Digitális technika

XVI.

Mikroszámítógép felépítése
Sínrendszerek
Utasítások felépítése, végrehajtása
Címzési módok

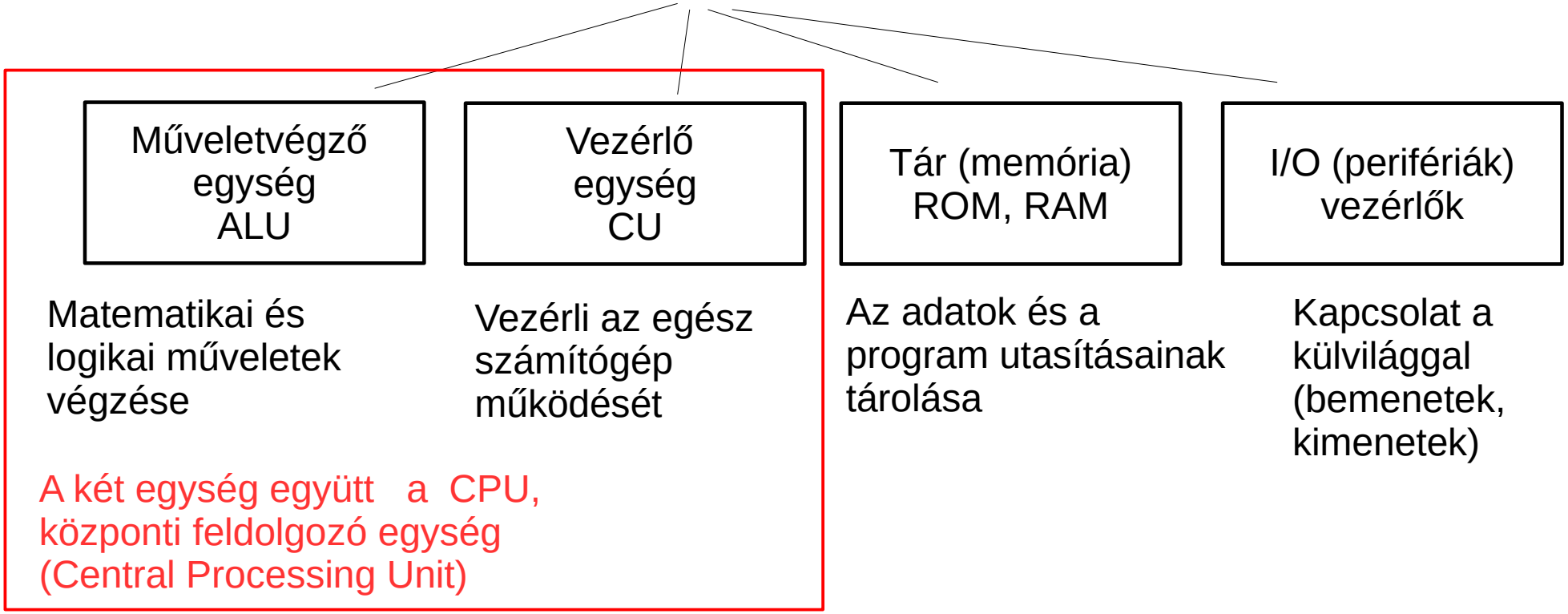
16.1. Számítógép felépítése

- Számítógép

  elektronikus berendezés, adatok bevitelét, tárolását, feldolgozását, eredmények megjelenítését tárolt program alapján automatikusan végzi

  számítási teljesítmény alapján lehet: mikroszámítógép, mini számítógép (szerverek), nagy számítógép, szuperszámítógép

- Mikroszámítógépek felépítése: 4 fő egységből állnak

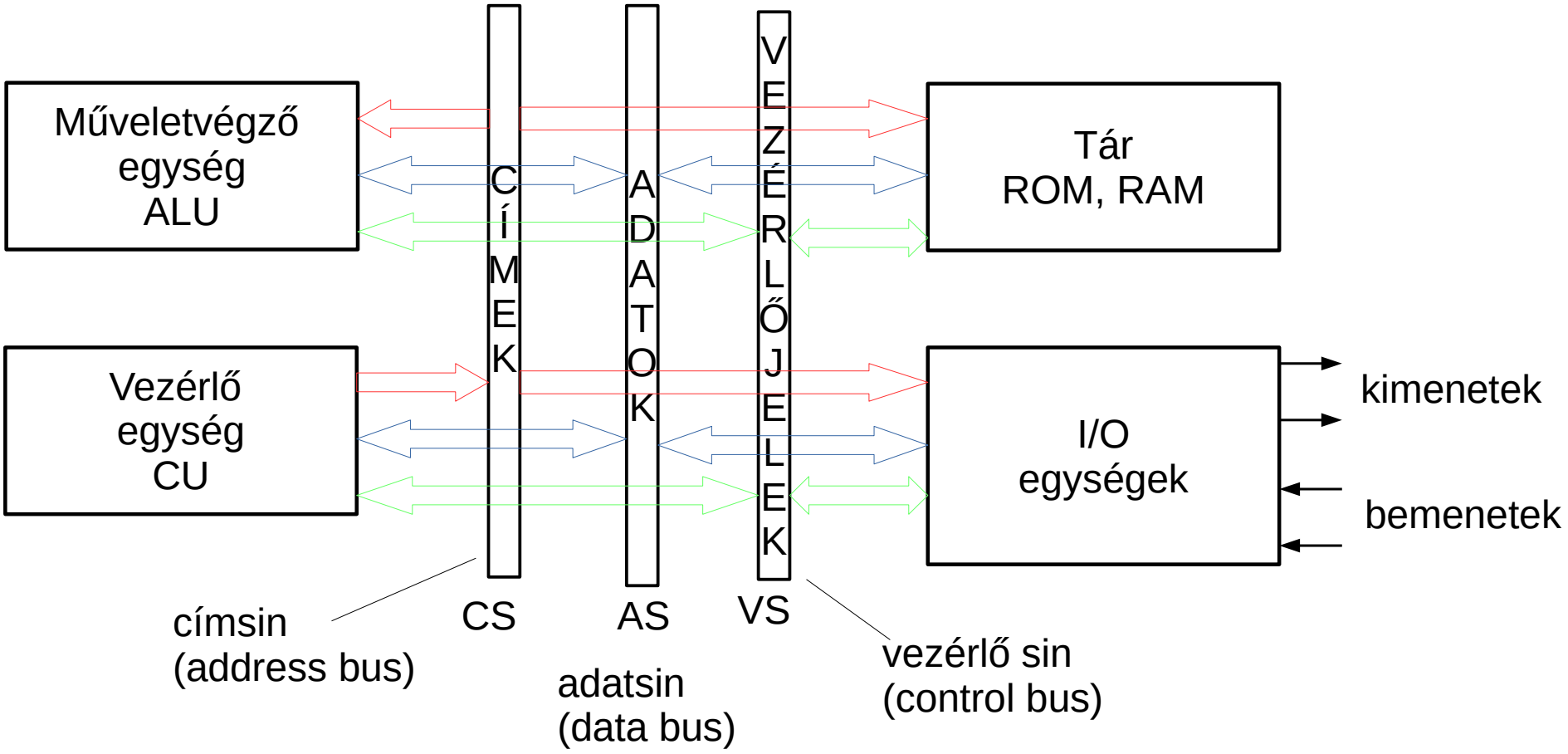


16.1. Számítógép felépítése

- Mikroszámítógépek egységeinek összekapcsolása:

  Az egységek közötti kapcsolatot, az adatok áramlását a sínrendszer (vezetékek) biztosítja.

  Funkció alapján a vezetékek 3 csoportba oszthatók → 3 sín (bus)



16.1. Számítógép felépítése

- Műveletvégző egység

  - ALU (aritmetic and logic unit) matematikai és logikai műveletek elvégzése,
  - műveletek elvégzése után különféle jelzőbitek beállítása (pl. nulla volt-e az eredmény)
  - több regisztert is tartalmaz (adatok átmeneti tárolására) → ezek közül a legfontosabb az akkumulátor, vagy munkaregiszter

- Vezérlőegység

  - CU (control unit), vezérli az egész számítógép működését
  - a tárolt program utasításait egyenként, sorban lehívja, dekódolja, majd a szükséges vezérlő jeleket előállítja (ALU, memória, regiszterek, I/O felé)
  - tartalmaz több regisztert, a legfontosabbak: utasítás regiszter (IR), utasításszámláló (PC)
  - az utasítás regiszterbe töltődik be mindig az aktuálisan beolvasott utasítás
  - az utasításszámláló (PC- program counter, vagy IP- instruction pointer) a soron következő utasítás címét tartalmazza → mindig automatikusan 1-el növekszik

- Memória

  - operatív tár (ROM, RAM), a működtető program utasításait és adatait tárolja
  - a vezérlőegység innen olvassa ki a soron következő utasítást, vagy a szükséges adatot, és ide menti el a tárolni kívánt adatokat

- Perifériák

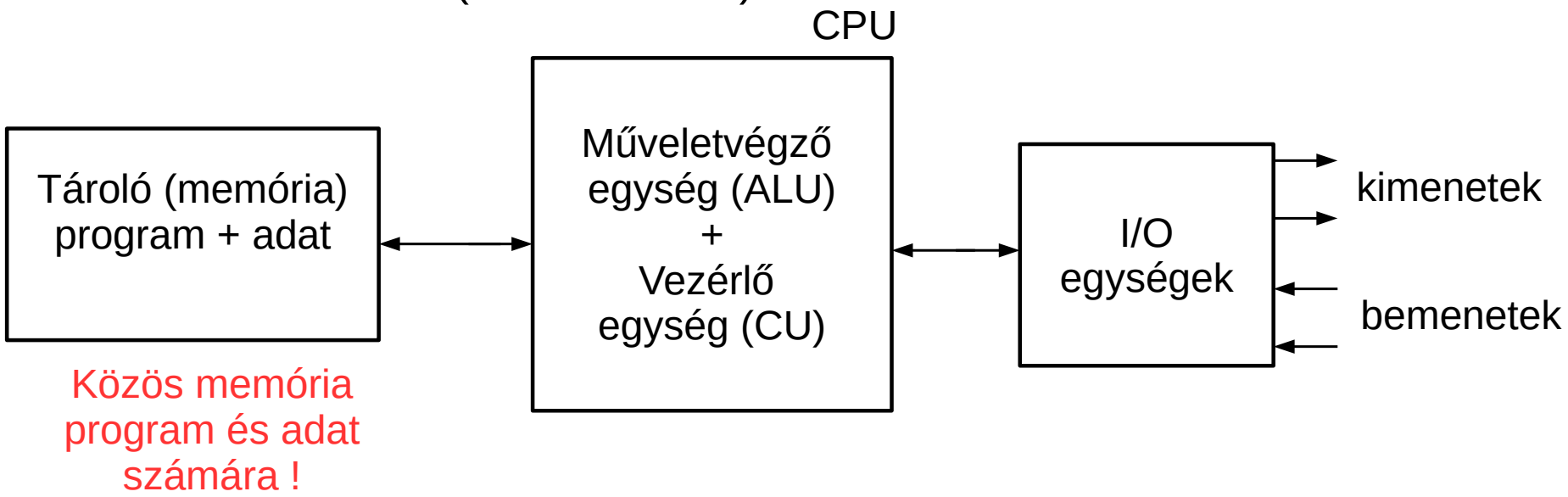
  - Input, output vezérlő, illesztő egységek → kapcsolat a külvilággal
  - a bemenetek, kimenetek (pl. billentyűzet, egér, monitor, nyomtató,..) kapcsolatát biztosítja a rendszerrel

