Számítógép architektúrák alapjai

Hallgatói jegyzet

Processzor szintű logikai architektúra

2023

Tartalom

1. Adattér 1
   1.1. Definíció 1
   1.2. Memóriatér 1
      1.2.1. Címtér 1
      1.2.2. Virtuális és fizikai memória 1
   1.3. Regisztertér 2
      1.3.1. Egyszerű regiszterkészlet 2
      1.3.2. Adattípusonként különböző regiszterkészletek 2
      1.3.3. Többszörös regiszterkészlet 3
2. Adatmanipulációs fa 5
   2.1. Definíció 5
   2.2. Szintjei 5
   2.3. Adattípusok 5
      2.3.1. Összetett adattípusok 5
      2.3.2. Elemi adattípusok 5
   2.4. Műveletek 7
      2.4.1. Utasítás végrehajtás 7
      2.4.2. Utasítás típusok 9
   2.5. Operandus típusok 9
   2.6. Címzési módok 11
3. Állapottér 12
   3.1. Definíció 12
   3.2. Felosztás 12
   3.3. Állapotjelzők / státusz indikátorok (flagek) 13
4. Állapotműveletek 13

Készítette: Kováts Máté

A jegyzet Durczy Levente 2019 őszi Számítógép architektúrák alapjai előadás videói alapján készült.

Segítséget jelentett Uhrin Ádám és Nagy Enikő korábbi jegyzetei.

1. Adattér

1.1. Definíció

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

Két típusa van:

- memóriatér
- regisztertér

1.2. Memóriatér

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

1.2.1. Címtér

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

Különbséget teszünk:

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

1.2.2. 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:

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

	Virtuális	Valós
Mérete	nagyobb	kisebb
Sebessége	lassabb	gyorsabb
Elhelyezkedés	háttértáron	lapkán
Láthatóság	CPU látja	programozó látja
Mit tárol	adatok	futó program

1.3. Regisztertér

Az adattér nagyteljesítményű, általában kis része. Nem része a címtérnek, ami azt jelenti, hogy vagy saját címtere van, vagy huzalozottan vannak behúzva a regiszterek és címzésre nincsen szükség.

Három típust különböztetünk meg:

- Egyszerű regiszterkészlet
- Adattípusonként különböző regiszterek

1.3.1. Egyszerű regiszterkészlet

Kronológiai sorrendben több típusa van:

- Egyetlen regiszter (akkumulátor regiszter)
  Nagy korlátozást és teljesítménycsökkenést eredményez, mivel állandóan a memóriához kell fordulni, ami lényegesen lassabb. Az 1940-es években használták.
- Több dedikált adatregiszter
  Már több regiszter hoztak létre a processzoron belül. Dedikált, mert mindegyiknek megvolt a szerepe (például: forrásregiszter, célregiszter). A teljesítmény lényegesen nőtt. 1950-60-as években használták.
- Univerzális regiszterkészlet
  Szabadon használható regiszterekkel látták el a processzorokat. Ezekkel a programozó már szabadon tervezhetett, ami jelentős teljesítmény növekedés jelentett és más programozási stílust eredményezett → gyakran használt változók folyamatosan a regiszterben maradhattak. 1960-70-es évek jellemző architektúrái használták.
- Stack regiszter
  Előnye: mert nem kell címezni és egyszerű utasítás → nagyon gyors.
  Hátránya: operandus kiolvasás csak szekvenciálisan lehet, mivel csak a legfelső adathoz férünk hozzá. Univerzális regiszterkészlettel párhuzamosan az 1960-70-es években jelent meg.

1.3.2. Adattípusonként különböző regiszterkészletek

Tulajdonképpen egyszerű regiszterkészletek csak különféle adattípusokhoz különálló regiszterkészletek. Amennyiben több műveletvégzőnk van tudunk egyszerre több adattípussal műveletet végezni (FX, FP) Leginkább a lebegőpontos adattípus használata esetén mutatkozik meg az előnye, mivel lebegőpontos számításoknál a mantisszával és karakterisztikával végzett műveleteknél is párhuzamosan működés érhető el, ami jelentős teljesítmény növekedést jelent.

