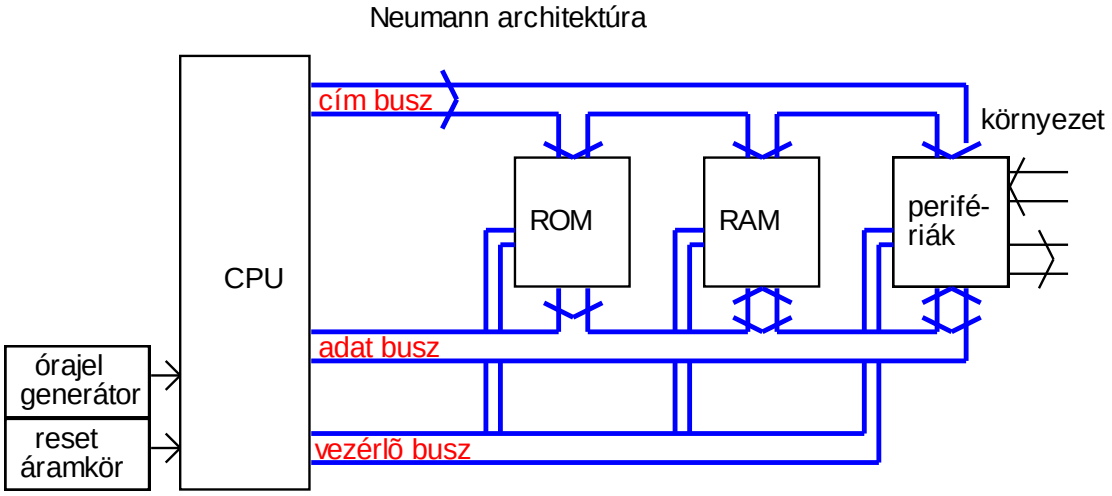
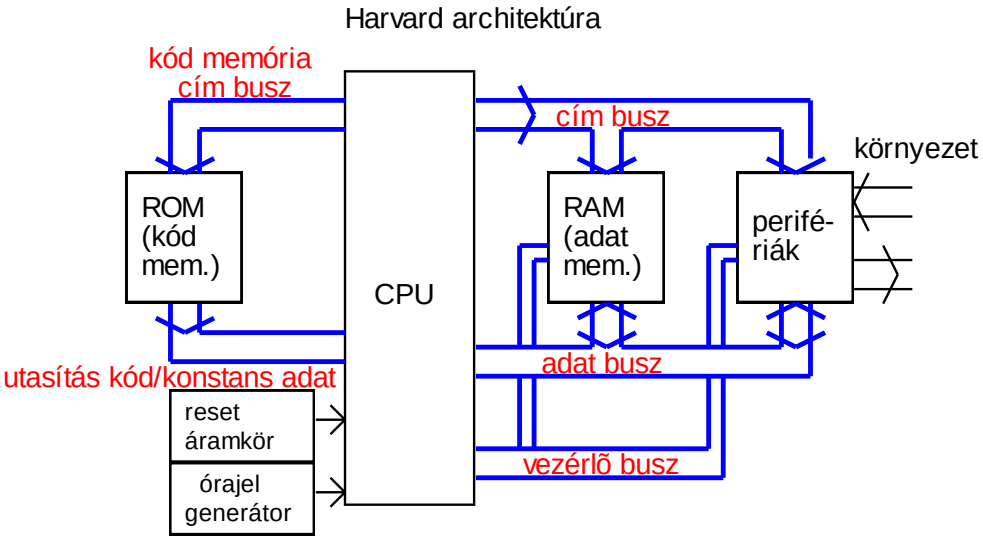
# *Egyszerű számítógép működése*

Egy Neumann és egy Harvard arcitektúrájú számítógép egyszerűsített blokkvázlatát mutatják az alábbi ábrák.



