ÓBUDAI EGYETEM

SZÁMÍTÓGÉP ARCHITEKTÚRÁK ALAPJAI

1-2. előadás

2025/2026/1

Durczy Levente

www.uni-obuda.hu

- **A számítógép architektúra fogalma**

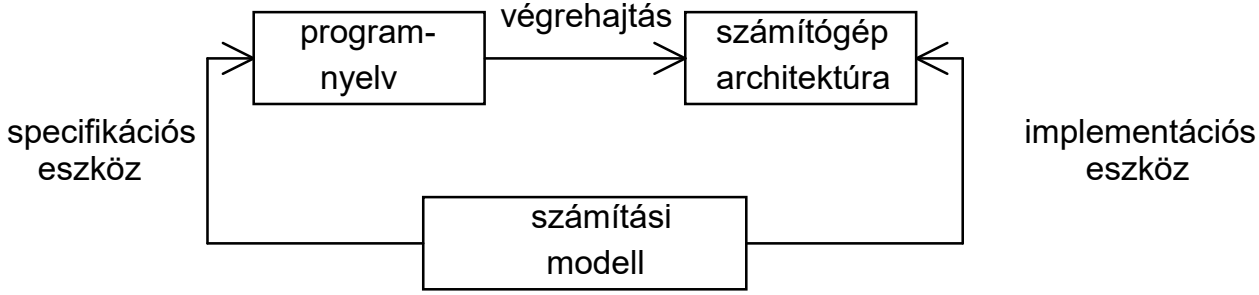
- A számítógép architektúra kifejezést 1964-ben az IBM 360-as gépcsalád "főkonstruktőre", **Amdahl** használta a minden idők egyik legsikeresebb számítógép-családja bejelentésekor. Az interpretálásuk a következő volt: "a számítógép struktúra, amit a gépi kódú programozónak értenie kell annak érdekében, hogy helyes programot tudjon írni az adott gépre". Értelmezésük szerint ez összegzi a **regiszterek**, a **memória**, az **utasítás-készlet**, a **címzési módok** valamint az **aktuális utasításkódok** meghatározását az implemetációval és a megvalósítással szembeállítva.

- Az implementációt, mint az aktuális hardver struktúrát, a megvalósítást pedig, mint a logikai technológiát és kapcsolódási pontot értelmezték.

- **A számítási modell:**

A számítási modell: a számításra vonatkozó alapelvek egy absztrakciója.

- - min hajtjuk végre a számítást;
- - hogyan van kifejezve a számítási feladat;
- - milyen úton vezérlik a végrehajtási sorrendet.



Számítási modellek csoportosítása:

1. Számítási modelljük szerint:
    a. szekvenciális
    b. párhuzamos
2. Vezérlés meghajtása szerint:
    a. vezérlés meghajtott
    b. adat meghajtott
    c. igény meghajtott
3. Probléma leírás szerint
    a. procedurális
    b. deklaratív

A három absztrakciós jellemzőből főként az alapján csoportosítjuk a modelleket, hogy:

Min hajtjuk végre a számítást?

1. Adat alapú modellek
    a. Neumann modell
    b. Adatfolyam modell
    c. Applikatív modell
2. Objektum alapú modellek
3. Predikátum logika alapú modellek
4. Tudás alapú modellek (AI, előre következtetés, visszafelé következtetés, …)
5. Hibrid modellek

ÓBUDAI EGYETEM

Adatalapú modellek:

Legfontosabb közös tulajdonságuk, hogy az adatok általában **típussal** rendelkeznek, azon belül is elemi (pl.: integer 16 bit,) vagy összetett adattípussal.
Az adattípus meghatározza az értelmezési tartományt, értékkészletet és az adaton elvégezhető műveletek halmazát. Ezeket a követelményjegyzékben és a programnyelvben pontosan meg kell határozni.

Neumann számítási modell

Absztrakciós jellemzők alapján:

- változók deklarálhatók,
- adatmanipuláló utasítások deklarálhatók
- vezérlésátadási utasítások deklarálhatók

imperatív (parancs) nyelvek

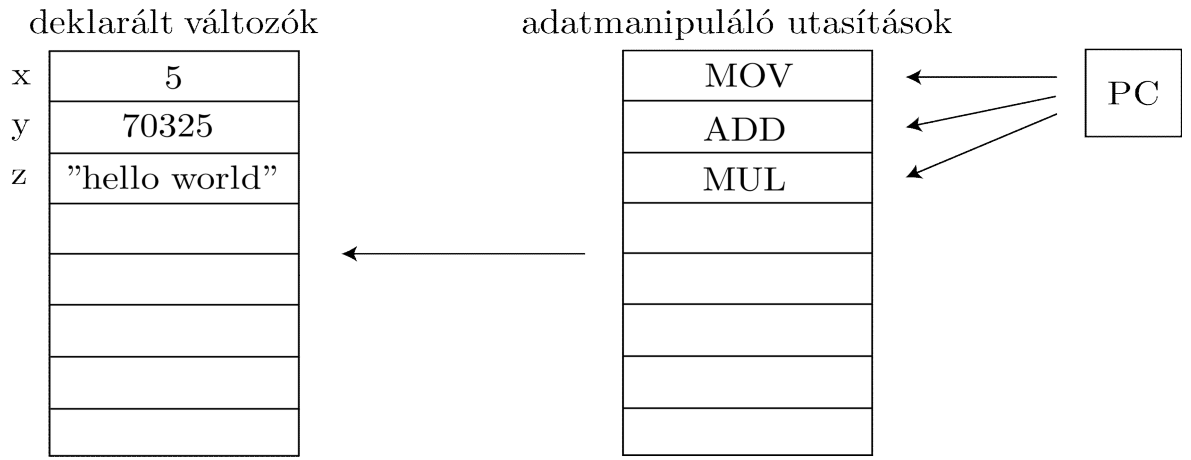
Min hajtjuk végre a számítást?

- A számítást adatokon hajtjuk végre
- Az adatokat változók képviselik
- Változók korlátlan számban változtathatnak értéket (**többszörös értékadás**)
- Adatok és utasítások azonos memóriaterületen helyezkednek el
- A számítási feladat elemi műveletek sorozataként értelmezhető

Adatalapú modellek:

Hogyan képezzük le a számítási feladatot?

