# Számítógép architektúrák alapjai

Készítette: Simon Péter [1]

2022. január 2.

[1] Hallgatói jegyzet Durczy Levente előadásai alapján

# Tartalomjegyzék

# 1. fejezet

# Alapfogalmak

## 1.1. Architektúra fogalma

A számítógép architektúra fogalmat először Amdahl, az IBM mérnőke használta először a 360-as család bejelentésekor. Definíciója szerint ez az a struktúra, amit a gépi kódú programozónak értenie kell, hogy helyes programot tudjon írni az adott gépre. Tehát a regiszterek, memória, utasításkészlet, címzési módok és utasításkódok összessége, mind logikai, mind hardveres szinten.

## 1.2. Számítási modellek

A számítási modell a számításra vonatkozó alapelvek egy absztrakciója. A számítási modelleket a következő absztrakciós jellemzőkkel írhatjuk le:

- min hajtjuk végre a számítást (általában adatokon - adat alapú)
- hogyan képezzük le a számítási feladatot
- milyen módon vezéreljük a végrehajtási sorrendet

### 1.2.1. A számítási modell, az architekrúra és a programnyelv kapcsolata

Egy számítógép tervezését a számítási modellel kell kezdeni, ami meghatározza, hogy mit szeretnénk csinálni. Ehhez szükség van egy specifikációs eszközre, amit a programnyelv képvisel (pl. Neumann modell megvalósítási eszköze a BASIC, Fortran). Ezután jön az architektúra, ami a számítási modell implementációs eszköze, a "vas". Ez hajtja végre az adott programnyelven definiált feladatokat.

### 1.2.2. Számítási modellek csoportosítása

**Számítási modelljük szerint**

- szekvenciális
- párhuzamos

**Vezérlés meghajtása szerint**

- vezérlés meghajtott
- adat meghajtott
- igény meghajtott

**Probléma leírása szerint**

- procedurális
- deklaratív

Első sorban aszerint különböztetjük meg őket, hogy min hajtjuk végre a számítást. Az adatalapú modellek:

- Neumann modell
- adatfolyam modell
- applikatív modell (igénymeghajtott)

Az adatalapú modelleken kívül léteznek még objektum alapú, predikátum logika alapú, tudás alapú és hibrid modellek. A mai processzorokban a Neumann és az adatfolyam modellek keverednek.

### 1.2.3. Adatalapú modellek közös tulajdonságai

- az adatok általában típussal rendelkeznek (pl. 16 bit int) - vannak elemi és összetett adattípusok
- a típus meghatározza az adat értelmezési tartományát, értékkészletét és az elvégezhető műveleteket

### 1.2.4. Neumann modell

A Neumann-elvű számítógépek a számításokat adatokon hajtják végre, amiket egy változó értékű változókészlet képvisel. A végrehajtási sorrend vezérlés meghajtott, tehát van egy statikus utasításszekvencia, amit egy speciális regiszter biztosít (program counter). A program counter egy inkrementálódó változó, mindig a végrehajtandó utasításra mutat. A végrehajtási sorrendtől vezérlési feladatokat ellátó utasításokkal lehet eltérni (pl. jump, if).

A Neumann elv követelményrendszere előírja változók létrehozását, adatmanipulációs és vezérlés átadási utasítások deklarálását. Az ilyen nyelveket hívjuk imperatív (parancs) programnyelveknek (pl. C, Pascal, Assembly).

Ezeket a követelményeket az architektúra kielégíti, pl. lehetővé teszi, hogy a memóriában elhelyezkedő változók korlátlan számban módosíthatók legyenek a program futása során. Ezen kívül biztosítja a megfelelő regisztereket az adatoknak és speciális regisztereket mint pl. program counter.

Az adatok és az utasítások a memóriában helyezkednek el. A számítási feladat műveletek elemi műveletek sorozataként értelmezhető. Egy számítási feladat leképezhető adat manipuláló utasítások sorozatával. Az adat manipuláló utasítások az utasítások sorrendjében vannak végrehajtva, ezért ez egy vezérlés meghajtott modell. A vezérlést a program counter biztosítja, a sorrendet a programozó határozza meg. Az explicit vezérlés átadó utasításokkal lehet eltérni az implicit szekvenciától.

Következmények:

- előzmény érzékenység: mivel az adatok változhatnak bármikor a végrehajtás során, a végrehajtás sorrendje nem mindegy
- alapvetően szekvenciális végrehajtást biztosít
- egyszerűen implementálható
- az adatmanipuláló utasítások nem szándékos állapotmódosulást okozhatnak (pl. overflow) - ezeket mellékhatásoknak hívjuk, kezelni kell őket

### 1.2.5. Adatfolyam modell

A számítást itt is adatokon hajtjuk végre, de:

- az adatokat bemenő adathalmaz képviseli
- egyszeres értékadás lehetséges
- a megoldandó feladatot adatfolyam gráffal és input adatok halmazával képezzük le (a gráfban a csomópontok a szakosodott végrehajtó egységek)
- szakosodott végrehajtó egységeket használ
- a végrehajtást az adat vezérli - adatvezérelt, azaz a szükséges adatok rendelkezésre állásakor azonnal működésbe lép a végrehajtó egység
- az adat meghajtott program utasításai semmilyen szempontból nem rendezettek
- az adatokat utasításon belül tároljuk és nem az operatív tárban
- magas a kommunikációs és szinkronizációs igénye

### 1.2.6. Applikatív modell

A számítási feladatot egy komplex függvény formájában adjuk meg. Ez is adatalapú modell, de deklaratív jellegű, tehát valamennyi, az adott probléma megoldásához szükséges tényt és relációt deklarálunk. Mindezt egy végrehajtó mechanizmus (a függvény) feldolgozza. A vezérlés igény meghajtott (lazy, lusta modell).

## 1.3. Az architektúráról bővebben

A 70-es években Bell és Newell kibővítette az architektúra definícióját és négy szintet határoztak meg:

- PMS - processor, memory, switches: a számítógépek globális leírása
- programozási szint:
  - magas szint
  - alacsony szint
- logikai tervezési szint
- áramköri szint

Az architektúra másik megfogalmazásban a külső jellemzők, a belső felépítés és a működés együttesét jelenti. Megkülönböztetünk logikai és fizikai architektúrát, valamint ezeken belül processzor és számítógép szinteket.

### 1.3.1. ISA

A processzor szintű logikai architektúra az ISA - Instruction Set Architecture. Ez írja le az utasításkészletet, pl. x86. Az ISA komponensei:

- adattér
- adatmanipulációs fa
- állapottér
- állapot műveletek

### 1.3.2. Fizikai architektúra

A számítógép szintű fizikai architektúra elemei:

- processzor
- memória
- buszrendszer

A processzor szintű fizikai architektúra négy részre bontható:

- műveletvégző egység (ALU)
- vezérlő
- I/O
- megszakításrendszer

Az I/O és a megszakításrendszer a más rendszerekkel összekapcsoló illesztő eszközök.

# 2. fejezet

# Adattér

## 2.1. Az adattér fogalma

Az adattér egy olyan tér, mely biztosítja az adatok tárolását oly módon, hogy azok a CPU által közvetlenül manipulálhatók legyenek. Az adattér közvetlenül címezhető a processzor által.

## 2.2. Típusai

Az adattér két típusát különböztetjük meg:

- memóriatér
- regisztertér

## 2.3. Memóriatér

A memóriateret leginkább a mérete jellemzi. A címzéséhez címbuszra van szükség, ennek szélessége meghatározza a memóriatér maximális méretét. A címtérnél megkülönböztetjük a modell címterét és a valós címteret, mivel az adott installáció nem feltétlen éri el az elméleti modell méretét.

### 2.3.1. Virtuális memória

A 60-as években a kis memóriák és az elméleti és valós memóriatér eltérő mérete miatt megjelent a virtuális memória. Alap koncepciói:

- két különböző memóriatér létezik: amit a programozó lát (virtuális memória) és a CPU által látott fizikai memória
- létezik olyan, a felhasználó számára transzparens folyamat, amely a program futása közben az éppen nem használt adatokat a valós memóriából a virtuális memóriába mozgatja, majd szükség esetén visszateszi
- létezik olyan (egyirányú) folyamat, amely a virtuális memória címeket dinamikusan, tehát futási időben valós címekké alakítja

## 2.4. Regisztertér

A regisztertér az adattér nagy teljesítményű, általában viszonylag kis része. Általában nem része a címtérnek.

### 2.4.1. Típusai

- egyszerű regiszterkészlet
- adattípusonként különböző regiszterek
- többszörös regiszterkészlet

### 2.4.2. Egyszerű regiszterkészlet

Az egyszerű regiszterkészlet típusai:

- egyetlen regiszter (akkumulátor), hátránya, hogy nagyon lassú
- több, dedikált adatregiszter
- univerzális regiszterkészlet, jelentős teljesítmény növekedést eredményezett, a programozó szabadon felhasználhatja a regisztereket (pl. a gyakran használt változók folyamatosan a regiszterben maradhattak)
- stack regiszterek

#### Stack regiszterek

A stack regisztereknél egy stack pointer (SP) mutat a "zsák" tetejére (mindig az utoljára betöltött adat). Előnye, hogy nem kell címezni, így egyszerű, rövid utasításokkal kezelhető és gyors. Hátránya viszont, hogy mindig csak a tetejéhez férünk hozzá, így az operandus kiolvasás csak szekvenciálisan történhet. A szekvenciális kiolvasás lassú, szűk keresztmetszetet jelent sok adatnál.

### 2.4.3. Adattípusonként különböző regiszterek

Itt különféle adattípusokhoz különféle regiszterek állank rendelkezésre, elsősorban lebegőpontos adatoknál. A lebegőpontos regisztereknél külön van bontva a mantissza és a karakterisztika, mivel mindkettőn műveletet kell végezni, így lehetőséget ad a párhuzamosításra a külön tárolás. Az Intel 1998-ban bevezette még a SIMD adattípust, amit multimédiás alkalmazásokhoz használtak.

### 2.4.4. Többszörös regiszterkészlet

Ez a legfejlettebb megoldás, egymásba ágyazott eljárások gyorsítására szolgál. Ehhez szükséges a kontextus fogalmának bevezetése. A kontextus a regisztertér állapota az állapottérrel együtt.

Az eljárások közötti váltásoknál szükséges a kontextusok közötti váltás, amit a kontextuskapcsoló végez. Ha a kontextusok közötti váltásnál a kontextust a memóriatérből kell betölteni, nagyon lassú lesz, ha viszont regiszterek között kell csak váltani, akkor gyors. Ezért vezettek be több regiszterkészletet, a cél minden kontextus számára különálló regiszterkészlet biztosítása. A programozó ezek közül csak egyet lát, a kontextus váltásokat a processzor kezeli. A megvalósításhoz szükség van még egy általános regiszterkészletre, ami a regiszterek közti paraméter átadást biztosítja.

**Típusai**

- több, egymástól független regiszterkészlet, hátránya, hogy a paraméter átadás csak az operatív táron keresztül történhet, ami lassítja az eljárást
- átfedő regiszterkészlet, egymásba ágyazott eljárásokhoz dolgozták ki, lényege, hogy egy regisztert három részre bontottak: ins, locals, outs. Úgy alakították ki, hogy az egyik regiszter kimenő területe ugyanazon a címen helyezkedik el, mint a másik bemenője, így nagyon könnyű az operátor átadás. Hátránya, hogy merev a struktúra és hogy a regiszterkészletek száma korlátozza a regisztertérben tartható kontextusokat.
- stack cache: 1982-ben vezették be, a mai processzorokban is hasonló struktúrákat alkalmaznak. A stack és a közvetlen elérésű cache kombinálása. Kezelése a compiler feladata, ami minden eljáráshoz hozzárendel egy regiszterkészletet. A hozzárendelt regiszterkészlet az aktiválási rekord, aminek az első elemére a stack pointer mutat. A regiszterek nem csak a stack pointeren kersztül érhetők el, hanem közvetlenül is, a displacement pointerrel. Bizonyos korlátok között az aktiválási rekord bármilyen hosszú lehet, tehát a kiosztás rugalmas. Előnye, hogy nincsenek üres helyek, így nincsenek fölöslegesen lefoglalt regiszterek és nincs túlcsordulás se.

# 3. fejezet

# Adatmanipulációs fa

## 3.1. Az adatmanipulációs fa fogalma

Az adatmanipulációs fa megmutatja a potenciális adatmanipulációs lehetőségeket. Bizonyos részfái megmutatják egy konkrét implementáció adatmanipulációs lehetőségeit.

## 3.2. Részei

- 1. szint: adattípusok, azaz miket értelmezünk az adott architektúrában (pl. 1 byte FX, 2 byte FX, FP, stb.)
- 2. szint: műveletek szintje, azaz az adattípusokkal milyen műveletek végezhetők (pl. összeadás, kivonás, logikai műveletek)
- 3. szint: operandusok típusai (pl. két operandusos, három operandusos). Megkülönböztetünk memória, regiszter és akkumulátor operandus típusokat.
- 4. szint: címzési módok (pl. regiszter+displacement, pc+displacement, index regiszter+displacement, direkt, indirekt)
- 5. szint: gépi kód

## 3.3. Adattípusok

Megkülönböztetünk elemi és összetett adattípusokat. Elsősorban az elemi adattípusokkal foglalkozunk. Az összetett adattípusok elemiekből épülnek fel, ha különböző típusokból áll össze, akkor rekordnak hívjuk, ha azonos típusokból, akkor megkülönböztetünk vektor (1D tömb), tömb (többdimenziós), szöveg, verem, sor, lista, fa, halmaz adattípusokat.

### 3.3.1. Elemi addattípusok

Megkülönböztetünk numerikus, karakteres, logikai, pixel és még több adattípust.

### Numerikus adattípusok

- FX
- FP
- BCD (binárisan kódolt decimális)
  - pakolt (1 byte két helyiérték, 4 bit egy decimális szám)
  - pakolatlan (zónázott)

### FX adattípusok

Kódolás szerint megkülönböztetünk:

- egyes komplemens
- kettes komplemens
  - előjeles (1, 2, 4, 8, 16 byte)
  - előjel nélküli
- többletes kódolás

### FP adattípusok

Megkülönböztetünk:

- normalizált
  - hexára normalizált
  - binárisra normalizált
    - VAX
    - IEEE
      - 1x pontosságú (32 bites)
      - 2x pontosságú (64 bites)
      - kiterjesztett pontosságú (128 bites)
- nem normalizált

Általában a normalizáltat használják.

