OBUDA UNIVERSITY YEAR OF JUBILEES
ÓE 15 145 630
TRADITION AND INNOVATION

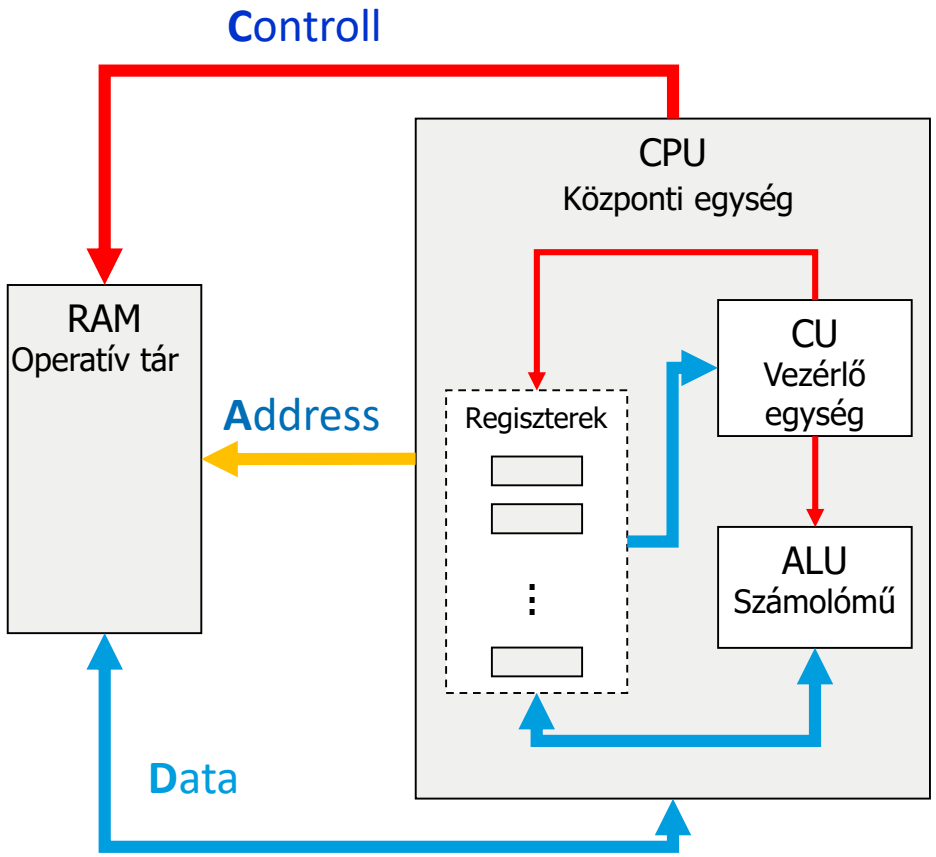
Óbudai Egyetem
Alba Regia Kar
Mérnöki Intézet

Műveletek és utasítások

Dr. Seebauer Márta
egyetemi docens

seebauer.marta@uni-obuda.hu

Egy processzor logikai felépítése



- **Utasításkészlet** – az elemi utasítások összessége, amit egy adott CPU ISA szinten értelmezni képes
- Két processzor egymással **csereszabatos (klónok),** ha ISA szinten kompatibilisek.
- **Felülről kompatibilitás -** egy CPU képes értelmezni és végrehajtani egy előző generációs processzor utasításait.
- **Művelet – utasítás – szoftver rés**

A mikroarchitektúra szintű fizikai architektúra

- műveletvégző egység (ALU – Arithmetical Logical Unit)
- vezérlő egység (CU – Control Unit)
- belső buszrendszer
- általános célú regiszterek
- speciális célú regiszterek
  - programszámláló regiszter (PC – Program Counter)
  - címregiszter (MAR – Memory Address Register)
  - adatregiszter (MDR- Memory Data Register)
  - akkumulátor (AC)
  - utasítás regiszter (IR – Instruction Register)
  - állapotregiszter (PSW Processor State Word)

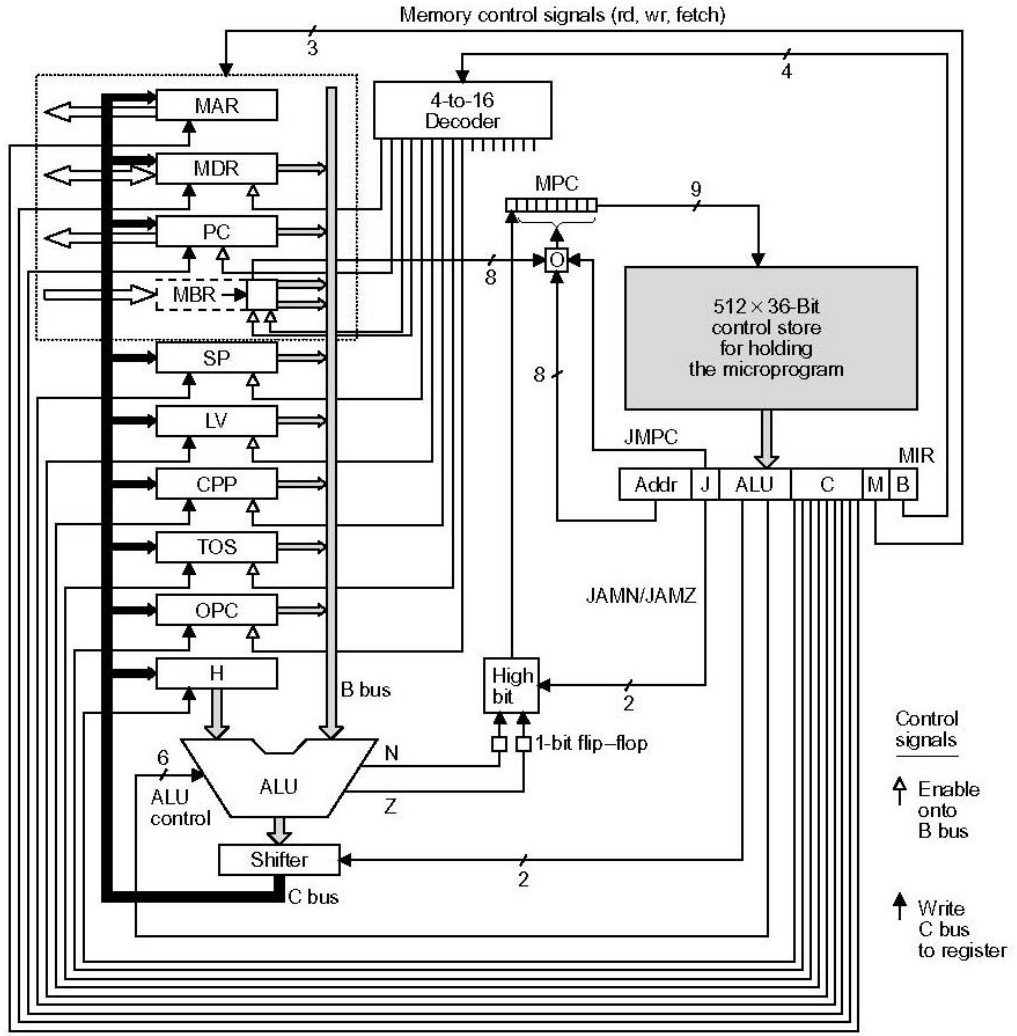


Figure 4-6. The complete block diagram of our example microarchitecture, the Mic-1.

Forrás: Tanenbaum

CISC processzorok

Complex Instruction Set Computer

- Legfőbb tervezési szempont a felülről kompatibilitás megtartása
- A komplex utasítások rövidebb programot eredményeznek, amelyek kevesebb tároló helyet igényelnek az operatív tárban
- A fordítóprogramok egyszerűbbek, mert csökken a magas szintű programnyelvek és a gépi nyelv közötti szemantikus (szotver) rés
- Az utasítások bonyolult műveletsor végrehajtását eredményezik, ami több órajel ciklust is igénybe vehet
- Nagyszámú utasítás és ezen belül különböző címzési módok használata
- Az utasítások változó hosszúságúak, a gyakrabban használt utasítások lehetnek rövidek
- Sokféle, a memóriát közvetlenül használó utasítás
- Mikroprogram vezérelt utasítás-végrehajtás, a mikroprogramok igen bonyolultak
- Mikroprogram tár nagy helyet foglal a processzor lapkáján (50% is lehet)
- Rossz ár/teljesítmény viszony

RISC processzorok

Reduced Instruction Set Computer

- Legfőbb tervezési szempont egy adott célra a leghatékonyabb architektúra létrehozása
- A lehető legtöbb feladatot a fordítóprogramra, és nem a hardverre kell bízni, a fordítóprogram bonyolult, optimalizáló funkciókat tartalmaz
- A lefordított program több utasításból áll, mint a CISC processzorok esetében
- Kevéssé bonyolult utasítások, egyszerű LOAD/STORE, műveletvégző és elágazó utasítások használata
- Kevés utasítás és címzési mód használata
- Az adatelérési idők csökkentésére 3-címes regiszter-regiszter utasítások és 3-utas regisztertár elérés, külön utasítás- és adatelérési utak kialakítása (Harvard architektúra)
- Az utasítások rövidek, átláthatóak, lehetővé teszik az utasítás-pipeline alkalmazását
- A memóriát csak a LOAD/STORE utasítások használják
- A műveleti utasítások végrehajtására egy gépi ciklus elegendő
- Nincs mikroprogram, a vezérlést áramkörök oldják meg (huzalozott utasításértelmezés), az utasítások dekódolása gyors
- A processzor egyszerű és átlátható architektúrája miatt a fejlesztési fázis lerövidül
- A processzor lapkáján több hely marad a regisztertárak, gyorsítótárak (cache) elhelyezésére

Az adatmanipulációs fa

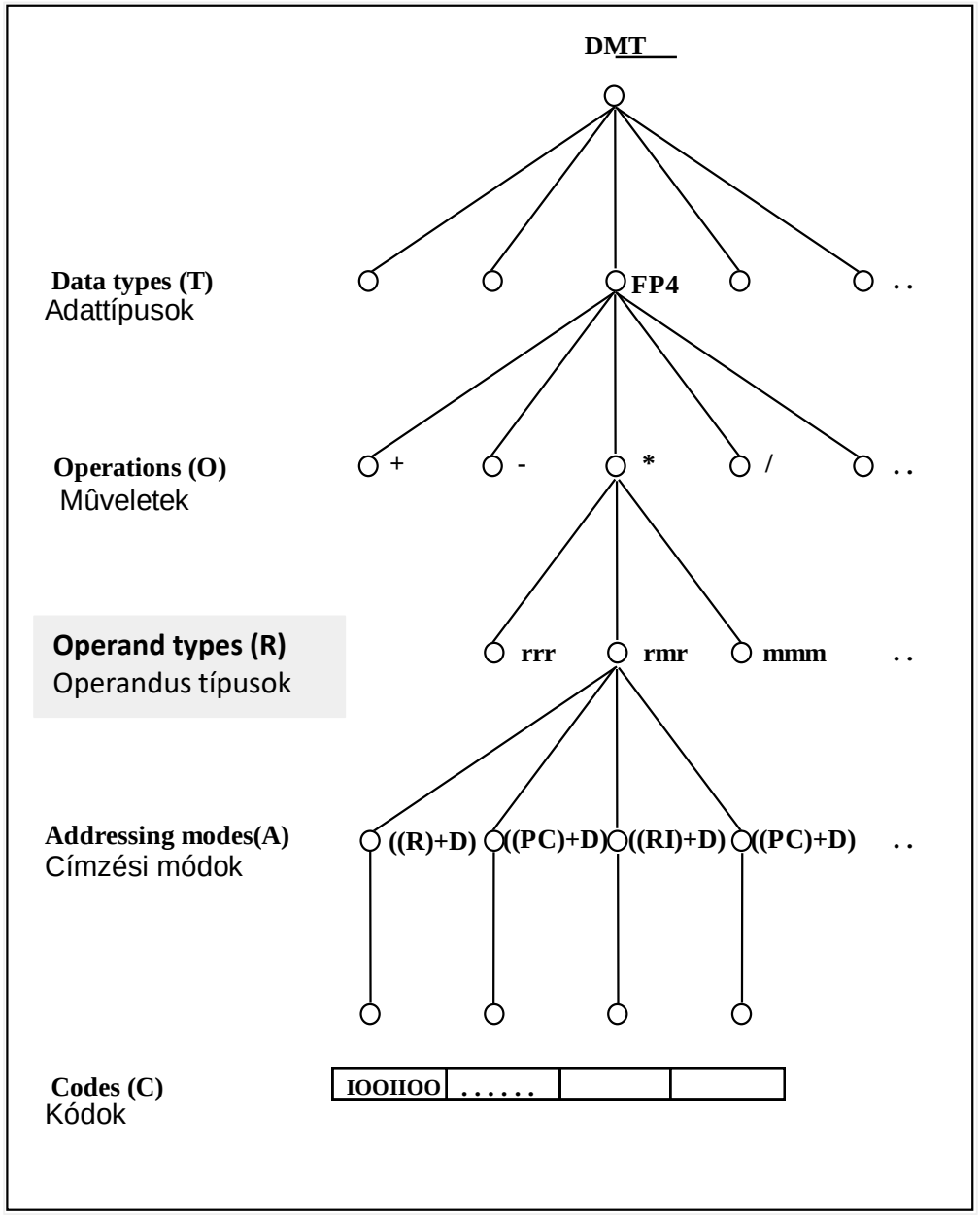
Az adatmanipulációs fa következő szintje minden adat típus esetén megállapítja, hogy

- mely utasítás-típusok megengedettek, ezen belül
- mely operandus-típus választható.

A számítógépet a program működteti. A processzor csak a gépi kódú programokat képes értelmezni.

Utasítás - a számítógép által végrehajtható, alapvető feladatok ellátására szolgáló művelet leírása, kódja.

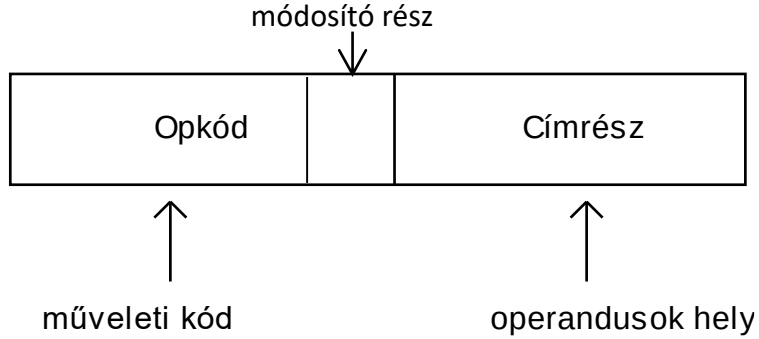
Az **operandus** az utasítás tárgya (forrás operandus) vagy eredménye (cél operandus). Az adott művelet input és output adatának helyét, vagy magát az értékét határozza meg.