Adatmanipuláló utasítások sorozatával:
A változók és az adatmanipuláló utasítások a **memóriában** helyezkednek el.
Az adatmanipuláló utasítások **szekvenciálisan megváltoztatják a változók értékét**.
A folyamat egy dedikált regiszter (Program Counter - PC) segítségével történik, ami mindig tartalmazza a következő utasítás címét.
Implicit statikus szekvencia.



Adatalapú modellek:

Milyen módon vezéreljük a végrehajtási sorrendet?

- Az implicit statikus szekvenciától eltérni csak **explicit vezérlésátadó utasításokkal** lehet (pl.: JMP ⟹ a PC értéke megváltozik!!!)
- Mivel a PC minden végrehajtás után inkrementál, ezáltal a következő utasítás címére fog mutatni, következésképpen vezérlésátadó utasításokkal befolyásolható a végrehajtási sorrend.
- Ezért hívjuk a Neumann modellt **vezérlés meghajtottnak**. A vezérlést a PC biztosítja, a **végrehajtás sorrendjét pedig a programozó**.

Következmények:

- **előzmény érzékenység**: …
- alapvetően szekvenciális végrehajtás
- egyszerűen implementálható
- adatmanipuláló utasítások állapot módosulást okozhatnak: bizonyos utasítások nem szándékolt állapot módosulást okozhatnak, például két 16 bites integer számot összeszorzunk túlcsordulás keletkezhet. ⟹ **Mellékhatás!** (kezelni kell a programvégrehajtás során)

Adatfolyam modell

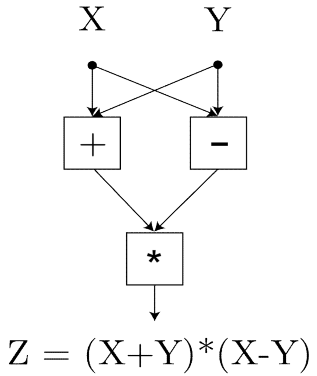
Absztrakciós jellemzők alapján:

Min hajtjuk végre a számítást?

- adatokon, de:
- Az adatokat bemenő adathalmaz képviseli
- Egyszeres értékadás lehetséges ⟹ nincs előzmény érzékenység!

Hogyan képezzük le a számítási feladatot?

- Adatfolyam gráf és input adatok halmaza segítségével
- Szakosodott végrehajtó egyégek (egyszerű, kicsi, gyors, több is lehet!)
- Párhuzamos végrehajtás lehetséges



Milyen módon vezéreljük a végrehajtási sorrendet?

- Adatvezérelt (Stréber modell): 1995-től az Intel Pentium Pro CPU-ban

- a végrehajtási sorrendet az adatok elérhetősége határozza meg. Amint megjelenik az adat, azonnal végrehajtja a műveletet, nem vár utasításra, nem vár PC-re (mivel nincs). **Az adatvezérelt program utasításai semmilyen szempontból nem rendezettek!**
- az adatok az utasításokon belül vannak tárolva (regiszterekben).
- érzékelni kell tudni az adatok rendelkezésre állását

Következmények:

- Nincs előzmény érzékenység
- Párhuzamos végrehajtás
- Nehezebben implementálható
  (igen magas a modell kommuniukációs és szinkronizációs igénye)

ÓBUDAI EGYETEM

Összehasonlítás

	Neumann modell	Adatfolyam modell
Min hajtjuk végre a számítást	adatokon	
Adatokat mik képviselik	változók	bemenő adathalmaz
Értékadás	többszörös	egyszeres
Adattárolás helye	közös operatív tár	utasításon belül (regiszterek)
Számítási feladat leképezése	adatmanipulációs utasítások	adatfolyam gráf
Végrehajtás	szekvenciális	Párhuzamos végrehajtás sok műveletvégző, azonnali műveletvégzés, szakosodott végrehajtó egységek
Végrehajtás vezérlése	vezérlés meghajtott: implicit szekvencia (PC), explicit vezérlésátadás	adatvezérelt: azonnali műveletvégzés amint lehetséges, adatok és utasítások nem rendezettek
Végrehajtás jellege	procedurális	
Következmények	előzményérzékeny	nincs előzményérzékenység
Implementáció	egyszerű	nehézkesebb (magasabb kommunikációs és szinkronizációs igény)

Logikai architektúra

Logikai architektúra az egy meghatározott leírási szinten a modell és a programozó által látott specifikáció, $\{M, S\}_L$ az úgynevezett funkcionális leírás (fekete dobozként tekintünk az architektúrára).

A számítógép szintű logikai architektúra rendszerszintű, azt vizsgáljuk milyen bemenetekre milyen eredményt produkál. Központi eleme az operációs rendszer, melyen keresztül vizsgálható.

Processzor szintű architektúra esetén ugyanúgy egy fekete dobozként tekintünk a processzorra, és a programozó feladata, hogy olyan bementet adjon amire az elvárt kimenet kapható. A processzornak van egy utasításkészlete (ISA). Minden programnyelven írt utasítás lefordítódik a processzor utasításkészletére és az alapján hajtódik végre. Komponensei:

- adattér
- adatmanipulációs fa
- állapottér
- állapotműveletek

Fizikai architektúra

A fizikai architektúra a modell és az implementáció együttes leírása adott absztrakciós szinten $\{M, I\}_L$

A számítógép szintű fizikai architektúra elemei:

- processzor
- memória
- Buszrendszer

A processzor szintű fizikai architektúra elemei:

- műveletvégző egység
- vezérlő
- I/O rendszer
- megszakítás rendszer

Adattér

Olyan tér, mely biztosítja az adatok tárolását oly módon, hogy azok a CPU által közvetlenül manipulálhatók legyenek.

Két részre osztható:

- memóriatér: nagyobb, lassabb, külső lapkán, olcsóbb
- regisztertér: kisebb gyorsabb, CPU lapkán, drágább

Memóriatér

Leginkább a mérete jellemzi. Minél több a memória, minél nagyobb a memóriatér, annál több adatot tudunk a processzor közelében tárolni.

Címtér

A memória eléréséhez címezni szükséges, hogy közvetlenül manipulálni tudjuk az adatot ehhez **címbuszra** van szükség, aminek így a mérete (szélessége) meghatározza a memóriatér maximális méretét.

Különbséget teszünk:

- modell címtere (elméleti címtér) és
- valós címtér (adott installációra jellemző) között

Virtuális és fizikai memória

Az elméleti és valós fizikai címtér különbsége, illetve a kis memória méretek miatt az 1960-as években megjelent a virtuális memória ötlete.