A blokkvázlaton a 4 fő egység és az azokat összekötő jelcsoportok (buszok) funkciója a következő:

- A **CPU** (Central Processing Unit) a fő egység, a többi ennek a működéséhez szükséges alegység.
- A kód memóriában találhatók az CPU által végrehajtamdó utasítások kódjai.
- Az adat memóriában tárolja a CPU a változókat. (Többnyire ennek egy részében van a szubrutinokhoz és interrupthoz szükséges STACK memória terület is. Azonban kisebb CPU-k esetén, azt önálló hardver STACK valósítja meg.)
- A perifériák teszik lehetővé, hogy a CPU a környezettel kommunikáljon, onnan adatokat kapjon és oda adatokat küldjön.
- A buszok teszik lehetővé az egységek közötti kommunikációt. A buszt vezérlő egységet busz masternek nevezik. Elsősorban a CPU a buszmaster (de speciális periféria is lehet, de ezt itt nem részletezzük). A továbbiakban csak a CPU-ra hivatkozunk.
    - A címbusz a rákapcsolódó egységek egy tároló rekeszének megcímzésére (kijelölésére) szolgál.
    - Az adatbuszon küld vagy fogad adatot a CPU a megcímzett egységtől.
    - A vezérlő buszon a CPU kijelöli az adatmozgatás irányát (RD: olvas a megcímzett egységből, WR: ír a megcímzett egységbe) és vezérli ezek időbeli lezajlását. (A vezérlő busznak egyéb, itt nem részletezett funkciójú jelei is vannak.)

- Neumann architektúrájú számítógép esetén a CPU a kódmemóriát és adat memóriát olvasásnál ugyanúgy kezeli, adatmemóriából is képes utasítást olvasni. A memóriákban vegyesen lehetnek utasítás kódok és adatok (konstansok és változók is). Mivel a fenti ábrán a kód memória (ROM) és az adat memória (RAM) ugyanazon a buszon van így ez természetesen teljesül.
- Harvard architektúrájú számítógép esetén a kód memóriához és adat memóriához külön cím és adatbusz tartozik. Ezért ebben az esetben a CPU csak az kódmemóriából képes utasítást olvasni, az adatmemóriában csak adatok lehetnek, utasítások nem.
    - A kódmemória oldali címbuszon adja ki a CPU a következő utasítás (utasításkód) címét. A következő utasítás címét mindig a **PC** (Program Counter) mutatja, ez a CPU része. Harvard architektúra esetén tehát ez a külön kódmemória címbusz.
    - Az kódmemória oldali adat buszon kapja meg a CPU a megcímzett utasítás kódját. Harvard architektúra esetén tehát ez a külön kódmemória adatbusz. Azonban a ***módosított Harvard architektúra*** esetén a kódmemóriából konstans adatot is képes olvasni (erre külön utasítása van).
    - Az adatmemória oldali címbuszon címzi meg (jelöli ki) a CPU azt a RAM vagy periféria rekeszt, amelyet olvasni vagy írni fog. Harvard architektúra esetén tehát ez egy külön adatmemória címbusz.
    - Az adatmemória oldali adat buszon olvassa vagy írja a CPU az adatot a memóriába vagy perifériába. Harvard architektúra esetén tehát ez a külön adatmemória adatbusz.
    - A Harvard felépítés előnye az, hogy miközben az aktuális utasítás az adatmemóriát használja, az utasítás memóriából elő tudja venni a következő utasítás kódját. Ez a párhuzamosítás gyorsabb működést eredményez.
- A reset áramkör a CPU és a perifériák alaphelyzetbe hozását végzi a bekapcsoláskor (és esetleg nyomógomb megnyomására)
- Az órajel generátor adja az órajelet a CPU-nak és többnyire a perifériáknak is.

Az utasítás végrehajtása 3 fázisra bontható. Ezt a CPU-ban levő vezérlő (szinkron sorrendi hálózatként megvalósított állapotgép) valósítja meg, a processzor többi részének megfelelő vezérlőjeleket adva.