16.1. Számítógép felépítése

- Számítógépek felépítésének, működésének elvei

  Neumann János 1946
  - teljesen elektronikus felépítés (gyors, megbízható)
  - kettes számrendszer használata (egyszerű)
  - belső elektronikus tár az utasítások és adatok tárolására
  - a tárolt utasítások alapján a gép önállóan futtatja a programot
  - soros utasítás végrehajtás (ez ma már nem teljesen így van !)
  - univerzális Turing-gép legyen (bármely aritmetikai és logikai műveletet végre tudja hajtani az alapműveletek véges számú ismétlésével)

- Princeton architektúra (Neumann-elv)



16.2. Számítógépek, mikroprocesszorok fejlődése

- 1944 MARK1 jelfogós számítógép
- 1946 ENIAC első elektronikus számítógép, Neumann-elv → 1. generáció
  ~30 tonna, ~18000 elektroncső → sok meghibásodás → gyakori leállás
  EDVAC
- 1948 tranzisztor felfedezése → 2. generációs számítógépek
  Kis méret, kis fogyasztás, megbízható működés !
  1958 nyomtatott áramkör
- 1962 integrált áramkör (IC) → 3. generáció
  Egy szilícium lapkán sok-sok alkatrész + az összekötő huzalozás elhelyezése
  SSI, MSI integrált áramkörök (n*10, n*100 tranzisztor)
-1970-es évek LSI áramkörök (n*1000 tranzisztor) → 4. generáció
  1971 az első mikroprocesszor, Intel 4004 CPU funkció egy IC-ben
  ~ 2300 tranzisztor, 4 bites adatok, 8 bites utasítások (4kB ROM, 1kB RAM)
  1976 az első mikrovezérlő, TMS1000 (Texas Instruments)
  Egytokos mikroszámítógép (CPU+ROM+RAM+I/O)
- VLSI áramkörök → 5. generáció

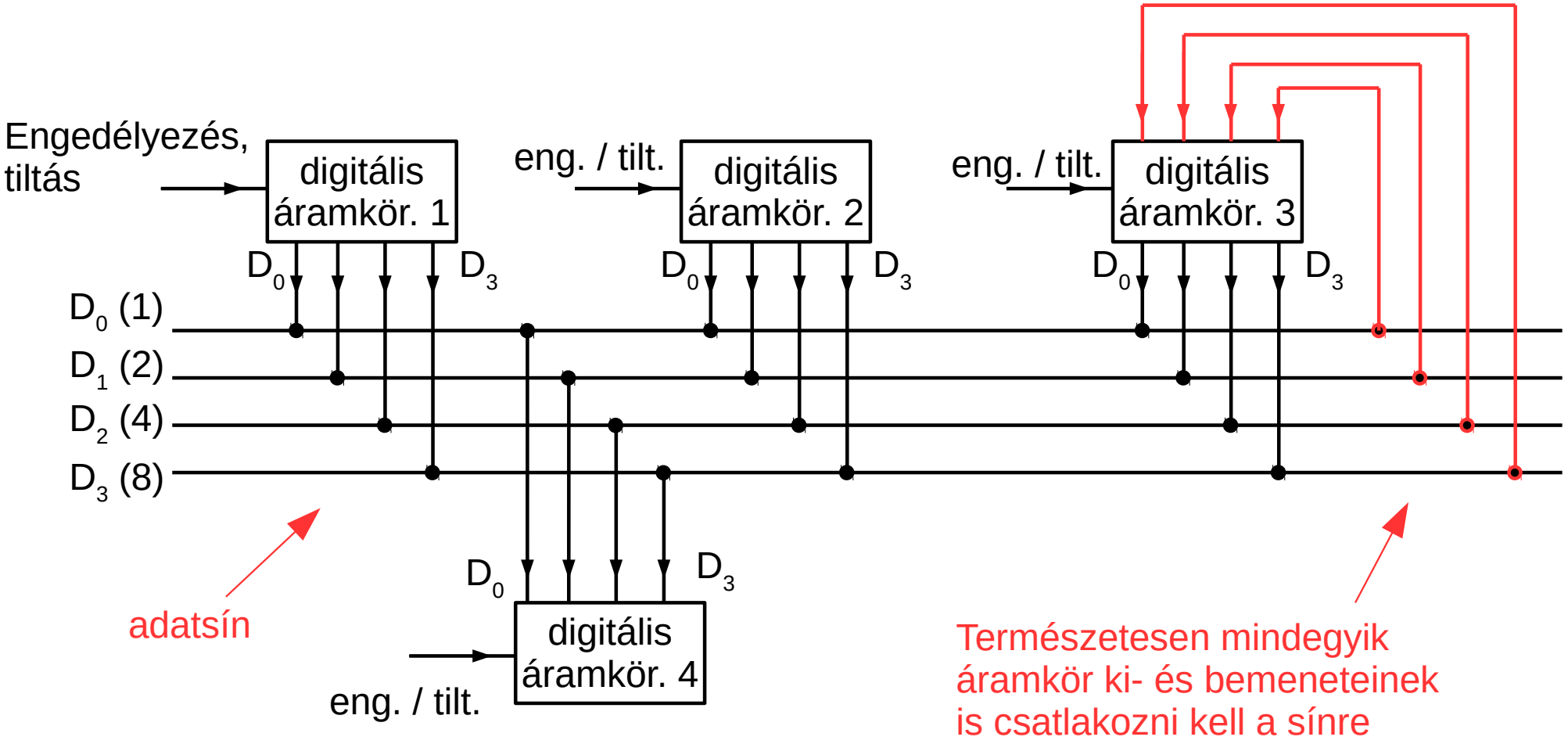
Mikroprocesszorok
- 1971 Intel 4004
- 1974 Intel 8080, Motorola 6800
- 1976 Zilog Z80, Intel8085, Texas TMS9900
- 1978 Intel 8086, Zilog Z8000
- 199x Intel (80486, Pentium), AMD

Mikrovezérlők
- 1976 Texas TMS1000
- 1978 Intel 8051, Motorola MC6801
- 1980 Zilog Z8
- 1982 Intel 2920, Texas TMS320
- 199x Microchip PIC
  ~ 35 utasítás → RISC
  Atmel AVR
  ~ 120 utasítás → CISC

16.3. Sínrendszerek kialakítása

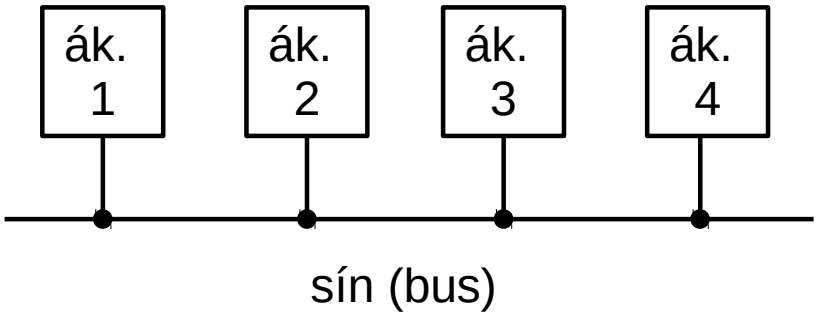
- Bus (sín): vezetékek együttese, amelyek valamilyen hasonló funkciót töltenek be, és az áramkörök erre kapcsolódnak rá párhuzamosan

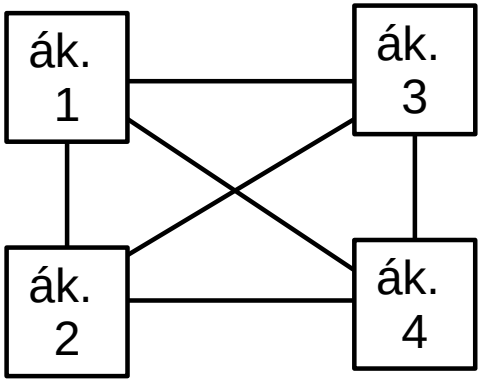
  pl. 4 bites adatok átvitele áramkörök között 4 vezetékkel lehetséges (data bus – adatsín), mindegyik helyi értékhez 1 vezeték tartozik



16.3. Sínrendszerek kialakítása

- Sínrendszer előnyei:
  - kevesebb be- és kimenet kell az áramköröknek így, mintha külön egyesével kötnénk össze azokat
  - rugalmasan lehet bővíteni, új áramköröket rá kötni





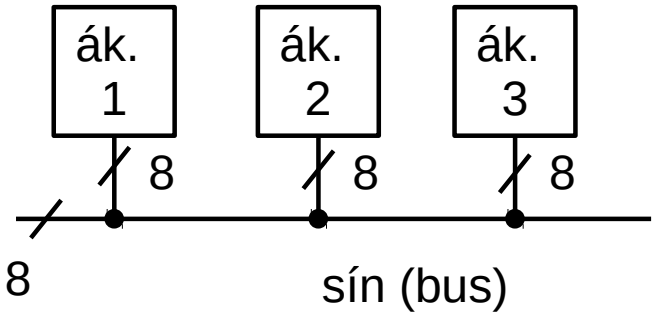
- Meg kell oldani, hogy egyszerre csak egy áramkör lehet az adó !!