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

1.3.3. Többszörös regiszterkészlet

Ez a legfejlettebb. Lényege, hogy egymásba ágyazott eljárások gyorsítására szolgál.

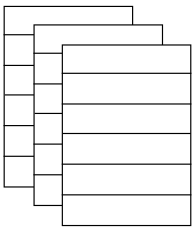
A regisztertér állapotát az állapottérrel együtt kontextusnak nevezzük. Amennyiben egymásba ágyazott eljárások esetén meghívunk egy új eljárást és a régi eljárás kontextusát a memóriába kell menteni az radikális teljesítmény csökkenést jelent. Kontextus váltást a kontextus kapcsolók végzik és ha ez regiszterkészletek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

Ezenfelül általános regiszterkészletre van szükség, ami biztosítja a regiszterek közötti kommunikációt. Hiszen a hívott eljárásnak szüksége lehet olyan adatra, mely a hívó eljárás kontextusában van.

Három típust különböztetünk meg:

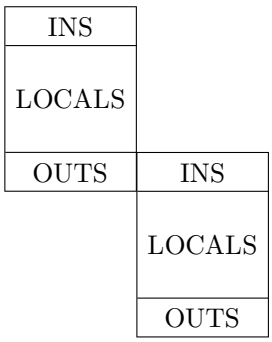
a) Több egymástól független regiszterkészlet

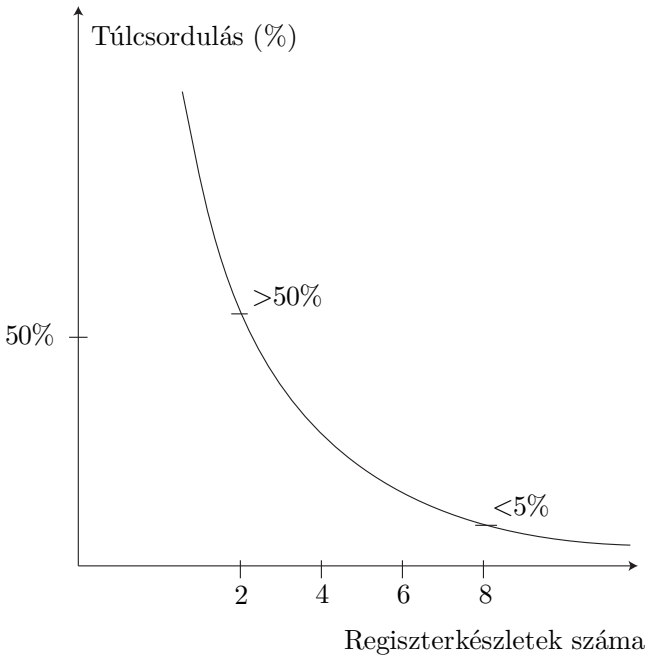
   Nincs semmilyen átfedés köztük. 1970-80-as években használták. Paraméter átadás csak az operatív táron keresztül történhet.



b) Átfedő regiszterkészlet

   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. Hátránya, hogy merev. Túlcsordulás a fix regiszterkiosztásból és regiszterkészletek számából fakadt