Az utasítás szerkezete

A **változó** hosszúságú, gyakran igen hosszú utasítások a CISC processzorokra jellemzőek. A RISC processzorok utasításai **rögzített** hosszúságúak és rövidek.

Törekedni kell az utasítás hosszának csökkentésére. Az utasítások száma egy programban és az utasítások hossza meghatározza, hogy a program mennyi helyet foglal el a memóriában, Az utasítások hossza hatással van a számítógép teljesítményére is. Ha hosszabb az utasítás, mint a processzor szószélessége, akkor a processzor csak több ütemben tudja beolvasni a memóriából.



Az **Opkód mező** hosszát a CPU utasításkészletébe tervezett utasítások számából határozzuk meg. Ha az utasítások száma N, akkor az Opkód mező minimális hossza $\log_2 N$. A mező hosszát célszerű nagyobbra tervezni a felülről kompatibilitás miatt, hogy az utasítások száma bővíthető legyen. Tehát az Opkód mező CISC processzorok esetén nem csökkenthető.

A **Címrész** határozza meg az operandusok elérési módját. Leginkább ez a mező csökkenthető a címzendő operandusok számának csökkentésével vagy különböző címzési módok alkalmazásával. Ez első esetben ugyanazon művelet elvégzéséhez több utasításra lesz szükség, a második esetben a címkiszámítás miatt nő az utasítások végrehajtási ideje.

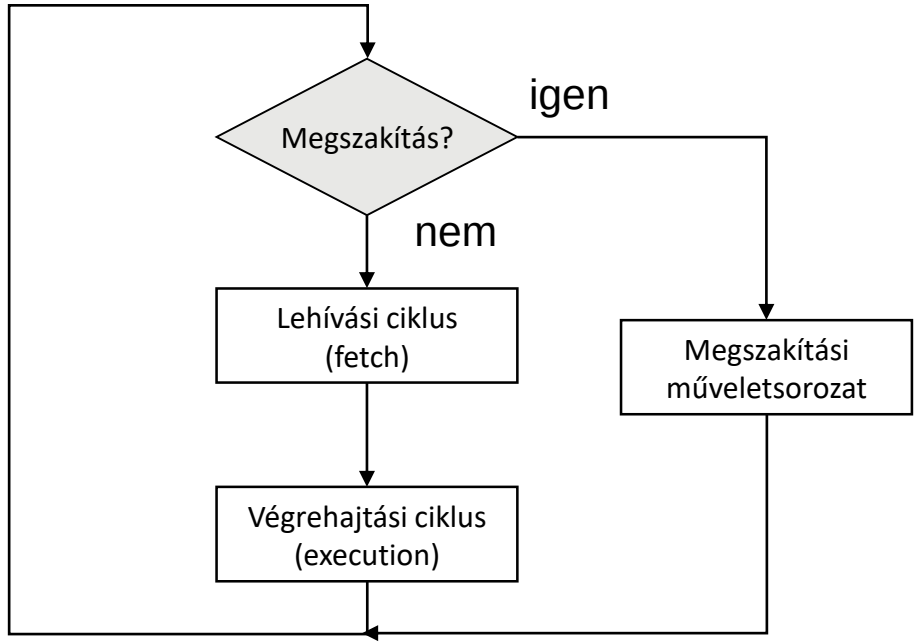
A Módosító rész határozza meg, hogyan kell a Címrészt értelmezni, ezért függ az alkalmazott operandus típusok és címzési módok számától. Vegyes architektúránál nagyobb, szabályos architektúra esetén kisebb.

Megszakítások és kivételkezelés

A programvégrehajtás időszakosan felfüggeszthető

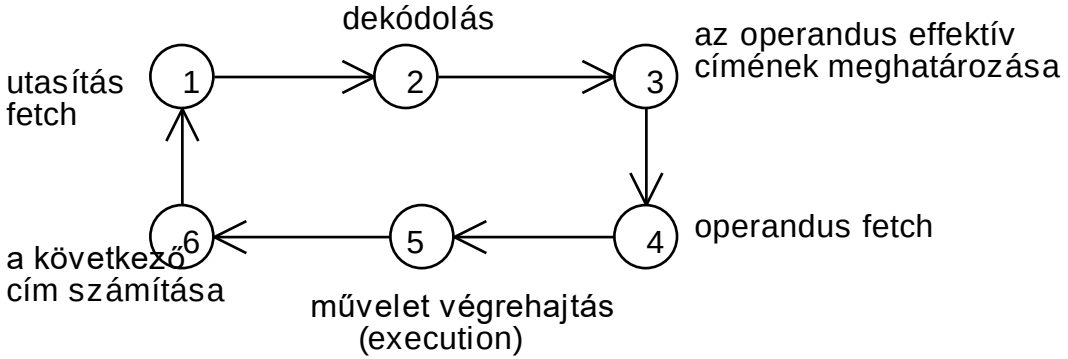
- megszakítások (**interrupt**) esetén, a processzortól független külső esemény idézi elő
  - Input/ output eszköz kiszolgálása
  - DMA megszakításkérés
  - BIOS rutin meghívása
  - Virtuális tár kezelése
- kivétel (**exeption**) esetén, processzoron belül fellépő esemény okozza
  - túlcímzés
  - paritáshiba
  - osztási hiba
  - Túlcsordulás

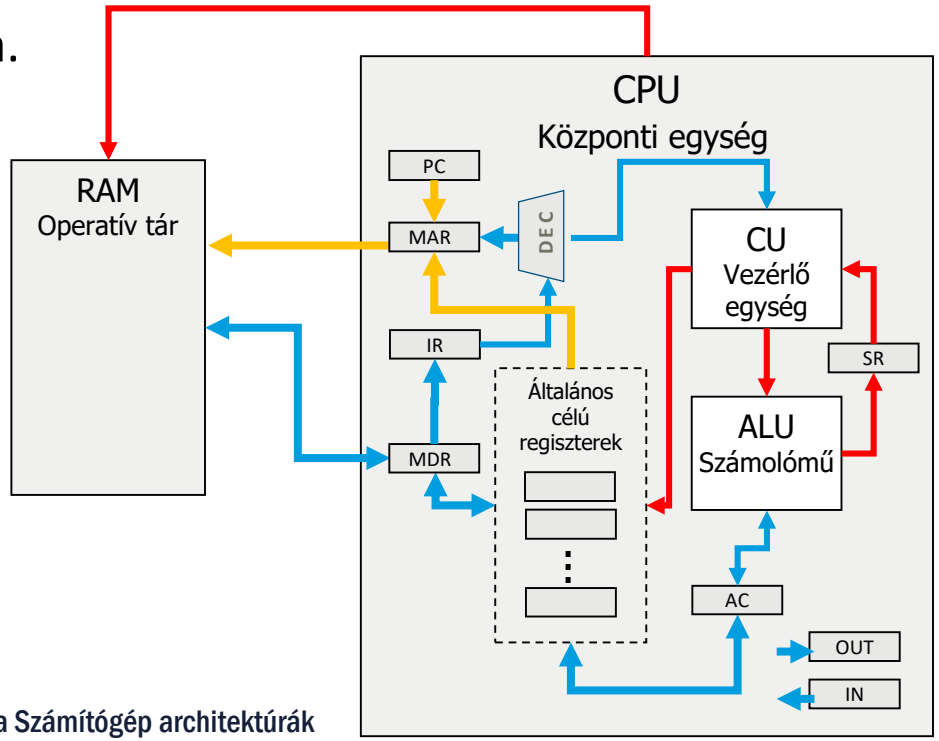
Minden soron következő utasítás lehívása (**fetch**) előtt a processzor megvizsgálja , hogy érkezett-e megszakítás kérés.



A szekvenciális utasításvégrehajtás folyamata

1. A soron következő utasítás beolvasása a memóriából az utasításregiszterbe. Az programszámláló beállítása a következő utasítás címére.
2. A beolvasott utasítás dekódolása, típusának meghatározása.
3. Ha az utasítás memóriabeli szót használ, a szó helyének a megállapítása.
4. Ha szükséges, a szó beolvasása egy regiszterbe.
5. Az utasítás végrehajtása. Az eredmény visszaírása a kijelölt helyre. Amennyiben szükséges, az állapotregiszter megfelelő bitjének beállítása.
6. Vezérlésátadó utasítás esetén a következő cím kiszámítása és betöltése a programszámlálóba.





MAR	Memory Address Register
MDR	Memory Data Register
IR	Instruction Register
PC	Program Counter Register
AC	Accumulator
SR	Status Register
IN	Input Register
OUT	Output Register
DEC	Decoder

Utasításvégrehajtás lépései

Utasításlehívás (fetch)

PC tartalmazza a soron következő utasítás címét

PC → MAR

(MAR) → MDR

MDR → IR

PC+1 → PC

Utasításfeldolgozás

LOAD

IR → DEC

$DEC_{címrész}$ → MAR

(MAR) → MDR

MDR → AC

Műveletvégzés, pl. ADD

IR → DEC

DEC → MAR

(MAR) → MDR

AC+MDR → AC

STORE

IR → DEC

$DEC_{címrész}$ → MAR

AC → MDR

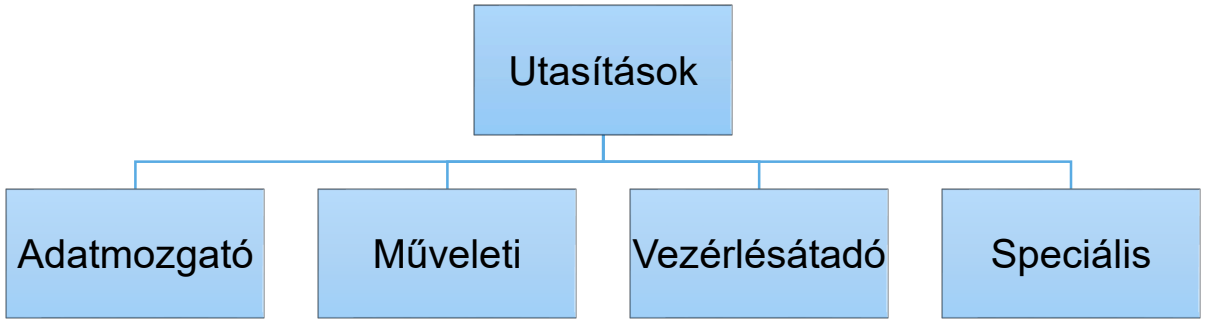
MDR → (MAR)

Feltétlen vezérlésátadás, JMP

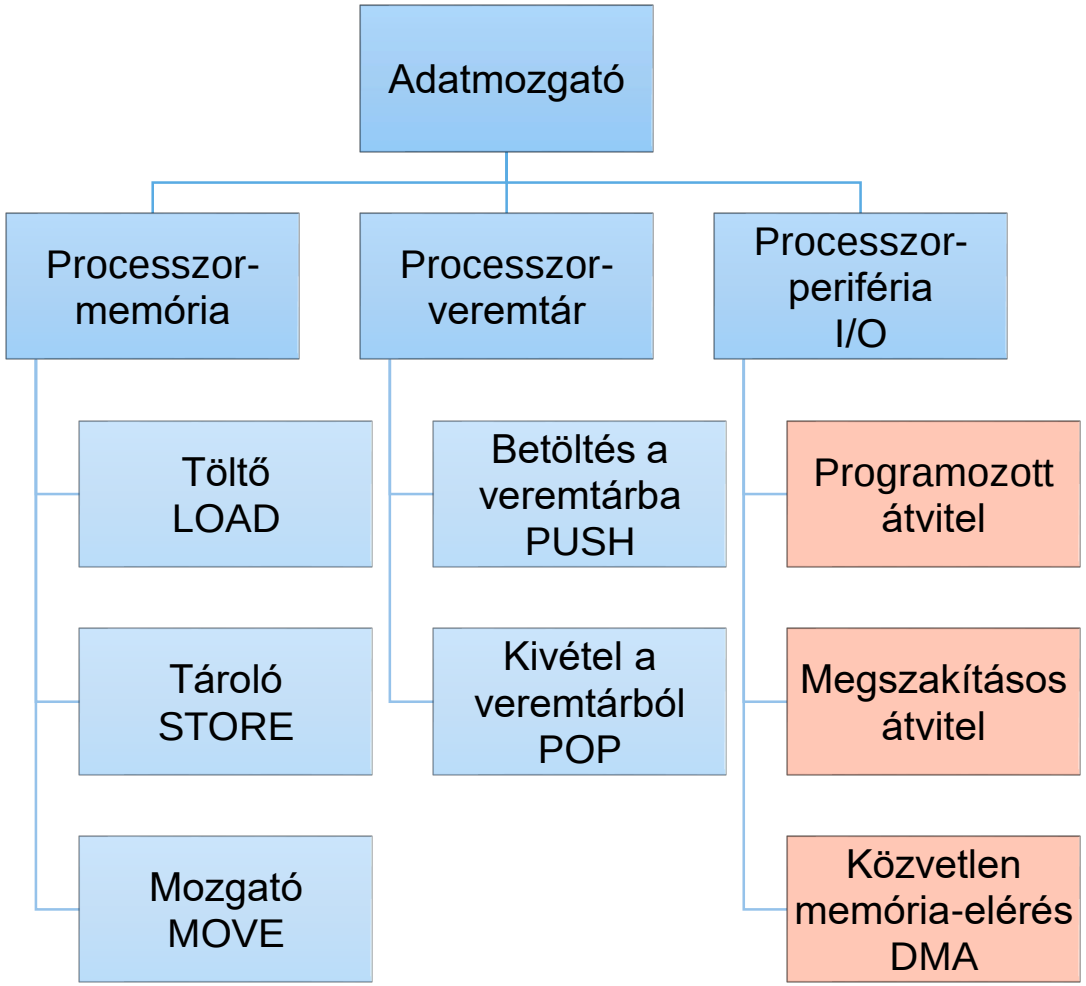
IR → DEC

$DEC_{címrész}$ → PC

Utasítástípusok



Adatmozgató utasítások



I/O műveletek

Ha a műveleti kód beviteli vagy kiviteli műveletet határoz meg, akkor az utasítás lehívását követő memória-hozzáférés elmarad. Az elemi műveletek:

Beviteli utasítás

IR → DEC a művelet dekódolása

DEC → MAR a címrész átvitele

IN → AC a bemeneti regiszter tartalmának átvitele az akkumulátorba

Visszatérés az utasítás-lehívó ciklushoz.