### Karakteres adattípusok

Megkülönböztetünk:

- EBDIC (8 bit, 60-as évek)
- ASCII
  - 7 bites (szabványos)
  - 8 bites (kiterjesztett)
- Unicode (2 byte)

**Logikai adattípusok**

Megkülönböztetünk:

- 1 byte
- 2 byte
- 4 byte
- változó hosszú

Példák: AND, OR eredményei. Általában 1 bit értékes (általában a legmagasabb helyiérték).

## 3.4. Műveletek

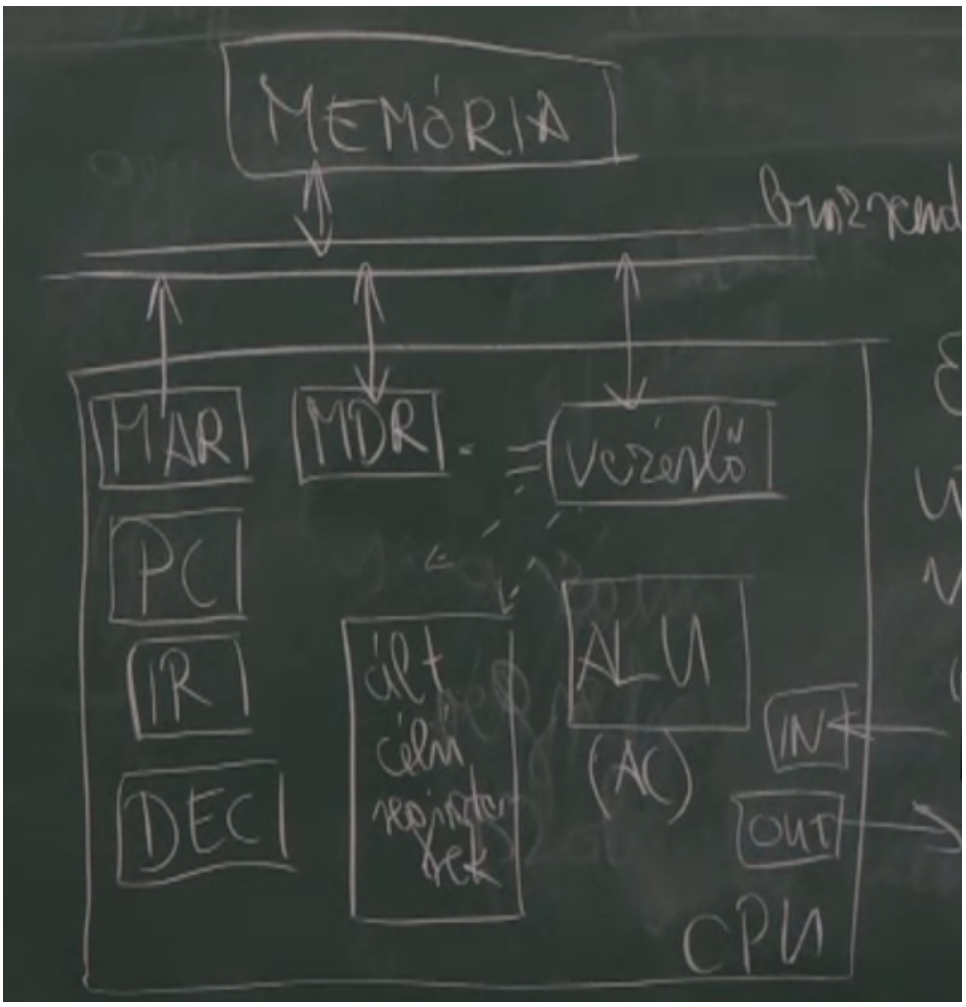
Az adatmanipulációs fa minden művelet esetén megállapítja, hogy milyen utasítás típusok vannak megengedve és milyen operandus típus választható. Az utasítás a számítógép által végrehajtható alapvető feladatok ellátására szolgáló elemi művelet leírása.

### 3.4.1. Utasítás végrehajtás menete

Egy gépi kódú utasítás általában két részből áll: MK, azaz műveleti kód és címrész. Az MK tartalmazza, hogy mit kell csinálni, a címrész pedig, hogy melyik címen lévő adattal. Pl.: ADD r3,r1,r2. Általánosan az utasítás végrehajtás lépései:

1. A processzor megnézi, hogy van-e megszakítás kérés, ha van, kiszolgálja azt
2. Ha nem történt megszakítás kérés, lehívja az utasítást
3. Végrehajtja az utasítást

A processzor főbb regiszterei láthatók a következő ábrán.




3.1. ábra. Egy CPU főbb regiszterei

Egy gépi kódú utasítás elemi műveletek sorozataként írható fel. Példa: ADD r1, r2 utasítás végrehajtása:

1. FETCH - az utasítást le kell hívni:
   (a) A processzor a program counter (PC) értékét betölti a memória címregiszterbe (MAR).
   (b) A címregiszterben lévő címen lévő értéket betölti a memória adatregiszterbe (MDR).
   (c) A memória adatregiszter tartalmát áttölti az utasításregiszterbe (IR)
   (d) A program counter értékét egy egységgel inkrementálja
2. Dekódolás és operandus betöltés
   (a) Az utasításregiszter tartalmát betölti a processzor a dekóderbe (DEC)
   (b) Az utasítás címrész tartalmát betölti a memória címregiszterbe
   (c) A memória címregiszter értékének címén található adatot betölti a memória adatregiszterbe
   (d) A memória adatregiszter tartalmát betölti az akkumulátorba
3. Művelet végrehajtás
   (a) A dekóder megfelelő (másik) címrésze bekerül a MAR-ba
   (b) A MAR értékének címén lévő adat bekerül az MDR-be
   (c) Az AC-be betölti az AC+MDR-t (elvégzi az összeadást)

4. Az eremény tárolása (STORE)
   (a) A dekóder címrésze (ugyanaz mint a dekódolásnál) bekerül a memória címregiszterbe
   (b) Az akkumulátor tartalmát áttölti az MDR-be
   (c) Az MDR tartalmát betölti az MAR által mutatott címre

Példa: JMP utasítás

1. IR-ből betölti a dekóderbe az utasítást
2. A dekóder címrészét betölti MAR-ba
3. A MAR által mutatott értékkel felülírja PC-t

### 3.4.2. Utasítások részei

- műveleti kód
- operandus
  - cél (dest op)
  - forrás (source op)

### 3.4.3. Utasítások típusai címek száma alapján

- 4 címes: cél, forrás 1, forrás 2, következő utasítás címe. Hátránya, hogy nehézkes és merev.
- 3 címes: cél, forrás 1, forrás 2. A következő utasítás címét helyettesítették az auto inkrementálódó PC-vel. RISC architektúrák használják. Előnye a szabadság és a párhuzamosítási lehetőség, hátránya, hogy bonyolultabbak és hosszabbak az utasítások.
- 2 címes: nincs cél operandus, csak forrás 1 és 2. Az eredmény felülírja a forrás 1-et. CISC processzorok használják (pl. x86). Előnye az egyszerűbb utasítás, hátránya, hogy felülírja az egyik forrás operandust.
- 1 címes: be kell tölteni az egyik operandust az akkumulátorba, majd a második operandussal módosítjuk AC tartalmát. Pl. ADD utasítás: LOAD[100], ADD[101]. Hátránya, hogy az utasítások száma nő, viszont az utasítások kisebbek és gyorsabban végrehajthatóak. Többszörös összeadásnál hasznos például.
- 0 címes: pl. NOP, CLEAR D (flag), PUSH, POP. A stack utasítások gyorsaságát is ez adja, nem kell külön megcímezni, hanem a stack pointer értékét használja. Előnye, hogy gyors, hátránya, hogy növelik az utasításkészletet.

A három címes utasítások mindhárom operandusa regiszter típusú, a két címeseknél viszont általában a második operandus lehet memória típusú is.

## 3.5. Operandus típusok

- AC - akkumulátor: gyors, de csak egy van belőle
- memória: nagy, hosszú címe van és viszonylag lassú
- regiszter: gyors, de korlátozott számú
- verem: nagyon gyors, de csak a tetejét látjuk, így sok adatot nem lehet benne tárolni
- immediate: nem egy címet adunk meg, hanem közvetlenül a programba írjuk be az operandus értékét. Nagyon gyors, de a módosításához módosítani kell a programot.

### 3.5.1. Architektúrák osztályozása operandus típusok szerint

- szabályos: csak egy fajta operandus típus engedélyezett (kivéve akkumulátor, mert abból csak egy van)
  - akkumulátor (akkumulátor-regiszter és akkumulátor-memória)
  - memória (memória-memória, memória-memória-memória)
  - regiszter (regiszter-regiszter, regiszter-regiszter-regiszter)
  - stack (stack-stack, stack-stack-stack)
- kombinált: a különböző operandus típusok megengedettek egy utasításon belül (pl. regiszter + memória) - nem homogén.

A szabályos architektúrák kezelése és tervezése könnyebb. Ilyenek pl. a RISC architektúrák (mint pl. ARM), amik 3 címes regiszter típusú operandusokkal dolgoznak. A memória eléréséhez két kivételes utasítás használható: LOAD és STORE.

Kombinált architektúra pl. a CISC architektúrák (mint pl. x86).

## 3.6. Címzési módok

A címzési mód maga a címszámítási algoritmus. Ezt 3, egymástól független elem kombinációja adja:

- címszámítás: jelzi, hogy abszolút, vagy relatív címzést használunk. Az abszolút címzés a teljes címet tartalmazza, a relatívnál pedig egy bázishoz képest számoljuk. A relatív gyorsabb, rövidebbek a címek, de deklarálni kell a bázis címet és a címszámítási algoritmust. Elsősorban a relatív címzés van használatban.
- cím módosítás (opcionális komponens): indexelés, auto inkrementálás, auto dekrementálás. Segítségével könnyebben meghatározható a következő operandus címe.
- deklarált (tényleges) cím meghatározása: azt jelenti, hogy a címet direkt vagy indirekt, illetve valós vagy virtuális címként interpretáljuk.

### 3.6.1. Relatív címszámítás

A CPU címtere nagyon nagy (4-64 TB), így abszolút címzéssel nagyon nagy címeket kéne kezelni. Ezért inkább a relatív címzést használják. Bázis címnek választható pl. a PC, a stack teteje, valamilyen index regiszter, stb.

### 3.6.2. Az indexelés és az index regiszter

Megkülönböztetünk egyedi és blokkos címzést, jellemzően blokkokat címzünk és töltünk be a gyorsítótárba. Ehhez szükség van egy index regiszterre, amiben az eltolás tárolódik. Több dimenziós blokkoknál több index regiszter van használatban.

## 3.7. Utasítás kód

Az adatmanipulációs fa legalsó szintje, gépi reprezentációnként különböző.

## 3.8. Állapottér

Az állapottér olyan, programból látható és nem látható (program transzparens) tárolókból áll, amelyek az adott programra vonatkozó állapotinformációkat hordozzák. A program transzparens információk a rendszerfunkciókhoz szükségesek, mint pl. virtuális memória kezelés és a megszakításkezelés. Az állapottér tehát feloszthető:

- transzparens
  - virtuális memória kezelés
  - megszakításkezelés
  - veremkezelés
- látható
  - PC
  - státusz indikátorok (flagek)
    - CC (Condition Code, IBM gépeknél): két bit, különböző állapotinfókat tartalmaz
    - univerzális állapotjelzők pl. carry, zero
    - adattípusonként különböző állapotjelzők pl. denormalizált szám vagy érvénytelen művelet (minden regiszterkészlethez definiáltak)
  - indexelés
  - címzési módok
  - debug

### 3.8.1. Flagek

Olyan kivételes események figyelésére és vezérlésére szolgálnak, melyek a program futása közben általánosságban jelenhetnek meg. Ilyen pl. a túlcsordulás vagy a nullával való osztás.

### 3.8.2. Állapot műveletek

Az állapotjelzőket speciális utasításokkal lehet manipulálni. Állapotműveletek PC esetén:

- inkrementálás
- dekrementálás
- felülírás (pl. egy utasításból vett címmel, ugróutasítással)

Flagek esetén:

- mentés
- beállítás
- reset
- load
- clear

# 4. fejezet

# Processzor szintű fizikai architektúra

## 4.1. Részei

- műveletvégző
- vezérlő
- I/O rendszer
- megszakításrendszer

A legfontosabbak a műveletvégző és a vezérlő részek, mivel ezek végzik a CPU két fő funkcióját (utasítás lehívás és végrehajtás).

## 4.2. Szinkron és aszinkron CPU-k

Kétféle CPU-t különböztetünk meg:

- aszinkron: nincs órajel, hanem az utasítás vége jelzés a következő utasítás megkezdésére. Előnye, hogy nincs holtidő, hátránya hogy az utasítás végének érzékeléséhez speciális áramkörök kellenek, ami viszonylag drága és az érzékelés is időt vesz igénybe.
- szinkron: a végrehajtást órajel vezérli

## 4.3. Műveletvégző egység

A műveletvégző egység tartalmazza a:

- regisztereket
- adatutakat
- kapcsolópontokat
- szűkebb értelemben vett ALU-t

### 4.3.1. Regiszterek

A műveletvégző egység tartalmaz látható és transzparens regisztereket. Láthatóból megkülönböztetünk univerzális és dedikált (pl. stack) regisztereket. A transzparens (rejtett) regiszterek általában az adatfeldolgozáshoz szükséges puffer regiszterek. A rejtett regiszterekre nem lehet hivatkozni, de számításba kell venni őket alacsony szintű programozásnál.

### 4.3.2. Adatutak

Ez nem adatbusz! Az adatbuszon értelmezett a címzés, adatutak esetén viszont nem beszélhetünk címzésről. Az adatutak a műveletvégző egység részeit kapcsolja össze, gyakorlatilag egy vezetékrendszer.

### 4.3.3. Kapcsolópontok

A regiszterkhez kapcsolópontokon keresztül csatlakoznak a vezetékek. A kapcsolók tranzisztorok, a regiszterek részei. A kimeneti kapcsoló három állású: zárt, nulla és egy. A bemeneti kapcsoló kétállású: nyitott és zárt. Az adatutakon egyszerre csak egy adat lehet, ezért a kapcsolópontok felelnek azért, hogy csak a megfelelő regiszter legyen nyitva. A vezérlő nyitja és zárja a kapcsolókat.