Ezáltal a memóriatér kettévált:

- a programozó által látott virtuális memóriára és
- a CPU által látott fizikai memóriára

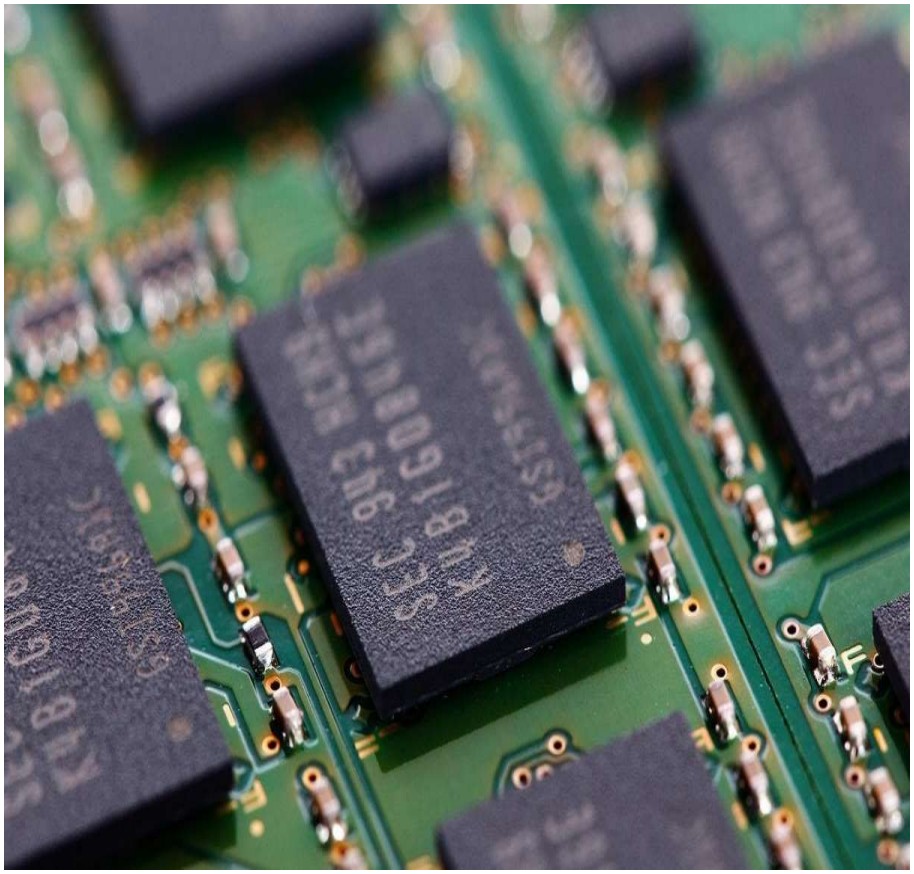
Átjárást a két memória között két folyamat tesz lehetővé:

- egy olyan egyirányú transzparens egyirányú folyamat, mely a program futása során a virtuális memória címeket dinamikusan (futási időben) valós, fizikai címekké alakítja (ezt az AGU, vagy MMU végzi), illetve
- egy olyan, a felhasználó számára transzparens kétirányú folyamat, amely a program futása közben az éppen nem használt adatokat a valós memóriából a virtuális memóriába mozgatja, majd szükség esetén visszaírja

ÓBUDAI EGYETEM

Összehasonlítás

	Virtuális memória	Fizika memória
Mérete	nagyobb	kisebb
Sebessége	lassabb	gyorsabb
Elhelyezkedés	háttértáron	lapkán
Láthatóság	Programozó látja	CPU látja
Mit tárol	Itt várnak az adatok	Itt fut a program



Regisztertér

Az adattér nagyteljesítményű, általában kis része.

Nem része a címtérnek (vagy saját címtere van, vagy huzalozottan vannak behúzva a regiszterek és címzésre nincsen szükség).

Három típust különböztetünk meg:

- Egyszerű regiszterkészlet
- Adattípusonként különböző regiszterek
- Többszörös regiszterkészlet

Egyszerű regiszterkészlet

- Egyetlen regiszter (akkumulátor regiszter)
  nehézkes, lassú
- Több dedikált adatregiszter
  mindegyiknek megvan a szerepe (például: forrásregiszter, célregiszter). A teljesítmény lényegesen nőtt. 1950-60-as évektől használják
- Univerzális regiszterkészlet
  jelentős teljesítmény növekedést jelentett és más programozási stílust eredményezett → gyakran használt változók a regiszterben maradhattak. 1960-70-es évektől
- Stack regiszter
  Előnye: mivel nem kell címezni és egyszerű utasítás → nagyon gyors
  Hátránya: operandus kiolvasás csak szekvenciálisan lehet

Adattípusonként különböző regiszterkészletek

Tulajdonképpen egyszerű regiszterkészletek csak különféle adattípusokhoz különálló regiszterkészletek. Amennyiben több műveletvégzőnk van, tudunk egyszerre több adattípussal műveletet végezni (FX, FP SIMD)

Pl.: 1998-ban az Intelnél bevezetett SIMD adattípus (multimédiás adattípus), mikor több adatot tárulunk egy regiszterben. Ehhez bevezették a SIMD regiszterkészletet.

Többszörös regiszterkészlet

Ez a legfejlettebb. Többek között egymásba ágyazott eljárások gyorsítására szolgál.

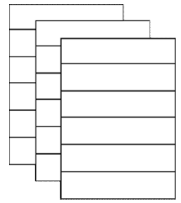
A regisztertér állapotát az állapottérrel együtt **kontextusnak** nevezzük. Amennyiben egymásba ágyazott eljárások esetén meghívunk egy új eljárást és a régi eljárás kontextusát a memóriába kell menteni az radikális teljesítmény csökkenést jelent. Kontextus váltást a kontextus kapcsolók végzik és ha ez regiszterkészletek között tud váltani akkor az nagyon gyors. Tehát a cél minden kontextus számára különálló regiszterkészlet biztosítása.

