Az **RDRAM** utódja az **XDR** (e**X**treme **D**ata **R**ate) **DRAM** az elődjéhez hasonlóan szintén integrált hűtővel rendelkezik:

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SO-DIMM

- A **SO-DIMM** (**S**mall **O**utline **DIMM**) a DIMM helytakarékos változata. Többféle érintkezőszámmal (100, 144 és 200) készül és DDR/DDR2/DDR3 SDRAM egyaránt található köztük. Az alábbi képen egy PC2-6400 DDR2 SO-DIMM látható.

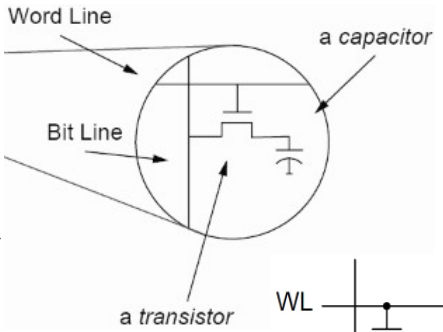


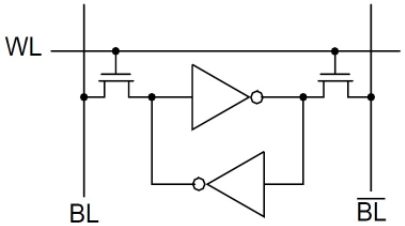
# Önálló hallgatói munka

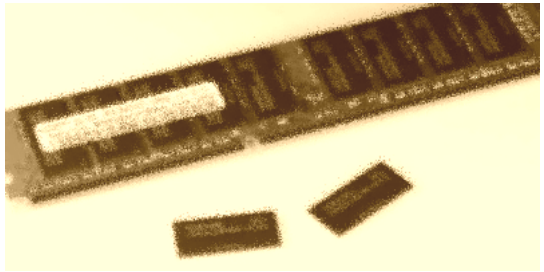
1. Számítsa ki a bitmezőt befoglaló négyzetekben létrejövő jelterjedési késleltetés arányát egy 16kB-os L1 és egy 1MB-os L2 cache esetén!
2. Van egy SDRAM chipünk. Az időzítései: 3-3-3-6. Feltéve, hogy előfeszített (precharged) állapotban van, és két bitet kell belőle kiolvasnunk, amelyek sorcíme különböző; a kezdéstől számítva mennyi idő múlva jelenik meg a második bit a kimeneten, ha a chip 100MHz frekvenciájú órajelet kap?
3. Oldja meg az előző feladatot 6-6-6-18 időzítéssel és 200MHz-es órajellel!
4. Egy PC3-6400-ként eladott DDR3 memóriamodul milyen belső és milyen külső órajellel működik?

# Összefoglalás

- SRAM és DRAM felépítése és ennek következményei
  - SRAM felépítése és működése
  - DRAM felépítése és működése
  - SRAM és DRAM összehasonlítása
- DRAM fejlődése
  - Aszinkron DRAM típusok
  - Szinkron DRAM típusok
  - Legfontosabb időzítések szinkron DRAM-oknál
- Memória modulok
- Feladatok



# Aktualitások

- Örömmel hirdetjük, hogy a **BME-Infokom Innovátor Kft. Hallgatói Ötletbörzét szervez a Híradástechnikai Tanszékkel** közösen, az alábbi **9 témában**:
  - 1) Infokommunikációs hálózatok tervezési és elemzési modelljei és módszerei
    2) Szélessávú hozzáférési hálózatok vizsgálata és elemzése
    3) Hálózat, információ és szolgáltatás biztonság
    4) Fix-mobil konvergencia
    5) Személyre szabott és helyfüggő média-tartalom szolgáltatások
    6) Intelligens közlekedési alkalmazások

    7) Rádiófrekvenciás azonosítás
    8) Nyomon követés és hely alapú szolgáltatások
    9) Intelligens környezet

http://www.bme-infokom.hu/palyazat/

# Kérdések?

# KÖSZÖNÖM A FIGYELMET!

Híradástechnikai Tanszék

Dr. Lencse Gábor
tudományos főmunkatárs
BME Híradástechnikai Tanszék
lencse@hit.bme.hu

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Számítógép Architektúrák: **Memóriakezelés feladatok**

**1. Feladat** Vegyünk egy 12 bites címtartománnyal rendelkező processzort. Fusson a processzoron egy program a memóriatartomány alsó felében, tehát a 0–2047 címtartomány legyen mindig elérhető. A program a fizikai memóriában, sorrendben az alábbi tartalmakat hivatkozza:

- 3987, 4200, 3518, 11092, 11093, 10070

A hivatkozott címek szemmel láthatóan meghaladják a processzor címtartományát. A feladatok a következők:

(a) Ha a tárbővítésre tömbkapcsolást alkalmazunk, 2 kB-os tömbökből építkezünk, és minden tömböt 2 kB-os határra illesztünk, hány tömbre van szükség (az állandó, mindig elérhetőt is beleértve)? Kizárólag a hivatkozott címeket akarjuk lefedni! A fenti hivatkozások során hány tömbátkapcsolásra lesz szükség, ha kezdetben az állandó 0-ás és az 1-es tömb van bekapcsolva?

(b) Ha a tárbővítést indexelt leképzéssel valósítjuk meg, és 1 kB-os ablakokat használunk,

- Hány ablakunk (indexregiszterünk) van?
- Állítsa be az összes indexregiszter értékét úgy, hogy a program futása közben ne kelljen állítani őket!

**2. Feladat** Legyenek a virtuális címek 24 bitesek, a fizikai címek 16 bitesek, a lapméret legyen $2^{12} = 4$ kB méretű, a laptábla pedig legyen egyszerű egyszintes laptábla 8 bites bejegyzésekkel. A feladatok a következők:

(a) A virtuális címekben hány bit tartozik a lapok azonosításához és hány a lapon belüli eltoláshoz? Rajzolja fel a virtuális címek tagozódását!

(b) Mekkora a teljes laptábla mérete?

(c) Hány lap fér a fizikai memóriába?

(d) Rajzolja fel a laptábla azon tartományát, ami az `A00000` és az `A03FFF` közé eső virtuális címek címfordításához kell! A laptábla bejegyzésekben tüntesse fel a virtuális tárkezelés működéséhez elengedhetetlen mezőket!

(e) Helyezze el a laptáblában az alábbi összerendeléseket:

- Az `A00` lap a fizikai memóriában a `15`-ös keretben található
- Az `A01` lap a háttértáron van
- Az `A02` lap a fizikai memóriában a `3`-mas keretben található
- Az `A03` lap a háttértáron van

(f) Feltéve, hogy nincs laphiba, hány memóriaművelet kell a címfordításhoz

- TLB találat esetén?
- TLB hiba esetén?

**3. Feladat** Legyenek a virtuális címek 16 bitesek, a fizikai címek 13 bitesek, a lapméret legyen $2^{10} = 1$ kB méretű. Használjunk 3 szintű hierarchikus laptáblát, melyben a laptáblák mérete a hierarchia minden szintjén egyforma. A laptábla bejegyzések 16 bitesek. A feladatok a következők:

(a) A virtuális címekben hány bit tartozik a lapon belüli eltoláshoz és hány a különböző szintű laptáblák indexeléséhez? Rajzolja fel a virtuális címek tagozódását!

(b) Mekkora az összes laptábla összegzett mérete?

(c) Optimális körülmények között mennyi a laptáblák által elfoglalt memória minimális mérete? Mennyivel kisebb ez, mintha egyszintű laptáblát használnánk?

(d) Hány lap fér a fizikai memóriába?

(e) Rajzolja fel az összes érintett laptáblát, melyek az `1001000000000000` és az `1001111111111111` közé eső virtuális címek címfordításához kell! A laptábla bejegyzésekben tüntesse fel a virtuális tárkezelés működéséhez elengedhetetlen mezőket!

(f) Helyezze el a laptáblában az alábbi összerendeléseket:

- Az `100100` lap a fizikai memóriában a `7`-es keretben található
- Az `100101` lap a háttértáron van
- Az `100110` lap a fizikai memóriában a `3`-mas keretben található
- Az `100111` lap a háttértáron van

(g) Feltéve, hogy nincs laphiba, hány memóriaművelet kell a címfordításhoz

- TLB találat esetén?
- TLB hiba esetén?

**4. Feladat** Legyenek a virtuális címek 16 bitesek, a fizikai címek 13 bitesek, a lapméret legyen $2^{10} = 1$ kB méretű. Használjunk inverz laptáblát, melyben a hash függvény értéke a virtuális cím 10.-13. bitje által meghatározott szám. A laptáblabejegyzések mérete: 16 bit. A feladatok a következők:

(a) A virtuális címekben hány bit tartozik a lapon belüli eltoláshoz és hány a lapok azonosításához? Rajzolja fel a virtuális címek tagozódását!

(b) Hány lap fér a fizikai memóriába?

(c) Mekkora a teljes laptábla mérete? Mennyivel kisebb ez, mintha egyszintű laptáblát használnánk? Mekkora a hash mutató tábla mérete?

(d) Rajzolja fel a teljes hash mutató táblát és a teljes laptáblát! A laptábla bejegyzésekben tüntesse fel a virtuális tárkezelés működéséhez elengedhetetlen mezőket!

(e) Helyezze el a laptáblában az alábbi összerendeléseket (itt most fontos, hogy ebben a sorrendben!):

- Az `100100` lap a fizikai memóriában a 7-es keretben található
- Az `100101` lap a háttértáron van
- Az `100110` lap a fizikai memóriában a 3-mas keretben található
- Az `100111` lap a háttértáron van
- A `010100` lap a fizikai memóriában a 5-ös keretben található

(f) Feltéve, hogy nincs laphiba, hány memóriaművelet kell a legjobb és a legrosszabb esetben címfordításhoz

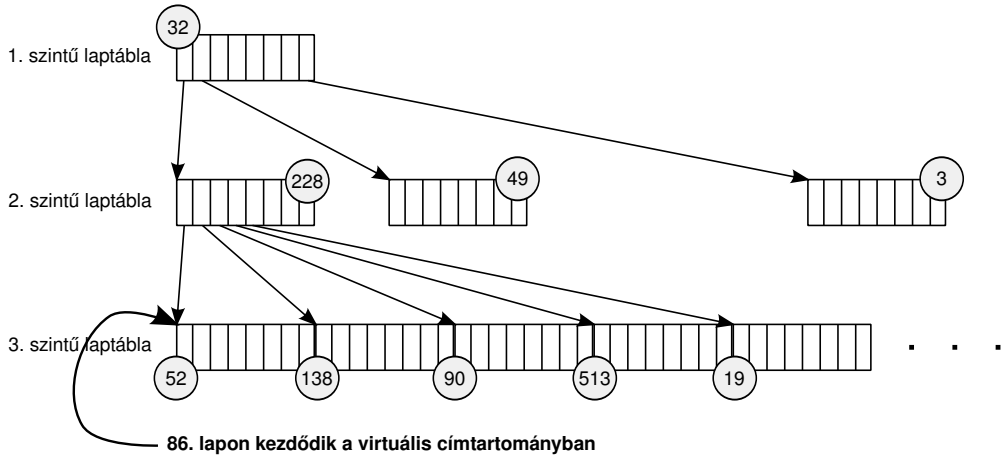
- TLB találat esetén?
- TLB hiba esetén?

Mikor áll elő a legrosszabb eset?

**5. Feladat** Egy processzor használjon 4 bejegyzéses TLB-t a címfordítás gyorsítására. A TLB egy adott pillanatban a következő két érvényes összerendelést tartalmazza:

| Valid | Lapszám | Keretszám |
|---|---|---|
| 1 | 88 | 90 |
| 1 | 1399 | 843 |
| 0 | ? | ? |
| 0 | ? | ? |

A laptábla legyen virtualizált, az alsó szint a 86. lapon kezdődik, egy szelete az alábbi ábrán látható (körökbe írt számok jelzik, hogy a laptáblák melyik keretben vannak):




Legyenek a virtuális címek 42 bitesek, az eltolás mező legyen 12 bites, a laptáblák minden szinten 1024 bejegyzést tartalmaznak, minden bejegyzés mérete 4 byte. A lapok mérete 4 kB. A fizikai címek 32 bitesek.

Hányszor kell a fizikai memóriához nyúlni, amikor az alábbi lapok címfordítását végzi a processzor? Melyik keret hányadik byte-ját és milyen sorrendben kell kiolvasni a fizikai memóriából a címfordítás kedvéért? Tételezzük fel, hogy minden érintett lap a fizikai memóriában van!

(a) 1399
(b) 2064
(c) 4101

Számítógép Architektúrák

# TLB és cache memória feladatok

**1. Feladat** Egy processzor használjon 4 bejegyzéses TLB-t a címfordítás gyorsítására, A TLB állapota egy adott pillanatban legyen a következő:

| Valid | Lapszám | Keretszám | Kor |
|---|---|---|---|
| 1 | 11 | 12 | 1 |
| 1 | 7 | 4 | 2 |
| 1 | 3 | 6 | 3 |
| 0 | 4 | 9 | 4 |

A TLB teljesen asszociatív, és ideális LRU algoritmust használ, a táblázat utolsó oszlopában tartja nyilván, hogy a bejegyzést milyen régen használta utoljára.

A laptábla legyen egyszerű, egyszintes laptábla, melynek tartalma legyen a következő:

| Valid | Keretszám |
|---|---|
| 1 | 5 |
| 0 | Háttértár |
| 0 | Háttértár |
| 1 | 6 |
| 1 | 9 |
| 1 | 11 |
| 0 | Háttértár |
| 1 | 4 |
| 0 | Háttértár |
| 0 | Háttértár |
| 1 | 3 |
| 1 | 12 |

A futó program az alábbi címfordításra váró virtuális címeket hivatkozza (ebben a sorrendben):

- 4095, 31272, 15789, 15000, 7193, 4096, 8912

Laphiba esetén az új lap az addig előforduló legnagyobb keretnél eggyel nagyobb sorszámú keretbe kerül.

A feladatok a következők:

(a) Mi lesz a TLB végső állapota, ha a lapméret 4 kB? Minden címfordításhoz adja meg, hogy TLB találat volt-e (T), TLB hiba volt-e (H), illetve, hogy laphiba volt-e (LH)?

(b) Mi lesz a TLB végső állapota, ha a lapméret 16 kB? Minden címfordításhoz adja meg, hogy TLB találat volt-e (T), TLB hiba volt-e (H), illetve, hogy laphiba volt-e (LH)?

(c) Értékelje, hogy a nagyobb lapmérettel mit nyertünk, illetve (ami a feladatból nem feltétlenül látszik), mit vesztünk?

**2. Feladat** Vegyünk egy 256 byte méretű cache-t 64 byte-os blokkmérettel. A cache kezdetben csupa érvénytelen blokkot tartalmaz.

Egy program az alábbi memóriablokkokról olvas (ebben a sorrendben):

- 1, 3, 8, 4, 3, 6, 8, 1

Adja meg a cache hibák számát és a cache végső tartalmát LRU algoritmus mellett

(a) direkt leképzés

(b) teljesen asszociatív szervezés

(c) két utas asszociatív szervezés

esetén.

**3. Feladat** Vegyünk egy 32 kB méretű cache-t 64 byte-os blokkmérettel. A CPU rendelkezzen 32 bites fizikai, 48 bites virtuális címekkel.

4-utas asszociatív szervezés mellett hány darab és hány bites komparátor dolgozik egy keresés során, ha

(a) Fizikailag indexelt cache-t használunk, fizikai tag-ekkel

(b) Virtuálisan indexelt cache-t használunk, virtuális tag-ekkel

**4. Feladat** Vegyünk egy 512 byte méretű cache-t 64 byte-os blokkmérettel. A cache fizikai címeket használ mind az indexeléshez, mind a tag-ekhez. A fizikai címek 16 bit szélesek.

Egy program az alábbi memóriacímekről olvas (ebben a sorrendben):

- 13, 136, 490, 541, 670, 74, 581, 980

(a) Adja meg a fenti címekhez tartozó "tag", "index" és "eltolás" mezőket

- teljesen asszociatív szervezés
- direkt leképzés
- két utas asszociatív szervezés

esetén.

(b) Adja meg a cache végső tartalmát mind a 3 esetben, LRU algoritmus mellett! (A cache kezdetben csupa érvénytelen blokkot tartalmaz)

**5. Feladat** Vegyünk egy 1 kB méretű cache-t 64 byte-os blokkmérettel. A cache kezdetben csupa érvénytelen blokkot tartalmaz. Lefuttatjuk az alábbi programot:

```
short int t[32][32];
int sum = 0;

for (int i=0; i<32; i++)
    for (int j=0; j<32; j++)
        sum += t[i][j];
```

Feltételezések: a `short int` 2 byte-os, a `t` tömb blokkhatáron kezdődik, a két dimenziós tömb a memóriában sor-folytonosan helyezkedik el, a cache direkt leképzést használ. Az i,j változó regiszterekben van tárolva, a cache-t nem terhelik.

(a) Hány cache hibát vált ki a fenti algoritmus? Számolja ki a cache hiba-arányt!

(b) Hány cache hibát vált ki a fenti algoritmus, ha megcseréljük a két ciklust? Számolja ki a cache hiba-arányt!

(c) Mekkora cache-re lenne szükség, hogy a megcserélt for ciklusokkal is ugyanolyan hibaarányt kapjunk, mint az eredeti algoritmussal?

# Számítógép Architektúrák
# Pipeline feladatok

**1. Feladat.** Legyen adott az alábbi utasítás sorozat:

```
i1:  R0 ← MEM [R1+8]
i2:  R2 ← R0 * 3
i3:  R3 ← MEM [R1+12]
i4:  R4 ← R3 * 5
i5:  R0 ← R2 + R4
```

Az utasítássorozatot egy, az órán tanult 5 fokozatú pipeline-t használó processzor hajtja végre. A feladatok a következők:

(a) Azonosítsa az összes előforduló adat-egymásrahatást!

(b) Adja meg az utasítássorozat ütemezését (melyik utasítás mikor melyik fázisban van)! Ha szünetet kell beiktatni, jelezze, hogy mi az oka! hazsnálja az alábbi jelöléseket:

- $A^*$: a szünet oka adategymásrahatás
- $F^*$: a szünet oka feldolgozási egymásrahatás
- $P^*$: a szünet oka procedurális egymásrahatás

(c) Minden egyes utasításnál jelölje, hogy amikor az EX fázisba ér, szükség van-e forwarding-ra! Ha igen, adja meg, hogy melyik pipeline regiszterből kell a a szükséges értéket visszacsatolni!

(d) Ha egységnyi ideig tart minden fázis, mennyivel gyorsabb a pipeline-al való végrehajtás, mint anélkül?

(e) Rendezze át az utasítássorozatot úgy, hogy az a lehető leggyorsabban fusson le, gyorsabban, mint eredetileg!

**2. Feladat.** Legyen adott az alábbi utasítás sorozat:

```
i1:  R5 ← MEM [R3+24]
i2:  R6 ← MEM [R4+16]
i3:  R7 ← R6 + R5
i4:  R8 ← R6 - R5
i5:  R5 ← R7 * R8
i6:  R4 ← R4 + 4
```

Az utasítássorozatot egy, az órán tanult 5 fokozatú pipeline-t használó processzor hajtja végre. A feladatok a következők:

(a) Azonosítsa az összes előforduló adat-egymásrahatást!

(b) Adja meg az utasítássorozat ütemezését (melyik utasítás mikor melyik fázisban van)! Ha szünetet kell beiktatni, jelezze, hogy mi az oka! hazsnálja az alábbi jelöléseket:

- $A^*$: a szünet oka adategymásrahatás
- $F^*$: a szünet oka feldolgozási egymásrahatás
- $P^*$: a szünet oka procedurális egymásrahatás

(c) Minden egyes utasításnál jelölje, hogy amikor az EX fázisba ér, szükség van-e forwarding-ra! Ha igen, adja meg, hogy melyik pipeline regiszterből kell a a szükséges értéket visszacsatolni!

(d) Ha egységnyi ideig tart minden fázis, mennyivel gyorsabb a pipeline-al való végrehajtás, mint anélkül?

(e) Rendezze át az utasítássorozatot úgy, hogy az a lehető leggyorsabban fusson le, gyorsabban, mint eredetileg!

**3. Feladat.** Legyen adott az alábbi utasítás sorozat:

```
i1:  D0 ← D1 * D2
i2:  D3 ← D0 + D5
i3:  MEM [R0+4] ← D3
i4:  MEM [R0+12] ← D0
```

Az utasítássorozatot egy, az órán tanult pipeline-t használó processzor hajtja végre. A D0-D5 regiszterek lebegőpontos számokat, az R0 egész számot tárol. A lebegőpontos összeadás késleltetése 4, a lebegőpontos szorzásé 7, minden egyéb egész műveleté 1, az iterációs idő minden esetben 1. Minden elképzelhető forwarding utat használhat!

(a) Azonosítsa az összes előforduló adat-egymásrahatást!

(b) Adja meg az utasítássorozat ütemezését (melyik utasítás mikor melyik fázisban van)! Ha szünetet kell beiktatni, jelezze, hogy mi az oka! hazsnálja az alábbi jelöléseket:

- $A^*$: a szünet oka adategymásrahatás
- $F^*$: a szünet oka feldolgozási egymásrahatás
- $P^*$: a szünet oka procedurális egymásrahatás

(c) Rendezze át az utasítássorozatot úgy, hogy az a lehető leggyorsabban fusson le, gyorsabban, mint eredetileg!

(d) Adja meg az átrendezett utasítássorozat ütemezését is, a szünetek okának megjelölésével!

**4. Feladat.** Legyen adott az alábbi utasítás sorozat:

```
i1:  D2 ← D0 * D1
i2:  MEM [R0+0] ← D2
i3:  D2 ← D0 + D1
i4:  MEM [R0+8] ← D2
```

Az utasítássorozatot egy, az órán tanult pipeline-t használó processzor hajtja végre. A D0-D5 regiszterek lebegőpontos számokat, az R0 egész számot tárol. A lebegőpontos összeadás késleltetése 4, a lebegőpontos szorzásé 7, minden egyéb egész műveleté 1, az iterációs idő minden esetben 1. Minden elképzelhető forwarding utat használhat!

(a) Azonosítsa az összes előforduló adat-egymásrahatást!

(b) Adja meg az utasítássorozat ütemezését (melyik utasítás mikor melyik fázisban van)! Ha szünetet kell beiktatni, jelezze, hogy mi az oka! hazsnálja az alábbi jelöléseket:

- $A^*$: a szünet oka adategymásrahatás
- $F^*$: a szünet oka feldolgozási egymásrahatás
- $P^*$: a szünet oka procedurális egymásrahatás

MŰEGYETEM 1782

BME

# BELÉPTETŐ RENDSZER TERVEZÉSE

*Számítógép-architektúrák*
*1. gyakorlat*

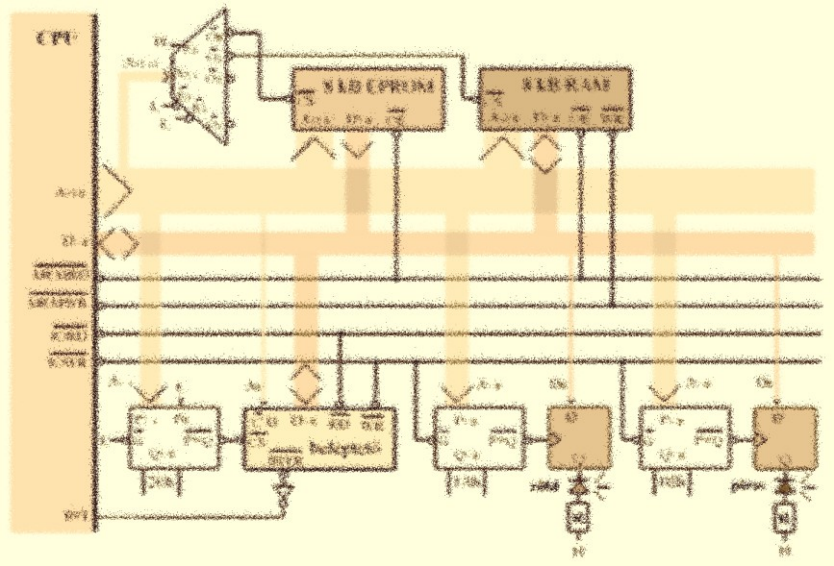

Dr. Lencse Gábor
tudományos főmunkatárs
BME Híradástechnikai Tanszék
lencse@hit.bme.hu

2012. szeptember 3.,
Budapest

# Tartalom

- A rendszer elemeinek megismerése
- A hardver feladat kitűzése
- A hardver megtervezése lépésről lépésre
- Alternatív megoldások vizsgálata
- Összefoglalás


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

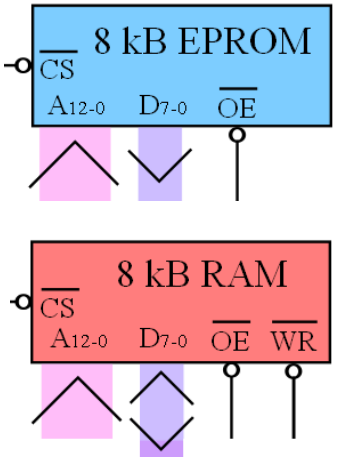


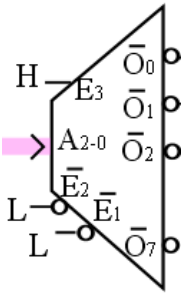
# A RENDSZER ÉPÍTŐELEMEI

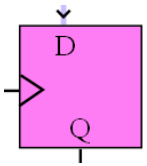

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

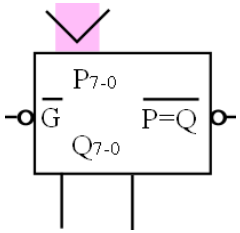
# Adottak a következő építőelemek

- hipotetikus mikroprocesszor
- beléptető periféria
- 8kB EPROM
- 8kB (statikus) RAM
- 3/8-as dekóder
- 2 db D tároló
  - Az órajel felfutó élére tárolja el a D bemenet értékét.
- 3 db 8 bites címkomparátor
  - működése: IF (/G=0)&(P=Q) THEN „/(P=Q)”=0 ELSE „/(P=Q)”=1
- 2 db 250 Ohmos ellenállás
- egy zöld és egy piros LED
  - nyitó irányban 2V feszültségeséssel
  - 5-15 mA áramigénnyel



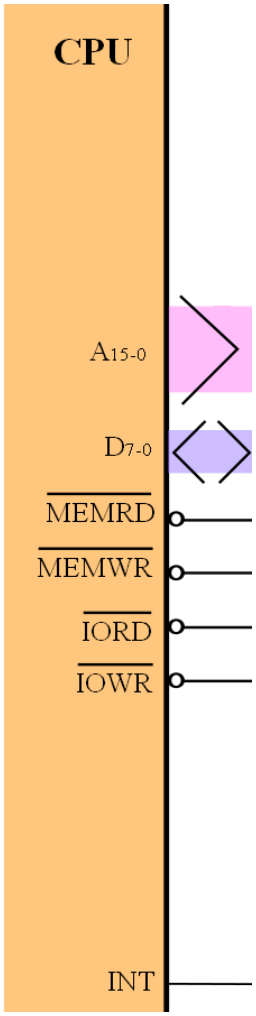



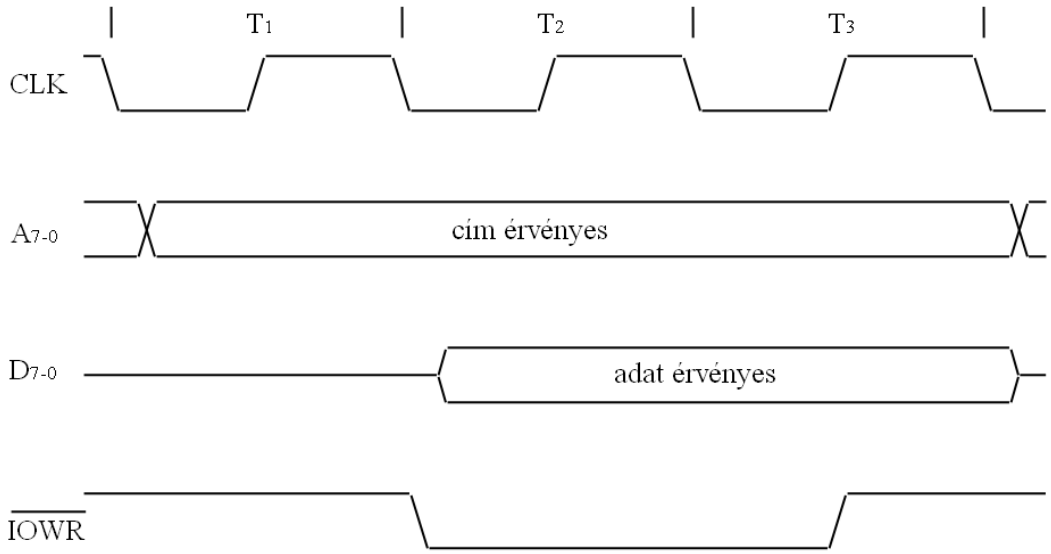
# A mikroprocesszor illesztési felülete

- $A_{15\text{-}0}$ címbusz
  - megcímezhető 64 kB memória, de csak 256 I/O port (az $A_{7\text{-}0}$ címbitekkel címezhető)
- $D_{7\text{-}0}$ adatbusz
- /MEMRD, /MEMWR, /IORD, /IOWR
  - 0 aktív kimenő vezérlő jelek
- INT: felfutó élre érzékeny IT bemenet



# A mikroprocesszor működése

- Működés
  - Kezdetben a 0000h címről indul.
  - Megszakításkor egy CALL 1000h utasítást hajt végre.
    - regisztereket nem ment, a további megszakításokat letiltja.
  - Utasításkészletét később mutatjuk be…
  - A perifériára való írás idődiagramja (az illesztéshez kell):



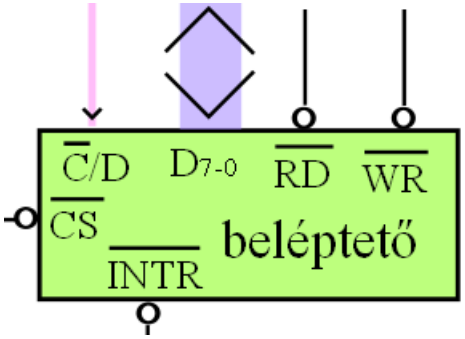
# A beléptető periféria főbb részei

- egy kártyaolvasó
  - a belépőkártyák azonosítója 0-255 egész szám
- egy 0-9 számjegyeket tartalmazó billentyűzet
  - más gomb, pl. törlés nincs
- egy relé vezérlő egység
  - nyitott vagy zárt állásba vezérelhető a zár nyelve
  - az ajtó zárt állásban is becsukódik, de csak nyitott állásban nyitható ki
- a processzor felé való illesztési felület


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A beléptető periféria illesztési felülete

- $D_{7-0}$: kétirányú adatbusz
- /CS, /RD és /WR: 0 aktív bemenő vezérlő jelek,
- (/C)/D (Command=0/Data=1): választó bemenet, hatása:
  - (/C)/D=0 érték esetén: $D_{7-0}$ olvasáskor státusz, íráskor zárvezérlés
  - (/C)/D=1 érték esetén: az adatregiszterből az utolsó eseményhez tartozó számérték (utoljára lehúzott kártya azonosítója vagy utoljára megnyomott számjegy) olvasható ki, az adatregiszter írása hatástalan.
- /INTR: alacsony logikai szinttel jelző esemény kimenet



# A beléptető periféria működése – 1

- Minden kártyához (kártyaazonosítóhoz) tartozik egy 4 decimális jegyből álló kód: a zárat akkor kell nyitni, ha a kártya lehúzása után megnyomott első 4 billentyű éppen a hozzá tartozó kódot adja. (Időzítéssel nem foglalkozunk!) A feladat megoldásához felhasználjuk a beléptető periféria alábbi funkcióit:
  - Amikor a kártyát lehúzzák, a beléptető /INTR lába alacsony szintűre vált, és a státuszregiszteréből olvasva a $D_0$ bit 1-es értékű lesz (akárhányszor is kiolvasható): ezek mindaddig fennállnak, amíg az adatregiszterből ki nem olvasták a kártya azonosítóját. Ezután adatregiszterből kiolvasható (csak egyszer!) a lehúzott kártya azonosítója, utána rögtön /INTR magasra vált, és a státuszregiszteréből olvasva a $D_0$ bit értéke 0 lesz.

(folytatjuk…)

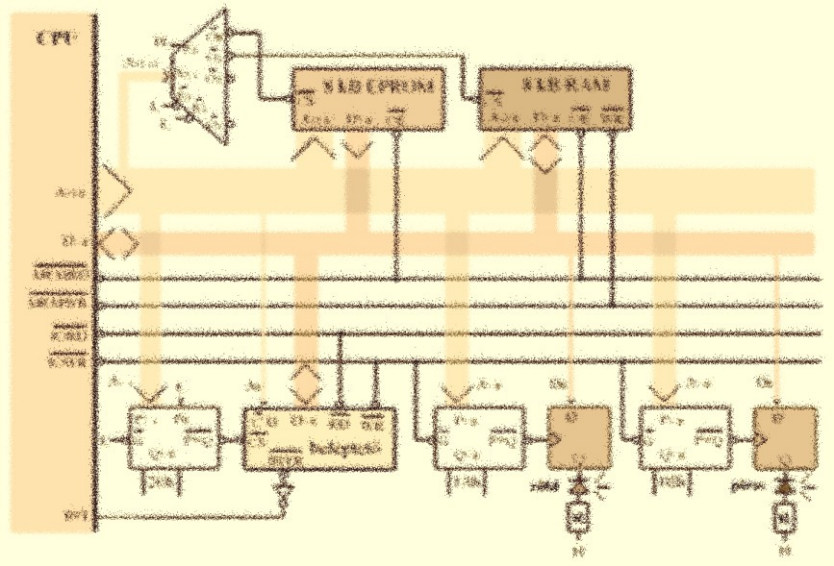

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A beléptető periféria működése – 2

(…folytatás)

- Amikor egy számjegyet beütöttek, a beléptető /INTR lába alacsony szintűre vált, és a státuszregiszteréből olvasva a $D_1$ bit 1-es értékű lesz (akárhányszor is kiolvasható): ezek mindaddig fennállnak, amíg az adatregiszterből ki nem olvasták a számjegy értékét. Ezután az adatregiszterből kiolvasható (csak egyszer!) a beütött számjegy értéke, utána rögtön /INTR magasra vált, és a státuszregiszteréből olvasva a $D_1$ bit értéke 0 lesz.
- A vezérlő regiszter $D_2$ bitjével állítható a zár állása: 0: zárás parancs, 1: nyitás parancs. A nyitás parancs után, amint az ajtót kinyitották rögtön, de legkésőbb 10s után (timeout: ha az ajtót addig nem nyitották ki) a beléptető zárnyelv vezérlője automatikusan átmegy zárt állapotba.
- A státuszregiszter $D_2$ bitjéből mindig kiolvasható a zár állása: 0: zárva, 1: nyitva


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

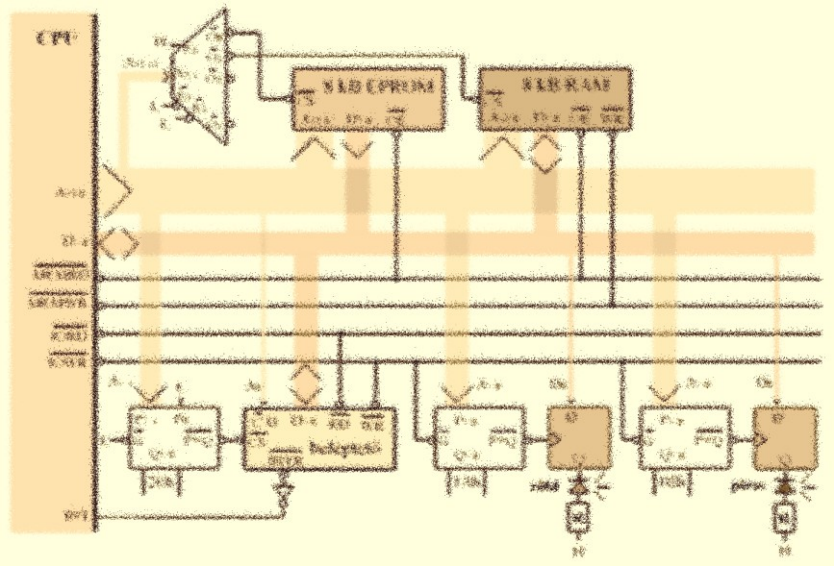



# A HARDVER SPECIFIKÁCIÓJA

# Az építendő hardver

- Illesszünk a hipotetikus mikroprocesszorhoz
  - 8kB EPROM-ot a 0000h címre
  - 8kB RAM-ot a 2000h címre
  - a beléptető perifériát a 2Eh báziscímre
    - így 2Eh/2Fh címeken érhető el
  - egy zöld és egy piros LED-et az 1Ah és az 1Bh portcímekre
    - D tárolók és ellenállások segítségével
    - az adatbusz $D_0$ bitjével lehessen az állapotukat vezérelni


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

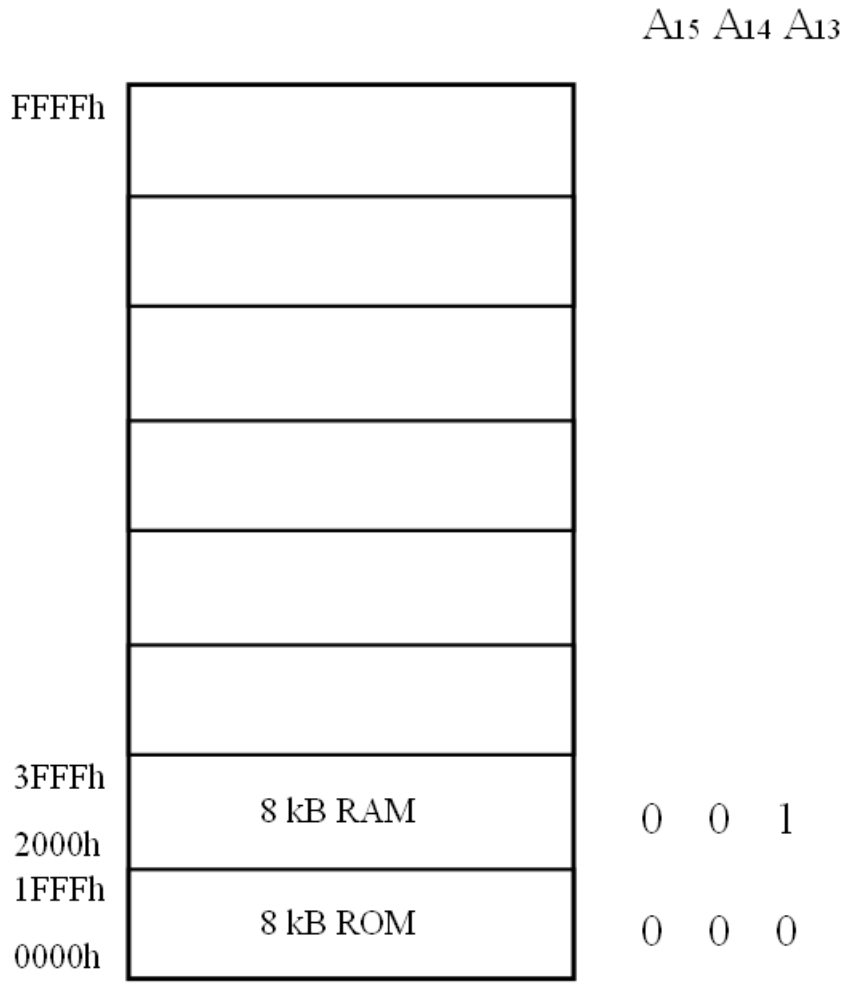


# A HARDVER TERVEZÉSÉNEK LÉPÉSEI


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

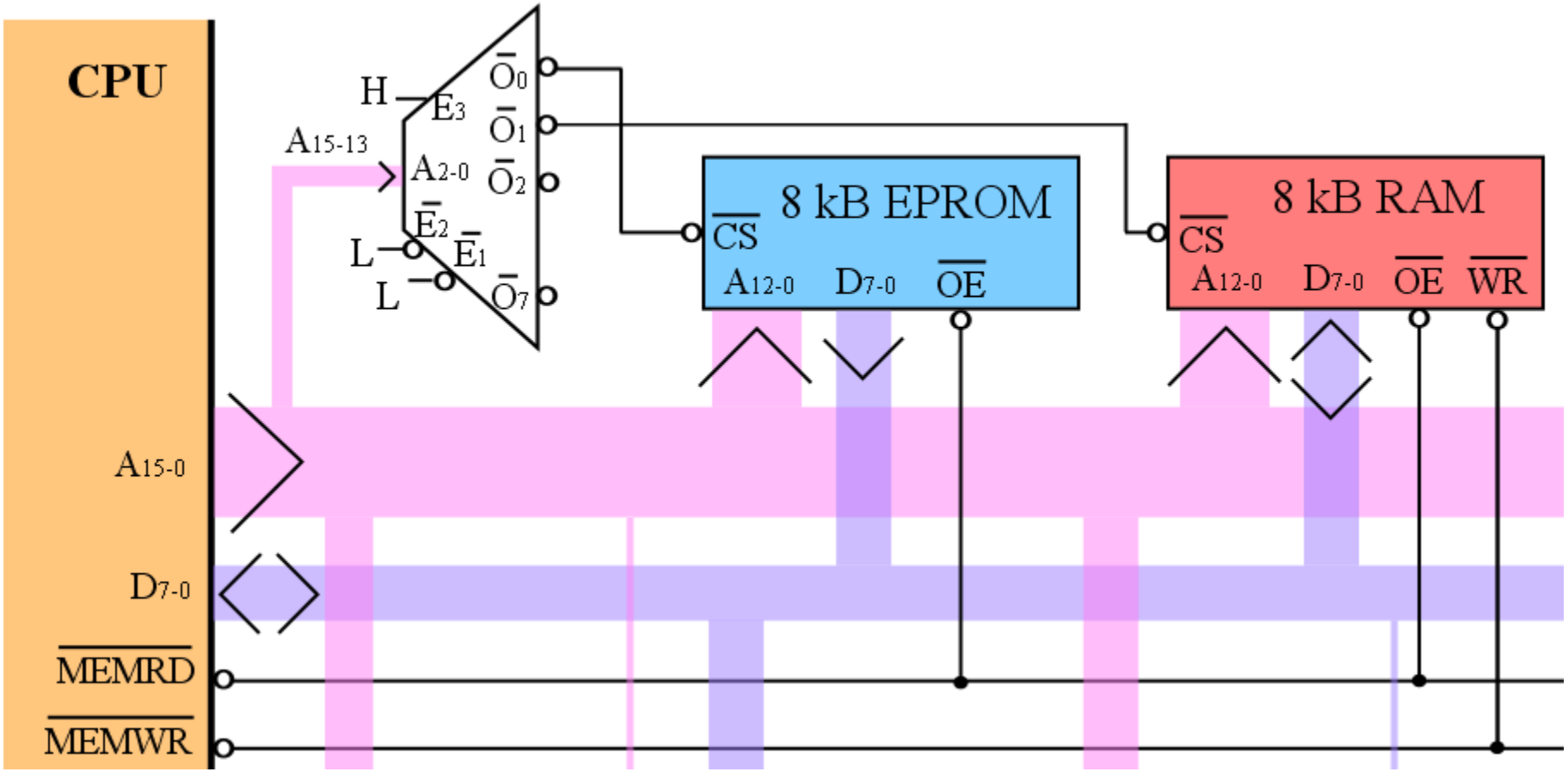
# Memóriák illesztése

- Készítsünk memóriatérképet!
  - Mellette: $A_{15\text{-}13}$ címbitek
  - egy 3/8-as dekóderrel az EPROM és a RAM illeszthető:
    - EPROM: $A_{15\text{-}13}$=000
    - RAM: $A_{15\text{-}13}$=001
    - 3/8-as dekóder $/O_0$ és $/O_1$
  - Méretük 8kB
    - 13 címbit kell: $A_{12\text{-}0}$



# Memóriák illesztése

- A CPU és a hozzá illesztett memóriák

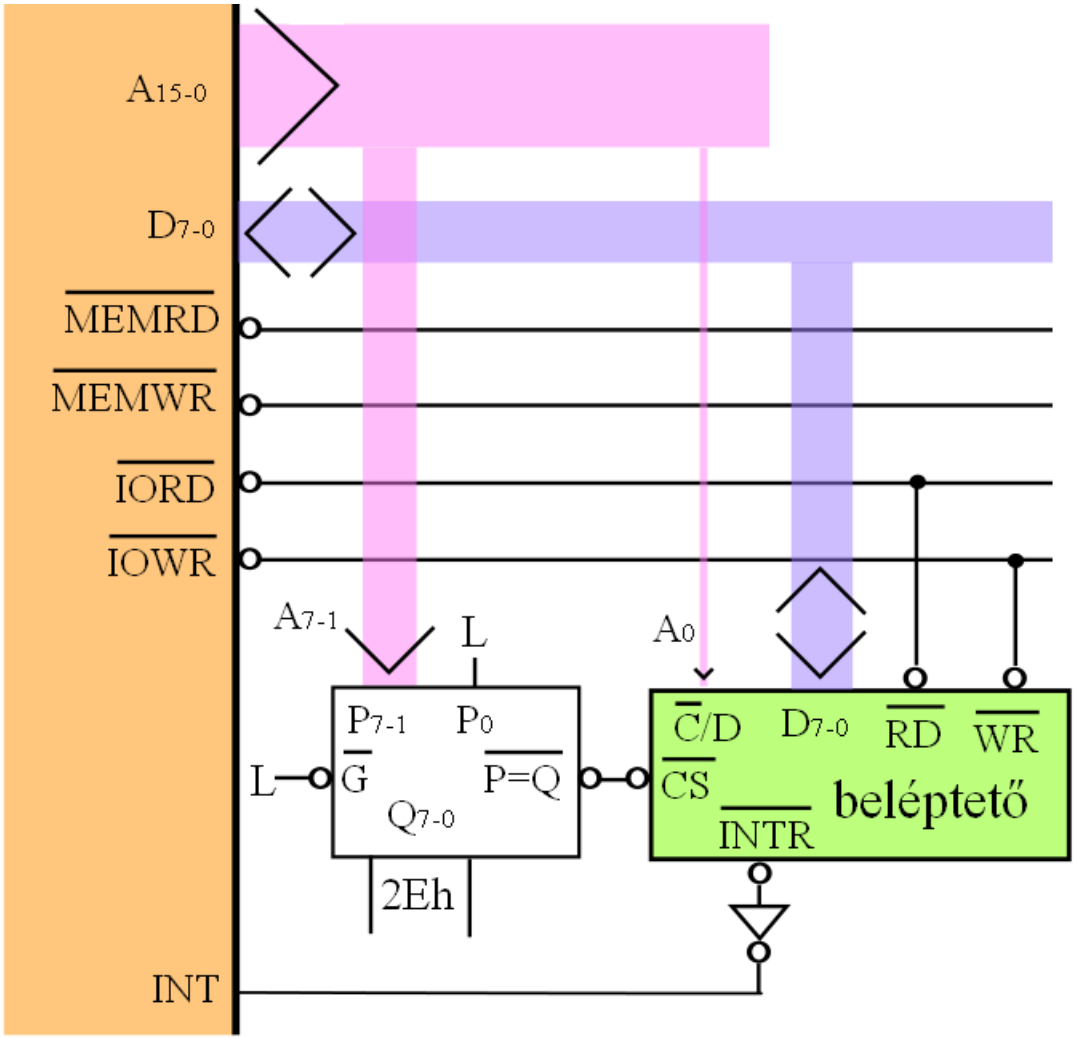


# A beléptető periféria illesztése

- A beléptető perifériát címkomparátorral a 2Eh báziscímre illesztjük:
  - a címkomparátor $P_{7-1}$ bemenetére a címbusz $A_{7-1}$ bitjeit kötjük, de az $A_0$ címbit helyett $P_0$-ra fixen logikai 0-t (L) kötünk, $Q_{7-0}$ bemenetére pedig az 2Eh értéket kapuzzuk.
  - A címkomparátor /G engedélyező bemenetét fixen logikai 0-ra (L) kötjük.
  - A címkomparátor 0 aktív /(P=Q) kimenetével engedélyezzük a beléptető perifériát annak /CS bemenetén.
- Az $A_0$ címbitet rákötjük a beléptető periféria (/C)/D bemenetére;
- az /IORD, /IOWR, és $D_{7-0}$ jeleket pedig értelemszerűen a beléptető periféria /RD, /WR, és $D_{7-0}$ lábaira kötjük.
- A periféria /INTR kimenetét inverteren keresztül kötjük a CPU felfutó élre érzékeny INT lábára.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A beléptető periféria illesztése





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# LED-ek meghajtása

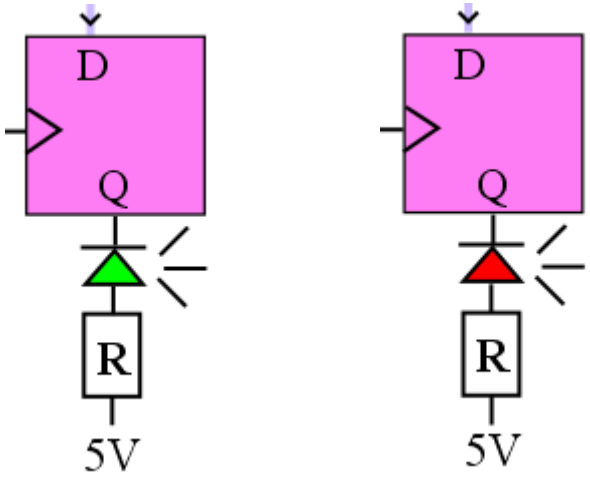
- A LED-ek meghajtásához fontoljuk meg:
  - A D tárolók TTL alkatrészek, kimeneti áram terhelhetőségük
    - logikai 1-nél: < 1mA
    - logikai 0-nál: 20mA környékén

    → csak logikai 0 esetén tudják a LED-eket meghajtani
  - Ha az egy LED-en nyitó irányban eső feszültség 2V, további veszteségekkel (kb. 1V) számolva a 250 Ohm-os áramkorlátozó ellenálláson még kb. 2V feszültségesés 8mA áramot biztosít; ez a LED-ek számára megfelelő.
  - Tehát a LED-eket a D tárolók kimenetéről 250-os ellenálláson keresztül tápfeszültségre (5V) kötjük.
  - FIGYELEM! Így a LED-ek kigyújtásához 0-t kell a D tárolókba írnunk, kioltásához pedig 1-et!