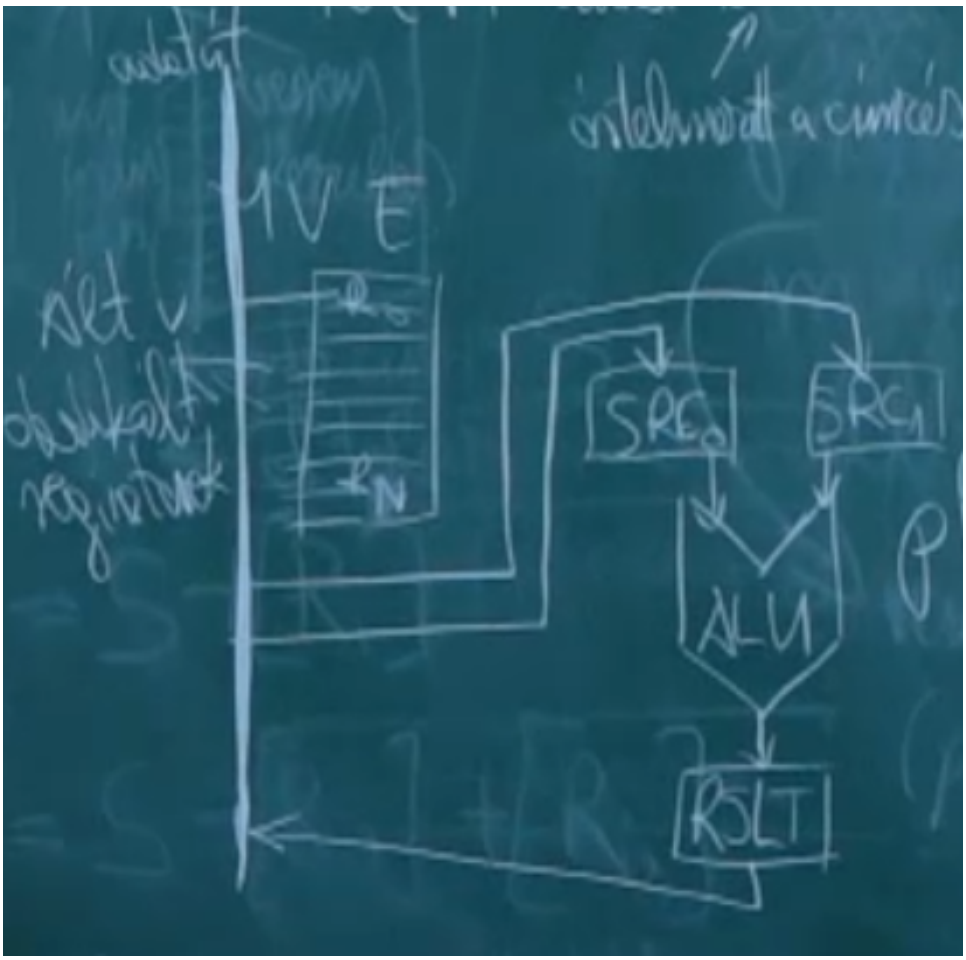
### 4.3.4. Csatolási módok

Az adatutakat csoportosíthatjuk csatolási mód szerint:

- egyutas
- kétutas
- háromutas

#### Egyutas csatolás

Előnye, hogy egyszerű és olcsó, hátránya, hogy lassú.




4.1. ábra. Egyutas adatút

Példa egyutas adatút működésére ADD r0,r1 művelet esetén:

1. Az R0 tartalmát be kell tölteni az egyik forrásregiszterbe, ezért a vezérlő megnyitja R0 és SRC0 kapcsolóit, hogy az adatúton keresztül a jel eljuthasson.
2. Ezután jön R1, itt R1 és SRC1 kapcsolóit nyitja a vezérlő.
3. Szinkronizáltan, órajelre megnyílnak SRC0 és SRC1 kimenő kapcsolói, a forrás operandusok eljutnak az ALU-ba, ahol megtörténik az összeadás.
4. Az ALU-ból bekerül az adat az eredményregiszterbe.
5. Végül az eredmény visszajut R0-ba.

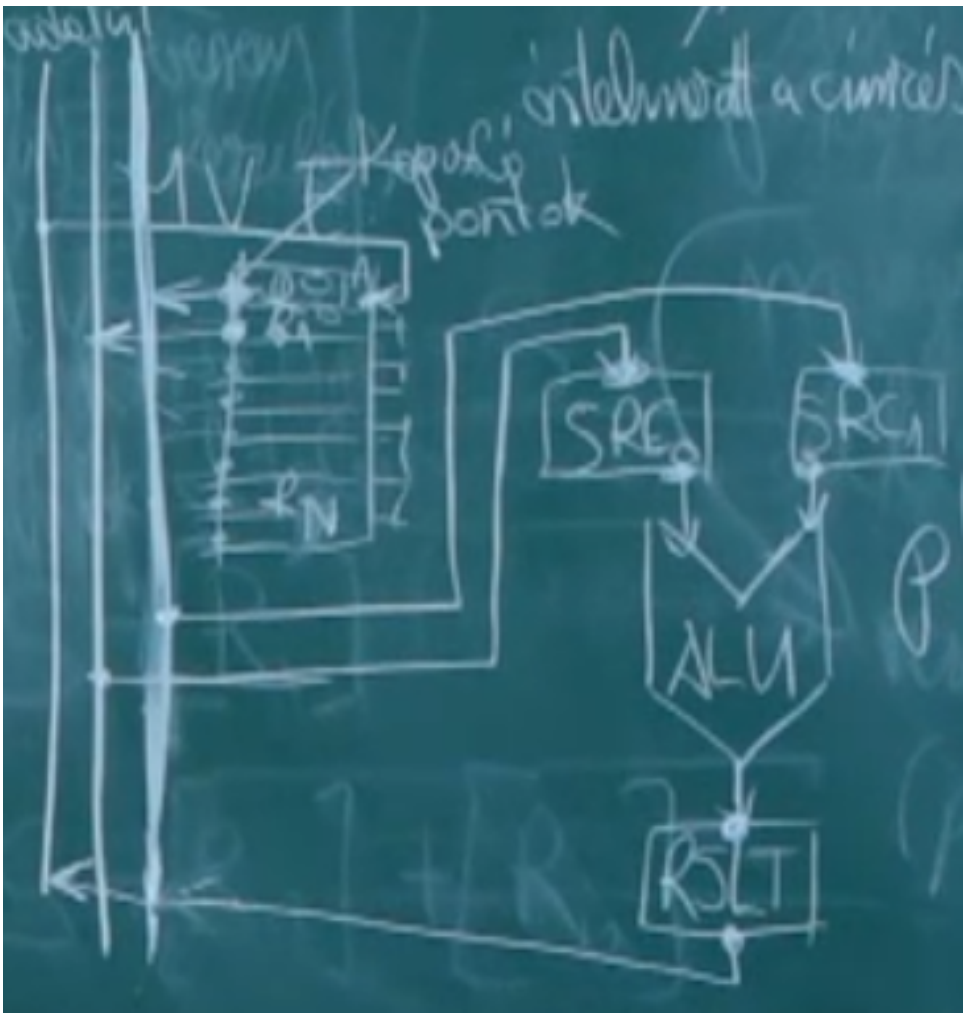
### Kétutas csatolás

A kétutas csatolás használatával egyszerre tölthető be a két operandus, így a betöltési idő a felére csökken.


4.2. ábra. Kétutas adatút

**Háromutas csatolás**

Itt a harmadik adatút a kimenetre van rákötve. Ezzel párhuzamos működés érhető el: a visszaírással együtt megtörténhet a következő utasítás forrás operandusainak betöltése.




4.3. ábra. Háromutas adatút

### 4.3.5. Műveletvégző (ALU)

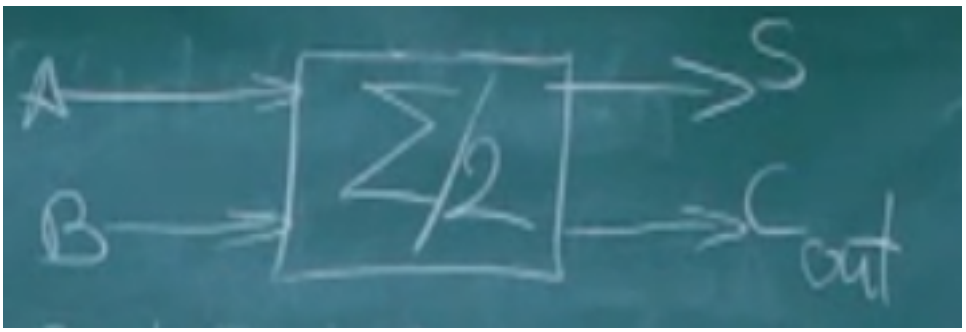
Az ALU (Aritmetikai Logikai Egység) végzi el a műveleteket. A leggyakoribb, legelemibb művelet az összeadás.

#### Az ALU által végzett műveletek

- FX: + - * /
- FP: + - * /
- BCD: + - * /
- Egyéb: eltolás, negálás, léptetés, logikai műveletek

#### Összeadó

Egy félösszeadónak két bemenete és két kimenete van, a második kimenet az átvitelre van (carry).

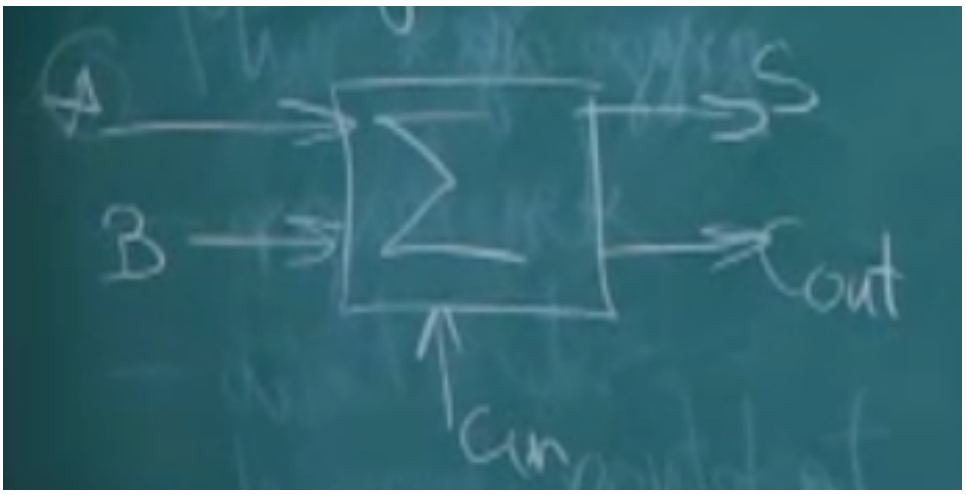



4.4. ábra. Félösszeadó

A carry bit meghatározásához egy AND kaput használ a félösszeadó (akkor van átvitel, ha mindkét bement 1), az ereményhez pedig egy XOR kaput.

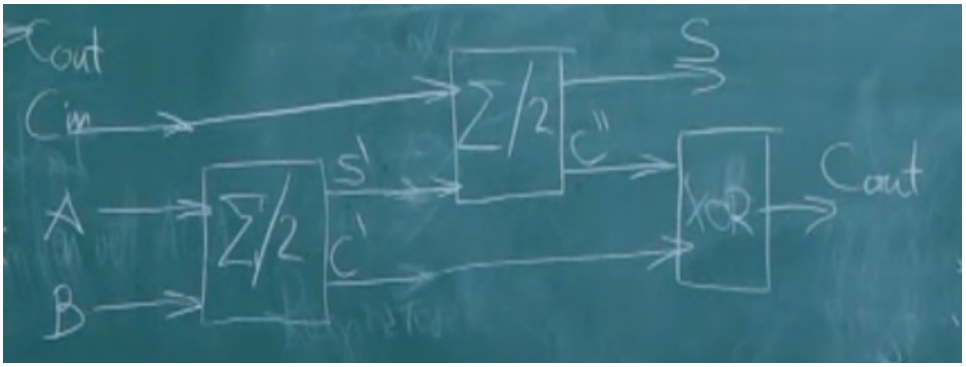

4.5. ábra. Félösszeadó kapujainak felépítése

Teljes összeadónál három bemenet van, hogy az előző összeadás carry kimenetét is figyelembe tudja venni.




4.6. ábra. Teljes összeadó

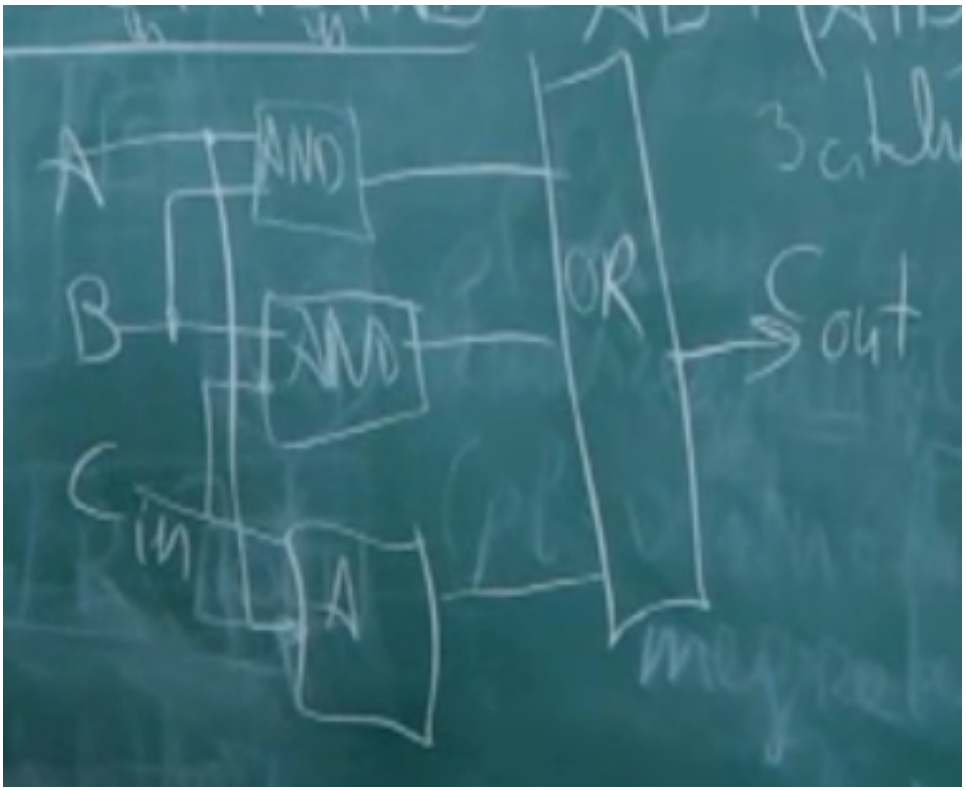
A teljes összeadó felépíthető félösszeadókból:




4.7. ábra. Teljes összeadó félösszeadókból

Ez 3 ciklusba kerül, a cél a folyamat gyorsítása. Az igazságtáblát felírva és a logikai függvényeket egyszerűsítve a következő kapukból építhető fel a teljes összeadó:


4.8. ábra. Teljes összeadó egyszerűsítése (eredmény kiszámítása)




4.9. ábra. Teljes összeadó egyszerűsítése (carry kiszámítása)

Ezzel a művelet már két óraciklus alatt is elvégezhető, azaz a teljesítmény 33%-kal növekszik.