Ezenfelül általános regiszterkészletre is szükség van, ami biztosítja a regiszterek közötti kommunikációt

Három típust különböztetünk meg:

a) Több egymástól független regiszterkészlet

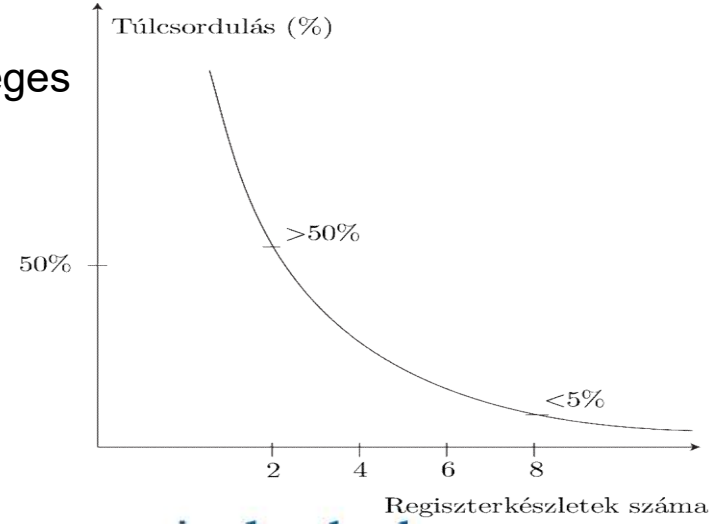
Nincs semmilyen átfedés köztük. 1970-80-as években használták. Paraméter átadás csak az operatív táron keresztül történhet.

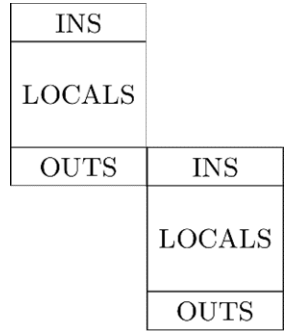


b) Átfedő regiszterkészlet (INS-LOCALS-OUTS)

Kifejezetten egymásba ágyazott eljárásokhoz fejlesztettek ki. Lényege, hogy egy regiszter lapot három részre bontottak: INS (bemenő paraméterek), LOCALS (lokális paraméterek) és OUTS (kimenő paraméterek). A regisztertéren belül a kimenő és bemenő regiszterek ugyanazon a címen voltak, ezzel a paraméterátadás nagyon könnyen megoldható volt.

Hátrány: merev, túlcsordulás lehetséges





c) Stack cache (1982-től)

A stack és közvetlen elérésű cache kombinálása.

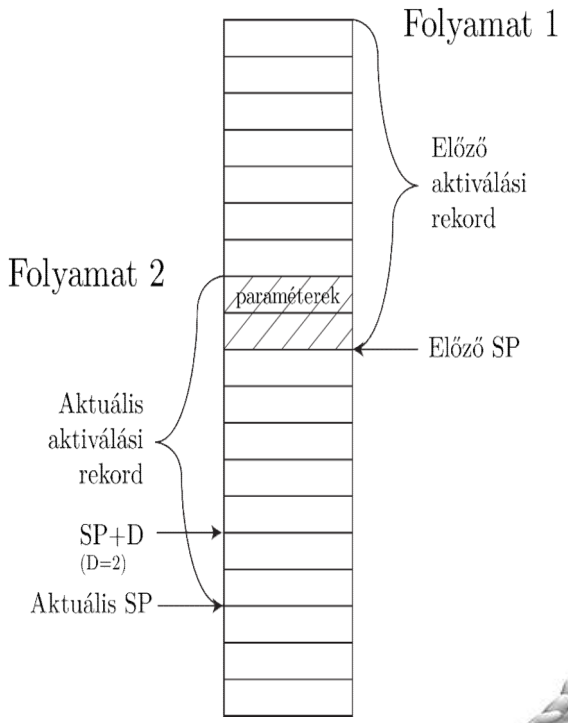
Amikor egy folyamatnak szüksége van egy regiszterkészletre, akkor a compiler kijelöl a stack pointerrel (SP) a regisztertérben egy pontot és attól számított meghatározott számú regisztert allokál (aktiválási rekord).

A SP mutat az aktiválási rekord első elemére, de nem csak itt lehet címezni, mert eltolási pointer (displacement) segítségével bármely regisztere közvetlenül címezhető.

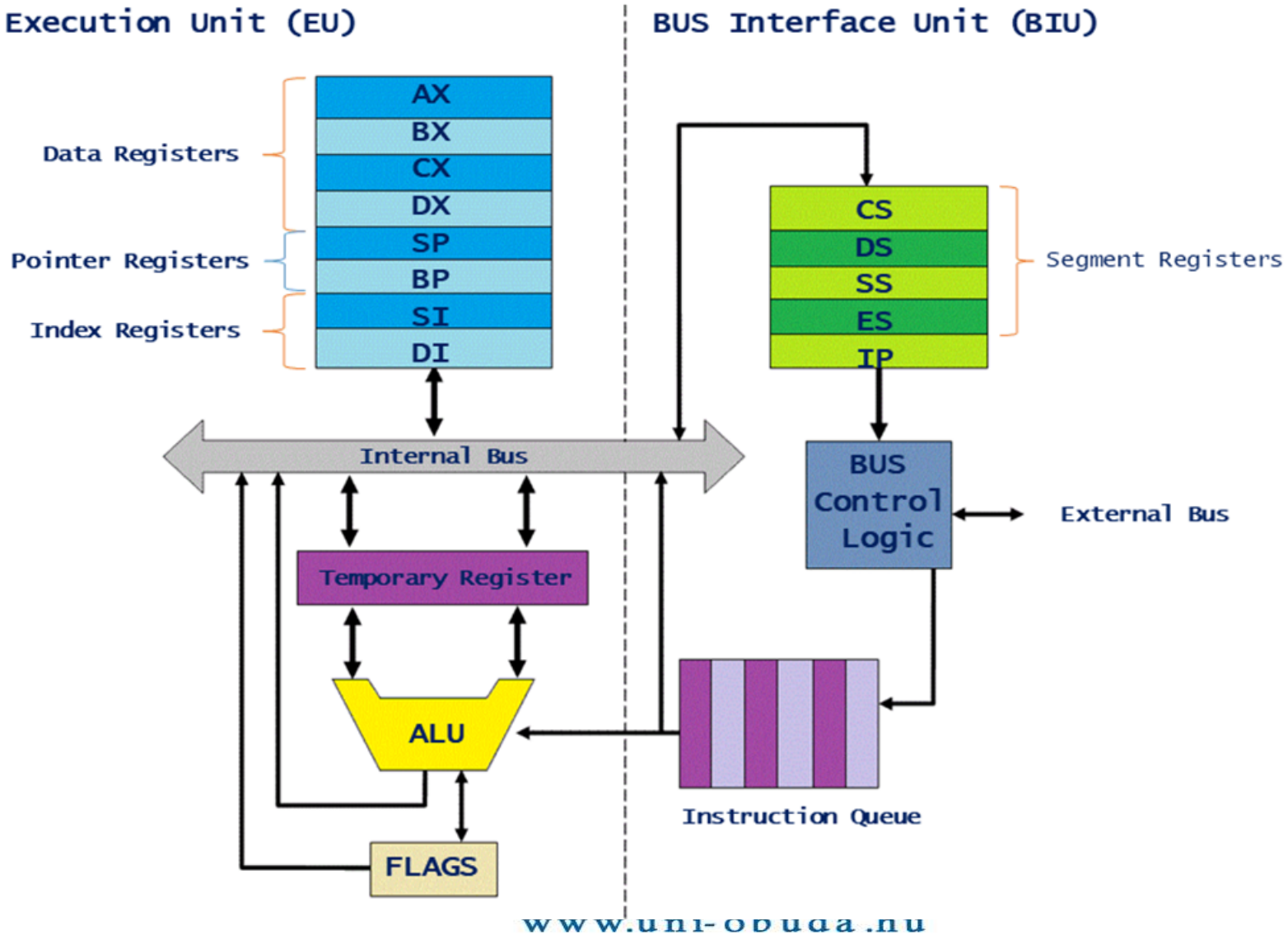
Aktiválási rekord bizonyos korlátok között bármilyen hosszú lehet → rugalmas kiosztás → nincsenek üres helyek és nincs túlcsordulás!