- ***FETCH***: Az ***utasítás beolvasása*** az utasítás regiszterbe. A FETCH ciklus alatt a vezérlő engedélyezi az utasítás regiszterbe írást. Maga az írás a vezérlő FETCH állapota alatti órajelre történik meg. Ugyanerre az órajelre a vezérlő átlép a DECODE állapotba és a PC-t (a kódmemóriát címző számlálót) növeli, hogy az következő utasítás várható címére mutasson. (Az ugró és szubrutin hívó utasítások ezt felülírják az EXECUTE ciklusban.)

- ***DECODE***: Az ***utasítás dekódolása***. Annak eldöntése, hogy melyik utasításról van szó. Ezt az utasítás kód néhány bitje kódolja. Ezektől függően a vezérlő a DECODE állapot alatt jövő órajelre az aktuális utasítás végrehajtásához szükséges vezérlő jeleket előállító *valamelyik* EXECUTE állapotba lép, a sok közül (ezt itt EXECUTE i-vel jelöljük). Ilyenből annyi van, ahány egymástól eltérő (a vezérlő által előállítandó) vezérlő jelet igénylő utasítás van. (A vezérlő jelek egy része közvetlenül az utasításregiszterből jön. Pl. az operandusok címe.)

- ***EXECUTE***: Az ***utasítás végrehajtása***. A vezérlő az EXECUTE i állapotban kiadja az aktuális utasítás végrehajtásához szükséges vezérlőjeleket. Az utasítás végrehjatása az EXECUTE i alatt érkező órajelre történik meg. Tehát amikor a vezérlő az EXECUTE i állapotból átlép a következő állapotba. (Ez általában a következő utasítást beolvasó FETCH állapot, ez alól kivétel, ha az utasítás alatt a vezérlő interruptot észlel.)

# A STACK memória

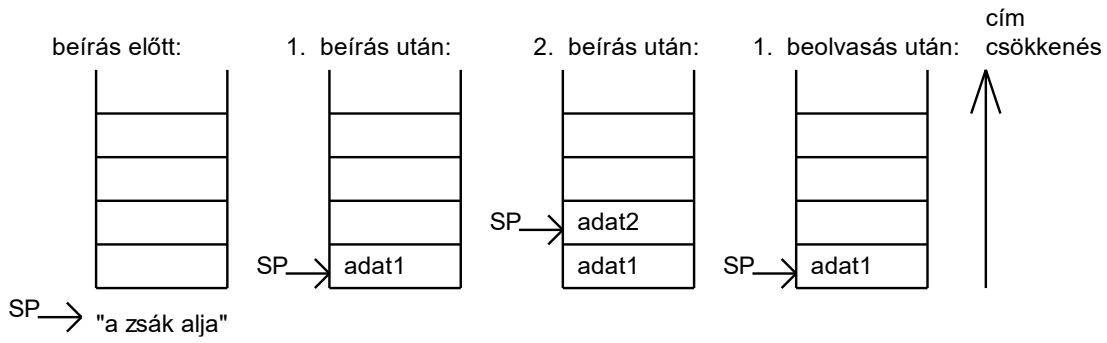
Az SP szerepe és verem tár (stack, zsák memória, LIFO) kezelése

- Az ún. stack memória egyes utasítások és az interrupt működéséhez szükséges.

- A stack a RAM memóriának egy speciálisan kezelt része vagy önálló hardwer egység. A kezelésének lényege, hogy ***az adatokhoz a beírással ellentétes sorrendben lehet csak hozzáférni*** (**L**ast **I**n **F**irst **O**ut LIFO). A stack-et a használójának nem kell címezni, az automatikus.