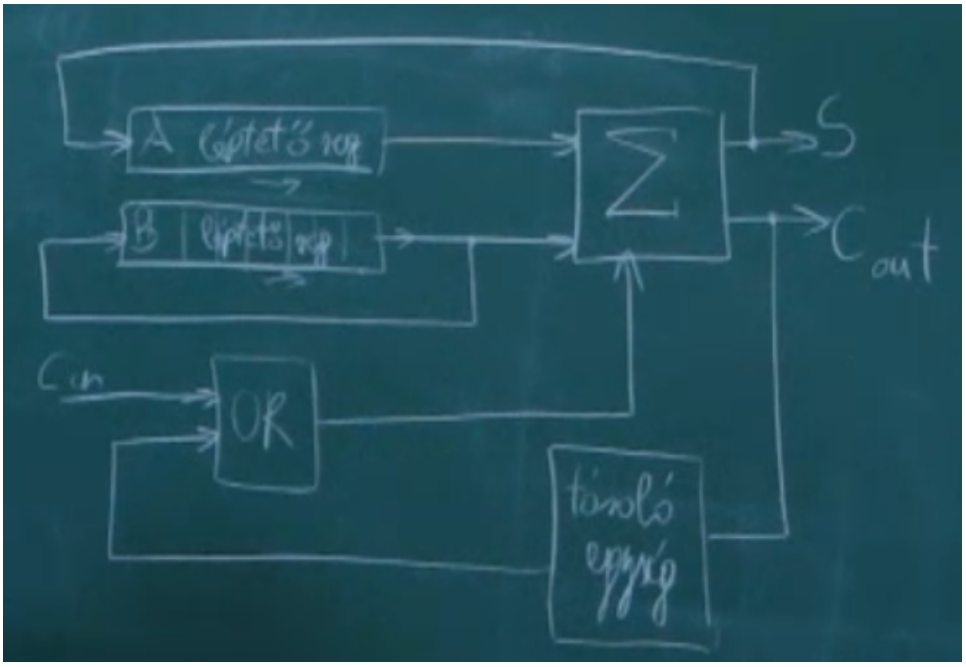
### n-bites összeadó

A gyakorlatban viszont általában nem biteket, hanem bitsorozatokat kell összeadni. Ezeknek a bitsorozatoknak a tárolása regiszterekben történik (byte, szó, duplaszó). A feladat tehát 2 db n-bites regiszter

összeadása.

Az első ilyen megoldás az n-bites soros összeadó volt. Ez a különböző helyiértékeken lévő biteket egymás után, külön-külön adja össze. A carry-t is figyelembe veszi, ezt a flip-flopok tárolják. Képes két különböző hosszú szám összeadására, ilyenkor a rövidebb számot automatikusan kiegészíti 0 helyiértékekkel. A megvalósításhoz egy darab teljes összeadót használ, valamint bevezették a léptető regisztert.

A B léptető regiszter kimenete rá van vezetve a bemenetére. A teljes összeadó bemenete a két léptető regiszter kimenetére csatlakozik. Így a léptető regiszter kimenete minden alkalommal bekerül a teljes összeadóba és a B léptető regiszter bemenetére is. Ekkor minden helyiérték egyel jobbra lép, a B regiszter tartalma változatlan marad, az A regiszterbe pedig bekerül az összeadás eredménye az összeadó kimenetéről.

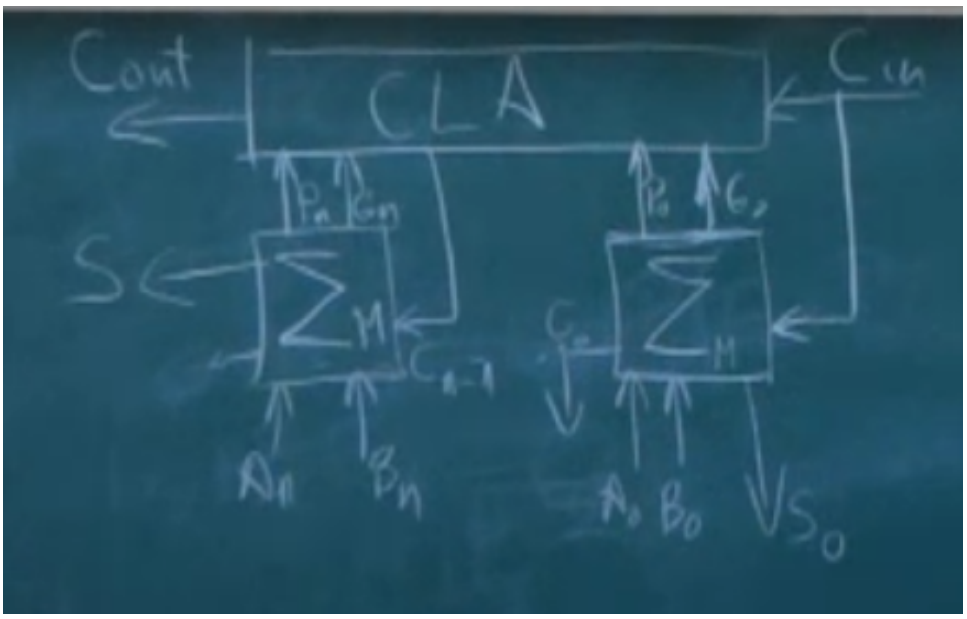



4.10. ábra. n-bites soros összeadó és a léptető regiszterek

A carry kezeléséhez a carry bementet először egy OR kapun kell átvezetni, így az előző összeadás carry-je és a bitenkénti összeadás carry-je is figyelembe van véve. A tároló egység, a flip-flop feladata, hogy a carry bemenetet csak az első összeadásnál vegye figyelembe.

Ezzel a módszerrel az összeadás megtörtént, de a soros végrehajtás miatt viszonylag lassú. Gyorsítási lehetőség az összeadás párhuzamosítása. Ehhez n darab 1 bites összeadóra van szükség, amik bemeneteire a két bemeneti regiszter megfelelő bitjei vannak rákötve. A teljes párhuzamos végrehajtást a carry bitek akadályozzák, mivel az n-edik bit előállításához szükség van az n-1-edik carry bit előállítására. A végrehajtási idő tehát nem sokat javult, viszont amennyiben nincs carry, úgy nagyon gyorsan előáll az eredmény. Tehát a végrehajtási idő hullámzó, a carry-től függ (ripple carry adder).

A további gyorsítás szimultán átvitelképzéssel lehetséges (carry lookahead - carry előrejelzés, más néven rekurzív módszer). A carry kimenetet adó logikai függvény vizsgálatával kiderül, hogy a végeredmény nem függ a bitenkénti carrytől. Az AB-t nevezzük generate-nek (G), az A+B-t propagate-nek (P). Ezt felhasználva 3 kapu segítségével előállítható az összes carry. Az ezt kiszámoló áramkör neve a CLA, azaz carry lookahead. A CLA bemenete az A és B és a bemeneti carry. Az összeadót úgy módosítjuk, hogy előállítsa P-t és G-t. A teljes összeadó carry kimenetét nem használjuk, hanem azt a CLA állítja elő.




4.11. ábra. A CLA-t használó összeadó felépítése

Ezzel az összeadás 3 ciklus segítségével megvalósítható. Korlátja, hogy egy OR kapu általában max 8 bemenetet kezel, tehát egy 32 bites összeadóhoz 4 db CLA-ra van szükség. Ezt segíthetjük egy plusz CLA-val, ami a CLA-k bemenő carry-jeit állítja elő.

Az összes mai összeadó ilyen rekurzív módszerek szerint működik.

### Szorzás megvalósítása

A szorzás is összeadások sorozatára vezethető vissza, így az előző módszerekkel megvalósítható. Sok szorzásnál ez lassú, így itt is gyorsításra, egyszerűsítésre van szükség.

Az egyszerűsítéshez felhasználható az összeadás, invertálás és léptetés. Régen ez gépi kódból történt, de manapság szakosodott műveletvégző egységek segítik ezeket a műveleteket.

Egyik egyszerűsítési lehetőség a gyűjtő regiszter és a léptetés használata. A gyűjtőt a művelet előtt nullázzuk.

A szorzás eredménye általában több bit helyet foglal, tehát nem valószínű, hogy belefér a forrásregiszterbe. Kettes számrendszerben ha A szám n bit hosszú, B pedig m bit hosszú, akkor elmondható, hogy $A * B <= m + n$. Két 8 bites szám esetén pl. az eredmény 16 bit hosszú lehet.

Gyorsítási lehetőségek:

- bitcsoporttal történő szorzás: pl. $7 * 9 = 63$, azaz $0111 * 1001$ szorzásánál először 10-al, majd 01-el szorzunk. Ha a bitcsoport 00, a szorzandót nem kell a gyűjtőhöz adni, csak 2-t léptetünk balra. Ha 01, a gyűjtőhöz a szorzandó egyszeresét adjuk hozzá és kettőt léptetünk balra. 10-nál a kétszeresét adjuk hozzá, azaz balra léptetünk egyet, hozzáadjuk, majd balra léptetünk kettőt (ha folytatódik a szorzás). 11-nél a 3-szorosát kell hozzáadni, majd kettőt léptetni balra (ehhez általában a 4-szeresét adjuk hozzá és kivonjuk az egyszeresét).
- Booth-féle algoritmus: probléma, hogy ha a szorzóban sok 1-es van, akkor lassú a szorzás. Ennek a gyorsításához a szorzóhoz közeli számmal szorzunk, majd a különbséget kivonjuk. Pl. 62-vel szorzásnál 64-el szorzunk (6 léptetés balra), majd kivonjuk a szorzandó kétszeresét.

### Osztás

Az osztás igényli a legtöbb időt, mivel bonyolult művelet. Különbség, hogy itt két kimenet van, az eredmény és a maradék, valamint kivételek is felléphetnek.

# 5. fejezet

# Lebegőpontos és BCD számok

## 5.1. Bevezetés

A lebegőpontos számok szükségesek, mivel:

- a fixpontos számok értelmezési tartománya viszonylag kicsi
- fixpontos számoknál a pontosság is viszonylag kicsi

A lebegőpontos számokat 1985-ben szabványosították (IEE754). Formájuk: $M * r^k$, ahol M a mantissza, r a radix és k a karakterisztika. Fontos, hogy r alapja megegyezzen az M számrendszerének alapjával.

## 5.2. Ábrázolás

A lebegőpontos számok ábrázolása normalizált formátumban történik, tehát a tizedes (kettes számrendszernél kettedes) pont után következik az első értékes számjegy. Példa: $0,1011 * 2^k$.

## 5.3. A mantissza lehetséges értékei

Ezzel a mantissza értéke fix intervallumba kerül, pl. 10-es számrendszernél $0,1 <= M < 1$. Ugyanez általánosan megfogalmazva: $1/r <= M < 1$.

## 5.4. Értelmezési tartomány

A lebegőpontos számok értelmezési tartománya függ:

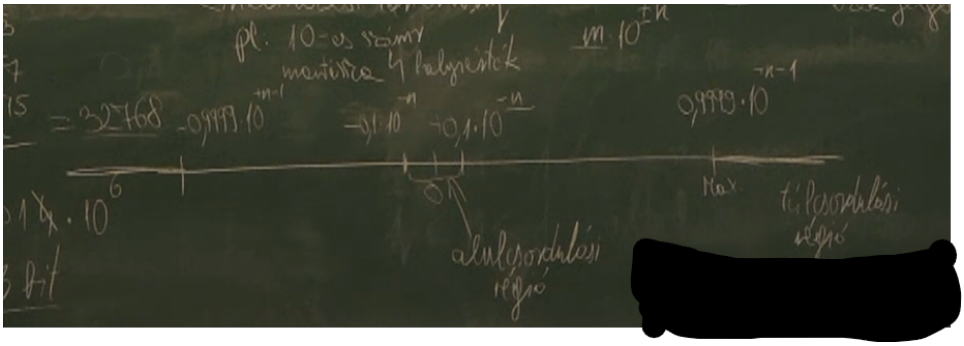
- a karakterisztika számára rendelkezésre álló bitek számától
- a radixtól

## 5.5. Pontosság

Példa: $0,3014 * 10^6$. Ekkor ha a mantisszára 3 bitet allokálunk, elveszítjük a 4-es számjegyet, csökken a pontosság. A pontosság tehát függ a mantissza hosszától (rendelkezésre álló bitek számától).

## 5.6. Alulcsordulás és túlcsordulás

Az értelmezési tartományon kívül eső értékeket nem tudjuk ábrázolni, így ott túlcsordulás keletkezik. Mivel a mantisszának 0,1-nél nagyobb vagy egyenlőnek kell lennie, ezért a nullát sem tudjuk ábrázolni, alulcsordulás keletkezik. Az alul- és túlcsordulási régiók nagysága elsősorban a karakterisztikára allokált bitektől függ.




5.1. ábra. Alul- és túlcsordulási régiók

Az architektúrának biztosítania kell a túlcsordulás felismerését, jelzését és kezelését.