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# LED-ek meghajtása

- Ügyeljünk, hogy a LED-ek rajzjele az elméleti áramirányt kövesse!

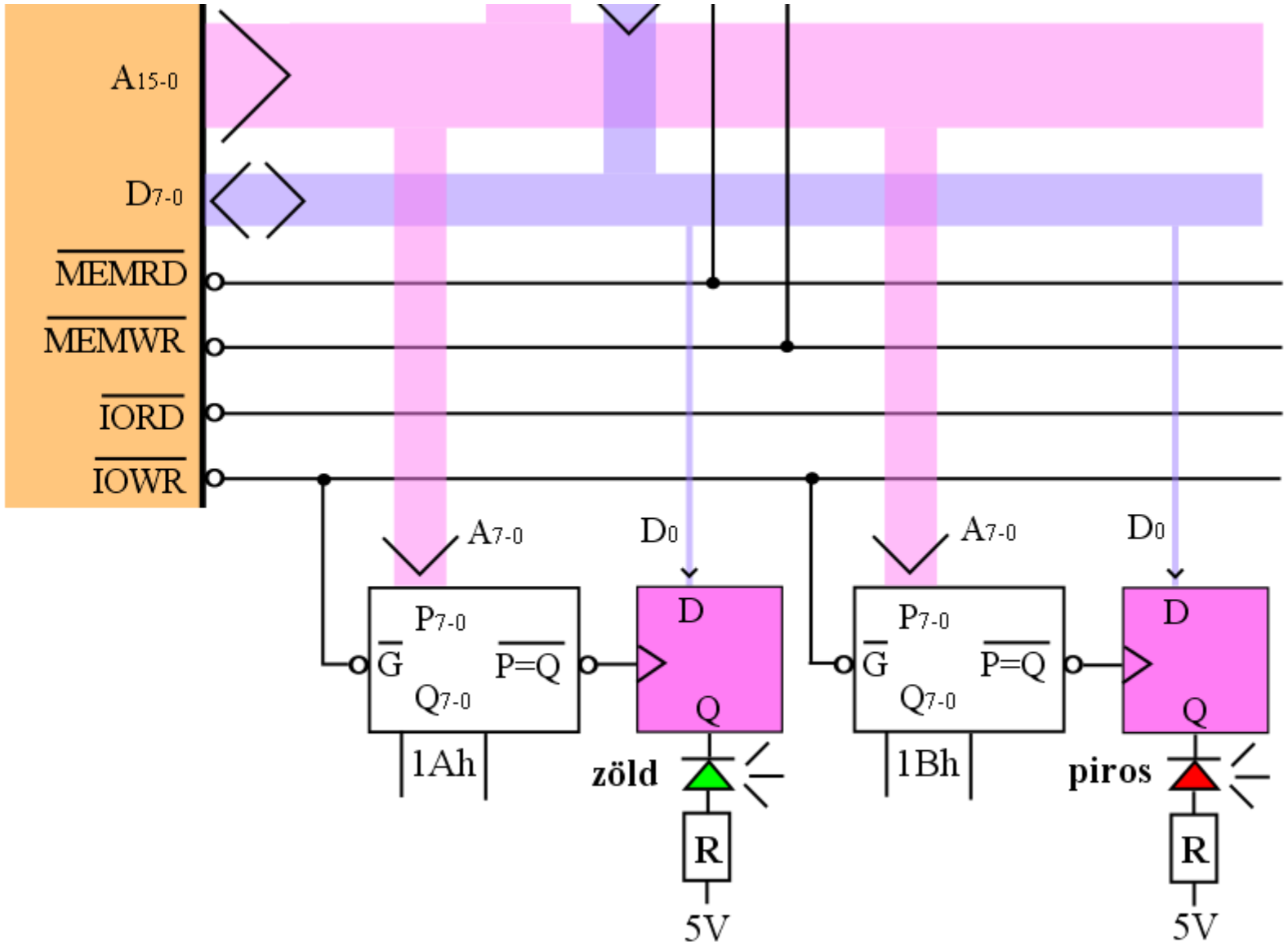


# D tárolók illesztése

- A D tárolókat az 1Ah és az 1Bh portcímekre illesztjük címkomparátorokkal.
  - A címkomparátorok $P_{7-0}$ bemenetére a címbusz $A_{7-0}$ bitjeit kötjük, $Q_{7-0}$ bemenetére pedig az 1Ah illetve 1Bh értékeket kapuzzuk.
  - A címkomparátorok engedélyezéséhez a /G engedélyező bemenetükre a CPU /IOWR kimenetét kötjük, mivel a tárolókat csak írnunk kell.
  - A címkomparátorok 0 aktív /(P=Q) kimenetét a D tároló felfutó élre érzékeny órajel bemenetére kötjük; így az /IOWR megszűnésekor a felfutó él hatására a kiválasztott D tároló eltárolja a D bemenetére kötött adatbusz $D_0$ bitjének ekkor még stabilan tartott értékét.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

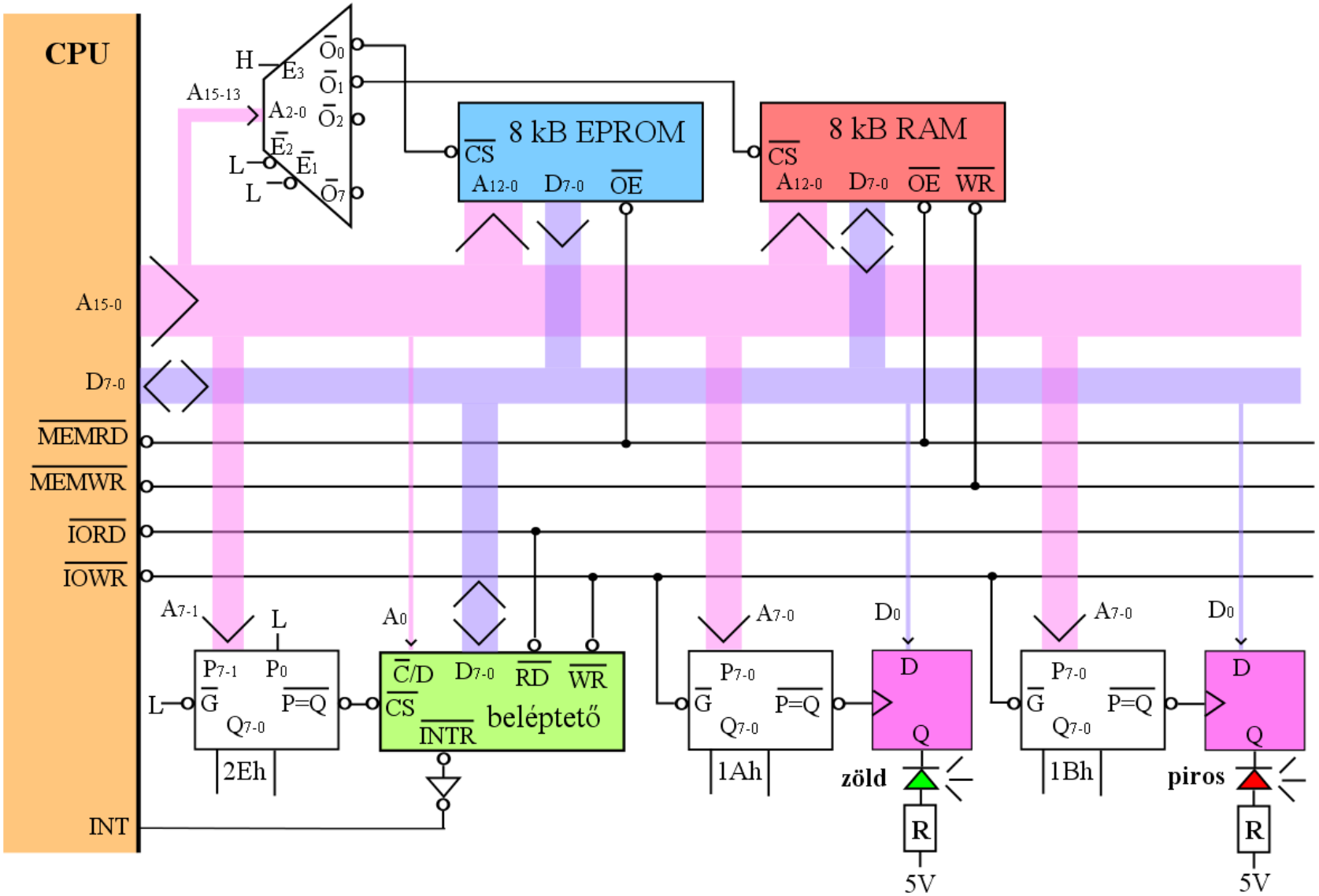
# D tárolók illesztése





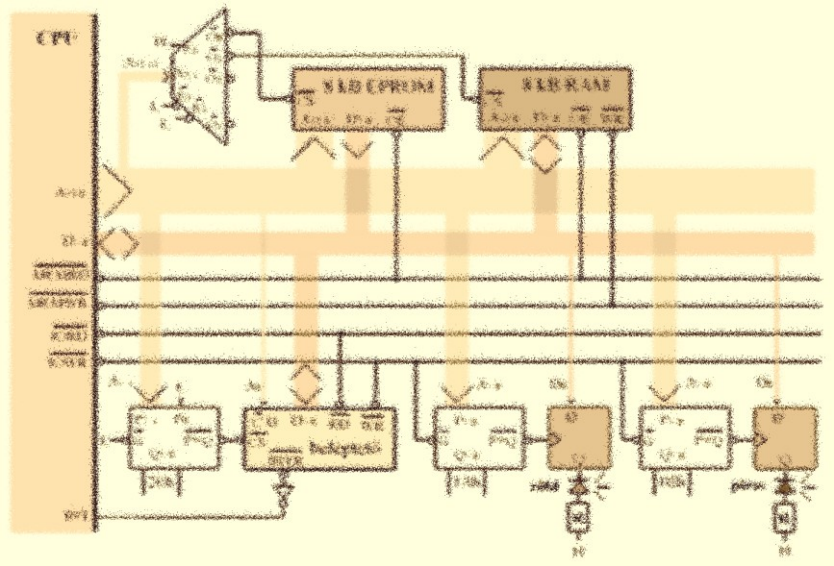
© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A teljes logikai kapcsolási rajz





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem




# ALTERNATÍV MEGOLDÁSOK VIZSGÁLATA

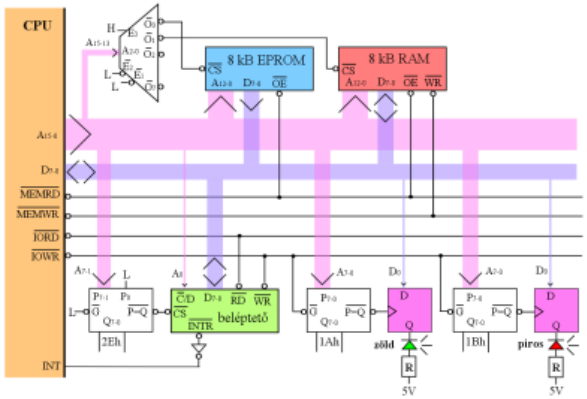
# Önálló hallgatói munka

- **Gondolkozzanak el önállóan az alábbi kérdéseken!**
  1. Hogyan módosítaná a hardvert, ha a 2000h címre nem 8 kB, hanem 16 kB RAM-ot kellene illesztenie?
  2. Milyen megoldást választana, ha a feladat úgy szólna, hogy a LED-eknek akkor kell világítani, ha vezérlésükhöz használt perifériacím 0. bitjére utoljára logikai 1 értéket írtunk és nem világítani, ha 0 értéket írtunk?
  3. Hogyan tudná egyszerűsíteni a rendszert, ha a specifikáció azt írja elő, hogy a piros és a zöld LED állapota mindig egymással ellentétes?


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Összefoglalás

- Megismerkedtünk egy beleptető rendszer építéséhez használható hipotetikus és valós építőelemekkel
- Megadtunk egy konkrét specifikációt
- Megterveztük a kívánt hardvert
  - Memóriát illesztettünk
  - Perifériát illesztettünk
  - LED-ek meghajtásáról gondoskodtunk
  - D tárolókat illesztettünk (csak írásra)




- Alternatív megoldásokat is megvizsgáltunk


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Kérdések?

## KÖSZÖNÖM A FIGYELMET!

Híradástechnikai Tanszék

Dr. Lencse Gábor
tudományos főmunkatárs
BME Híradástechnikai Tanszék
lencse@hit.bme.hu

MŰEGYETEM 1782

BME

# BELÉPTETŐ RENDSZER TERVEZÉSE

*Számítógép-architektúrák*
*2. gyakorlat*

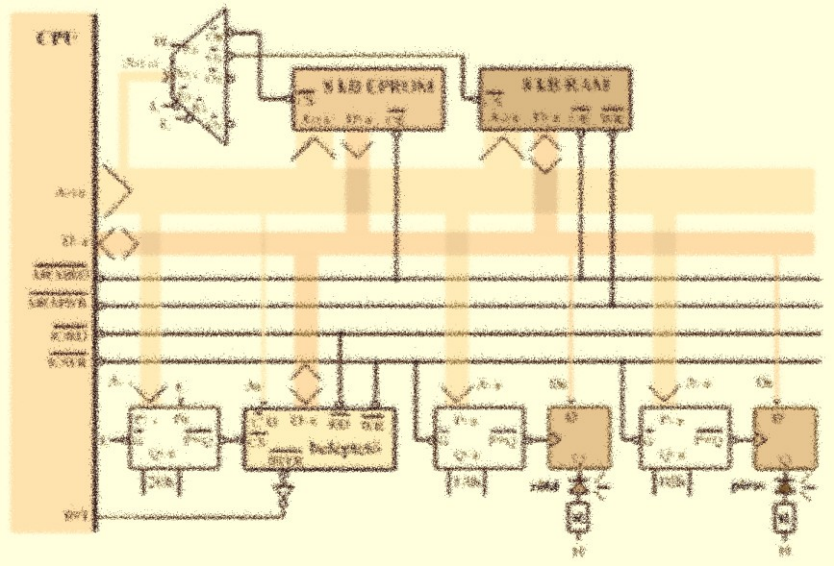

Dr. Lencse Gábor
tudományos főmunkatárs
BME Híradástechnikai Tanszék
lencse@hit.bme.hu

2012. szeptember 3.,
Budapest

# Tartalom

- Ismétlés
  - a beléptető periféria működése
  - a múlt héten elkészült hardver
- A szoftver feladat kitűzése
- A szoftver tervezésének menete (tervezési döntések)
- A hipotetikus processzor utasításkészlete
- Folyamatábrák rajzolása
- Assembly nyelvű program írása
- Alternatív megoldások vizsgálata
- Összefoglalás


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



# ISMÉTLÉS: A BELÉPTETŐ PERIFÉRIA MŰKÖDÉSE + A KÉSZ HARDVER


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A beléptető periféria működése – 1

- Minden kártyához (kártyaazonosítóhoz) tartozik egy 4 decimális jegyből álló kód: a zárat akkor kell nyitni, ha a kártya lehúzása után megnyomott első 4 billentyű éppen a hozzá tartozó kódot adja. (Időzítéssel nem foglalkozunk!) A feladat megoldásához felhasználjuk a beléptető periféria alábbi funkcióit:
  - Amikor a kártyát lehúzzák, a beléptető /INTR lába alacsony szintűre vált, és a státuszregiszteréből olvasva a $D_0$ bit 1-es értékű lesz (akárhányszor is kiolvasható): ezek mindaddig fennállnak, amíg az adatregiszterből ki nem olvasták a kártya azonosítóját. Ezután adatregiszterből kiolvasható (csak egyszer!) a lehúzott kártya azonosítója, utána rögtön /INTR magasra vált, és a státuszregiszteréből olvasva a $D_0$ bit értéke 0 lesz.

(folytatjuk…)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

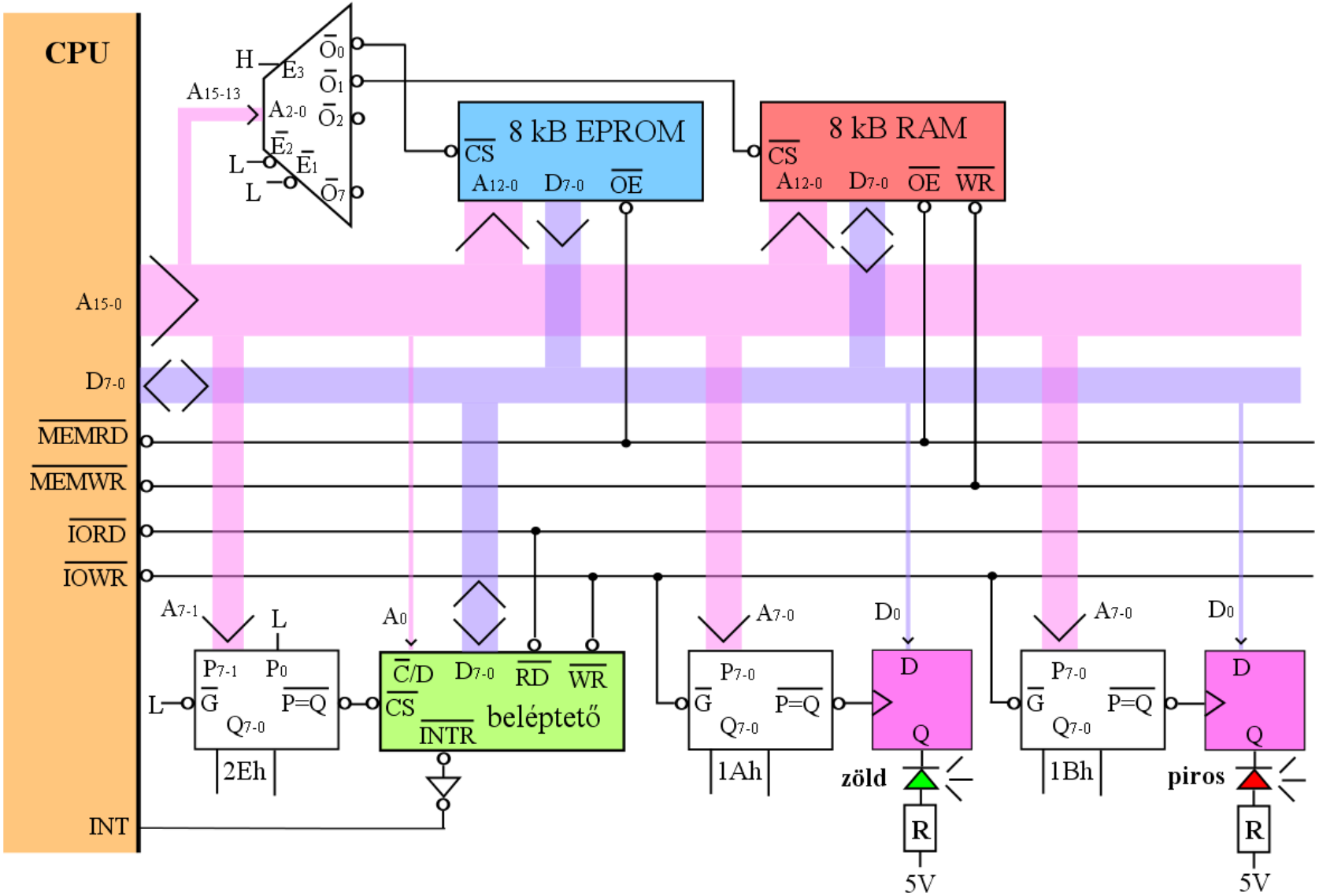
# A beléptető periféria működése – 2

(…folytatás)

- Amikor egy számjegyet beütöttek, a beléptető /INTR lába alacsony szintűre vált, és a státuszregiszteréből olvasva a $D_1$ bit 1-es értékű lesz (akárhányszor is kiolvasható): ezek mindaddig fennállnak, amíg az adatregiszterből ki nem olvasták a számjegy értékét. Ezután az adatregiszterből kiolvasható (csak egyszer!) a beütött számjegy értéke, utána rögtön /INTR magasra vált, és a státuszregiszteréből olvasva a $D_1$ bit értéke 0 lesz.
- A vezérlő regiszter $D_2$ bitjével állítható a zár állása: 0: zárás parancs, 1: nyitás parancs. A nyitás parancs után, amint az ajtót kinyitották rögtön, de legkésőbb 10s után (timeout: ha az ajtót addig nem nyitották ki) a beléptető zárnyelv vezérlője automatikusan átmegy zárt állapotba.
- A státuszregiszter $D_2$ bitjéből mindig kiolvasható a zár állása: 0: zárva, 1: nyitva

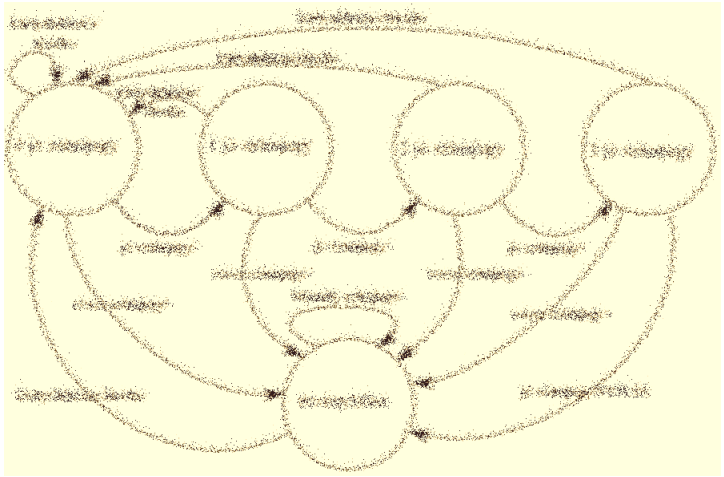

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A teljes logikai kapcsolási rajz





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



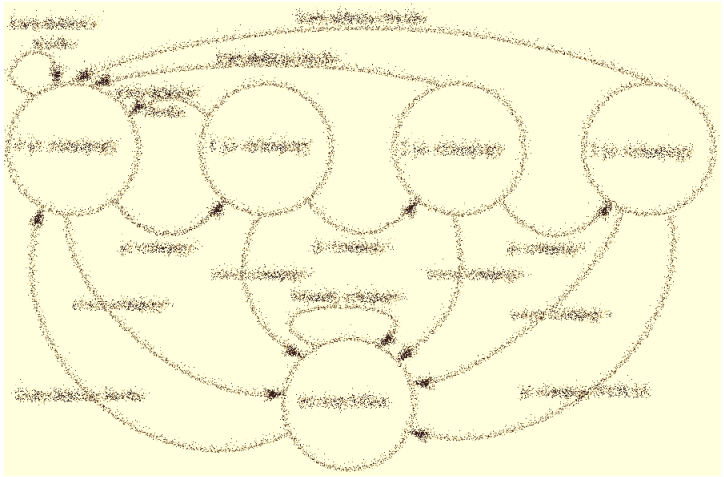
# A SZOFTVER FELADAT KITŰZÉSE


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A készítendő szoftver feladata

- A beléptető periféria zárja kezdetben zárt állásban van (így indul, ez nem a mi dolgunk).
- Követelmény, hogy:
  1. A LED-eknek mindenkor tükrözniük kell a zár állapotát (beleértve a nyitást, az automatikus zárást, valamit az ajtó fizikai nyitása által kiváltott zárást is).
  2. Kártyalehúzást követő helyes 4 jegyű kód megadása esetén a zárat nyitni kell (függetlenül attól, hogy éppen zárt vagy nyitott állásban van). Minden más bemeneti szekvencia esetén a zárat nem szabad kinyitni.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



# A szoftver tervezésének menete (tervezői döntések)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Előfeltevés, észrevétel

- Előfeltevés
  - Éljünk azzal az előfeltevéssel, hogy a kártyalehúzás és a számjegyleütés események közül egyszerre csak az egyik következik be! (A rendszer működése kellően gyors a felhasználóhoz képest.)
- Észrevétel
  - Mivel a zárásról semmiféle értesítést sem kapunk, az 1. követelményt csak úgy tudjuk kielégíteni, ha folytonosan kiolvassuk a zár állapotát és (szükség esetén)* beállítjuk a LED-eket.

* - pontosítandó, hogy csak <u>szükség esetén</u> vagy <u>mindig</u> (lásd a következő oldalon)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# További észrevételek

- Észrevételek:
  1) A zár állapotát lekérdező ciklusban a LED-ek állapotát
     a) minden lekérdezés után beállíthatjuk
     b) az előző állapotukat a programban nyilvántartva elegendő csak változás esetén állítanunk
  2) A zár állapotát lekérdező ciklusban a zár állapota alapján
     a) a LED-ek állapotát zártra és nyitottra is állíthatjuk (ekkor a kártyakód és a számjegyek alapján elegendő a zárnak nyitás parancsot adnunk és a LED-ekkel nem kell foglalkoznunk)
     b) a LED-ek állapotát elegendő csak zártra állítanunk, ha a zár nyitottra állításakor elvégezzük a LED-ek nyitottra állítását is
  3) A kártyalehúzás és a számjegyek leütésének figyelésére
     a) megszakítást használhatunk
     b) folytonos állapotolvasást végezhetünk


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Tervezői döntések (a felvetésekre)

- A következő önkényes tervezői döntéseket hoztuk
  1) A zár állapotát lekérdező ciklusban a LED-ek állapotát minden lekérdezés után beállítjuk.
     (Így az állapotukat nem kell nyilvántartanunk.)
  2) A zár állapotát lekérdező ciklusban a zár állapota alapján a LED-ek állapotát zártra és nyitottra is állítjuk.
     (Így csak itt állítunk LED-eket, a nyitás parancs kiadásakor a LED-ek állításával nem kell foglalkoznunk.)
  3) A kártyalehúzás és a számjegyek leütésének figyelésére megszakítást használunk.
     (Ezzel a döntéssel elestünk annak a lehetőségétől, hogy a programszámlálót használjuk a program állapotának a tárolására!)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# További tervezői döntések – 1

- Használjunk a program állapotának tárolására egy állapotváltozót, nevezzük AV-nek.
- Hordozza AV azt az értéket, hogy az utolsó kártyalehúzást követően hány helyes számjegy érkezett (lehetséges értékek 0-tól 4-ig) és egy másik számértékkel kódoljuk azt, ha nem volt még kártyalehúzás vagy érvénytelen számjegy jött.
- Könnyen belátható, hogy csak egy ilyen "érvénytelen" állapotra van szükség, ugyanis csak egyfajta esetben lehet folytatás: ha kártyalehúzás jön.

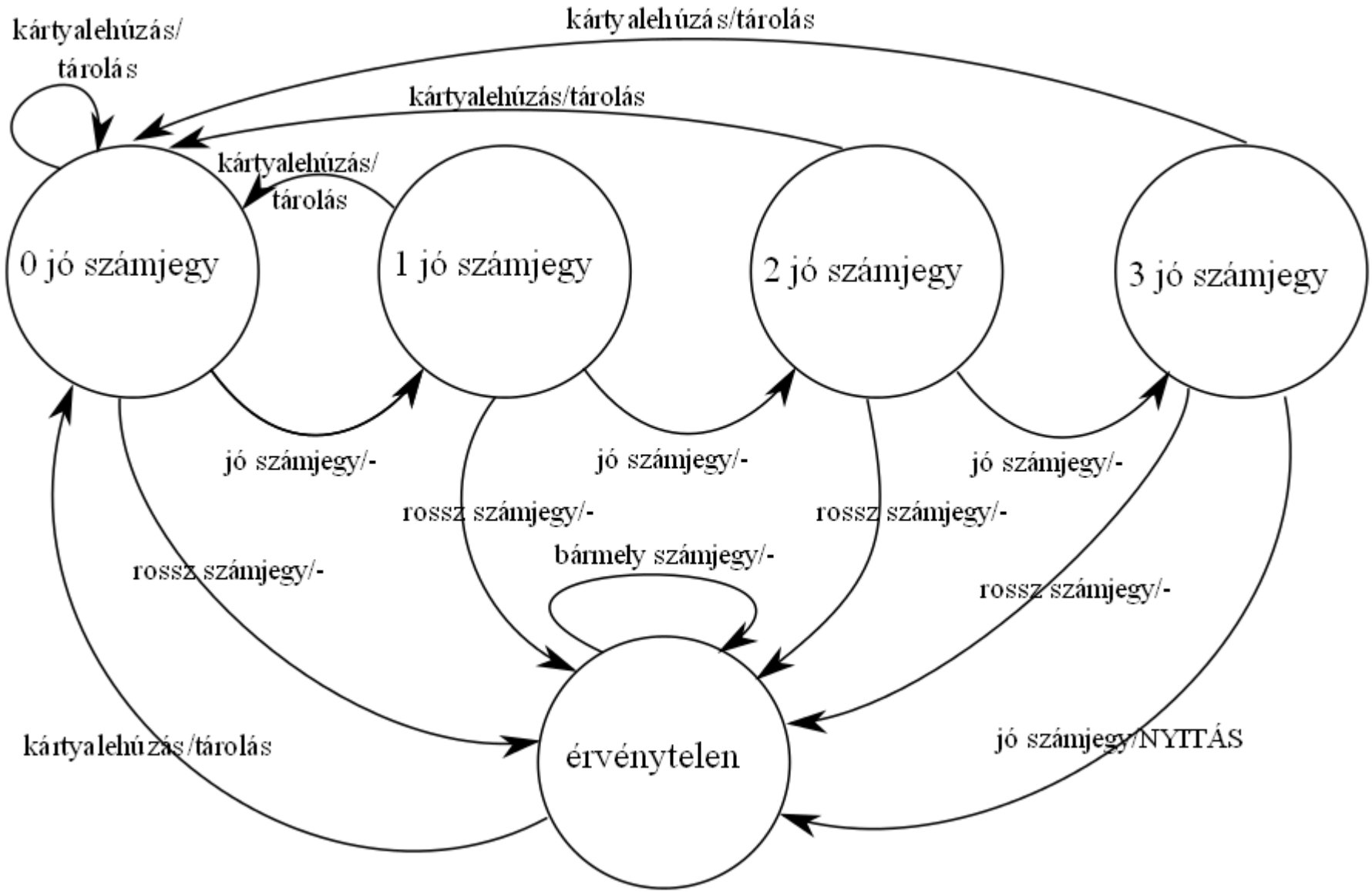

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# További tervezői döntések – 2

- Az is nyilvánvaló, hogy az AV=4 esetet sem szükséges megkülönböztetnünk az érvénytelentől, mert ha egyszer már elvégeztük a nyitást, a továbbiakban ez az állapot nem különbözik az érvénytelentől (ismét kártyalehúzásra kell várnunk).
- Tehát AV={0, 1, 2, 3, ERVTLN}
- Készítsük el az állapotgráfot!


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A beléptető rendszer állapotgráfja





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Hogyan tovább?

- Az állapotgráfnak megfelelő véges automatát formálisan is lekódolhatnánk, de helyette inkább gondolkozzunk egy kicsit, mert az megtérül a programozáskor! ;-)
- Az inputnak két fajtája lehetséges: kártyalehúzás vagy számjegy érkezése.
  - Látható, hogy a kártyalehúzás az adott állapottól függetlenül mindig AV:=0-t eredményez, és tárolni kell a kártya azonosítóját.
  - Csak számjegy érkezése esetén érdekes az AV értéke. Ekkor
    - ha az AV értéke ERVTLN, akkor nincs teendőnk
    - különben szükségképpen AV eleme {0, 1, 2, 3}, ekkor
      - ha a beérkezett számjegy megfelelő, akkor az AV értéke eggyel nő;
      - ha nem megfelelő, akkor AV:=ERVTLN lesz.

    ha ezek után AV értéke 4 lett, akkor NYITÁS parancsot adunk és AV:=ERVTLN.

# Adatszerkezetek

- KODTABLA: 4 db 256 byte-os táblázat egymás után: az i. táblázat j. karaktere a j. kártyaazonosítóhoz tartozó i. számjegy; j=0-255, i=0-3
- AV: egy darab státusz bájt, melynek jelentése:
  - i=0-3: a legutolsó kártyaolvasás után már jött i db számjegy és az jó volt
    (Vegyük észre, hogy itt i=0 éppen úgy kezelhető mint i=1-3!)
  - legyen az értéke 255 minden egyéb esetben (ERVTLN)
- K_AZON: egy bájt, ami az utolsó kártyaazonosító értékét tárolja, de csak AV=0-3 esetén érvényes


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Még egy tervezői döntés

- Igazítsuk a KODTABLA kezdetét 256 bájtos laphatárra!
  - Ez egyszerűsíti az elemeinek a címzését. A mindig használható KODTABLA_KEZDOCIME+256*AV+K_AZON cím 16 biten való kiszámítása helyett így, ha a KODTABLA AV-edik 256 bájtos lapján keressük a K_AZON-adik számjegyet, akkor elegendő AV-t a KODTABLA kezdőcímének magasabb helyiértékű bájtjához hozzáadni, K_AZON értékét pedig KODTABLA kezdőcímének alsó helyértékű (a laphatárra igazítás miatt 0 értékű) bájtja helyett használni.
  - Ez csak egy trükk, természetesen szükség esetén a 8 bites aritmetikából felépíthető a 16 bites aritmetika! ☺


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A főprogram működése

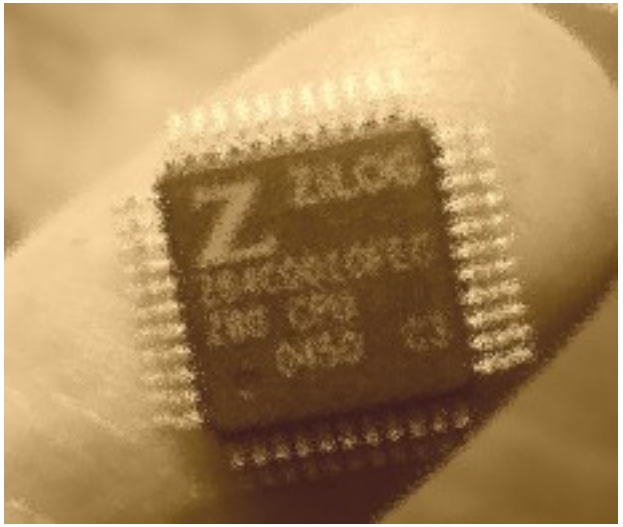
- A főprogram
  - beállítja a verem helyét
  - az állapotváltozó értékét
  - engedélyezi a megszakításokat
  - végtelen ciklusban
    - kiolvassa a zár állapotát
    - megjeleníti a LED-eken.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A megszakítási rutin működése

- Az IT rutin menti a használt regisztereket, majd:
  - Ha kártyalehúzás jött, akkor az azonosítót tárolja és AV:=0;
  - Különben számjegy jött (mert másért nem lett volna IT), tehát:
    - ha AV=255, akkor a számjegyet eldobjuk, állapot változatlan,
    - ellenkező esetben csakis 0-3 állapot lehet, így: ellenőrizzük, hogy jó-e a számjegy,
      - ha nem: AV:=255;
      - ha igen: AV++; amennyiben az állapot 4 lett, akkor: NYITÁS (csak a perifférián) és AV:=255;
- Végül az IT rutin visszatölti a mentett regiszterek értékét, engedélyezi a megszakítást és visszatér.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



# A HIPOTETIKUS PROCESSZOR UTASÍTÁSKÉSZLETE


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Regiszterkiosztás

- A hipotetikus processzornak a következő regiszterei vannak: A, F, B, C, D, E, H, L regiszterek 8 bitesek, és az SP regiszter 16 bites az alábbi értelmezésekkel:
  - A - akkumulátor: kitüntetett regiszter, az aritmetikai és logikai műveletek egyik operandusaként használjuk, és a műveletek eredménye is benne képződik; valamint az IN (perifériáról való bevitel) és OUT (perifériára való kivitel) utasítások implicit operandusa is ez.
  - F - jelzőbitek regisztere: a jelzőbitek közül csak a Z-t használjuk, értéke pontosan akkor 1, ha az utolsó aritmetikai vagy logikai művelet eredménye 0. Az A regiszterrel együtt az AF regiszterpárt alkotja, ami verem műveleteknél együtt kezelhető.
  - BC, DE, HL - regiszterpárok: együtt 16 bites regiszterként használhatók, de az egyes regiszterek külön-külön is használhatók.
  - SP - veremmutató


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Címzési módok – 1

- Akkumulátor címzés: az egyik operandus és a művelet eredménye implicit módon az akkumulátor (de nem nevezzük meg). Ezzel találkozunk az aritmetikai és logikai műveleteknél valamint az IN/OUT utasításoknál.

  AND B ; A ← A&B

  OUT 38h ; $port_{38h}$ ← A

- Regisztercímzés: operandusként regisztert vagy regiszterpárt adunk meg.

  LD A, B ; A ← B

- Közvetlen adatcímzés (immediate): közvetlenül az utasítás kódja után, az utasítás részeként szerepel az operandus

  LD B, 10 ; B ← 10


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Címzési módok – 2

- Direkt memóriacímzés: az operandus memóriabeli címét adjuk meg zárójelben (azért kell a zárójelpár, hogy a közvetlen adatcímzéstől meg tudjuk különböztetni)

  LD (3000h), D ; MEM[3000h] ← D

- Indirekt címzés: a címet tartalmazó regiszterpár zárójelben megadva szerepel (itt is azért kell a zárójelpár, hogy a regisztercímzéstől meg tudjuk különböztetni).

  LD A,(HL) ; A ← MEM[HL]


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Felhasznált mnemonikok – 1

- LD: adatmozgatás: hova, mit
- PUSH / POP: regiszter vagy regiszterpár értékének mentése a verembe / visszatöltése a veremből
- IN / OUT: a megadott portról bevitel vagy oda kivitel
- AND: bitenkénti logikai ÉS művelet a megadott operandus és az akkumulátor között
- ADD: 8 bites összeadás a megadott operandus és az akkumulátor között
- INC: a megadott regiszter értékének növelése 1-gyel
- CMP: a jelzőbitek állítása az
  "akkumulátor tartalma mínusz a megadott érték"
  művelet eredménye szerint
  (de az akkumulátor tartalma nem változik meg)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Felhasznált mnemonikok – 2

- JMP: feltétel nélküli ugrás (a megadott címre)
- JZ / JNZ: ugrás, ha az előző aritmetikai vagy logikai művelet eredménye 0 / nem 0
- CALL: szubrutin hívása
- RET: szubrutinból vagy megszakításból való visszatérés
- EI: megszakítások engedélyezése

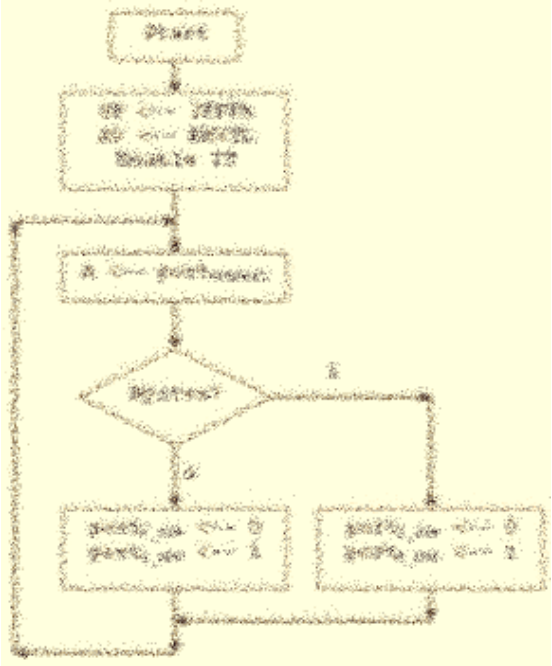

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Felhasznált fordítói direktívák

- EQU: szimbólumhoz érték rendelése
- ORG (origin): program memóriabeli kezdőcímének megadása
- $: az utasítás elhelyezési számláló aktuális értéke
- DB (define byte): adatok elhelyezése a gépi kódba az elhelyezés számláló értéke szerinti helyre
- END: a fordítás befejezése

- További elemek:
  - Címkék: sor elején kezdődnek, kettősponttal zárulnak
  - Megjegyzések: pontosvesszőtől (”;”) az adott sor végéig


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



# FOLYAMATÁBRÁK RAJZOLÁSA

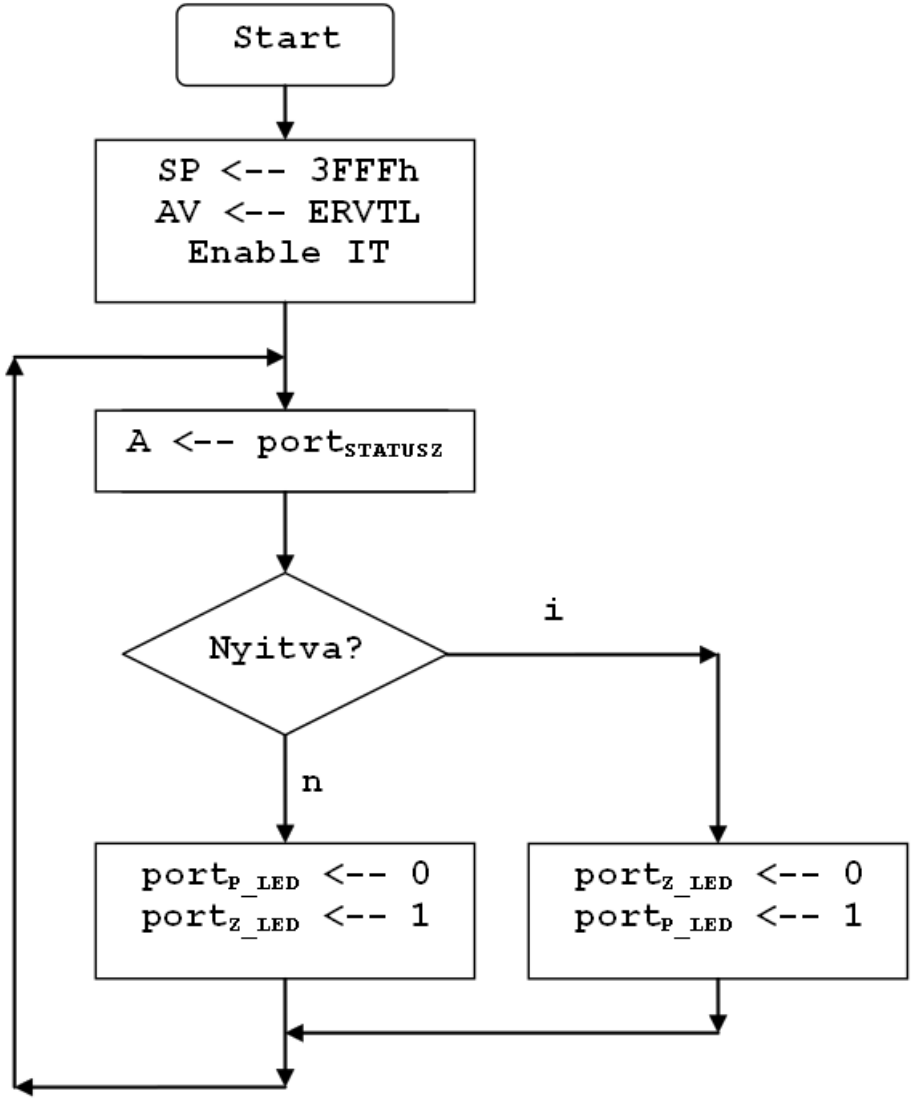

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Miért és hogyan használjuk?