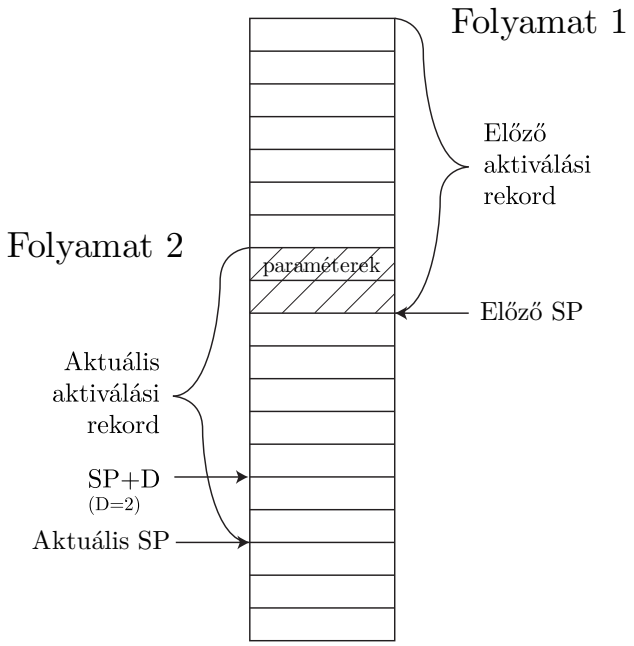


c) Stack cache (1982)

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). 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.

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



2. Adatmanipulációs fa

2.1. Definíció

Megmutatja a potenciális adatmanipulációs lehetőségeket. Bizonyos részfái megmutatják egy konkrét implementáció adatmanipulációs lehetőségeit.

2.2. Szintjei

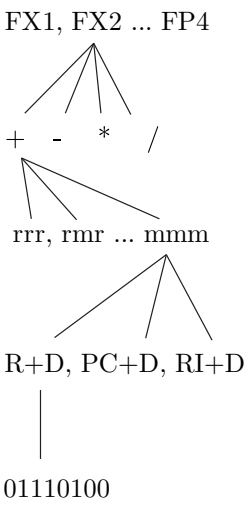
1. Adattípusok: leírja, hogy milyen adattípusokat értelmezünk az adott arcgitektúrán

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

2.3. Adattípusok

- elemi
- összetett (adatszerkezet)

2.3.1. Összetett adattípusok

Elemei adattípusokból épülnek fel.

- különböző részekből áll össze: rekord
- azonos típusokból áll össze:
    - tömb: 1D-s tömb neve a vektor
    - szöveg
    - verem
    - sor (FIFO)
    - lista
    - fa (általában bináris)
    - halmaz

2.3.2. Elemi adattípusok

- numerikus
- karakteres
- logikai

- pixel

Numerikus elemi adattípusok:

1. **BCD** (binárisan kódolt decimális)
   - pakolt: 1 byte-on 2db decimális szám
   - pakolatlan (zónázott): 1 byte-on 1db decimális szám
2. **FX** (fixpontos)

   Kódolás szerint különböztetjük meg.

   - egyes komplemens
   - kettes komplemens
     - előjeles
       - 1 byte (félszó)
       - 2 byte (szó)
       - 4 byte (dupla szó)
       - 16 byte (quadro szó)
     - előjel nélküli
   - többletes kódolás
3. **FP** (lebegőpontos)
   - nem normalizált
   - normalizált
     - hexára (tipikusan IBM gépeknél)
     - binárisra

       szabványok szerint:
       - VAX
       - IEEE 754
         - egyszeres pontosságú (32 bit)
         - kétszeres pontosság (64 bit)
         - kiterjesztett pontosságú (128 bit)

Karakteres elemi adattípusok:

- EBDIC (8 bit)

  1960-as évektől IBM-nél használt
- ASCII
  - 7 bites (szabványos)
  - 8 bites (kiterjesztett)
- UNICODE (2 byte)

Logikai elemi adattípusok:

Általában 1 bit értékes és ez a legmagasabb helyiérték.