Az architektúra lehetőségei túlcsordulás esetén:

- kijelzi a túlcsordulást és beállítja a legnagyobb megengedett értéket
- kijelzi a túlcsordulást és előjeles végtelent jelez ki

Alulcsordulás esetén:

- kijelzi az alulcsordulást és nullára konvertál
- kijelzi az alulcsordulást denormalizált számot jelzi ki

A kijezést flagek segítségével teszi.

A szabvány előírja, hogy ha a mantissza nulla, a karakterisztikának is nullának kell lennie.

## 5.7. Pontosság növelése

Probléma, hogy a tízes számrendszerben pontosan megadható számok egy része kettes számrendszerben végtelen tizedes törtként ábrázolható (pl. 0.3). A pontosság növelésére alkalmazott módszerek:

- rejtett bit használata: a mantissza tört utáni részénél az első számjegy (bit) mindig 1, így nincs információtartalma. Ezért azt nem tároljuk, így egy bittel pontosabb számot tudunk tárolni.

- őrző bitek: a szabvány elvárása, hogy a pontatlanság kisebb legyen, mint a normalizált eredmény legkisebb számjegye. Probléma, hogy normalizáláskor értékes biteket veszíthetünk el, ezért a CPU-n belül a regiszterek több biten tárolják a mantisszát. Általában +3-15 bit. Ezek az őrző bitek, amik a memóriában tárolás során nem jelennek meg. Az őrző bitek felhasználása:
  - a rejtett bit balra léptetésekor értékes bitet tudunk beléptetni
  - tárolási formátum kérésekor kerekített értéket tárolhatunk
  - normalizáláskor értékes biteket tudunk felhasználni

## 5.8. Lebegőpontos műveletvégzés jellemzői

- kódolás: mantissza kódolása 2-es komplemens, karakterisztika kódolása többletes kódolással. Ennek oka, hogy a többletes kód kialakítása gyorsabb, de elsősorban csak összeadás és kivonás elvégzésére alkalmas. Szorzást, osztást egyszerűbb kettes komplemenssel kódolt számokon könnyebb.

## 5.9. IEEE 754 további jellemzői

A szabvány fő célja a kompatibilitás megteremtése különböző CPU-k között. A hardvernek és a szoftvernek együtt kell biztosítania a szabványnak való megfelelést.

A szabvány legfontosabb fejezetei:

- adattípus
- formátumok
  - szabványos (háttértáron tárolás)
    - egyszeres pontosságú (32 bit)
    - kétszeres pontosságú (64 bit)
  - kiterjesztett (CPU-n belül)
    - egyszeres pontosságú (32 bit)
    - kétszeres pontosságú (64 bit)
- műveletek
- kerekítések
- kivételek kezelése

A szabvány által meghatározott műveletek:

- négy aritmetikai művelet
- maradékképzés
- gyökvonás
- bináris-decimális konverzió
- értelmezett a végtelennel való műveletvégzés
- kerekítések
  - legközelebbire kerekítés

- nullára kerekítés (őrző bitek levágása)
- kerekítés pozitív végtelen felé
- kerekítés negatív végtelen felé

A kivételek felbukkanása általában megszakítást eredményez. Kivételek lehetnek pl. overflow, underflow, nullával osztás, gyökvonás negatív számból.

Először a lebegőpontos egységek koprocesszorokban voltak, később az Intel 80486 CPU-nál már a processzorba volt integrálva.

## 5.10. Műveletek megvalósítása

- A+B: azonos hatványra hozás és összeadás
- A*B: A és B mantisszájának szorzása és a karakterisztikák összeadása

A szorzásnál fontos, hogy a két művelet párhuzamosan elvégezhető, így növelhető a sebesség. Ennek a megvalósítása dedikált FP műveletvégzővel történik: egy adatbuszon keresztül a mantissza a mantissza egységbe, a karakterisztika pedig a karakterisztika egységbe kerül, a végeredményt pedig a vezérlő rakja össze.

## 5.11. BCD számábrázolás

Megjelenésének fő oka a fixpontos és a lebegőpontos számok pontatlansága. Elsősorban adminisztratív és tudományos alkalmazásoknál használják (pl. számológépek). A lebegőpontos konvertálás lehet pontatlan, de a kódolás pontos megfeleltetés, nincs kerekítés.

BCD kódolásnál minden számjegyet 4 biten ábrázolunk. Mivel 4 bit 16 féle értéket vehet fel, keletkezik 6 darab érvénytelen tetrád, ami nincs használatban BCD kódolásnál.

### 5.11.1. Ábrázolási formátumok

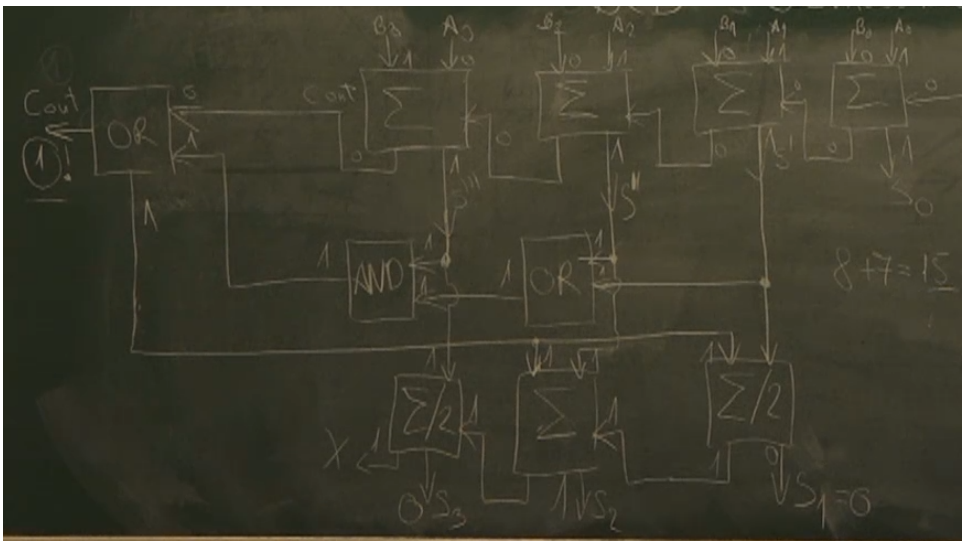
- zónázott: 1 byte = 1 számjegy, minden tetrádot megelőz 4 zóna bit
- pakolt: 1 byte = 2 számjegy (pl. Intel)

### 5.11.2. Műveletvégzés logikai szinten

Példa: 8+7=15. BCD kódolva ez 1000+0111=1111, ami egy érvénytelen tetrádot eredményez, így 10-et ki kell vonni belőle (azaz hozzáadunk -10-et). Kettes komplemens esetben 1010 invertálva 0101, majd hozzáadva egyet 0110. Ezzel előáll az 5 mint eredmény.

### 5.11.3. Műveletvégzés áramköri szinten

A megvalósításhoz szükség van 4 db teljes összeadóra. Az áramkörnek el kell végeznie az érvénytelen tetrádok kezelését is. Az előző példánál az áramkör működése:




5.2. ábra. BCD összeadó áramkör

### 5.11.4. Összegzés

A BCD kódolás előnye tehát, hogy teljesen pontos, viszont jelentősen több helyet foglal, mint a lebegőpontos számok, valamint lassabb a végrehajtás is sok esetben. A BASIC programnyelv ezt használja.

## 5.12. Az ALU egyéb műveletei

Az aritmetikai műveleteken kívül egyéb műveleteket is ellát az ALU, ezek általában egyszerűbb áramköröl segítségével megoldhatók.

- Boole műveletek (mind a 16 fajta) pl.:
  - AND
  - NOR
  - XOR
- léptetés
- invertálás
- komparálás (feltételes ugrás)
- LOAD/STORE címszámítás - valós időben címet generál a MAR számára
- karakteres műveletek

# 6. fejezet

# A vezérlőegység

A vezérlőegység az egyik legbonyolultabb áramkör a processzoron belül, a felépítése általában titkos. Célja a vezérlőjelek előállítása.

## 6.1. Fejlődése

A vezérlőegységeknek két típusa létezik: szekvenciális (centralizált) és párhuzamos (decentralizált). A párhuzamos vezérlés komplexebb, mint a szekvenciális. Az első számítógépeknél huzalozott, szekvenciális vezérlést alkalmaztak.

1957-ben változtattak rajta először a mikroprogramozott vezérlés megjelenésével. 1964-ben jelent meg a szuperskalár elv, majd 1967-ben a futószalag. Ezek a vezérlési módok már komplexitás szerint párhuzamosak voltak, tartalmaztak huzalozott és mikroprogramozott részeket is.

## 6.2. Ütemező

A vezérlő lelke az ütemező, a szekvenciális vezérlők a vezérlő jelek fix szekvenciáját állítja elő (huzalozott vezérlés). A párhuzamos elvet követő ütemezők mikroprogramok segítségével állítják elő a vezérlő jeleket.

## 6.3. Huzalozott vezérlő

### 6.3.1. A tevezés lépései

1. igazságtábla felírása
2. logikai függvények felírása
3. azonos átalakítások (egyszerűsítés)
4. áramköri elemek számának minimalizálása
5. végrehajtási idő minimalizálása
6. megvalósítás

### 6.3.2. Előnye

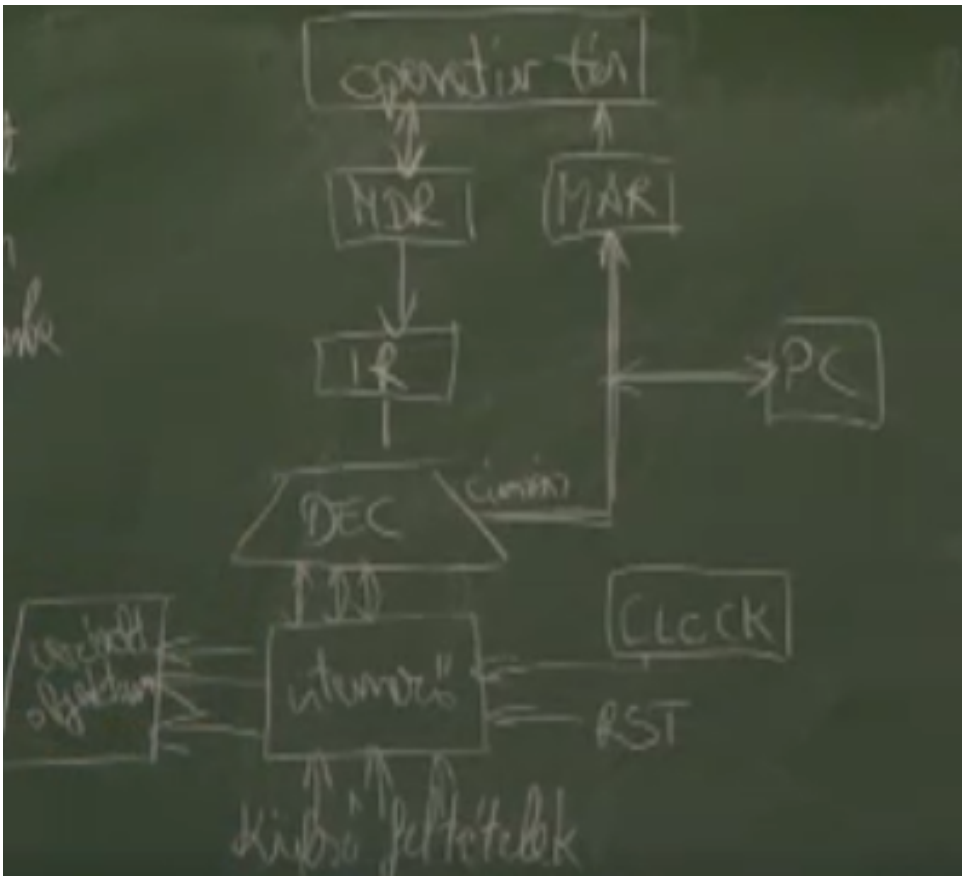
Mivel előre elkészített áramkörökkel működik programok helyett, nagyon gyors.

### 6.3.3. Hátránya

A bonyolult áramkör nehezen átlátható, módosítása pedig nehézkes.

### 6.3.4. Működése

A vezérlés alapvető elve, hogy egy (forrás) regiszter tartalmát egy módosító áramkörön keresztül egy másik (cél) regiszterbe vezetjük. Működési sémája:




6.1. ábra. A huzalozott vezérlő működési elve

Az ütemező feladata az összes többi objektum vezérlése. Ilyen objektumok pl.:

- buszok
- ALU
- PC
- IR
- IO

- adatutak

A módosító áramkörök léptethetik, invertálhatják, komparálhatják az adatokat. A vezérlő feladata még a kapuáramkörök megnyitása is.

## 6.4. Mikroprogramozott vezérlő

A huzalozotthoz képest átláthatóbb, könnyen módosítható, viszont lassú. A vezérlőjeleket egy mikroprogram állítja elő. Minden CPU utasítás egy vagy több mikroutasítással valósul meg. A mikroművelet aktivál egy specifikus vezérlő vonalkészletet. A mikroprogram kimenete a vezérlő jel. A mikroprogram fejlesztése nagy hardverismeretet igényel, ezért a gyártók írják a saját processzoraikhoz.

A mai processzorok kombinálják a huzalozott és a mikroprogramozott vezérlést.

# 7. fejezet

# Buszrendszer

A buszrendszer egységek közötti kommunikációra szolgál (pl. CPU-MEM-perifériák), a kommunikáció infrastrukturális része. Egy történeti fejlődés eredménye, az adatkommunikációs megoldások közül ez bizonyult a legjobbnak. Az első szabványos buszrendszert az IBM hozta létre 1981-ben. Fontos, hogy ez egy nyílt szabvány volt, minden mai buszrendszernek ez az alapja, tehát minden buszrendszer lényege, hogy szabadon használható, nyílt architektúrára épül. A szabványok definiálják a vezetékköteget és a csatlakozók kiosztását, a szabványok függetlenek a processzor belső architektúrájától, tehát platformfüggetlen megoldást jelentenek.