Kiviteli utasítás

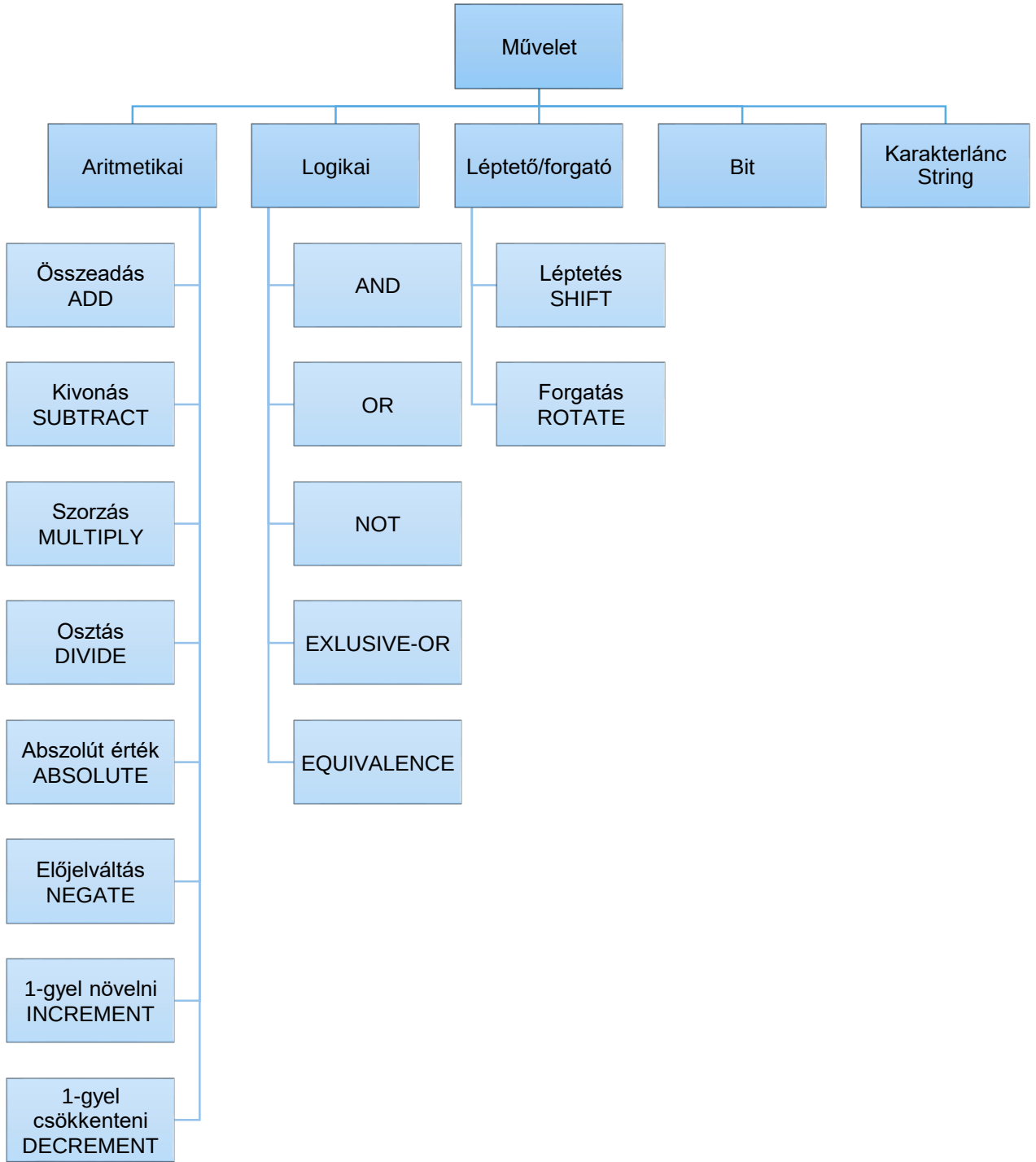
IR → DEC a művelet dekódolása

DEC → MAR a címrész átvitele

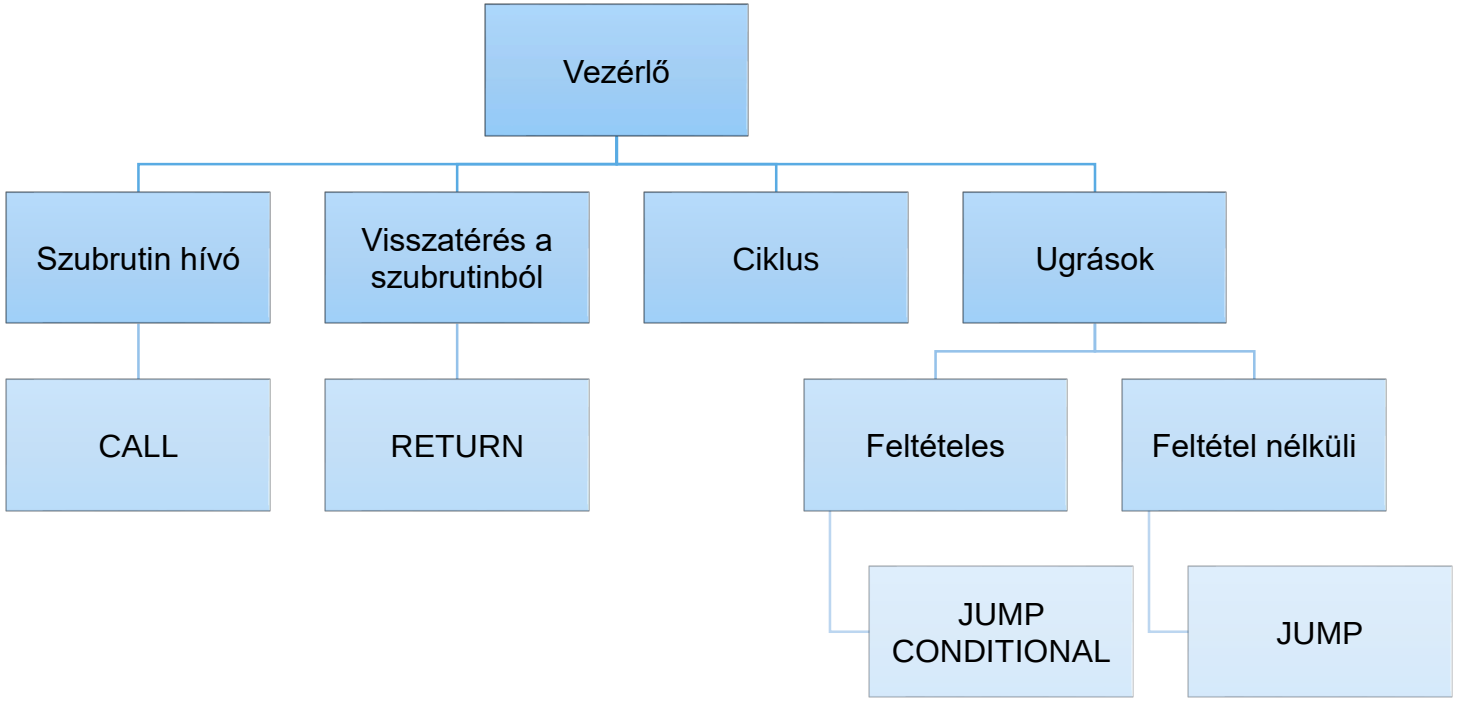
AC → OUT az akkumulátor tartalmának átvitele a kimeneti regiszterbe

Visszatérés az utasítás-lehívó ciklushoz.

Műveletvégző utasítások



Vezérlésátadó utasítások



Feltételes ugrás utasítás

A feltételes ugrások csak akkor hajtódnak végre, ha a megadott feltétel igaz.

Meg kell vizsgálni a megadott feltételt, pl. az A regiszter tartalmát,

- ha az nulla, akkor az utasítás címrésze átkerül a PC regiszterbe, tehát ugrást hajt végre.
- ha az A regiszter tartalma nem nulla, akkor a soron következő utasításra tér át.

Az ugrás lehet egy feltétel bit értékétől függő is. A feltétel biteket az állapotregiszterben tároljuk. Ezen ugrások előtt be kell állítani a feltételbiteket. Ezt beállíthatja az ALU automatikusan az elvégzett művelet eredménye alapján, vagy külön aritmetikai, de szívesebben összehasonlító utasítással állítják be.

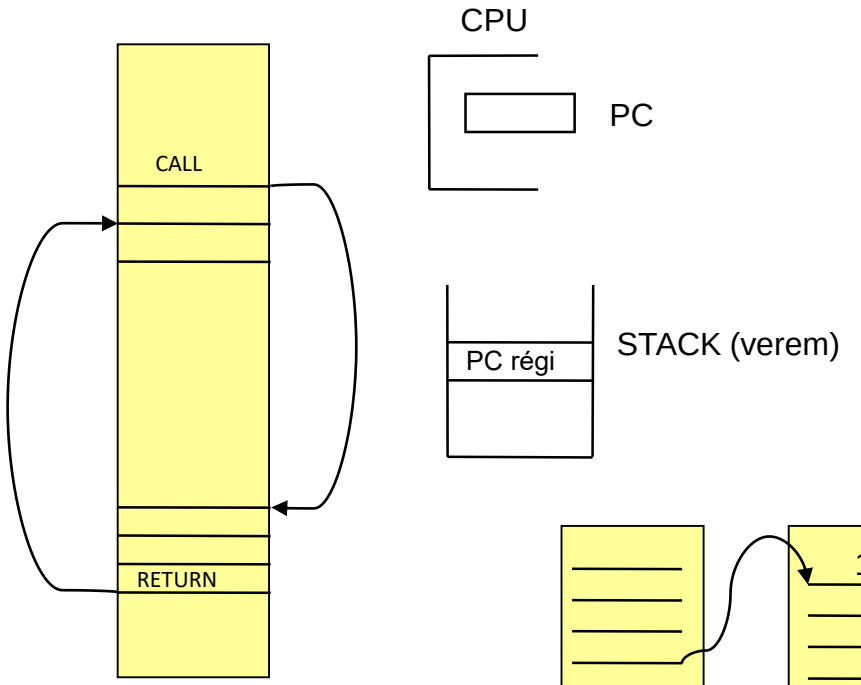
Példa

```
SUB op1,op2
```

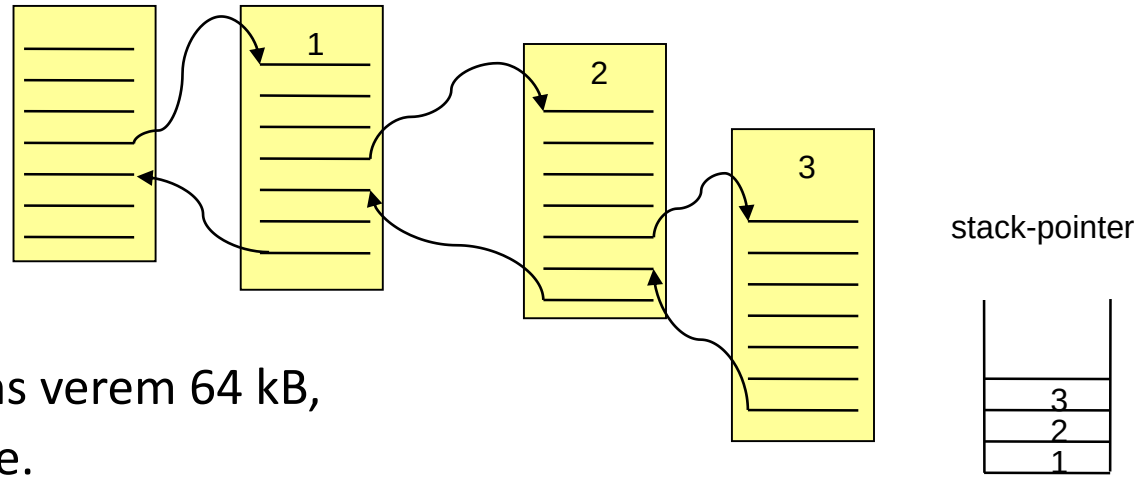
helyett inkább:

```
CMP op1,op2
```

Szubrutinok

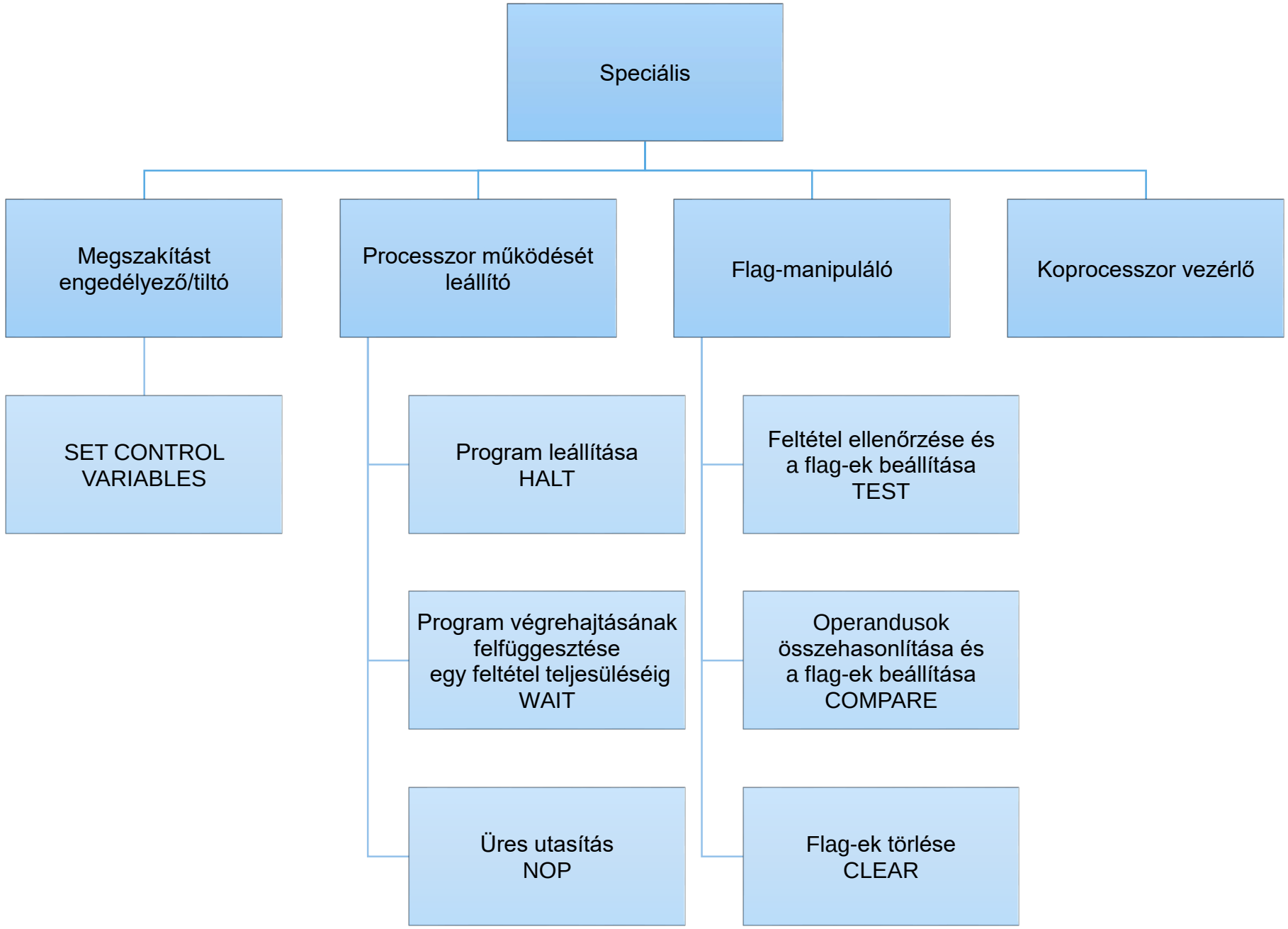


Return hatására a veremből a $PC_{régi}$ bekerül a PC-be és a CALL utáni utasításra adja át a vezérlést