Például: AND, OR

- 1 byte
- 2 byte
- 4 byte
- változó hosszúságú

2.4. Műveletek

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

2.4.1. Utasítás végrehajtás

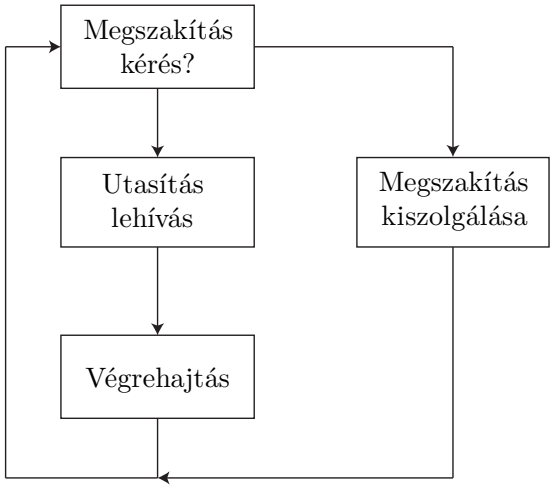
Utasítás definíciója: A számítógép által végrehajtható alapvető feladatok ellátására szolgáló elemi művelet leírása.

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

- műveleti kód: tartalmazza, hogy MIT kell csinálni
- címrész: megmondja, hogy hol található, hogy MIVEL

Utasítás végrehajtás:

A processzor megnézi, hogy érkezett-e megszakítás kérés. Amennyiben nincs, történik egy utasítás lehívás (FETCH), végrehajtás (EXECUTION) és a végén újra megszakítás ellenőrzés. Ha történt megszakítás kérés, akkor azt ki kell szolgálni.



Szekvenciális utasításvégrehajtás menete processzor szinten:

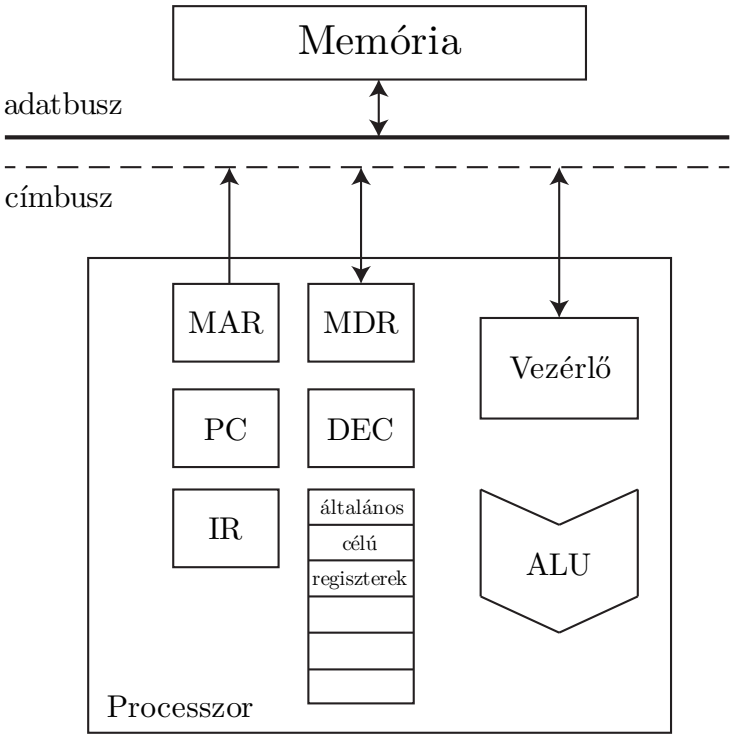
Processzor regiszterei:

- 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 Accumulator Register (AC)



Egy gépi kódú utasítás elemi műveletek sorozataként írható fel:

1. FETCH

   MAR ← PC — innen tudjuk honnan kell lehívni a következő utasítást
   MDR ← [MAR] — MAR által mutatott címen lévő értéket töltjük be
   IR ← MDR
   PC ← PC+1

2. EXECUTION — elemi műveletek
   1. LOAD — dekódolás és operandus betöltés

      DEC ← IR
      MAR ← DECcímrész
      MDR ← [MAR]
      AC ← AC

   2. EXECUTION — ADD r1, r2 esetén

      DEC ← IR
      MAR ← $DEC_{címrész}$
      MDR ← [MAR]
      AC ← AC+MDR

   3. STORE

      DEC ← IR
      MAR ← $DEC_{címrész}$
      MDR ← AC
      [MAR] ← MDR

4. INTERRUPTION például JMP
   DEC ← IR
   PC ← $DEC_{címrész}$

2.4.2. Utasítás típusok

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

$$opd := ops_1 \ @ \ ops_2$$

Ennek függvényében az utasításokat a címek számában különböztetjük meg:

- **4 címes**: d, ops_1, ops_2, következő utasítás címe
  Nehézkes és merev struktúra.
- **3 címes**: d, ops_1, ops_2. PC tárolja a következő utasítás címét, mely autoinkrementál.
  Mai RISC architektúrákban alkalmazzák.
  Előny:
  - cél operandussal nem kell egyik forrás operandust felülírni
  - párhuzamosítási lehetősége

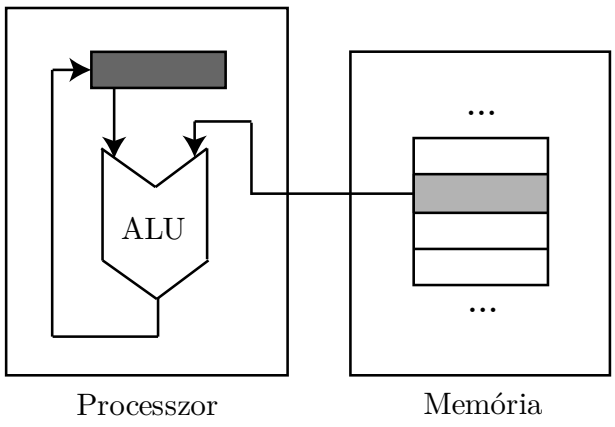
  Hátrány: még mindig hosszú utasítások, sok memóriát igényel
- **2 címes**: ops_1, ops_2. Az eredmény felülírja az ops_1-et: $ops_1 = ops_1 @ ops_2$
  Mai CISC architektúrákban alkalmazzák
- **1 címes**: be kell tölteni az operandust AC-be, majd a második operandussal az AC tartalmát módosítjuk. Például LOAD[100] majd ADD[101]
  Előny: utasítások rövidebbek, gyorsabbak
  Hátrány: utasítások száma nő
- **0 címes:** Példák: NOP, CLEAR D, PUSH, POP