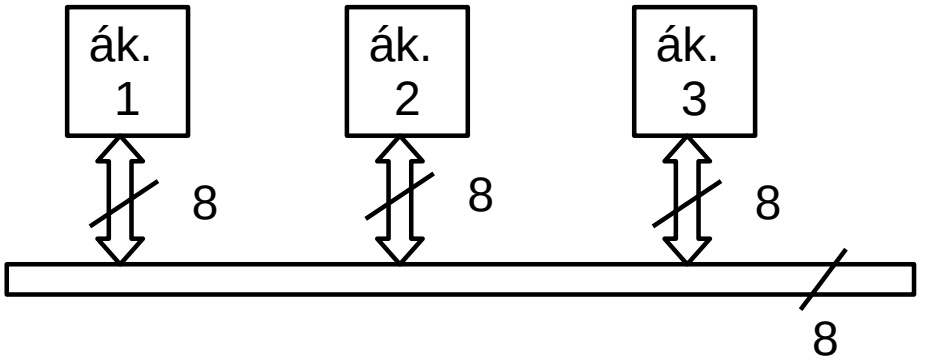
  Az engedélyező/tiltó bemenetekkel kell biztosítani, hogy egyszerre csak egy áramkör küldjön adatokat a sínre, a többi kimenetei addig pl. nagy impedanciás állapotban legyenek (ha tri-state kimenetek vannak)

- Tri-state kimenetű áramkörök közvetlenül rákötheték a sínre, open-collector kimenetek esetén felhúzó ellenállások szükségesek, totem-pole kimenetek pedig közvetlenül nem köthetők sínre

16.3. Sínrendszerek kialakítása

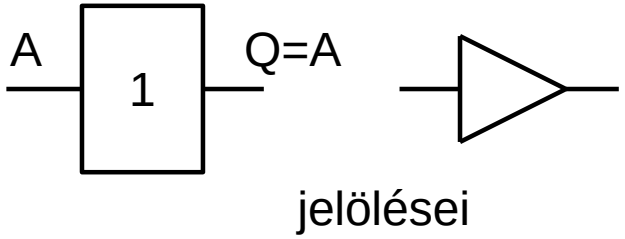
- Általában egyszerűsítve ábrázoljuk a sínt→ a sok vonal helyett csak egy vonal (vagy egy vastagabb nyíl), esetleg jelölve külön a vezetékek száma

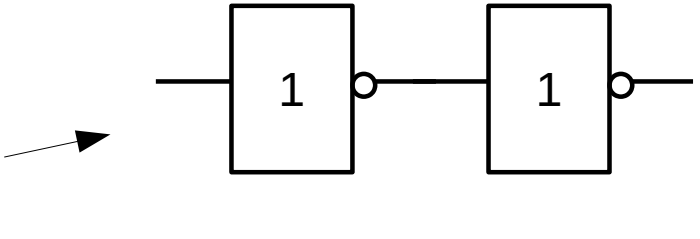




- Sokszor az áramköröknek nincs külön bemenete-kimenete, hanem kétirányú kivezetései vannak, ezt tri-state kimenetű puffer áramkörökkel lehet megvalósítani

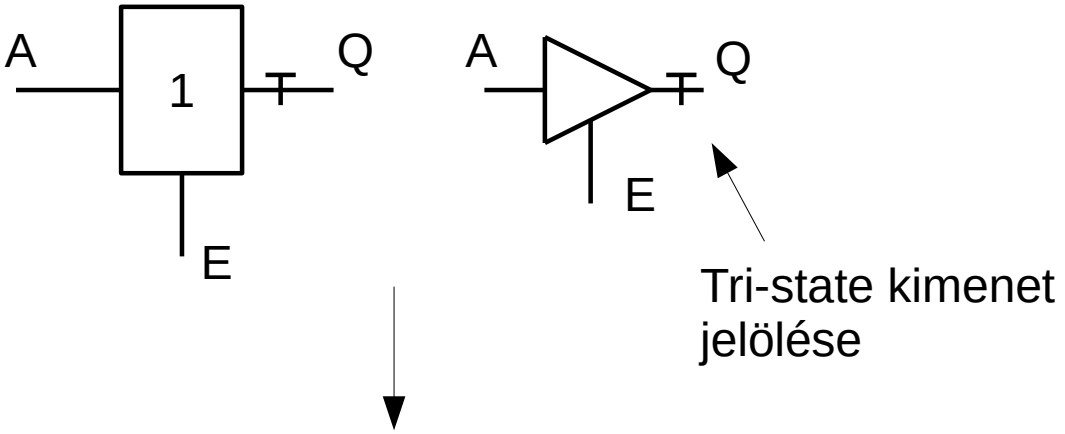
- Puffer, vonali meghajtó áramkör (digital buffer)
  - Erősítő, illesztő elem → akkor alkalmazzuk ha a fan-out értéket kell növelni
  - az inverterhez hasonló, de nem invertál





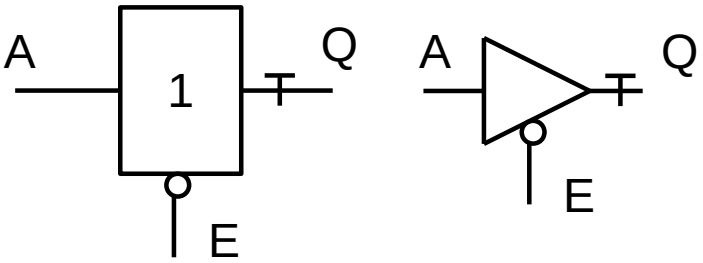
16.3. Sínrendszerek kialakítása

- Puffer, tri-state kimenettel → ilyenkor van engedélyező / tiltó bemenet is !



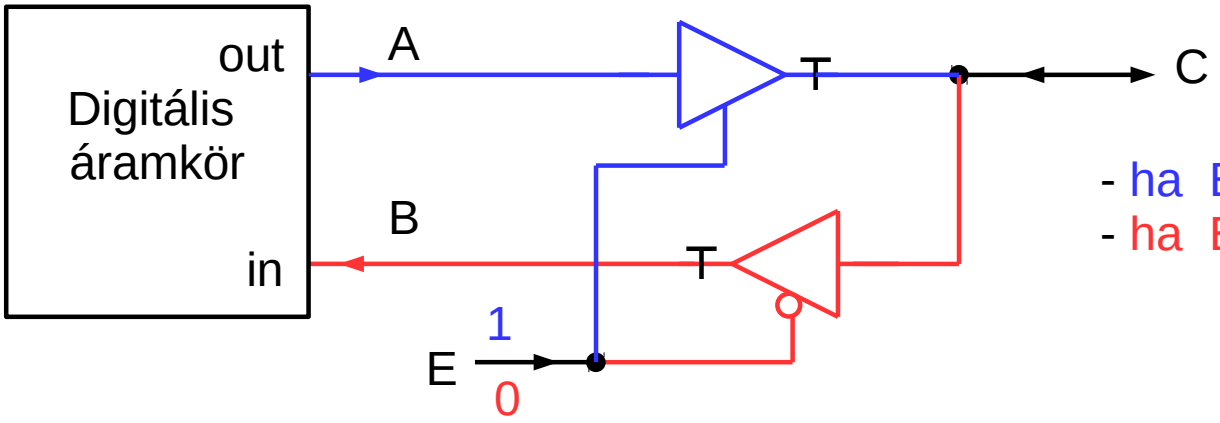
- ha E=0 → Q=Z (nagy impedanciás)
- ha E=1 → Q=A

az engedélyező/tiltó
bemenet lehet invertáló is !



- ha E=1 → Q=Z
- ha E=0 → Q=A

- Felhasználása kétirányú jelvezeték kialakítására



- ha E=1 akkor A → C (C=A)
- ha E=0 akkor C → B (B=C)

16.4. Számítógép utasítások

- Gépi kód

  Az adott mikroprocesszor (mikrovezérlő) saját nyelve. Bináris utasítások!!
  Z80 processzor esetén pl.
  10000001 → A regiszter tartalmához hozzáadja C regiszter tartalmát (ADD A,C)
  PIC esetén pl. 0100 101 0000011 → a 3. regiszter 5. bitjének nullázása (BCF 0x03,5)
  pl.2 00000100000000 → W regiszter nullázása (CLRW)
  Hátránya: használata nehézkes, időigényes, → lassú program fejlesztés!

- Assembly nyelv

  szimbolikus nyelv → a gépi kód minden utasításához
  egy rövid név (mnemonik) tartozik
  Z80 esetén pl. ADD A,C (A ← A+C)
  PIC esetén pl. BCF 0x03,5 (BCF f,b –> bit clear f reg.)
  pl.2 CLRW → W regiszter nullázása (clear W)

  előnye: gyors működésű, kis méretű programok készítése
  hátránya: használata még így is nehézkes, fordítóprogram kell hozzá ! → assembler

- Magas szintű programozási nyelv

  közelebb van az emberi logikához, nyelvhez → használata egyszerű
  elrejti a hardvert! → ez hátrány is lehet
  hátránya: nagyobb méretű, lassúbb programok
  fordítóprogram kell hozzá !
  ilyen nyelvek: C, Java, ...

16.4. Számítógép utasítások

- Gépi kódú utasítások felépítése

MK - Műveleti kód	CM	Cím(ek)

Cím(ek) → az operandusokat, vagy azok címeit tartalmazza

CM → módosító rész (elhagyható)
a műveleti kód vagy a címek pontos értelmezése

- Négy címes utasítás

Alapvetően ilyen kellene, de túl bonyolult, túl sok helyet foglal (~ 50-60 bit lenne)
→ célszerű egyszerűsíteni !

MK	1. operandus címe	2. operandus címe	eredmény címe	következő utasítás címe

- Három címes utasítás

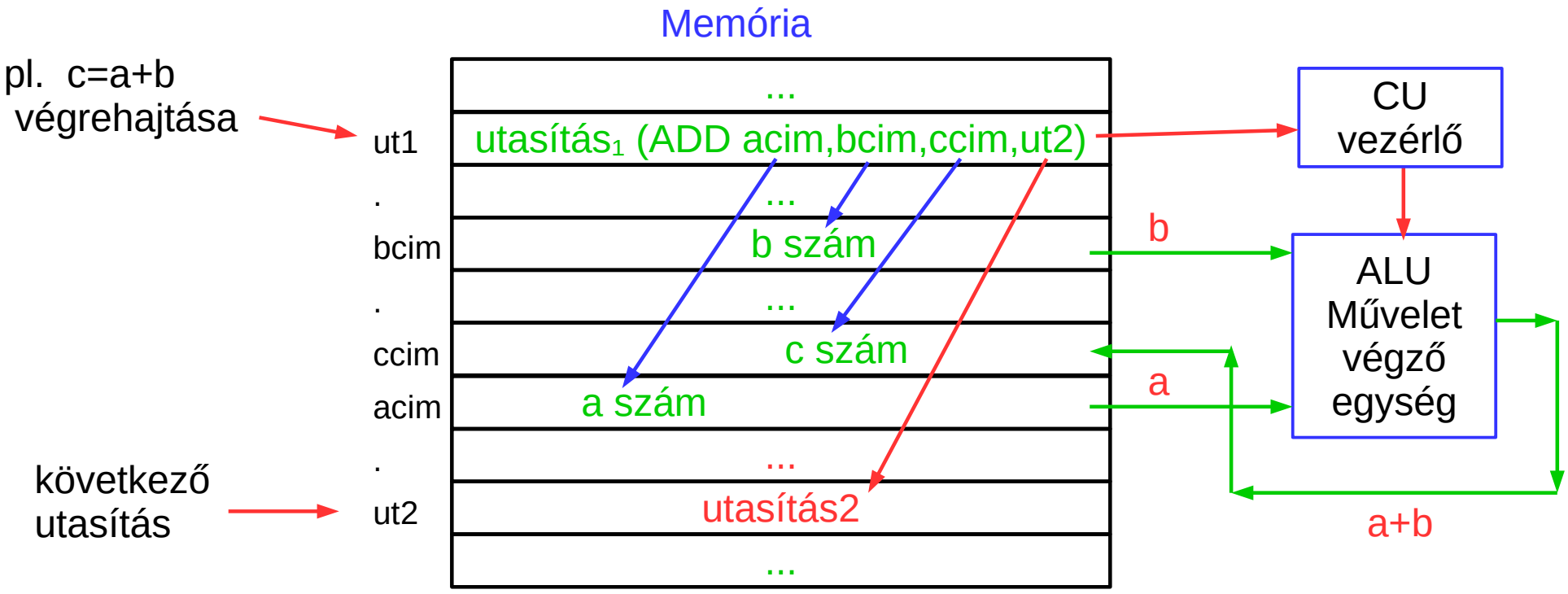
MK	1. operandus címe	2. operandus címe	eredmény címe