Az IBM PC-kben a stack-szegmens verem 64 kB, efölött túlcsordulás következik be.

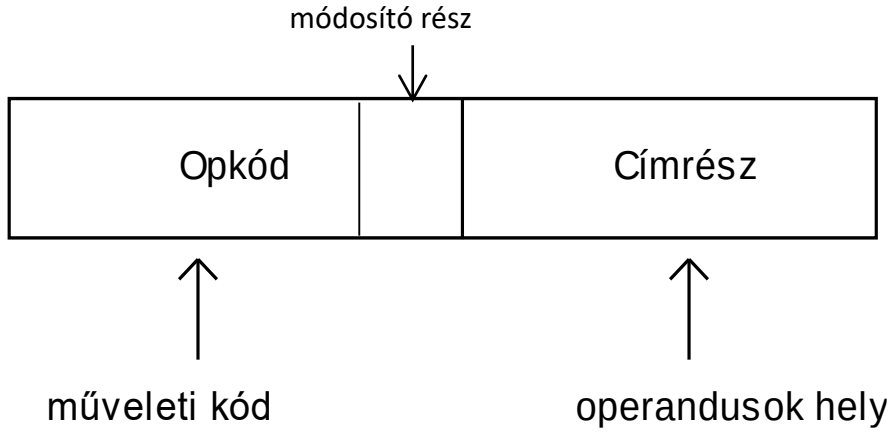
Speciális vezérlő utasítások



Az utasításszerkezet kialakítása

Az utasítások cím részének csökkentése

- négycímes
- háromcímes
- kétcímes
- egycímes
- nulla címes



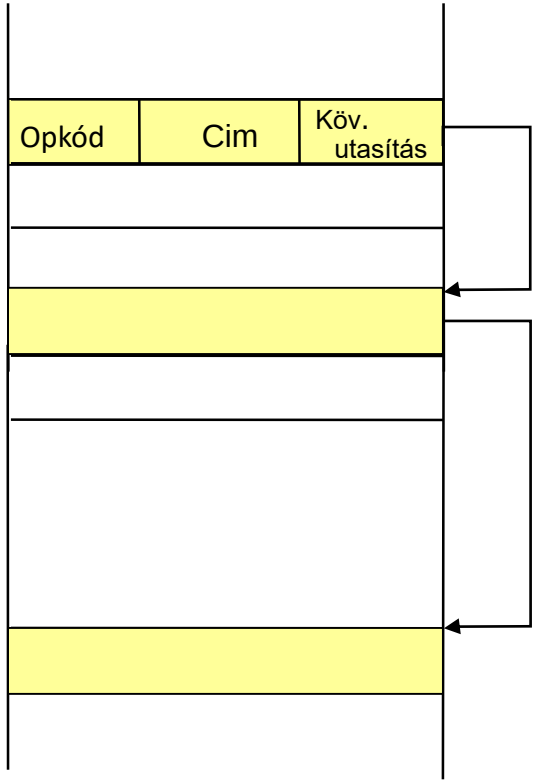
Négycímes utasítás

Opkód	1. operandus címe	2. operandus címe	Eredmény címe	Következő utasítás címe

PC bevezetése

Ez már csak történeti kategória: az első gépek tervezése során a negyedik cím a következő utasítás címét adta meg. Így működött például az EDVAC. Ma már csak firmware-ekben használják ezt a módzsert.

Ritka kivételtől eltekintve az utasításokat lineárisan is lehet tárolni a memóriában, tehát úgy, hogy a következő utasítás a következő címen szerepeljen. A vezérlést úgy lehet megtervezni, hogy az n-ik cím alatt található utasítás végrehajtása után az (n+1)-edik című utasításra térjen át, kivételt képez a vezérlésátadó utasítás, amelyik megadja a következő utasítás címét. A negyedik operandus helyett vezették be a PC-regisztert

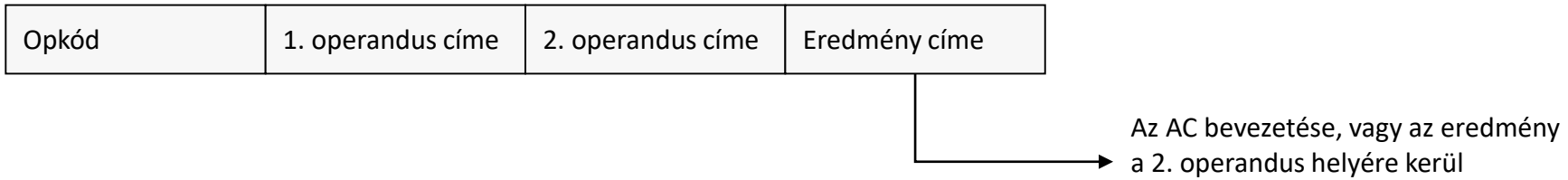


Neumann a Felsőfokú Tanulmányok Intézetének gépét, az IAS-t már egycímesre tervezte.

Hátránya

- hosszú utasítások - memória-pazarló
- adatrögzítési hibák veszélye
- karbantartás, azaz új utasítások beszúrásának nehézkessége

Háromcímes utasítás



Ez egy explicit deklarálása annak, hova kerüljön az eredmény.

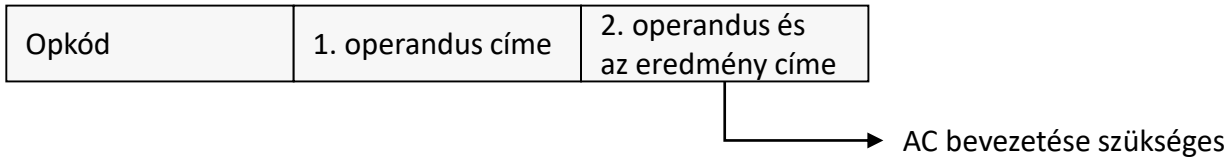
A háromcímes utasítástípussal szembeni fő Neumann-i érv, hogy szinte soha nincs szükség arra, hogy két, a memóriában elhelyezkedő számot összeadjunk - vagy valamilyen más műveletet végezzünk közöttük, majd az eredményt visszaírjuk a memóriába.

Általában az a helyzet, hogy a gép által kiszámított eredmény az aritmetikai egységnél (az akkumulátorban) marad, mert további műveletet kell rajta végezni. Még ennek a műveletnek az eredménye is gyakran az aritmetikai egységben marad további műveletek végrehajtása céljából. Ezt láncolt műveleteknek nevezzük.

További hátránya: hosszú az utasítás, sok memóriát igényel

Előnye: párhuzamosítási lehetőség: RISC

Kétcímes utasítás



Az célrekesz előző tartalma elvész.

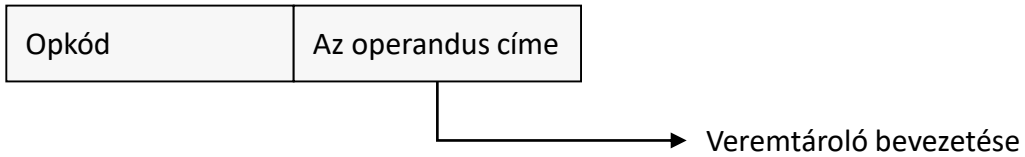
Az eredmény helyének implicit deklarálása.

Ilyen típusok: IBM 360/370 (rendelkeznek egy és háromcímű utasításokkal is)

Előnye: rövidebb

Hátránya: azt az operandust, ahova az eredmény kerül felülírja. Szükség esetén gondoskodni kell a kimentéséről

Egycímes utasítás



A művelet mindig a megcímzett tároló hely és az akkumulátor között kerül végrehajtásra

Egy aritmetikai művelet (pl. összeadás) elvégzéséhez három utasításra van szükség.

1. egyik operandust át kell vinni az egyik címről az aritmetika egységbe (akkumulátorba);
2. el kell hozni a másik operandust egy másik címrõl és hozzáadni ahhoz, amelyik már az aritmetikai egységben (akkumulátorban) volt;
3. az eredményt egy harmadik címen el kellett tárolni.

A korai gépek közül ilyen volt az EDSAC, Ferranti Mark 1, Whirlwind I, Univac I, IBM 701, IBM 704. Ilyen még a hatvanas évekből pl. IBM 1130, 1800, System/7, Siemens 300.

Előnye: rövid, egyszerű utasítás - memória-takarékos, RISC gépek

Hátránya: több utasítással valósítható csak meg egy olyan művelet, amely kétcímesnél eggyel is megoldható

Nulla címes utasítás

A műveleti kód után nem áll operatív tár cím, ezért veremtároló használatára van szükség.

Egy operandust implicit operandusnak nevezünk, ha az utasítás maga egyértelműen meghatározza . A programozónak az utasítással kapcsolatosan nem kell specifikálnia semmit.

Előnye: rövid utasítások, hiszen nincsen címrész

Hátránya: igen speciálisak, tehát felduzzaszthatják az utasítás-készletet

Például

NOP (no operation)

CLEARD (a D flag törlése: az implicit operandus a D flag)

PUSH, POP CALL, RET (a veremtároló tartalma ezen utasítások implicit operandusa)

Példa *X=AxB+CxC*

Egycímes gép

Művelet	Comment
LOAD A	A betöltése AC-be
MULTIPLY B	AC←ACxB
STORE T	AC tartalmának eltárolása T címre
LOAD C	C betöltése AC-be
MULTIPLY C	AC←ACxC
ADD T	AC←AC+T
STORE X	AC tartalmának eltárolása T címre

Háromcímes gép

Művelet	Comment
MULTIPLY T,A,B	T←AxB
MULTIPLY X,C,C	X←CxC
ADD X,X,T	X←X+T

Kétcímes gép

Művelet	Comment
MOVE T,A	T←A
MULTIPLY T,B	T←TxB
MOVE X,C	X←C
MULTIPLY X, C	X←XxC
ADD X,T	X←X+T

Nullacímes gép

Művelet	Comment
PUSH A	A betöltése a stack-be
PUSH B	B betöltése a stack-be
MULTIPLY	A és B kivétele a stack-ből és AxB betöltése a stack-be
PUSH C	C betöltése a stack-be
PUSH C	C ismételt betöltése a stack-be
MULTIPLY	C és C kivétele a stack-ből és CxC betöltése a stack-be
ADD	CxC és AxB kivétele a stackből és összegük betöltése a stack-be
POP X	Stack tetejének eltárolása X címre

Processzorok utasításszerkezete

A jelenlegi **CISC** processzorok többsége egy-, illetve a két címes utasítástípust használ.

Ortogonális utasításkészlet – A vegyes architektúrákra jellemző, minden utasítás típus esetében megenged mindenféle címzési módot.

A **RISC** architektúrák - mivel szabályos regiszter-architektúrák - ezért előnyben részesítik a három címes formátumot, hogy kiküszöböljék a rendeltetési címre vonatkozó korlátozásokat.

Operandus típusok

A továbbiakban meg kell határoznunk az **operandus** típusát. Az operandusokat akkor tartjuk ugyanazon típusúnak, ha ugyanazon adattérből vettük őket, vagy ugyanabba az adattérbe tesszük. Természetesen az ideális megoldás a felhasználó szempontjából az lenne, ha az operandusként tetszőleges adatteret megadhatna.

Tehát a lehetséges operandusok halmaza

- akkumulátor operandus
- memória operandus
- regiszter operandus
- verem-tároló operandus
- immediate operandus.