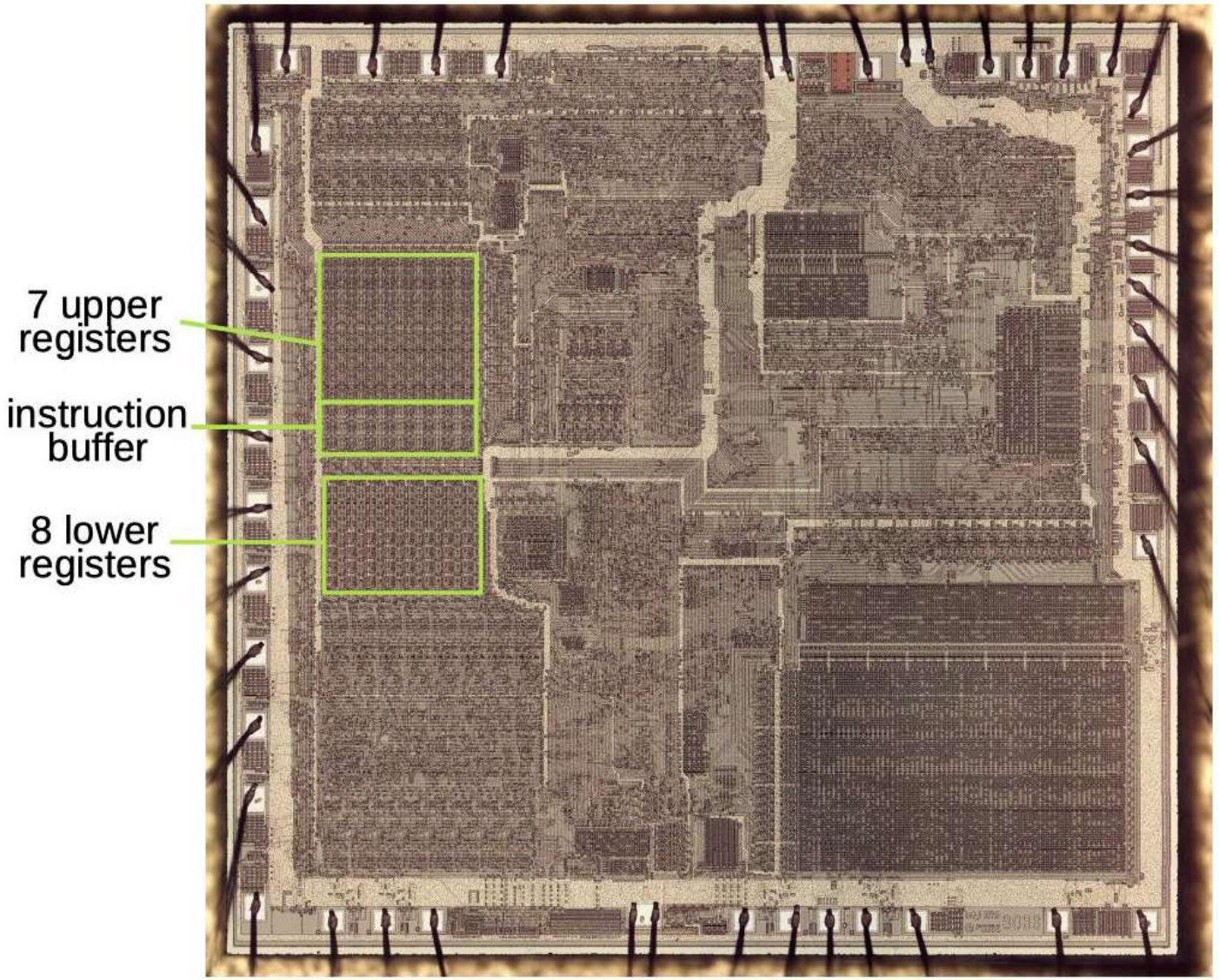
Az egymást követő eljárások aktiválási rekordjai átfedik egymást (input-output paraméter átadás megoldott!)

Minden folyamatnak a compiler pontosan úgy fogja lefoglalni az aktiválási rekordját, hogy az átfedő regiszterekben benne legyenek az átadandó paraméterek.