- Rövid, egyszerű assembly nyelvű programok (néhányszor tíz sor) megírása előtt általában célszerű (és elegendő is) folyamatábrát készíteni.
- A folyamatábrába olyan lépéseket szoktunk írni, amit vagy közvetlenül egy gépi utasítással vagy legfeljebb 2-3 utasítással el tudunk végezni. Ezeket kifejezhetjük szavakkal, de még jobb, ha formálisan írjuk le őket (mert így egyértelmű); például az adatmozgatásokat regiszterekkel, memória/port címekkel és nyilakkal jelezzük:
  - A <-- (HL): Az A regiszterbe (akkumulátor) betöltjük a HL regiszterpár (értéke) által megcímzett memóriarekesz tartalmát.
  - $port_{5Bh}$ <-- A: Az akkumulátor értékét kiírjuk az 5Bh számú portra.
- A folyamatábra így természetesen processzorfüggő.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

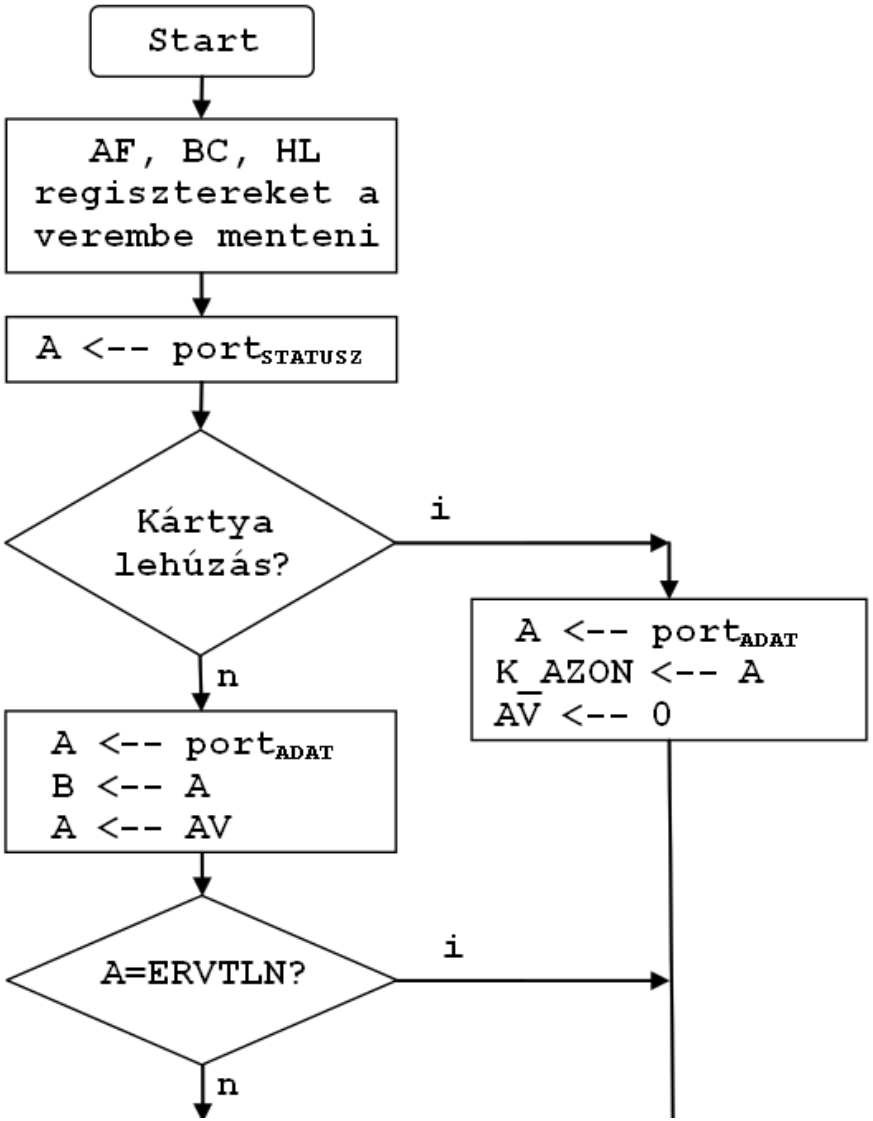
# A főprogram folyamatábrája





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

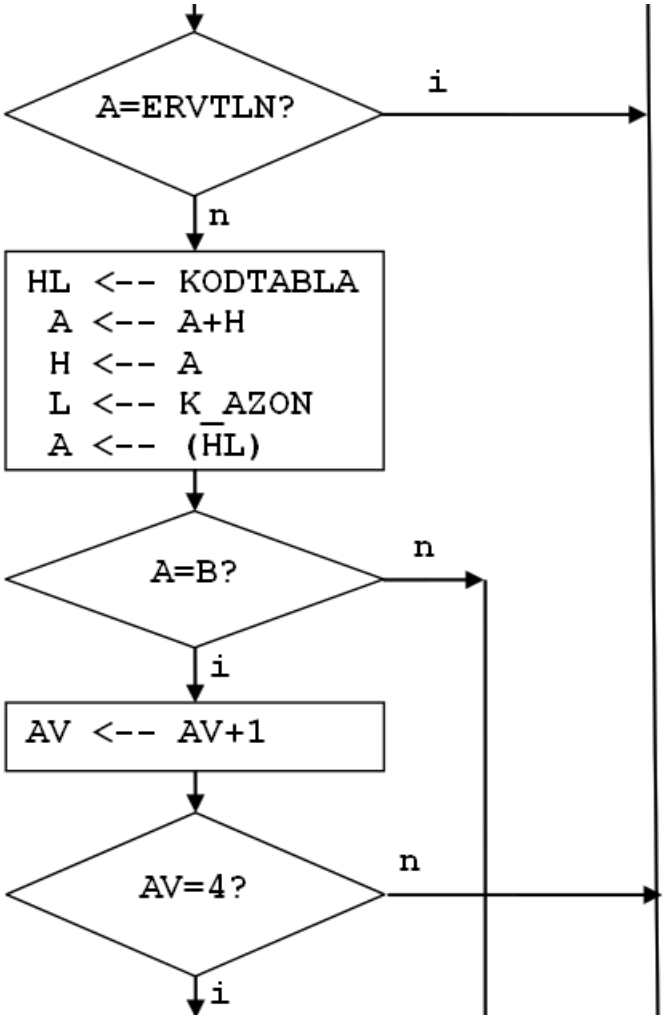
# Az IT rutin folyamatábrája – 1



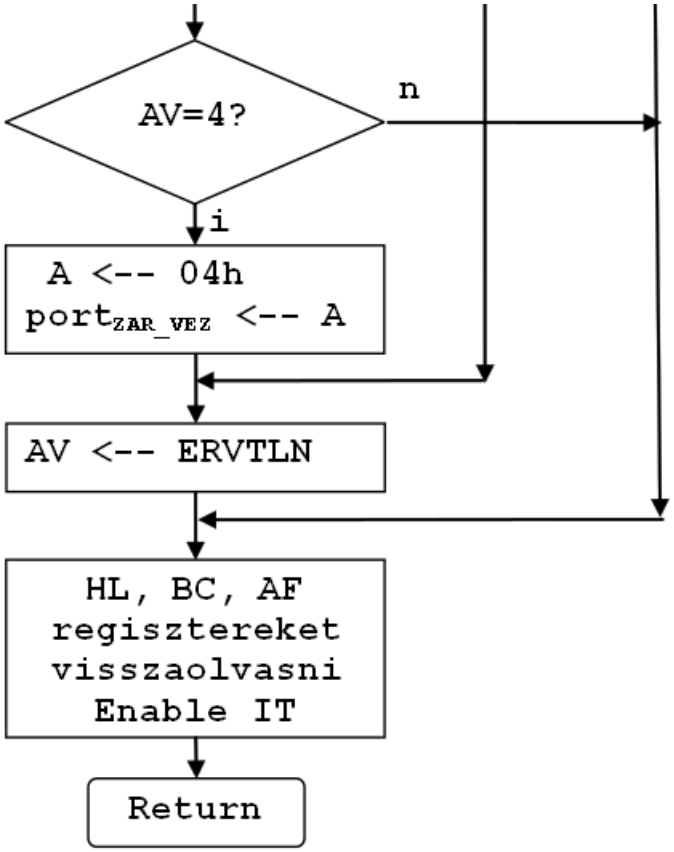


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

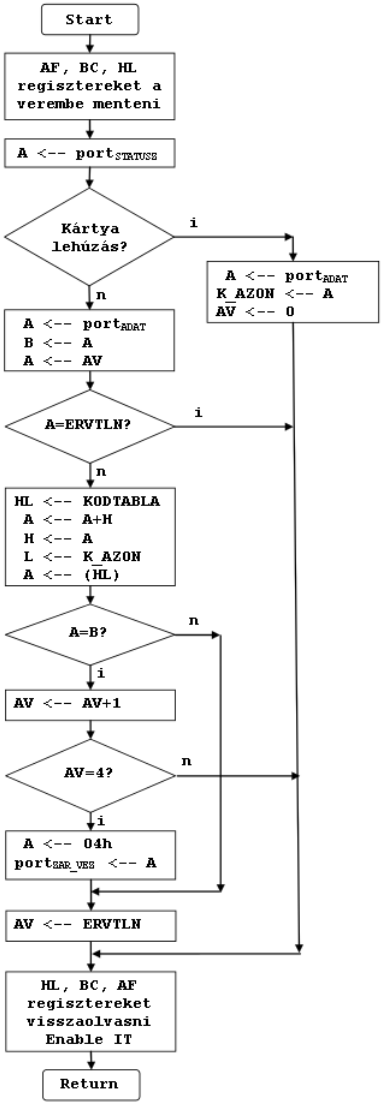
# Az IT rutin folyamatábrája – 2



# Az IT rutin folyamatábrája – 3



# Az IT rutin folyamatábrája egyben



# ASSEMBLY NYELVŰ PROGRAM ÍRÁSA


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Assembly nyelvű program – 1

```
; hardver címek
Z_LED    EQU     1Ah             ; zöld LED portcíme
P_LED    EQU     1Bh             ; piros LED portcíme
STATUSZ  EQU     2Eh             ; periféria státusz reg. portcíme
ZAR_VEZ  EQU     2Eh             ; periféria zárvez. reg. portcíme
ADAT     EQU     2Fh             ; periféria adat reg. portcíme
;
; egyéb konstansok
HOSSZ    EQU     04              ; belépőkártyákhoz kódjának hossza
KLE      EQU     00000001b       ; maszk kártyalehúzás teszteléséhez
SZJ      EQU     00000010b       ; maszk számjegy teszteléséhez
ZARALL   EQU     00000100b       ; maszk a zár állapot kiolv.-hoz
NYISD    EQU     00000100b       ; a nyitás parancs kódja
ERVTLN   EQU     255             ; az érvénytelen állapot kódja
LAP_MER  EQU     100h            ; 256 byte (trükkhöz)
;
```


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Assembly nyelvű program – 2

```
; változók címei
AV      EQU     2000h           ; állapotváltozót itt tároljuk
K_AZON  EQU     2001h           ; kártyaazonosítót itt tároljuk
; A kódtáblát majd a főprogram után az EPROM-ban helyezzük el...
;
; itt kezdődik a főprogram
        ORG     0000h           ; programkód elhelyezése a 0 címtől
        LD      SP,3FFFh        ; veremmutató beáll. a RAM tetejére
        LD      A,ERVTLN        ; az érvénytelen állapot kódja
        LD      (AV),A          ; állapotváltozó beállítása
        EI                      ; megszakítások engedélyezése
```


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Assembly nyelvű program – 3

```
; most jön a program főciklusa
FOCIKL: IN      STATUSZ         ; periféria állapotának kiolvasása
        AND     ZARALL          ; zárállapot vizsgálata
        JNZ     NYITVA          ; ha a D2 bit 1-es volt: nyitva van
; ha nem ugrott el, akkor tudjuk, hogy zárva van!
        LD      A,0             ; kigyújtás
        OUT     P_LED           ; piros LED világítson
        LD      A,1             ; kioltás
        OUT     Z_LED           ; zöld LED ne világítson
        JMP     FOCIKL          ; végtelen ciklus
NYITVA: LD      A,0             ; kigyújtás
        OUT     Z_LED           ; zöld LED világítson
        LD      A,1             ; kioltás
        OUT     P_LED           ; piros LED ne világítson
        JMP     FOCIKL          ; végtelen ciklus
;
```

# Assembly nyelvű program – 4

```
; A kódtáblát laphatárra helyezzük az alábbi apró trükkel:
; (a fenti programkód rövidsége miatt az alábbi sor nyilván
;  "ORG 0100h"-val ekvivalens)
        ORG     ($+LAP_MER-1) AND NOT (LAP_MER-1)
KODTABLA: DB    1, 4, 5, 2, 3, 4, 7, 8, 8, 0, 2, 3, 6, 2, 4, 9
        DB      4, 4, 3, 1, 7, 8, 0, 1, 3, 6, 4, 1, 4, 7, 6, 5
        DB      ...
; összesen 64 sor, soronként 16 értékkel,
; azaz 64x16=1024=4x256 db számjegy
;
```

# Assembly nyelvű program – 5

```
; most jön a megszakítási rutin
        ORG     1000h           ; az IT rutin elhelyezése
; minden olyan regiszter értékét elmentjük, amit használni fogunk
        PUSH    AF              ; az A regiszter és a flag-ek,
        PUSH    BC              ; a BC regiszterpár,
        PUSH    HL              ; a HL regiszterpár mentése
        IN      STATUSZ         ; mi volt a megszakítás oka?
        AND     KLE             ; kártyalehúzás jött?
        JNZ     LEHUZ           ; igen
; biztos, hogy számjegy volt, nem mert kártyalehúzás!
        IN      ADAT            ; számjegy kiolvasása (IT törlése)
        LD      B,A             ; a számjegy mentése
        LD      A,(AV)          ; állapotváltozó betöltése
        CMP     ERVTLN          ;
        JZ      IT_VEGE         ; ha érvénytelen, akkor kész
```

# Assembly nyelvű program – 6

```
; az AV értéke szükségképpen a [0,HOSSZ-1] intervallumba esik!
; a várt számjegy a KODTABLA+LAP_MER*AV+K_AZON címen van:
        LD      HL,KODTABLA     ; kódtábla kezdőcíme, (L=0!)
        ADD     H               ; A:=A+H
        LD      H,A             ; A KODTABLA-ban az (AV). lapon ...
        LD      L,(K_AZON)      ; ... keressük a (K_AZON). értéket
        LD      A,(HL)          ; a várt számjegy kiolvasása
        CMP     B               ; B-ben van a beérkezett számjegy
        JNZ     ROSSZJ          ; hibás a beütött számjegy
; a beütött számjegy helyes volt, ezért AV 1-gyel nő:
        LD      A,(AV)
        INC     A
        LD      (AV),A
        CMP     HOSSZ           ; elértük már a kódsorozat hosszát?
        JNZ     IT_VEGE         ; még nem
```


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Assembly nyelvű program – 7

```
; ha mar elértük: nyitunk + AV:=ERVTLN
NYITAS: LD      A,NYISD         ; nyitás parancs kódja
        OUT     ZARVEZ          ; parancs kiadása
ROSSZJ: LD      A,ERVTLN        ; az új állapot: érvénytelen
        LD      (AV),A          ; állapot eltárolása
        JMP     IT_VEGE         ; ugrás az IT rutin végére
; kártyalehúzás feldolgozása
LEHUZ:  IN      ADAT            ; belépőkártya kódjának kiolvasása
        LD      (K_AZON),A      ; kártyakód eltárolása
        LD      A,0             ; ez lesz az új állapot
        LD      (AV),A          ; állapotváltozó beállítása
; a regisztereket fordított sorrendben kell visszatölteni!
IT_VEGE: POP    HL
        POP     BC
        POP     AF
        EI
; Az IT engedélyezésének hatása egy utasítást késik!
        RET                     ; visszatérés az IT rutinból
        END                     ; vége a fordításnak
```


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# ALTERNATÍV MEGOLDÁSOK VIZSGÁLATA


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Önálló hallgatói munka – 1

- A hallgatók önállóan gondolják végig, hogy miben különbözne a program, ha nem használnánk megszakítást, hanem helyette állapotolvasással kérdeznénk le, hogy történt-e kártyalehúzás, illetve érkezett-e számjegy!
- Módosítsák ennek megfelelően a programot önállóan!

FIGYELEM! Először mindenki készítse el a saját megoldását és csak utána nézze meg a tárgy jegyzetében szereplő minta megoldást!


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Önálló hallgatói munka – 2

- Gondolkozzanak el önállóan az alábbi kérdéseken!
  1. A megszakításos és az állapotolvasásos megoldás közül melyiket biztosan nem lehet megvalósítani RAM nélkül? Miért?
  2. Hogyan tudná a másikat megvalósítani RAM nélkül? (Mit, hova helyezne el?)
  3. Hogyan módosítaná a programot, ha a KODTABLA nem illeszkedne laphatárra? (A kritikus számításhoz használjon 8 bites műveleteket, a mnemonikokat értelemszerűen állapítsa meg!)
  4. Hogyan módosulna a fenti számítás akkor, ha a KODTABLA (mint kétdimenziós tömb) indexeit felcserélnénk, azaz a 4db 256 bájt méretű tömb helyett 256 db 4 bájtos tömb lenne
  5. Hogyan valósíthatnánk meg a megalkotott állapotgráf szisztematikus lekódolását, és ez mekkora munkát jelentene!

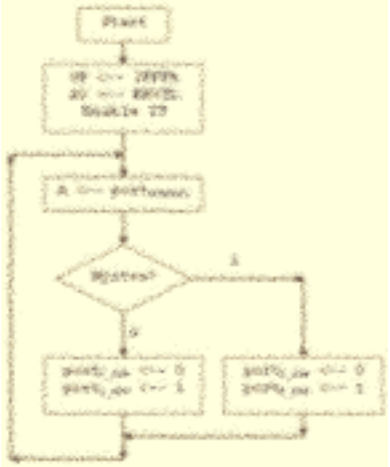
# Önálló hallgatói munka – 3

6. Hogyan tudná elkerülni a főprogramban az elágazás használatát, ha felhasználhatja még a XOR (bitenkénti kizáró vagy) utasítást, valamint:
   a) az SHR (bitenkénti eltolás jobbra) utasítást?
   b) módosíthatja a beléptető periféria státusz regiszterében a bitek specifikációját?
   c) módosíthatja a hardver specifikációját?


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Összefoglalás

- Megterveztük a beléptető szoftverét (tervezési döntések)
- Megismertük hipotetikus processzor utasításkészletét
- Folyamatábrákat rajzoltunk
- Megírtuk az assembly nyelvű programot
- Alternatív megoldásokat is megvizsgáltunk

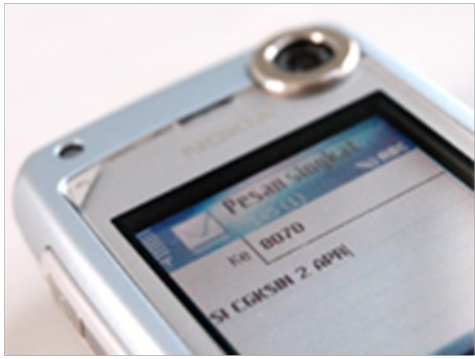


# Kérdések?

## KÖSZÖNÖM A FIGYELMET!

Híradástechnikai Tanszék

Dr. Lencse Gábor
tudományos főmunkatárs
BME Híradástechnikai Tanszék
lencse@hit.bme.hu



# Számítógép Architektúrák

*Az ajtónyitó megvalósítása Arduino-val*

Horváth Gábor
docens
BME Híradástechnikai Tanszék
ghorvath@hit.bme.hu

2012. szeptember 10.
Budapest

# Tartalom

- A gyakorlat célja:
  - Megmutatni, hogy a manapság elterjedt és olcsó mikrokontrollerekkel milyen könnyű dolgozni
- Tartalomjegyzék:
  - Az Arduino bemutatása
    - Hardver
    - Programozás
    - Szenzorok
  - Az ajtónyitó megvalósítása
    - A kijelző
    - Az RFID olvasó
    - A billentyűzet
    - A teljes hardver
    - A teljes kód

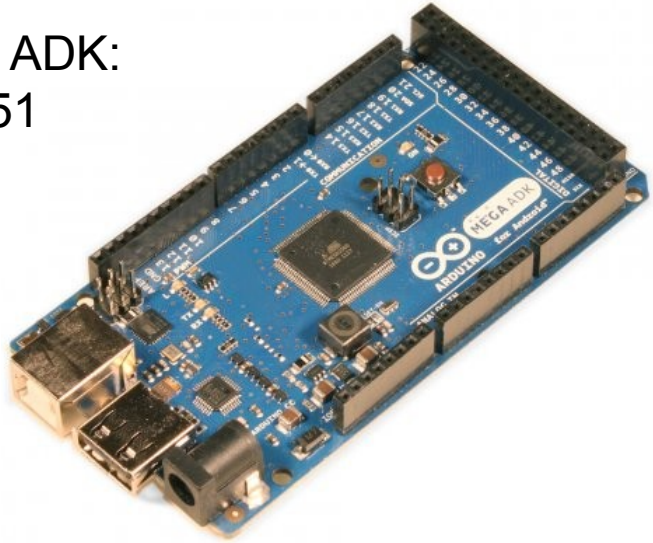
# Arduino

- 2005, Ivrea, Olaszország
- Célja: egyszerű prototípusfejlesztés
- Valójában egy teljes család (http://arduino.cc)
- Ami közös bennük:
  - Atmel AVR 8 bites processzor (**Harvard** architektúra!)
  - Beépített flash memória a programnak
  - Beépített EEPROM tartós adattárolásra
  - Beépített RAM
  - Ki/bemenetek:
    - Digitális
    - Analóg
  - USB port-on át programozható (nem mindegyik)
  - C++-szerű nyelven programozható

# Az Arduino család

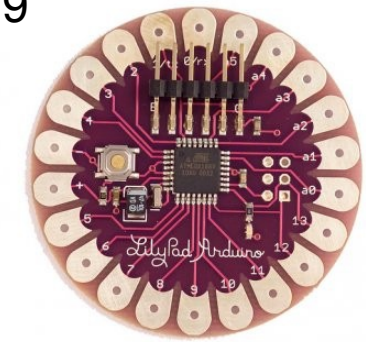
Leonardo:
€18

Mega ADK:
€51




Mini:
€15

LilyPad:
€19



# Az Arduino család

- Eltérések:
  - Bemenetek/kimenetek száma
  - Memória mérete (flash/EEPROM/RAM)
  - Néhány jellegzetesség:
    - A Mega ADK-t hozzá lehet kötni az Android eszközökhöz
    - A LilyPad ruhára varrható

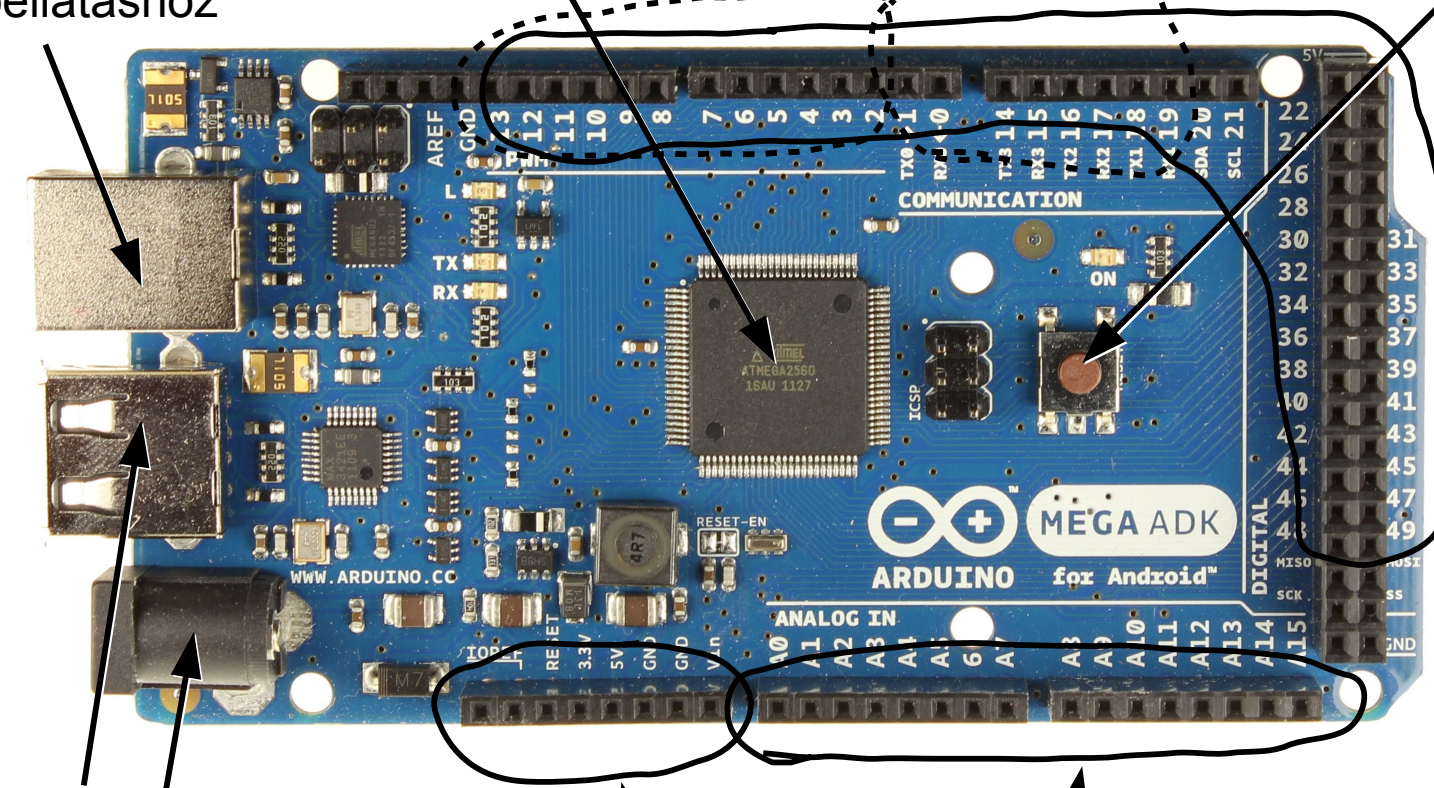
| | CPU órajel | Flash | RAM | EEPROM | Digitális I/O | Analóg I/O |
|---|---|---|---|---|---|---|
| Leonardo | 16 MHz | 32 kB | 2.5 kB | 1 kB | 20 | 12/7 |
| Mega ADK | 16 MHz | 256 kB | 8 kB | 4 kB | 54 | 16/15 |
| Mini | 16 MHz | 32 kB | 2 kB | 1 kB | 14 | 8/6 |
| LilyPad | 8 MHz | 16 kB | 1 kB | 512 byte | 14 | 6/6 |

# Átverés!

- Micsoda??
  - €51, 8 bites CPU-ért, 16 MHz-ért, pár kB memóriáért?
  - Amikor ott a Raspberry Pi 35$-ért?

    (700 MHz 32 bites ARM, 256 kB RAM, erős GPU, HD filmlejátszás, HDMI kimenet, Ethernet port, stb.)
- Más a célja:
  - Arduino:
    - Hangsúly: I/O, minden mennyiségben, pofonegyszerűen
    - Bekapcsolom, és 1 másodperc múlva már megy is
    - Az €51 a legnagyobb modell, az olcsó Leonardo a legtöbb dologra elég
  - Raspberry Pi:
    - Hangsúly: általános célú (programozás oktatás)
    - Op. rendszer (linux) kell rá! Sokáig tart, mire elindul.
    - Van programozható periféria ki/bemenete, de nehéz használni (érteni kell a linux kernelhez)

# Arduino Mega ADK



# *Programozás*

# Programozás

- Open-source cross-platform fejlesztői környezet
  (http://arduino.cc/en/Main/Software)
- Programnyelv: C++-hoz hasonló (kiterjesztés: .ino)
- Fordítás:
  - A fejlesztőkörnyezet AVR kódot fordít (cross compiler)
  - USB-n keresztül feltölti a mikrokontroller flash memóriájába
- Hibakeresés:
  - Amit az Arduino a default soros portjára ír, azt USB-n keresztül megmutatja a fejlesztőkörnyezet

# A fejlesztőkörnyezet

Szintaktikai ellenőrzés

Feltöltés az Arduino-ra

Új fájl

Fájl megnyitás

Fájl mentés

Soros port monitor

ajtonyito | Arduino 1.0.1

File Edit Sketch Tools Help

ajtonyito

```
#include <LiquidCrystal.h>
#include <Key.h>
#include <Keypad.h>

int RFIDResetPin = 2;
int LEDPin = 13;

char userID[17];
int index = 0;
enum estate { start, rfid, code};
estate state = start;

LiquidCrystal lcd(49, 47, 48, 46, 44, 42);

const byte ROWS = 4;
const byte COLS = 3;

char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
```

Done uploading.

```
Binary sketch size: 7,314 bytes (of a 258,048 byte maximum)
```

4 Arduino Mega 2560 or Mega ADK on /dev/ttyACM0

# A programozási nyelv

- Az Arduino a programot „sketch”-nek hívja
- Adattípusok:
  - int: 16 bites egész
  - long: 32 bites egész
  - boolean: logikai típus (true/false), 1 bitet foglal (ilyen C++-ban nincs)
  - float: 32 bites lebegőpontos
  - char: ASCII karakter tárolására szolgál (1 byte)
  - stb.
- Operátorok, ciklusok, elágazások: mint a C++
- class-ok használata megengedett, preprocessor direktívákat elfogad
- Két függvény van, amit kötelező megírni:
  - **void setup () { … }** - az ide írt kód induláskor fut le, egyetlenegyszer
  - **void loop () { … }** - inicializálás után ez a függvény fut le újra és újra (ha befejeződik, újraindul a függvény elejétől)

# *Ki/Bemenetek*

# Digitális ki- és bemenetek

- A digitális lábak ki és bemenetként is szolgálhatnak egyaránt
  - De egyszerre csak az egyik szerepet tölthetik be
  - Beállítás:
    - **pinMode (4, INPUT);** – a 4-es lábat bemenetre állítja
    - **pinMode (5, OUTPUT);** – az 5-ös lábat kimenetre állítja
- A digitális lábak magasba, ill. alacsonyba húzása:
  - **digitalWrite (5, HIGH);** – az 5-ös lábra logikai 1-et tesz (5V)
  - **digitalWrite (5, LOW);** – az 5-ös lábra logikai 0-át tesz (0V)
- A digitális lábak olvasása:
  - **int val;**
  - **val=digitalRead (4);** – a 4-es láb logikai értékének olvasása
  - **if (val==HIGH) …**, vagy **if (val==LOW) …**

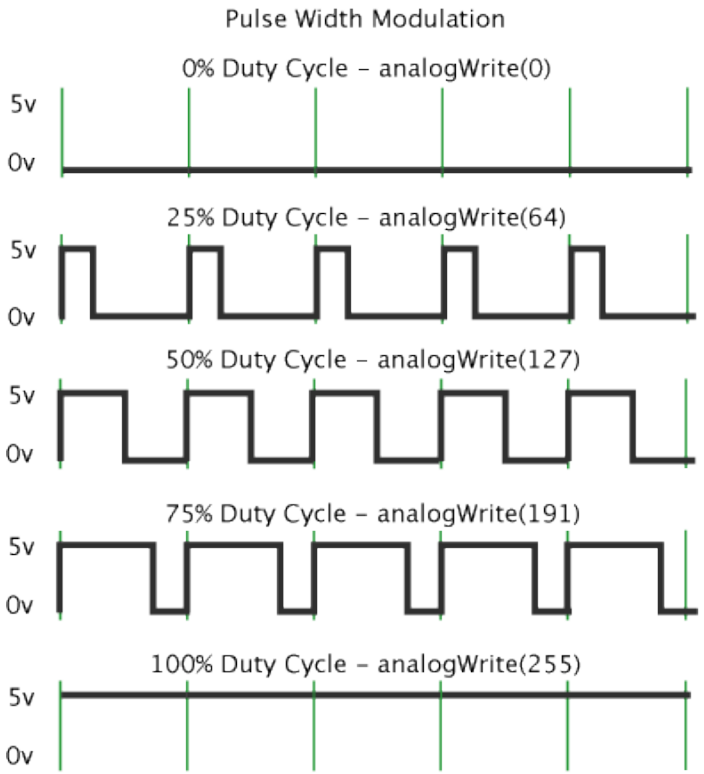
# Analóg bemenetek

- Nemcsak HIGH és LOW értékeket lehet róla olvasni
- ADC (Analog-Digital Converter) 10 bites felbontással
  → 1024 különböző feszültségszintet tud megkülönböztetni
- 0V és 5V között 1024 szint → 4.88 mV különbséget tud tenni
- Használata:
  - **int val;**
  - **val = analogRead (3);** - a 3-mas analóg bemenetről mintát vesz
  - **val** értéke: 0-tól (0V esetén) 1023-ig (5V)
- Az 5V maximum referencia változtatható az **analogReference()** függvénnyel, de 5V a maximum

# PWM kimenetek

- Folyamatos analóg kimenet (pl. 4.2V) nem tud előállítani
- Viszont a kimenet gyors ki-be kapcsolgatásával „átlagos értelemben” tetszőleges feszültséget elő tud állítani

  → **PWM: Impulzusszélesség-moduláció**
- Erre szolgál az **analogWrite ()** függvény
- Paraméterek: láb száma, kitöltöttség (Duty Cycle, 0…255 között)



# Soros ki- és bemenetek

- A kommunikációhoz max. 2 ér kell:
  - RX: a vételhez (ha szükség van vételre)
  - TX: az adáshoz (ha szükség van adásra)
- Az Arduino ADK 4 soros portot tartalmaz
  - Az elsőt használja Debug célra a fejlesztőkörnyezet USB-n keresztül
- Használata: **class Serial**-on keresztül
  - Előregyártott példányok: **Serial**, **Serial1**, **Serial2**, **Serial3**
  - *Inicializálás*: **Serial1.begin (9600);** - port megnyitása 9600 bps sebességre
  - *Írás*: **Serial1.write (…);** - elküld 1 bájtot, egy NULL terminált string-et, vagy egy tömböt
  - *Írás*: **Serial1.print (...);** - a paraméterét string-é kovertálja (ha nem az), és elküldi. A **Serial1.println (...);** még egy soremelést is utánatesz.
  - *Olvasás*: **int kaptam=Serial1.read();** - egy megérkezett bájt kiolvasása (-1, ha nincs)
  - *Érkezett-e adat*: **int mennyi = Serial1.available();** - megadja, hány bájt érkezett
  - *Lezárás*: **Serial1.end ();** - a port lezárása után a megfelelő láb digitális ki- és bemenetként használható tovább

# Soros ki- és bemenetek

- Működése:
  - Vételkor Interrupt keletkezik
  - Az Interrupt-ot lekezeli az Arduino:
    - Beleteszi az érkezett bájtot egy tárolóba (buffer)
  - A read() tulajdonképpen a tárolóból olvas
  - Az available() a tárolóban lévő bájtok számát adja vissza

# *Memóriák*

# Memóriák

- Az Arduino az alábbi memóriákkal rendelkezik:
  - **Flash memória**: ez tárolja a programot
    - Tápfeszültség nélkül is megmarad a tartalma
  - **RAM**: ebben vannak a változók és a stack is
    - Tápfeszültség kell a tartalom megőrzéséhez
  - **EEPROM**: tartósnak szánt adatokhoz
    - Tápfeszültség nélkül is megmarad a tartalma
- Az AVR processzorok Harvard architektúrát követnek
  - 2 címtér:
    - Az utasításokat és konstansokat a flash memóriából veszi
    - A változókhoz/stack-hez a RAM-ot használja
  - És az EEPROM?
    - Ahhoz külön függvénykönyvtár van

# A flash és a RAM

- Flash memória:
  - A fejlesztőkörnyezet tölti meg
  - A programból nem írható. De olvasható! Speciális függvényekkel.
- RAM:
  - Kevés van belőle, ezért spórolni kell
  - Nem változó tömböket inkább a flash-be írjuk:

```
int kodok[] = {21342, 2442, -34887, ...};
char szoveg[] = „megszentségteleníthetetlenségeiteknek”;
k = kodok[4];
c = szoveg[7];
```

    - Helyett:

```
PROGMEM prog_uint16_t kodok[] = {21342, 2442, -34887, ...};
PROGMEM prog_uchar szoveg[]  = „megszentségteleníthetetl...”;
k = pgm_read_word_near (kodok + 4);
c = pgm_read_byte_near (szoveg + 7);
```

# EEPROM

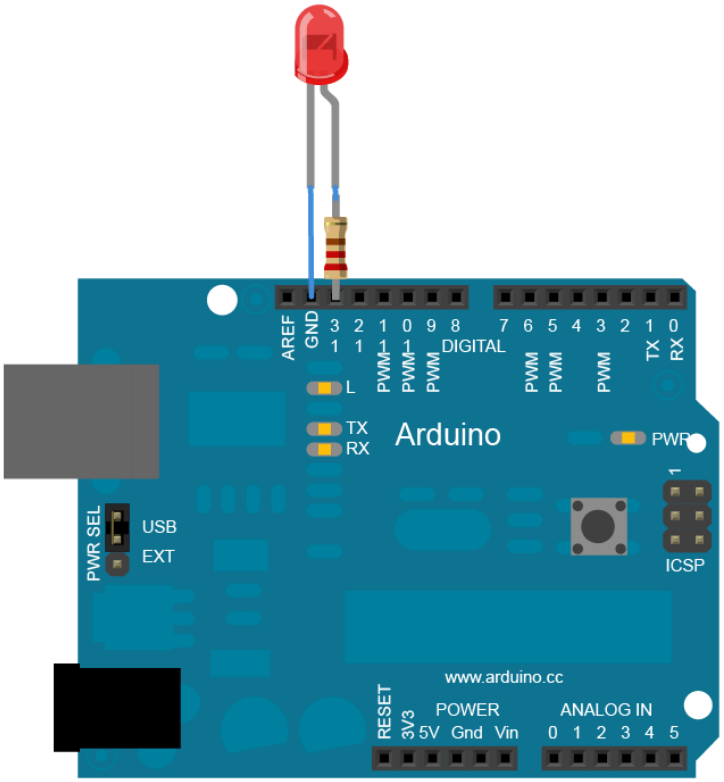
- A rendszer része az EEPROM könyvtár
- #include <EEPROM.h>
- Írása:
  - **EEPROM.write (cím, adat);**
    - A cím int típusú
    - Az adat byte típusú
- Olvasása:
  - **byte a;**
    **a = EEPROM.read (42);**
    - Kiolvassa a 42-edik byte-ot az EEPROM-ból

# *Perifériák illesztése*

# Digitális kimeneti perifériák

- **Példa**:
  LED villogtatás (Nem Arduino ADK, de ugyanaz az elv)
  Alkatrész: LED, 220 Ohmos ellenállás

  LED nélkül is megy!
  A 13-mas lábon van beépített LED.

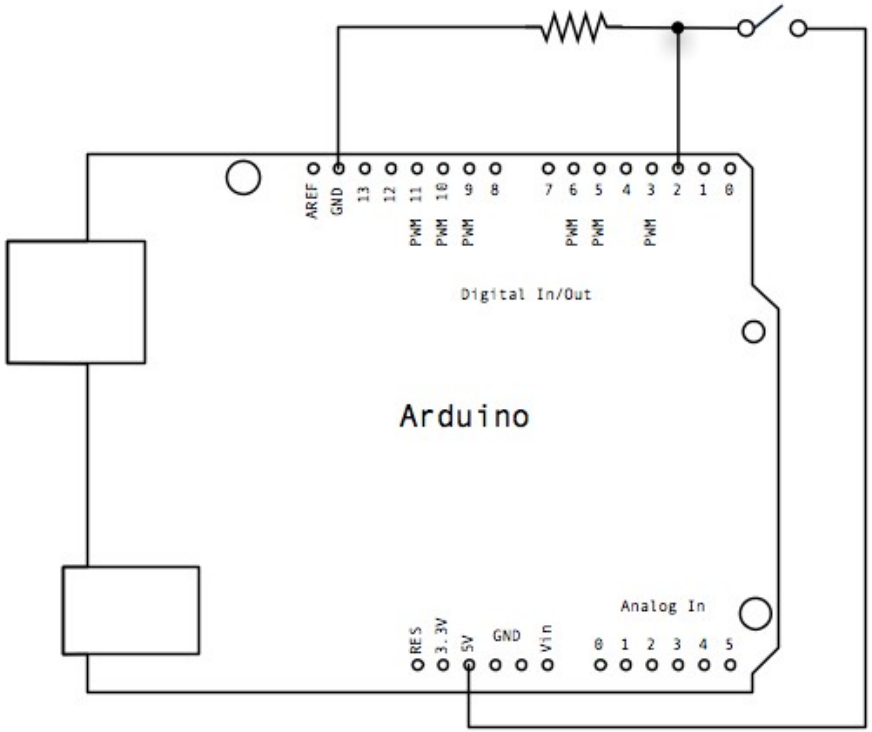



- **Kód**:

```
int ledPin = 13;
void setup () {
    pinMode (ledPin, OUTPUT);
}
void loop () {
    digitalWrite (ledPin, HIGH);
    delay (1000);
    digitalWrite (ledPin, LOW);
    delay (1000);
}
```

# Digitális bemeneti perifériák

- **Példa**:
  Nyomógomb állapotának lekérdezése, a beépített LED kigyújtása

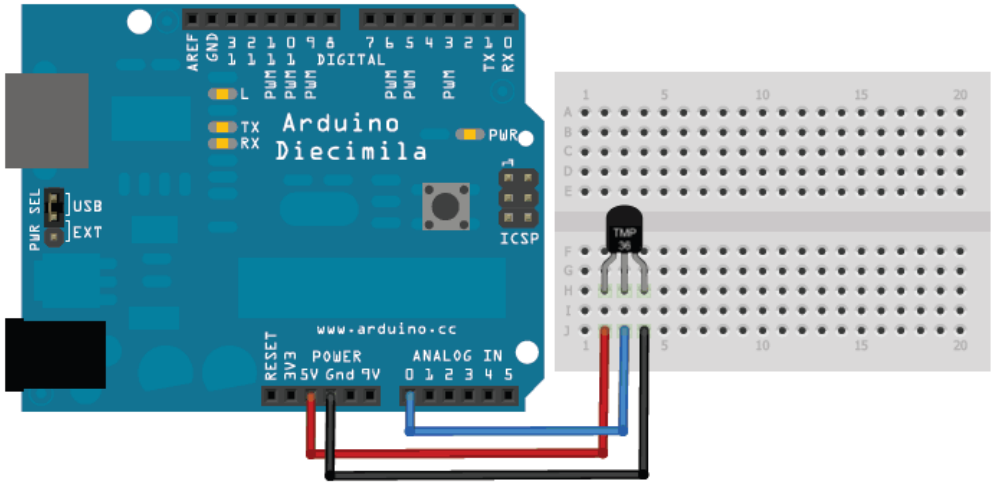



- **Kód**:

```
const int buttonPin = 2;
const int ledPin =  13;
int buttonState = 0;
void setup () {
    pinMode (ledPin, OUTPUT);
    pinMode (buttonPin, INPUT);
}
void loop () {
    buttonState = digitalRead (buttonPin);
    digitalWrite (ledPin, buttonState);
}
```

# Analóg bemeneti perifériák

- **Példa**:
  TMP36 Hőmérséklet szenzor illesztése




- **Kód**:

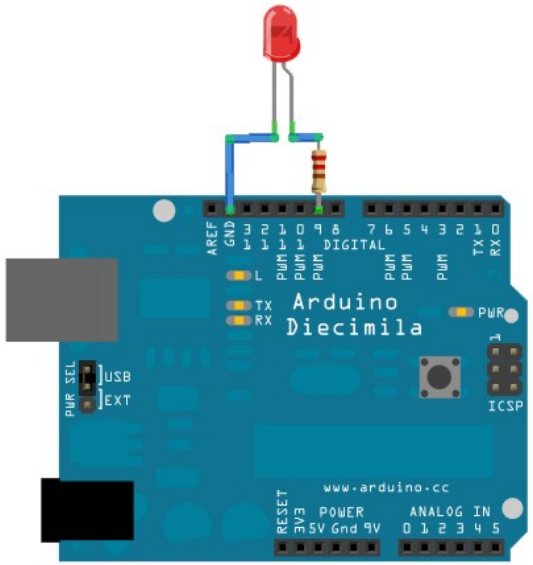
```
int sensorPin = 0;
void setup () {
    Serial.begin (9600);
}
void loop () {
    int reading = analogRead (sensorPin);
    float voltage = reading * 5.0 / 1024.0;
    float temperature = (voltage - 0.5) * 100;
    Serial.println (temperature);
    delay(1000);
}
```

# Analóg bemeneti perifériák

- A legérdekesebb perifériák az analóg bemeneti perifériák:
  - Filléres szenzorok tömkelege:
    - 3 tengelyes gyorsulásmérő (3 analóg bemenetet használ)
    - Alkohol szenzor
    - Szénmonoxid szenzor
    - Szálló por koncentráció szenzor
    - Elhajlásmérő szenzor
    - Erő (súly) mérő szenzor
    - Vibráció szenzor
    - Giroszkóp (2 tengelyes → 2 analóg bemenet kell)
    - Távolságmérő (infra fénnyel vagy ultrahanggal)
    - Hőmérő
    - Páratartalommérő
    - Stb.
- A mért fizikai mennyiséget analóg jellé alakítják

# PWM kimeneti perifériák

- **Példa**:
  LED halványítás/fényesítés




- **Kód**:

```
int ledPin = 9;
void setup ()  {
}
void loop ()  {
    for (int fadeValue = 0 ; fadeValue <= 255; fadeValue +=5) {
        analogWrite (ledPin, fadeValue);
        delay (30);
    }
    for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -=5) {
        analogWrite (ledPin, fadeValue);
        delay (30);
    }
}
```

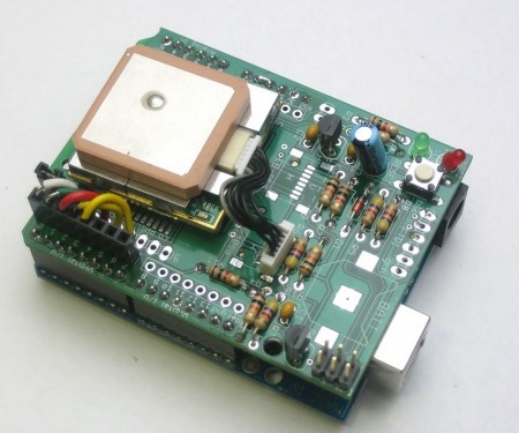
# Soros perifériák

- Példák soros portra csatlakozó perifériákra:
  - RFID modul: soros porton küldi a leolvasott kártya kódját
  - GPS modul: soros porton küldi rendszeresen a koordinátákat
  - GSM/GPRS modul: soros porton kell vezérelni, illetve az átviendő adatok is a soros porton mennek
  - Stb.

# Pajzsok (Shield)

- Az Arduino tetejére illeszthető komplett perifériák
  - Nagyon sok van:
    - GPS, LCD vezérlő, SD kártya kezelő, WIFI, Bluetooth, ZigBee, GSM, ...

GPS Shield
€18



MP3 Shield
€25

2.8” TFT és érintőképernyő
€40

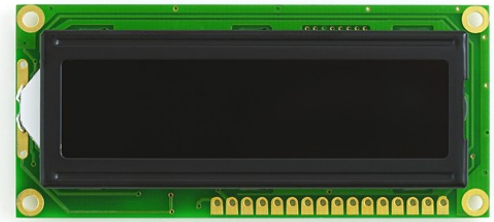
## *Az ajtónyitó megvalósítása*

# Ajtónyitó

- Milyen alkatrészek is kellenek?
  - Kellene egy kis kijelző, amire üzeneteket írhatunk a felhasználónak
  - Kellene egy numerikus billentyűzet
  - Kellene egy kártyaleolvasó – RFID jó lesz
  - Egy relé, ami az ajtót nyitja
  - Egy megfelelően választott Arduino a népes családból

# A kijelző

- Olcsón vehetünk egy 2 soros, 16 oszlopos kijelzőt (2000 Ft)



- Lábak:
  - *Adatbusz*: D0...D7, de 4 bites üzemmódban is megy, nekünk az is elég: D4...D7-et kötjük be
  - *RS*: ezzel lehet jelezni a kijelzőnek, hogy épp egy ASCII karaktert küldünk neki, vagy vezérlő parancsot
  - *EN*: engedélyezés, a kijelző ekkor olvassa el a neki küldött adatot
  - *RW*: ha nem adatot küldünk a kijelzőnek, hanem tőle kérdezünk le információt – ezt nem kötjük be
  - *Tápfeszültség* (5V)
  - *Kijelző kontraszt* (potméter)
  - *Háttérvilágítás tápellátása*
- 6 adatvezetéket használunk: RS, EN, D7, D6, D5, D4

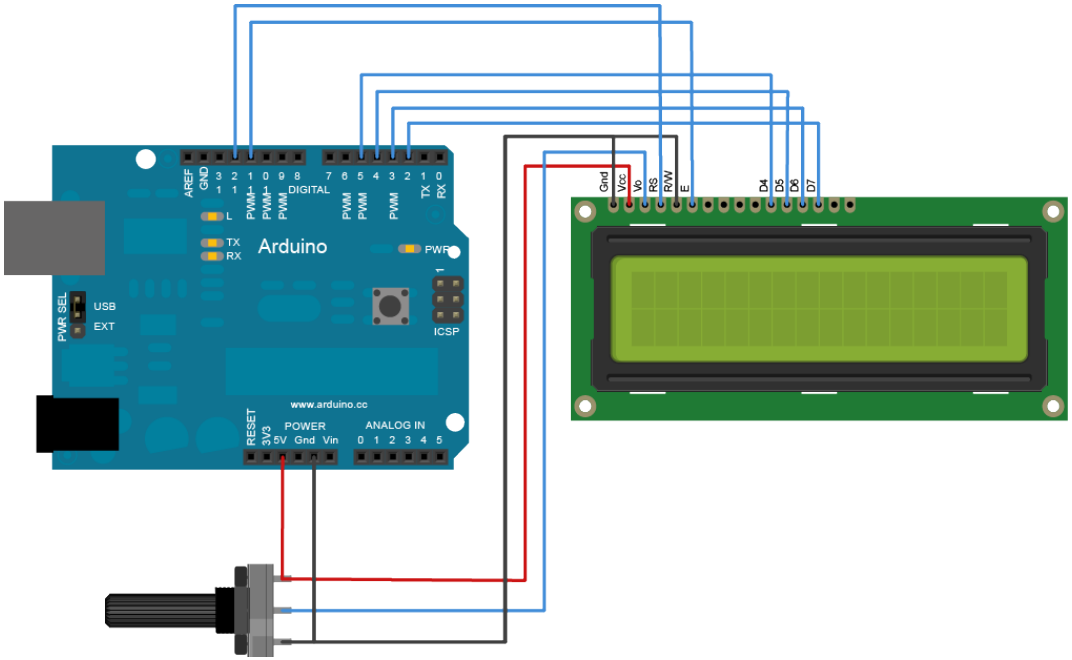
# A kijelző használata

- **Bekötés**:
  Google „arduino display”

- **Használata**:
  **LiquidCrystal** class példányosítása
  (write(), setCursor(), blink(), clear(), stb.)

- **Példa**:

```
#include <LiquidCrystal.h>
const int numRows = 2;
const int numCols = 16;
// beállítjuk, melyik lábakra kötöttük
LiquidCrystal lcd (12, 11, 5, 4, 3, 2);
void setup () {
   lcd.begin (numCols,numRows);
}
void loop () {
   for (int thisLetter = 'a'; thisLetter <= 'z'; thisLetter++) {
      for (int thisRow = 0; thisRow < numRows; thisRow++) {
         for (int thisCol = 0; thisCol < numCols; thisCol++) {
            lcd.setCursor (thisCol, thisRow);
            lcd.write (thisLetter);
            delay (200);
         }
      }
   }
}
```



# A numerikus billentyűzet

- Bárhol beszerezhető, 4x3-mas (900 Ft)

- Lábak:
  - 7 láb, a 4 sorhoz és a 3 oszlophoz
  - Ha megnyomunk egy gombot, rövidre zárja a sor és oszlopvezetékét

# A numerikus billentyűzet használata

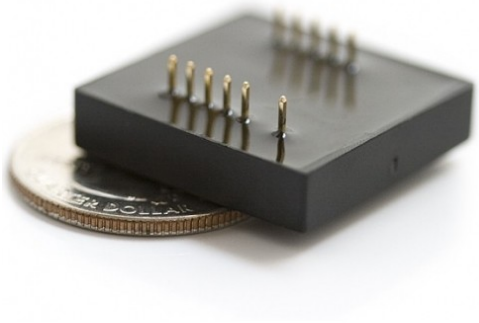
- Google „arduino keypad”
- **Használata**:
  **Keypad** class példányosítása
  (getKey(), waitForKey(), getState(), stb.)

- **Példa**:

```
#include <Keypad.h>
const byte ROWS = 4;
const byte COLS = 3;
char keys[ROWS][COLS] = {  {'1','2','3'},   {'4','5','6'},   {'7','8','9'},   {'*','0','#'}};
byte rowPins[ROWS] = {32, 22, 24, 28}; // sorok lábait hova kötöttük az Arduino-n
byte colPins[COLS] = { 30, 34, 26 };  // oszlopok lábait hova kötöttük az Arduino-n

Keypad keyPad = Keypad ( makeKeymap (keys), rowPins, colPins, ROWS, COLS );

#define ledpin 13
void setup () {
   digitalWrite (ledpin, HIGH);
}
void loop () {
   char key = keyPad.getKey();
   if(key)  {
      switch (key)  {
         case '*':
            digitalWrite(ledpin, LOW);
            break;
         case '#':
            digitalWrite(ledpin, HIGH);
            break;
      }
   }
}
```

# Az RFID olvasó

- Rádiofrekvenciás azonosítás
- Minden kártyának garantáltan egyedi, 12 hexa karakteres kódja van
- Leolvasó: drágát vettünk (7000 Ft), mert pont csak azt lehetett kapni
- Kártya: olcsó (250 Ft/db, kártya, korong, kulcstartó, stb.)