A következő utasítás címét mindig egy speciális regiszter tartalmazza (PC vagy IP) !
- alapesetben mindig a következő memória címet → minden utasítás után PC tartalmát 1-el növelni kell !
- plusz ugró utasítások !! → vezérlés átadásokhoz, program elágazásokhoz, ciklusokhoz kellenek

16.4. Számítógép utasítások

- Négy címes utasítás értelmezése

MK	1. operandus címe	2. operandus címe	eredmény címe	következő utasítás címe



16.4. Számítógép utasítások

- Két címes utasítás

~ 32 bit

Az eredmény mindig az egyik operandus helyére íródik vissza →
plusz adatmozgató utasítások is szükségesek (MOVE)

- Egy címes utasítás

~ 12-16 bit → kis számítógépek

Az eredmény és a 2. operandus helye is rögzített !! → egy speciális regiszer → az accumulator (ACC vagy A) vagy munkaregiszter (PIC-ek esetén W regiszter)
→ plusz adatmozgató utasítások is szükségesek a memória és ACC között !!
LOAD vagy LD (betöltés memóriából ACC-be), STORE (ACC mentése memóriába)

- Nulla címes utasítás

egyetlen operandusú művelet esetén
- regiszter tartalmakkal végzett műveletek (törlés, invertálás, léptetés, adat mozgatás)
- veremtár műveletek
- közvetlen operandus megadás → az utasítás tartalmazza magát az operandust

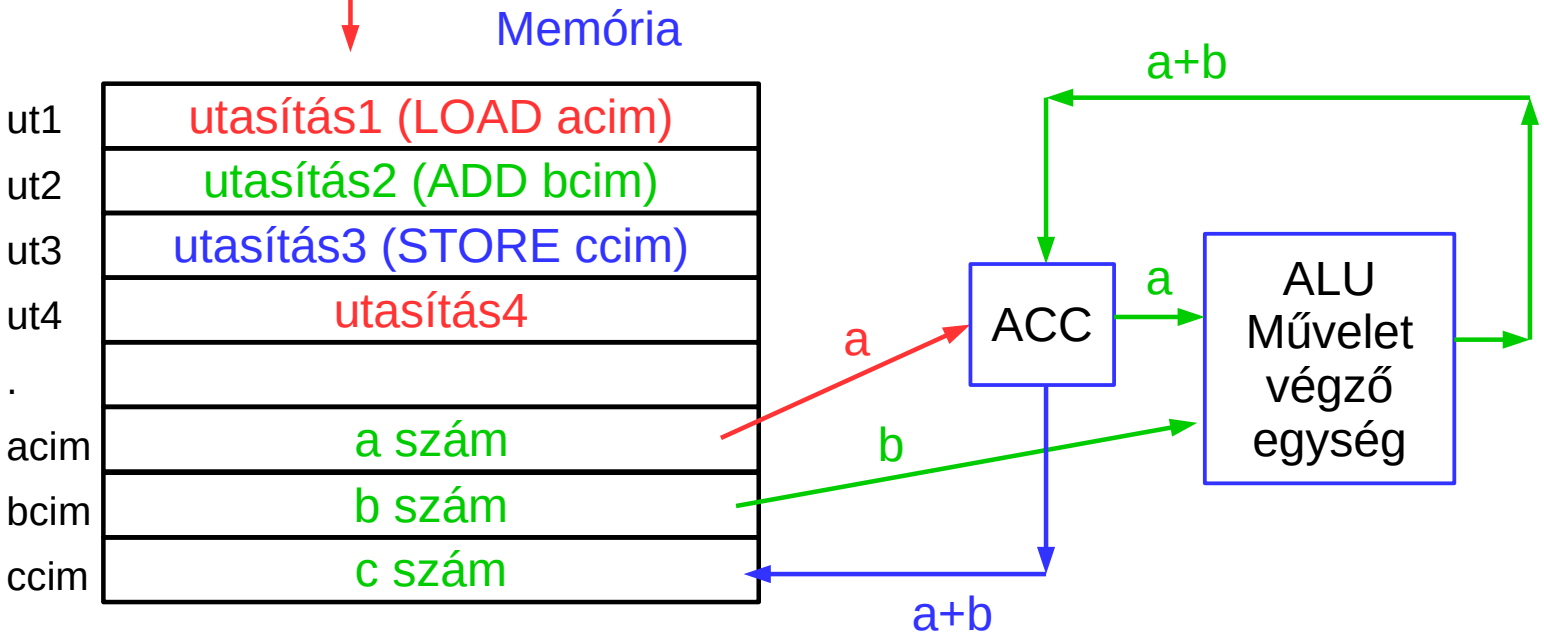
16.4. Számítógép utasítások

- Egy címes utasítás értelmezése

MK	1. operandus címe

pl. c=a+b végrehajtása csak több utasítással lehetséges !

LOAD acim → 'a' szám betöltése ACC-be
ADD bcim → 'b' szám hozzáadása ACC tartalmához ('a'),
→ az eredmény ACC-be kerül
STORE ccim → ACC lementése 'c' számba



16.4. Számítógép utasítások

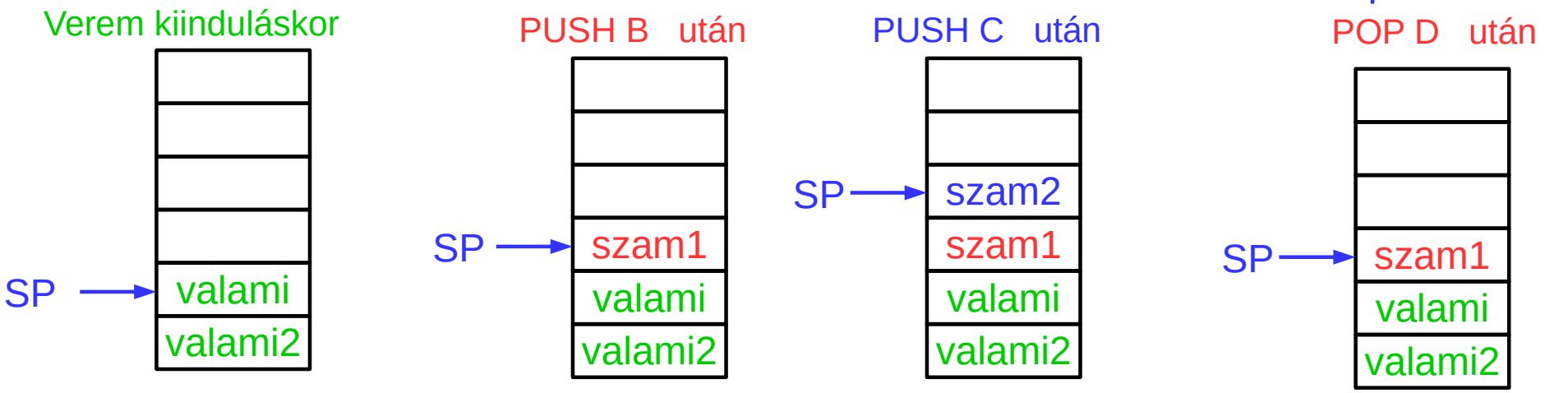
Veremtár (stack)

- a RAM elkülönített helyén kerül kialakításra, külön az adatoktól, program utasításoktól
- a program futása során, speciális mentési feladatokra használjuk →függvény híváskor (CALL) vagy megszakítás esetén
- ide mentődnek el automatikusan a megszakított, otthagyott program legfontosabb adatai (PC, FLAG regiszterek tartalma) → a függvény /megszakítás befejezésekor ezek automatikusan visszatöltődnek → lehet folytatni az eredeti programot
- de mi is lementhetünk ide számunkra fontos regiszterek tartalmát (függvények használatakor) → két speciális utasítás →
  - PUSH forrásregiszter → regiszter tartalmának lementése a verembe
  - POP célregiszter → regiszterbe adatbeolvasása a veremből
- LIFO tár !! → last in – first out → a legutoljára beírt adatot lehet legelőször visszaolvasni
- az SP regiszter tárolja a veremtár aktuális címét, a legutoljára beírt adat címét

Stack kezelése

legutoljára berakott adatot lehet legelőször visszavenni

pl. B regiszterben szam1, C regiszterben szam2 érték van →



16.5. Gépi ciklusok

- Gépi ciklus

  - Egy számítógépes program utasításokból áll (gépi kódú utasítások)
  - **a gépi kódú utasítások gépi ciklusokból állnak → ezek egyszerű műveletek**
  - 1 gépi ciklus általában néhány órajel ciklus alatt hajtódik végre

- Gépi ciklusok típusai

  - **Utasítás lehívás (fetch)** → az utasítás beolvasása a memóriából, →
    a vezérlő ezután értelmezi, dekódolja és irányítja a többi lépést
    Minden utasítás első gépi ciklusa ez.
  - **Memória olvasás**
    Memória megcímzése, majd a kért rekeszből az adat beírása egy regiszterbe
  - **Memória írás**
    Memória megcímzése, majd adat beírása a memória rekeszbe egy regiszterből
  - **Port olvasás** → Periféria megcímzése, majd adat beolvasása belőle egy regiszterbe
  - **Port írás** → Periféria megcímzése, majd adat kiírása egy regiszterből
  - **Belső műveletek**
    A processzor belsejében zajlanak (vezérlőegység és ALU), pl. ALU műveletek
  - **Várakozás** (wait)
    Szünet beiktatása → a processzor vár egy külső egységre
  - **Megszakítás végrehajtása**
  - **Veremtár írás**
  - **Veremtár olvasás**

16.5. Gépi ciklusok

- Gépi ciklusok időzítése

a gépi ciklusok helyes végrehajtásához, az egyes lépések megfelelő sorrendjéhez, késleltetésekhez, ütemezéshez szükség van néhány **vezérlőjelre**

tipikus vezérlőjelek

- órajel (CLK) → az órajel periódusidő ütemezi a gépi ciklusok végrehajtását
  egy gépi ciklus végrehajtása általában néhány ütem (néhány órajel periódus)