- **Intel 8086 CPU regiszter block diagramm**



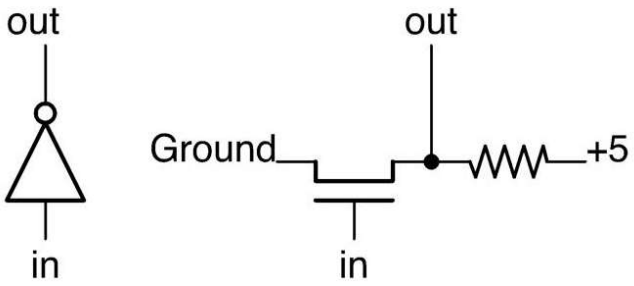


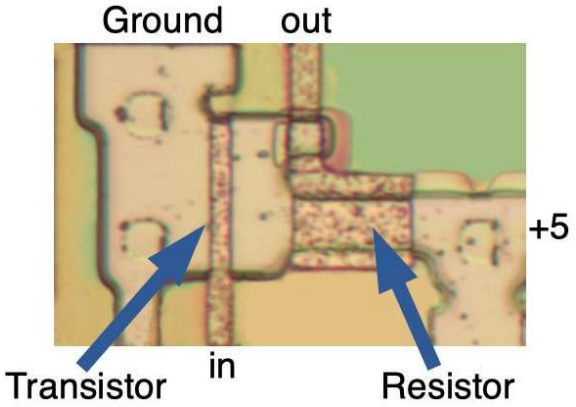
Intel 8086 CPU:

regiszterek

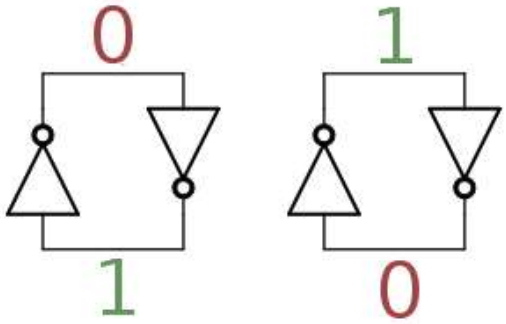
(15 db 16 bites reg. +
6 Byte utasítás puffer)

Regiszter fizikai kialakítása:



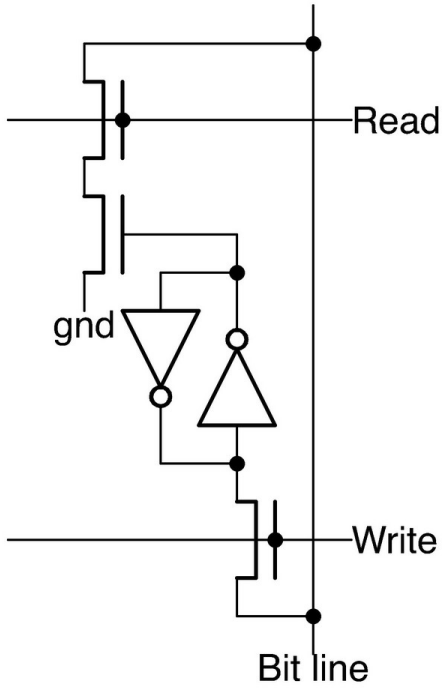


Inverter pár: ez tárolja a bitet



Regiszter cella felépítése:

- 3 plusz tranzisztor
- 1 olvasáshoz, 1 íráshoz,
- 1 jelerősítéshez



Adatmanipulációs fa

Definíció

Megmutatja a potenciális adatmanipulációs lehetőségeket.

Bizonyos részfái megmutatják egy konkrét implementáció adatmanipulációs lehetőségeit.

1. Szintjei

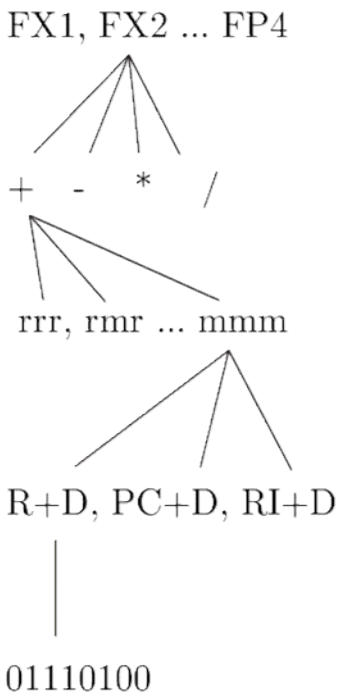
1. Adattípusok: leírja, hogy milyen adattípusokat értelmezünk az adott arcgitektúrán

2. Műveletek: megmutatja, hogy a adott adattípusokon milyeen műveletek értelmezhetőek

3. Operandus típusai: megkülönböztetjük az operandusokat számuk és típusaik szerint

4. Címzési módok: engedélyezett címzések

5. Gépi kód: minden architektúrán más



FX: fixpontos
FP: lebegőpontos
r: regiszter
m: memória
D: eltolás
PC: program számláló
RI: indexregiszter

Adattípusok

- elemi
- összetett (adatszerkezet)

Összetett adattípusok

Elemei adattípusokból épülnek fel.

- Ha különböző típusú részekből áll: - rekord-nak nevezzük
- azonos típusokból áll össze:
  - tömb: (az 1 dimenziós tömb neve: **vektor**)
  - szöveg
  - verem
  - sor (FIFO)
  - lista
  - fa (általában bináris)
  - halmaz

Elemi adattípusok

- numerikus
- karakteres
- logikai
- pixel

Numerikus elemi adattípusok:

1. **FX (fixpontos):**
   egyes komplemens - kettes komplemens - többletes kódolás
   előjeles - előjel nélküli
   1 byte (félszó) - 2 byte (szó) - 4 byte (dupla szó)
2. **FP (lebegőpontos):**
   nem normalizált - normalizált → hexára (tipikusan IBM gépeknél)
   → binárisra
   Méret szabvány szerint (IEEE 754):
   egyszeres pontosságú (32 bit) - kétszeres pontosság (64 bit) - kiterjesztett pontosságú (128 bit)
3. **BCD (binárisan kódolt decimális):**
   Pakolt byte (1 byte-on 2db decimális szám) - pakolatlan (zónázott)

Karakteres elemi adattípusok:

1. **ASCII**
   7 bites (szabványos) - 8 bites (kiterjesztett)
2. **UNICODE (2 byte)**

Műveletek:

Az adatmanipulációs fa minden művelet esetén megállapítja, hogy milyen utasítás típusok vannak megengedve és milyen operandus típus választható.

Utasítás végrehajtás:

Utasítás (def.): A számítógép által végrehajtható alapvető feladatok ellátására szolgáló elemi művelet leírása.