- Soros porton kommunikál
- Lábak:
  - Tápfeszültség (5V)
  - Külső antenna (nekünk a belső bőven elég)
  - Formátum kiválasztó (mi az ASCII módot választjuk)
  - 2 vonal adatátvitelre (csak az egyiket használjuk, soros portként)
  - LED/berregő a kártyaleolvasáskor (nem kötünk rá ilyet)
  - Reset
- Arduinohoz 2 vezetéket kötünk: Reset (digitális lábra), D0 (soros RX lábra)

# Az RFID olvasó használata

- Google „arduino id-12”
- **Használata**:
  soros eszközként

- **Példa**: (reset a 2-es, D0 az RX1 lábra kötve)

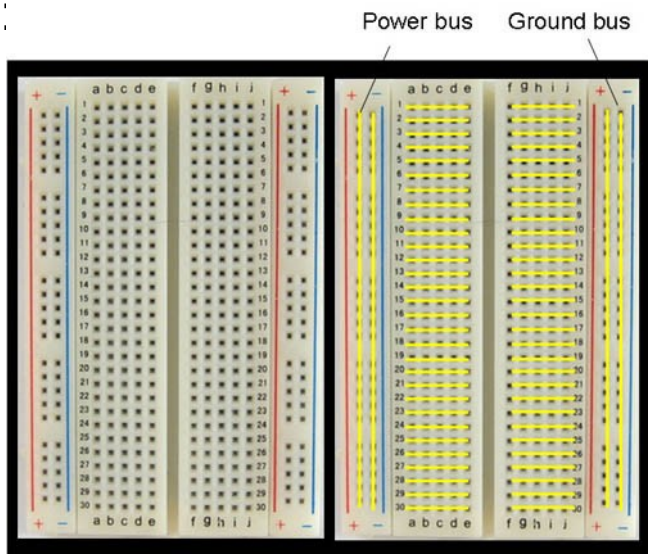
```
int RFIDResetPin = 2;                    // A 2-es digitális lábra kötöttük a leolvasó reset lábát
char userID[13];                         // Ebbe a tömbbe olvassuk majd be a kártya azonosítóját
void setup () {
   Serial.begin(9600);                   // 0-ás soros port beállítása 9600 bps-re (debug-hoz)
   Serial1.begin(9600);                  // 1-es soros port beállítása 9600 bps-re (RFID-t ide kötöttük)
   pinMode(RFIDResetPin, OUTPUT);        // RFID Reset láb beállítása „kimenet” módba
   digitalWrite(RFIDResetPin, LOW);      // RFID reset, felfutó élet csinálunk. Alacsonyba visszük...
   delay (100);                          // … kicsit várunk ...
   digitalWrite(RFIDResetPin, HIGH);     // … majd magasba emeljük.
}
void loop () {
   while (Serial1.available()) {         // Ha jött byte az RFID soros portjától, feldolgozzuk.
      int readByte = Serial1.read();     // Kiolvassuk a következő byte-ot
      if (readByte == 2)                 // Az ASCII 0x2 az „üzenet eleje” karakter
         index = 0;                      // Visszapörgetjünk az indexet, a userID-t az elejétől töltögetjük.
      else if (readByte == 3) {          // Az ASCII 0x3 az „üzenet vége” karakter
         userID[index] = 0;              // Lezárjuk, NULL terminált karakterlánccá tesszük.
         Serial.println(userID);         // Elküldjük a debug rendszernek. Ki fogja írni a PC-n.
      }
      else
         userID[index++] = readByte;     // Ha az üzenet közepén vagyunk, eltároljuk a karaktert.
   }
}
```

# A megfelelő Arduino kiválasztása

- Szempontok:
  - Hány digitális ki- és bemenet kell:
    - A kijelzőnek: 6
    - A billentyűzetnek: 7
    - Az RFID olvasónak: 2
    - Összesen: 15
    - A Leonardo 20-at kezel, de a beszerzéskor még nem létezett. Az elődje 14-et tudott, ezért az ADK-t választottuk. (54-et tud)
  - További ki- és bemenetek:
    - Analóg be, PWM ki nem kell
    - Soros port kell az RFID-nek
    - Ha debug-olni is akarunk, ahhoz kell még egy soros port (USB-n a PC serial monitor-jához)
    - A Leonardo pont 2 soros portot tud (az elődje csak 1-et tudott)
- Ideális választás: **Leonardo**

# A kapunyitó összeállítása

- Breadboard segítségével, forrasztás nélkül összehuzaloztuk
  - Breadboard: lyukrendszer szabványos lyuktávolsággal és megadott összekötöttséggel:




- De nem úsztuk meg teljesen a forrasztást:
  - A kijelzőnek és a billentyűzetnek nem volt tüskesora
    - Tüskesort vettünk és ráforrasztottuk
  - Az RFID tüskesora nem volt kompatibilis a breadboard-dal
    - Átalakítót vettünk, és ráforrasztottuk
- *Forrasztás megtanulható pár youtube tutorial-ból!*

# A kapunyitó szoftvere

- 3 állapotú állapotgépet valósítunk meg:
  - *start* állapot: kiinduló állapot, várjuk a kártyalehúzást
  - *rfid* állapot: lehúzták a kártyát, az RFID olvasó épp küldi a kártya ID karaktereit
  - *code* állapot: számjegy leütésére vár
- A 12 karakteres kártya ID-t összefűzzük a beírt 4 betűs kóddal

  → Így egy 16 karakteres string-et kapunk
- Ha ez a 16 karakter megegyezik a letároltak valamelyikével, akkor kinyitjuk az ajtót

# A kapunyitó szoftvere - 2

```
#include <LiquidCrystal.h>
#include <Key.h>
#include <Keypad.h>

int RFIDResetPin = 2;
int LEDPin = 13;

char userID[17];                                  // ide kerül a kártya ID (12 karakter), utánaírva a lenyomott gombok
int index = 0;
enum estate { start, rfid, code };                // állapotgép aktuális állapota
estate state = start;

LiquidCrystal lcd(49, 47, 48, 46, 44, 42);        // ide kötöttük a kijelzőt

const byte ROWS = 4;
const byte COLS = 3;

char keys[ROWS][COLS] = {  {'1','2','3'},   {'4','5','6'},   {'7','8','9'},   {'*','0','#'} };

byte rowPins[ROWS] = {32, 22, 24, 28};            // ide kötöttük a billentyűzet sor-vezetékeit
byte colPins[COLS] = { 30, 34, 26 };              // ide kötöttük a billentyűzet oszlop-vezetékeit

Keypad keyPad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

// Az elfogadott kártyák és kódjaik. Első 12 karakter: kártya ID, utána írt 4 karakter: a számkód
char* codes[] = {"010B4CF292261234", "010B4CED58F37899", "010B11C56FB19024", "010B1147E8B41290",
            "010B11C5F12F7085", "010B112F3F0B0963", "010B11481C4F7412", "010B1148095A3254",
            "010B1147F3AF6325", "010B114806551589", "010B1147FEA28563", "010B11C56DB33574",
            "010B4CF0D5637412", "010B4CF26DD96521", "010B4CE9B9164589", NULL};
```

# A kapunyitó szoftvere - 3

- A kötelező **setup ()** függvény:

```
void setup() {

    Serial.begin(9600);                 // 0-ás soros port beállítása 9600 bps-re (debug-hoz)
    Serial1.begin(9600);                // 1-es soros port beállítása 9600 bps-re (RFID-hoz)

    pinMode(RFIDResetPin, OUTPUT);      // RFID Reset láb beállítása „kimenet” módba
    digitalWrite (RFIDResetPin, LOW);   // RFID reset, felfutó élet csinálunk. Alacsonyba visszük...
    delay (100);                        // … kicsit várunk ...
    digitalWrite (RFIDResetPin, HIGH);  // … majd magasba emeljük.

    pinMode(LEDPin, OUTPUT);            // Beépített LED lábának beállítása „kimenet”-re

    lcd.begin(16, 2);                   // 16 oszlopos, 2 soros kijelzőnk van
    lcd.print("J\224het a k\240rtya!");  // Írjuk ki rögtön a szöveget
}
```

# A kapunyitó szoftvere - 4

- A kötelező **loop ()** függvény:

```
void loop () {
   while (Serial1.available()) {                    // Először feldolgozzuk az RFID olvasó által küldött kártya ID-t
      int readByte = Serial1.read();                // Soron következő karakter
      if (state == start && readByte == 2) {        // Ha a kiinduló állapotban megjött az „üzenet eleje” karakter (ASCII 0x2),
         state = rfid;                              // átmegyünk a kártya ID olvasó állapotba (rfid)
         index = 0;
      }
      else if (state == rfid && readByte == 3) {    // Ha megjött az „üzenet vége” karakter (ASCII 0x3),
         state = code;                              // átmegyünk a billentyűzetfigyelő állapotba (code)
         lcd.clear();                               // Töröljük a kijelzőt,
         userID[index] = 0;                         // lezáró 0-ál teszünk a kártya ID végére,
         lcd.print(userID);                         // kiírjuk a kártya ID-t a kijelzőre
         lcd.setCursor (0,1);                       // Második sorba megyünk
         lcd.print("J\224het a k\242d!");           // Kiírjuk, hogy jöhet a kód
      }
      if (state==rfid && readByte != 2 && readByte != 10 && readByte != 13 && index<12)    // ha még nincs vége az RFID kód elküldésének
         userID[index++] = readByte;                // eltároljuk a soros porton érkezett karaktert
   }

   char key = keyPad.getKey();                      // Megnézzük, volt-e gombnyomás
   if (state == code && index == 16) {              // Ha kódot olvastunk, és vége (mert megvan a 12 betűs kártya ID és a 4 betűs kód)
      userID[index] = 0;                            // Lezáró 0-át teszünk a végére
      Serial.println(userID);                       // Kiírjuk a soros terminálra debug célból
      checkCardAndCode();                           // Ellenőrizzük a helyességét, és végrehajtjuk, amit kell
      state = start;                                // Visszatérünk kiinduló állapotba
      lcd.clear ();                                 // Kijelző törlése
      lcd.setCursor (0,0);                          // Kurzor a sarokba
      lcd.print("J\224het a k\240rtya!");           // Kiírjuk, hogy jöhet a kártya
   }
   else if (state == code && key)                   // Ha jött egy számjegy, de még nem írta be mind a 4-et
      userID[index++] = key;                        // Hozzáírjuk a többihez
}
```

# A kapunyitó szoftvere - 5

- A kártya és a kód ellenőrzése:

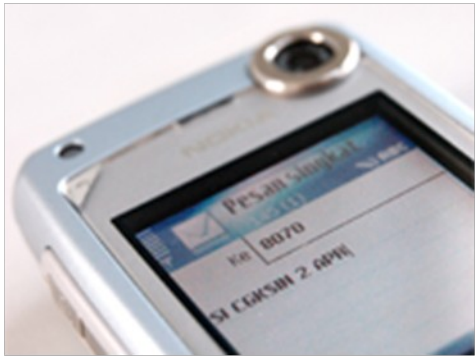
```
void checkCardAndCode () {
   lcd.clear ();                          // kijelző törlése
   lcd.setCursor (0, 0);                  // kurzor a sarokba
   int ix = 0;
   while (codes[ix]) {                    // végignézzük az összes tárolt kódot
      if (!strcmp(userID, codes[ix])) {   // ha egyezik, kész
         lcd.print ("Rendben!");          // szólunk, hogy nyílik a kapu
         digitalWrite (LEDPin, 1);        // kigyújtjuk a LED-et
         break;                           // nem keresünk tovább
      }
      ix++;
   }
   if (!codes[ix])                        // ha a lista végére értünk, és nincs meg,
      lcd.print ("Rossz k\242d!");        // közöljük a rossz hírt

   delay (1000);                          // várunk 1 másodpercet, hogy el lehessen olvasni

   digitalWrite (LEDPin, 0);              // LED lekapcsolása
   digitalWrite (RFIDResetPin, LOW);      // RFID reset, felfutó élet csinálunk. Alacsonyba visszük...
   delay (100);                           // … kicsit várunk ...
   digitalWrite (RFIDResetPin, HIGH);     // … majd magasba emeljük.
}
```

# Összegzés

- Megismertük az Arduino-t
  - Milyen kimenetei/bemenetei vannak
  - Hogyan kell ezekre perifériát illeszteni
  - Hogyan kell a hozzátartozó szoftvert megírni
- Megvalósítottuk az ajtónyitót
  - Bolti, megvásárolható eszközökből
  - Forrasztás csak az összekötögetések miatt kellett
  - Megírtuk a szoftverét

# Kitekintés

- Ezeregy Arduino klón létezik
  - Legálisan, mert open source a hardver is
- Ezeregy Arduino-szerű mikrokontroller kártya kapható
  - OLIMEXINO: AVR mikrokontrollerrel
  - DUINOMITE: PIC mikrokontrollerrel
  - MAPLE: ARM Cortex M3 processzorral
  - NETDUINO: 32 bites ARM processzorral, .NET környezetben programozható (Visual Studio-val)
  - stb...



# Számítógép Architektúrák

*Perifériakezelés a PCI-ban és a PCI Express-ben*

Horváth Gábor
docens
BME Híradástechnikai Tanszék
ghorvath@hit.bme.hu

2012. október 1.
Budapest

# Tartalom

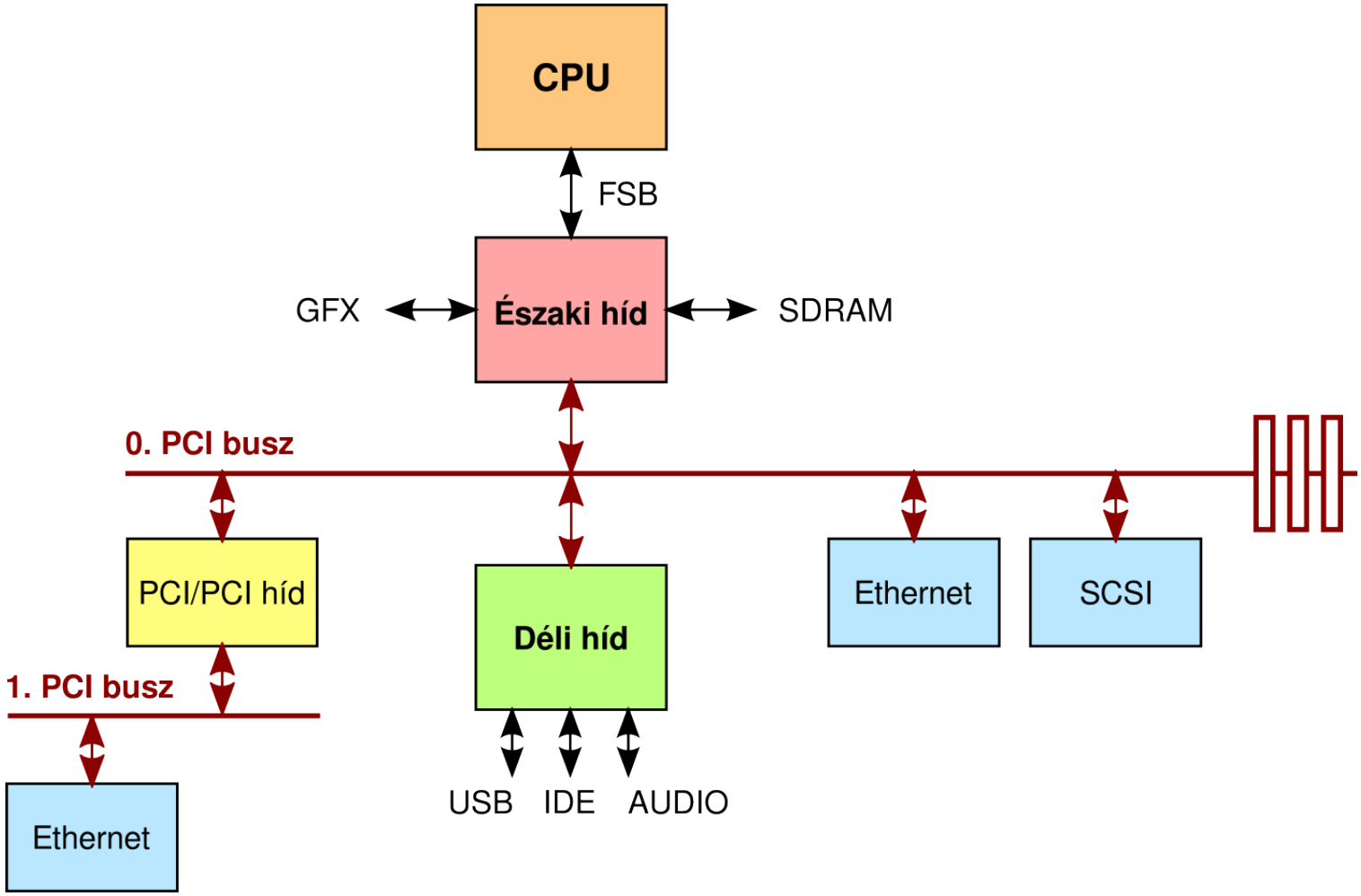
- A gyakorlat célja:
  - A PCI és PCI Express megismerése
- Tartalomjegyzék:
  - PCI alapú rendszerek
    - Egyszerű PCI adatátvitel
    - Arbitráció
    - Megszakításkezelés
    - Konfigurálás
  - PCI Express rendszerek
    - Adatátvitel a PCI Express-ben
    - Megszakításkezelés
    - Konfigurálás

# A PCI

- PCI = Peripheral Component Interfész, 1992, Intel
- Főbb tulajdonságok:
  - Processzorfüggetlen (PC, Macintosh, DEC, HP, SUN szerverek)
  - Automatikus periféria felismerés és konfiguráció támogatása
  - Átvitel: egyenként vagy burst módban
  - 32 vagy 64 bites adategységek
  - Szinkron busz, órajel: 33 MHz, 1995-től 66 MHz
  - Egy PCI buszon max 32 eszköz
  - Egy rendszerben max 256 PCI busz
  - Alacsony fogyasztás támogatása

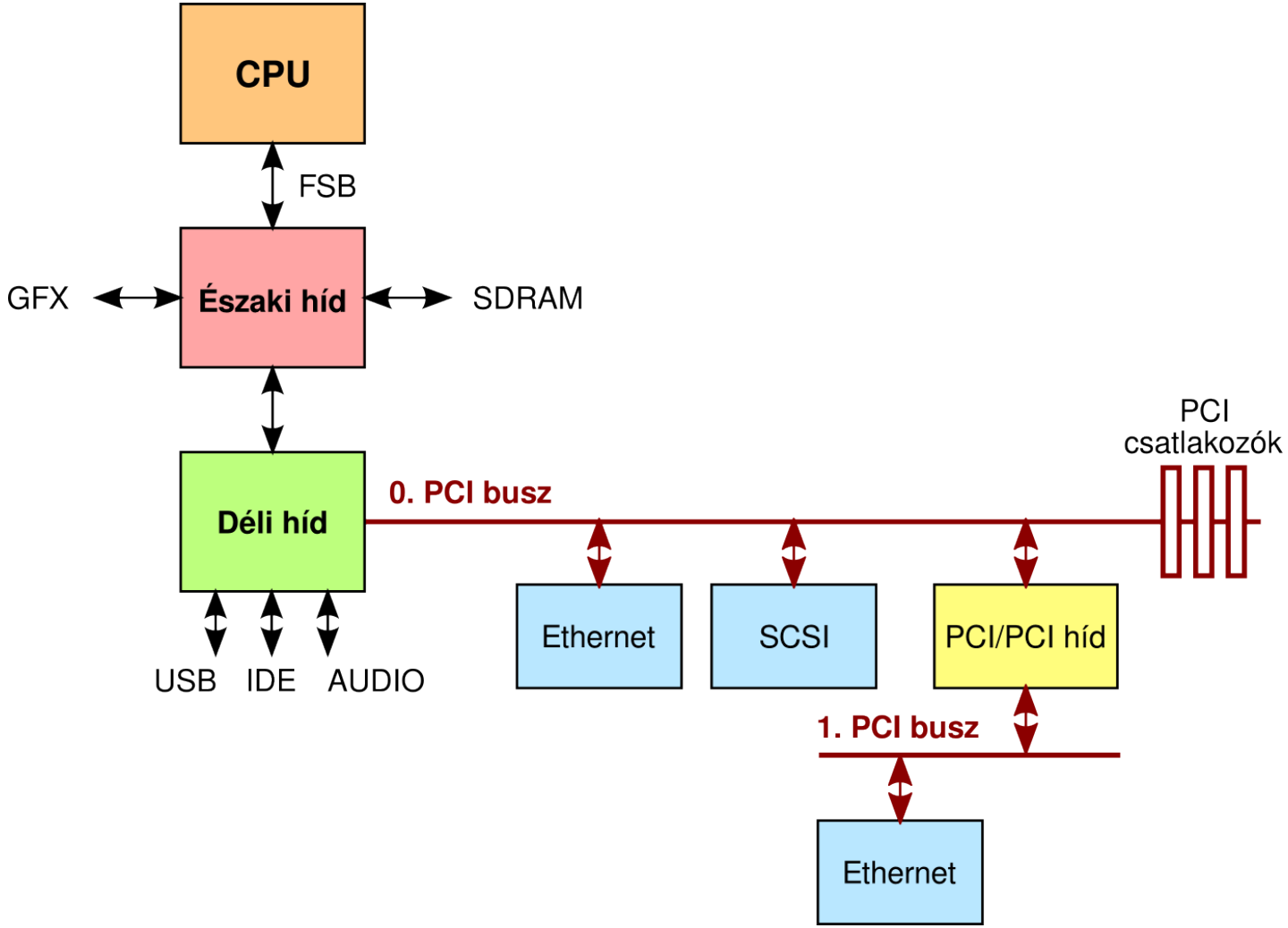
# PCI-t használó rendszerek

- PCI helye az Intel Pentium Pro-ban:



# PCI-t használó rendszerek

- PCI helye a PC-ben a Pentium 4-től kezdve:



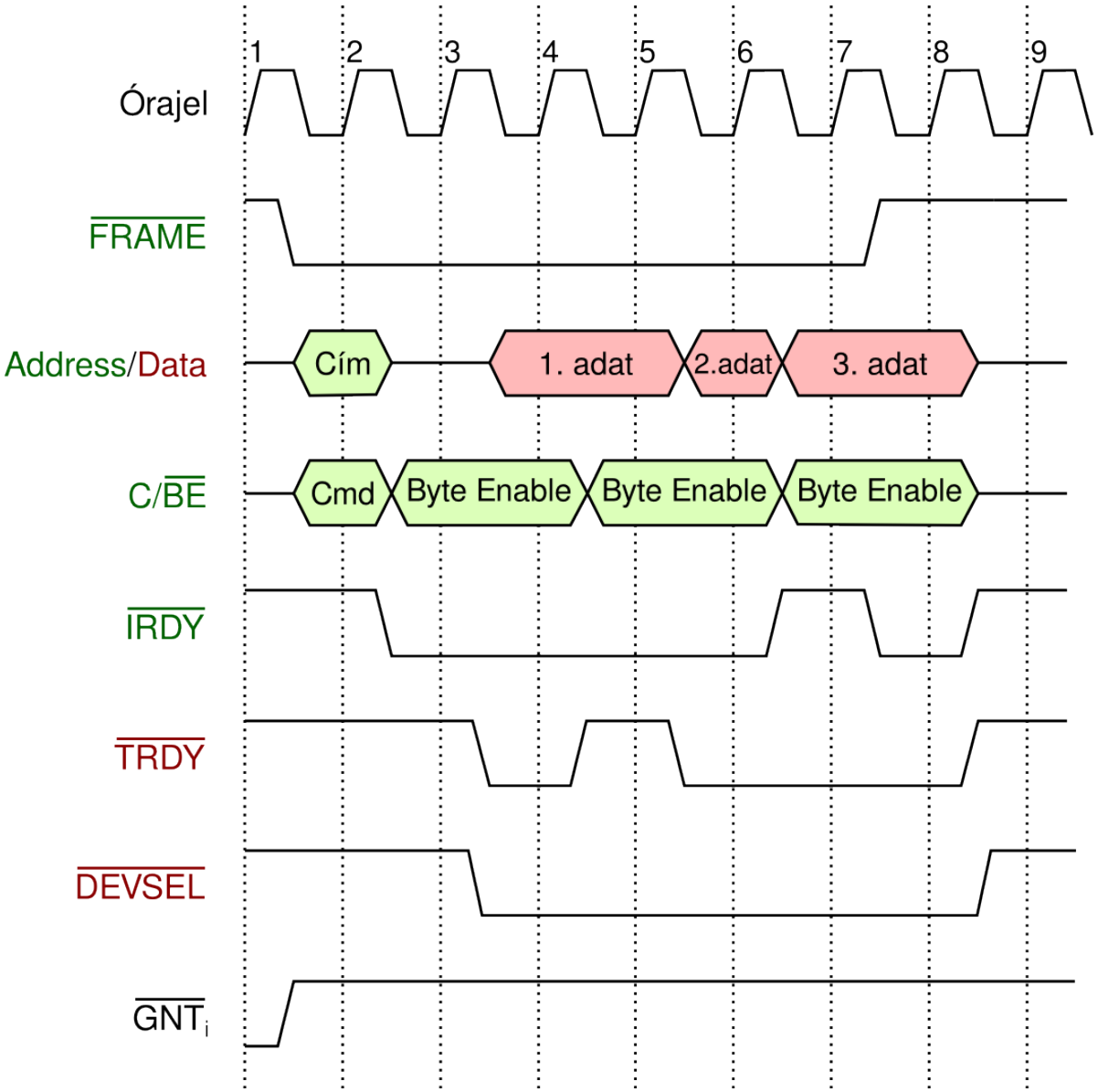
# A PCI elemei

- A PCI busz elemei:
  - Host/PCI híd
    - Processzor I/O és memória kérések ↔ PCI tranzakciók
  - PCI eszközök
    - Max 32/busz, elektromos okokból inkább max. 10
    - Egy eszközön 1-8 function (logikai periféria)
  - PCI/PCI híd
    - Egy PCI sínre egy másik sínt illeszt
- Illeszthető perifériák max száma elméletileg:
  - 8 periféria/eszköz * 32 eszköz/busz * 256 busz/rendszer = 65536 periféria/rendszer

# Adatátvitel

- Minden PCI perifériának max. 6 ablak
  - Ablak lehet:
    - A memória címtartomány egy része
    - Az I/O címtartomány egy része
  - CPU egy PCI periféria memória ablakára végez memóriaműveletet → a perifériával cserél adatot
  - CPU egy PCI periféria I/O ablakára végez perifériaműveletet → a perifériával cserél adatot
  - Az eszköz meg tudja mondani, hogy neki hány és mekkora memória és I/O ablak kell
  - A BIOS ill. az op. rendszer osztja ki az ablakokat
    - A PCI perifériákban be kell állítani az ablakok kezdetét
    - A PCI eszköz figyeli az ablakait célzó műveleteket, ha ilyet lát, válaszol

# Adatátvitel



# Adatátvitel

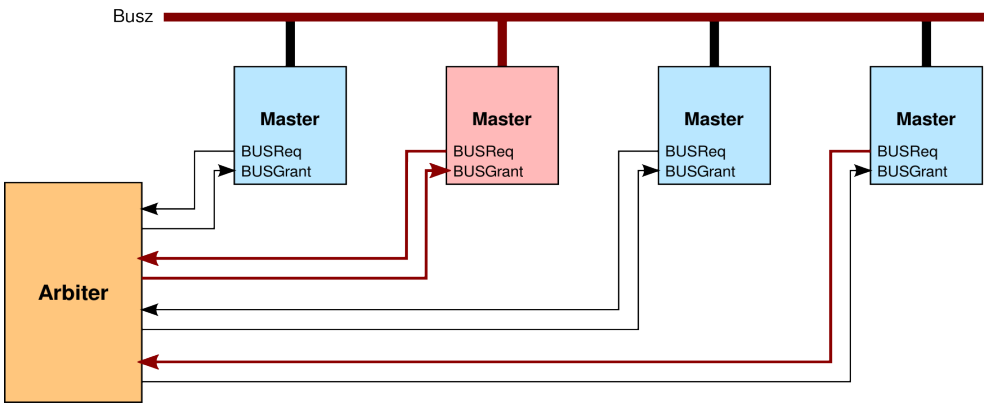
- Forgalomszabályozás:
  - Az $\overline{\text{IRDY}}$ jelzéssel jelzi a kezdeményező, hogy kész a következő adat átvitelére
  - A $\overline{\text{TRDY}}$ jelzéssel jelzi a megcímzett periféria, hogy kész a következő adat átvitelére
- Tranzakciós modellek:
  - Programozott I/O. Kezdeményező: CPU, irányultság: periféria
  - DMA. Kezdeményező: periféria, irányultság: memória
  - Peer-to-peer adatátvitel. Kezdeményező: periféria, irányultság: periféria

# PCI parancsok

| C/$\overline{BE}$ | Parancs |
|---|---|
| 0000 | Megszakítás nyugtázás (Interrupt Acknowledge) |
| 0010 | I/O olvasás (I/O Read) |
| 0011 | I/O írás (I/O Write) |
| 0110 | Memória olvasás (Memory Read) |
| 0111 | Memória írás (Memory Write) |
| 1010 | Konfiguráció olvasás (Configuration Read) |
| 1011 | Konfiguráció írás (Configuration Write) |

# Arbitráció

- Párhuzamos arbitráció:




- Rejtett arbitráció:
  - Versengés és eredményhirdetés az aktuális tranzakció ***közben***
- Algoritmus:
  - Fair
  - Figyelembe veheti a periféria késleltetés-érzékenységét
- Példa:
  - A, B késleltetésérzékeny, X, Y, Z nem az
  - Sorrend: A, B, X, A, B, Y, A, B, Z, A, B, X, A, B, Y, A, B, Z, stb.

# Megszakításkezelés

- Megszakításkérés kétféle képpen:
  - Az interrupt vezeték egyikével
  - Üzenettel jelzett megszakítással (MSI)
- Interrupt vezetékekkel:
  - Minden csatolón 4 van
  - Lehetnek osztott vagy dedikált vezetékek
  - Az, hogy a processzorban hányas megszakításként látszik, lekérdezhető az eszköztől
  - Több eszköz megszakítása is leképezhető ugyanarra a CPU megszakításra → szoftver polling deríti fel a kiváltó perifériát
- Üzenettel jelzett megszakítás:
  - A periféria egy speciális adatot ír egy speciális címre
  - Host/PCI híd figyeli a címet, az írást CPU interrupt-ra képezi le

# Konfiguráció

- Perifériánként 64 db 32 bites konfigurációs regiszter
- Konfigurációs regiszter kiolvasása (PC-n!):
  - 0CF8h I/O cím ← konfigurációs regiszter címe
  - Regiszter tartalma: kiolvasható a 0CFCh I/O címről
- Konfigurációs regiszter megváltoztatása (PC-n!):
  - 0CF8h I/O cím ← konfigurációs regiszter címe
  - 0CFCh I/O cím ← konfigurációs regiszter új értéke
- Konfigurációs regiszterek címzése:

| 31 | 30...24 | 23...16 | 15...11 | 10...8 | 7...2 | 1...0 |
|---|---|---|---|---|---|---|
| 1 | Fenntartott | Busz szám | Eszköz | Function | Reg. szám | 00 |

# Konfiguráció

- Konfigurációs regiszter írása/olvasása:
  - Host/PCI híd I/O műveletet lát a 0CF8h és a 0CFCh címen
  - Látja, melyik eszköz van megcímezve

    → beállítja az $\overline{\text{IDSEL}}$ vezetékét 0-ba, a többiekét 1-be
  - Konfiguráció írás/olvasás tranzakciót indít

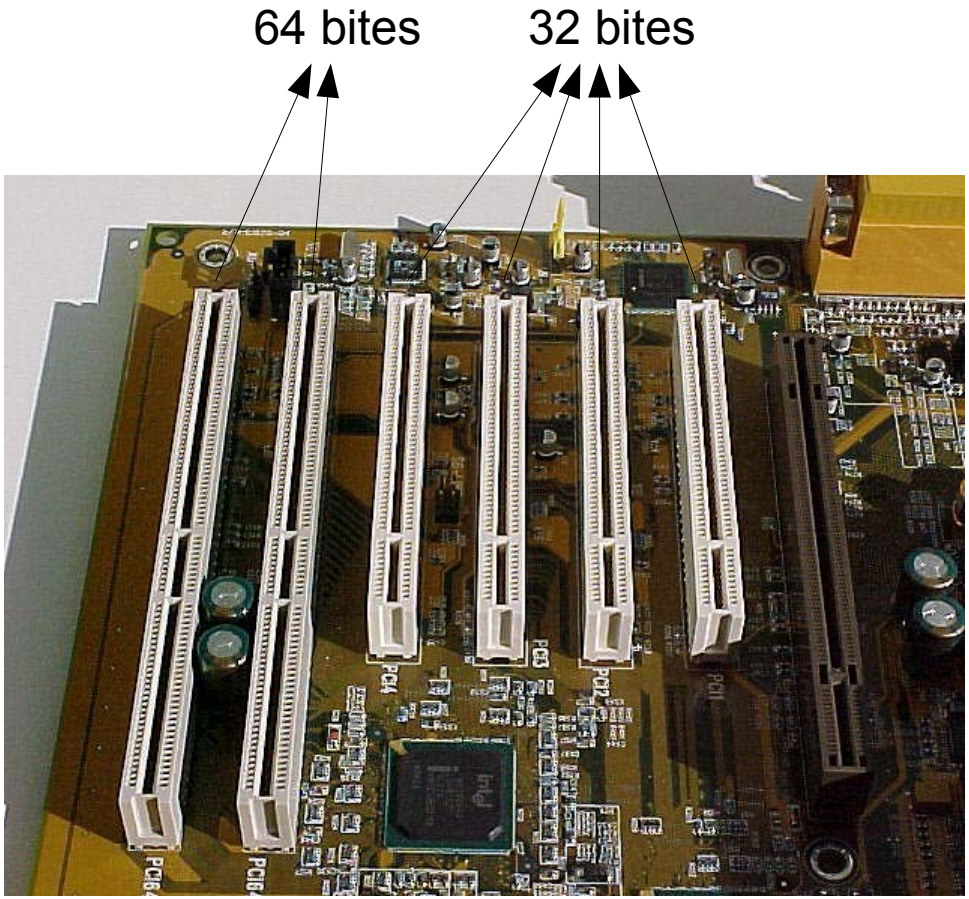
    → csak a kijelölt periféria figyel!
    - C/BE vonalak: konfiguráció írás/olvasás kódja
    - A/D vonalakra:
      – Konfiguráció típusa
      – A megcímzett function száma
      – Kért regiszter száma
    - Regiszter tartalmának átvitele: mint az egyszerű adatátvitelnél

# Konfiguráció

- Regiszterek tartalma:
  - A 64 regiszterből 16 kötött:
    - VendorID, DeviceID, Revision
    - Class Code
    - Interrupt Pin
    - Interrupt Line
    - Base Address Register 0...5
- Rendszerindítás:
  - Eszközök szisztematikus lekérdezése
  - Ablakok kiosztása
  - Device driver betöltése
  - További eszközfüggő konfiguráció
  - ...

# PCI csatoló

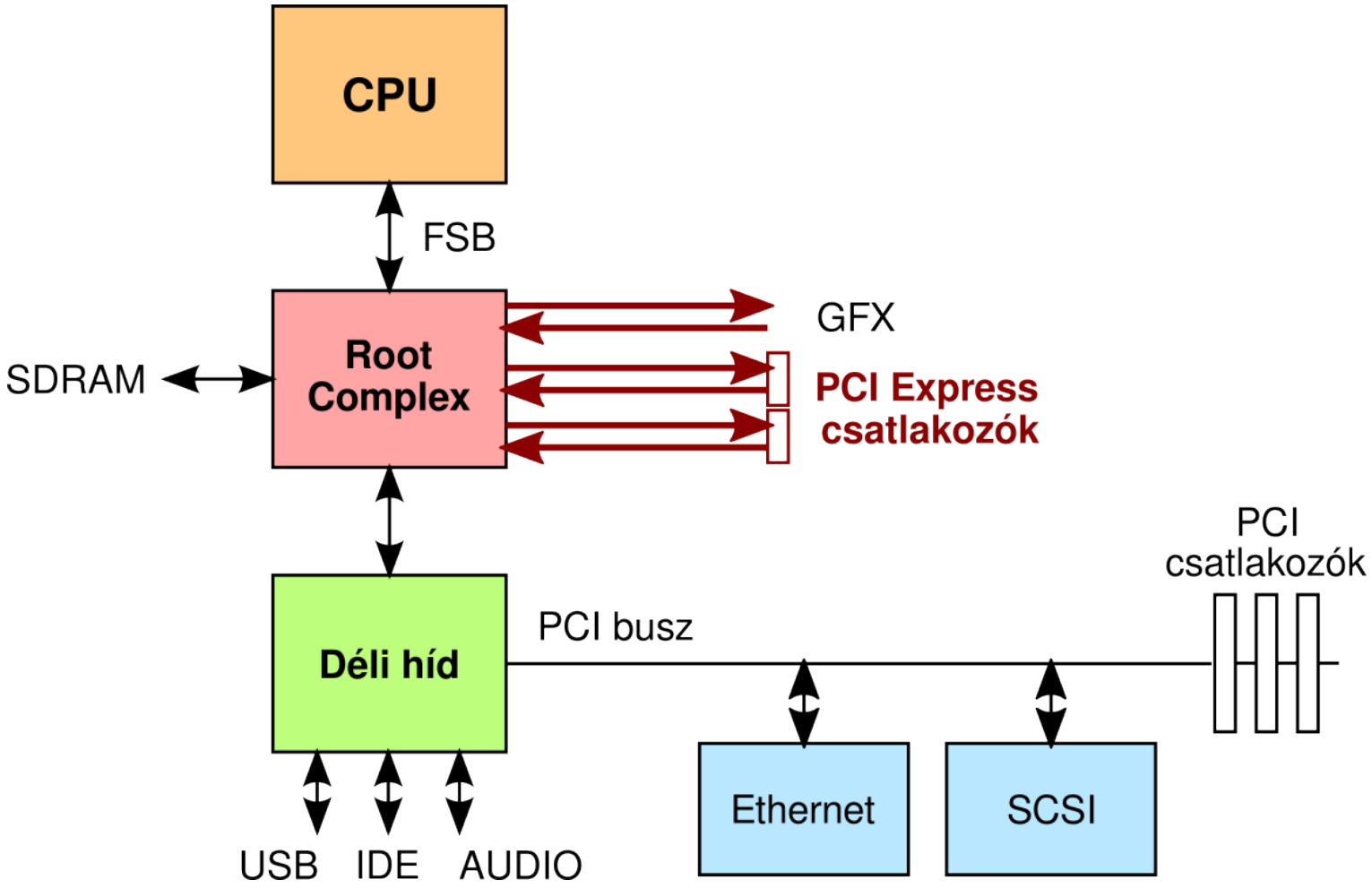
- Kötelezően használandó:
  - Órajel, Reset
  - A/D[0...31]
  - C/$\overline{\text{BE}}$[0...3]
  - $\overline{\text{FRAME}}$
  - $\overline{\text{IRDY}}$, $\overline{\text{TRDY}}$, $\overline{\text{STOP}}$
  - $\overline{\text{DEVSEL}}$, $\overline{\text{IDSEL}}$
  - Paritás, $\overline{\text{PERR}}$, $\overline{\text{SERR}}$
  - $\overline{\text{REQ}}$, $\overline{\text{GNT}}$
- Opcionális:
  - A/D[32...63]
  - C/$\overline{\text{BE}}$[4...7]
  - $\overline{\text{INTA}}$, $\overline{\text{INTB}}$, $\overline{\text{INTC}}$, $\overline{\text{INTD}}$
  - $\overline{\text{REQ64}}$, $\overline{\text{ACK64}}$
  - $\overline{\text{CLKRUN}}$



# PCI Express

- Célok:
  - PCI-nál nagyobb adatátviteli sebesség
  - Lehető legteljesebb szoftveres PCI kompatibilitás
- Leglátványosabb eltérések:
  - Soros átvitel
    - Így kiküszöbölhető a párhuzamos esetben tapasztalható jelek közötti csúszás
  - Pont-pont összeköttetések
    - Nem osztott a közeg → nincs versengés, várakozás, éhezés, arbitráció

# Egyszerű PCI Express rendszer



# Egyszerű PCI Express rendszer

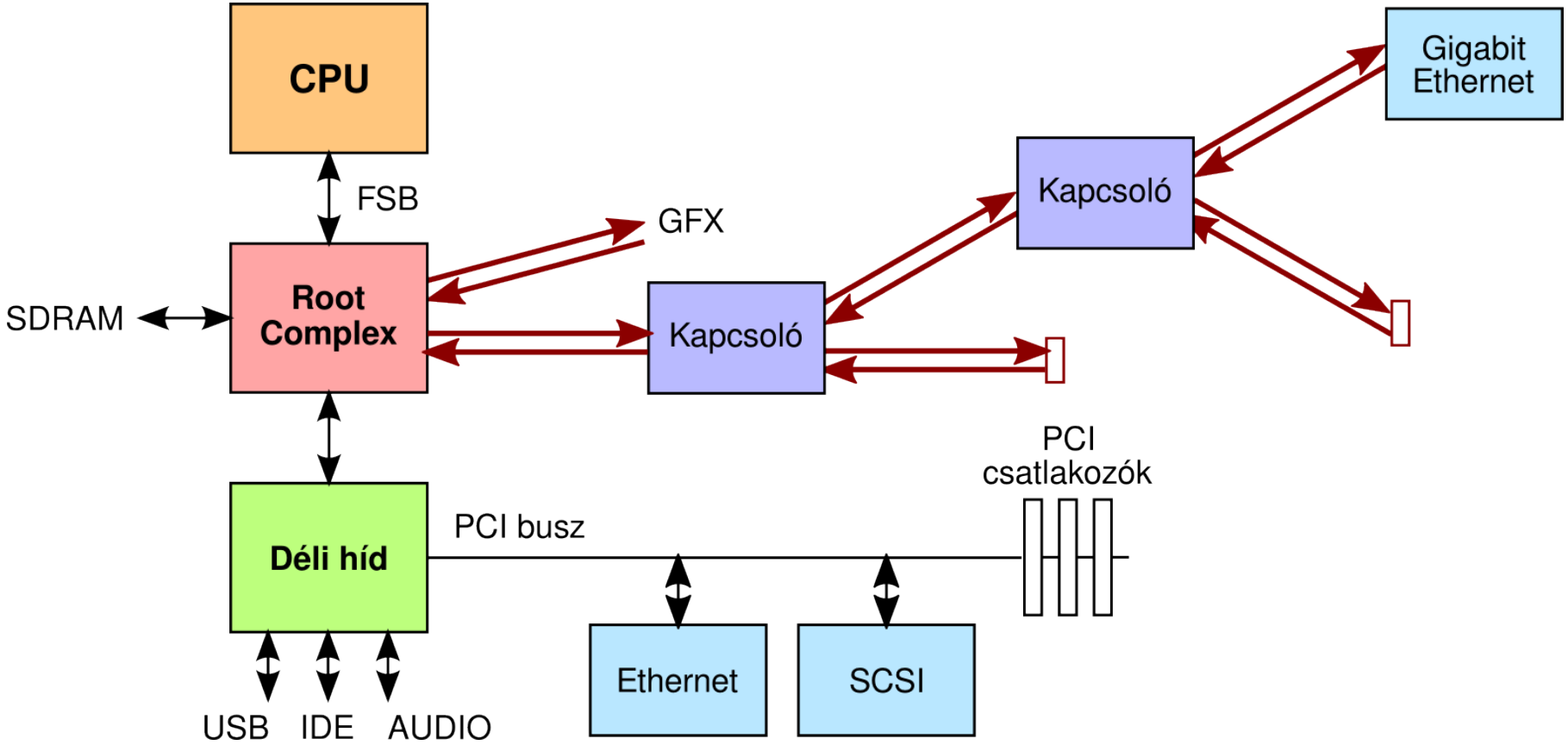
- Adatátvitel sorosan:
  - Órajel nincs (automatikus, a 0/1 átmenetekből, PLL-lel)
  - Egy érpár: 500 MB/s
  - Két eszköz között full duplex (kétirányú) kapcsolat:

    → 2 soros érpár
  - Két eszköz több full duplex soros kapcsolattal is össze lehet kötve
    - 1 full duplex soros kapcsolat neve: ***lane*** (pálya)
    - Szabvány: 1x, 4x, 8x, 16x, 32x lane
    - Több lane esetén a bitek párhuzamosan vihetők át

      → **De nem szinkronban !!!**

      – 32x pálya esetén 32x500 MB/s = 16 GB/s

# PCI Express rendszer kapcsolókkal

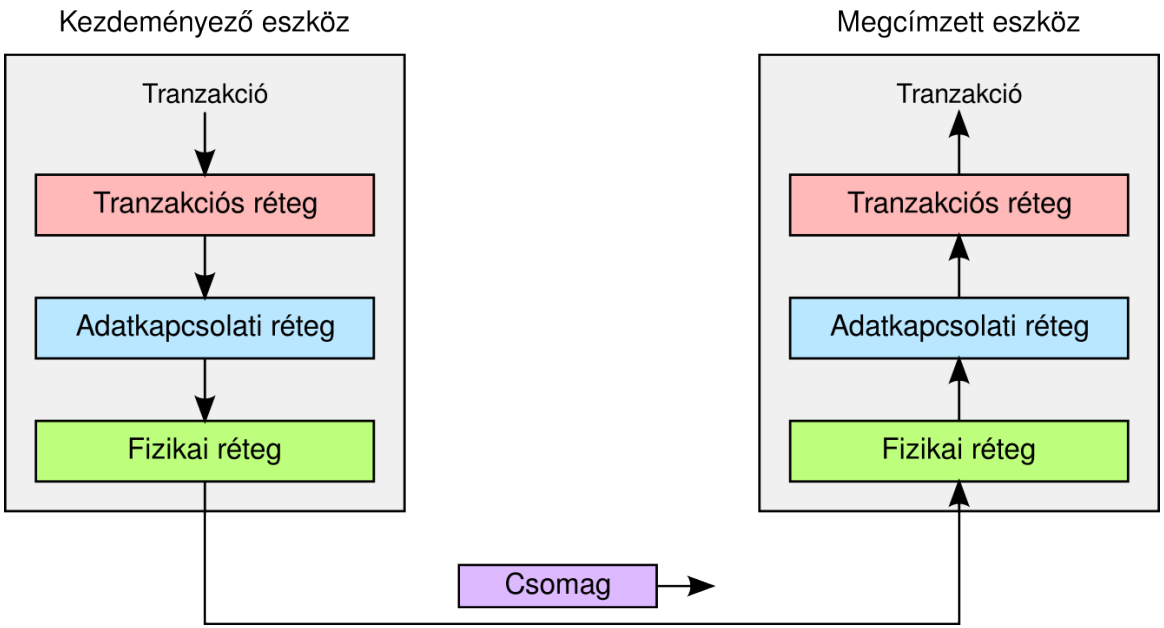


# PCI Express rendszer kapcsolókkal

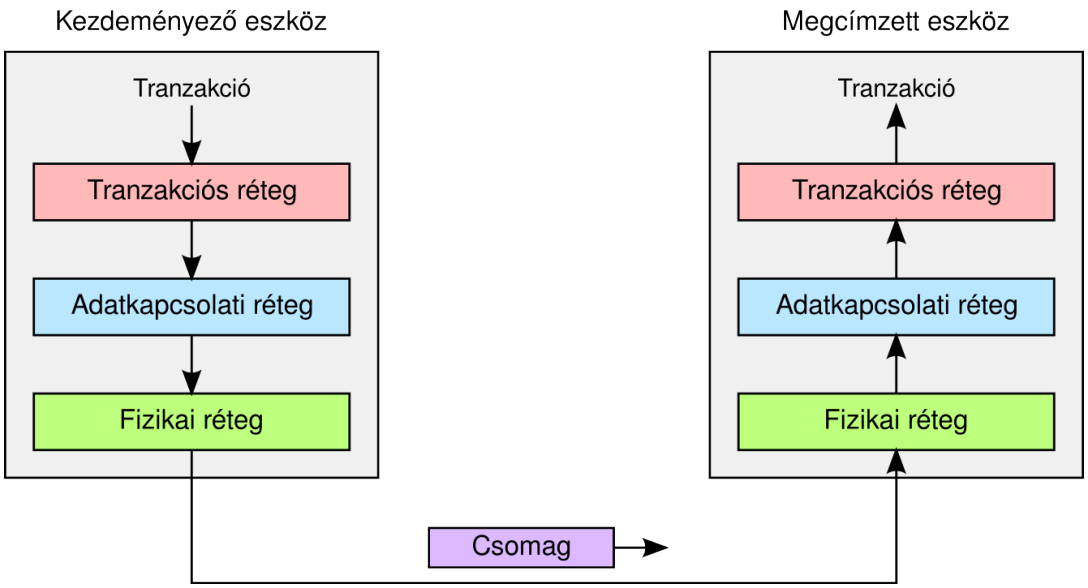
- Csomagkapcsolt hálózat:
  - Eszközök kapcsolókon keresztül kötődnek a Root Complexhez
  - Tranzakciók csomagként utaznak

# Tranzakciók továbbítása

- Tranzakció 3 rétegen megy keresztül, mielőtt a soros érpárra kerül:

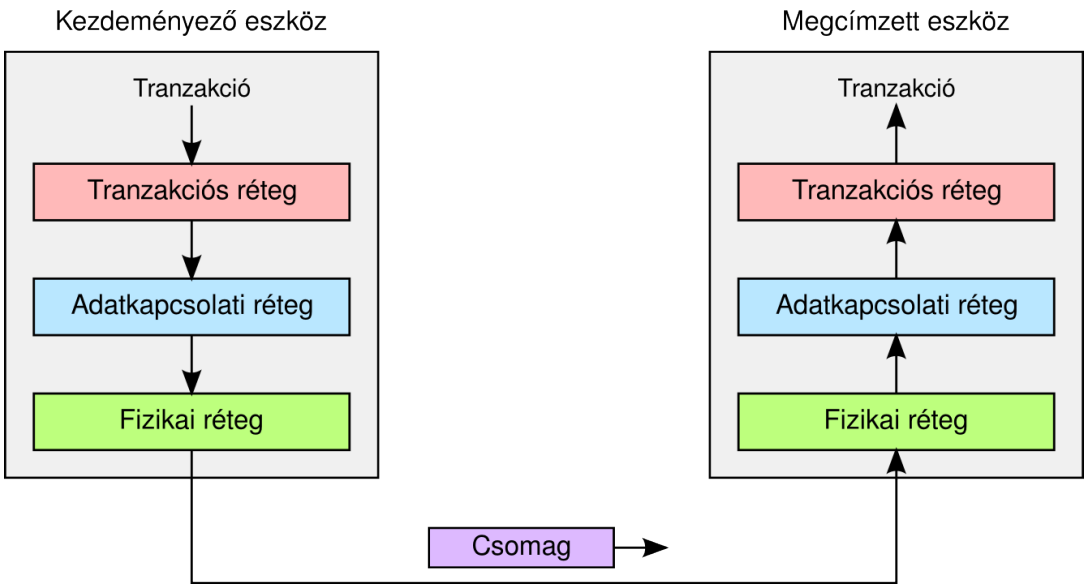


# Tranzakciók továbbítása



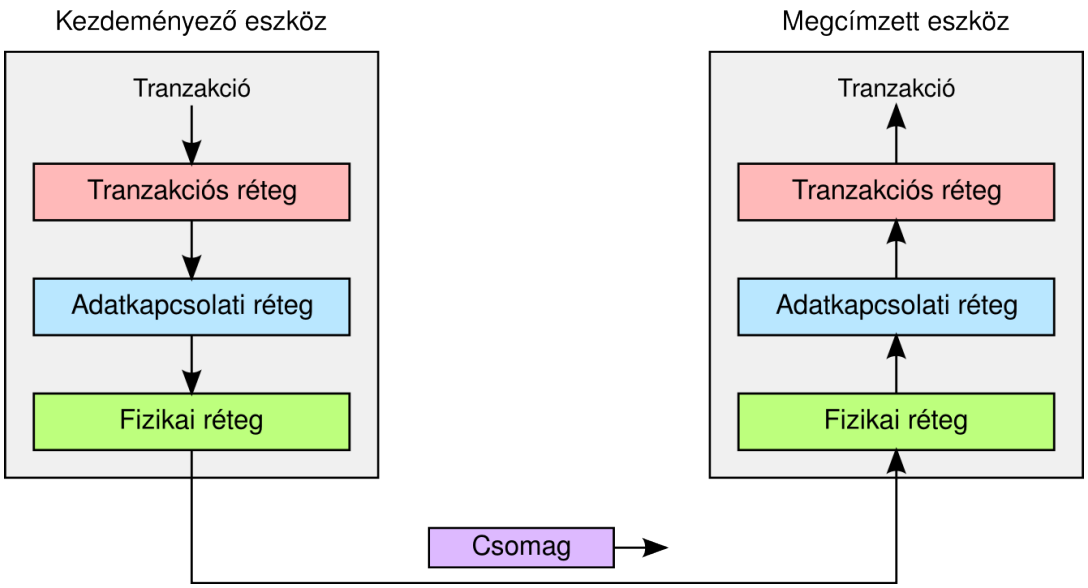

- Tranzakciós réteg: tranzakció becsomagolása
  - Fejléc:
    - Command, bájtsorrend (PCI: C/$\overline{BE}$)
    - Cím (PCI: Address)
  - Rakomány:
    - Átvinni kívánt adat
  - CRC

# Tranzakciók továbbítása



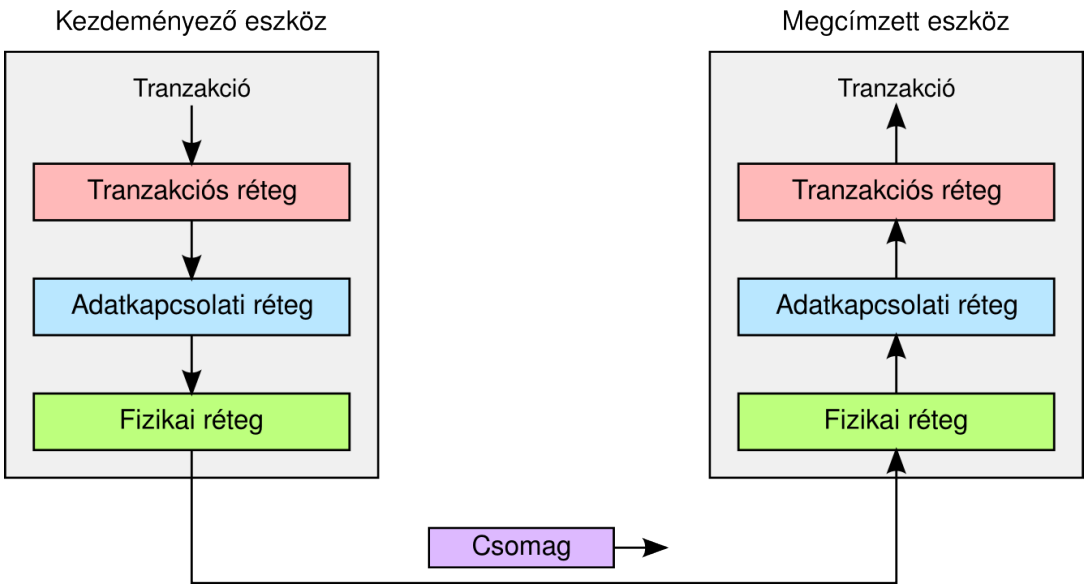

- Adatkapcsolati réteg: csomag hibamentes eljuttatása a szomszédba
  - Újabb fejléc: sorszám (auto-inkrementált)
  - Újabb CRC (sorszámot is védi)

# Tranzakciók továbbítása

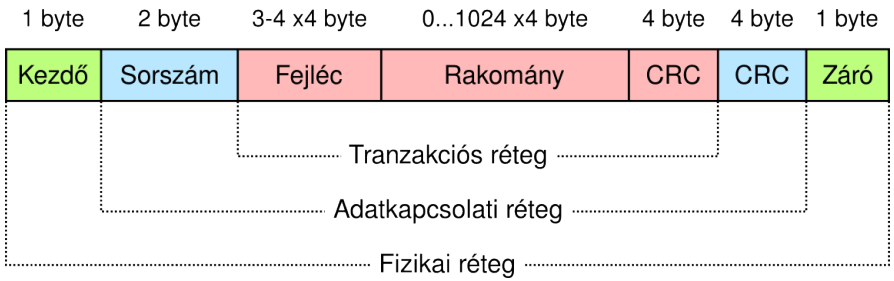



- Fizikai réteg: csomag átvitele a soros érpáron
  - Csomaghatárok kijelölése speciális bájtokkal
  - Megfelelő számú 0/1 átmenet biztosítása:
    - Bitek összekeverésével
    - 8 bitet 10 biten kódolással

# Tranzakciók továbbítása



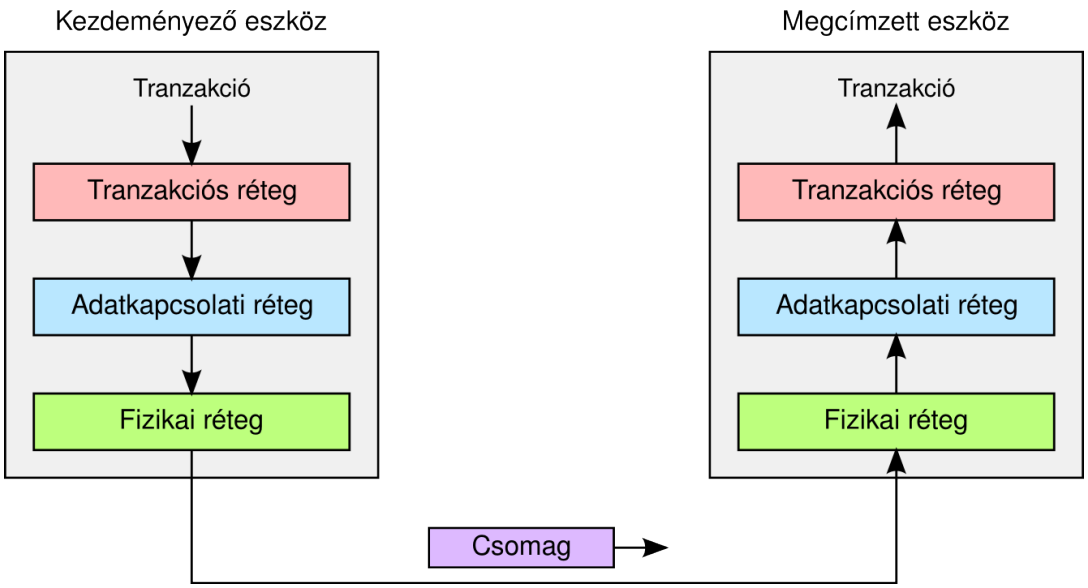

- Csomag formátuma:




- 1 db 32 bites adat átvitele:

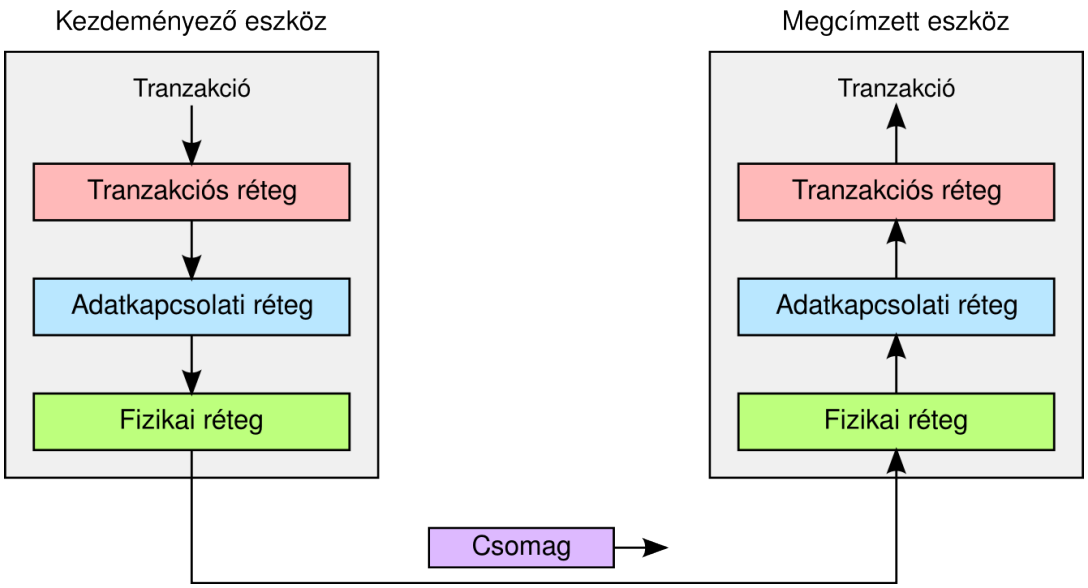
  1+2+3*4+1*4+4+4+1 = 28 bájt

# Tranzakciók továbbítása



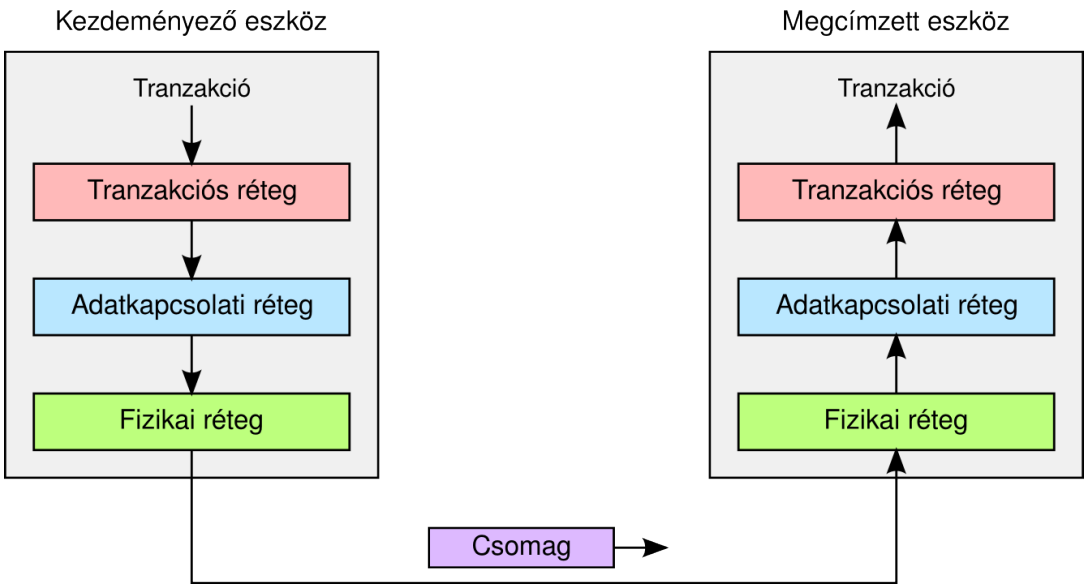

- Vevő oldal, fizikai réteg:
  - Csomaghatárok azonosítása
  - 8-ból 10 bites kódolás visszaállítása
  - Bitösszekeverés visszaállítása

# Tranzakciók továbbítása




- Vevő oldal, adatkapcoslati réteg:
  - CRC és sorszám ellenőrzése
  - Ellenőrzés függvényében negatív vagy pozitív nyugta küldése
  - Negatív nyugta esetén a feladó újraküldi a csomagot

# Tranzakciók továbbítása




- Vevő oldal, tranzakciós réteg:
  - CRC ellenőrzése
  - Tranzakció végrehajtása

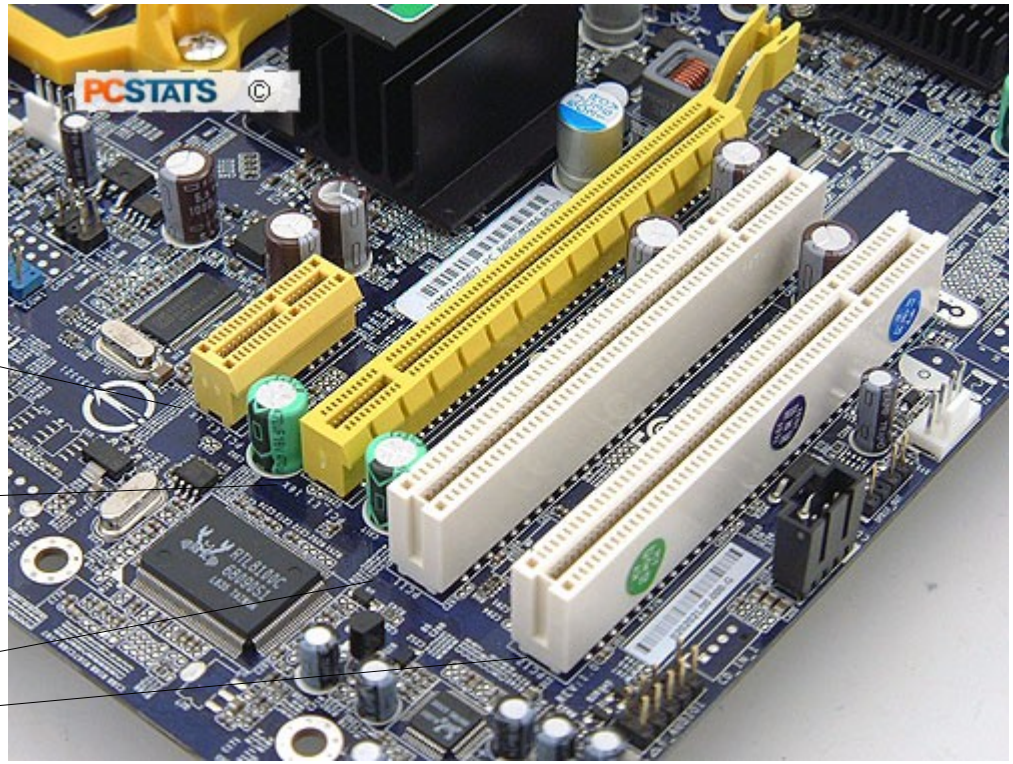
# Tranzakciók továbbítása

- Posted vs. Non-posted tranzakciók:
  - Posted: válasz nem szükséges, pl. írási tranzakciók
  - Non-posted: válasz szükséges, pl. olvasási tranzakciók
    - A tranzakció csak egy olvasási kérés
    - A megcímzett periféria egy újabb tranzakcióban válaszol a kért adatokkal a rakományban
- QoS: fejlécben „forgalmi osztály” mező
  - Sürgősség jelölése
  - A sürgős tranzakció a továbbítás során megelőzheti a nem sürgőset
- Forgalomszabályozás
  - Beérkező csomagok feldolgozásig átmeneti bufferben tárolódnak
  - Fogadó fél rendszeresen jelzi a küldőnek a szabad buffer méretét
  - Ha a küldő nem lát elég buffert a fogadónál, vár a küldéssel

# Megszakításkezelés és konfigurálás

- Megszakításkezelés
  - Üzenet alapú (MSI) → mint a PCI-ban
  - Jelvezeték alapú:
    - A periféria két speciális üzenetet küldhet a Root Complex-nek:
      - Vegye úgy, hogy 0-ba húzta az egyik interrupt vonalat
      - Vegye úgy, hogy 1-be emelte az egyik interrupt vonalat
    - Az így jelzett megszakítás kezelése mint PCI-ban
- Konfiguráció:
  - Nem csak 64 regisztere lehet egy perifériának, hanem 1024
  - Első 64 írása/olvasása: mint PCI-ban (teljes kompatibilitás)
  - Továbbiak: másféle címzés (64 bites, nem részletezzük)

# PCI Express csatoló



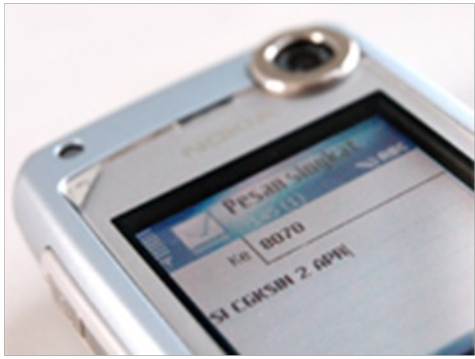

PCIe x1

PCIe x16

PCI

# Összegzés

- Mit tanultunk?
  - PCI: busz, párhuzamos
  - PCI Express: pont-pont, soros
  - Hogy juttatja el a PCI és a PCI Express a CPU kéréseit a perifériáknak?
  - Hogy oldották meg a gyakorlatban az arbitrációt a PCI-ban?
  - Hogy oldották meg a gyakorlatban a megszakításkezelést?
  - Hogy oldották meg a gyakorlatban a perifériák konfigurálását?



# Számítógép Architektúrák

*Az USB csatolófelület*

Horváth Gábor
docens
BME Híradástechnikai Tanszék
ghorvath@hit.bme.hu

2012. október 12.
Budapest

# Tartalom

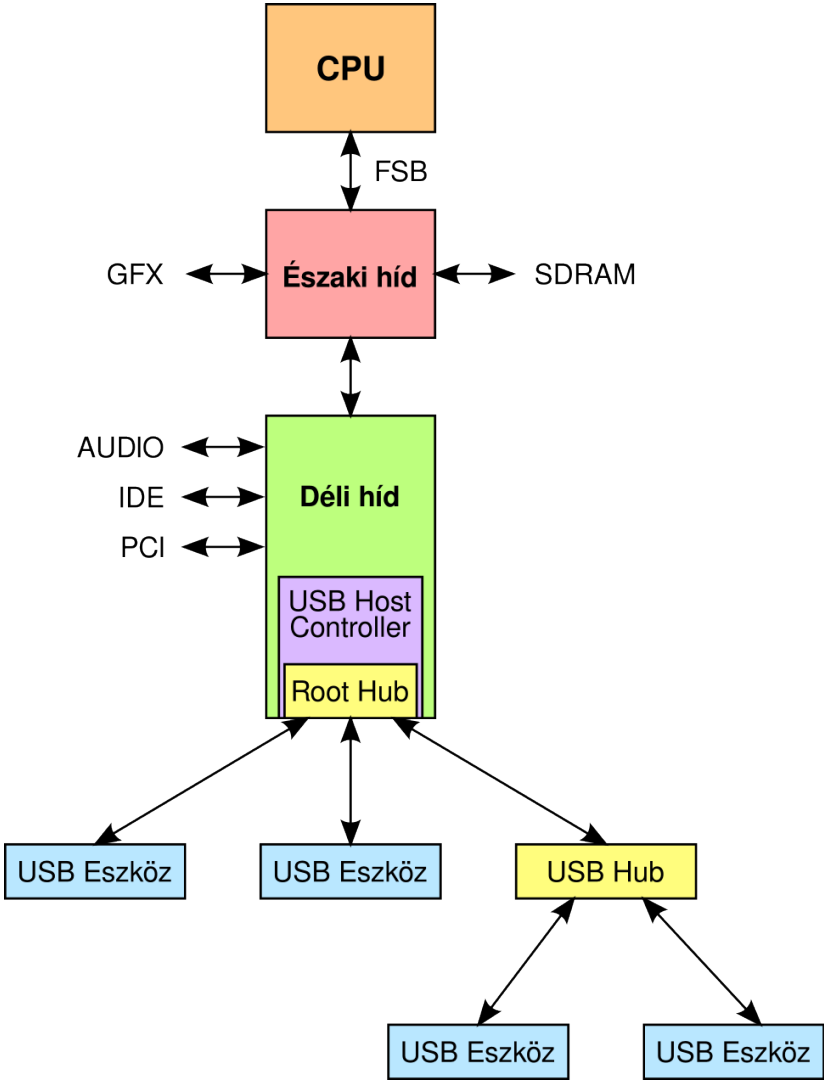
- A gyakorlat célja:
  - Az USB csatolófelület megismerése
- Tartalomjegyzék:
  - Motiváció
  - Az USB felépítése
  - Az USB 1.1
    - Adatátvitel
    - Interrupt-ok
    - Csatlakozók, tápellátás
    - Konfigurálás
  - Az USB 2.0
    - Mi változott?
  - Az USB 3.0
    - Mi változott?

# Motiváció

- 90-es évek: a PC-s perifériák csatlakozása körül káosz
  - Ezernyi gyártóspecifikus csatlakozó
  - Ha nem volt az alaplapon, illesztőkártya kellett
  - A PC-be tehető illesztőkártyák száma korlátos
  - Az ezernyi csatlakozóhoz különböző kábelek, hosszabbítók kellenek
- USB megjelenése
  - 1994, Compaq, DEC, Intel, Microsoft, NEC, Nortel
  - Elsődleges célpont a PC, de azóta minden architektúra támogatja
- Főbb tulajdonságok:
  - Szabványos fizikai csatlakozók
  - Több periféria köthető 1 csatlakozóra
  - Max. 127 eszköz vezérlőnként
  - Perifériák nem terhelik a címteret és nem kell interrupt vonal sem
  - Automatikus periféria felismerés és konfiguráció támogatása
  - A számítógép kikapcsolása nélkül illeszthető/eltávolítható perifériák
  - Olcsó vezérlőáramkörök, alkatrészek
  - A tápellátás az USB kábelen keresztül megoldható (általában)
  - Alacsony fogyasztás támogatása

# Az USB felépítése

- Fa topológia
  - Levelek: a perifériák
  - Csomópontok: USB hub-ok
  - Gyökér: root hub
    (a host controller része)
- Periféria lehet:
  - Egyszerű eszköz
  - Összetett eszköz:
    - Fizikailag egy eszköz
    - Logikai több eszköz



# Mitől busz

- Megszoktuk, hogy a busz osztott közeg
  - Minden szereplő mindent hall
- De ez egy fa!
- Mégis busz:
  - Egyetlen master van, a root hub
  - Amit a master beszél, azt mindenki hallja (terjed a fában)
  - Akire vonatkozik, az odafigyel, a többi nem
  - Amit a periféria mond, az a fában felfelé haladva éri el a mastert
- Nem kell arbitráció!
  - Mivel csak 1 master van
  - Senki nem szólalhat meg, csak ha a master erre utasítja
- Ha busz, akkor megvan a hátránya is:
  - Minél többen csatlakoznak rá, annál lassabb lesz

# USB 1.1

# Az USB 1.1

- 1998, az első igazán elterjedt változat
- Adatsebességek
  - 1.5 Mb/s (Low Speed, LS)
  - 12 Mb/s (Full Speed, FS)

# Tranzakciók és keretek

- A perifériákkal **tranzakciók** segítségével kommunikálunk
- LS és FS sebességű tranzakciók
  - Az LS tranzakciókat az LS és az FS eszközök is látják
  - Az FS tranzakciókat csak az FS eszközök látják
- Mindig a master, a root hub indítja
- Tartalma:
  - Megcélzott periféria azonosítója
  - Ki- vagy bemeneti művelet
  - Átvinni kívánt adatok
    - Kimeneti irány esetén a root hub teszi a buszra
    - Bemeneti irány esetén a megcélzott periféria
- A tranzakciók **keretekben** utaznak
  - Minden keret 1 ms ideig tart (FS sebességgel: 1500 bájt/keret)
  - Minden keret több tranzakciót is szállíthat
  - Csak *teljes* tranzakciókat szállíthat (lehet, hogy nem lesz kitöltve)

# Adatátviteli módok

- A perifériák sokfélék
  - Van, akinek a kis késleltetés a legfontosabb
  - Van, akinek a hibamentes átvitel
  - Stb.
- **Adatátviteli módok**: hogyan vigyük át az adatot
- USB-ben:
  - Izokron
  - Bulk
  - Control
  - Interrupt

# Izokron adatátviteli mód

- Késleltetésérzékenyeknek
- Cél: folyamatos, ingadozásmentes adatfolyam
- Hangszóró, kamera, stb.
- Jellemzők:
  - Csak FS periféria jöhet szóba
  - A periféria minden keretben garantáltan kap lehetőséget tranzakcióra
    → Senki nem szoríthatja ki onnan
  - A rendszer **nem** garantálja a hibamentességet
    → Úgy sincs idő újraküldeni, ha hibás az adat
  - Rakomány: max. 1023 bájt, overhead: 9 bájt

# Bulk adatátviteli mód

- Nem számít a késleltetés, csak a hibamentes átvitel
- Printer, szkenner, pendrive, stb.
- Elviselik, ha néha kimaradnak 1-1 keretből
- Jellemzők:
  - Csak FS periféria jöhet szóba
  - A késleltetésre nincs garancia
  - Minden hibadetektáló és javító eszközt bevet
  - Adatméret: 8, 16, 32 vagy 64 bájt, overhead: 13 bájt

    → Óriási az overhead!!!

# Control adatátviteli mód

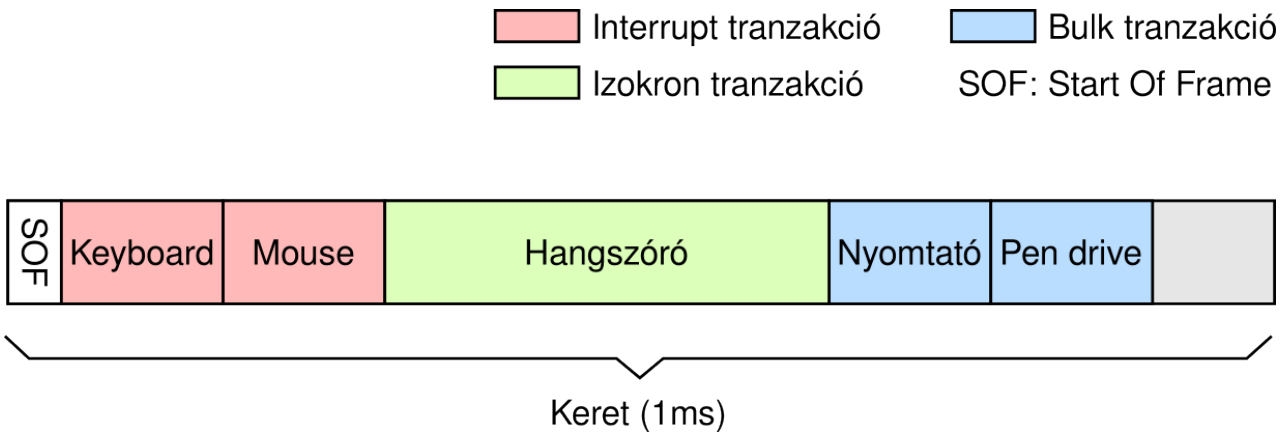
- A perifériák konfigurálását, beállítását szolgálják
- Szabványos és gyártóspecifikus parancsok küldésére
- Jellemzők:
  - LS és FS eszközök is támogatják
  - Ha van átvinni kívánt control tranzakció, a rendszer helyet szorít neki
    - De csak a keretméret 10%-a erejéig
  - Garantált hibamentes átvitel
  - Adatméret:
    - LS esetben: max. 8 bájt, overhead: 63 bájt
    - FS esetben: max. 64 bájt, overhead: 45 bájt

# Interrupt adatátviteli mód

- A periféria nem kezdeményezhet! Interruptot sem!
- **Polling**-al kell kérdezgetni, van-e közlendője

  → Erre szolgál az interrupt tranzakció
- Jellemzők:
  - LS és FS eszközök is támogatják
  - Irányultság: mindig beviteli
  - Az eszköz meg tudja mondani, milyen sűrűn szeretné, hogy kérdezgessék
    - 1 és 255 keretidő között állítható
  - Garantált hibamentes átvitel
    - Hiba esetén ismétlés a következő keretben
  - Adatméret:
    - LS esetben: max. 8 bájt, overhead: 19 bájt
    - FS esetben: max. 64 bájt, overhead: 13 bájt

# A keretek kitöltése



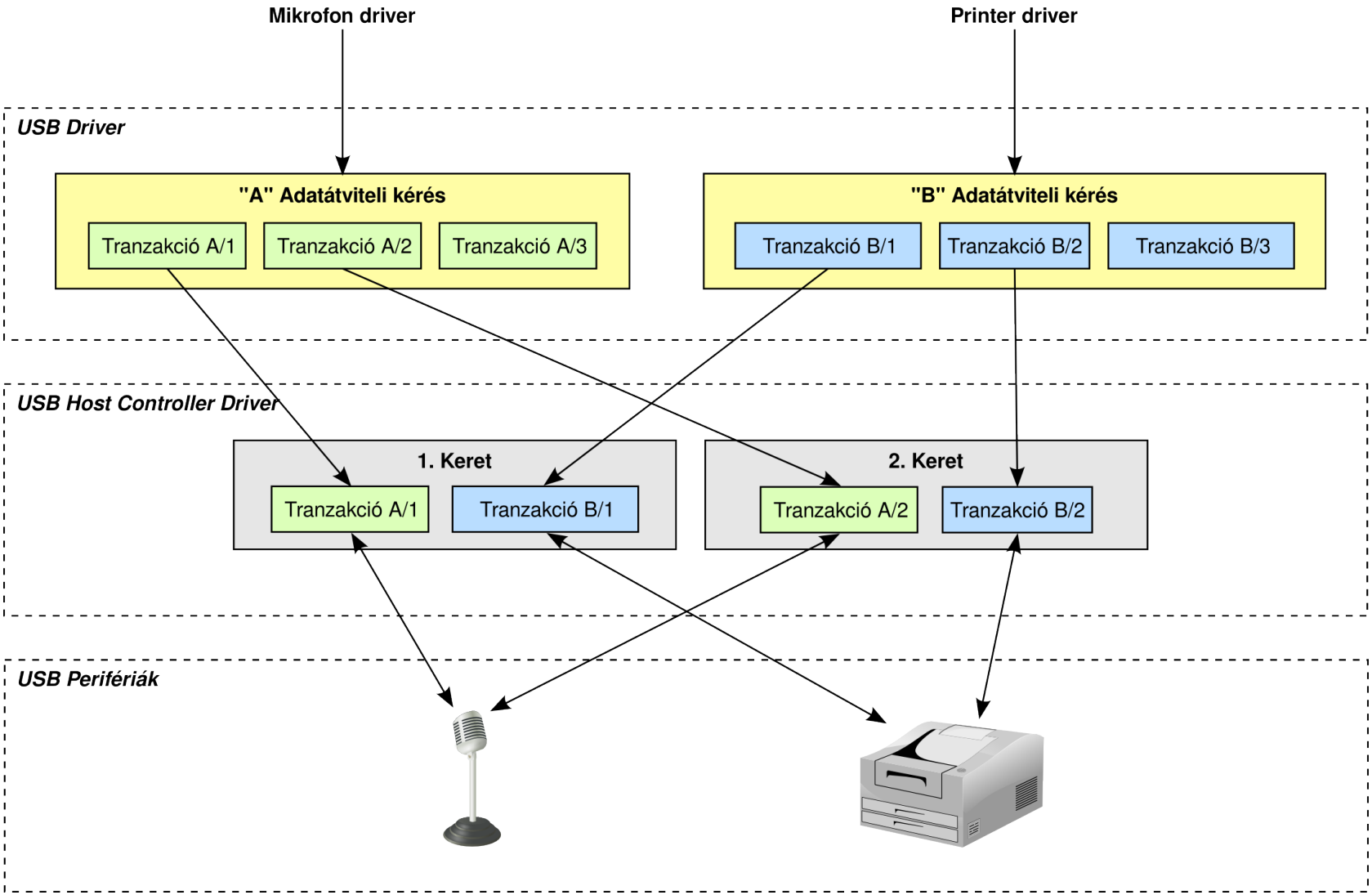

- Feltöltés:
  - Az izokron és interrupt tranzakcióknak elsőbbsége van
    - De nem foglalhatnak többet a keret 90%-ánál
    - Ha csatlakozik egy új periféria, amivel több lenne 90%-nál

      → nem engedik belépni
  - A fennmaradó 10%-ban elsőbbsége van a control tranzakcióknak
  - Maradék: bulk. Ha marad.

# Az adatátvitel folyamata

- **Hardver szereplők:**
  - Host controller a root hub-bal
  - Hub-ok
  - Perifériák
- **Szoftver szereplők:**
  - A periféria driver
  - Az USB driver
  - Az USB Host Controller driver
- **Az adatátvitel:**
  - A periféria driver kezdeményezi
    - Összeállítja az adatátviteli kérést (IRP: I/O Request Packet)
    - Továbbítja az USB driver-nek
  - Az USB driver
    - Az IRP-t feldarabolja a tranzakció max. méretének megfelelő darabokra
    - Továbbítja az USB Host Controller driver-nek
  - A Host Controller Driver
    - Tranzakciókat képez
    - A tranzakciókat keretekhez rendeli
    - 1 ms-onként kibocsát egy keretet

# Az adatátvitel folyamata



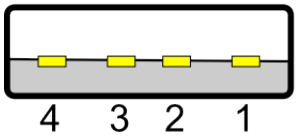
# Tápellátás

- A jelvezetékek mellett „föld” és „+5V” vezeték is van

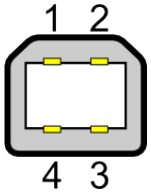
  → Perifériák ebből is megoldhatják a tápellátásukat!
- A root hub minden csatlakozójából max. 500mA vehető fel
  - Ebből gazdálkodhat minden rá kötött eszköz
- Szabvány előírás: minden hub porton legyen min. 100mA
  - Vagyis USB-ről táplálkozó hub-nak max. 4 port-ja lehet (500mA-t kap, 4x100mA kell a kimenő port-okon, +saját tápellátás)
- Vannak saját tápellátással rendelkező hub-ok
  - Minden kimenő port-jukra biztosíthatják a max. 500mA keretet
- USB perifériák (saját táp nélkül):
  - A konfigurációs fázisban max. 100mA-t vehetnek fel
  - A konfigurációs fázisban közlik, mennyi áramot igényelnek
  - Ha nincs ennyi, nem léphetnek be a rendszerbe

# Kábelek, csatlakozók

- 4 vezeték
  - Táp: 0V, 5V
  - Jelátvitel: D+, D–, differenciális, soros átvitel
- Kábelek
  - LS: egyszeresen árnyékolt (vékony, hajlékony, olcsó)
  - FS: kétszeresen árnyékolt, az adatvezetékek megcsavarva
- Csatlakozók
  - Type A: periféria a hubhoz
  - Type B: hub a perifériához (ha külön van)




Type A




Type B

# Konfigurálás

- Rendszerindításkor, új eszköz csatlakozásakor
  - A hub érzékeli, hogy új eszköz csatlakozott
    (a periféria magasba húzza FS esetben a D+-t, LS esetben a D– -t)
  - Az USB driver időnként végigkérdezi a hub-okat, jött-e új periféria az egyik portra. Ha jött, konfigurálja őket:

    1) Szól a hub-nak, hogy reset-elje a portot
    Reset hatására a periféria a 0-s címre hallgat

    2) Kiolvassa a periféria eszközleíró táblázatait
    (0-s címre küldött control tranzakciókkal)
    - Benne vannak a gyártó, típus, eszköz jellege, paraméterei, stb.

    3) Egyedi címet ad a perifériának
    (0-s címre küldött control tranzakcióval)

    4) Ellenőrzi, beléphet-e:
    - Kielégíthető-e a tápellátási igénye
    - Izokron és interrupt adatátviteli igénye kielégíthető-e

    5) Ha minden rendben, az új eszköz beléphet az adatforgalomba

# USB 2.0

# Az USB 2.0

- 2000-ben jelent meg
- Legfontosabb újdonság:
  - Nagy sebességű adatátvitel: 480 Mb/s (high speed, HS)
- Ami nem változott:
  - LS és FS eszközök változatlanul működnek
  - Kábelek, csatlakozók maradtak (kiegészültek)
  - Ugyanazok az adatátviteli módok
  - Tápellátás, konfigurálás ugyanúgy
- Ami változott:
  - Keretidő: 1ms-os keretek helyett 125μs-os **mikró keretek**
  - HS sebességgel 1500 helyett 7500 bájt rakomány / mikró keret
  - Egy vadonatúj **forgalomszabályozás**i eljárás

# Nagy sebességű tranzakciók

- **Izokron adatátviteli mód:**
  - 1 helyett 3 izokron tranzakciót enged meg ugyanazon perifériához
  - A max. 1023 bájt helyett max. 1024 vihető át
- **Bulk adatátviteli mód:**
  - Tranzakciónként 512 bájt lett (fix)
  - Új forgalomszabályozás!
    - USB 1.1:
      - Root hub nem tudja, hogy az eszköz kész-e a fogadásra
      - Leküldte az adatot, és a negatív nyugtából tudta meg
      - Feleslegesen leküldött adat foglalja a kereteket!
    - USB 2.0:
      - PING tranzakció
      - Megkérdezi a perifériát, mehet-e az adat
      - Periféria nem csak negatív vagy pozitív nyugtát küldhet
      - Új nyugtatípus: „vettem, de egy kis időt kérek a következőig”

# Nagy sebességű tranzakciók

- **Control tranzakciók:**
  - Rögzített 64 bájtos méret
  - PING forgalomszabályozás
- **Interrupt tranzakciók**
  - Adatmennyiség max. 1024-re nőtt (8, ill. 64 bájtról)
  - Egy mikró-keretben 3 intrerrupt tranzakció is mehet ugyanazon perifériához

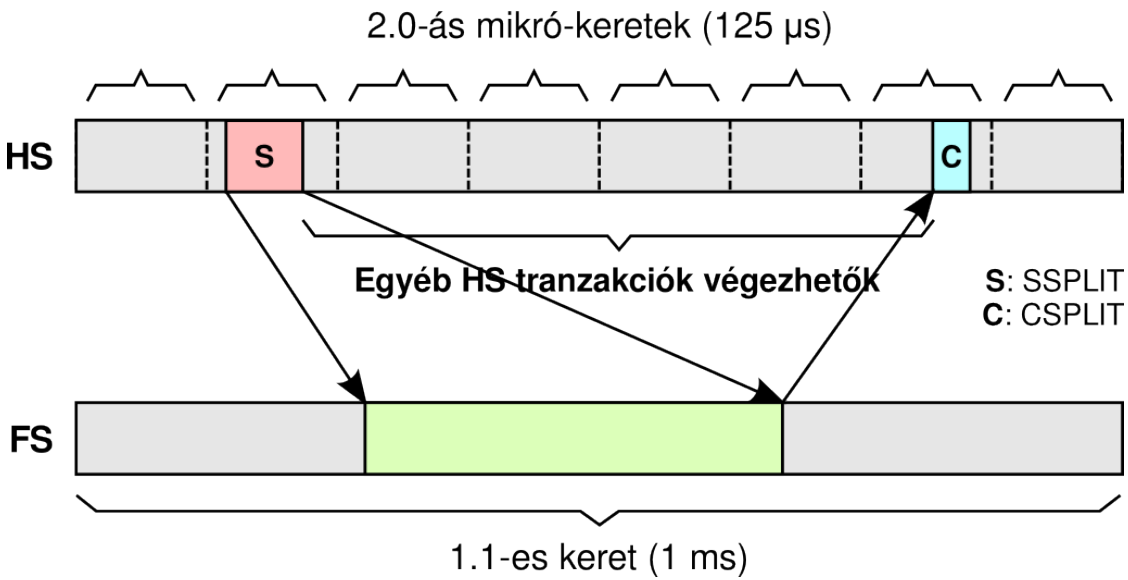
# LS és FS tranzakciók USB 2.0-ban

- USB 2-ben más a keretidő és a bitsebesség

  → LS és FS eszközök alapból nem tudnak részt venni benne!
- A kompatibilitást a hub-ok biztosítják:
  - A rájuk kötött LS és FS eszközöknek azt a látszatot keltik, hogy USB 1.1-re csatlakoznak
- A megoldás részletei:
  - A root hub nagy sebességgel leküldi a hub-nak a tranzakciókat
  - A hub szép lassan LS/FS sebességgel lebeszéli az eszközzel
  - Amikor a root hub szerint már vége kellett, hogy legyen, újra ránéz a végeredményre
  - **Osztott tranzakció**!