- $\overline{\text{MEMWR}}$ → memória írás (általában 0 aktív szintű)
- $\overline{\text{MEMR}}$ → memória olvasás (általában 0 aktív szintű)
- $\overline{\text{IOWR}}$ → port (periféria) írás (általában 0 aktív szintű)
- $\overline{\text{IOR}}$ → port (periféria) olvasás (általában 0 aktív szintű)

egyszerre csak az egyik 0 értékű → meghatározza, hogy milyen gépi ciklus zajlik

vagy esetleg MEMWR, MEMR, IOWR, IOR helyett

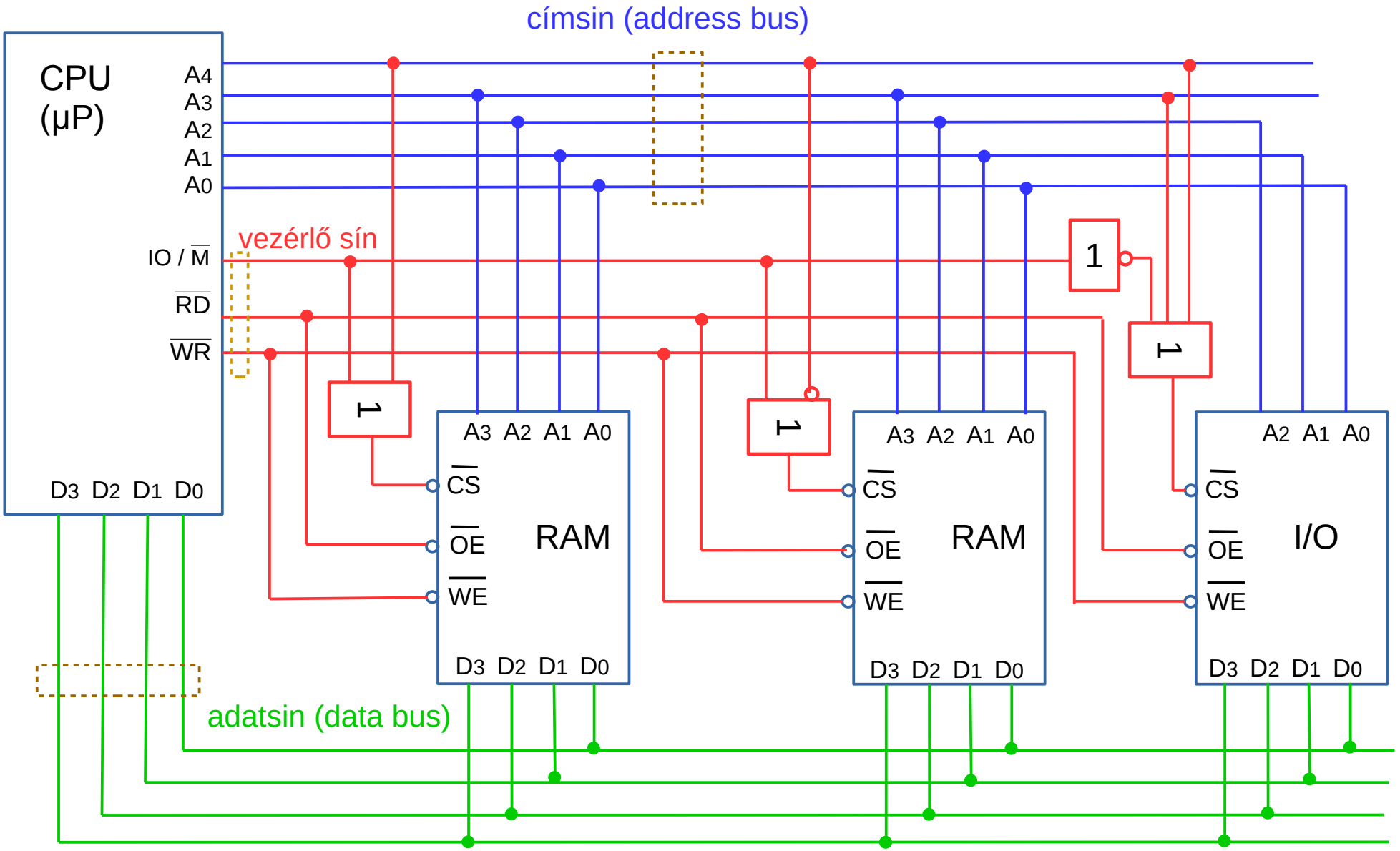
- IO/$\overline{\text{M}}$ → periféria (port) vagy memória művelet jelzése
- $\overline{\text{RD}}$ → olvasás művelet (memória vagy port)
- $\overline{\text{WR}}$ → írás művelet (memória vagy port)

pl. memória olvasás esetén → IO/$\overline{\text{M}}$ =0 és $\overline{\text{RD}}$=0
port írása esetén → IO/$\overline{\text{M}}$ =1 és $\overline{\text{WR}}$=0