Az **akkumulátor** a processzorban az ALU mellett helyezkedik el, gyors elérésű, de csak egy van belőle, ezért nem kell címezni, de ami szűk keresztmetszetet okoz

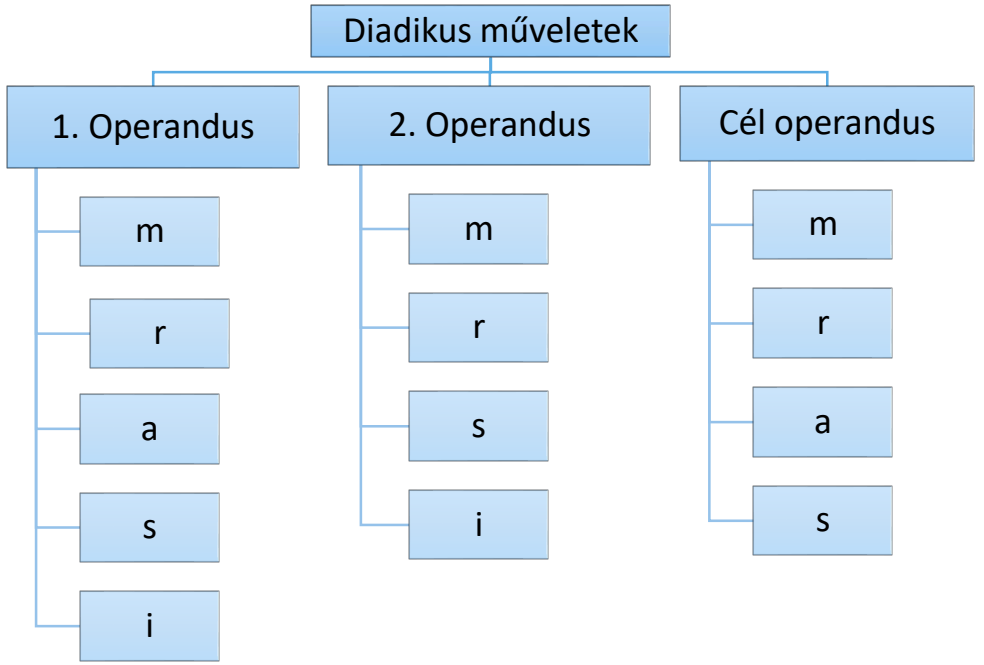
A **memória** operandus helye az operatív tár, tág lehetőségek, nagy kapacitású, de nagyon lassú elérésű, és a nagy kapacitása miatt hosszú címmel rendelkezik

Az általános célú **regiszterkészlet** a processzor lapkáján, gyors elérésű, rövid címmel rendelkezik, de korlátozott a regiszterek száma, ezért drága

A **verem-tároló** lehet a processzor lapkáján, akkor gyors elérésű, csak a tetejét látjuk, szűk keresztmetszet

Literálok vagy **immediate**, gyors elérésű, hiszen az utasításban helyezkedik el, tulajdonképpen egy konstans érték, aminek az értéke csak a programmódosítással változtatható

ÓE 15 145 630 Utasítástípusok a tervezési térben



A 3-címes diadikus műveletek lehetséges kombinációit kifejezhetjük a következőképpen is:

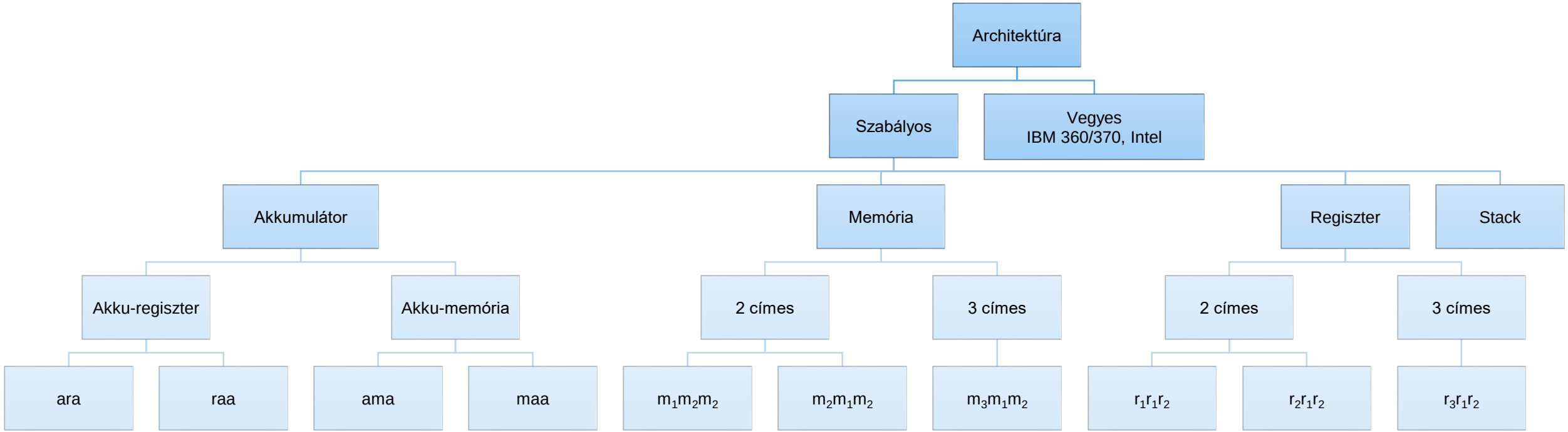
{ a, m, r, s, i} x {m, r, s, i} x { a, m, r, s}

Tehát az absztrakt architektúrának kell specifikálnia

- minden művelet vonatkozásában, mely utasítás típusok lehetségesek és
- minden operandus vonatkozásában, mely operandus típusok lehetségesek.

Felhasználói szemmel nézve az lenne az ideális, ha valamennyi lehetséges változat biztosított lenne. Viszont ez az általánosítás sem a helyfoglalás (túl nagy kódteret igényelne), sem pedig a hatékonyság szempontjából nem előnyös.

Architektúrák osztályozása az operandus típusok szerint



Szabályos architektúrák

Van néhány architekturális stílus, amely figyelmet érdemel, ezeket **szabályos architektúráknak** nevezzük. Bizonyos architektúrák ugyanazt az operandus típust biztosítják valamennyi adatmanipuláció esetén (rrr , mmm, ama vagy sss), kivéve természetesen a LOAD és a STORE utasítást.

A szabályos architektúrákat széles körben használják. A támogatott utasítás-típusnak megfelelően külön nevük van, mint például a Accumulátor, Regiszter, vagy más néven LOAD/STORE, Memória és a Stack architektúrák.

architecture type	**instruction type of the data-manipulation instructions**
accumulator-based architecture	aam, ama
register-architecture (load-store-architecture)	r r r → 2-address format *(RT/ PC)*; 3-address format *(SPARC, 88000)*
memory-architecture	mmm → 2-address format *(TMS9900)*; 3-address format
stack-architecture	s s s *(B5000, HP 3000)*

Akkumulátor alapú architektúrák

Az akkumulátor alapú architektúrák

aam, ama, aar, ara

utasítás-formátumokat kínálnak. Nagyon gyorsak, de az akkumulátor szűk keresztmetszetet alkot.

Jelenleg már nem aktuálisak, mivel nem hatékony a szükséges adatáramlás.

Regiszter vagy LOAD/STORE architektúrák

Valamennyi adat-manipulácó esetén

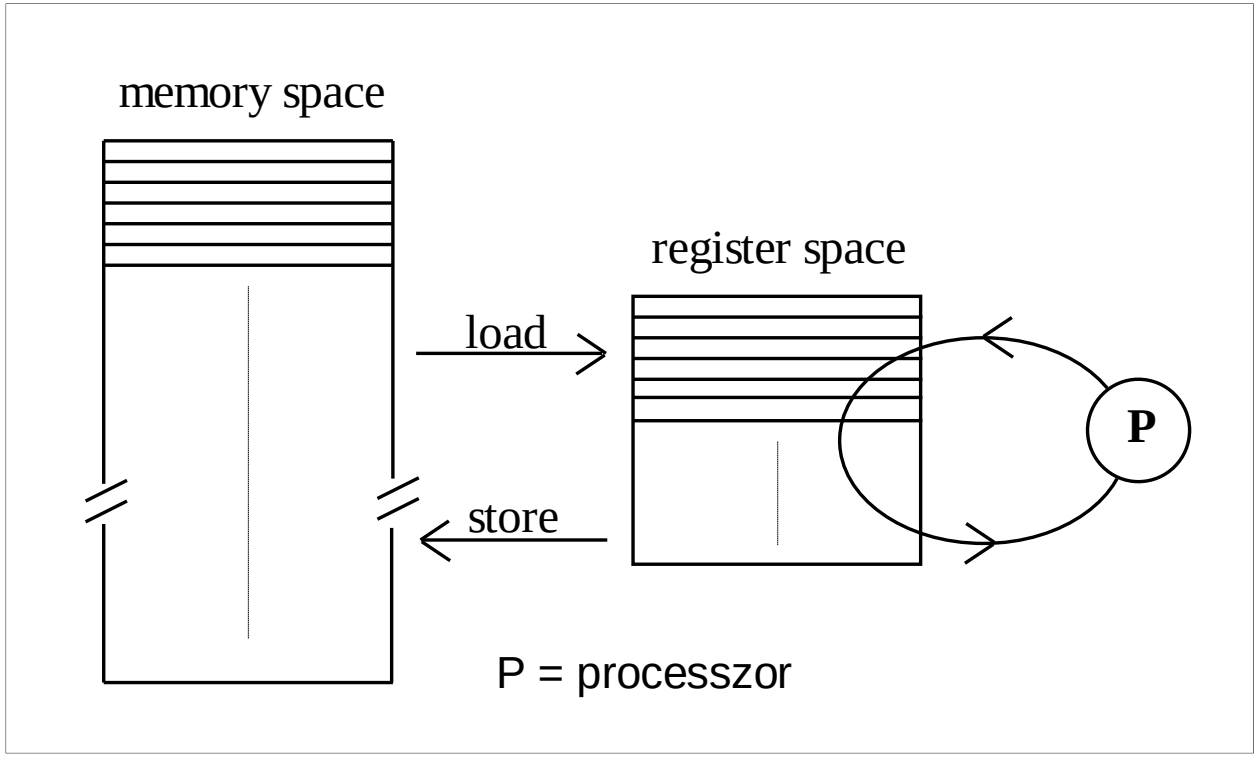
rrr

utasítás-formátumot biztosítanak.

Minden művelet a regisztertérben valósul meg, ami gyors műveletvégzést tesz lehetővé. Ennek az architektúrának a következménye a nagy teljesítmény. A műveletek előkészítéséhez szükséges LOAD és a STORE műveletek ugyanakkor

rm

típusúak.



A RISC architektúrák, amelyeket jelenleg a legmodernebb architektúra osztálynak tartanak, leginkább ezen az elven működnek.

A legtöbb RISC architektúra 3-címes formátumú, hogy megőrizze annak a szabadságát, hogy a regiszter-téren belül bárhova tárolhassa az eredményt (M88000, SPARC). Ez az architektúra típus elég nagy regiszter-térrel rendelkezik (minimum 32 regiszter).

Memória alapú architektúrák

A memória architektúrák, melyek utasítás-formátuma

mmm

széles elérési teret biztosítanak, két kritikus hátránnyal rendelkeznek:

- alacsony a teljesítményük, mivel a memóriához való hozzáférés lassú
- hatékonyságuk azért is alacsony, mivel túl hosszú utasítások használatára kényszerítenek és ennek következtében túlságosan memória-igényesek.

Stack architektúrák

A stack architektúrák utasításformátuma

sss

A stack egy sor előnyt biztosít a végrehajtás során.

- a stack operandust használó utasítás rövid, mivel nem igényel címzést, a kifejezések értéke könnyen meghatározható stack használatával
- procedúra hívás esetén pedig a stack alkalmazható paraméter átadásra és eredmény tárolásra vagy a stack használható a program status megőrzésére a megszakítás idejére, stb.

Az előnyeik ellenére a stack architektúrák sohasem érték el a fő irányvonal szintjét. Ez valószínűleg a stack több korlátja miatt történt így

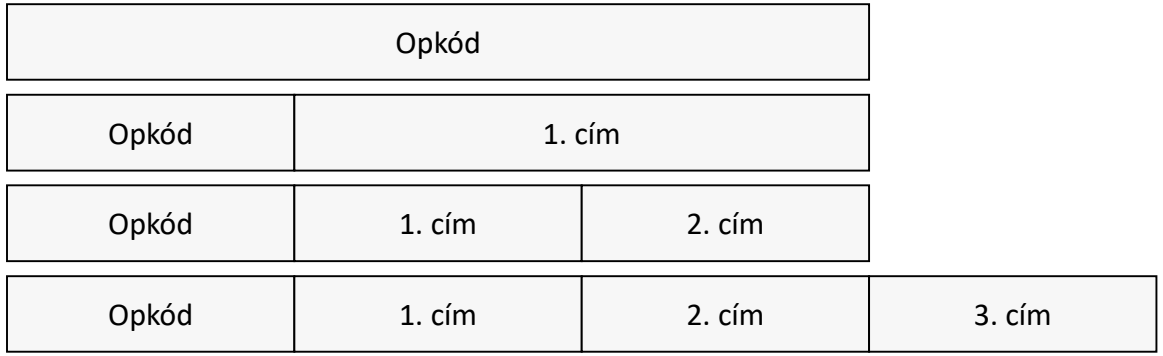
- az adatokat csak szekvenciálisan lehet fetch-elni a stack-ből, és közvetlen elérésükhöz nem áll rendelkezésre eszköz. Tehát, a processzornak igen kötött lehetősége van az operandusok elérésére, mivel a processzor kizárólag a stack tetejét látja.

Címzési módszerek

Egy processzor az utasításkészlete állhat fix vagy változó hosszúságú utasításokból.

Az utasítás bitjeinek nagyobb részét nem a műveleti kód, hanem az operandusok címe foglalja el. Pl. A műveleti kód számára 256 utasításból álló utasításkészlet esetén 8 bit elegendő, de ha a memóriacímek 32 bitesek, egy három címes utasításhoz elvileg 104 bitre lenne szükség. A címrész különböző címzési módszerek alkalmazásával csökkenthető.

- a többször használt adatot regiszterbe töltjük, így egyetlen LOAD utasítással a memóriából áttölthető. A regiszterek használata kettős előnnyel jár: rövidebben címezhetőek és lényegesen gyorsabbak
- a címrész csökkentése
  - 2, 1 vagy 0 címes utasítások használata
  - relatív címek alkalmazása. Előny, mert így a kód hordozhatóvá válik.



Közvetlen címzés

Az operandust közvetlenül az utasításban adjuk meg.

Hátránya

- csak konstans érték adható meg
- korlátozott a felhasználható bitek száma

Pl.

MOV	R1	4

Direkt címzés

A memóriabeli operandus megadása a teljes címével.

Az utasítás mindenkori végrehajtása ugyanazt a memóriamezőt érinti, tehát a változó értéke módosulhat, de a helye nem.

Jellemzően olyan globális változók esetében használják, amelyeknek a címe fordításkor ismert.

Regisztercímzés

Amíg a direkt címzés a memóriahelyet címez, addig a regisztercímzés a regiszter címére mutat.

A fordítóprogramok elemzik a forráskódot, és a gyakran használt változókat regiszterekben helyezik el.