## 7.1. Jellemzői

- az eszközök kizárólag ezen keresztül kommunikálnak
- egy buszrendszerre általában egyszerre több egység kapcsolódik →meg kell oldani az adatátvitelben részt vevő eszközök kijelölését, meg kell határozni az átvitel irányát és meg kell oldani az összehangolást és szinkronizálást.
- a felsorolt problémák megoldására a legfontosabb a szabványosítás - szabványos jelhasználat, vezetékkiosztás
- fontos tulajdonsága a sebesség, hogy az adatkommunikáció ne jelentsen szűk keresztmetszetet
- a buszrendszer transzparens - a rendszer kezeli, a felhasználó számára nem látható

## 7.2. Csoportosítása

- átvitel iránya szerint
- átvitt tartalom szerint
- átvitel jellege szerint
- összekapcsolt területek alapján
- átvitel módja szerint

### 7.2.1. Átvitel módja szerint

- soros
- párhuzamos

### 7.2.2. Átvitel iránya szerint

- szimplex (csak egy irányba mehet az adat, pl. címbusz)
- félduplex (egyszerre csak egy irányban közlekedik az adat)
- duplex (egyszerre két irányba mehet az adat, pl. adatbuszok)

### 7.2.3. Átvitel jellege szerint

- dedikált - minden mindennel össze van kötve pl. Intel QuickPath Interconnect
- shared (osztott)

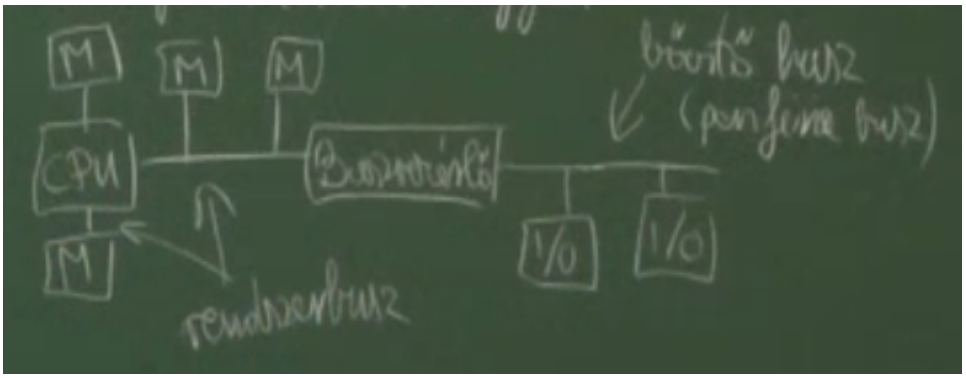
A megosztott busz egyszerűbb felépítésű, olcsóbb és skálázhatóbb (nem kell mindent mindennel összekötni), de buszvezérlő utasításokra van szükség az ütközések elkerülésére. Ehhez buszvezérlő vonalakra van szükség. Hátránya az osztott buszoknak, hogy viszonylag lassúak, bonyolult a vezérlésük és meghibásodás esetén több eszköz kieshet.

### 7.2.4. Átvitt tartalom szerint

- adatbusz
- címbusz
- vezérlőbusz

### 7.2.5. Összakapcsolt területek szerint

Összekapcsolt területek szerint megkülönböztetünk rendszerbuszt és bővítő (periféria) buszt.




7.1. ábra. Rendszerbuszok és bővítő buszok

Bővítő busz pl. a SATA és az USB.

## 7.3. Címbusz

Ezen a vezetékkötegen áramlik az eszközök címzésére szolgáló adat.

## 7.4. Adatbusz

A fejlődése során a sávszélességet folyamatosan növelték: 8,16,32,64 bit. Fokozatos bővítéssel fejlesztettek.

### 7.4.1. Buszvezérlő

A mai rendszerekben az adatbusz vezetéke közös lehet a címbusszal, időbeli multiplexeléssel. Ehhez szükség van egy buszvezérlőre és egy vezérlővonalra. A vezérlőjelek mutatják meg, hogy éppen adat, vagy cím közlekedik a buszon. A vezérlőjeleket csoportosíthatjuk:

- adatátvitelt vezérlő jelek
- megszakítást vezérlő jelek
- buszvezérlő jelek
- egyéb vezérlő jelek

#### Adatátvitelt vezérlő jelek

- M/IO jelek: megmutatják, hogy memória, vagy IO cím található a buszon
- R/W (read/write) jelek: megadja az adatáramlás irányát a CPU felől nézve
- WD/B (word/byte): megadja az adat méretét
- D/S (data strobe) jel: az adat felhelyezését jelzi a memória számára
- A/S (address strobe) jel: a cím buszra helyezését jelzi az eszköz számára
- ready jel: az átvitel befejezését vagy a busz rendelkezésre állását jelzi

#### Megszakítást vezérlő jelek

- megszakítást kérő jelek
- megszakítást visszaigazoló jelek

#### Buszvezérlő jelek

- busz foglalással kapcsolatos jelek
- visszaigazolás

#### Egyéb vezérlő jelek

- reset
- CLK

## 7.5. Soros és párhuzamos buszok

Kezdetben soros buszokat alkalmaztak, később párhuzamosak terjedtek el. A soros busz előnye, hogy kevesebb vezeték szükséges, a párhuzamosoknál viszont hatékonyabb az adatátvitel (egyszerre általában nem 1 bitet kell átvinni, párhuzamos busszal lehetséges több bit átvitele egyidőben).

## 7.6. A PCI szabvány

A PCI ma a legfejlettebb párhuzamos busz. Legnagyobb előnye, hogy a PCI buszra csatlakoztatott perifériákat a CPU úgy látja, mintha a saját címterében lenne, azaz mintha azok közvetlenül a rendszerbuszra csatlakoznának. Következmény, hogy a CPU az általa látott címtérből látja el memóriacímekkel a perifériákat.

A PCI specifikáció magában foglalja a fizikai méreteket, csatlakozókiosztást, adatátviteli időzítést, protokollokat.

Főbb jellemzői:

- megosztott párhuzamos architektúrát használ
- minden eszköz közös cím, adat és vezérlő buszt használ
- több master esetén arbitrálás szükséges - buszfoglalás
- egy időben egy master működhet
- egyetlen közös, nagy teljesítményű busz

Az eredeti PCI szabvány nem tudta kiszolgálni a multimédiás alkalmazásokat, ezért megjelent az AGP (Advanced Graphics Port) nevű kiterjesztése, ami egy viszonylag olcsó és széleskörűen alkalmazott technológia volt. Ma már a legtöbb helyen áttértek a PCIe buszokra.

## 7.7. Az USB szabvány

Ez is perifériák csatlakoztatására használják, fejlesztését 1994-ben kezdték. Az első működőképes verziója az USB 1.1-es szabvány volt, 1998-ban jelent meg. Az USB 3.0 tíz évvel később, 2008-ban jelent meg, majd 2013-ban a 3.1.

Csatlakozók:

- hagyományos (2 érpár, max. 10 Gbit/sec)
- USB-C (4 érpár, 24 csatlakozó, szimmetrikus, akár 20 Gbit/sec)

Az USB-C egyik célja minél több kábel kiváltása, ezért számos protokollt támogatnak, akár notebook töltésére is használható.

## 7.8. Thunderbolt

Az Intel által kifejlesztett buszrendszer, PCIe alapokra fejlesztették, nemrég tették nyitottá a szabványt. A Thunderbolt támogatja az USB-C eszközöket is (visszafelé nem igaz).

## 7.9. PCI express

2004-ben jelent meg, a mai napig használják, gyakorlatilag egyeduralkodó a belső buszok között. A PCIe egy nagy sebességű, platformfüggetlen soros busz, ezért kevesebb érintkezővel rendelkezik. Hotplug funkciója segítségével menet közben cserélhetők és lekapcsolhatók az eszközök.

Topológiája pont-pont, az eszközöket különálló vezetékek kapcsolják a buszvezérlőhöz. Full duplex átvitellel rendelkezik bármely két végpont között, több végpont tud párhuzamosan kommunikálni. Többféle szélességű aljzattal rendelkezik (1x, 4x, 8x, 16x, 32x), tehát a szabvány rugalmas.

A busz protokoll csomagokba ágyazza be az adatokat, hasonlóan az Ethernet működéséhez. A szélességek azt jelentik, hogy hány darab soros érpár van a buszon. Ezzel eléggé hasonlít a párhuzamos buszokra, viszont a csomagokba ágyazása miatt nem kell az érpárokat szinkronizálni. Következmény, hogy jelentősen nőtt a teljesítmény.

### 7.9.1. A párhuzamos adatátvitel hátránya

A párhuzamos buszok nagy csatlakozási felülettel rendelkeznek, valamint a nagy frekvenciás átvitel nem megoldható, problémákba ütközik:

- delay skew: nagy frekvenciás jel esetén már viszonylag kis vezetékhossz esetén is elcsúszás lehet a jelekben, tehát a párhuzamos jelek nem egy időpillanatban érkeznek meg. Egy bizonyos frekvencia fölöött nem biztosítható a szinkronizáció.
- elektromágneses interferencia: az EM interferencia zajt jelent és zavarja a jelet
- vezetékek közötti áthallás: nagy frekvencián minden vezeték tekercsként működik, hosszabb vezetékeknél egyre súlyosabb az áthallás

A párhuzamos adatátvitel korlátai nagyobb távolságokon jobban jelentkeznek, ezért is teszik közel a CPU-t és a memóriát az alaplapokon.

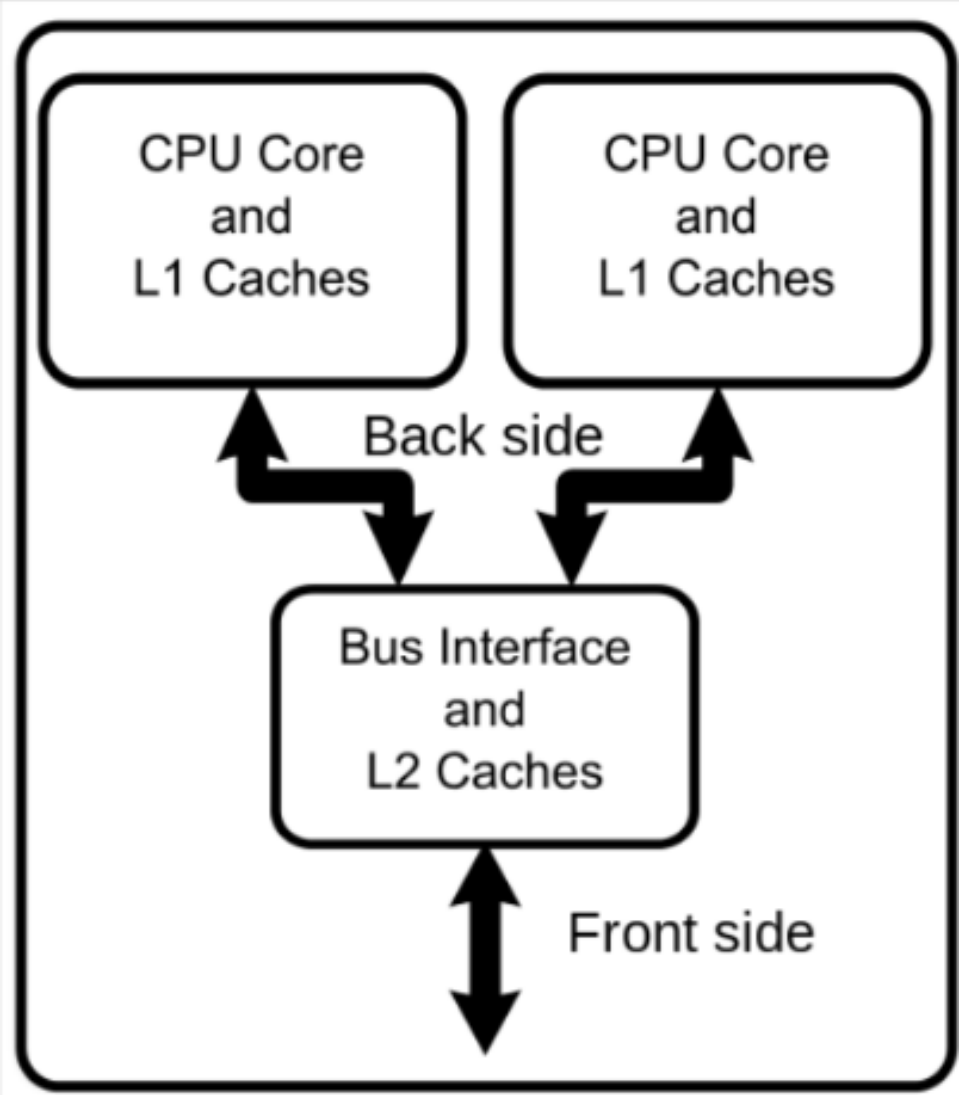
### 7.9.2. A soros adatátvitel hátránya

Több bit átvitelénél sorba kell állítani a biteket, egyesével kell küldeni a biteket. Ehhez kódolást használnak, amihez plusz hardverre van szükség.

## 7.10. Buszrendszerek gyakorlati megvalósítása

Az alaplapi chipkészletek először az IBM 286-os processzorával jelentek meg. Célja a korábbi években használt egyetlen buszrendszer kiváltása a teljesítmény növelése érdekében. Az Intel Pentium esetében két ilyen chip volt, az északi és a déli híd.

A buszokat feloszthatjuk FSB (Front Side Bus) és Back Side Buszokra. Az FSB a perifériákkal kommunikált, a Back Side Bus pedig a processzoron belüli kommunikációért felelt.




7.2. ábra. Front side és back side buszok

### 7.10.1. FSB

Az FSB két részre oszlik: északi és déli híd. A north bridge gyorsabb, a memóriákkal és a grafikus interfésszel való kapcsolatot biztosítja (PCIe, AGP). A south bridge (IO controller) a perifériákkal tartja a kapcsolatot (pl. PCI). A vezérlők frekvenciája jelentősen kisebb a processzoránál, ezt a nagyobb sávszélességgel kompenzálja.

Később az északi hidat a processzorba integrálták.

## 7.11. Fejlődés okai és irányai

A buszrendszerek fejlődését a processzorok sebességének nagy léptékű fejlődése tette szükségessé. Ezért a legfőbb cél az IO szűk keresztmetszet mérséklése. Ennek érdekében bevezették a gyorsítótárakat és a blokkos adatátvitelt.