16.5. Gépi ciklusok

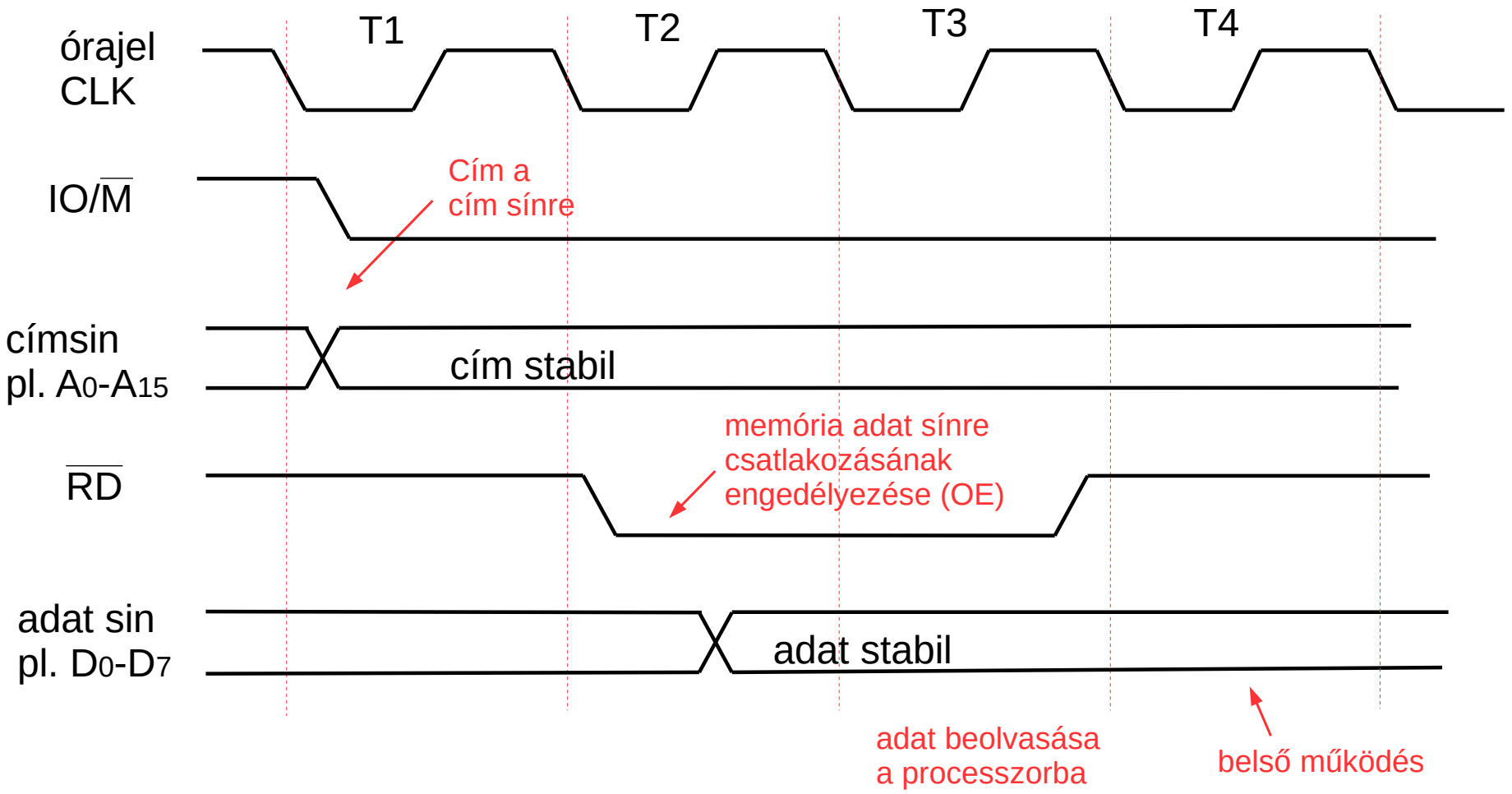
- Gépi ciklusok időzítése

  hardver kialakítása



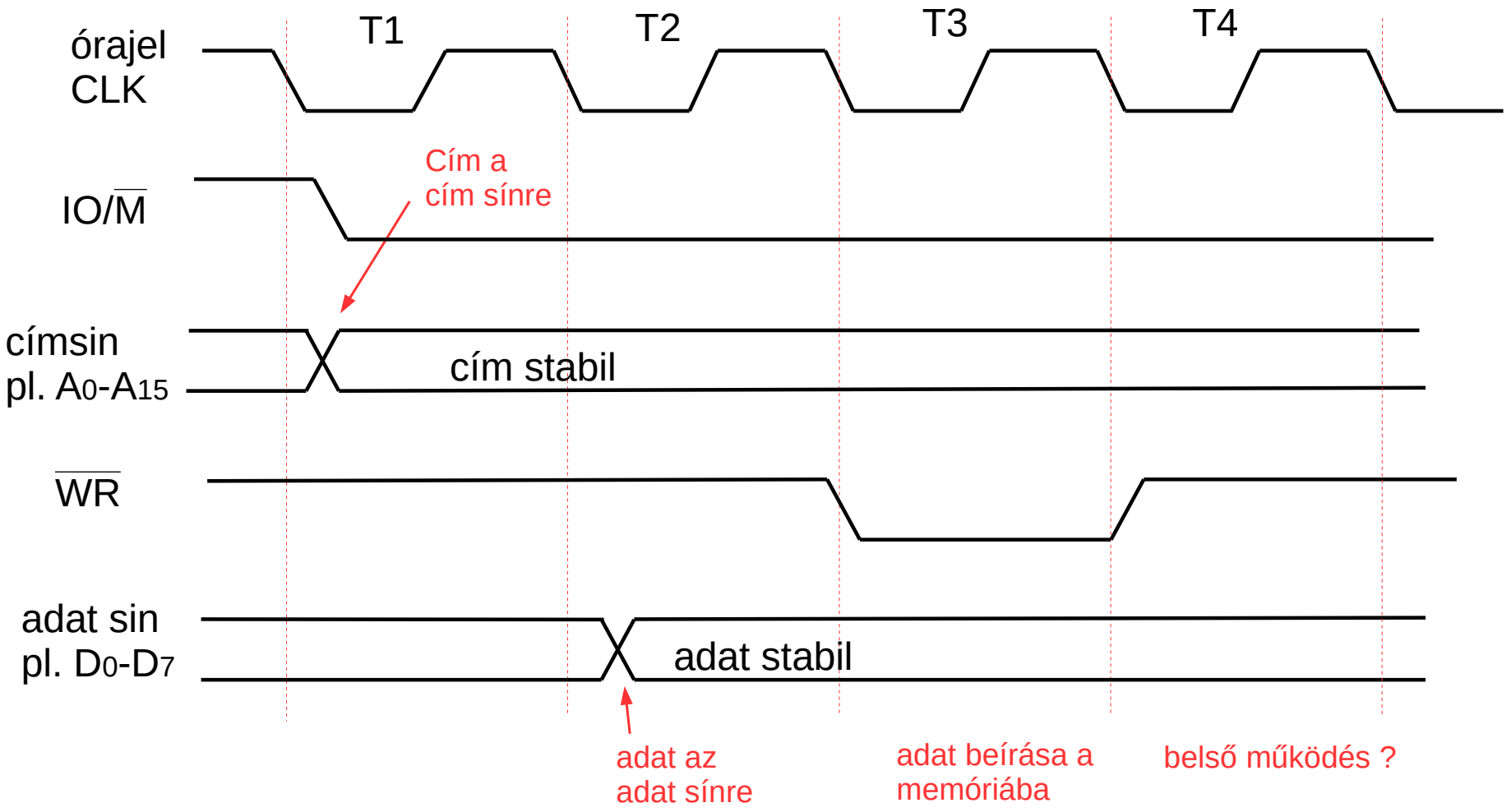
16.5. Gépi ciklusok

memória olvasás



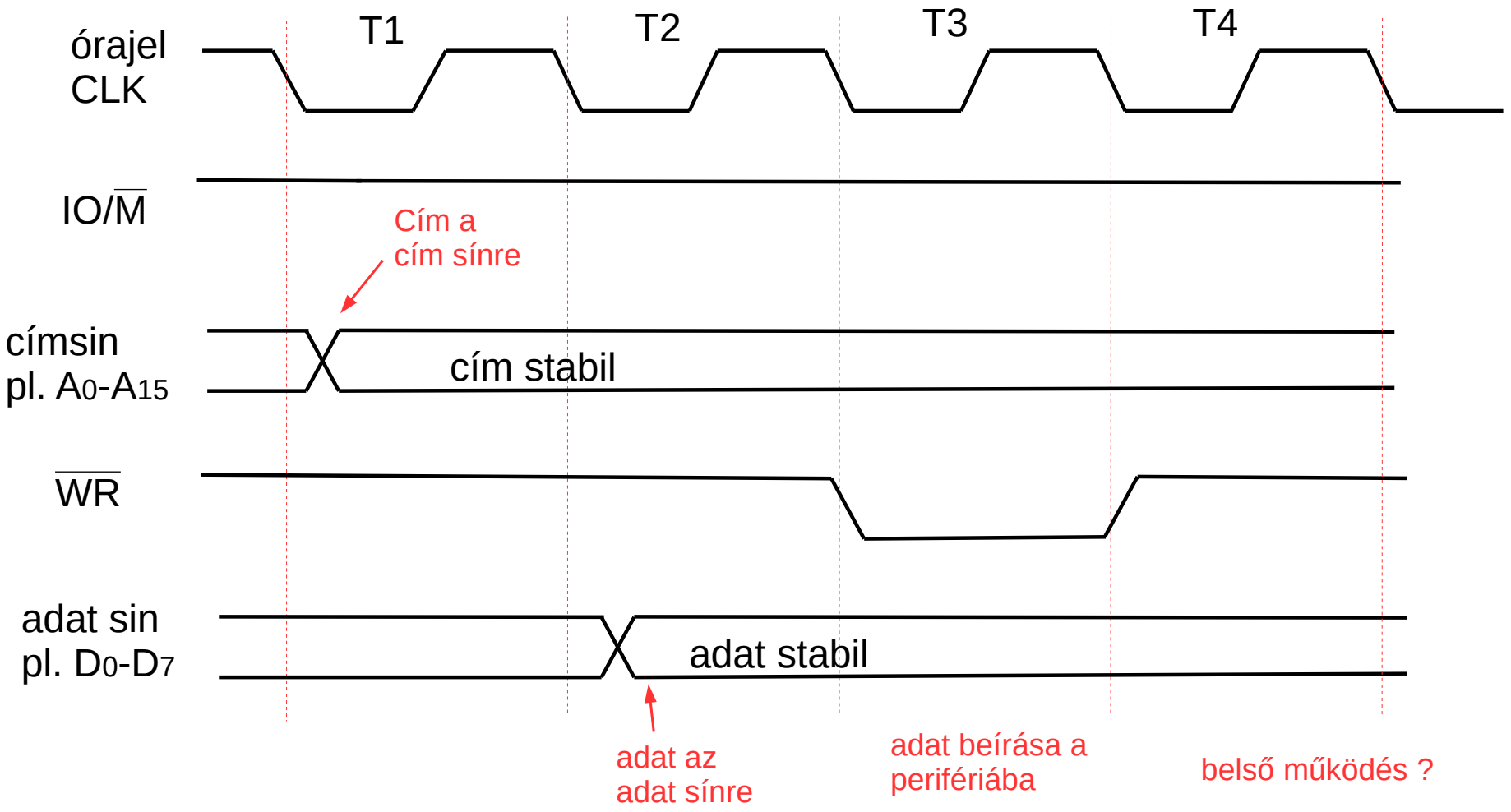
16.5. Gépi ciklusok

memória írás

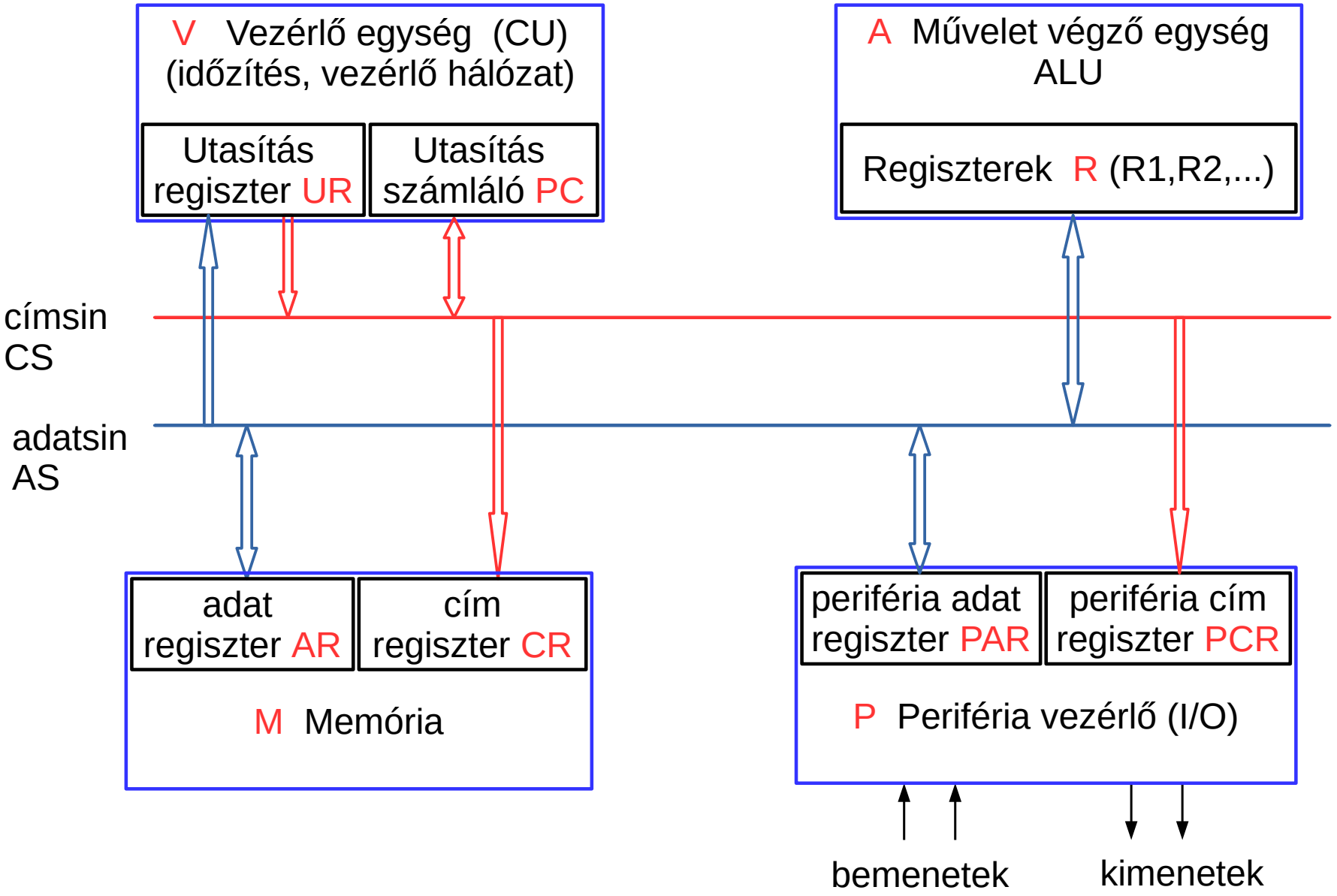


16.5. Gépi ciklusok

port írás



16.6. Alap utasítások végrehajtása



16.6. Alap utasítások végrehajtása

1. Utasítás beolvasás (Fetch)

CS ← PC utasítás címe a címsinre
CR ← CS utasítás címe beíródik a memória címregiszterébe
PC ← PC+1 következő utasításcím előkészítése
AR ← M(CR) utasítás a memória rekeszből az adat regiszterbe
AS ← AR az utasítás kódja az adatsínre kerül
UR ← AS az utasítás kódja az adatsínről a vezérlő utasítás regiszterébe
V ← UR(MK) az utasítás műveleti kód részének dekódolása (vezérlő) →
→ ebből tudja meg a vezérlő a további teendőket

2. Adat beolvasás memóriából egy regiszterbe (Load)

Fetch !! minden utasítás első gépi ciklusa
CS ← UR(CIM) az operandus címe a címsínre
CR ← CS az adat címe beíródik a memória címregiszterébe
AR ← M(CR) az adat a memória rekeszből az adat regiszterbe
AS ← AR az adat az adatsínre kerül
R ← AS az adat az adatsínről az R regiszterbe kerül (ez legtöbbször az ACC)

3. Adat mentése a memóriába egy regiszterből (Store)

Fetch !! minden utasítás első gépi ciklusa
CS ← UR(CIM) a kívánt memória cím a címsínre
CR ← CS az adat címe beíródik a memória címregiszterébe
AS ← R az adat (a regiszter tartalma) az adatsínre kerül
AR ← AS az adat az adatsínről az adat regiszterbe
M(CR) ← AR az adat beíródik a memória megfelelő rekeszébe

16.6. Alap utasítások végrehajtása

4. Összeadás (Add)

1. operandus címe az utasításban 2. operandus R2 regiszterben, eredmény R2 regiszterbe
Fetch !! minden utasítás első gépi ciklusa
CS ← UR(CIM) az 1. operandus címe a címsínre
CR ← CS az 1. adat címe beíródik a memória címregiszterébe
AR ← M(CR) az 1. adat a memória rekeszből az adat regiszterbe
AS ← AR az 1. adat az adatsínre kerül
R1 ← AS az 1. adat az adatsínről az R1 regiszterbe kerül
R2 ← R1+R2 műveletvégzés (ALU)

5. Adat beolvasása perifériáról (In)

Fetch
CS ← UR(CIM) a kívánt periféria cím a címsínre
PCR ← CS az adat címe beíródik a periféria címregiszterébe
PAR ← P(PCR) az adat a periféria megfelelő regiszteréből az adat regiszterbe
AS ← PAR az adat az adatsínre kerül
R ← AS az adat az adatsínről az R regiszterbe kerül

6. Ugrás (Jump)

Ha el kell térni az utasítások eredeti sorrendjétől (pl. elágazás, függvényhívás, ...)
Fetch
CS ← UR(CIM) a kívánt ugrási cím a címsínre
PC ← CS a következő utasítás címe beíródik az utasítás számlálóba

16.6. Alap utasítások végrehajtása

Utasítások csoportosítása

- logikai

AND OR NOT XOR

- aritmetikai

Összeadás, kivonás, (szorzás), ...