  Előny: rövid

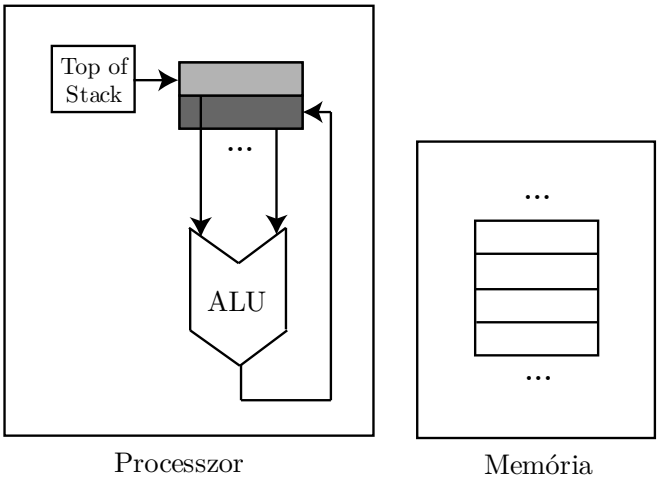
  Hátrány: növeli az utasításkészletet

2.5. Operandus típusok

- AC akkumulátor
  Előnye: gyors
  Hátránya: csak egy van belőle

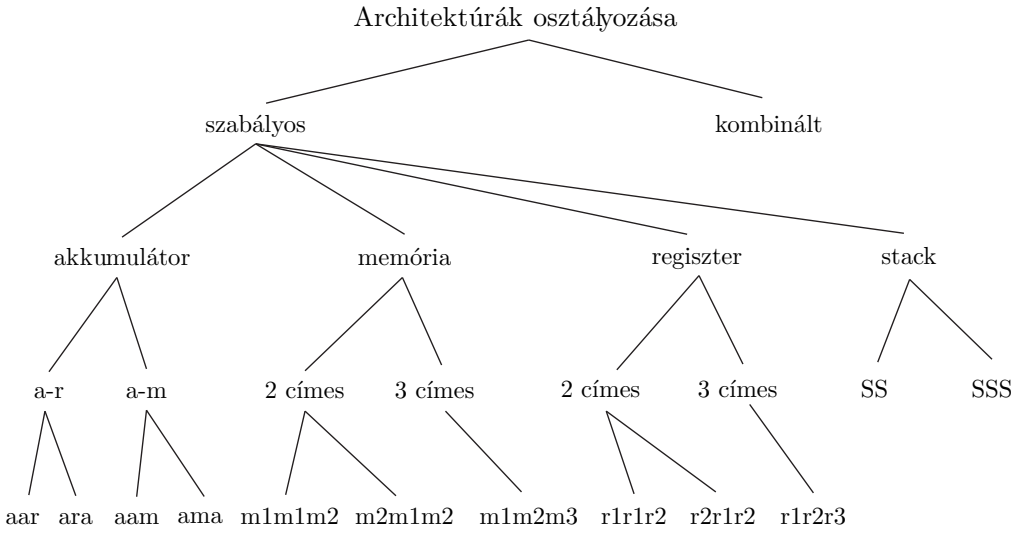


- memória (m)
  Előnye: nagy méret
  Hátránya: hosszú címe van, lassú
- regiszter (r)
  Előnye: gyors
  Hátránya: korlátozott számú
- verem (s)
  Előnye: gyors
  Hátránya: csak a tetejét látjuk (szűk keresztmetszet)



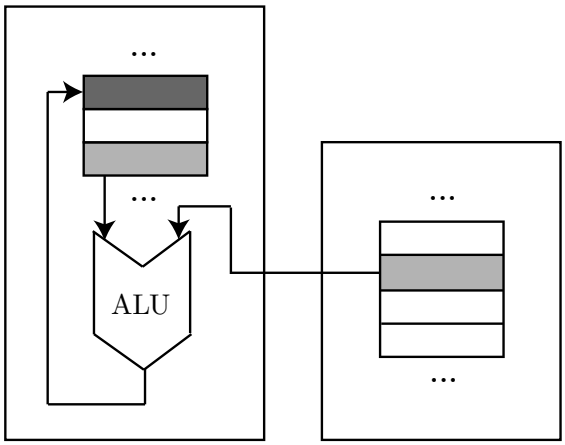
- immediate (i)
  Közvetlenül beírt érték. Például: ADD r1, 3
  Előnye: gyors, mert nem szükséges hozzá regiszter
  Hátránya: csak programból változtatható.

Operandus típus szerint az architektúrák osztályozása:



Kombinált: különböző operandus típusok megengedettek egy utasításon belül

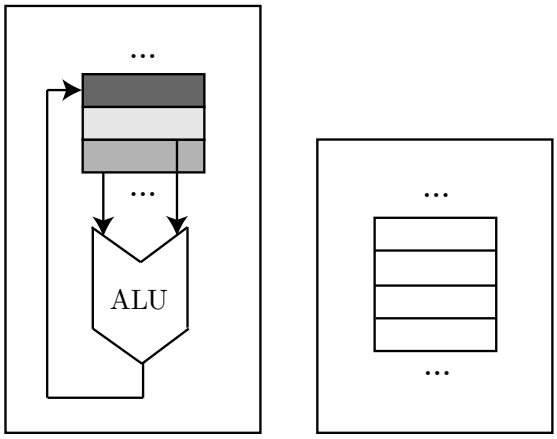
Például: CISC architektúráknál r+m (nem homogén)



Szabályos architektúrák: homogének, ugyanazt az operandus típust biztosítják valamennyi adatmanipuláció esetén

Például: RISC architektúrák

A memória LOAD és STORE utasításokkal címezhető és érhető el



2.6. 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, komplett teljes címet tartalmazza → hosszú

   Relatív címzés: bázis címhez képest számoljuk → rövidebb

   Deklarálni szükséges egy bázis címet és a címszámítási algoritmust.