A processzorok fejlődésével nem tudtak lépést tartani a hagyományos párhuzamos buszok. A probléma

a perifériák csatlakoztatásánál is felmerült (pl. USB), sokszor az eszköz gyorsabb volt, mint amit a busz ki tudott szolgálni.

## 7.12. IO eszközök és memória kapcsolata

- megosztott busz koncepció (pl. PCI)
- pont-pont kapcsolat, dedikált buszok (PCIe, HyperTransport, QPI)

## 7.13. HyperTransport

Az AMD fejlesztette ki ezt a buszrendszert 2001-ben. Jellemzői:

- kétirányú soros/párhuzamos kapcsolat
- alacsony késleltetés
- szélessávú kapcsolat
- rugalmas
- csomagokba ágyazza az adatokat
- 4, 8, 32 db soros vonalat tud kezelni
- általában CPU lapkára integrált, de nagy sávszélességű IO busznak is használják
- skálázható
- több csatorna alakítható ki
- erre csatlakozik az Ethernet, SATA, PCI, PCIe vezérlő is
- nem rendelkezik dedikált címtérrel, hanem a memóriában leképzett IO-val rendelkezik

Kétféle egységet tartalmaz:

- tunnel
- cave - ez az utolsó egység, ez zárja le a HT láncot, amire a tunneleket fel lehet fűzni (max 4)

## 7.14. QuickPath Interconnect

Az Intel fejlesztése, feladata részben az FSB kiváltása volt. 2008-tól kezdték alkalmazni, először Xeon, később a Core i processzorokban is. Lapkára integrált memóriavezérlővel és NUMA memóriahozzáféréssel rendelkezik. Osztott memóriát használ, tehát a processzor 2 memóriavezérlővel rendelkezik és mindkettőhöz hozzá van csatlakoztatva 2-2 memória slot. A processzor tehát nem mindegy, hogy melyik oldalról tölti be a memóriát.

A QPI is csomagkapcsolt rendszer, 5 rétegű architektúrát definiál (fizikai, kapcsolati, útválasztási, szállítási, protokoll). 2x20 vezetéket tartalmaz (adó és vevő).

Több processzoros rendszer esetén a QPI minden processzort minden processzorral összeköt.

# 8. fejezet

# Memóriák

A memóriák nagyságrendekkel gyorsabbak a merevlemezeknél, de jelentősen lassabbak a gyorsítótáraknál.

Csoportosításuk:

- írható-olvasható (RAM)
  - DRAM (operatív tár)
  - SRAM (gyorsítótárak)
- főképp csak olvasható (CMOS)
- csak olvasható (ROM)

## 8.1. ROM

Ma általában flash tárolóval oldják meg, tartalmuk maradandó.

## 8.2. RAM

Random Access Memory, tartalma nem maradandó.

### 8.2.1. SRAM

Statikus memória, a tápfeszültség megszűnéséig tárolja az adatot. Félvezető, flip-flop memóriában tárolja az adatot, általában regisztereknek és cache-nek használják. Jellemzői:

- energiatakarékos
- gyors
- drága
- az adatokat nem kell frissíteni

A dinamikus RAM-nál nagyságrendekkel gyorsabb.

### 8.2.2. DRAM

Néhány pikofarados kondenzátorokból és 1 tranzisztorból épül fel egy cella. Ez egy idő után kisül, így az adatot frissíteni kell. Jellemzői:

- olcsó
- alacsony fogyasztás
- kis helyigény

#### Típusai

- klasszikus, aszinkron DRAM
- szinkron DRAM
  - SDR - Single Data Rate: az órajelnek csak a felmenő élén történik átvitel
  - DDR - Double Data Rate: felmenő és lemenő élre is történik adatátvitel

#### Felépítése

A memória több logikai részre - bankokra - van felépítve, hogy egyszerre több memória műveletet is végre tudjon hajtani. Olvasása futószalag elven történik, tehát a memória művelet parancs kiadása után az eredmény csak több óraciklus után jelenik meg. Ebből fakad a késleltetés, azaz a latency, ami a memória egy fontos teljesítmény paramétere.

### 8.2.3. Prefetch eljárás

A prefetch eljárás fajtája megmutatja, hogy adott óraciklus alatt hány bit tölthető az IO pufferbe. 2n eljárásnál pl. ez a szám 2.

#### Fejlődése

- DDR-400: 2n prefetch
- DDR2: alacsonyabb órafrekvencia, alacsonyabb fogyasztás, 4n prefetch
- DDR3: 8n prefetch
- DDR4: ugyanolyan órafrekvencia mellett nagyobb a késleltetés, mint DDR3-nál. Ugyanúgy 8n prefetch-t használ, viszont létrehoztak bankcsoportokat. Megjelenik a CRC és a chipenkénti extra paritás, így növekedett a stabilitás.

#### Az olvasás lépései

- bank megnyitása
- oszlopblokkok olvasása a megnyitott sorból
- bank zárása
- legalább tRP várakozás, mielőtt a bankot újra megnyithatnánk

Tehát a legjobb az, ha a megnyitott bankok soraiból olvasunk.

## 8.3. Fejlődési trendek

- nő az órafrekvencia és a sávszélesség
- nő a prefetch
- csökken a lapkaméret és a tápfeszültség

## 8.4. DIMM-ek

- 64 bites szervezés: 168-284 pin
- registered DIMM: a DRAM chipek és a memóriasín közé helyeznek egy regisztert (ami minden műveletet pufferel), így kisebb elektromos terhelést jelent a vezérlő számára és több modul esetén nagyobb a stabilitás, ezért elsősorban szerverekhez használják.
- ECC chip: Error Control Coding, míg a paritás bit segítségével a memória felfedezheti az egyszeres hibát, viszont hibajavításra nem képes, az ECC chip segítségével javítani is tudja az adatot, valamint két bit hibát is képes felismerni.
- PLL chip: fázis zárt hurok, az órajel elcsúszás mentesítésre szolgál, ezt is elsősorban szerverekben használják

# 9. fejezet

# I/O rendszer

A CPU-t és a perifériákat összekötő egységet hívjuk I/O egységnek. Például egy billentyűzet csatlakozásánál az alaplapi USB vezérlő számít I/O egységnek.

## 9.1. Fejlődésük

1. A CPU közvetlenül vezérli a perifériát (megszakítás nélküli programozott IO - wait for flag)
2. Egy I/O modul kerül kialakításra
   - megszakítás nélkül, wait for flag
   - megszakításos vezérlés
3. DMA segítségével közvetlen hozzáférés a perifériához
4. IO csatorna - lassabb perifériák számára. IO-ra specializált utasításkészlettel rendelkezik és a központi operatív tárat használja
5. IO processzor - az IO csatorna továbbfejlesztése, saját működésre képes egység, saját memóriával

## 9.2. Programozott I/O

Működés szerint megkülönböztetünk lekérdezéses és megszakításos I/O-t, címzés szerint pedig különálló címterű és memóriában leképzett címterű I/O-t.

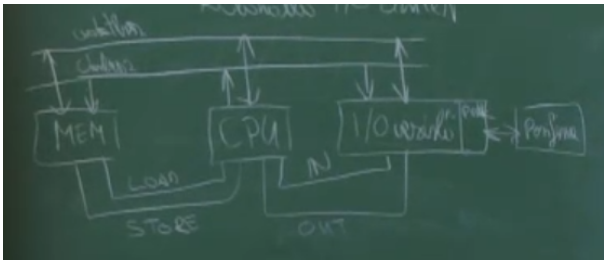
### 9.2.1. Lekérdezéses és megszakításos vezérlések

A megszakítás nélküli vezérlés nagy hátránya, hogy az utasítás kiküldése után a CPU egy ciklusban várakozik a periféria válaszára, és amíg ez meg nem érkezett, blokkolta a további műveleteket. Ezért jelentek meg a megszakításrendszerek, a processzor az utasítás kiadása után más feladatokkal foglalkozott, majd az IO egység megszakítással jelzi a művelet befejezését. Következmény, hogy nő a teljesítmény.

### 9.2.2. Különálló címterű IO

A különálló I/O címtérnél a CPU két címteret lát: az I/O és az operatív tár. A két címzés közös buszt használ, ezért külön utasítások tartoznak az IO műveletekhez. Következmény, hogy lehet azonos egy

memória és egy IO cím. Ehhez kellenek busz vezérlő jelek (M/IO jel) és speciális utasítások (pl. IN X, OUT X).



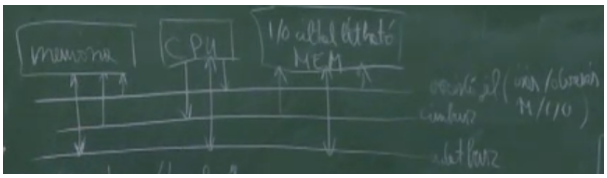

9.1. ábra. A memória és IO címzés

**Értékelés**

A különálló címterű I/O előnye az egyszerű megvalósítás, viszont terheli a CPU-t és plusz utasításokra van szükség.

### 9.2.3. Memóriában leképzett IO

Ekkor a memória egy része az IO egységeknek fenntartott, az adatok olvasása és írása LOAD/STORE utasításokkal történik. Az IO vezérlő ilyenkor hozzáfér a rendszerbuszhoz, gyorsabb az átvitel.




9.2. ábra. Memóriában leképzett IO címzése

Példa: a grafikus vezérlő megengedi a processzor számára, hogy közvetlenül címezze a framebuffert, ami a képernyőn adott pillanatban megjelenő kép adatait tartalmazza.

### 9.2.4. Az I/O vezérlő

Az IO vezérlő egy kártyán helyezkedik el, ezen a kártyán vannak az IO portok (a rendszerbusz és a periféria közötti csatlakozási pont). Az IO port egy egyedi azonosítóval ellátott eszköz, saját címe van és legalább egy adatregisztert tartalmaz. A vezérlő tartalmaz:

- parancsregisztert, ide írja a CPU a kívánságait
- adatregisztereket (data input és output)
- állapotregisztert - állapotinformációkat közöl a perifériáról, minden bitje mást jelent. Ilyen állapotok pl.:
  - megszakítás kérés
  - ready
  - foglalt (busy)

- jelenlét ellenőrző regiszter - megmondja, hogy van-e eszköz kapcsolva az I/O portra
- eszköz tulajdonságait tartalmazó regiszter (plug and play)

Bonyolultabb perifériák esetén egyes funkciókhoz akár több regiszter is tartozhat.

### 9.2.5. Működése

Az IO működését átviteli módszerek alapján két csoportra osztjuk:

- feltétel nélküli adatátvitel (direkt programozott)
- feltételes adatátvitel

Feltétel nélküli adatátvitelnél a perifériának mindig adatátvitelre alkalmas állapotban kell lennie. Ellenőrzésre sem az átvitel előtt, sem után nincs szükség és szinkronizálás sincs a CPU és a periféria között. Ilyen működésű pl. a kijelző, a processzor nem foglalkozik azzal, ha a monitor pixelhibás. Hátránya, hogy nincs visszacsatolás.

Feltételes adatátvitelnél valamilyen feltétel teljesülése esetén valósul meg az adatátvitel. Ilyen pl. a nyomtatók. Ennek a két csoportja a lekérdezéses és a megszakításos adatátvitel. Lekérdezésesnél a processzor folyamatosan figyeli az állapotregiszter tartalmát. Fejlettebb változata a megszakításos működés, ennél csak az IO regiszter által küldött megszakításnál kérdezi le az állapotregisztert és a megszakítás kiszolgálása eredményezi az adatátvitelt. Ez a módszer viszont még mindig lassú és sok megszakítással jár, mivel mindent a CPU vezérel. Ennek a kiküszöbölésére találták ki a Direct Memory Access-t (DMA).

## 9.3. DMA

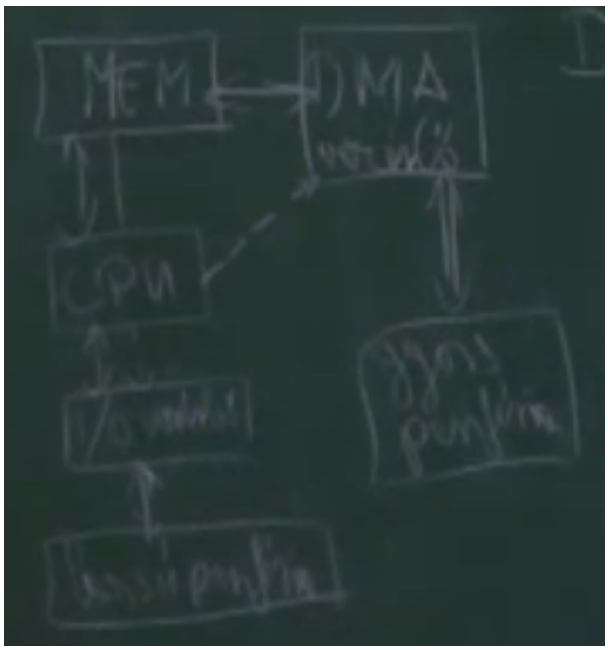
Adatátvitel szerint csoportosítjuk:

- blokkos
- ciklusloPásos

A DMA-t alkalmazzák:

- nagy mennyiségű adat átvitelénél (általában blokkos átvitel)
- gyors perifériáknál

Az adatátvitel a CPU közreműködése nélkül történik. A DMA mellett megmarad az IO vezérlő is, ami a lassú perifériák számára biztosítja az adatátvitelt.




9.3. ábra. A DMA felépítése

A DMA lényege, hogy viszonylag kis mértékű komplexitás növekedés segítségével tehermentesíthető a CPU és megvalósítható a közvetlen adatátvitel. A DMA a processzor helyett végzi a perifériákról a memóriába beolvasást. A DMA vezérlő tulajdonságai:

- címet kell tudnia generálni (a memória rekeszek címzéséhez)
- buszvezérlési funkciókat kell megvalósítania

Ezt a módszert nevezzük közvetlen tárhozzáférésnek. Előnye, hogy jóval kevesebb megszakítás szükséges.