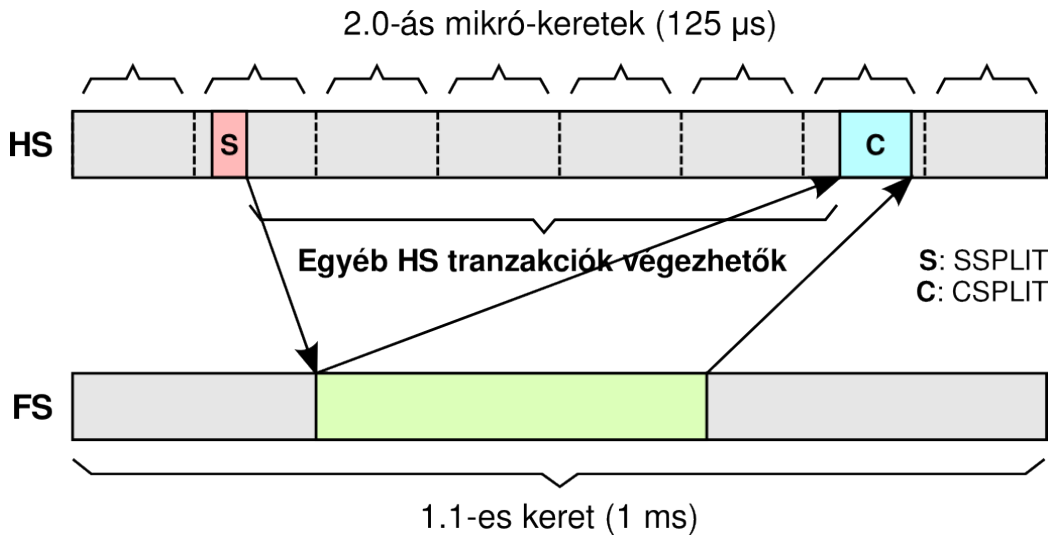
# Osztott kiviteli tranzakció

- A root hub egy SSPLIT tranzakcióban elküldi:
  - A célperiféria címét
  - A neki szánt adatot
- Később a root hub visszanéz: CSPLIT tranzakció
  - Bekéri a hub-tól a nyugtát
- A CSPLIT megérkezéséig a nyugtát a hub tárolja



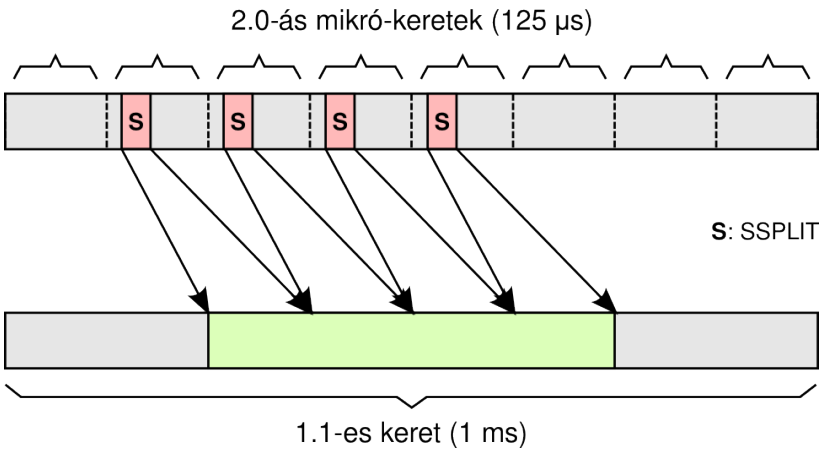
# Osztott bemeneti tranzakció

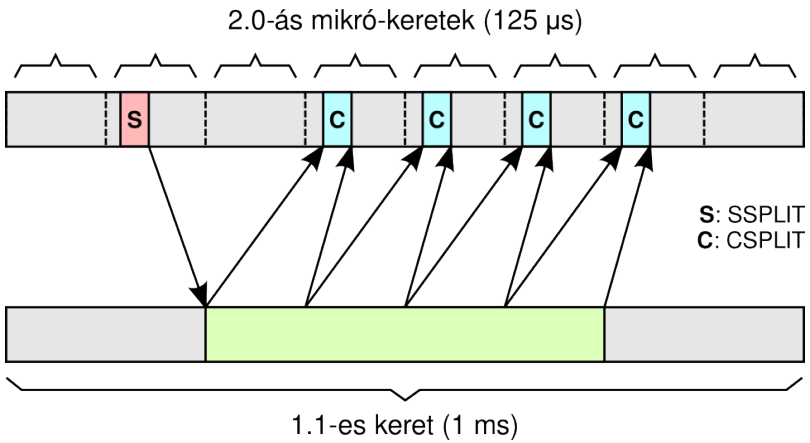
- A root hub egy SSPLIT tranzakcióban elküldi:
  - A célperiféria címét
  - A kérést
- Később a root hub visszanéz: CSPLIT tranzakció
  - Bekéri a hub-tól a periféria által küldött adatokat
  - És a nyugtát
- A CSPLIT megérkezéséig az adatokat és a nyugtát a hub tárolja



# Osztott izokron tranzakciók

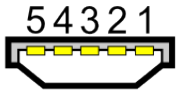
- Cél: egyenletes, állandó adatátviteli ráta
- Az FS izokron tranzakciókat feldarabolva küldi az FS eszköz hub-jához
  - Minden mikró keretben egy darab
  - Így egyenletesebb lesz az átviteli sebesség
  - Így egyenletesebb lesz a mikró keretek terhelése



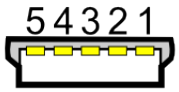



# Új csatlakozók

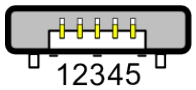
- Mobil eszközök igényeinek megfelelően:
  - Mini
  - Mikró
- Mobil eszközök sajátossága:
  - Hoszt és periféria is lehet
  - Kár lenne ehhez 2 csatlakozó
- Kombinált foglalatok:
  - Van aljzat, amibe az „A” és „B” csatlakozó is belemegy
  - „A”-t beledugva host-ként viselkedik
  - „B”-t beledugva egyszerű periféria
- Lábak:
  - 0V, +5V
  - D+, D–
  - „Szerep” láb: földelve host, nyitva periféria



Mini-A



Mini-B



Micro-A

Micro-B

# USB 3.0

# Az USB 3.0

- 2008-ben jelent meg
- Legfontosabb újdonság:
  - Szuper sebességű adatátvitel: 3.8 Gb/s (super speed, SS)
- Ami nem változott:
  - Fa topológia, hub-okkal a csomópontokban
  - A 4 adatátviteli mód megmaradt
  - Ugyanúgy kell az eszközöket konfigurálni
  - Továbbra is a root hub az egyetlen master

  → Szoftver szinten nincs változás
- Változások:
  - Fizikai szinten szinte minden. Teljesen új rendszer.
  - Kompatibilitás?
    - Megoldás: az USB 3.0 tulajdonképpen két külön busz egymás mellett
      - Egy USB 2.0-ás busz az LS, FS és HS tranzakcióknak
      - Egy külön busz az SS tranzakcióknak

# Szupergyors adatátvitel

- Változások részletesen (SS tranzakciók)
  - **Megszűnt az üzenetszórás**
    - Forgalomirányítással érnek célba az adatok
    - A hub-ok csak arra továbbítják a tranzakciót, amerre a címzett van
  - **Store and forward csomagtovábbítás**
    - A hub megvárja, amíg teljesen megjön hozzá a tranzakció
    - Csak ezután küldi tovább
  - **Egy helyett két érpáron zajlik a kommunikáció**
    - Oda-vissza irányba párhuzamosan
    - A PCI Express-nél látott módon (bitkeverés, 8-ból 10-be kódolás, stb.)
  - **Jobb forgalomszabályozás**
    - Nem kell mindent egyenként nyugtázni (végén közös nyugta)
  - **Jobb energia menedzsment**
    - Finomabb felbontásban szabályozható a perifériák ébrenléte
    - Mivel nincs üzenetszórás, a perifériák többet alhatnak
  - **Áramfelvétel:**
    - A port-onkénti minimum 100 mA-ről 150 mA-re nőtt
    - A maximum 500 mA-ről 900 mA-re nőtt

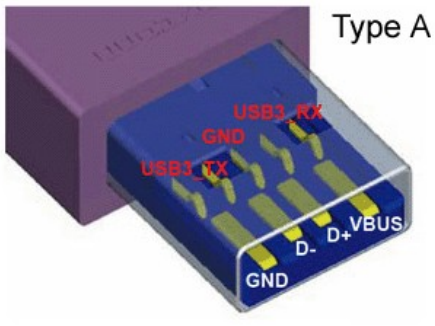
# A tranzakciók célba juttatása

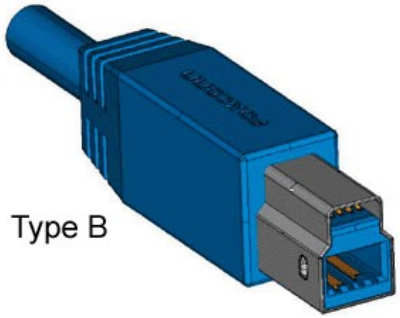
- A tranzakciók útvonalát a root hub határozza meg
  - Ismeri az USB fát
  - Minden tranzakciót kiegészít egy listával
    - Benne van, hogy melyik útba eső hub merre továbbítsa
- Részletek:
  - Az USB fa max. mélysége: 6

    → Egy tranzakció max. 5 hub-on haladhat át
  - Egy USB hub port-jainak max. száma: 15
  - A tranzakciókhoz egy **route string**-et ragaszt
    - Ez 5 db 4 bites szám
    - Az i. köztes hub kiolvassa az i. 4 bites számot
    - Ha 0, akkor maga a hub a tranzakció célpontja
    - Ha nem, akkor a 4 bites szám határozza meg a portot, amerre küldenie kell

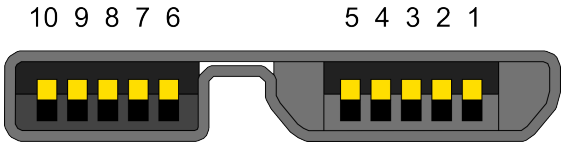
# Új kábelek és csatlakozók

- Új kábel kell: benne fut az USB 2.0 csavart érpárja, és az USB 3.0 két csavart érpárja is → jóval vastagabb (drágább)
- Csatlakozók:
  - 2 részre bontották: USB 2.0 és USB 3.0 részre



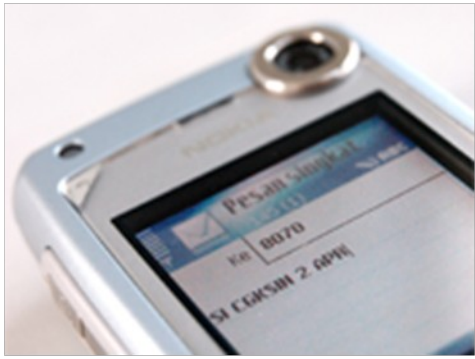




- Érdekesség: Type „B”-ben van még egy táp
  - Ezt a periféria táplálja
  - ??? nem pont őt kell táplálni ???
  - Ez teszi lehetővé a vezeték nélküli USB átvitel használatát!
- Micro is van:



# Összegzés

- Mit **ne** tanuljunk meg
  - A számértékeket (mi hány bájt)
  - A kábelek, csatlakozók lábkiosztását
  - Mi hány amper és mi hány volt
- Mit tanuljunk meg feltétlenül?
  - Az adatátviteli módokat
    - Hogy hogyan igazodik egy csatlakozófelület a sokféle periféria sokféle igényéhez
  - Hogy hogyan tud működni egy busz egyetlen master-rel
  - Hogy hogyan osztozkodnak az eszközök a busz sávszélességén
  - Hogy az USB-ben nincs igazi interrupt (csak polling)
  - Hogy hogyan oldották meg, hogy az USB 2.0 kompatibilis maradjon a régi eszközökkel
  - Hogy mi kellett ahhoz, hogy az USB 3.0 olyan gyors lehessen
  - Hogy hogyan oldották meg, hogy az USB 3.0 kompatibilis maradjon a régi eszközökkel
  - Hogy miként oldották meg az eszközök konfigurálását



# Számítógép Architektúrák

*1. ZH megoldása*

Horváth Gábor
docens
BME Híradástechnikai Tanszék
ghorvath@hit.bme.hu

2012. október 12.
Budapest

# „B” csoport

# Tesztek

- A mágneses adathordozó egy blokkjának állapota a következő: ←←←→→→←→←

  A váltáshoz 1-es bitet, a váltás hiányához 0-ás bitet rendelünk.
- Milyen bitsorozat tartozik a megadott mágneses mintázathoz:
  - **00100111**
- Változtassa meg a harmadik bitet (1-től számolva)!
- Milyen mintázatot kapunk?
  - ←←←←←←→←→

# Tesztek

- Sorolja fel, milyen összetevőkből áll egy szektor olvasási kérés teljes kiszolgálási ideje!
- Melyeket lehet ezek közül átlapolni?
  - **Parancsfeldolgozási késleltetés**
  - **Seek idő**
  - **Forgási idő**
  - **Adatleolvasási idő**
  - **Adatok átküldése az interfészen**
  - **Az utóbbi kettő átlapolható**

# Tesztek

- Ha egy merevlemez minden sávjában ugyanannyi, egyforma méretű szektor található, akkor
  - A külső sávokban nagyobb az adatsűrűség
  - A belső sávokban nagyobb az adatsűrűség
  - Minden sávban egyforma az adatsűrűség

# Tesztek

- Ha egy merevlemez minden sávjában ugyanannyi, egyforma méretű szektor található, akkor
  - A külső sávokban nagyobb az adatsűrűség
  - **A belső sávokban nagyobb az adatsűrűség**
  - Minden sávban egyforma az adatsűrűség

# Tesztek

- Mely 3 koordinátával azonosít egy szektort fizikailag a merevlemez?
  - **Cilinder**
  - **Fej**
  - **Szektorszám**

# Tesztek

- Mi a PCI és a PCI Express viszonya kompatibilitás szempontjából?
  - Szoftver szinten kompatibilisek
  - Hardver szinten kompatibilisek
  - Egyáltalán nem kompatibilisek
  - Minden szempontból kompatibilisek

# Tesztek

- Mi a PCI és a PCI Express viszonya kompatibilitás szempontjából?
  - **Szoftver szinten kompatibilisek**
  - Hardver szinten kompatibilisek
  - Egyáltalán nem kompatibilisek
  - Minden szempontból kompatibilisek

# Tesztek

- A PCI ablakok milyen címtartományra vonatkoznak?
  - Csak a memória címtartományára
  - Csak a periféria címtartományra
  - Mindkettőre vonatkozhatnak, az op. rendszer határozza meg a rendszer indításakor
  - Mindkettőre vonatkozhatnak, a periféria határozza meg, hogy melyik ablak melyikre vonatkozzon

# Tesztek

- A PCI ablakok milyen címtartományra vonatkoznak?
  - Csak a memória címtartományára
  - Csak a periféria címtartományra
  - Mindkettőre vonatkozhatnak, az op. rendszer határozza meg a rendszer indításakor
  - **Mindkettőre vonatkozhatnak, a periféria határozza meg, hogy melyik ablak melyikre vonatkozzon**

# Tesztek

- Milyen arbitrációt használ a PCI?
  - Semmilyet, mert csak 1 master lehet a buszon
  - Daisy chain alapú arbitrációt
  - Párhuzamos arbitrációt

# Tesztek

- Milyen arbitrációt használ a PCI?
  - Semmilyet, mert csak 1 master lehet a buszon
  - Daisy chain alapú arbitrációt
  - **Párhuzamos arbitrációt**

# Tesztek

- Mit jelent a PWM kimenet az Arduino-n?
  - Impulzusszélesség-modulációt
  - Impulzusfrekvencia-modulációt
  - Impulzusszám-modulációt

# Tesztek

- Mit jelent a PWM kimenet az Arduino-n?
  - **Impulzusszélesség-modulációt**
  - Impulzusfrekvencia-modulációt
  - Impulzusszám-modulációt

# Tesztek

- Mely állítások igazak a Daisy chain alapú és a PIC alapú interrupt kezelésre?

| | |
|---|---|
| Daisy chain alapú interrupt kezelés | Körbenforgó interrupt kiszolgálás megvalósítható |
| | Az eszközök prioritása nem változtatható |
| PIC alapú interrupt kezelés | Tetszőlegesen bővíthető |
| | Nem működik, ha egyidejűleg több eszköz is jelez interruptot |

# Tesztek

- Mely állítások igazak a Daisy chain alapú és a PIC alapú interrupt kezelésre?

Daisy chain alapú interrupt kezelés

PIC alapú interrupt kezelés

Körbenforgó interrupt kiszolgálás megvalósítható

Az eszközök prioritása nem változtatható

Tetszőlegesen bővíthető

Nem működik, ha egyidejűleg több eszköz is jelez interruptot

# Tesztek

- Miért jelet gondot, ha a polling intervallum
  - Túl kicsi:
    - **Nagy terhelés a processzornak**
  - Túl nagy:
    - **Lemaradunk egy eseményről (adatvesztés)**

# Tesztek

- Jelölje meg az igaz állításokat!
  - A DMA vezérlőt a periféria programozza fel
  - A DMA vezérlő alkalmazása csökkenti a CPU interrupt terhelését
  - Több DMA vezérlő is lehet egy buszon
  - A DMA vezérlő minden adategység sikeres átvitelét interrupt-tal jelzi a processzornak

# Tesztek

- Jelölje meg az igaz állításokat!
  - A DMA vezérlőt a periféria programozza fel
  - **A DMA vezérlő alkalmazása csökkenti a CPU interrupt terhelését**
  - **Több DMA vezérlő is lehet egy buszon**
  - A DMA vezérlő minden adategység sikeres átvitelét interrupt-tal jelzi a processzornak

# Tesztek

- Forgalomszabályozás hiányában mikor és milyen hibák fordulhatnak elő?

| | |
|---|---|
| Adat egymásrafutás | Küldő gyorsabb, mint a fogadó |
| Adathiány | Fogadó gyorsabb, mint a küldő |

# Tesztek

- Forgalomszabályozás hiányában mikor és milyen hibák fordulhatnak elő?

Adat egymásrafutás → Küldő gyorsabb, mint a fogadó

Adathiány → Fogadó gyorsabb, mint a küldő

# Tesztek

- Tegye ki a "<", a ">", vagy az "=" relációs jelet a RISC és CISC viszonyával kapcsolatban:
  - Utasítások száma:
    - RISC < CISC
  - Regiszterek száma:
    - RISC > CISC
  - Címzési módok száma:
    - RISC < CISC
  - Program mérete:
    - RISC > CISC

# Tesztek

- Egy utasításkészletet ortogonálisnak nevezünk, ha
  - Ha egy adott célra csak egy utasítást tartalmaz
  - Ha egy adott célra sokféle, kényelmesen használható utasítás is rendelkezésre áll
  - Ha minden címzést használó utasítás minden címzési módot használhat
  - Ha minden címzést használó utasítás csak egyetlen címzési módot használhat

# Tesztek

- Egy utasításkészletet ortogonálisnak nevezünk, ha
  - Ha egy adott célra csak egy utasítást tartalmaz
  - Ha egy adott célra sokféle, kényelmesen használható utasítás is rendelkezésre áll
  - **Ha minden címzést használó utasítás minden címzési módot használhat**
  - Ha minden címzést használó utasítás csak egyetlen címzési módot használhat

# Tesztek

- Adat és utasítás egyidejűleg (párhuzamosan) olvasható ki
  - A Neumann architektúrában
  - A Harvard architektúrában
  - A módosított Harvard architektúrában

# Tesztek

- Adat és utasítás egyidejűleg (párhuzamosan) olvasható ki
  - A Neumann architektúrában
  - **A Harvard architektúrában**
  - **A módosított Harvard architektúrában**

# Tesztek

- Mely tulajdonság mely architektúrá(k)ra jellemző?

| | |
|---|---|
| Neumann architektúra | Az adatok és az utasítások külön memóriában vannak |
| Harvard architektúra | Az adatok a programból módosíthatók |
| Módosított Harvard architektúra | Önmódosító program írása lehetséges |
| | Az op. rendszer bármikor betölthet a diszkről egy programot |

# Tesztek

- Mely tulajdonság mely architektúrá(k)ra jellemző?

Neumann architektúra

Harvard architektúra

Módosított Harvard architektúra

Az adatok és az utasítások külön memóriában vannak

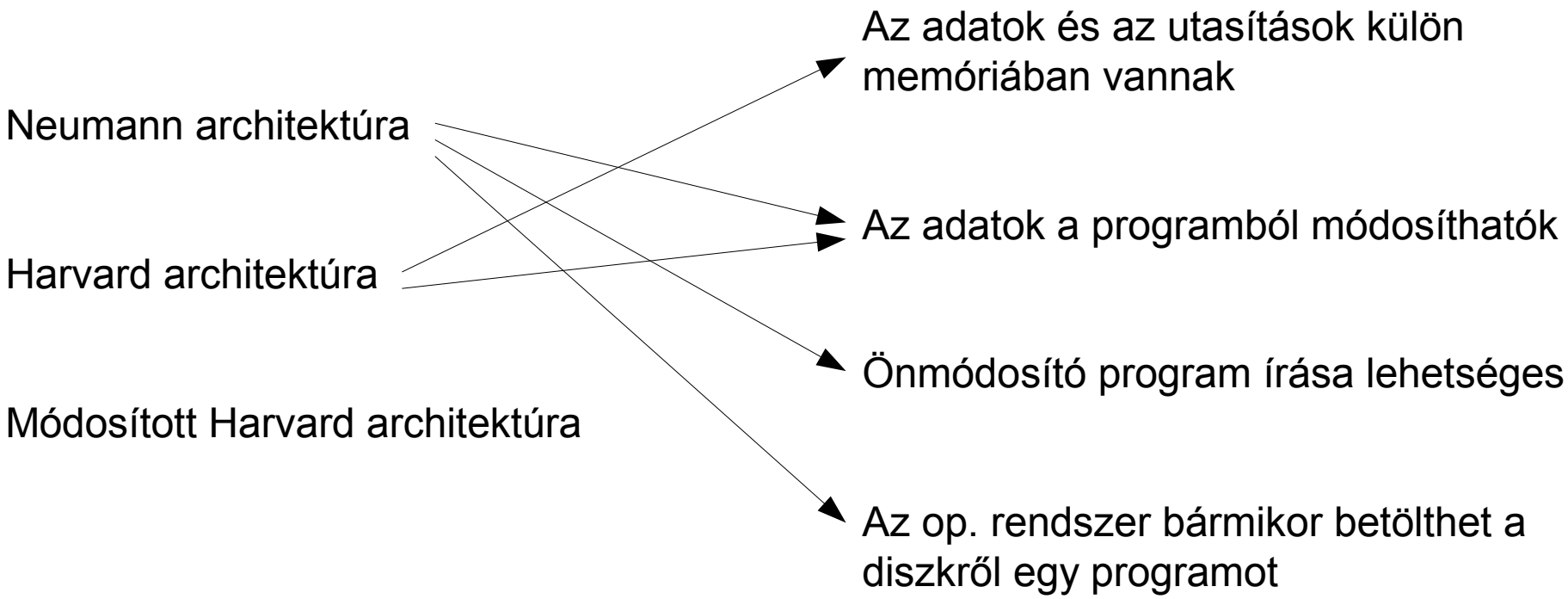
Az adatok a programból módosíthatók

Önmódosító program írása lehetséges

Az op. rendszer bármikor betölthet a diszkről egy programot

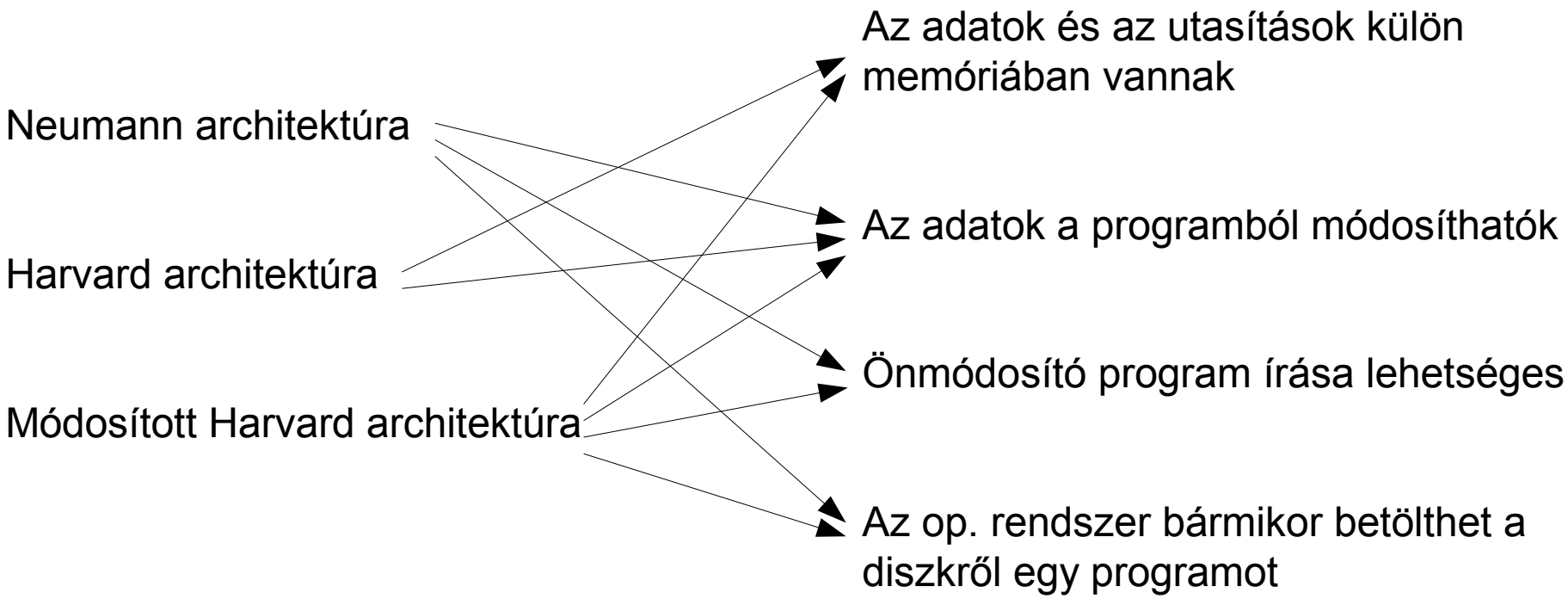
# Tesztek

- Mely tulajdonság mely architektúrá(k)ra jellemző?



# Tesztek

- Mely tulajdonság mely architektúrá(k)ra jellemző?



# Tesztek

- Jelölje be az igaz állítást eredményező lehetőségeket!
- Az igényvezérelt architektúrában az igénygráf irányított élei megadják,
  - hogy egy operandus kiszámolásához mely műveletet kell elvégezni
  - hogy mi a soron következő végrehajtható utasítás
  - hogy egy művelet eredménye mely más műveletek operandusa

# Tesztek

- Jelölje be az igaz állítást eredményező lehetőségeket!
- Az igényvezérelt architektúrában az igénygráf irányított élei megadják,
  - **hogy egy operandus kiszámolásához mely műveletet kell elvégezni**
  - hogy mi a soron következő végrehajtható utasítás
  - hogy egy művelet eredménye mely más műveletek operandusa

# Tesztek

- Milyen információfeldolgozási modell szerint működnek az alábbiak?

Neumann elvű számítógép

Vezérlésáramlásos

Harvard architektúrájú számítógép

Adatáramlásos

Táblázatkezelő program

Igényvezérelt

Funkcionális programozási nyelv

# Tesztek

- Milyen információfeldolgozási modell szerint működnek az alábbiak?

Neumann elvű számítógép → Vezérlésáramlásos

Harvard architektúrájú számítógép → Vezérlésáramlásos

Táblázatkezelő program → Adatáramlásos

Funkcionális programozási nyelv → Igényvezérelt

# Tesztek

- Milyen formális eszközzel írunk le egy programot
  - A vezérlésáramlásos modellben:
    - **Folyamatábra**
  - Az adatáramlásos modellben:
    - **Precedenciagráf / függőségi gráf**

# 1. Számpélda

- Processzorunk órajel frekvenciája 1GHz. A számítógéphez egy 100 Mbit/s segességű hálózati interfészt kötünk, melyen 1500 bájtos (=12000 bit) csomagok közlekednek. A periféria állapotának lekérdezése (mely nem tartalmazza a hálózati csomag tartalmát) 600 órajelet igényel. Az interrupt feldolgozási idő 400 órajel. Ha jött csomag, annak átvitele 5000 órajelet vesz igénybe. Jelenleg épp egy 24 Mbit/s sebességű letöltés van folyamatban a hálózaton.

# 1. Számpélda

- Processzorunk órajel frekvenciája 1GHz. A számítógéphez egy 100 Mbit/s segességű hálózati interfészt kötünk, melyen 1500 bájtos (=12000 bit) csomagok közlekednek. A periféria állapotának lekérdezése (mely nem tartalmazza a hálózati csomag tartalmát) 600 órajelet igényel. Az interrupt feldolgozási idő 400 órajel. Ha jött csomag, annak átvitele 5000 órajelet vesz igénybe. Jelenleg épp egy 24 Mbit/s sebességű letöltés van folyamatban a hálózaton.
- Hány ms-onként kell lekérdezni a perifériát, hogy biztosan ne maradjunk le egy csomag érkezéséről?
  - Pont annyi időnként, amennyi időnként ***elméletileg*** jöhet csomag:
  - (12000 bit / csomag) / ($100*10^6$ bit / s)
    = $0.12*10^{-3}$ s / csomag = **0.12 ms**

# 1. Számpélda

- Processzorunk órajel frekvenciája 1GHz. A számítógéphez egy 100 Mbit/s segességű hálózati interfészt kötünk, melyen 1500 bájtos (=12000 bit) csomagok közlekednek. A periféria állapotának lekérdezése (mely nem tartalmazza a hálózati csomag tartalmát) 600 órajelet igényel. Az interrupt feldolgozási idő 400 órajel. Ha jött csomag, annak átvitele 5000 órajelet vesz igénybe. Jelenleg épp egy 24 Mbit/s sebességű letöltés van folyamatban a hálózaton.
- Mekkora terhelést jelent a periféria kezelése a processzor számára, ha a csomagok érkezését polling-al figyeljük?
  - $0.12*10^{-3}$ s / lekérdezés → 1 / ($0.12*10^{-3}$) lekérdezés / s
    - Minden lekérdezés 600 órajelig tart:
      1 / ($0.12*10^{-3}$) lekérdezés / s * 600 órajel / lek. = $5*10^{6}$ órajel / s
  - Ha ténylegesen van csomag, azt is el kell venni!
    - 24 Mbit/s mellett milyen gyakran jön csomag:
      ($24*10^{6}$ bit / s) / (12000 bit / csomag) = 2000 csomag / s
    - 2000 csomag / s * 5000 órajel / csomag = $10*10^{6}$ órajel / s
  - Teljes terhelés:
    - 1GHz = $10^{9}$ órajel / s
    - ($15*10^{6}$ órajel / s) / ($10^{9}$ órajel / s) = 0.015 = **1.5%**

# 1. Számpélda

- Processzorunk órajel frekvenciája 1GHz. A számítógéphez egy 100 Mbit/s segességű hálózati interfészt kötünk, melyen 1500 bájtos (=12000 bit) csomagok közlekednek. A periféria állapotának lekérdezése (mely nem tartalmazza a hálózati csomag tartalmát) 600 órajelet igényel. Az interrupt feldolgozási idő 400 órajel. Ha jött csomag, annak átvitele 5000 órajelet vesz igénybe. Jelenleg épp egy 24 Mbit/s sebességű letöltés van folyamatban a hálózaton.
- Mekkora terhelést jelent a processzor számára, ha interrupt-ot használunk a csomag érkezések jelzésére?
  - Előbb kiszámoltuk, hogy 24 Mbit/s mellett milyen gyakran jön ténylegesen csomag
  - Minden csomag 1 interrupt, és el is kell venni a perifériától:
    - $(24*10^6$ bit / s) / (12000 bit / csomag) = 2000 csomag / s
    - 2000 csomag / s * (400 órajel / csomag + 5000 órajel / csomag) = $10.8*10^6$ órajel / s
  - Teljes terhelés:
    - 1GHz = $10^9$ órajel / s
    - ($10.8*10^6$ órajel / s) / ($10^9$ órajel / s) = 0.0108 = **1.08%**

# 2. Számpélda

- Egy merevlemez 3 db kétoldalas lemezt tartalmaz, melyek mindegyikén 20000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs. Az átlagos seek idő 4 ms. Mérésekkel megállapítottuk, hogy az egy szektorra vonatkozó olvasási kérések kiszolgálási ideje 10 ms.

# 2. Számpélda

- Egy merevlemez 3 db kétoldalas lemezt tartalmaz, melyek mindegyikén 20000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs. Az átlagos seek idő 4 ms. Mérésekkel megállapítottuk, hogy az egy szektorra vonatkozó olvasási kérések kiszolgálási ideje 10 ms.
- Ha a parancsfeldolgozási késleltetéstől és az interfészen való átviteli időtől eltekintünk, milyen gyorsan forog a lemez? (fordulat/perc-ben megadva)
  - Olvasási kérések (az eltekintések mellett): seek idő + forgási késleltetés
  - → forgási késleltetés = 10 ms - 4 ms = 6 ms
  - Egy véletlen igény forgási késleltetése a teljes körülfordulási idő fele → körülfordulási idő: 12 ms
  - (60000 ms / perc) / (12 ms / fordulat) = **5000 fordulat / perc**

# 2. Számpélda

- Egy merevlemez 3 db kétoldalas lemezt tartalmaz, melyek mindegyikén 20000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs. Az átlagos seek idő 4 ms. Mérésekkel megállapítottuk, hogy az egy szektorra vonatkozó olvasási kérések kiszolgálási ideje 10 ms.
- Adjuk meg CHS koordináta rendszerben a lemez kapacitását!
  - **Cilinderek: 20000** (megegyezik a sávok számával)
  - **Fejek: 6** (mivel ennyi adathordozó réteg van)
  - **Szektorok: 1000** (sávonként értendő)

# 2. Számpélda

- Egy merevlemez 3 db kétoldalas lemezt tartalmaz, melyek mindegyikén 20000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs. Az átlagos seek idő 4 ms. Mérésekkel megállapítottuk, hogy az egy szektorra vonatkozó olvasási kérések kiszolgálási ideje 10 ms.
- A lemez kapacitása bájtokban mérve:
  - Sávok száma összesen: 20000 * 6
  - Szektorok száma: sávok száma * szektorok/sáv = 20000*6*1000
  - Merevlemez kapacitása: szektorok száma * egy szektor mérete = 20000 * 6 * 1000 * 500 = **$6 * 10^{10}$**

# 2. Számpélda

- Egy merevlemez 3 db kétoldalas lemezt tartalmaz, melyek mindegyikén 20000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs. Az átlagos seek idő 4 ms. Mérésekkel megállapítottuk, hogy az egy szektorra vonatkozó olvasási kérések kiszolgálási ideje 10 ms.
- Meddig tart egy szektor leolvasása az adathordozóról? (Ha már ott van a fej)
  - Pont addig, amíg egy szektornyit elfordul a lemez a fej alatt
  - Seek, forgási idő nem számít! A feladat szerint már ott a fej!
  - 12 ms / fordulat → 12 ms / 1000 szektor = **0.012 ms / szektor**

# 2. Számpélda

- Egy merevlemez 3 db kétoldalas lemezt tartalmaz, melyek mindegyikén 20000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs. Az átlagos seek idő 4 ms. Mérésekkel megállapítottuk, hogy az egy szektorra vonatkozó olvasási kérések kiszolgálási ideje 10 ms.
- Ha a parancsfeldolgozási idő 0.1 ms, az adatátviteli interfész sebessége pedig $50*10^6$ bájt/s, akkor átlagosan mennyi ideig tart egy 2000 bájtos kérés teljes kiszolgálása?
  - Parancsfeldolgozás: 0.1 ms
  - Seek: 4 ms
  - Forgási dő: 6 ms (véletlen pozícióba átlag egy fél fordulat kell)
  - Leolvasási idő: 0.012 ms / szektor, de 2000 bájt az 4 szektor, vagyis 4 * 0.012 ms = 0.048 ms
  - Adatátviteli idő: átlapolódás miatt csak az utolsó szektor átviteli ideje kell: (500 bájt / szektor) / ($50*10^6$ bájt / s) = $10^{-5}$ s / szektor = 0.01 ms
  - Összesen: 0.1 + 4 + 6 + 0.048 + 0.01 = **10.158 ms**

# „A” csoport

# Tesztek

- Mikor hajtható végre egy művelet a tanult 3 információfeldolgozási modellben?

| | |
|---|---|
| Vezérlésáramlásos modell | Amikor szükség van az eredményére |
| Adatáramlásos modell | Amikor a vezérlő token odaér |
| Igényvezérelt modell | Amikor minden operandusa rendelkezésre áll |

# Tesztek

- Mikor hajtható végre egy művelet a tanult 3 információfeldolgozási modellben?

| | |
|---|---|
| Vezérlésáramlásos modell | Amikor szükség van az eredményére |
| Adatáramlásos modell | Amikor a vezérlő token odaér |
| Igényvezérelt modell | Amikor minden operandusa rendelkezésre áll |

# Tesztek

- Hogyan kell jelezni a program leírásában a párhuzamosítható részeket
  - A vezérlésáramlásos modellben:
    - **Fork/Join primitívekkel**
  - Az adatáramlásos modellben:
    - **Sehogy**
  - Az igényvezérelt modellben:
    - **Sehogy**

# Tesztek

- Jelölje be az igaz állítást eredményező lehetőségeket!
- A precedenciagráf irányított élei megadják,
  - hogy egy operandus kiszámolásához mely műveletet kell elvégezni
  - hogy mi a soron következő végrehajtható utasítás
  - hogy egy művelet eredménye mely más műveletek operandusa

# Tesztek

- Jelölje be az igaz állítást eredményező lehetőségeket!
- A precedenciagráf irányított élei megadják,
  - hogy egy operandus kiszámolásához mely műveletet kell elvégezni
  - hogy mi a soron következő végrehajtható utasítás
  - **hogy egy művelet eredménye mely más műveletek operandusa**

# Tesztek

- Melyek a Neumann architektúra főbb komponensei?
  - Processzor
  - Cache
  - Perifériák
  - Memória
  - Rendszerbusz
  - Háttértár

# Tesztek

- Melyek a Neumann architektúra főbb komponensei?
  - **Processzor**
  - Cache
  - **Perifériák**
  - **Memória**
  - Rendszerbusz
  - Háttértár

# Tesztek

- Honnan tudja egy Neumann elvű számítógép processzora, hogy egy memóriaeléréskor a meghivatkozott memóriabeli objektumot milyen típusúnak tekintse?
  - Onnan, hogy a memóriában az objektumok típusa is le van tárolva
  - Az objektumok típusa nincs ugyan letárolva, de tudja, milyen típusú, hiszen ő tette bele
  - Onnan, hogy az utasítás, amivel a program a memóriához nyúl, milyen típusú adatra vonatkozik

# Tesztek

- Honnan tudja egy Neumann elvű számítógép processzora, hogy egy memóriaeléréskor a meghivatkozott memóriabeli objektumot milyen típusúnak tekintse?
  - Onnan, hogy a memóriában az objektumok típusa is le van tárolva
  - Az objektumok típusa nincs ugyan letárolva, de tudja, milyen típusú, hiszen ő tette bele
  - **Onnan, hogy az utasítás, amivel a program a memóriához nyúl, milyen típusú adatra vonatkozik**

# Tesztek

- Az összetartozó párok összekötésével jelezze a fix és a változó utasításkódolás főbb tulajdonságait!

| | |
|---|---|
| Fix hosszúságú utasításkódolás | Helytakarékosabb, rövidebb programhoz vezet |
| Változó hosszúságú utasításkódolás | A processzor gyorsabban tudja dekódolni |

# Tesztek

- Az összetartozó párok összekötésével jelezze a fix és a változó utasításkódolás főbb tulajdonságait!

Fix hosszúságú utasításkódolás

Változó hosszúságú utasításkódolás

Helytakarékosabb, rövidebb programhoz vezet

A processzor gyorsabban tudja dekódolni

# Tesztek

- A hexadecimális 0x0A0B0C0D számot adja meg
  - Little endian bájtsorrenddel tárolva:
    - **0D0C0B0A**
  - Big endian bájtsorrenddel tárolva:
    - **0A0B0C0D**

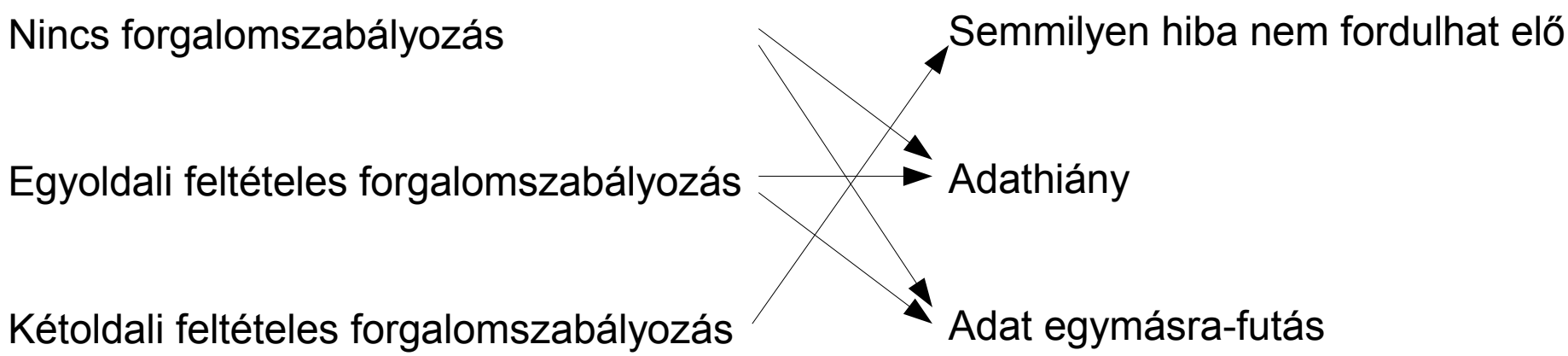
# Tesztek

- Melyik forgalomszabályozás mellett milyen hibák fordulhatnak elő az adatátvitel során:

| | |
|---|---|
| Nincs forgalomszabályozás | Semmilyen hiba nem fordulhat elő |
| Egyoldali feltételes forgalomszabályozás | Adathiány |
| Kétoldali feltételes forgalomszabályozás | Adat egymásra-futás |

# Tesztek

- Melyik forgalomszabályozás mellett milyen hibák fordulhatnak elő az adatátvitel során:



# Tesztek

- Tegyük fel, hogy a CPU támogatja az interruptok használatát, és ez engedélyezve is van. Mely állítások igazak?
  - A lassú perifériákat célszerű polling-gal kezelni, mert úgy kisebb a processzor terhelése
  - A lassú perifériákat célszerű polling-gal kezelni, mert úgy gyorsabban értesülünk a periféria megváltozott állapotáról
  - A gyors perifériát célszerű interrupt-tal kezelni, mert úgy kisebb a processzor terhelése

# Tesztek

- Tegyük fel, hogy a CPU támogatja az interruptok használatát, és ez engedélyezve is van. Mely állítások igazak?
  - A lassú perifériákat célszerű polling-gal kezelni, mert úgy kisebb a processzor terhelése
  - A lassú perifériákat célszerű polling-gal kezelni, mert úgy gyorsabban értesülünk a periféria megváltozott állapotáról
  - **A gyors perifériát célszerű interrupt-tal kezelni, mert úgy kisebb a processzor terhelése**

# Tesztek

- Van egy bemeneti perifériánk, és egy processzorunk. Milyen forgalomszabályozást célszerű használni az adatátviteli hibák elkerülésére, ha
  - A periféria sebessége befolyásolható:
    - **Kétoldali feltételes forgalomszabályozás**
  - A periféria sebessége nem befolyásolható:
    - **Egyoldali feltételes forgalomszabályozás**

# Tesztek

- Mely állítások igazak a Daisy chain alapú és a párhuzamos arbitrációra?

Tud úgy elromlani egy periféria, hogy másokat elzárjon a busztól

Daisy chain alapú arbitráció

Ha elromlik az arbiter, leáll a rendszer

Párhuzamos arbitráció

Igazságosabb kiszolgálást is lehetővé tesz

Könnyen bővíthető új perifériával

Nem működik, ha egyidejűleg több eszköz is bejelenti igényét a buszra

# Tesztek

- Mely állítások igazak a Daisy chain alapú és a párhuzamos arbitrációra?

Daisy chain alapú arbitráció

Párhuzamos arbitráció

Tud úgy elromlani egy periféria, hogy másokat elzárjon a busztól

Ha elromlik az arbiter, leáll a rendszer

Igazságosabb kiszolgálást is lehetővé tesz

Könnyen bővíthető új perifériával

Nem működik, ha egyidejűleg több eszköz is bejelenti igényét a buszra

# Tesztek

- Milyen eszköz található az Arduino analóg bemenetein?
  - Analóg-digitális átalakító (ADC)
  - Digitális-analóg átalakító (DAC)
  - PWM generátor

# Tesztek

- Milyen eszköz található az Arduino analóg bemenetein?
  - **Analóg-digitális átalakító (ADC)**
  - Digitális-analóg átalakító (DAC)
  - PWM generátor

# Tesztek

- Milyen forgalomszabályozást használ a PCI?
  - Semmilyet
  - Az egyik fél jelezni tudja, ha kész az adatátvitelre
  - Mindkét fél jelezni tudja, ha kész az adatátvitelre

# Tesztek

- Milyen forgalomszabályozást használ a PCI?
  - Semmilyet
  - Az egyik fél jelezni tudja, ha kész az adatátvitelre
  - **Mindkét fél jelezni tudja, ha kész az adatátvitelre**

# Tesztek

- Hogyan tudja használni egy PCI eszköz a rendszerszintű DMA vezérlőt?
  - A csatolófelületen erre szolgáló lábak segítségével
  - Speciális tranzakciók segítségével
  - Sehogy

# Tesztek

- Hogyan tudja használni egy PCI eszköz a rendszerszintű DMA vezérlőt?
  - A csatolófelületen erre szolgáló lábak segítségével
  - Speciális tranzakciók segítségével
  - **Sehogy**

# Tesztek

- Mely adatátviteli lehetőségeket támogatja a PCI és a PCI Express? (Kezdeményező → megszólított)
  - CPU → PCI periféria
  - PCI periféria → CPU
  - PCI periféria → memória
  - memória → PCI periféria
  - PCI periféria → PCI periféria

# Tesztek

- Mely adatátviteli lehetőségeket támogatja a PCI és a PCI Express? (Kezdeményező → megszólított)
  - **CPU → PCI periféria**
  - PCI periféria → CPU
  - **PCI periféria → memória**
  - memória → PCI periféria
  - **PCI periféria → PCI periféria**

# Tesztek

- Mely 3 koordinátával lehet azonosítani egy pontot egy több adathordozó réteggel ellátott forgólemezes adattárolón?
  - **Radiális távolság**
  - **Egy fix referenciához képesti szög**
  - **Az adathordozó réteg száma**

# Tesztek

- Ha egy merevlemez minden sávjában ugyanannyi, egyforma méretű szektor található, akkor
  - A külső sáv szektorait gyorsabban kell leolvasni
  - A belső sáv szektorait gyorsabban kell leolvasni
  - Minden szektort ugyanannyi ideig tart leolvasni

# Tesztek

- Ha egy merevlemez minden sávjában ugyanannyi, egyforma méretű szektor található, akkor
  - A külső sáv szektorait gyorsabban kell leolvasni
  - A belső sáv szektorait gyorsabban kell leolvasni
  - **Minden szektort ugyanannyi ideig tart leolvasni**

# Tesztek

- Rajzolja fel a merevlemez egy szektorának felépítését!
  - Rés
  - Előtag
  - „Adat kezdődik” jel
  - Adatok
  - Hibadetektáló kód (CRC)
  - Hibajavító kód (ECC)
  - Töltelék terület

# Tesztek

- A mágneses adathordozó egy blokkjának állapota a következő: →←←←→→←→←

  A váltáshoz 1-es bitet, a váltás hiányához 0-ás bitet rendelünk.
- Milyen bitsorozat tartozik a megadott mágneses mintázathoz:
  - **10010111**
- Változtassa meg a harmadik bitet (1-től számolva)!
- Milyen mintázatot kapunk?
  - →←←→←←→←→

# 1. Számpélda

- 1. Processzorunk órajel frekvenciája 100MHz. A számítógéphez egy billentyűzetet kötünk, melyen átlagosan 10 karaktert ütnek le másodpercenként, de két leütés akár 50 ms-onként is követheti egymást. A billentyűzet állapotának lekérdezése (mely tartalmazza a lenyomott gomb kódját is) 500 órajelet igényel. Az interrupt feldolgozási idő ezen felül még 100 órajel.

# 1. Számpélda

- 1. Processzorunk órajel frekvenciája 100MHz. A számítógéphez egy billentyűzetet kötünk, melyen átlagosan 10 karaktert ütnek le másodpercenként, de két leütés akár 50 ms-onként is követheti egymást. A billentyűzet állapotának lekérdezése (mely tartalmazza a lenyomott gomb kódját is) 500 órajelet igényel. Az interrupt feldolgozási idő ezen felül még 100 órajel.
- Hányszor kell másodpercenként lekérdezni a billentyűzetet, hogy ne maradjunk le semmiről?
  - 1 lekérdezés / 50 ms = 1 lekérdezés / 0.05 s =
    = **20 lekérdezés / s**

# 1. Számpélda

- 1. Processzorunk órajel frekvenciája 100MHz. A számítógéphez egy billentyűzetet kötünk, melyen átlagosan 10 karaktert ütnek le másodpercenként, de két leütés akár 50 ms-onként is követheti egymást. A billentyűzet állapotának lekérdezése (mely tartalmazza a lenyomott gomb kódját is) 500 órajelet igényel. Az interrupt feldolgozási idő ezen felül még 100 órajel.
- Mekkora terhelést jelent a processzor számára, ha a billentyűzet kezelésére polling-ot használ?
  - 20 lekérdezés / s * 500 órajel / lekérdezés = $10^4$ órajel / s
  - 100MHz = $10^8$ órajel/s
  - Terhelés: ($10^4$ órajel / s) / ($10^8$ órajel/s) = $10^{-4}$ = **0.01%**

# 1. Számpélda

- 1. Processzorunk órajel frekvenciája 100MHz. A számítógéphez egy billentyűzetet kötünk, melyen átlagosan 10 karaktert ütnek le másodpercenként, de két leütés akár 50 ms-onként is követheti egymást. A billentyűzet állapotának lekérdezése (mely tartalmazza a lenyomott gomb kódját is) 500 órajelet igényel. Az interrupt feldolgozási idő ezen felül még 100 órajel.
- Mekkora terhelést jelent a processzor számára, ha a billentyűzet kezelésére interrupt-ot használ?
  - 10 esemény / s * (500 órajel / esemény + 100 órajel / esemény) = $6*10^3$ órajel / s
  - 100MHz = $10^8$ órajel/s
  - Terhelés: ($6*10^3$ órajel / s) / ($10^8$ órajel/s) = $6*10^{-5}$ = **0.006%**

# 2. Számpélda

- Egy merevlemez 2 db kétoldalas lemezt tartalmaz, melyek mindegyikén 30000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs, a lemezek állandó 10000 fordulat/perc-el forognak.

# 2. Számpélda

- Egy merevlemez 2 db kétoldalas lemezt tartalmaz, melyek mindegyikén 30000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs, a lemezek állandó 10000 fordulat/perc-el forognak.
- Adjuk meg CHS koordináta rendszerben a lemez kapacitását!
  - **Cilinderek: 30000** (megegyezik a sávok számával)
  - **Fejek: 4** (mivel ennyi adathordozó réteg van)
  - **Szektorok: 1000** (sávonként értendő)

# 2. Számpélda

- Egy merevlemez 2 db kétoldalas lemezt tartalmaz, melyek mindegyikén 30000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs, a lemezek állandó 10000 fordulat/perc-el forognak.
- A lemez kapacitása bájtokban mérve:
  - Sávok száma összesen: 30000 * 4
  - Szektorok száma: sávok száma * szektorok/sáv = 30000*4*1000
  - Merevlemez kapacitása: szektorok száma * egy szektor mérete = 30000 * 4 * 1000 * 500 = **$6 * 10^{10}$**

# 2. Számpélda

- Egy merevlemez 2 db kétoldalas lemezt tartalmaz, melyek mindegyikén 30000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs, a lemezek állandó 10000 fordulat/perc-el forognak.
- Mennyi ideig tart, míg a lemez megtesz egy fordulatot?
  - (60 sec / perc) / (10000 fordulat / perc) =

    $6*10^{-3}$ sec/fordulat = **6 ms / fordulat**

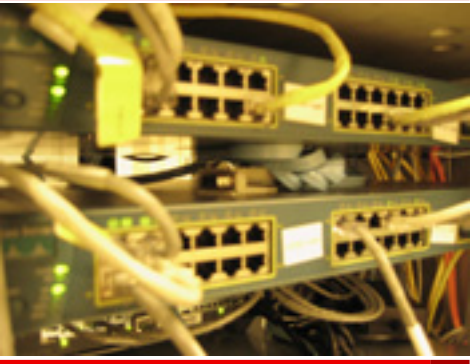
# 2. Számpélda

- Egy merevlemez 2 db kétoldalas lemezt tartalmaz, melyek mindegyikén 30000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs, a lemezek állandó 10000 fordulat/perc-el forognak.
- Meddig tart egy szektor leolvasása az adathordozóról? (Ha már ott van a fej)
  - Pont addig, amíg egy szektornyit elfordul a lemez a fej alatt
  - Seek, forgási idő nem számít! A feladat szerint már ott a fej!
  - 6 ms / fordulat → 6 ms / 1000 szektor = **0.006 ms / szektor**

# 2. Számpélda

- Egy merevlemez 2 db kétoldalas lemezt tartalmaz, melyek mindegyikén 30000 sáv található, minden sávban 1000 szektorral. A szektorok mérete 500 bájt. ZBR nincs, a lemezek állandó 10000 fordulat/perc-el forognak.
- Ha a parancsfeldolgozási idő 0.1 ms, az átlagos seek idő pedig 5 ms, az adatátviteli interfész sebessége pedig $50*10^6$ bájt/s, akkor átlagosan mennyi ideig tart egy 2000 bájtos kérés teljes kiszolgálása?
  - Parancsfeldolgozás: 0.1 ms
  - Seek: 5 ms
  - Forgási dő: 3 ms (véletlen pozícióba átlag egy fél fordulat kell)
  - Leolvasási idő: 0.006 ms / szektor, de 2000 bájt az 4 szektor, vagyis 4 * 0.006 ms = 0.024 ms
  - Adatátviteli idő: Kiszámoljuk egy szektor átviteli idejét az interfészen: (500 bájt / szektor) / ($50*10^6$ bájt / s) = $10^{-5}$ s / szektor = 0.01 ms
    Átlapoljuk a leolvasási idővel. Vegyük észre, hogy az interfész lassabb, mint a leolvasás. Emiatt az adatátvitel+leolvasás összesen = Első szektor leolvasása + mind a 4 szektor átvitele = 0.024 ms + 0.04 ms = 0.044 ms
  - Összesen: 0.1 + 5 + 3 + 0.044 = **8.144 ms**

MŰEGYETEM 1782



BME

# MEMÓRIA TECHNOLÓGIÁK

*Számítógép-architektúrák*
*4. gyakorlat*


Dr. Lencse Gábor
tudományos főmunkatárs
BME Híradástechnikai Tanszék
lencse@hit.bme.hu

2011. október 3.,
Budapest

# Tartalom

- Emlékeztető: mit kell már tudnom a memóriákról?
- SRAM és DRAM felépítése és ennek következményei
  - SRAM felépítése és működése
  - DRAM felépítése és működése
  - SRAM és DRAM összehasonlítása
- DRAM fejlődése
  - Aszinkron DRAM típusok
  - Szinkron DRAM típusok
  - Legfontosabb időzítések szinkron DRAM-oknál
- Memória modulok
- Feladatok
- Összefoglalás

# EMLÉKEZTETŐ: MIT KELL MÁR TUDNOM A MEMÓRIÁKRÓL?

# Emlékezető – ellenőrzés céljából

- Ismertnek tételezzük fel a alapvető memóriatípusokat, illesztési felületüket, műveleiket és azok idő diagramjait.
  - ROM (maszk programozott, PROM, EPROM): olvasás
  - SRAM: írás, olvasás
  - DRAM: írás, olvasás, frissítés
  - NVRAM: írás, olvasás, store, recall

*Akinek hiányosságai vannak, olvassa el a róluk szóló részt a*

***Memória technológiák***

*anyag elején a weben található jegyzetben!*

( `http://sites.google.com/site/szgarch` )

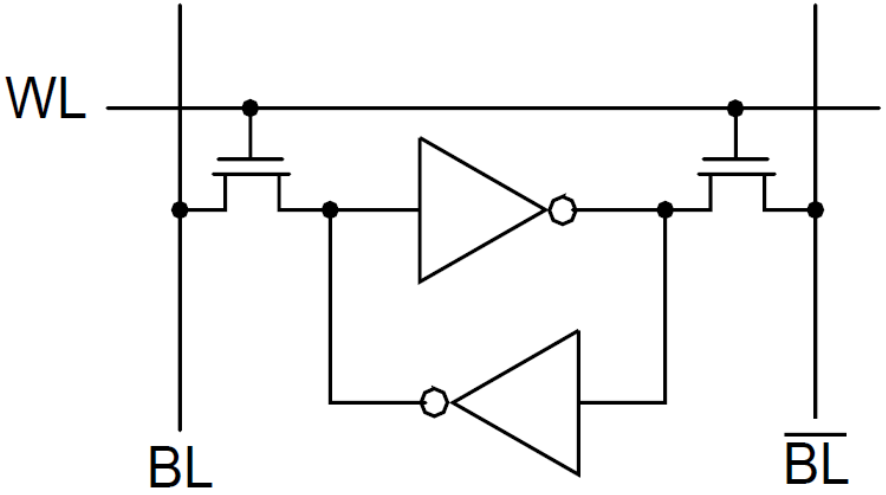

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SRAM ÉS DRAM FELÉPÍTÉSE ÉS ENNEK KÖVETKEZMÉNYEI


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM egy cellájának felépítése – 1

- A statikus RAM egy cellája (ami egy bit tárolására képes) – logikai áramkörként tekintve – két keresztbe kötött inverterből áll:
  - az egyik inverter bemenete a bitet,
  - a másiké annak inverzét tárolja.