2. Cím módosítás (opcionális)
   Arra szolgál, hogy a következő operandus címét minél egyszerűbben meghatározzuk. Indexeléshez, autoinkrementáláshoz és autodekrementáláshoz szükséges.
3. Deklarált (tényleges) cím meghatározása
   A címet direkt vagy indirekt, illetve valós vagy virtuális címként tekintjük.

Manapság a CPU címtere nagyon nagy (~ 4 - 64 TB), azért inkább a relatív címzést használják:

bázis (S) + eltolási cím (D)

Gyakorlatilag az letolás mérete határozza meg, hogy mekkorára csökken az adott címtér.
Bázis cím lehet: PC, top of STACK, R_i (index regiszter)

Indexelés:
Hatékony mód az adatblokkok elérésére. Beolvasás nem egyesével, hanem blokkokban történik. Az adatblokk kezdőcíme lesz a bázis cím (S), Ri regiszter fogja tartalmazni az eltolást (R_i = D).

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

Ez egy dimenziód blokk esetén, de több dimenziós blokknál dimenziónként kell megadni az eltolást:

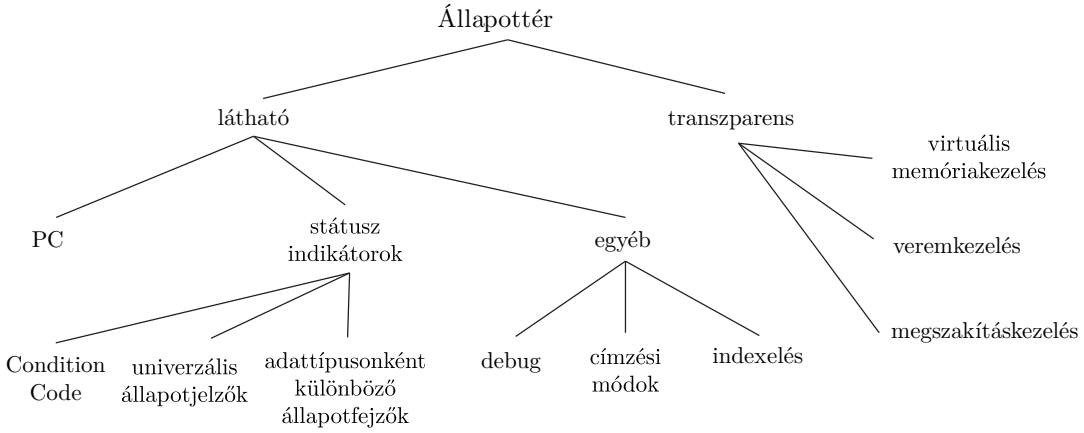
$$Ye = S + [R_{i1}] + [R_{i2}]$$

3. Állapottér

3.1. Definíció

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

3.2. Felosztás



Condition code: IBM-nél használták 2 bites állapotinformációk tárolására

Univerzális állapotjelzők: minden típusra azonos. Például:

- CARRY
- ZERO
- NEG

Adattípusonként különböző állapotjelzők: egyes adattípusoknál használatosak. Például FP-nél:

- denormalizált szám
- alul csordulás
- nullával való osztás

Minden regiszterkészlet típushoz definiálnak külön státusz indikátor készletet.

3.3. Állapotjelzők / státusz indikátorok (flagek)

Olyan kivételes események figyelésére és vezérlése szolgálnak, melyek a program futása közben általánosságban jelenhetnek meg.

4. Állapotműveletek

Az állapotjelzőket speciális utasításokkal lehet manipulálni.

PC állapotműveletei:

- inkrementálás
- dekrementálás
- felülírás (utasításból vett címmel)

Flagek állapotműveletei:

- Save
- Set
- Reset
- Load
- Clear