Gyakran használt címzési mód.

A LOAD/STORE architektúrák műveletvégző utasításokban kizárólag ezt a címzési módot alkalmazzák.

Regiszter-indirekt címzés

A specifikált operandus címét egy regiszter tartalmazza, amelynek a címét definiáljuk az utasításban. Ezt a címet mutatónak (pointernek) hívjuk.

Előnye

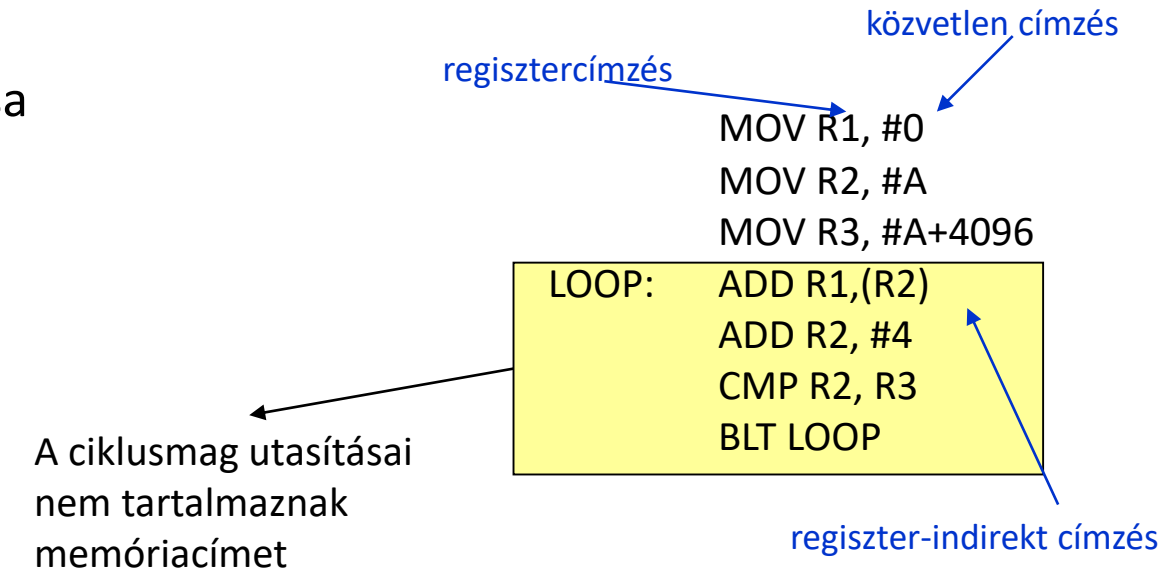
- nem kell a memóriacímet az utasításban tárolni
- a cím értéke a program végrehajtása során változhat

Példa

A egydimenziós 1024 elemű tömb elemei összegének kiszámítása. Az elemek 4 bájtos egész számok

#0 – konstans 0

#A – Az A tömb memóriacíme

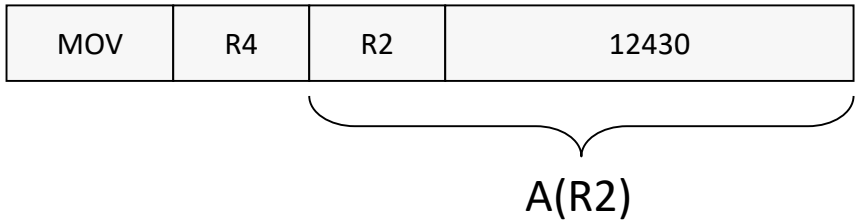


Indexcímzés

A hivatkozott memória címét egy regiszter tartalmának és egy konstans értéknek az összege határozza meg.

Ezt eltolásnak (offset) nevezzük, amely az utasításban memóriacím hosszúságú mezőt foglal.

MOV R4, A(R2)



Példa

Adott *A* és *B* 1024 elemű egydimenziós tömbök. Keressük, létezik-e legalább egy olyan A_i/B_i pár, amelynek egyike sem 0. Ehhez kiszámítjuk páronként A AND B logikai szorzatot, amelyeknek képezzük OR művelettel az akkumulált összegét.

```
        MOV R1, #0
        MOV R2, #0
        MOV R3, #4096
LOOP:   MOV R4, A(R2)
        AND R4, B(R2)
        OR R1, R4
        ADD R2, #4
        CMP R2, R3
        BLT LOOP
```

Bázisindex címzési mód

A hivatkozott memóriacím két regiszter tartalmának összegéből és egy esetleges eltolásból összegéből áll. Az egyik regiszter a bázis, a másik az index.

Ezt a címzési módot alkalmazó gépek a gyakorlatban 8 és 16 bites eltolást biztosítanak.

Példa

```
        MOV R1, #0
        MOV R2, #0
        MOV R3, #4096
        MOV R5, #A
        MOV R6, #B
LOOP:   MOV R4, (R2+R5)
        AND R4, (R2+R6)
        OR R1, R4
        ADD R2, #4
        CMP R2, R3
        BLT LOOP
```

Veremcímzési mód

Fordított lengyel jelölés

A változók sorrendje az infix és a prefix jelölésben megegyezik, a műveleti jelek sorrendje azonban nem azonos.

A fordított lengyel jelölésben a műveletek sorrendje megegyezik a kifejezés kiértékelésekor azok végrehajtási sorrendjével.

Infix	Reverse Polish notation
A + B × C	A B C × +
A × B + C	A B × C +
A × B + C × D	A B × C D × +
(A + B) / (C – D)	A B + C D – /
A × B / C	A B × C /
((A + B) × C + D)/(E + F + G)	A B + C × D + E F + G + /

Step	Remaining string	Instruction	Stack
1	8 2 5 × + 1 3 2 × + 4 – /	BIPUSH 8	8
2	2 5 × + 1 3 2 × + 4 – /	BIPUSH 2	8, 2
3	5 × + 1 3 2 × + 4 – /	BIPUSH 5	8, 2, 5
4	× + 1 3 2 × + 4 – /	IMUL	8, 10
5	+ 1 3 2 × + 4 – /	IADD	18
6	1 3 2 × + 4 – /	BIPUSH 1	18, 1
7	3 2 × + 4 – /	BIPUSH 3	18, 1, 3
8	2 × + 4 – /	BIPUSH 2	18, 1, 3, 2
9	× + 4 – /	IMUL	18, 1, 6
10	+ 4 – /	IADD	18, 7
11	4 – /	BIPUSH 4	18, 7, 4
12	– /	ISUB	18, 3
13	/	IDIV	6

Forrás: Tanenbaum

Címzési módok elágazó utasításokban

Címekre nemcsak az adatokon végzett műveleteknél van szükség. Az elágazó vagy eljáráshívó utasítások szintén igényelnek olyan címzési módot, amellyel az ugrási cím megadható.

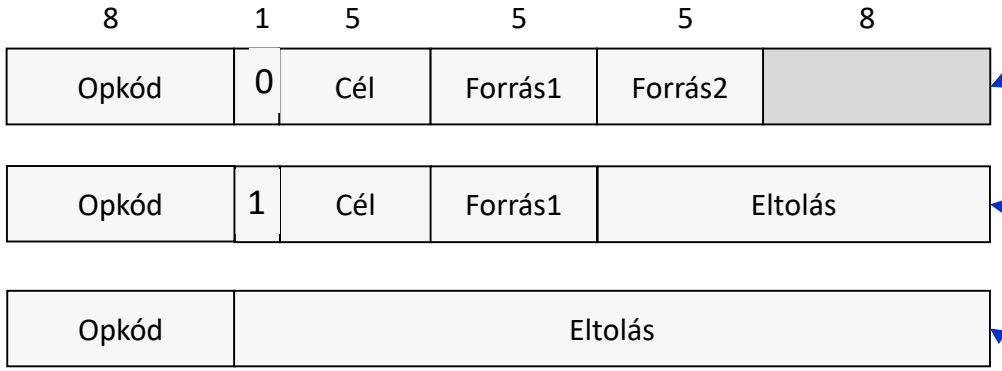
- direkt címzés – a célcímet teljes egészében közvetlenül az utasításban adjuk meg
- regiszter-indirekt címzés – az ugrási címet kiszámítjuk, és egy regiszterben tároljuk, amely címére vonatkozva hajtjuk végre az ugrást. Hatékony módszer, de hiba esetén nehezen deríthető fel.
- indexmód – a célcím egy regiszter tartalmához viszonyított eltolási érték
- PC-relatív címzés – az eltolási érték hozzáadódik az utasításszámláló aktuális értékéhez. Ez egyszerű index címzés, ahol a

Műveleti kód és a címzési mód ortogonalitása

Ideális eset, amely a fordítóprogram számára biztosítja a jó minőségű kód előállítását

- az utasítások és a címzési módok szabályos szerkezete
- minimális számú utasításformátum
- minden műveleti kód megenged minden értelmes címzési módot
- minden regiszter elérhető minden regiszter módban (beleértve ez FP veremkeret mutatót, SP veremmutatót, PC utasításszámlálót)

Ideális három címes fix hosszúságú utasításkészlet



aritmetikai-logikai utasítások

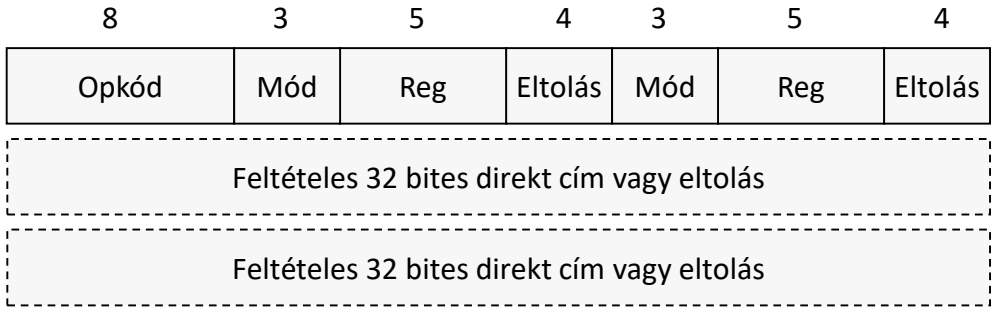
közvetlen címzés
LOAD/STORE utasítások index módban

elágazás, eljáráshívás
LOAD/STORE utasítások közvetlen címzéssel

Műveleti kód és a címzési mód ortogonalitása

Kétcímes gép változó utasítás hosszúsággal

- mindkét operandus lehet memóriaszó
- műveletek
  - regiszter-memóriaszó
  - memóriaszó-regiszter
  - regiszter-regiszter
  - memóriaszó-memóriaszó
- rendkívül költséges a memória hozzáférés miatt
- egyszerű és átlátható fordítás



Címzési módok (Mód=3 bit)

- közvetlen
- direkt
- regiszter
- regiszter-indirekt
- index
- veremcímzés
- 2 további

Utasításkészlet

Egy processzor utasításkészlete – azoknak az elemi utasításoknak az összessége, amelynek végrehajtására a legalsó, hardver szinten a processzor alkalmas.

A legalsó szint azt jelenti, hogy az utasítás végrehajtása vagy huzalozott módon, áramköri szinten valósul meg, vagy az elemi lépéseket vezérlő mikroprogram segítségével.

Az utasításkészlet jellemzése

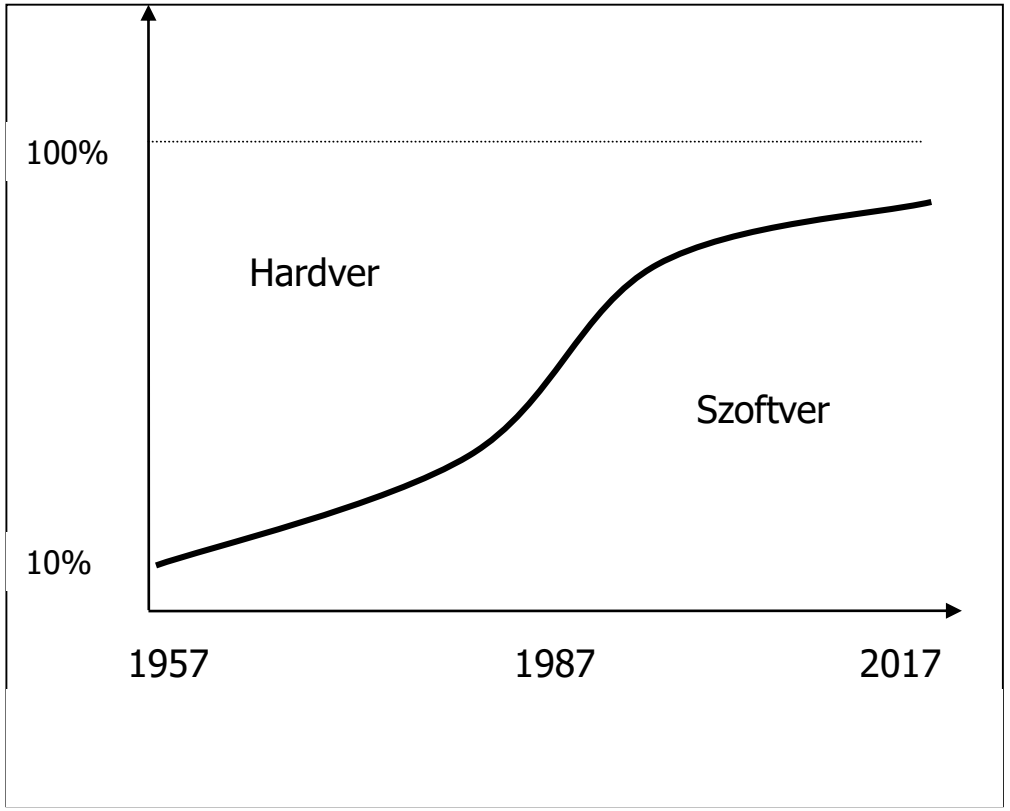
- a rendelkezésre álló elemi utasítások száma és tartalma
- az utasításokkal kezelhető feltételek száma
- az utasítások jellemzőinek következetes használati lehetősége, a kivételek minimális száma
- az utasításkészlet kialakítása mennyire nyújt hatékony támogatást
  - a programíráshoz
  - a programok fordításához
  - az ellenőrizhetőséghez