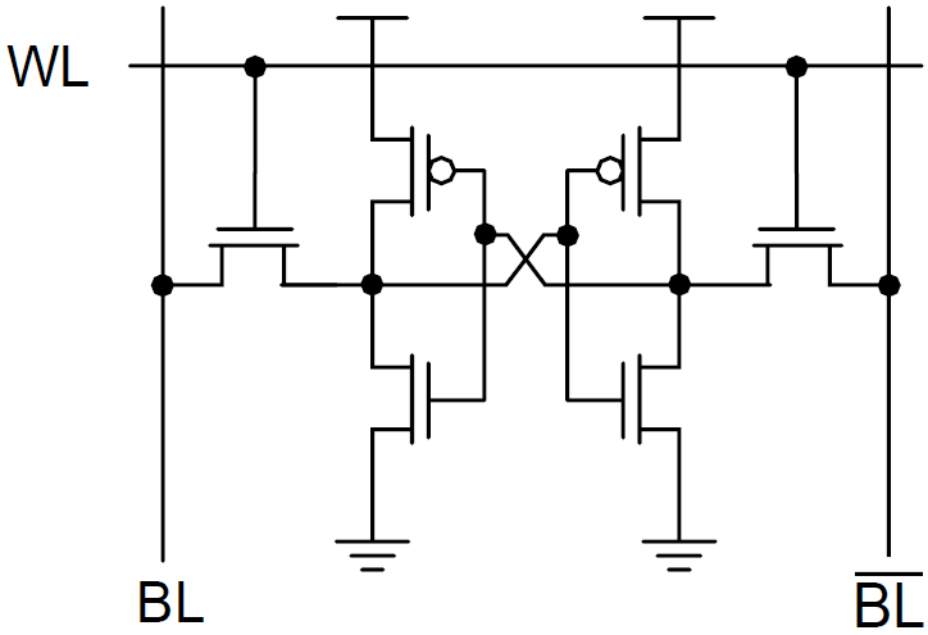



- Ez az áramkör mindaddig képes megőrizni a bit értékét, amíg tápfeszültséggel ellátjuk.
- A cella kiolvasását és írását további két tranzisztorral valósítják meg, ezeket *hozzáférési* (access) tranzisztoroknak hívjuk.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM egy cellájának felépítése – 2

- Mivel az inverterek megvalósításához is 2-2 tranzisztor szükséges, így cellánként összesen 6 tranzisztort használnak fel.




- Az ábrán WL (**W**ord **L**ine) a *szóvezetéket*, BL (**B**it **L**ine) a *bitvezetéket* jelenti.

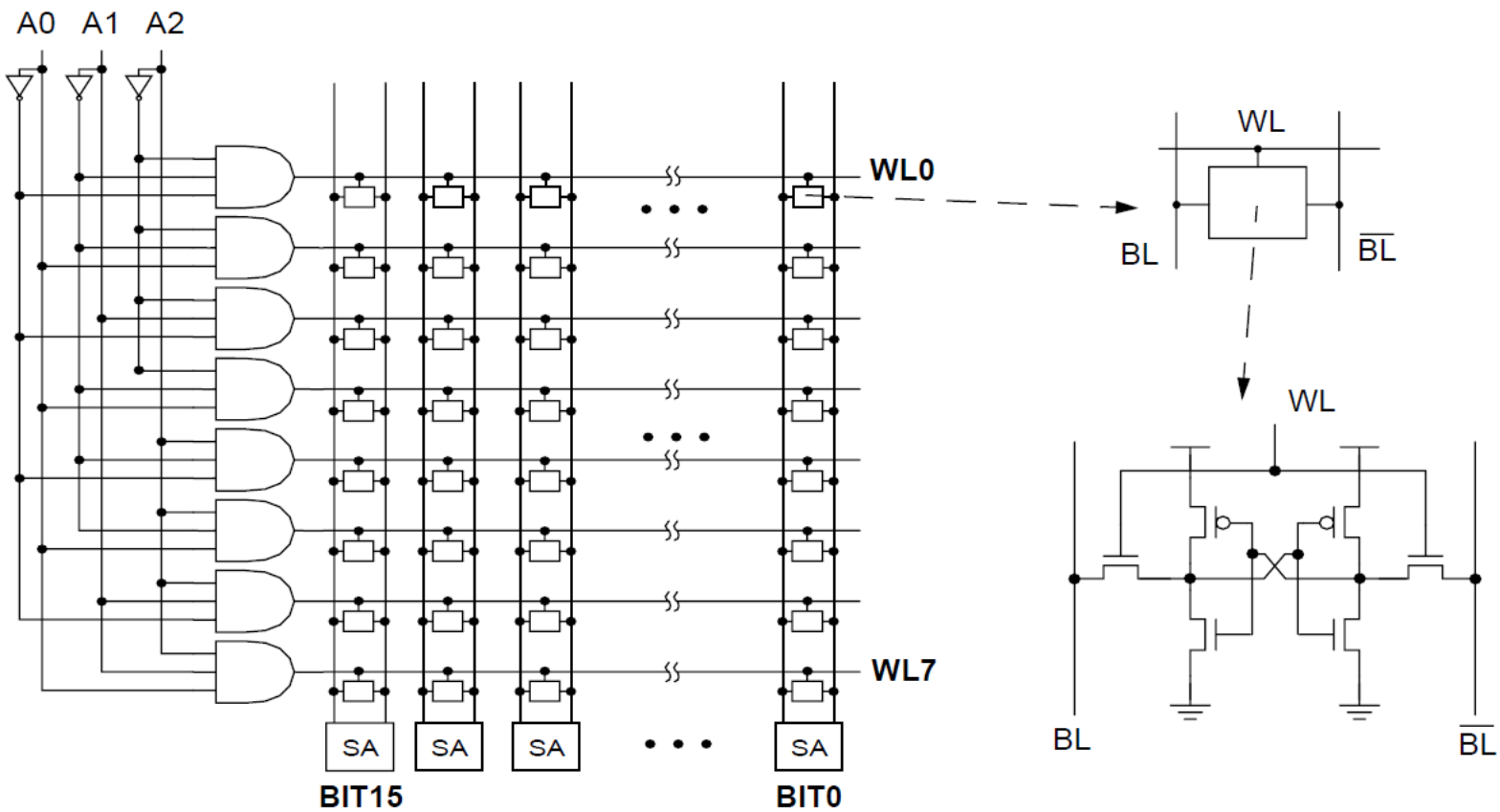
# Az SRAM felépítése – 1

- A cellákat mátrixba rendezik. Az egyszerűség kedvéért most a *szószervezésű* megoldással foglalkozunk; legyen a *címbitek* száma $n$, így a *szavak száma* $k=2^n$, a *szavak szélessége* pedig $m$ bit, ekkor a mátrix $k$ sorból és $m$ oszlopból áll.
- A következő dián egyszerű példaként egy nagyon kis méretű memóriát látunk a következő értékekkel: címbitek száma $n$=3, dekódolt címvezetékek (az ábrán $WL_0$-$WL_7$ szóvezetékek) száma $k$=8, a szószélesség $m$=16.
  - A bitmezőtől balra egy inverterekkel és ÉS kapukkal megvalósított 3/8-as dekódert látunk; a cellák tartalmát pedig a jobb oldalon kinagyítva és kirészletezve találjuk.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM felépítése – 2

- Az *érzékelő erősítőket* SA-val (sense amplifier) jelöltük.
- A rajzon nem szerepelnek, de minden bitvezetékhez tartoznak még a *előfeszítő* (precharge) áramkörök is.



# Az SRAM olvasása – 1

1. Az előfeszítő áramkörök a ponált és negált bitvezetékeket egyaránt logikai 1 szintre húzzák fel, majd a bitvezetékekről lekapcsolódnak. A bitvezetékek kapacitása miatt a bitvezetékek feszültségszintje megmarad.

   (Ez azért hasznos, mert a celláknak könnyebb/gyorsabb egy bitvezetéket 0-ra lehúzni, mint 1-re felhúzni.)

1. Ha az *i*. memóriaszót szeretnénk kiolvasni ($0<=i<2^n$), akkor a $WL_i$ szóvezetékre logikai 1 értéket adunk.

   (Ezt az ábrán a bitmezőtől balra található dekóderrel valósítjuk meg.)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM olvasása – 2

3. Az *i*. memóriaszó összes bitjének hozzáférési tranzisztorain át a bitek értéke megjelenik a bitvezetékeken:
   - amelyik bit értéke Q=1, ott BL értéke 1 marad, /BL-t pedig 0-ra húzza le a bit inverze (/Q);
   - a Q=0 értékű bitnél pedig a helyzet éppen fordított.
3. Az érzékelő erősítők érzékelik BL és /BL különbségét, és a kimenetükön rendelkezésünkre áll az *i*. memóriaszó bitjeinek értéke.

   (Minél érzékenyebbek, annál gyorsabb a kiolvasás.)

- Az olvasás a kiolvasott memóriaszó értékét nem változtatja meg.
- A bitek felismerését segíti (és gyorsítja), hogy nem abszolút jelszinteket, hanem BL és /BL különbségét kell felismerni.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM írása – 1

1. A ponált (BL) és negált (/BL) bitvezetékekre ráadjuk a beírandó bitek logikai értékének megfelelő feszültséget:
   - ha a *j*. bit értéke 1, akkor $BL_j$=1, $/BL_j$=0;
   - ha a *j*. bit értéke 0, akkor $BL_j$=0, $/BL_j$=1.
1. Ha az *i*. memóriaszót szeretnénk írni ($0<=i<2^n$), akkor a $WL_i$ szóvezetékre logikai 1 értéket adunk.
3. Az *i*. memóriaszó összes bitjének értéke a hozzáférési tranzisztorain keresztül felveszi a bitvezetékek által rákényszerített értéket, mivel a bitvezetékek meghajtó tranzisztorai erősebbek, mint a cellák tranzisztorai.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

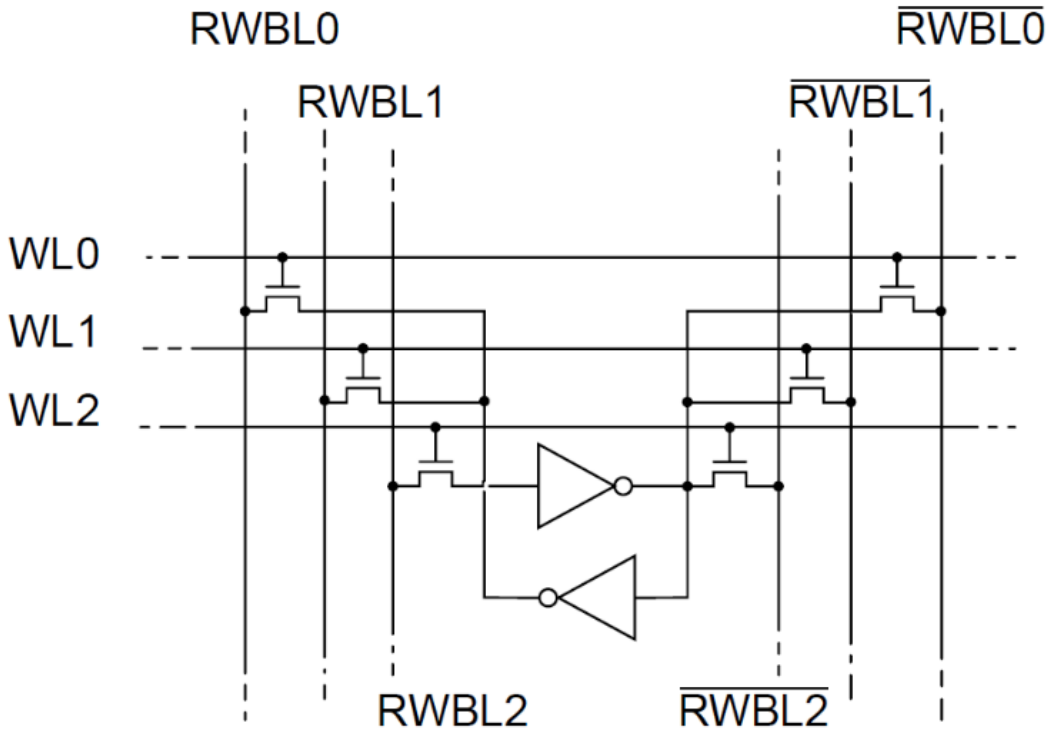
# Az SRAM írása – 2

- A megfelelő működéshez természetesen a tranzisztorok megfelelő méretezésére van szükség.
- Az írásnál szintén fontos szerepe van a két bitvezetéknek: mindkét inverter bemenetét megfelelő szintre beállítva billentjük át a bistabil multivibrátort.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Több portos SRAM megvalósítása

- Amennyiben több portos memóriát szeretnénk (ha egyidejűleg több eszköznek hozzá kell férnie), akkor a hozzáférési tranzisztorok száma nő: portonként egy pár tranzisztorra van szükség.
  - Tehát például 3 db írásra és olvasásra egyaránt használható port esetén 4+3x2=10 tranzisztorra van szükség, és egy cella a így néz ki:



# Az SRAM makrostruktúrája – 1

- Az SRAM struktúrájának bemutatásakor szószervezést használtunk. Ez komoly probléma, hiszen például egy kifejezetten kis kapacitásúnak számító 8kB-os SRAM esetén a címbitek száma $n$=13, a szóvezetékek (más néven dekódolt címvezetékek) száma $k=2^{13}=8192$, míg a szószélesség $m$=8; tehát a bitmező 8192 sorból és 8 oszlopból áll, ami meglehetősen aránytalan.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM makrostruktúrája – 2

- Erre a problémára viszonylag kis méret esetén elégséges megoldás lehet a *bitszervezés*, ahol a címbitek felét sorcímként, a másik felét oszlopcímként használjuk és a dekódolt címvezetékek metszéspontjában egy-egy ÉS kapuval engedélyezzük a cellát, de mivel ez bitenként egy-egy ÉS kaput igényelne, ezért a gyakorlatban a *módosított bitszervezés*t szokták használni, ahol *sor dekódert* és *oszlop multiplexert* alkalmaznak.
- Ezt a megoldást a DRAM struktúrájánál fogjuk bemutatni.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM makrostruktúrája – 3

- Bizonyos méret felett azonban nem egyetlen bitmezővel, hanem hierarchikus felépítésű struktúrával valósítják meg az SRAM-ot (és a DRAM-ot is).


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Az SRAM interfész lehetőségei – 1

- Szinkronizációs szempontból az SRAM-nak kétféle interfésze lehet:
  - Aszinkron interfész
  - Szinkron interfész

- Aszinkron interfész
  - Az első gyakorlaton bemutatott illesztési felület aszinkron: órajeltől függetlenül, a /CE, /OE, /WE jelekkel vezérelve, a lehető legrövidebb időn belül teljesíti a kapott parancsot.
  - Amennyiben az SRAM-ot mikroprocesszorhoz (vagy mikrokontrollerhez) illesztve operatív tárként használjuk, ez az interfész a szokásos megoldás.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

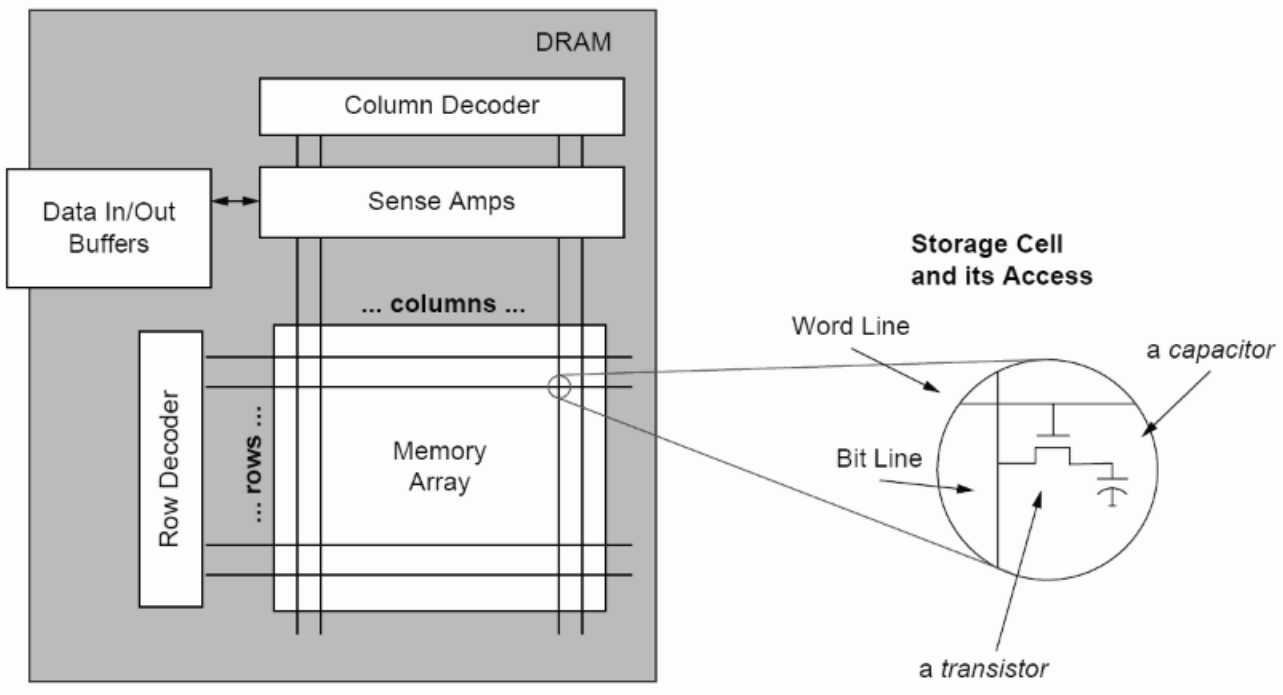
# Az SRAM interfész lehetőségei – 2

- Szinkron interfész
  - A szinkron interfésznél a korábban bemutatott illesztési felület kiegészül még egy (esetleg két) órajel bemenettel (CLK).
  - Ez a megoldás lehetővé teszi, hogy az átvitel *pipelined burst mode*ban történjen: például olvasáskor nemcsak a megadott címről, hanem a rákövetkező bizonyos számú címről is megtörténik a kiolvasás, és az egymást követő memóriarekeszekből való olvasás és a CPU felé való átvitel egyes fázisai egymással átlapolódnak.
  - Ezt a megoldást cache esetén szokták használni, hiszen így egy teljes cache sort igen hatékonyan át lehet vinni.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

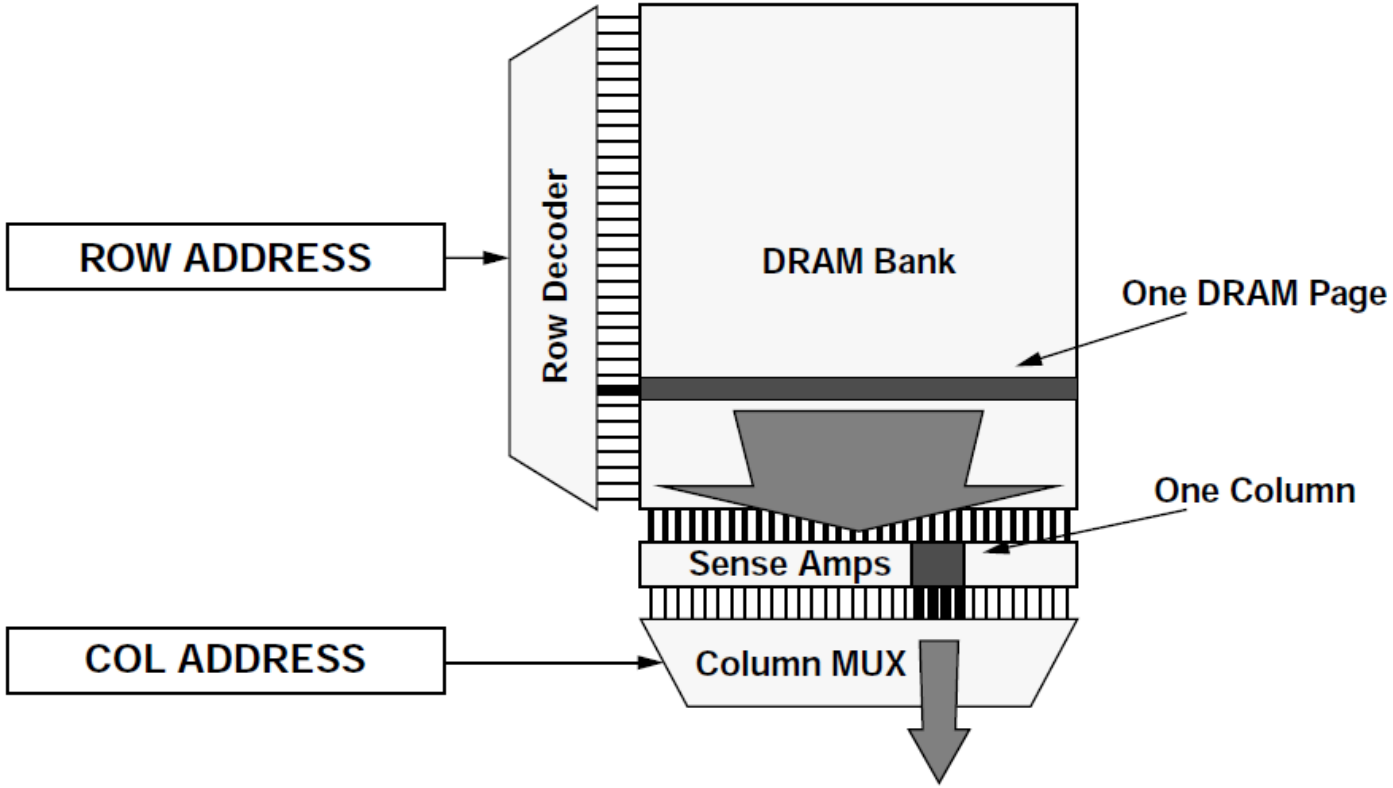
# A DRAM felépítése – 1

- A dinamikus RAM egy cellája (ami egy bit tárolására képes) logikailag egy tranzisztorból és egy kondenzátorból áll.
- A cellákat mátrixba rendezik; egy adott bitet a sor- és oszlopszáma azonosít.



# A DRAM felépítése – 2

- A cellákból egyszerre mindig egy sort lehet kezelni, ezt a sorcím alapján a sor dekóder választja ki; a sor bitjei közül pedig szükség esetén az oszlop multiplexerrel választunk. (Ezt módosított bit szervezésnek hívják.)





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM olvasása – 1

1. Az SRAM-hoz hasonlóan a bitvezetékeket itt is előfeszítjük, de az SRAM-mal ellentétben nem logikai 1-re, hanem csak a logikai 0 és 1 szintje közé "félútra".
2. A sorcím által kijelölt dekódolt sorvezetékre (a bevezető ábrán Word Line néven szerepel) logikai 1-et adunk, ennek hatására a kiválasztott sor tranzisztorai összekapcsolják a kondenzátorokat a hozzájuk tartozó bitvezetékkel, így a kondenzátor töltése módosítja a bitvezeték feszültségszintjét.
   - Az olvasás az SRAM-mal ellentétben destruktív!


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM olvasása – 2

3. Az érzékelő erősítők érzékelik a feszültségnek az előfeszített szinthez képest való eltérését, majd előállítják a korrekt logikai szintet, ami a kondenzátorokba vissza is íródik.
   - Így regeneráljuk a tönkretett tartalmat.
   - Amíg a /RAS jel alacsony szintje fennáll, a kiválasztott sor *aktív* állapota fennmarad. Angolul úgy fejezik ki, hogy "the page is open".
3. Az oszlop multiplexer az oszlopcím alapján kiválasztja a kért bitet és az megjelenik a DRAM kimenetén.
4. A /RAS jel felfutó élének hatására a sor aktív állapota megszűnik.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM adatszélessége

- A kívánt adatszélesség elérésére többféle módunk van.
  - Lehet tokonként csak egy bitet kezelni és egy memória modulra a kívánt számú DRAM tokot rátenni.
  - Lehet DRAM tokonként több bitet is kezelni;
    - akár úgy, hogy egymással párhuzamosan működtetünk több bitmezőt,
    - de úgy is, hogy egy oszlopcím hatására egymás mellett több bitet (pl. 4-et) választunk ki.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM írása

1. A sorcímmel kiválasztott sor aktiválása ugyanúgy történik, mint olvasás esetén (lásd ott: 1-3. lépések).
2. Az oszlopcímmel kiválasztott bit értékét az érzékelő erősítőre az adatbemeneten kapott értéknek megfelelően "rákényszerítjük".
3. Az egész sor visszaíródik (frissül), benne az oszlopcímmel kiválasztott bit az új értéket veszi fel.
4. A /RAS jel felfutó élének hatására a sor aktív állapota megszűnik.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM frissítése – klasszikus

- Az eredeti megoldás szerint (amikor a DRAM lényegében semmi támogatást sem ad a frissítéshez) a frissítés az olvasás 1-3. és 5. lépésével történik.
- Az olvasáshoz képest a 4. lépés azért marad el, mert a /RAS=0 tartama alatt nincs /CAS=0.
- Ehhez a megoldáshoz szükséges, hogy a DRAM kívülről megkapja a frissítendő sor címét.
  - A címet az egyszerű, 8 bites gépek idején tipikusan a CPU-nak kellett előállítania; ehhez egy számlálót használt, amiben nyilvántartotta, hogy éppen melyik sor frissítése következik. Adott időközönként egy megszakítási rutin elvégezte a következő esedékes sor frissítését.
  - Később a frissítés memória vezérlő feladata lett.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM frissítése – korszerű – 1

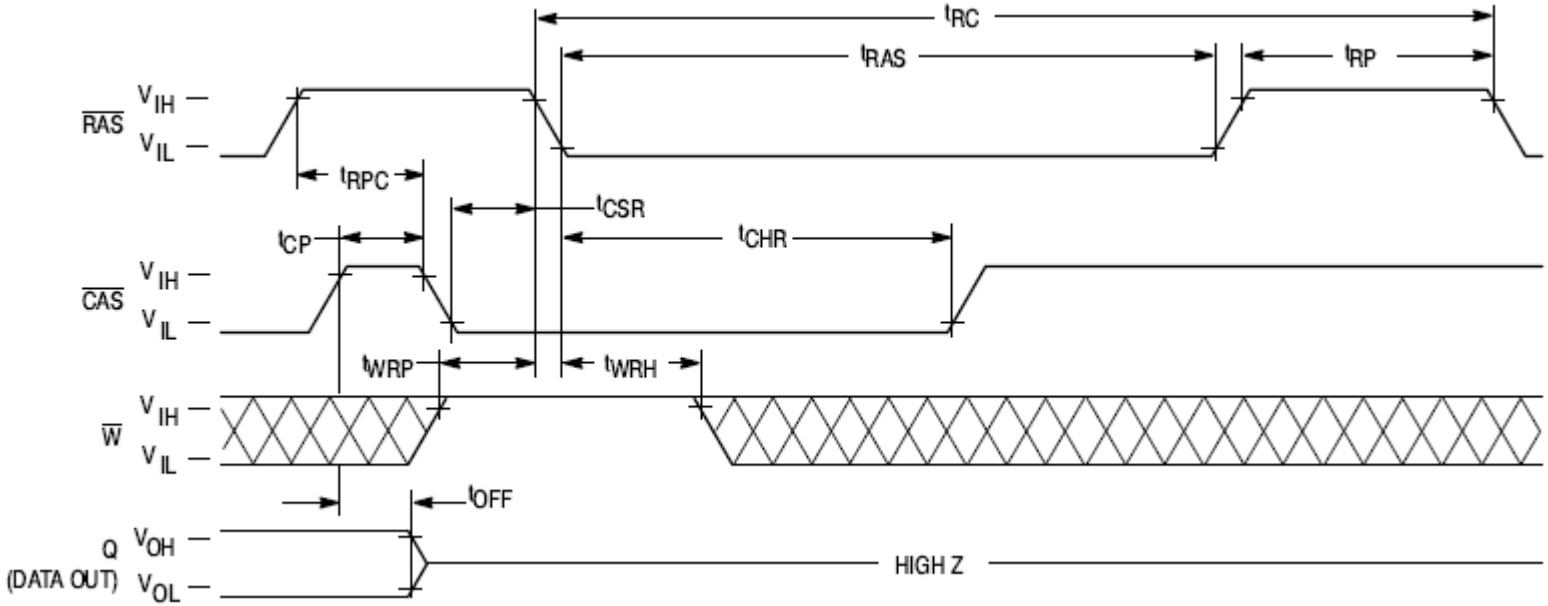
- DRAM chipeket kiegészítették egy számlálóval, ami nyilvántartja, hogy melyik sort kell frissíteni, és a frissítés protokollját is módosították:
  - Amennyiben a /CAS jel szintje 0 lesz még a /RAS jel 0 szintje előtt, majd ezt követi a /RAS=0, akkor ez (a korábban illegális eset) azt jelenti, hogy a kapott címet figyelmen kívül hagyva, a DRAM-nak a belső számlálója szerinti sort kell frissítenie.
  - Ilyenkor természetesen növeli is a számláló értékét.

Ezt *CBR frissítés*nek (CAS-before-RAS refresh) nevezik.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# A DRAM frissítése – korszerű – 2

- Az alábbi ábra egy katalógusból származik, pusztán a CBR frissítés illusztrációjának szánjuk, az egyes időzítésekkel most nem foglalkozunk.





© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SRAM és DRAM összehasonlítása – 1

- Költség szempontjából
  - Mivel egy SRAM cella 6 tranzisztorból áll, egy DRAM cella pedig egy tranzisztorból és egy kondenzátorból, ráadásul az SRAM-nál bitenként két bitvezeték van, a DRAM-nál pedig csak egy, ezért adott méretű lapkán DRAM esetén több (6-8x annyi) bit fér el, következésképpen olcsóbb.
  - Szintén az olcsóságot segíti elő, hogy DRAM-nál a címet két részletben viszik át, ami csökkenti a tok lábszámát, de a DRAM fent megismert működése miatt további lassulást nem okoz.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SRAM és DRAM összehasonlítása – 2

- Sebesség szempontjából
  - Működési sebességet tekintve azonban az SRAM nyer, ennek oka szintén a belső felépítésében keresendő: a bistabil multivibrátor állapota gyorsabban érzékelhető (néhány ns), mint a kondenzátor csekélyke töltésének állapota (néhányszor 10ns).
- Felhasználás egyszerűsége szempontjából
  - Az illesztés is SRAM esetén kényelmesebb: a cím egyben kezelhető; ráadásul frissítésre sincs szükség.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SRAM és DRAM összehasonlítása – 3

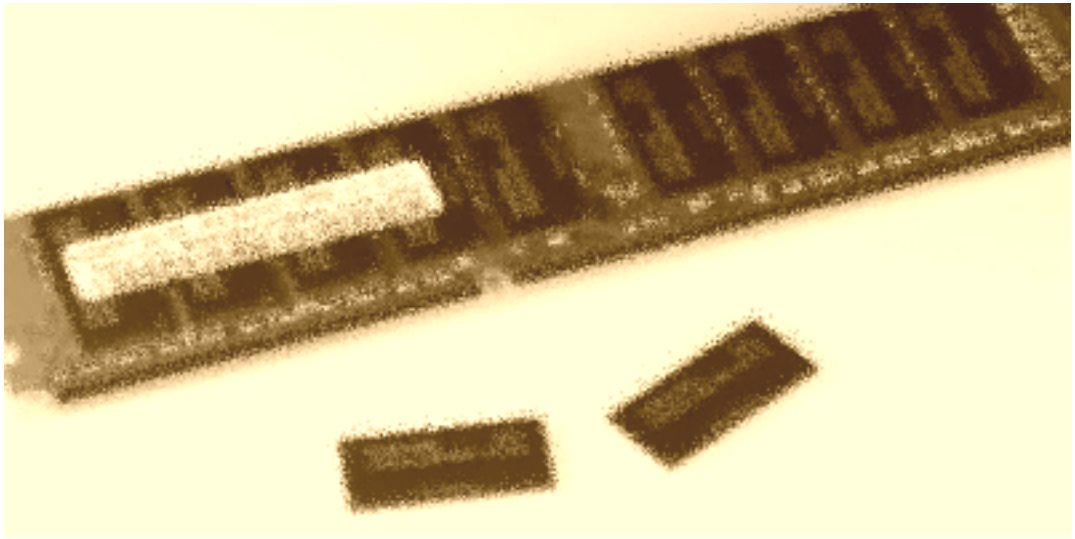
- Célszerű felhasználási terület szempontjából
  - A fentiekből következően számítógépek operatív memóriája céljára DRAM-ot használnak.
  - Az SRAM-ot cache céljára és (cache nélküli) kis rendszerek operatív memóriájaként is alkalmazzák.
  - A CPU és az SRAM azonos technológiával gyártható, ezért azokat könnyű egy lapkára integrálni (belső cache), míg a DRAM gyártási technológiája eltérő.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Elvi korlát az elérési időre

- Felmerül a kérdés, hogy ha az SRAM és DRAM közötti sebességkülönbség oka a technológiában keresendő, akkor miért van értelme több szintű cache alkalmazásának - különösen akkor, ha ezek mindegyikét a CPU-val azonos lapkára integrálják.
- A válasz nagyon egyszerű: a jelterjedési késleltetés miatt. A biteket tároló cellákat (akár egy, akár több bitmezőt használunk is) lényegében mindenképpen két dimenzióban kell elrendeznünk, így a befoglaló négyzet oldala a bitszám (azaz a memóriakapacitás) négyzetgyökével arányos: így ha például a cache kapacitását két nagyságrenddel növeljük, akkor a jelterjedési késleltetés egy nagyságrenddel nő.

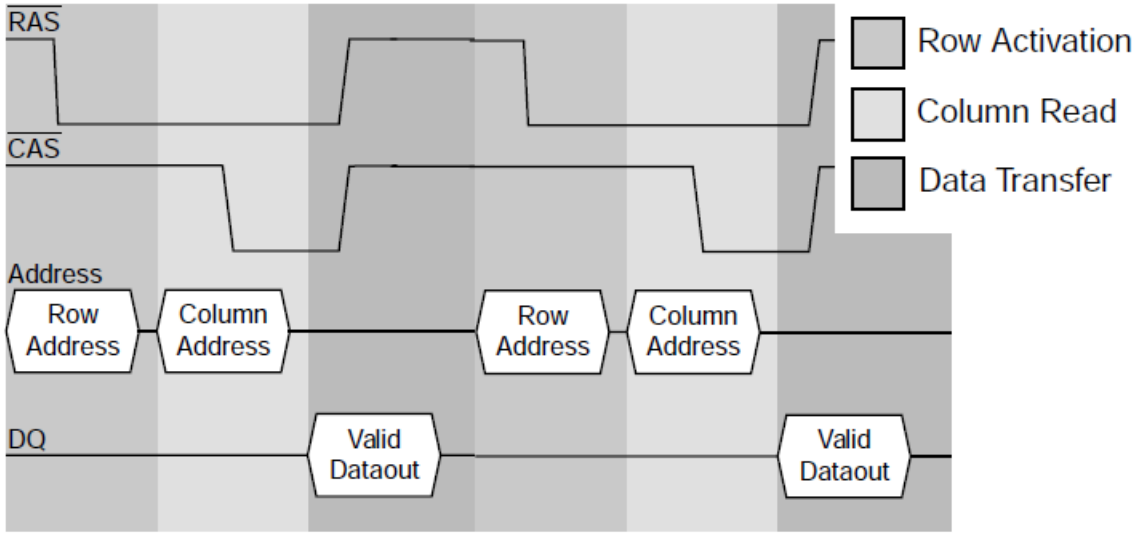

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



# A DRAM FEJLŐDÉSE

# A klasszikus aszinkron DRAM

- Teljesítményének fokozására több, általában egymásra épülő megoldás is született, a továbbiakban ezeket vesszük sorra. Összehasonlítási alapként tekintsük az alábbi ábrát; itt minden olvasásánál újra és újra meg kell adni először a sorcímet, majd az oszlopcímet.



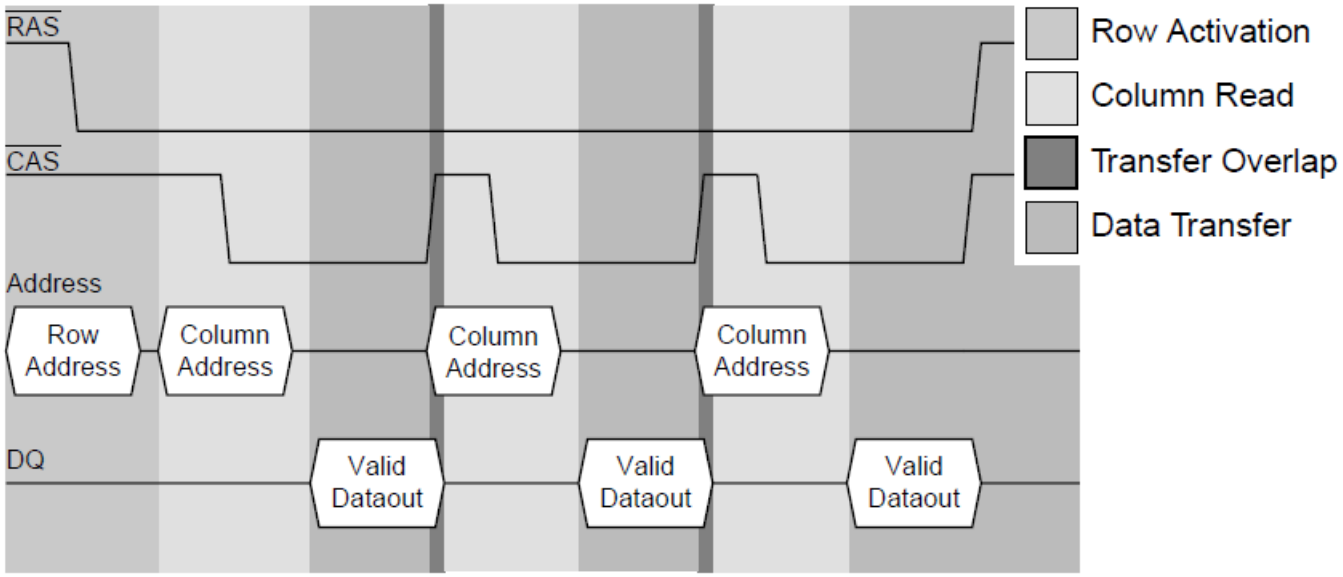
# FPM DRAM – 1

- A DRAM olvasásának átlagos ideje lerövidíthető, ha megengedjük, hogy azonos sorcímről történő olvasás esetén a /RAS jel alacsonyan tartása mellett, csupán a CAS jellel vezérelve egymás után több oszlopcímről is olvashassunk.
- Láttuk, hogy amíg a /RAS=0 fennáll, a sorcím által kiválasztott sor összes bitje elérhető az érzékelő erősítőkben, tehát csak ki kell választani közülük a számunkra még szükséges további biteket.
- Ezt a megoldást (gyors) *lapolvasás*nak (**F**ast **P**age **M**ode) nevezzük; az azonos sorcímhez tartozó oszlopokat egy memória "lap"-nak tekintjük.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# FPM DRAM – 2

- Figyeljük meg a megoldás nyereségét az alábbi ábrán!
  - A sorcímet laponként egyszer elegendő átvinni.
  - Apró átlapolás is megfigyelhető az érvényes adat és az új oszlopcím átvitelénél, de ennek mértéke nem jelentős.



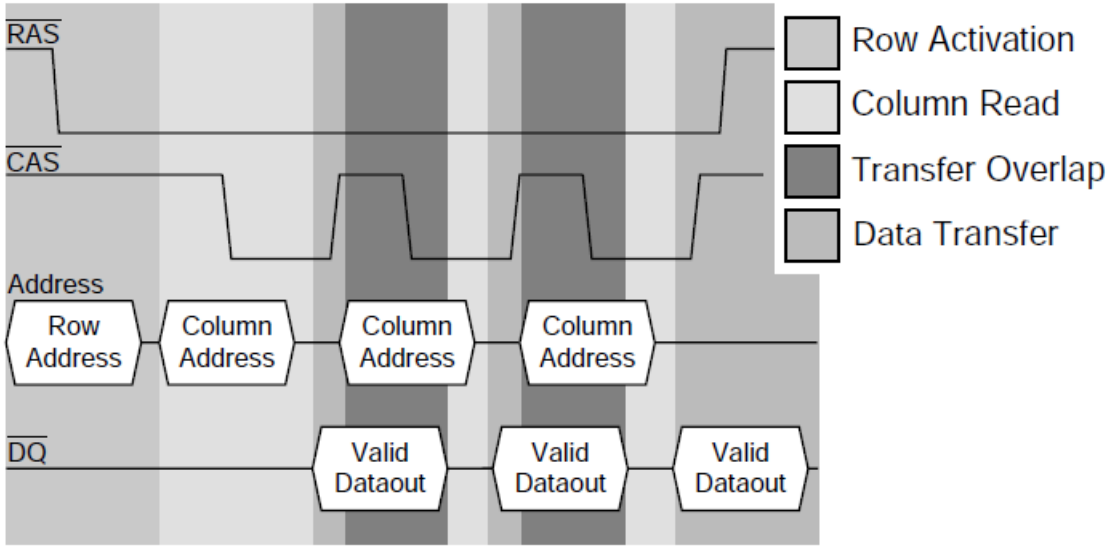
# EDO DRAM – 1

- A következő lépés az EDO (**E**xtended **D**ata **O**ut) DRAM, ami az FPM DRAM kiegészítése egy kimeneti latch-csel; így az adat kívülről még elérhető marad, miközben már új oszlopcímet adunk meg. Ezzel igen jelentős átlapolás érhető el a bitmezőből kiolvasott adat átvitele és az új oszlopcím megadása között.
- Megjegyzés: Az EDO DRAM-nál bevezették az /OE bemenetet, a CPU (vagy memória vezérlő) ezzel tudja jelezni, hogy meddig van szüksége az adatra.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# EDO DRAM – 2

- Figyeljük meg az átlapolást az érvényes adat és az új oszlopcím átvitele között!



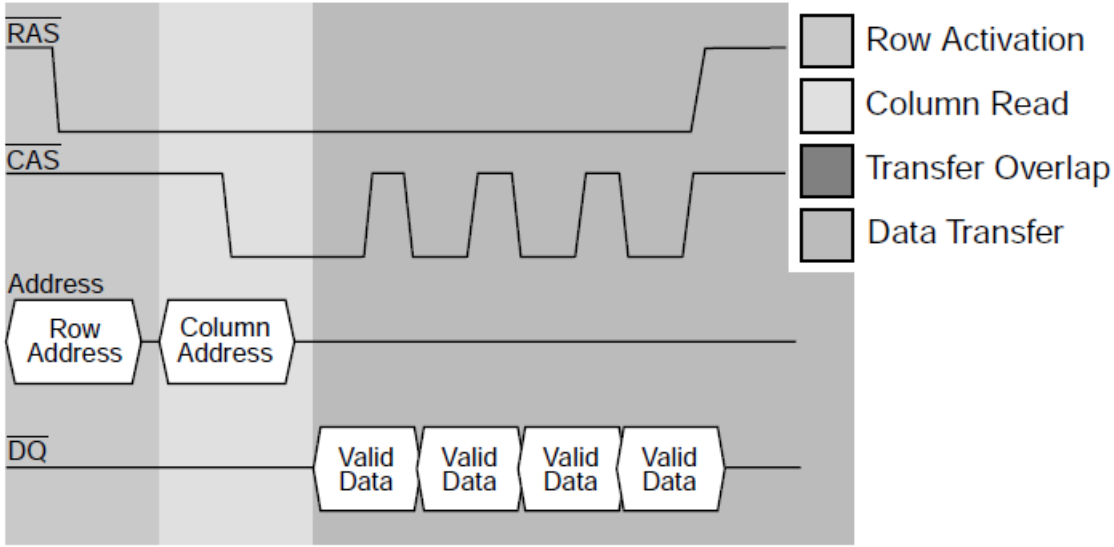
# BEDO DRAM – 1

- Ismét egy újabb lépés a BEDO (**B**urst-Mode **EDO**) DRAM, ahol már nem is kell megadni az új oszlopcímet, hanem az oszlopcímet egy belső számláló egyesével növeli a kezdeti értékről.
- Így természetesen csak egymást követő címek érhetők el, de a gyakorlatban úgyis ez a szokásos eljárás.
- (Ez a megoldás is kifejezetten hasznos, de kevésbé terjedt el.)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# BEDO DRAM – 2

- Figyeljük meg, hogy az időegységenként átvitt adatok mennyisége tovább nő!