### 9.3.1. A DMA vezérlő

A DMA saját regiszterekkel rendelkezik, pl. IO cím, IO data, DC, saját belső vezérlővel és rejtett regiszterekel is felszerelt. Működési lépései:

1. felparaméterezés: a processzor betölti a szükséges adatokat a DMA-ba (ez programozott IO-val történik). Az átadott paraméterek: írási/olvasási műveletet kell végrehajtani (irány), az IO egység címe, a memóriacím kezdőértéke (IO address regiszter), átvivendő adat jellege (pl. byte, félszó, szó), olvasandó/írandó egységek száma (DC-be), átvitel módja (blokkos/ciklusloposos), DMA csatornáihoz rendelt prioritási érték, résztvevő egységek (pl. mem-mem, IO-mem, IO-IO)
2. második lépésben eltér a blokkos adatátvitel és a ciklusloposos:
   - blokkos átvitel esetén az átvitel idejére a DMA lefoglalja a buszt, a paraméterek bekerülnek a DMA regisztereibe
   - ciklusloposos átvitelnél nem a teljes átvitelre foglalja le a buszt, csak bizonyos időtartamokra (olyan ciklusoknál, amikor a CPU nem használja a rendszerbuszt, azaz decode és execute fázisokban - ezek a DMA töréspontok). Ezt kevesebb adat esetén használják.
3. a DMA megszakítás kérést küld a processzornak (DMA request)

4. a CPU egy DMA acknowledge jelzést küld a DMA vezérlőnek, amiben engedélyezi a buszhasználatot
5. az IO címregiszterbe a DMA betölti az első címet, majd kiteszi a címbuszra
6. beolvassa a perifériából az információt az IO data regiszterbe
7. az IO data regiszterből betölti az operatív tárba az adatot
8. dekrementálja a Data Counter regiszter tartalmát és megnézi, hogy nulla-e
9. ha nulla, küld egy megszakítást a processzornak, amiben jelzi, hogy befejeződött az átvitel. Ha nem nulla, inkrementálja a címregisztert és újrakezdi a beolvasást.
10. a processzor ellenőrzi az adatátvitelt és visszaveszi a vezérlést, megszünteti a buszhasználat engedélyezését

A ciklusloPásos végrehajtást ma már nem nagyon használják, mivel a futószalag és a párhuzamos végrehajtás miatt a buszrendszer gyakorlatilag folyamatosan használatban van.

## 9.4. I/O csatorna

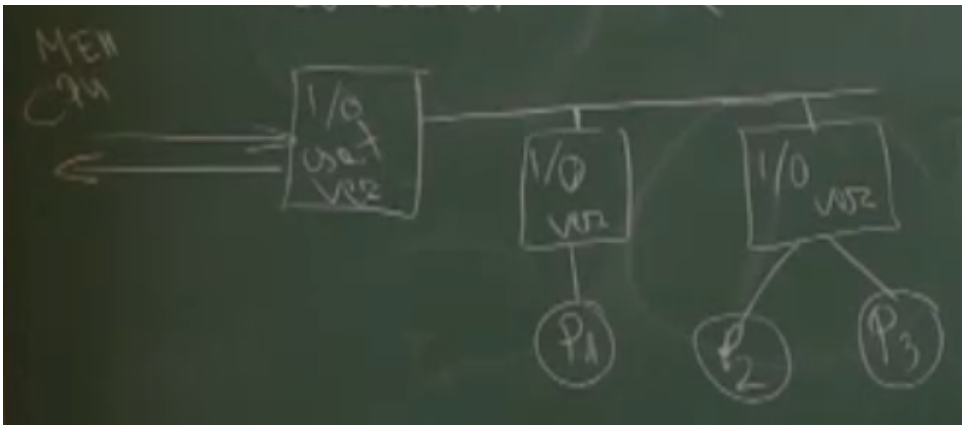
Az IO csatorna a DMA kiterjesztése lassabb perifériákhoz. Típusai:

- szelektor csatorna
- multiplexer csatorna

Ekkor sem hajt végre IO műveleteket a CPU. Az utasítások az IO vezérlő számára a memóriában vannak tárolva. Itt a processzor nem paraméterezi fel a DMA-t, hanem a memóriába írt programon keresztül specifikálja az adatátvitelt, amit a DMA végrehajt.

### 9.4.1. Szelektor csatorna felépítése

Nagyobb sebességű IO eszközökhöz való, egyetlen IO vonallal rendelkezik. Ennek oka, hogy egy perifériánál nem kell sokat várni az adat küldésére (kis mennyiség és viszonylag nagy sebesség), így elég egy csatorna is.

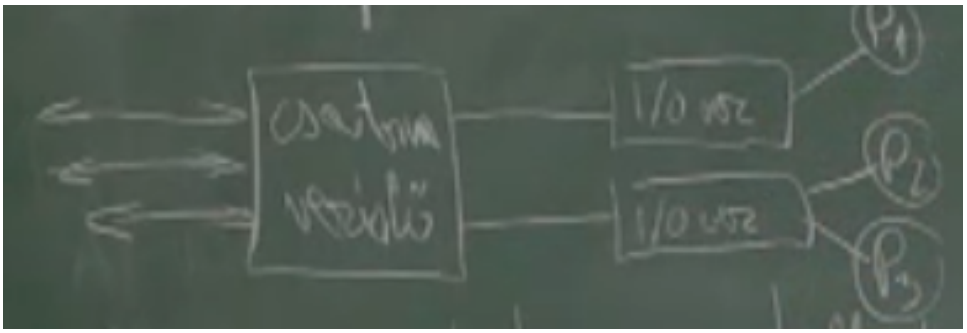



9.4. ábra. A szelektor csatorna típusú IO csatorna

Itt egyszerre csak egy egység kommunikálhat.

### 9.4.2. Multiplexer csatorna felépítése

Olyan eszközökhöz való, amik lassú adatátvitelre képesek. Itt van egy csatorna vezérlő, amire párhuzamosan több IO vezérlő csatlakozhat.



9.5. ábra. A multiplexer csatorna típusú IO csatorna

**Típusai**

- byte multiplexer - byteonként küldi az adatokat
- blokk multiplexer - blokkonként küldi az adatokat

A választás az adott eszközöktől függ, cél az átvitel maximalizálása.

# 10. fejezet

# Megszakítások

A megszakítás célja a feldolgozás vagy végrehajtás közben váratlannak tekinthető események kezelése. Nem csak a reagálás a cél ezekre az eseményekre, hanem a folyamatosan változó körülmények között az optimális működés biztosítása is.

## 10.1. Általános folyamata

Minden utasítás végrehajtás után van egy utasítás töréspont, amikor a processzor megvizsgálja, hogy érkezett-e megszakítás. Ha a válasz nem, akkor a következő utasítás következik. Ha viszont igen, a processzor megvizsgálja, hogy a megszakítás elfogadható-e. Amennyiben nem elfogadható, folytatja a következő utasítással, viszont ha igen, akkor megvizsgálja az adott megszakítást, menti az adott állapot kontextusát, kiszolgálja a megszakítást, visszatölti a mentett kontextust és jön a következő utasítás.

Fontos szempont, hogy megszakítás esetén az adott folyamat kontextusát tárolni kell. Ezután be kell tölteni a megszakítási rutin állapotát, végre kell hajtani a megszakítást és végül vissza kell adni a vezérlést.

## 10.2. Kialakulása

Kezdetben az IO adatátvitel gyorsítására alkalmazták a megszakításokat.

## 10.3. Megszakítási okok prioritási sorrendben

1. Géphibák: hibafigyelő áramkörök jelzései - általában nem letilthatók (pl. hőmérséklet szenzorok jelzései, ventillátor meghibásodása, paritás ellenőrzés, watchdog). Egy részük hibajelző kódok segítségével kommunikál.
2. IO források: IO befejezés, IO kezdeményezés, minden perifériákkal kapcsolatos állapotjelzés
3. Külső források: reset gomb, hálózati kommunikáció
4. Programozási (szoftveres) források
    - szándékos: pl. rendszerhívás
    - hibakezelés: mindig valamilyen utasítás végrehajtása során alakul ki

## 10.4. Szoftveres megszakítások

Hibákból származó programozási megszakítások lehetnek:

- memóriavédelem megsértése (több, saját memóriaterülettel rendelkező program esetén)
- tényleges tárkapacitás túlcímzése
- címzési előírások megsértése pl. 32 bites címek esetén csak minden negyedik byte címezhető
- aritmetikai, logikai végrehajtás közben fellépő hibák (underflow, overflow, tartomány túllépése, nullával osztás...)

## 10.5. Megszakítások csoportosítása

Megkülönböztetünk szinkron és aszinkron megszakításokat. A szinkron megszakítás egy adott program ugyanazon paramétereivel történő futtatása során mindig ugyanott jelentkezik (várható megszakítás). Az aszinkron megszakítások ezzel szemben viszont véletlenszerűen lépnek fel (nem várható megszakítás, pl. hardverhiba).

Csoportosíthatjuk a megszakításokat még aszerint is, hogy az utasítások végrehajtása között (pl. utasítás végrehajtásának eredménye képpen) vagy közben lépnek fel. Utasítások között felléphetnek pl. túlcsordulási hibák. Ezek kezelése rögtön az utasítás után elkezdődhet. A program folytatása a megszakítás kezelésének eredményétől függ.

Az utasítások közben felmerülő megszakítások nincsenek szinkronban a végrehajtási ciklussal. Ilyenek pl. a hardvermegszakítások. Ezek kezelése is az utasítás után kezdődhet meg.

A megszakításokat feloszthatjuk még aszerint is, hogy a programozó kérte, vagy nem kérte az adott megszakítást.

A megszakítás után a megszakított program folytatódhat, vagy be is fejeződhet.

További csoportosítás a maszkolható és nem maszkolható megszakítások. A maszkolható megszakítások letilthatók. Tipikus nem maszkolható megszakítások a hardverhibákat jelzők.

## 10.6. Megszakítás kiszolgálása

1. Előkészítés: ha valamelyik egység megszakítás kérést bocsát ki, aktiválódik az INTR vezérlővonal (interupt request)
2. Az első kérdés, hogy a megszakítás elfogadható-e. Régen a megszakítás vonalon csak egyszintű megszakítás lehetett, de ma már ez egy megszakítás áramkör, ami fogadja a bejövő megszakításokat és tudja kezelni a prioritásokat. Ez az áramkör van rákötve a CPU interrupt bementére, feladata eldönteni, hogy melyik megszakítás legyen kiszolgálva.
3. A megszakítás érvényre juthat, ha:
   (a) az aktuális folyamat (program vagy másik megszakítás) megszakítható
   (b) megfelelő a prioritás nagysága
   (c) az adott megszakítás nincs maszkolva, azaz letiltva
4. Ha a megszakítást elfogadta a megszakítás áramkör, az INTACK vezérlővonallal jelzi az elfogadást és aktiválja az INTR vonalat

### 10.6.1. A processzor feladata

- eltárolja az aktuális kontextust egy programtól független tárolóban
- betölti a PC-be a megszakítást kezelő utasítások kezdőpontját

## 10.7. Megszakításrendszerek szintjei

- egyszintű
- többszintű
  - egyvonalú
  - többvonalú

### 10.7.1. Egyszintű megszakítási rendszer

Egyszintű megszakításoknál addig nem kezdődhet másik megszakítás, amíg vissza nem térünk a normál állapotba. Tehát ilyenkor a megszakítások nem megszakíthatók, a megszakítások egymás után hajtódnak végre. Hátrány, hogy a magasabb prioritású megszakításoknál is előfordulhat, hogy várakozniuk kell.

### 10.7.2. Egyvonalú, többszintű megszakítási rendszer

Ennél a megszakítások is megszakíthatók, a megszakítások fontossági sorrendben futnak le. Ha egy megszakítás futása során egy magasabb prioritású megszakítás érkezik, az alacsonyabb prioritású megszakítást megszakítja a rendszer és elkezdi végrehajtani a magasabb prioritásút. A rendszer problémája, hogy a valóságban sok megszakítás van, ezért nem lehet minden megszakításhoz különböző prioritást rendelni.

### 10.7.3. Többszintű, többvonalú megszakítási rendszer

Fejlesztés az egyvonalú rendszerekhez képest, hogy megszakítási osztályokat hoznak létre és ezekhez az osztályokhoz rendelnek prioritási szinteket. Az osztályokon belül a megszakítástípusok egyszintű prioritást használnak.

## 10.8. Összefoglalás

A megszakítások lényege, hogy csak akkor terhelje a processzort, amikor az szükséges, de akkor minnél hamarabb alkalmazkodni tudjon a rendszer a felmerült körülményhez.

# 11. fejezet

# A tranzisztorok fejlődése

## 11.1. Bevezetés

A processzorokat szilícium szeletek, "waferek" felhasználásával gyártják. A waferek mérete folyamatosan növekszik a gyártástechnológia fejlődésével.

Egy integrált áramkör létrehozása 2-300 lépésből áll, kb 2-3 hónap. A gyártáshoz speciális gyárra van szükség.

A tranzisztorokat a minimális csíkszélességgel jellemezhetjük. A csíkszélesség mást jelenthet gyártónként, kisebb eltérések lehetnek.

A csíkszélesség lassan eléri a szilícium atomi méretét, ahol korlátokba ütközik a további csökkentés.

## 11.2. Moore törvény

Moore szerint a processzorokban a tranzisztorok száma két évente duplázódik, a csíkszélesség pedig csökken. Ma már laposodik a görbe.

## 11.3. Tranzisztorok típusai

- MOSFET: régebben használták őket, nagy frekvencián viszont nagyon melegedtek, ezért a teljesítmény nem volt tovább növelhető.
- feszített szilícium (2006-2009)
- HKMG (2012)
- FinFET (2014)

A fejlődés során a szükséges feszültség és a csíkszélesség is csökkent.

## 11.4. MOSFET tranzisztorok

## 11.5. Feszített szilícium technológia

## 11.6. HKMG tranzisztorok

## 11.7. FinFET tranzisztorok

A FinFET tranzisztorokat Tri-gate vagy 3D tranzisztoroknak is hívják.

Hasonló elven működig, mint a HKMG tranzisztorok, de a kialakítása változott.

A FinFET tranzisztorok teljesítménye kb. kétszerese a hagyományos tranzisztoroknak, a fogyasztás pedig kb. a 25-öd része. Az 1 Wattra jutó teljesítmény a hagyományos NMOS tranzisztorokhoz képest 7-8-szoros.

## 11.8. Slew rate

A slew rate (meredekségi ráta) adja meg a tranzisztoroknál a jelváltozás meredekségét (mennyi idő alatt vált kikapcsoltból bekapcsolt állapotba a tranzisztor).