ADD (összeadás) SUB (kivonás) INC (növelés 1-el) DEC (csökkentés 1-el)

CLRx (egy regiszter nullázása)

- bit

Bitek 1-be vagy 0-ba állítása, bitek léptetése jobbra, balra, ...

SETB vagy BSF (egy bit 1-be állítása) CLRB vagy BCF (egybit törlése)

- adat mozgató

Adat betöltése memóriából regiszterbe, adat kiírása regiszterből memóriába, adat betöltése regiszterből regiszterbe

MOV vagy LOAD, STORE vagy LD

- program vezérlő

Ugró, függvény hívó utasítások, speciális utasítások

JMP (ugrás egy címre) CALL (függvény hívás)

RET (visszatérés függvényből)

NOP (üres utasítás → nem csinál semmit)

16.7. Címzési módok

Az egy címes utasítás

D → eltolás (displacement)

CM → módosító rész (elhagyható)
a műveleti kód vagy a címek pontos értelmezése

- Az utasítások címrésze (D) általában az operandusok tényleges (effektív) címének kiszámításához szükséges információt tartalmazza (nem biztos hogy közvetlenül a címet)
- **az effektív cím kiszámításának lehetőségei többfélék lehetnek → címzési módok** (cím módosítás)

A többféle címzési mód okai:
- az utasítás címrésze általában nem elég hosszú a teljes memória eléréséhez
- egy adat sorozat elemein kell azonos műveleteket végrehajtani
- a program áthelyezhetőségét kell biztosítani

1. Közvetlen adat címzés

Immediate (vagy literális)
az utasítás címrészében maga az operandus (a szám) van
- gyors, mert nem kell plusz memória olvasás
- csak egyszerű (és viszonylag kis értékű) operandusok

pl. MOVLW k → szám (k) betöltése W regiszterbe, 0=<k<=255 (PIC)
(1100xxkkkkkkkk)

16.7. Címzési módok

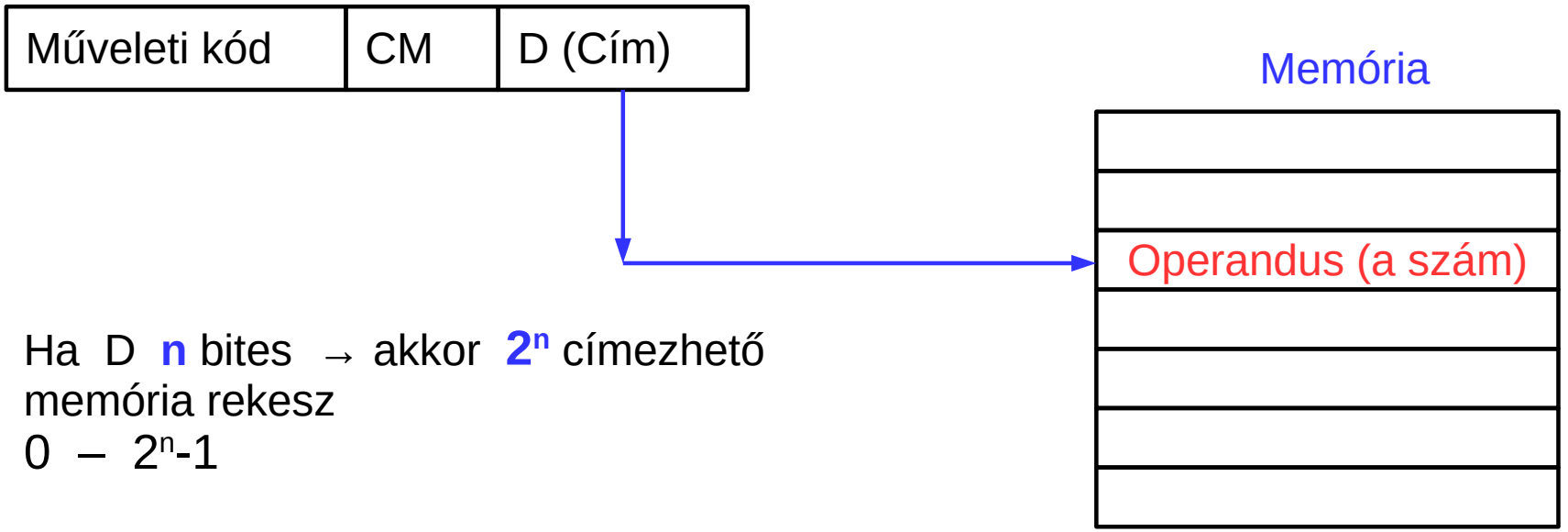
2. Regiszter címzés

a címrészben azt a regisztert jelöljük ki, ahol az operandus (a szám) van

pl. LD A,B → B regiszter tartalmának (a szám) betöltése A (ACC) regiszterbe (Z80)
01111000 → 01 – MK, 111 – 'A' regiszter kódja, 000 – 'B' regiszter kódja

3. Direkt címzés

D az operandus (a szám) tényleges memóriabeli címe → $C_{eff}=D$



Ha D **n** bites → akkor **2^n** címezhető memória rekesz
0 – 2^n-1

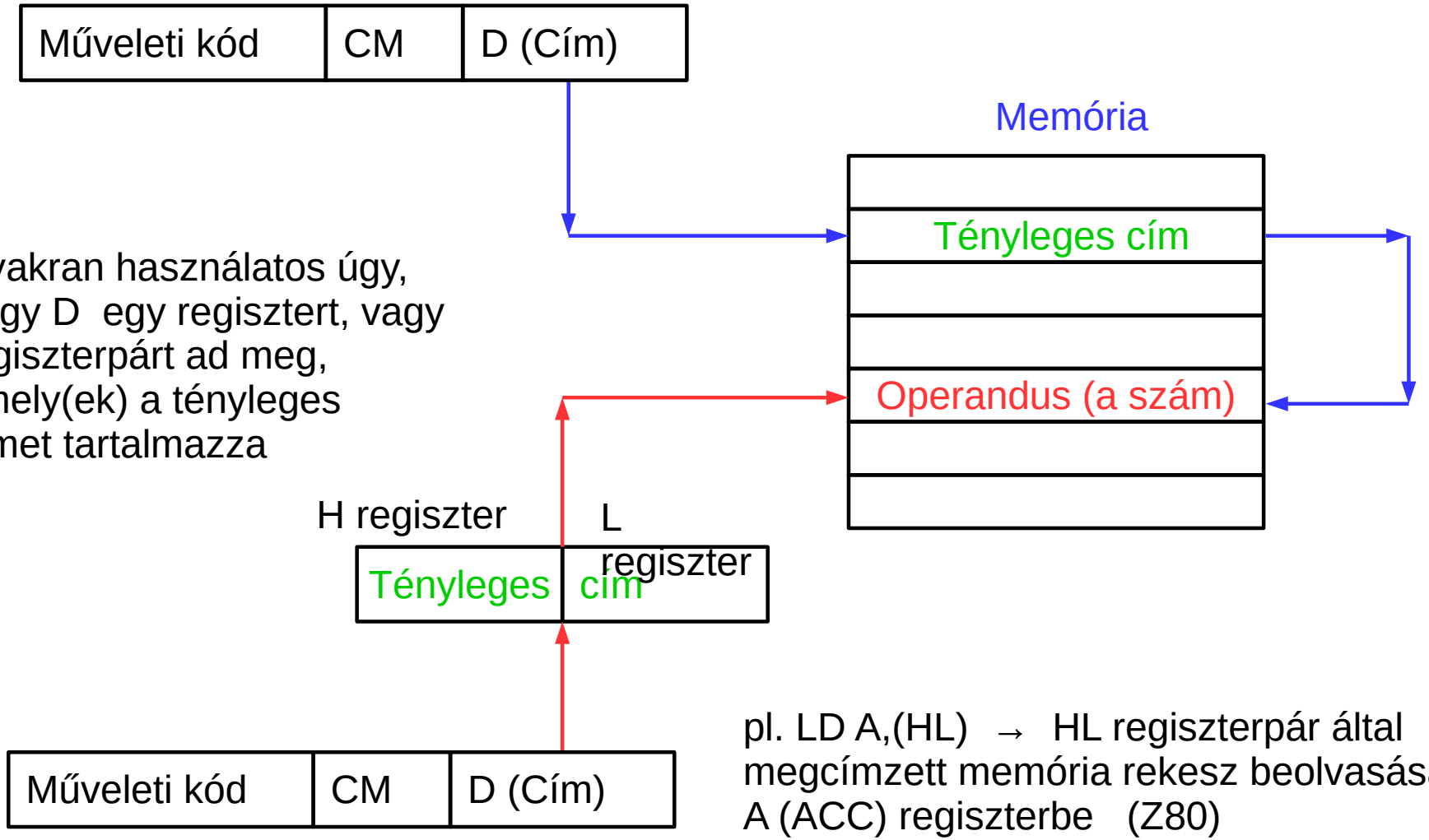
pl. n=8 → 256 megcímezhető rekesz (0-255)

16.7. Címzési módok

4. Indirekt címzés

D a tényleges cím címét adja meg → $C_{eff}=M(D)$

Ha D nem elég hosszú a tényleges cím megadásához
(vagy pl. a program állította elő azt !)