Számítógépes műveletek végrehajtásának HW/SW aránya

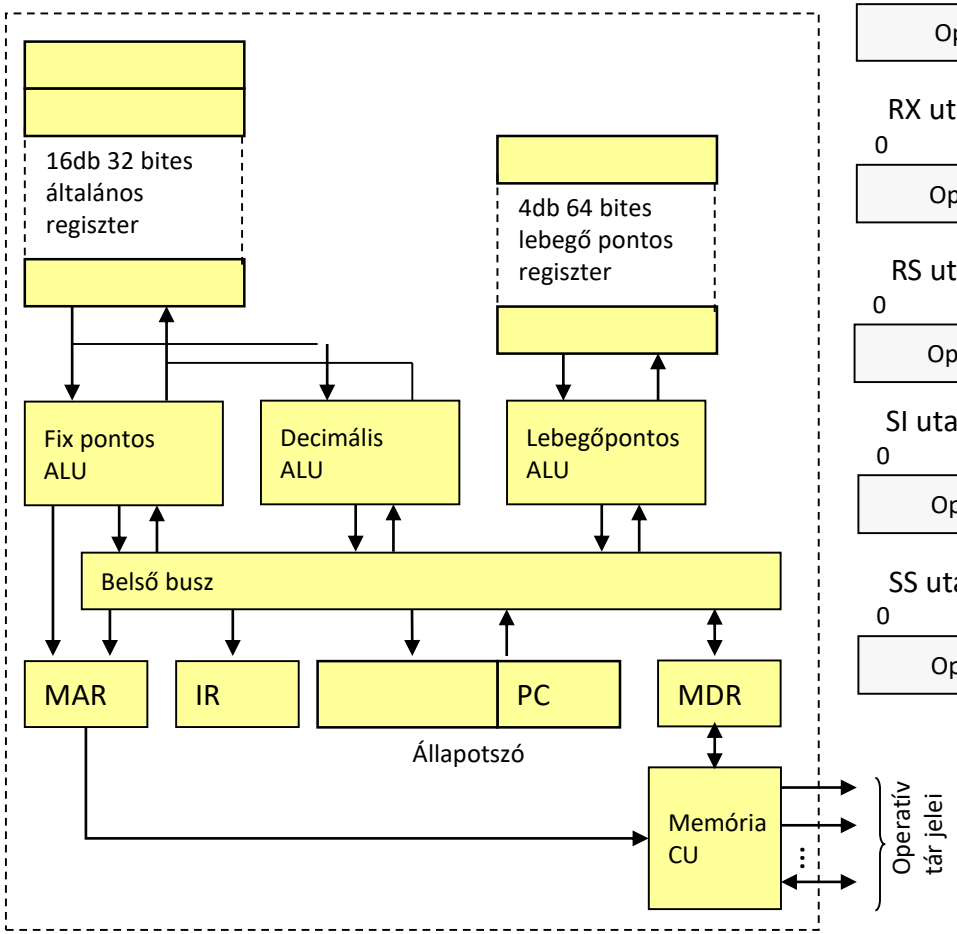
Egy számítógépes rendszer hardverből és szoftverből áll. A kettő egymás nélkül nem létezhet, de aránya folyamatosan változik.

A szoftver technológiák fejlődésével, a hálózati és elosztott rendszerek elterjedésével egyre több szoftver réteg rakódik egymásra, ami egyre nagyobb memória kapacitást és hatékonyabb működést követel a hardvertől.

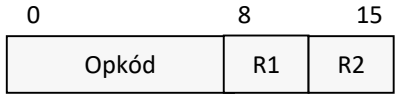
Az ISA szint tehát időben változik, és meghatározása a processzor gyártók kezében van.



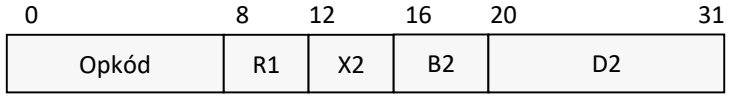
IBM/360-370 CPU belső architektúrája és utasításszerkezete



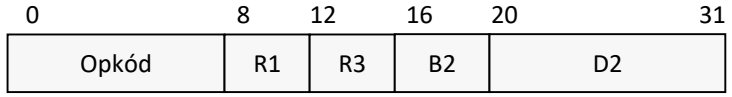
RR utasítás



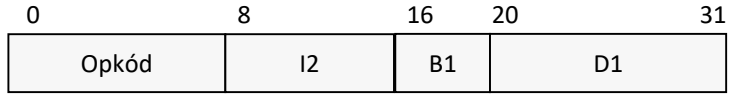
RX utasítás



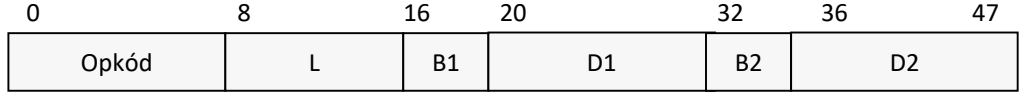
RS utasítás



SI utasítás

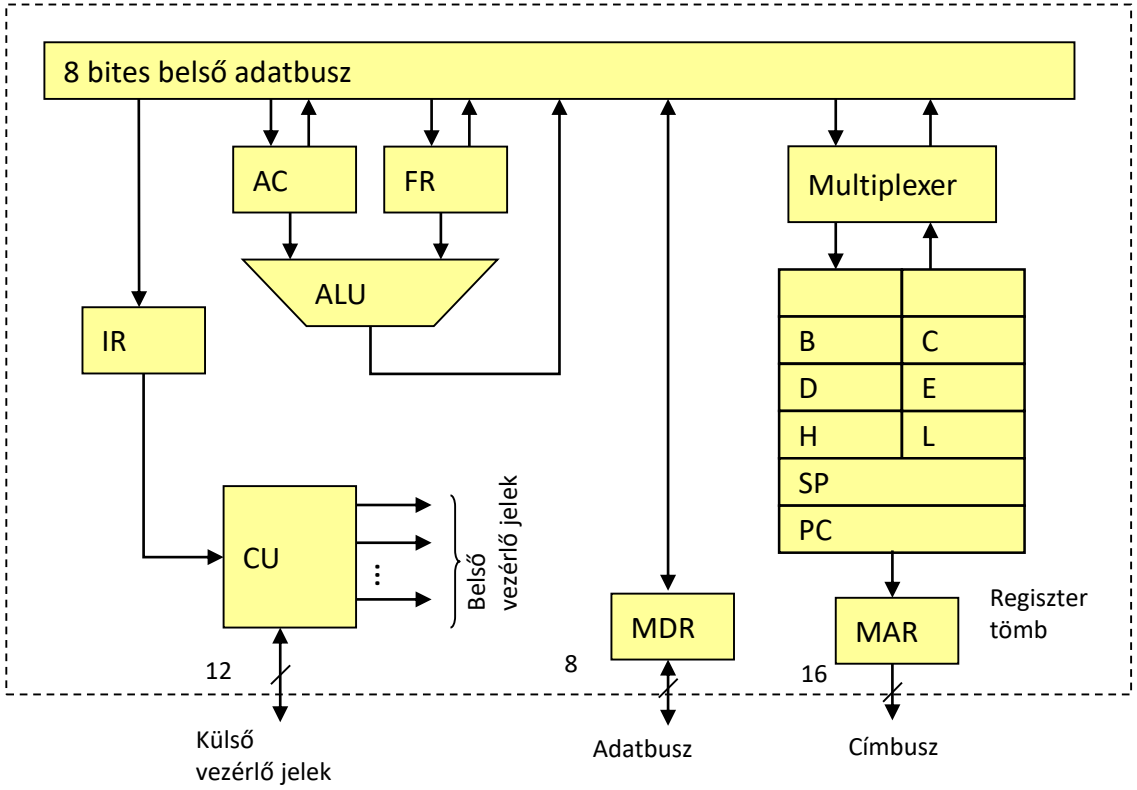


SS utasítás



R - általános célú regiszter
X - index regiszter
B - bázis regiszter
D - cím eltolás
L -operandus hossza
I - immediate operandus

Intel 8080 belső architektúrája és utasításkészlete



Adatmozgató utasítások

Rx	One of: A,B,C,D,E,H,L,[HL]
imm8	An 8-bit immediate value
imm16	A 16-bit immediate value
RP	One of: BC, DE, HL, SP
BP	One of: BC, DE, HL, PSW (i.e., A and flags)
BD	One of: BC, DE
addr	A 16-bit address

8-Bit Transfers

MOV Rd,Rs	Copies data from Rs to Rd (Rs and Rd not both [HL])
MVI Rd,*imm8*	Moves 8-bit immediate into Rd
LDA A,[*addr*]	Loads 8 bits from memory at *addr* into A register
STA [*addr*],A	Stores A register into memory at *addr*
LDAX A,[BD]	Loads 8 bits from memory address in BC/DE into A
STAX [BD],A	Stores A register into memory address in BC/DE

16-Bit Transfers

LHLD HL,[*addr*]	Loads HL register with 16 bits found at *addr* and *addr*+1
SHLD [*addr*],HL	Stores HL register contents at *addr* and *addr*+1
LXI RP,*imm16*	Moves 16-bit immediate into register pair
PUSH BP	Puts 16 bits of BP onto stack's top, after doing SP=SP-2
POP BP	Places value at stack's top into BP, then SP=SP+2
SPHL	Puts contents of HL into SP (stack pointer)
PCHL	Puts contents of HL into PC (program counter)
XTHL	Exchanges HL with top of stack ([SP])

Műveleti utasítások

8-Bit Arithmetic

ADD A,Rs	Adds contents of register Rs to A register
ADC A,Rs	Adds contents of Rs to A, along with carry bit
SUB A,Rs	Subtracts contents of register Rs from A register
SBB A,Rs	Subtracts contents of Rs from A, borrowing carry bit
CMP A,Rs	Compares contents of Rs with A register
ADI A,*imm8*	Adds 8-bit value to contents of A register
ACI A,*imm8*	Adds 8-bit value with carry into A register
SUI A,*imm8*	Subtracts 8-bit value from contents of A register
SBI A,*imm8*	Subtracts 8-bit value with borrow from A register
CPI A,*imm8*	Compares 8 bit data with contents of A register
INR Rd	Increments register Rd by one
DCR Rd	Decrements register Rd by one
DAA	Convert A register to packed Binary Coded Decimal

16-Bit Arithmetic

DAD HL,RP	Adds contents of register RP to contents of HL register
INX RP	Increments register RP
DCX RP	Decrements register RP

Jelölések

Rx	One of: A,B,C,D,E,H,L,[HL]
imm8	An 8-bit immediate value
RP	One of: BC, DE, HL, SP

Logikai utasítások

8-Bit Logic

CMA	Complement A register
AND A,Rs	Logically ANDs contents of Rs with A register
OR A,Rs	Logically ORs contents of Rs with A register
XOR A,Rs	Exclusive-OR contents of Rs with A register
ANI A,*imm8*	Logically ANDs 8-bit value with contents of A register
ORI A,*imm8*	Logically ORs 8-bit value with contents of A register
XRI A,*imm8*	Exclusive-OR 8-bit value with A register
RAL	Rotate A left: Bit0=C C=Bit7
RAR	Rotate A right: Bit7=C C=Bit0
RLC	Rotate A left through carry: Bit0=Bit7 C=Bit7
RRC	Rotate A right through carry: Bit7=Bit0 C=Bit0

Jelölések

Rx	One of: A,B,C,D,E,H,L,[HL]
imm8	An 8-bit immediate value

Ugró és egyéb utasítások

Jumps, Calls, and Returns

JMP *addr*	Unconditional jump to location *addr*
CALL *addr*	Unconditional subroutine call to location *addr*
RET	Unconditional return from subroutine

Other Instructions

IN *port*	Data from *port* placed in A register
OUT *port*	Data from A register placed in *port*
CMC	Complement carry flag
STC	Set carry flag to 1
HLT	Halt CPU and wait for interrupt
NOP	No operation
DI	Disable Interrupts
EI	Enable Interrupts
RST 0–7	Call to memory address 8 · argument

Feltételes ugrás

condition	JUMP	CALL	RET
Nonzero	JNZ	CNZ	RNZ
Zero	JZ	CZ	RZ
No Carry	JNC	CNC	RNC
Carry	JC	CC	RC
Parity Odd	JPO	CPO	RPO
Parity Even	JPE	CPE	RPE
Plus	JP	CP	RP
Minus	JM	CM	RM

Jelölések

addr	A 16-bit address
port	A port number (0–255)

Intel processzorok utasításszerkezete

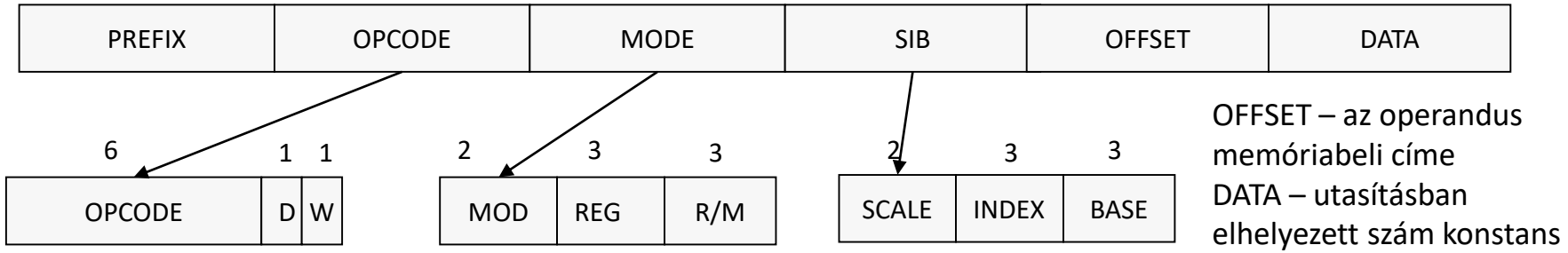
i8086/88 i80286

0-3	1	0-1	0	0-2	0-2

i80386/486 Pentium

0-5	1-2	0-1	0-1	0-4	0-4

PREFIX - Módosító bájt
0xFF – menekülő kód a második műveleti kód bájt jelzésére



OFFSET – az operandus memóriabeli címe
DATA – utasításban elhelyezett szám konstans

MODE – operandus helye

D - adatforgalom iránya
W – operandus mérete (byte/szó)