# SDRAM – 1

- Az SDRAM (**S**ynchronous **DRAM**) egy igen jelentős fejlődési lépés. Míg korábban a /CAS és /RAS bemenetek értéke tetszőleges időpontban változhatott, és azok közvetlenül vezérelték a DRAM belső működését, addig az SDRAM-ban minden bemeneti változás egy külső órajellel szinkronizáltan történik.
- Ez azért előnyös, mert az SDRAM tervezői így kihasználhatják, hogy az IC nem akármikor, hanem csak előre meghatározott időpontokban kaphat valamilyen parancsot.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

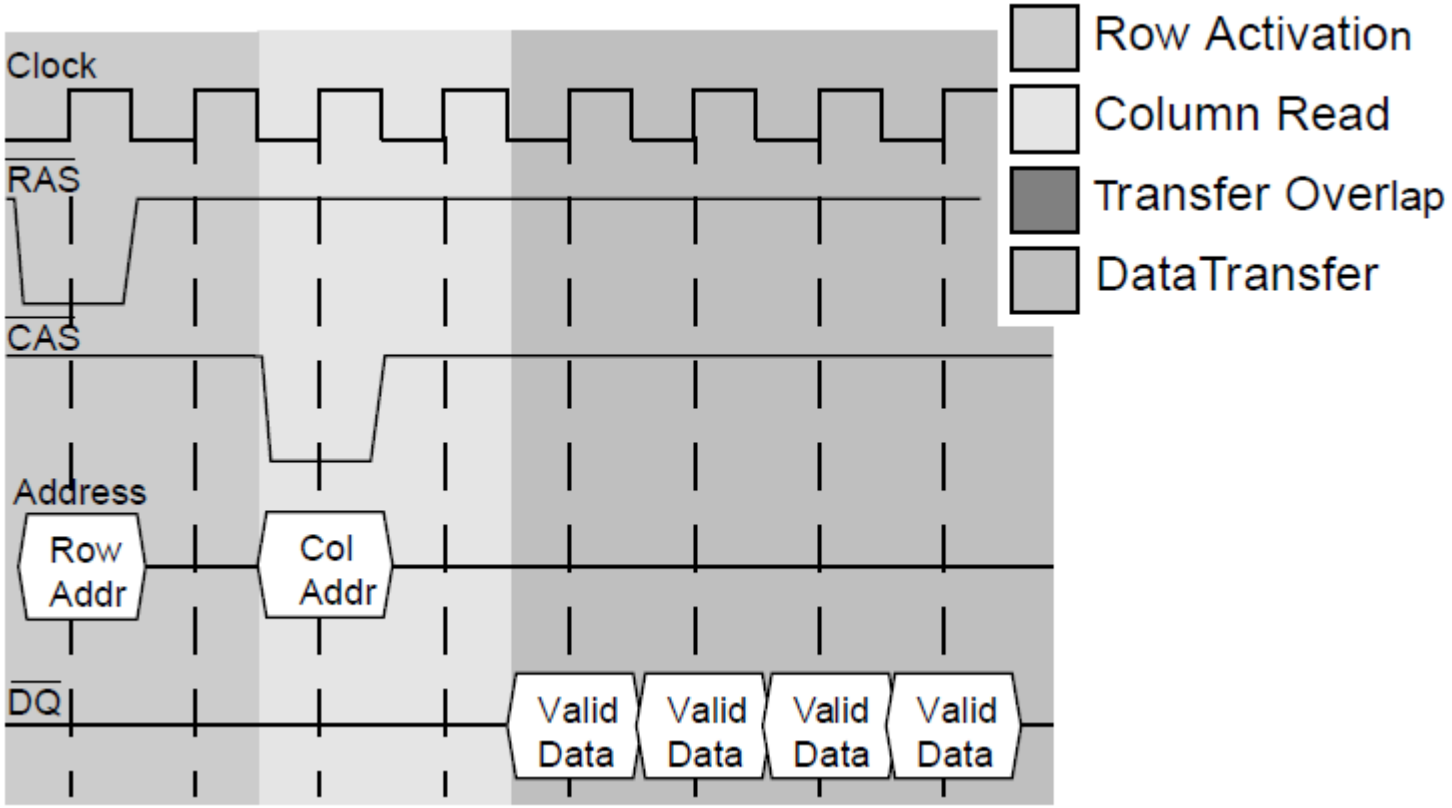
# SDRAM – 1

- A BEDO DRAM-hoz hasonlóan az SDRAM is képes automatikusan növekvő oszlopcímekről egymás után több adat átvitelére (burst), de attól eltérően itt már ezt nem kell a /CAS jellel újra és újra igényelni, hanem az adatok megjelenítése az órajel ütemére történik, és az SDRAM-ban van egy programozható regiszter, amiben az egymás után átviendő adatok számát be lehet állítani (például a cache sor hosszának függvényében).
- Bár az SDRAM megjelenésekor teljesítményében nem volt jobb a BEDO DRAM-nál, hosszú távon nyertes lett, mert a szinkronizált működés egyre nagyobb működési frekvenciát tett lehetővé.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SDRAM – 2

- Figyeljük meg, hogy az órajellel ütemezetten az újabb adatok megjelennek a kimeneten – minden további kérés nélkül!



# SDRAM – 3

- A további, ma is általánosan elterjedt típusok mind az SDRAM továbbfejlesztett változatai, ezeket röviden áttekintjük, de előbb megismerjük a legfontosabb időzítéseket.
- Megjegyzés: A későbbi, DDRx SDRAM-októl való megkülönböztetés érdekében az eredeti SDRAM-ot SDR (**S**ingle **D**ata **R**ate) SDRAM-nak is szokták nevezni.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SDRAM időzítések

- Mivel minden órajellel szinkronizáltan történik, ezért az időzítéseket nem ns-ban, hanem a szükséges órajel ciklusok számában szokták megadni. Természetesen ezek az órajel frekvenciájának függvényében értelmezhetők, és szükség esetén ns-ra is átszámíthatók. (Aszinkron DRAM esetében pedig természetesen ns-ban adják meg őket.)
- A következőkben megadjuk azt a négy időzítést, amivel az egyes SDRAM memória modulokat jellemezni szokták.
- Az időzítések értékét a következőkben használt sorrendben, a számértékeket egymástól kötőjellel elválasztva szokták feltüntetni, például: 3-4-4-8.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# CAS Latency

- Definíció
  - A /CAS=0-tól az adat megjelenéséig eltelt idő. (Természetesen egy már aktív sor esetén.)
- *Magyarázat*
  - *Eddig tart, amíg az érzékelő erősítőben már jelen levő bitek közül az oszlopcím által kiválasztott bit a kimeneten megjelenik.*
- *Jelölések*
  - $t_{CL}$, $t_{CAS}$, ($t_{ACT}$)
- Alternatív megnevezés
  - **A**ccess **C**olumn **T**ime


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# RAS to CAS Delay

- Definíció
  - A /RAS=0-tól legalább ennyi időnek el kell telnie a /CAS=0 megjelenéséig.
- *Magyarázat*
  - *Eddig tart a sor "megnyitása": a sor kiválasztásától addig eltelő idő, amíg a kiválasztott sor tartalma megjelenik az érzékelő erősítőben. Ez után lehet olvasni vagy írni.*
- *Jelölés*
  - $t_{RCD}$


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Row Precharge time

- Definíció
  - A /RAS jelnek legalább ennyi ideig magas szintűnek kell lennie, azaz egy aktív sor lezárásától kezdve ennyi időnek kell eltelnie, mire egy másikat meg lehet nyitni.
- *Magyarázat*
  - *Ennyi idő szükséges a bitvezetékek előfeszítéséhez.*
- *Jelölés*
  - $t_{RP}$


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Row Address Strobe time

- Definíció
  - A /RAS jelnek legalább ennyi ideig alacsony szintűnek kell lennie.
- *Magyarázat*
  - *Ez az idő ahhoz szükséges, hogy a sor kiolvasása és visszaírása rendben megtörténjen.*
- *Jelölés*
  - $t_{RAS}$
- Alternatív megnevezések
  - **R**ow **A**ctive **T**ime
  - Active to Precharge (time)


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# DDR SDRAM

- A DDR (**D**ouble **D**ata **R**ate) SDRAM azonos órajel esetén az SDR SDRAM-hoz képest (közel) kétszer akkora átviteli sebességre képes.
- Ezt úgy oldja meg, hogy az órajelnek mind a felfutó, mind a lefutó élénél végez adatátvitelt. Míg az SDR SDRAM minimális burst hossza 1, a DDR SDRAM-é 2.
- A szabványos nevében már a fizikai órajel frekvenciájának a dupláját használja, így a DDR-200 SDRAM valójában 100MHz-en működik.
  - A moduljai 64 bit szélesek, és a nevükben figyelembe veszik azt is, hogy hány byte információt visznek át párhuzamosan. Például egy 200MHz-en működő 64 bit széles DDR-400-at PC-3200 néven lehet megvásárolni, és maximum 3200MB/s átviteli sebességre képes.

# DDR2 SDRAM

- A DDR2 SDRAM belső órajele csak fele a külsőnek, így adott belső órajel mellett a külső kétszer akkora lehet, és a külső órajelciklusonként visz át két bitet, tehát a belső órajel minden periódusa alatt 4-et.
- Míg a DDR-nél 2 bites volt, itt már 4 bit-es a prefetch puffer (a DDR3-nál pedig 8 bites).
- Nézzünk itt is egy tipikus számpéldát.
  - Egy DDR2-800 SDRAM-nak a belső órajele 200MHz, a külső 400MHz, a "800" úgy jön ki, hogy a külső órajel fel- és lefutó élénél is van adatátvitel; a 64 bites szélesség miatt pedig PC2-6400 modulként adják el, és maximum 6400 MB/s átviteli sebességre képes.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# DDR3 SDRAM

- A DDR3 SDRAM belső órajele már csak negyede a külsőnek, így adott belső órajel mellett a külső négyszer akkora lehet, és a külső órajelciklusonként visz át két bitet, tehát a belső órajel minden periódusa alatt 8-at.
- Nézzünk itt is egy tipikus számpéldát.
  - Egy DDR3-1600 SDRAM-nak a belső órajele 200MHz, a külső 800MHz, az "1600" úgy jön ki, hogy a külső órajel fel- és lefutó élénél is van adatátvitel; a 64 bites szélesség miatt pedig PC3-12800 modulként adják el, és maximum 12800 MB/s átviteli sebességre képes.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# DDRx SDRAM

- A DDR sorozat tagjai egymással sem mechanikailag sem feszültségszint tekintetében nem kompatibilisek.
  - A disszipáció csökkentése érdekében egyre kisebb feszültségszinteket használnak.
- A DDR4 piaci megjelenése 2012-re várható.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

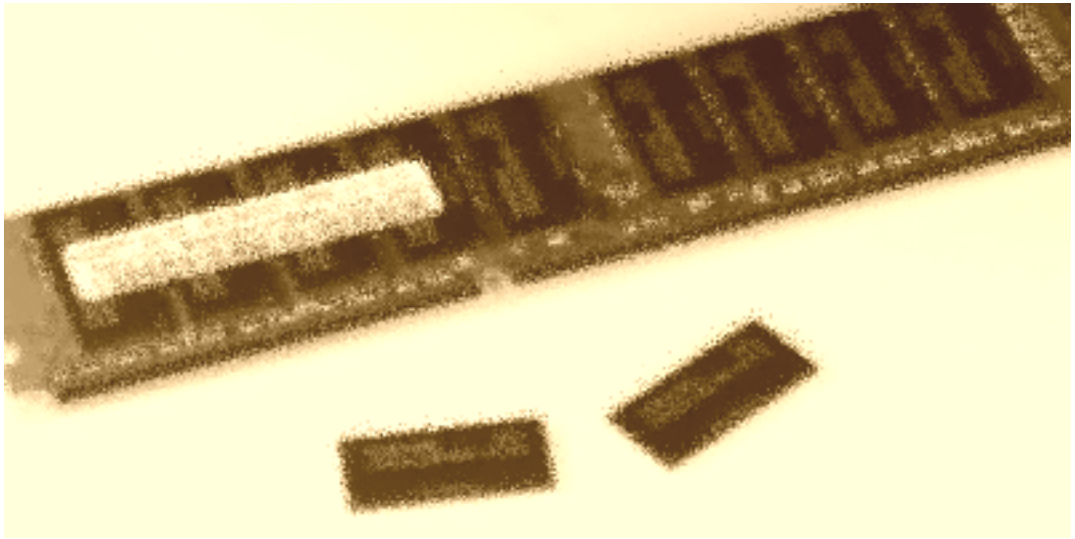
# A GDDR sorozat

- A GDDR (**G**raphics **DDR**) sorozat videokártyákhoz készült.
- A számozás 1-gyel (vagy kettővel) előbbre tart a DDR sorozatnál, így például a GDDR3 közelítőleg a DDR2-nek felel meg. Ahhoz képest van néhány előnye, ebből rendszertechnikailag számunkra az az érdekes, hogy egyidejűleg írható és olvasható.
- A GDDR4 a DDR3-ra épül és a GDDR3 utódjának szánták, de nem igazán váltotta be a hozzá fűzött reményeket.
- A GDDR5 is DDR3-ra épül, és 2008 óta kapható.

# A Rambus irányzat

- A Rambus cég nem gyárt memóriát csak fejleszt és licencel. Első termékével az **RDRAM**-mal (**R**ambus **DRAM**) forradalmi újításokba fogott (például lényegesen keskenyebb buszt használ; a parancsait nem 1 bites vonalak jelei, hanem több bites üzenetek hordozzák), de a teljesítményhez képest az ár túl magas volt, így bár egy ideig terjedt, aztán az olcsóbb DDR sorozat kiszorította.
- A következő terméke az **XDR** (e**X**treme **D**ata **R**ate) DRAM jelenleg a DDR2 és a GDDR4 vetélytársa. Olyan esetekben van esélye, ahol a kis lábszám előnyt jelent.
- 2008-ban elkészült az **XDR2 DRAM** specifikációja is, de azóta sem talált gyártóra.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



# MEMÓRIA MODULOK


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

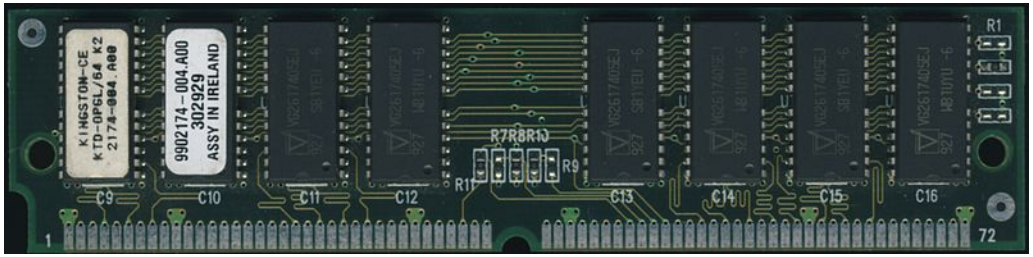
# SIMM – 1

- A SIMM (**S**ingle **I**n-line **M**emory **M**odule) memória modulok csak az egyik oldalon tartalmaznak memória IC-ket és bár a csatlakozási felületnél mindkét oldalon vannak érintkezőik, ezek redundánsak.
- Főleg az 1980-as és 90-es években használták őket, 30 és 72 érintkezős kivitelben készültek, az előbbiek 8/9 bitesek, az utóbbiak 32/36 bitesek voltak - attól függően, hogy tartalmaztak-e hibajavításra (ECC) 8 bitenként +1 bitet. Ezek általában még aszinkron DRAM-ok voltak, de készült még DDR SDRAM is SIMM kivitelben.


© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SIMM – 2

- Felül 8/9 bites aszinkron DRAM, alatta 32/36 bites FPM DRAM, legalul 32/36 bites  EDO DRAM



# DIMM – 1

- A DIMM (**D**ual **I**n-line **M**emory **M**odule) memória modulok mindkét oldalon tartalmaznak memória IC-ket, adatszélességük is 64 bit, és kihasználják mindkét oldalon az érintkezőket.
  - Az Intel Pentiummal terjedtek el, így már nem kellett őket párban használni, mint a SIMM modulokat.
- Bár készültek FPM és EDO DRAM-ok is DIMM kivitelben, de a DIMM-ek már főleg szinkron DRAM-ok, azon belül bejárták a technológiai fejlődést. Az érintkezők számában is változatosak.

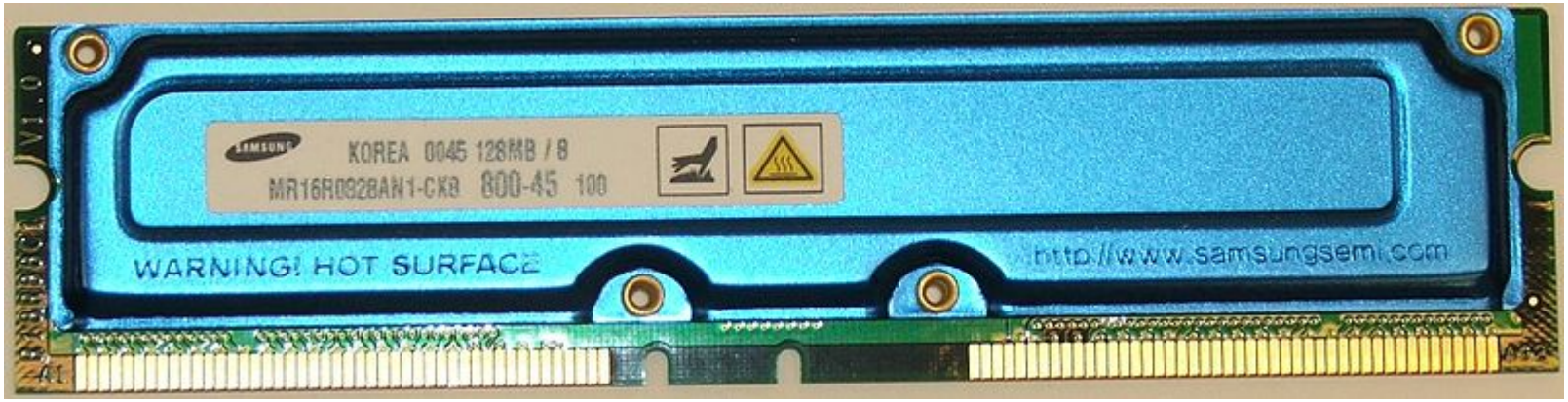

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# DIMM – 2

- Az alábbiak közül a felső egy 168 érintkezős SDRAM, az alsó egy 184 érintkezős DDR SDRAM:

# RIMM

- Bár az **RDRAM** (**R**ambus **DRAM**) vagy **DRDRAM** (**Direct Rambus DRAM**) modulok is SIMM vagy DIMM kivitelűek, megkülönböztetésül mégis **RIMM**-nek (**R**ambus **I**n-line **M**emory **M**odule) szokták őket nevezni.
  - Jól látható jellemzőjük az integrált hűtő - de integrált hűtő lehet más SIMM/DIMM modulon is!



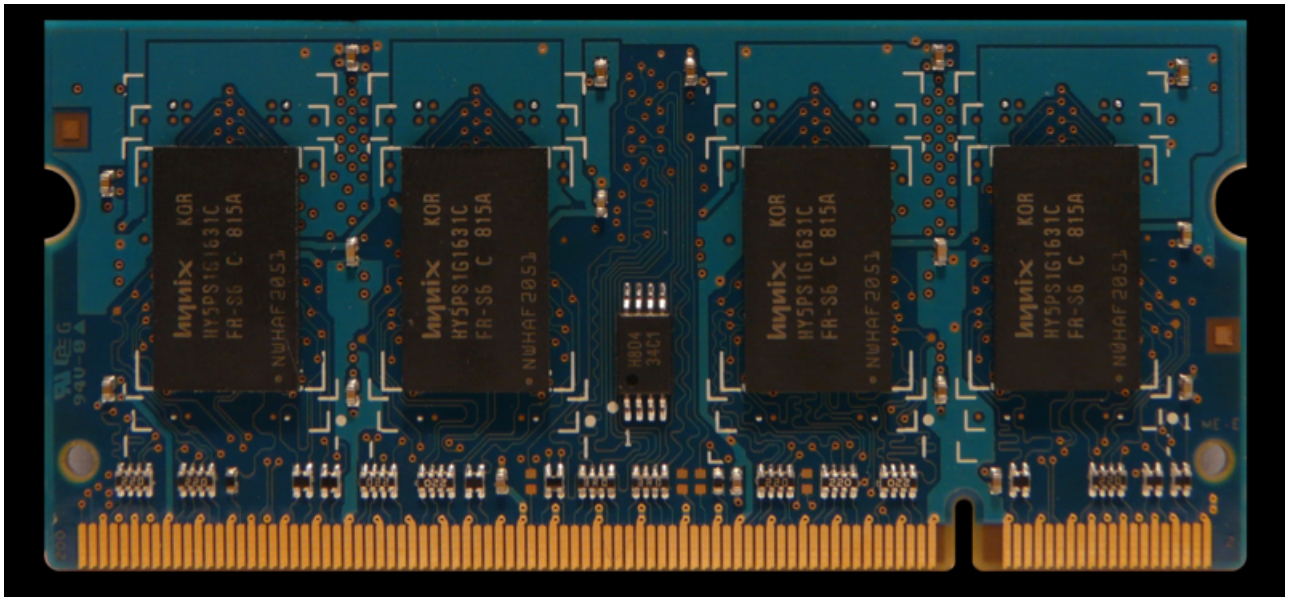
# XDR DRAM

- Az **RDRAM** utódja az **XDR** (e**X**treme **D**ata **R**ate) **DRAM** az elődjéhez hasonlóan szintén integrált hűtővel rendelkezik:

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# SO-DIMM

- A **SO-DIMM** (**S**mall **O**utline **DIMM**) a DIMM helytakarékos változata. Többféle érintkezőszámmal (100, 144 és 200) készül és DDR/DDR2/DDR3 SDRAM egyaránt található köztük. Az alábbi képen egy PC2-6400 DDR2 SO-DIMM látható.

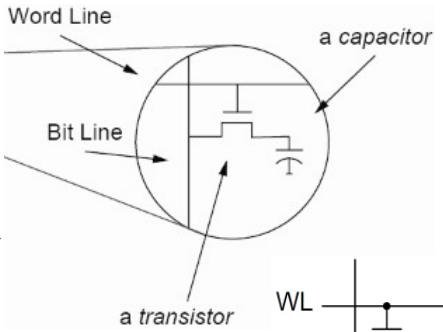


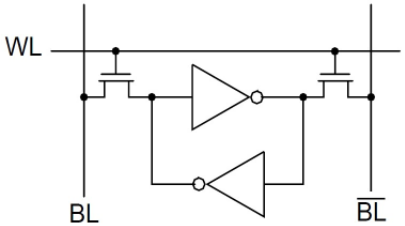
# Önálló hallgatói munka

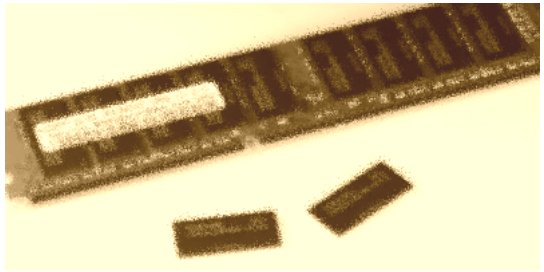
1. Számítsa ki a bitmezőt befoglaló négyzetekben létrejövő jelterjedési késleltetés arányát egy 16kB-os L1 és egy 1MB-os L2 cache esetén!
2. Van egy SDRAM chipünk. Az időzítései: 3-3-3-6. Feltéve, hogy előfeszített (precharged) állapotban van, és két bitet kell belőle kiolvasnunk, amelyek sorcíme különböző; a kezdéstől számítva mennyi idő múlva jelenik meg a második bit a kimeneten, ha a chip 100MHz frekvenciájú órajelet kap?
3. Oldja meg az előző feladatot 6-6-6-18 időzítéssel és 200MHz-es órajellel!
4. Egy PC3-6400-ként eladott DDR3 memóriamodul milyen belső és milyen külső órajellel működik?

# Összefoglalás

- SRAM és DRAM felépítése és ennek következményei
  - SRAM felépítése és működése
  - DRAM felépítése és működése
  - SRAM és DRAM összehasonlítása
- DRAM fejlődése
  - Aszinkron DRAM típusok
  - Szinkron DRAM típusok
  - Legfontosabb időzítések szinkron DRAM-oknál
- Memória modulok
- Feladatok



# Aktualitások

- Örömmel hirdetjük, hogy a **BME-Infokom Innovátor Kft. Hallgatói Ötletbörzét szervez a Híradástechnikai Tanszékkel** közösen, az alábbi **9 témában**:
  - 1) Infokommunikációs hálózatok tervezési és elemzési modelljei és módszerei
    2) Szélessávú hozzáférési hálózatok vizsgálata és elemzése
    3) Hálózat, információ és szolgáltatás biztonság
    4) Fix-mobil konvergencia
    5) Személyre szabott és helyfüggő média-tartalom szolgáltatások
    6) Intelligens közlekedési alkalmazások

    7) Rádiófrekvenciás azonosítás
    8) Nyomon követés és hely alapú szolgáltatások
    9) Intelligens környezet

http://www.bme-infokom.hu/palyazat/

# Kérdések?

# KÖSZÖNÖM A FIGYELMET!

Híradástechnikai Tanszék

Dr. Lencse Gábor
tudományos főmunkatárs
BME Híradástechnikai Tanszék
lencse@hit.bme.hu

© Dr. Lencse Gábor, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

# Számítógép Architektúrák: **Memóriakezelés feladatok**

**1. Feladat** Vegyünk egy 12 bites címtartománnyal rendelkező processzort. Fusson a processzoron egy program a memóriatartomány alsó felében, tehát a 0–2047 címtartomány legyen mindig elérhető. A program a fizikai memóriában, sorrendben az alábbi tartalmakat hivatkozza:

- 3987, 4200, 3518, 11092, 11093, 10070

A hivatkozott címek szemmel láthatóan meghaladják a processzor címtartományát. A feladatok a következők:

(a) Ha a tárbővítésre tömbkapcsolást alkalmazunk, 2 kB-os tömbökből építkezünk, és minden tömböt 2 kB-os határra illesztünk, hány tömbre van szükség (az állandó, mindig elérhetőt is beleértve)? Kizárólag a hivatkozott címeket akarjuk lefedni! A fenti hivatkozások során hány tömbátkapcsolásra lesz szükség, ha kezdetben az állandó 0-ás és az 1-es tömb van bekapcsolva?

(b) Ha a tárbővítést indexelt leképzéssel valósítjuk meg, és 1 kB-os ablakokat használunk,

- Hány ablakunk (indexregiszterünk) van?
- Állítsa be az összes indexregiszter értékét úgy, hogy a program futása közben ne kelljen állítani őket!

**2. Feladat** Legyenek a virtuális címek 24 bitesek, a fizikai címek 16 bitesek, a lapméret legyen $2^{12} = 4$ kB méretű, a laptábla pedig legyen egyszerű egyszintes laptábla 8 bites bejegyzésekkel. A feladatok a következők:

(a) A virtuális címekben hány bit tartozik a lapok azonosításához és hány a lapon belüli eltoláshoz? Rajzolja fel a virtuális címek tagozódását!

(b) Mekkora a teljes laptábla mérete?

(c) Hány lap fér a fizikai memóriába?

(d) Rajzolja fel a laptábla azon tartományát, ami az `A00000` és az `A03FFF` közé eső virtuális címek címfordításához kell! A laptábla bejegyzésekben tüntesse fel a virtuális tárkezelés működéséhez elengedhetetlen mezőket!

(e) Helyezze el a laptáblában az alábbi összerendeléseket:

- Az `A00` lap a fizikai memóriában a `15`-ös keretben található
- Az `A01` lap a háttértáron van
- Az `A02` lap a fizikai memóriában a `3`-mas keretben található
- Az `A03` lap a háttértáron van

(f) Feltéve, hogy nincs laphiba, hány memóriaművelet kell a címfordításhoz

- TLB találat esetén?
- TLB hiba esetén?

**3. Feladat** Legyenek a virtuális címek 16 bitesek, a fizikai címek 13 bitesek, a lapméret legyen $2^{10} = 1$ kB méretű. Használjunk 3 szintű hierarchikus laptáblát, melyben a laptáblák mérete a hierarchia minden szintjén egyforma. A laptábla bejegyzések 16 bitesek. A feladatok a következők:

(a) A virtuális címekben hány bit tartozik a lapon belüli eltoláshoz és hány a különböző szintű laptáblák indexeléséhez? Rajzolja fel a virtuális címek tagozódását!

(b) Mekkora az összes laptábla összegzett mérete?

(c) Optimális körülmények között mennyi a laptáblák által elfoglalt memória minimális mérete? Mennyivel kisebb ez, mintha egyszintű laptáblát használnánk?

(d) Hány lap fér a fizikai memóriába?

(e) Rajzolja fel az összes érintett laptáblát, melyek az `1001000000000000` és az `1001111111111111` közé eső virtuális címek címfordításához kell! A laptábla bejegyzésekben tüntesse fel a virtuális tárkezelés működéséhez elengedhetetlen mezőket!

(f) Helyezze el a laptáblában az alábbi összerendeléseket:

- Az `100100` lap a fizikai memóriában a `7`-es keretben található
- Az `100101` lap a háttértáron van
- Az `100110` lap a fizikai memóriában a `3`-mas keretben található
- Az `100111` lap a háttértáron van

(g) Feltéve, hogy nincs laphiba, hány memóriaművelet kell a címfordításhoz

- TLB találat esetén?
- TLB hiba esetén?

**4. Feladat** Legyenek a virtuális címek 16 bitesek, a fizikai címek 13 bitesek, a lapméret legyen $2^{10} = 1$ kB méretű. Használjunk inverz laptáblát, melyben a hash függvény értéke a virtuális cím 10.-13. bitje által meghatározott szám. A laptáblabejegyzések mérete: 16 bit. A feladatok a következők:

(a) A virtuális címekben hány bit tartozik a lapon belüli eltoláshoz és hány a lapok azonosításához? Rajzolja fel a virtuális címek tagozódását!

(b) Hány lap fér a fizikai memóriába?

(c) Mekkora a teljes laptábla mérete? Mennyivel kisebb ez, mintha egyszintű laptáblát használnánk? Mekkora a hash mutató tábla mérete?

(d) Rajzolja fel a teljes hash mutató táblát és a teljes laptáblát! A laptábla bejegyzésekben tüntesse fel a virtuális tárkezelés működéséhez elengedhetetlen mezőket!

(e) Helyezze el a laptáblában az alábbi összerendeléseket (itt most fontos, hogy ebben a sorrendben!):

- Az `100100` lap a fizikai memóriában a 7-es keretben található
- Az `100101` lap a háttértáron van
- Az `100110` lap a fizikai memóriában a 3-mas keretben található
- Az `100111` lap a háttértáron van
- A `010100` lap a fizikai memóriában a 5-ös keretben található

(f) Feltéve, hogy nincs laphiba, hány memóriaművelet kell a legjobb és a legrosszabb esetben címfordításhoz

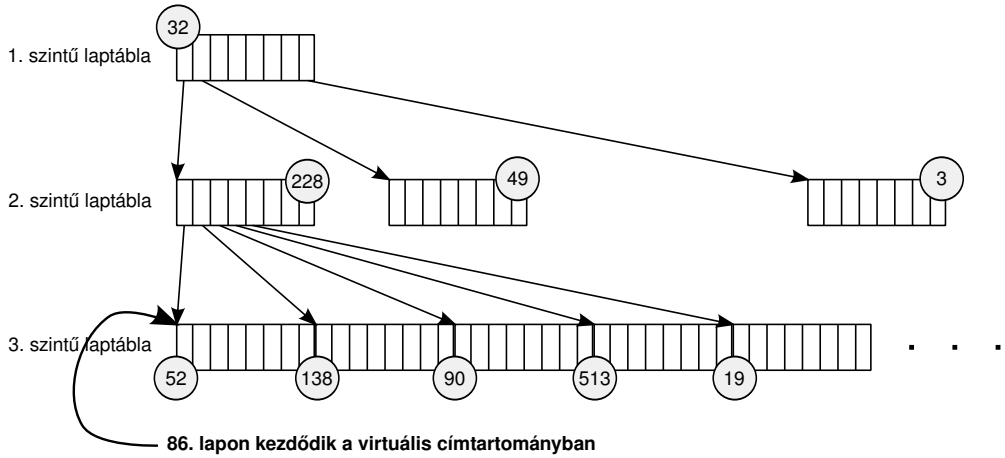
- TLB találat esetén?
- TLB hiba esetén?

Mikor áll elő a legrosszabb eset?

**5. Feladat** Egy processzor használjon 4 bejegyzéses TLB-t a címfordítás gyorsítására. A TLB egy adott pillanatban a következő két érvényes összerendelést tartalmazza:

| Valid | Lapszám | Keretszám |
|---|---|---|
| 1 | 88 | 90 |
| 1 | 1399 | 843 |
| 0 | ? | ? |
| 0 | ? | ? |

A laptábla legyen virtualizált, az alsó szint a 86. lapon kezdődik, egy szelete az alábbi ábrán látható (körökbe írt számok jelzik, hogy a laptáblák melyik keretben vannak):




Legyenek a virtuális címek 42 bitesek, az eltolás mező legyen 12 bites, a laptáblák minden szinten 1024 bejegyzést tartalmaznak, minden bejegyzés mérete 4 byte. A lapok mérete 4 kB. A fizikai címek 32 bitesek.

Hányszor kell a fizikai memóriához nyúlni, amikor az alábbi lapok címfordítását végzi a processzor? Melyik keret hányadik byte-ját és milyen sorrendben kell kiolvasni a fizikai memóriából a címfordítás kedvéért? Tételezzük fel, hogy minden érintett lap a fizikai memóriában van!

(a) 1399
(b) 2064
(c) 4101

Számítógép Architektúrák

# TLB és cache memória feladatok

**1. Feladat** Egy processzor használjon 4 bejegyzéses TLB-t a címfordítás gyorsítására, A TLB állapota egy adott pillanatban legyen a következő:

| Valid | Lapszám | Keretszám | Kor |
|---|---|---|---|
| 1 | 11 | 12 | 1 |
| 1 | 7 | 4 | 2 |
| 1 | 3 | 6 | 3 |
| 0 | 4 | 9 | 4 |

A TLB teljesen asszociatív, és ideális LRU algoritmust használ, a táblázat utolsó oszlopában tartja nyilván, hogy a bejegyzést milyen régen használta utoljára.

A laptábla legyen egyszerű, egyszintes laptábla, melynek tartalma legyen a következő:

| Valid | Keretszám |
|---|---|
| 1 | 5 |
| 0 | Háttértár |
| 0 | Háttértár |
| 1 | 6 |
| 1 | 9 |
| 1 | 11 |
| 0 | Háttértár |
| 1 | 4 |
| 0 | Háttértár |
| 0 | Háttértár |
| 1 | 3 |
| 1 | 12 |

A futó program az alábbi címfordításra váró virtuális címeket hivatkozza (ebben a sorrendben):

- 4095, 31272, 15789, 15000, 7193, 4096, 8912

Laphiba esetén az új lap az addig előforduló legnagyobb keretnél eggyel nagyobb sorszámú keretbe kerül.

A feladatok a következők:

(a) Mi lesz a TLB végső állapota, ha a lapméret 4 kB? Minden címfordításhoz adja meg, hogy TLB találat volt-e (T), TLB hiba volt-e (H), illetve, hogy laphiba volt-e (LH)?

(b) Mi lesz a TLB végső állapota, ha a lapméret 16 kB? Minden címfordításhoz adja meg, hogy TLB találat volt-e (T), TLB hiba volt-e (H), illetve, hogy laphiba volt-e (LH)?

(c) Értékelje, hogy a nagyobb lapmérettel mit nyertünk, illetve (ami a feladatból nem feltétlenül látszik), mit vesztünk?

**2. Feladat** Vegyünk egy 256 byte méretű cache-t 64 byte-os blokkmérettel. A cache kezdetben csupa érvénytelen blokkot tartalmaz.

Egy program az alábbi memóriablokkokról olvas (ebben a sorrendben):

- 1, 3, 8, 4, 3, 6, 8, 1

Adja meg a cache hibák számát és a cache végső tartalmát LRU algoritmus mellett

(a) direkt leképzés

(b) teljesen asszociatív szervezés

(c) két utas asszociatív szervezés

esetén.

**3. Feladat** Vegyünk egy 32 kB méretű cache-t 64 byte-os blokkmérettel. A CPU rendelkezzen 32 bites fizikai, 48 bites virtuális címekkel.

4-utas asszociatív szervezés mellett hány darab és hány bites komparátor dolgozik egy keresés során, ha

(a) Fizikailag indexelt cache-t használunk, fizikai tag-ekkel

(b) Virtuálisan indexelt cache-t használunk, virtuális tag-ekkel

**4. Feladat** Vegyünk egy 512 byte méretű cache-t 64 byte-os blokkmérettel. A cache fizikai címeket használ mind az indexeléshez, mind a tag-ekhez. A fizikai címek 16 bit szélesek.

Egy program az alábbi memóriacímekről olvas (ebben a sorrendben):

- 13, 136, 490, 541, 670, 74, 581, 980

(a) Adja meg a fenti címekhez tartozó "tag", "index" és "eltolás" mezőket

- teljesen asszociatív szervezés
- direkt leképzés
- két utas asszociatív szervezés

esetén.

(b) Adja meg a cache végső tartalmát mind a 3 esetben, LRU algoritmus mellett! (A cache kezdetben csupa érvénytelen blokkot tartalmaz)

**5. Feladat** Vegyünk egy 1 kB méretű cache-t 64 byte-os blokkmérettel. A cache kezdetben csupa érvénytelen blokkot tartalmaz. Lefuttatjuk az alábbi programot:

```
short int t[32][32];
int sum = 0;

for (int i=0; i<32; i++)
    for (int j=0; j<32; j++)
        sum += t[i][j];
```

Feltételezések: a `short int` 2 byte-os, a `t` tömb blokkhatáron kezdődik, a két dimenziós tömb a memóriában sor-folytonosan helyezkedik el, a cache direkt leképzést használ. Az i,j változó regiszterekben van tárolva, a cache-t nem terhelik.

(a) Hány cache hibát vált ki a fenti algoritmus? Számolja ki a cache hiba-arányt!

(b) Hány cache hibát vált ki a fenti algoritmus, ha megcseréljük a két ciklust? Számolja ki a cache hiba-arányt!

(c) Mekkora cache-re lenne szükség, hogy a megcserélt for ciklusokkal is ugyanolyan hibaarányt kapjunk, mint az eredeti algoritmussal?

# Számítógép Architektúrák
# Pipeline feladatok

**1. Feladat.** Legyen adott az alábbi utasítás sorozat:

```
i1:  R0 ← MEM [R1+8]
i2:  R2 ← R0 * 3
i3:  R3 ← MEM [R1+12]
i4:  R4 ← R3 * 5
i5:  R0 ← R2 + R4
```

Az utasítássorozatot egy, az órán tanult 5 fokozatú pipeline-t használó processzor hajtja végre. A feladatok a következők:

(a) Azonosítsa az összes előforduló adat-egymásrahatást!

(b) Adja meg az utasítássorozat ütemezését (melyik utasítás mikor melyik fázisban van)! Ha szünetet kell beiktatni, jelezze, hogy mi az oka! hazsnálja az alábbi jelöléseket:

- $A^*$: a szünet oka adategymásrahatás
- $F^*$: a szünet oka feldolgozási egymásrahatás
- $P^*$: a szünet oka procedurális egymásrahatás

(c) Minden egyes utasításnál jelölje, hogy amikor az EX fázisba ér, szükség van-e forwarding-ra! Ha igen, adja meg, hogy melyik pipeline regiszterből kell a a szükséges értéket visszacsatolni!

(d) Ha egységnyi ideig tart minden fázis, mennyivel gyorsabb a pipeline-al való végrehajtás, mint anélkül?

(e) Rendezze át az utasítássorozatot úgy, hogy az a lehető leggyorsabban fusson le, gyorsabban, mint eredetileg!

**2. Feladat.** Legyen adott az alábbi utasítás sorozat:

```
i1:  R5 ← MEM [R3+24]
i2:  R6 ← MEM [R4+16]
i3:  R7 ← R6 + R5
i4:  R8 ← R6 - R5
i5:  R5 ← R7 * R8
i6:  R4 ← R4 + 4
```

Az utasítássorozatot egy, az órán tanult 5 fokozatú pipeline-t használó processzor hajtja végre. A feladatok a következők:

(a) Azonosítsa az összes előforduló adat-egymásrahatást!

(b) Adja meg az utasítássorozat ütemezését (melyik utasítás mikor melyik fázisban van)! Ha szünetet kell beiktatni, jelezze, hogy mi az oka! hazsnálja az alábbi jelöléseket:

- $A^*$: a szünet oka adategymásrahatás
- $F^*$: a szünet oka feldolgozási egymásrahatás
- $P^*$: a szünet oka procedurális egymásrahatás

(c) Minden egyes utasításnál jelölje, hogy amikor az EX fázisba ér, szükség van-e forwarding-ra! Ha igen, adja meg, hogy melyik pipeline regiszterből kell a a szükséges értéket visszacsatolni!

(d) Ha egységnyi ideig tart minden fázis, mennyivel gyorsabb a pipeline-al való végrehajtás, mint anélkül?

(e) Rendezze át az utasítássorozatot úgy, hogy az a lehető leggyorsabban fusson le, gyorsabban, mint eredetileg!

**3. Feladat.** Legyen adott az alábbi utasítás sorozat:

```
i1:  D0 ← D1 * D2
i2:  D3 ← D0 + D5
i3:  MEM [R0+4] ← D3
i4:  MEM [R0+12] ← D0
```

Az utasítássorozatot egy, az órán tanult pipeline-t használó processzor hajtja végre. A D0-D5 regiszterek lebegőpontos számokat, az R0 egész számot tárol. A lebegőpontos összeadás késleltetése 4, a lebegőpontos szorzásé 7, minden egyéb egész műveleté 1, az iterációs idő minden esetben 1. Minden elképzelhető forwarding utat használhat!

(a) Azonosítsa az összes előforduló adat-egymásrahatást!

(b) Adja meg az utasítássorozat ütemezését (melyik utasítás mikor melyik fázisban van)! Ha szünetet kell beiktatni, jelezze, hogy mi az oka! hazsnálja az alábbi jelöléseket:

- $A^*$: a szünet oka adategymásrahatás
- $F^*$: a szünet oka feldolgozási egymásrahatás
- $P^*$: a szünet oka procedurális egymásrahatás

(c) Rendezze át az utasítássorozatot úgy, hogy az a lehető leggyorsabban fusson le, gyorsabban, mint eredetileg!

(d) Adja meg az átrendezett utasítássorozat ütemezését is, a szünetek okának megjelölésével!

**4. Feladat.** Legyen adott az alábbi utasítás sorozat:

```
i1:  D2 ← D0 * D1
i2:  MEM [R0+0] ← D2
i3:  D2 ← D0 + D1
i4:  MEM [R0+8] ← D2
```

Az utasítássorozatot egy, az órán tanult pipeline-t használó processzor hajtja végre. A D0-D5 regiszterek lebegőpontos számokat, az R0 egész számot tárol. A lebegőpontos összeadás késleltetése 4, a lebegőpontos szorzásé 7, minden egyéb egész műveleté 1, az iterációs idő minden esetben 1. Minden elképzelhető forwarding utat használhat!

(a) Azonosítsa az összes előforduló adat-egymásrahatást!

(b) Adja meg az utasítássorozat ütemezését (melyik utasítás mikor melyik fázisban van)! Ha szünetet kell beiktatni, jelezze, hogy mi az oka! hazsnálja az alábbi jelöléseket:

- $A^*$: a szünet oka adategymásrahatás
- $F^*$: a szünet oka feldolgozási egymásrahatás
- $P^*$: a szünet oka procedurális egymásrahatás

Számítógép Architektúrák

# Pipeline feladatok 2.

## 1. Feladatok a regiszter átnevezés témakörében

### 1.1. Feladat

Legyen adott az alábbi utasítás sorozat:

```
i1:  D2 ← MEM [R0+0]
i2:  D2 ← D0 * D2
i3:  D8 ← D4 / D2
i4:  D4 ← MEM [R1+0]
i5:  D6 ← D0 + D4
i6:  D8 ← D8 - D6
i7:  MEM [R1+8] ← D8
```

A feladatok a következők:

(a) Azonosítsa az előforduló WAW és WAR adat-egymásrahatásokat!

(b) Szüntesse meg a WAW és WAR adat-egymásrahatásokat regiszter átnevezés segítségével! A regiszter átnevezést végezze el minden utasításra szisztematikusan, ott is, ahol nem lenne rá szükség! A szükséges lebegőpontos fizikai regisztereket jelölje `T0, T1, T2, ...` stb.!

### 1.2. Feladat

Legyen adott az alábbi utasítás sorozat:

```
i1:  D1 ← D2 - D3
i2:  D4 ← D1 + D2
i3:  D6 ← D4 + D1
i4:  D0 ← D2 / D6
i5:  D1 ← MEM [R0+0]
i6:  D4 ← D1 + D2
i7:  D6 ← D4 + D1
```

A feladatok a következők:

(a) Azonosítsa az előforduló WAW és WAR adat-egymásrahatásokat!

(b) Szüntesse meg a WAW és WAR adat-egymásrahatásokat regiszter átnevezés segítségével! A regiszter átnevezést végezze el minden utasításra szisztematikusan, ott is, ahol nem lenne rá szükség! A szükséges lebegőpontos fizikai regisztereket jelölje `T0, T1, T2, ...` stb.!

## 2. Feladatok az elágazásbecslés témakörében

### 2.1. Feladat

A processzor a program végrehajtása közben az alábbi sorrendben, az alábbi címeken lévő feltételes ugró utasításokat értékeli ki (a zárójelben "T" jelzi, ha az ugrás ténylegesen bekövetkezett, és "N", ha nem):

- 464 (T), 543 (N), 777 (N), 543 (N), 777 (N), 464 (T), 777 (N), 464 (T), 543 (T)

A processzor egyszerű, 2 bites állapotgépre alapozott dinamikus elágazásbecslést használ, és 4 ugró utasítást tud követni. A feladat a következők:

(a) Hány bitet foglal a PHT?

(b) Írja fel a PHT tartalmát minden egyes feltételes ugrás végrehajtása után! Jelezze, hogy a becslő eltalálta-e az ugrás kimenetelét! Kezdetben minden állapotgép értéke 1.

## 2.2. Feladat

A processzor a program végrehajtása közben az alábbi sorrendben, az alábbi címeken lévő feltételes ugró utasításokat értékeli ki (a zárójelben "T" jelzi, ha az ugrás ténylegesen bekövetkezett, és "N", ha nem):

- 464 (T), 543 (N), 777 (N), 543 (N), 777 (N), 464 (T), 777 (N), 464 (T), 543 (T)

A processzor korrelált elágazásbecslést alkalmaz globális, 2 bites állapotgépeket tartalmazó PHT-vel, úgy, hogy az utolsó 2 feltételes ugrás kimenetelét tudja nyomon követni.

A feladat a következők:

(a) Hány bitet foglal a PHT?

(b) Írja fel a PHT tartalmát minden egyes feltételes ugrás végrehajtása után! Jelezze, hogy a becslő eltalálta-e az ugrás kimenetelét! Kezdetben minden állapotgép értéke 1, a globális ugrási előzmény regiszter értéke binárisan 11.

## 2.3. Feladat

Egy processzorban az ugrási címek becslésére BTB-t használnak. A BTB találati arány (vagyis hogy egy ugrási utasításra egyáltalán tartalmaz becslést) 90%. BTB találat esetén a becslés pontossága 90%. Ha az ugró utasítás nincs a BTB-ben, 4 ciklus, ha benne van, de rossz a becslés, 3 ciklusnyi késleltetést szenved a futó program végrehajtása. A programban a feltételes elágazások aránya 15%. Most feltesszük, hogy semmi más (semmilyen más egymásrahatás) nem lassítja a program futását, és ha nincs ugrás, a pipeline minden ciklusban végezni tud egy utasítás végrehajtásával.

(a) Átlagosan hány ciklusonként végez a pipeline egy utasítás végrehajtásával?

(b) Hogy viszonyul az eredmény a BTB nélküli esethez, amikor minden ugró utasítás egységesen 2 szünet beiktatását vonja maga után?

Számítógép Architektúrák

# VLIW/EPIC technikák gyakorlat

**1. Feladat.** Legyen adott az alábbi utasítássorozat:

```
i1:  R2 ← MEM [R0+0]
i2:  R3 ← R0 * R2
i3:  R8 ← R4 / R3
i4:  R5 ← MEM [R1+8]
i5:  R6 ← R2 + R5
i6:  R9 ← R5 / R6
i7:  R10 ← R6 * R9
```

Az utasítássorozatot lefordítjuk egy olyan VLIW processzorra, melyben az alábbi utasítások helyezhetők el egy utasításcsoportban:

- 2 db memóriaművelet (végrehajtási idő: 3 órajel)
- 2 db egész aritmetikai művelet, vagy ugró utasítás (végrehajtási idő: 1 órajel)

A feladatok a következők:

(a) Ütemezze a megadott utasítássorozatot a megadott VLIW processzoron! Határozza meg az utasításcsoportok tartalmát, és a csoportok végrehajtásának idejét!

(b) Mennyivel gyorsabb az így kapott program, mintha hagyományos 1-utas processzoron futtatnánk (pipeline nélkül)?

(c) Hány utasításcsoportot állít elő a fordító klasszikus, és hányat dinamikus VLIW architektúra esetén?

**2. Feladat.** Legyen adott az alábbi utasítássorozat:

```
i1:  R1 ← R2 - R3
i2:  R4 ← MEM [R1]
i3:  R6 ← R4 + R1
i4:  R0 ← R2 / R6
i5:  R7 ← MEM [R9]
i6:  R8 ← R7 + R2
i7:  R7 ← R7 + R6
```

Az utasítássorozatot lefordítjuk egy olyan VLIW processzorra, melyben az alábbi utasítások helyezhetők el egy utasításcsoportban:

- 2 db memóriaművelet (végrehajtási idő: 3 órajel)
- 2 db egész aritmetikai művelet, vagy ugró utasítás (végrehajtási idő: 1 órajel)

A feladatok a következők:

(a) Ütemezze a megadott utasítássorozatot a megadott VLIW processzoron! Határozza meg az utasításcsoportok tartalmát, és a csoportok végrehajtásának idejét!

(b) Mennyivel gyorsabb az így kapott program, mintha hagyományos 1-utas processzoron futtatnánk (pipeline nélkül)?

(c) Hány utasításcsoportot állít elő a fordító klasszikus, és hányat dinamikus VLIW architektúra esetén?

**3. Feladat.** Adott az alábbi C kód:

```
if (a < 0) {
    b = 1;
    c = 2;
}
else {
    d = 3;
    e = 4;
}
```

A feladatok a következők:

(a) Írja át a fenti C kódot a tárgyban használt pszeudo-assembly nyelvre! Hány utasításra van szükség?

(b) Írja át a fenti C kódot a tárgyban használt pszeudo-assembly nyelvre, predikátumok alkalmazásával! Hány utasításra van szükség?

(c) Hány utasítást kell végrehajtani a predikátumokkal és az anélkül megvalósított kódban, ha

- az $a < 0$ az esetek 20%-ában teljesül?
- az $a < 0$ az esetek 80%-ában teljesül?

(d) Ideális esetben hány lépésben hajtható végre a fenti kód 4 végrehajtó egységgel rendelkező VLIW processzorral predikátumokkal és anélkül?