Gyakran használatos úgy, hogy D egy regisztert, vagy regiszterpárt ad meg, amely(ek) a tényleges címet tartalmazza

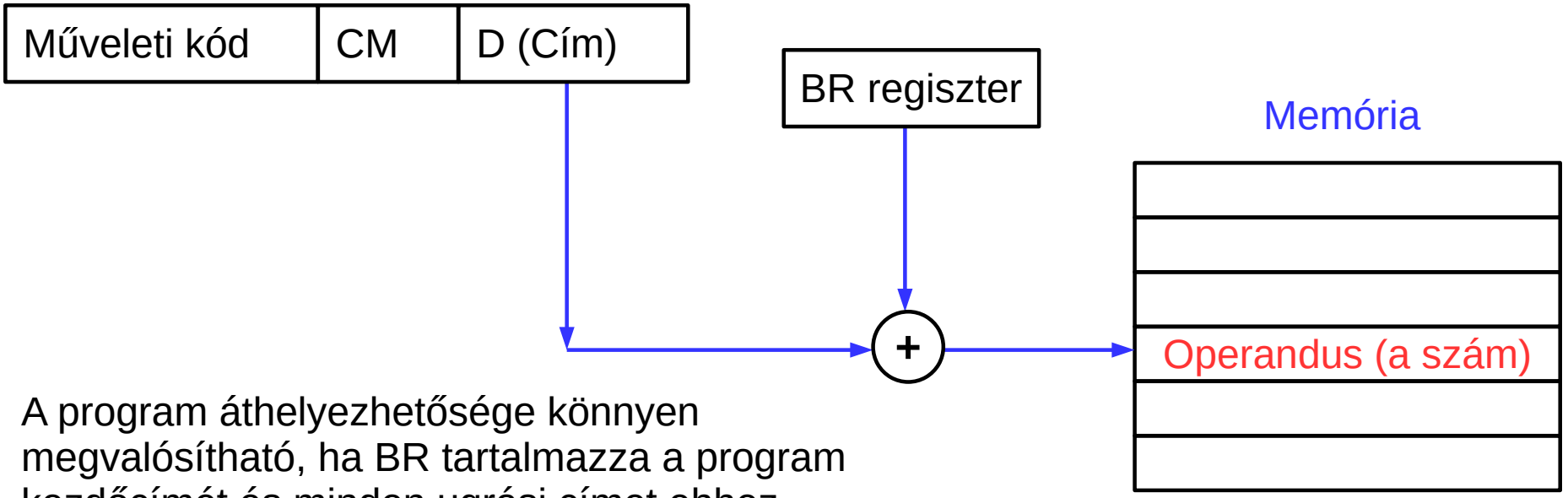
pl. LD A,(HL) → HL regiszterpár által megcímzett memória rekesz beolvasása A (ACC) regiszterbe (Z80)

16.7. Címzési módok

5. Bázis relatív címzés

D egy bázis regiszterhez viszonyítva adja meg a tényleges címet
A bázis regiszter (BR) tartalmához képest +(-) D értéke ! → D előjeles is lehet

C_{eff}=BR+D



A program áthelyezhetősége könnyen megvalósítható, ha BR tartalmazza a program kezdőcímét és minden ugrási címet ehhez viszonyítva adunk meg → áthelyezéskor csak BR tartalmát kell módosítanunk

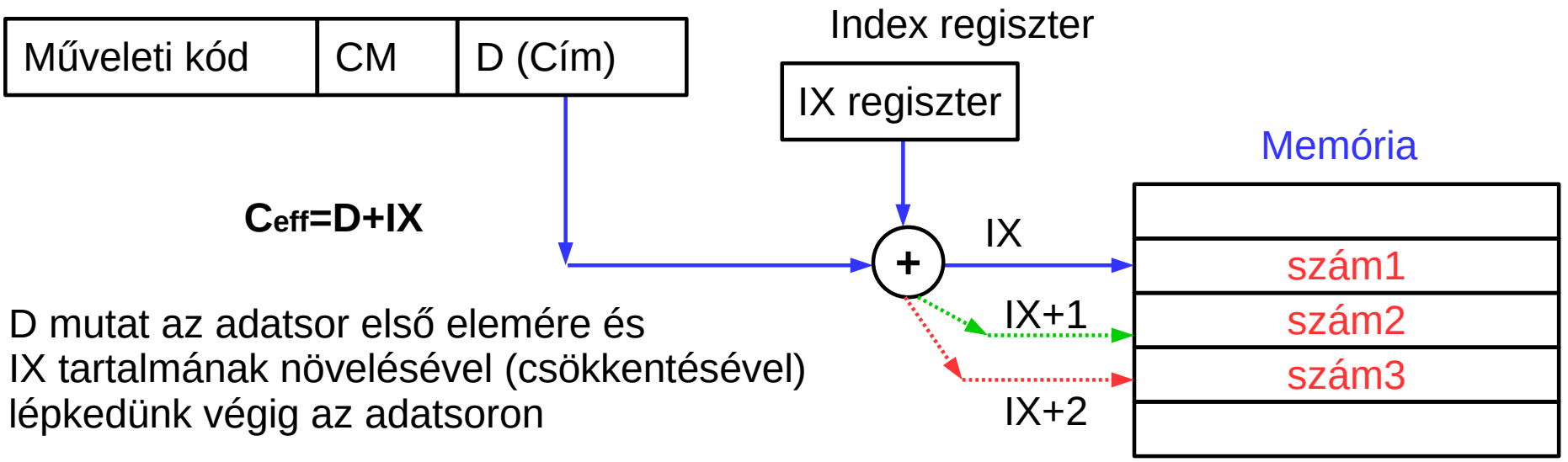
6. Önrelatív címzés

D egy bázis regiszterhez viszonyítva adja meg a tényleges címet itt is
De a bázis regiszter itt a PC regiszter (utasításszámláló) → **C_{eff}=PC+D**

16.7. Címzési módok

7. Indexelt címzés

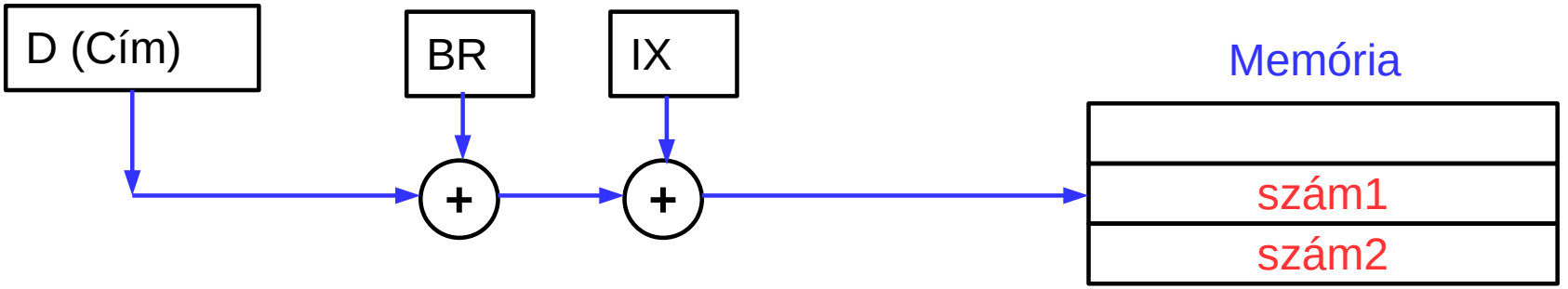
D tartalma itt is egy regiszter tartalmához hozzáadva adja meg a tényleges címet
Adatsorozatokon végzett műveletsorozatnál hasznos



D mutat az adatsor első elemére és
IX tartalmának növelésével (csökkentésével)
lépkedünk végig az adatsoron

8. Összetett címzés

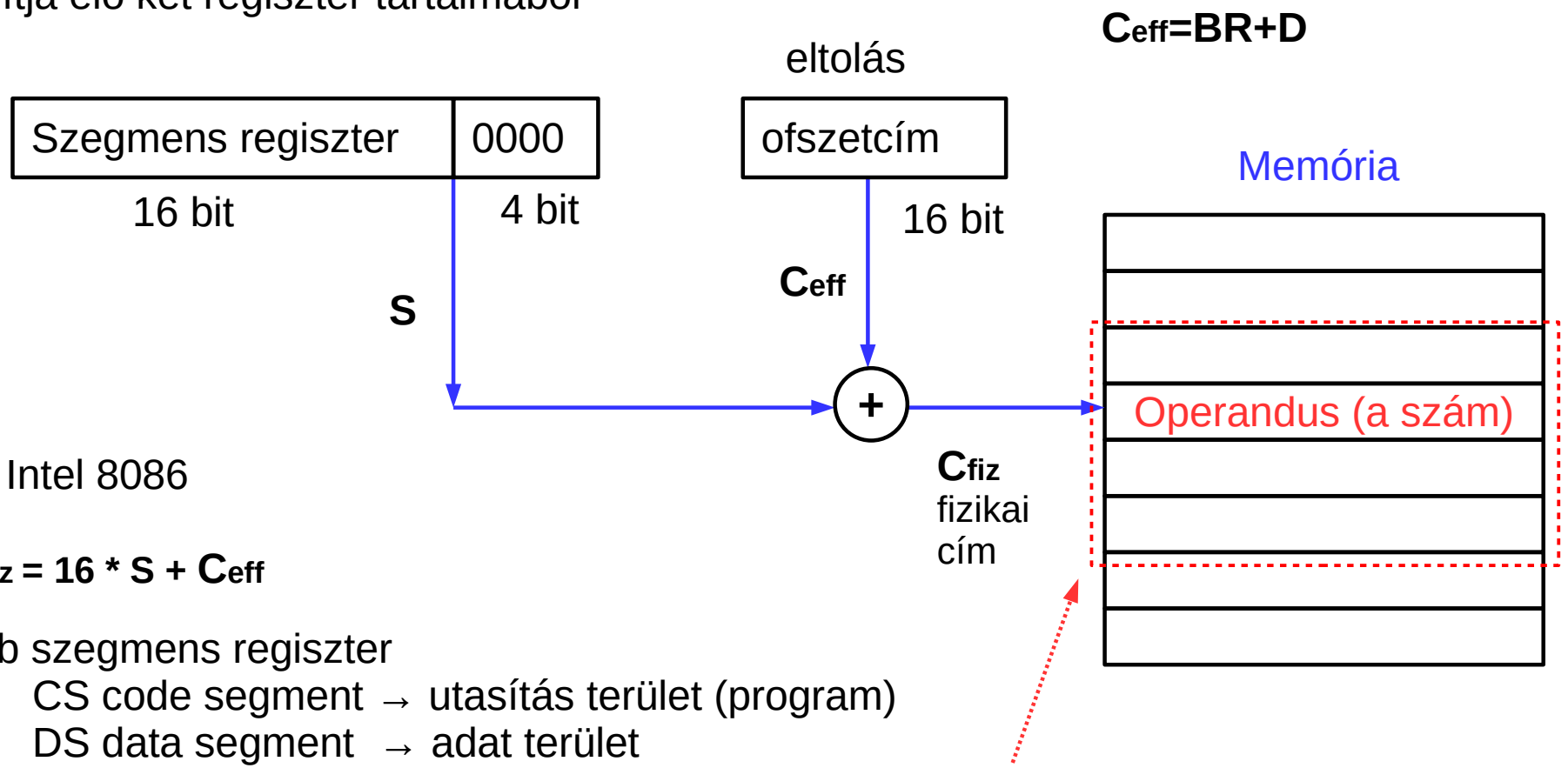
- több címzési mód együttes használata
- általában relatív + indexelt pl. bázisrelatív-indexelt → **C_{eff}=BR+D+IX**



16.7. Címzési módok

9. Szegmentált címzés

Nagy a címtér → nem tudjuk elérni a teljes címtartományt bázis relatív vagy indirekt címzéssel sem (vagy túl hosszú lenne a cím) → a címet aritmetika állítja elő két regiszter tartalmából



pl. Intel 8086

$C_{fiz} = 16 * S + C_{eff}$

4db szegmens regiszter

- CS code segment → utasítás terület (program)
- DS data segment → adat terület
- ES extra segment → extra adat terület
- SS stack segment → veremtár

Szegmens mérete 64kB, ha az ofszetcím 16 bites