A processzor csak gépi kódú utasításokat tud értelmezni, ennek általános formája:

MK	Címrész

- műveleti kód: tartalmazza, hogy MIT kell csinálni (utasítás mező)
-címrész: megmondja, hogy hol található, hogy MIVEL (operandus mező)

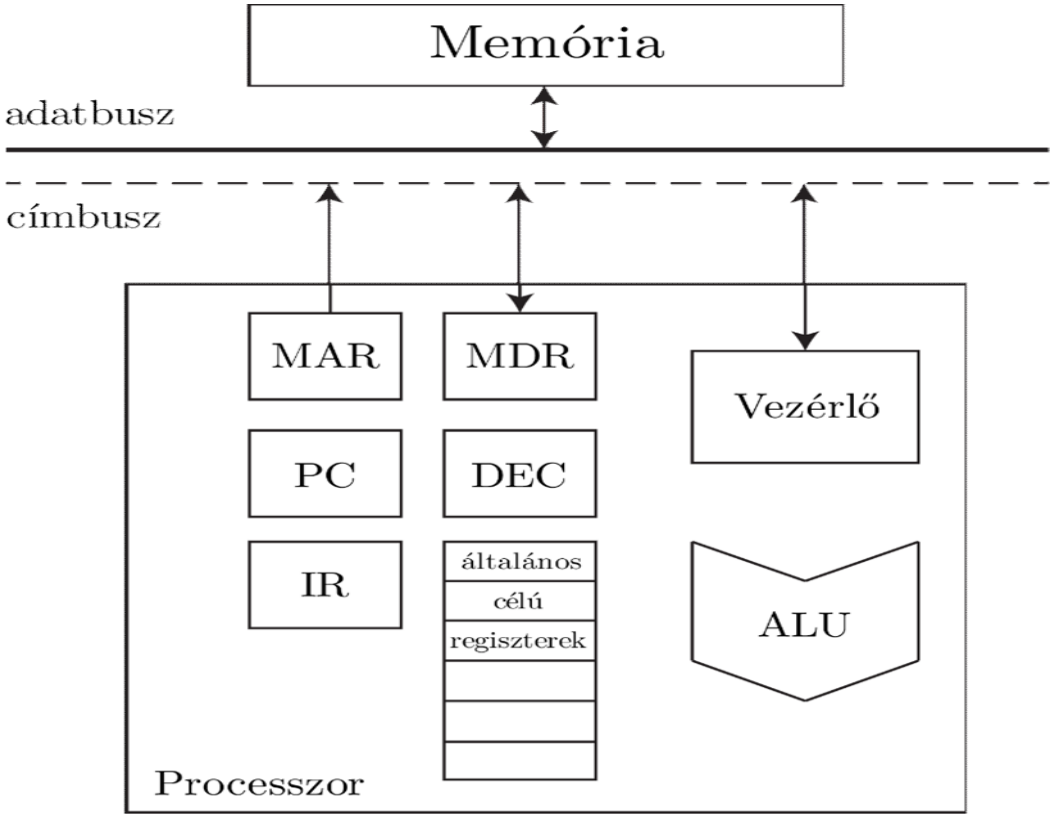
Szekvenciális utasításvégrehajtás menete processzor szinten:

Processzor regiszterei:

- Memory Address Register (MAR)
- Memory Data Register (MDR)
- Program Counter (PC)
- Instruction Register (IR)
- általános célú regiszterek

Processzor részei:

- Dekóder (DEC)
- Vezérlő
- Aritmecital Logical Unit (ALU), benne található az Accumulator Register (AC)



ÓBUDAI EGYETEM

Egy gépi kódú utasítás elemi műveletek sorozataként írható fel (pl.: ADD m_1, m_2)

1. FETCH (utasítás lehívás):
   - MAR ⟸ PC
   - MDR ⟸ [MAR] MAR által mutatott memóriacímen lévő értéket töltjük be
   - IR ⟸ MDR
   - PC ⟸ PC+1
2. EXECUTION 1: dekódolás és első operandus betöltés
   - DEC ⟸ IR
   - MAR ⟸ $DEC_{címrész}$
   - MDR ⟸ [MAR]
   - AC ⟸ MDR
3. EXECUTION 2:
   - DEC ⟸ IR
   - MAR ⟸ $DEC_{címrész}$
   - MDR ⟸ [MAR]
   - AC ⟸ AC+MDR
4. STORE:
   - DEC ⟸ IR
   - MAR ⟸ $DEC_{címrész}$
   - MDR ⟸ AC
   - [MAR] ⟸ MDR

5. Vezérlés átadás (JMP):
   - DEC ⟸ IR
   - MAR ⟸ $DEC_{címrész}$
   - PC ⟸ [MAR]

Utasítás típusok:

Egy utasításnál a műveleti kód mellett megtalálhatók az operandusok (/cím), mely lehet forrás operandus (s) vagy cél operandus (d). Egy tetszőleges műveletet úgy írhatunk fel:

opd := ops1 @ ops2

Ennek függvényében az utasításokat az operandusok (címek) számában különböztetjük meg:

- **4 címes:** d, ops1, ops2, következő utasítás címe

  Nehézkes és merev struktúra.

- **3 címes:** d, ops1, ops2.

  (itt PC tárolja a következő utasítás címét, mely autoinkrementál)

  Előny: - cél operandussal nem kell egyik forrás operandust felülírni
  - párhuzamosítás lehetősége

  Hátrány: - még mindig hosszú utasítások, sok memóriát igényel

- **2 címes:** ops1, ops2. (az eredmény felülírja az ops1-et: ops1= ops1@ops2)

  mai CISC architektúrákban alkalmazzák

- **1 címes:**

  be kell tölteni az operandust AC-be, majd a második operandussal az AC tartalmát módosítjuk. Például LOAD[100] majd ADD[101]

  Előny: utasítások rövidebbek, gyorsabbak

  Hátrány: utasítások száma nő

- **0 címes utasítás:** (NOP, CLEAR D, RST)
  Előny: rövid, gyors
  Hátrány: növeli az utasításkészletet

Operandus típusok:

- AC akkumulátor (E: gyors H: csak egy van belőle)
- memória (m) (E: nagy méret H: hosszú címe van, lassú)
- regiszter (r) (E: gyors H: kevés van belőle)
- verem (s) (E: gyors H: csak a tetejét látjuk (szűk keresztmetszet))
- immediate (i) Közvetlenül beírt érték. Például: ADD r1, 3
  (E: gyors, nem szükséges hozzá regiszter H: csak programból változtatható)

Szabályos architektúrák:

Homogének, ugyanazt az operandus típust biztosítják valamennyi adatmanipuláció esetén
Pl.: a RISC architektúrák ún. „regiszter architektúrák”
A memória csak és kizárólag LOAD és STORE utasításokkal címezhető és érhető el!