SIB – kiegészítő információ az operandus helyéről

A Pentium II címzési módszerei

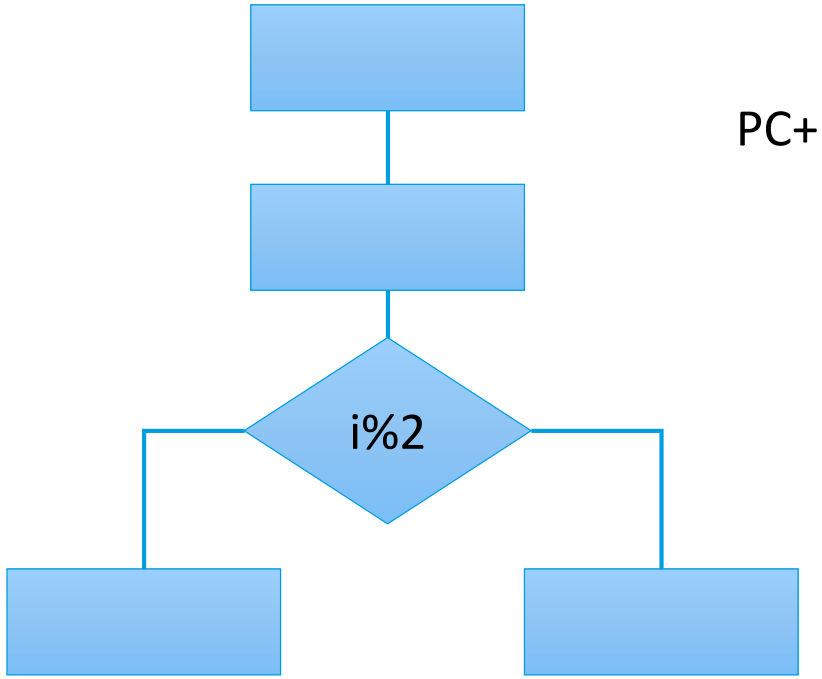
A Pentium II címzési módjai igen szabálytalanok, és attól is függenek, hogy 16 vagy 32 bites az utasítás.
32 bites utasításnál a támogatott címzési módok:

- közvetlen
- direkt
- regiszter
- regiszter-indirekt
- index
- egy speciális mód a tömbelemek címzésére

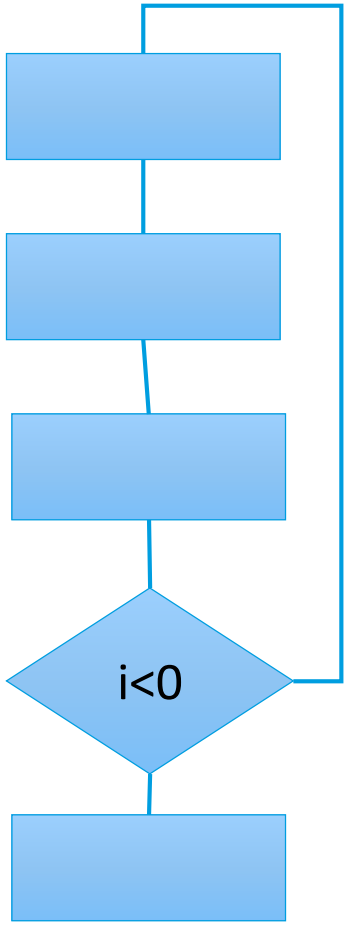
	MOD			
R/M	00	01	10	11
000	M[EAX]	M[EAX + OFFSET8]	M[EAX + OFFSET32]	EAX or AL
001	M[ECX]	M[ECX + OFFSET8]	M[ECX + OFFSET32]	ECX or CL
010	M[EDX]	M[EDX + OFFSET8]	M[EDX + OFFSET32]	EDX or DL
011	M[EBX]	M[EBX + OFFSET8]	M[EBX + OFFSET32]	EBX or BL
100	SIB	SIB with OFFSET8	SIB with OFFSET32	ESP or AH
101	Direct	M[EBP + OFFSET8]	M[EBP + OFFSET32]	EBP or CH
110	M[ESI]	M[ESI + OFFSET8]	M[ESI + OFFSET32]	ESI or DH
111	M[EDI]	M[EDI + OFFSET8]	M[EDI + OFFSET32]	EDI or BH

Nem minden címzési mód alkalmazható minden utasításban, és nem minden regiszter használható minden címzési módban. Ez megnehezíti a fordító programok írását és bizonytalan kódot eredményez.
A MODE bájt tartalma határozza meg a címzési módot. Az egyik operandust a MOD és az R/M mezők határozza meg, a másik operandus mindig regiszter, és a REG mező tartalmazza.
Néhány módban egy SIB bájt követi a MODE bájtot. Ekkor szorozni kell az indexregisztert a SCALE-tól függően (1,2,4,8), hozzá kell adni a bázisregiszterhez és a MOD-tól függően hozzá kell adni 8 vagy 32 bites eltolási értéket

Állapottér szerepe



PC+1

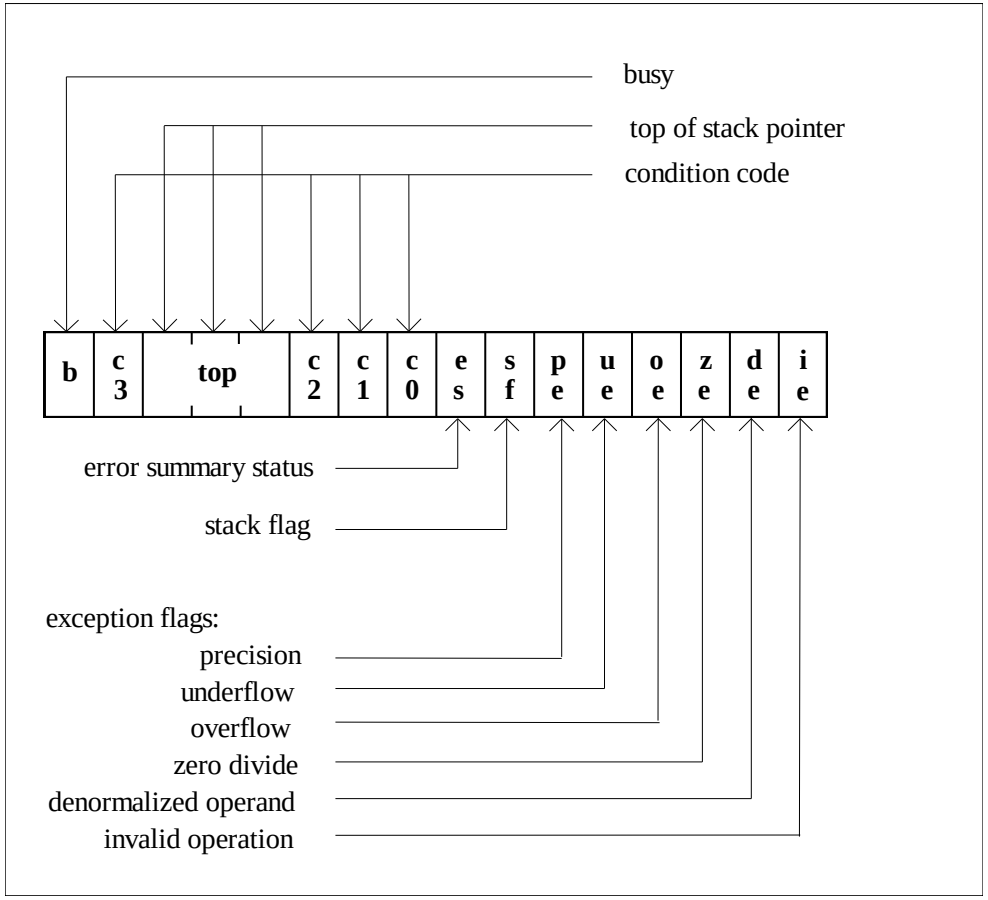


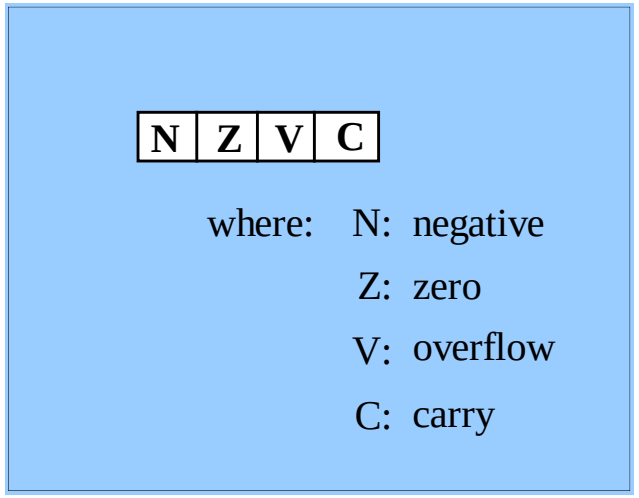
A Neumann számítási modell állapot-átmenet szemantikát követ.

Amikor az adattéren utasítást (manipulációt) hajtunk végre, változik az állapottér (flag-ek) állapota, pl. előjel, zéró, átvitel (carry), páros/páratlan, túlcsordulás, stb.

Állapotjelzők

Az állapotjelzők dedikált indikátorok, közös néven flag-ek. Az állapotjelzőket az állapotszó – PSW tartalmazza, amit az állaporegiszterben tárolunk. Például, az i8086 esetében az adatmanipulációk vonatkozásában a következő flag-eket deklarálták.



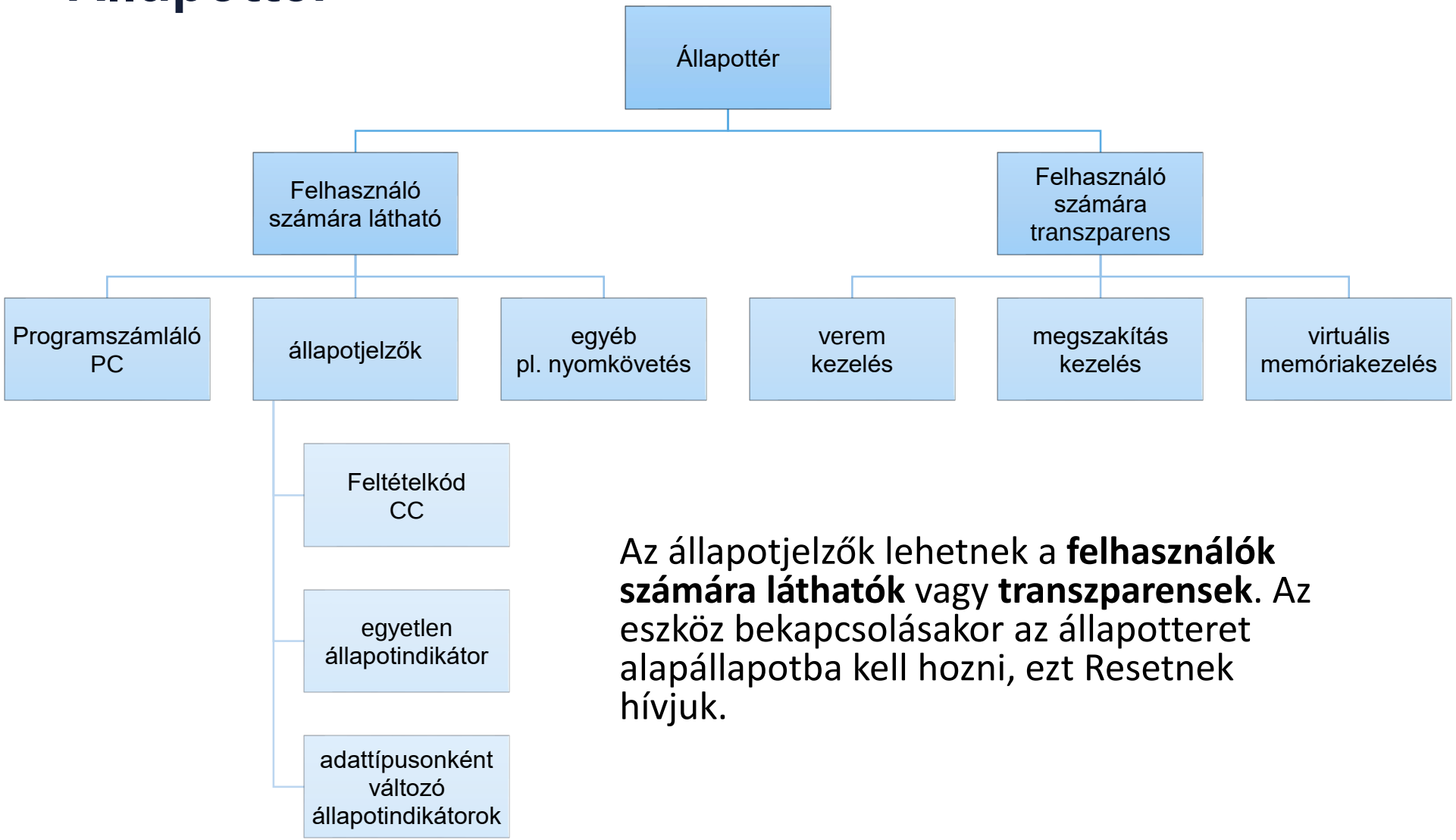


A jelenleg bevezetett architektúrák esetében egyébként a különböző adattípusok esetében önálló flag-készletet deklarálnak.

A lebegőpontos adatok számára ennél a processzornál különálló 32-bites állapotregisztert deklaráltak, melyből csak egy kis rész látható a programozó számára

Önálló flag-ek vannak a fixpontos (és logikai) valamint a lebegőpontos operandusok számára

Állapottér



Az állapotjelzők lehetnek a **felhasználók számára láthatók** vagy **transzparensek**. Az eszköz bekapcsolásakor az állapotteret alapállapotba kell hozni, ezt Resetnek hívjuk.

Állapotműveletek

Az állapotinformációt gyakran tágabb értelemben használjuk, amit Program Status Word (PSW) vagy kontextusnak hívunk. Ennek a célja egy hatékony mechanizmus biztosítása az állapot információ mentésére és visszatöltésére eljárás-híváskor vagy multitaszk üzemmód stb. esetén.

A PSW általában tartalmazza a felhasználó által látott status információt és a rendszer által látott információ egy bizonyos részét. A PSW gyakran korrelál a vezérlőegység státuszával.

A felhasználó által látható állapot helyzet legtöbb komponensét speciális utasítással lehet manipulálni:

- beállítani (set)
- tesztelni (test)
- kiinduló állapotba hozni (reset)
- betölteni (load)
- menteni (store).

A programszámláló (PC) egyébként egy különleges szerepet játszik az állapotműveletek vonatkozásában. Létezik egy egész vezérlés-átadásra szolgáló utasítás készlet.