Címzési módok:

A címzési mód maga a címszámítási algoritmus. Három egymástól független elem kombinációja:

1. Címszámítás: jelzi, hogy abszolút vagy relatív címzést használunk
   - Abszolút cím: pontosan megmondja melyik címet használjuk, a teljes címet tartalmazza
   - Relatív címzés: egy bázis címhez képest számoljuk ⟶ rövidebb
     (ehhez deklarálni szükséges egy bázis címet és a címszámítási algoritmust!)

   Mivel a CPU címtere nagyon nagy (~ 4 - 64 TB), azért inkább a relatív címzést használják:
   bázis (S) + eltolási cím (D)

   Gyakorlatilag az eltolás mérete határozza meg, hogy mekkorára csökken az adott címtér.

   Bázis cím lehet: PC, top of STACK, R_i (index regiszter), …

2. Cím módosítás (opcionális):
   Arra szolgál, hogy a következő operandus címét minél egyszerűbben meghatározzuk.
   (pl: indexelés, autoinkrementálás, autodekrementálás)

   Ide tartozik az adatblokkok beolvasásakor szükséges címmódosítás is!

3. Deklarált (tényleges) cím meghatározása:
   A címet direkt vagy indirekt, illetve valós vagy virtuális címként tekintjük.

Indexelés:

Hatékony mód az adatblokk egyes elemeinek elérésére.

A beolvasás nem egyesével, hanem blokkokban történik.

Az adatblokk kezdőcíme lesz a bázis cím (S), R_i regiszter fogja tartalmazni az eltolást ($R_i = D$).

Bármely Y_e (effektív cím) = S + $[R_i]$ megadható.

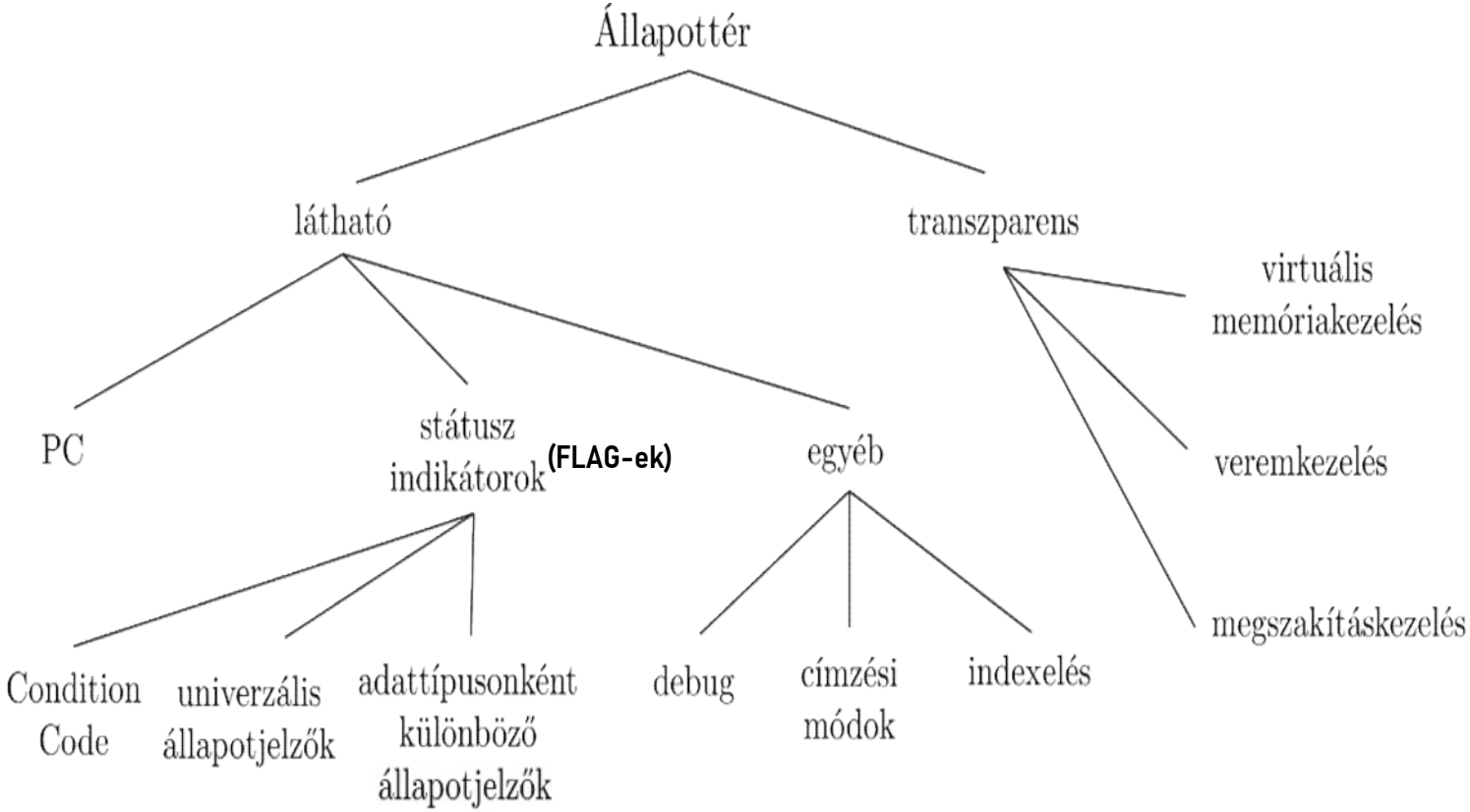
Két dimenziós blokk esetén:

$Y_e = S + [R_{i1}] + [R_{i2}]$

Állapottér:

Def.: Olyan programból látható és nem látható (program transzparens) tárolókból áll, melyek az adott programra vonatkozó állapotinformációkat hordozzák.

Felosztása:



Indikátorok (flagek):

Olyan kivételes események figyelésére és vezérlése szolgálnak, melyek a program futása közben általánosságban jelenhetnek meg (pl.: carry, overflow, …).

Minden regiszterkészlet típushoz definiálnak külön státusz indikátor készletet!

Állapotműveletek:

Az állapotjelzőket speciális utasításokkal lehet manipulálni.

PC állapotműveletei:

- inkrementálás
- dekrementálás
- felülírás (utasításból vett címmel)

Flagek állapotműveletei:

- Save
- Set
- Reset
- Load
- Clear

Logikai architektúra vége!