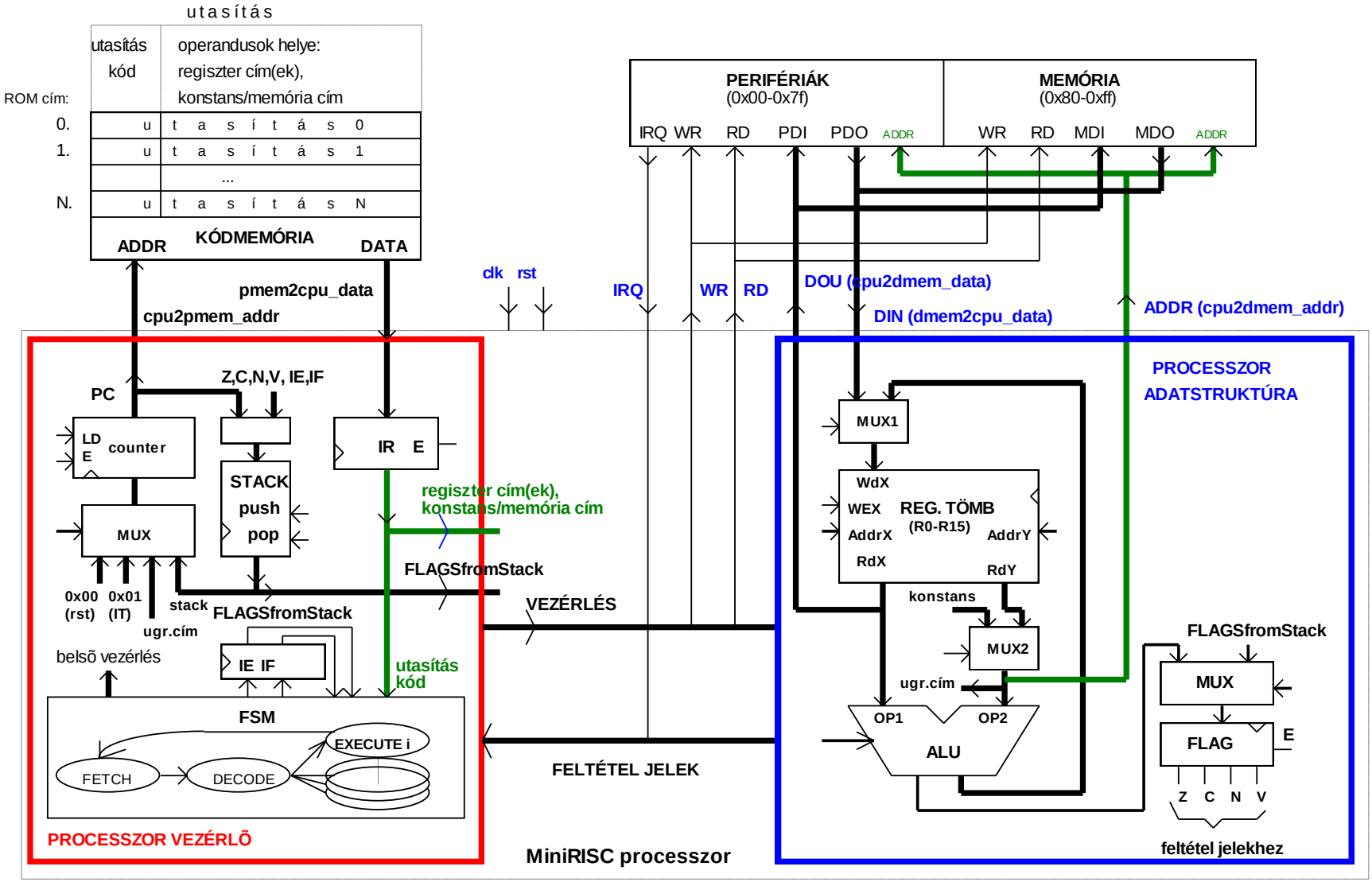
- A beírás ill. kiolvasás helyét (címét) a stack pointer (**SP**) tartja nyilván.

- A beírás előtt az SP általában a következő üres helyre mutat és oda íródik be az adat. Olvasáskor ahova az SP éppen mutat, onnan olvas, majd az olvasás után rögtön a következő (utoljára beírt) adatra mutat.

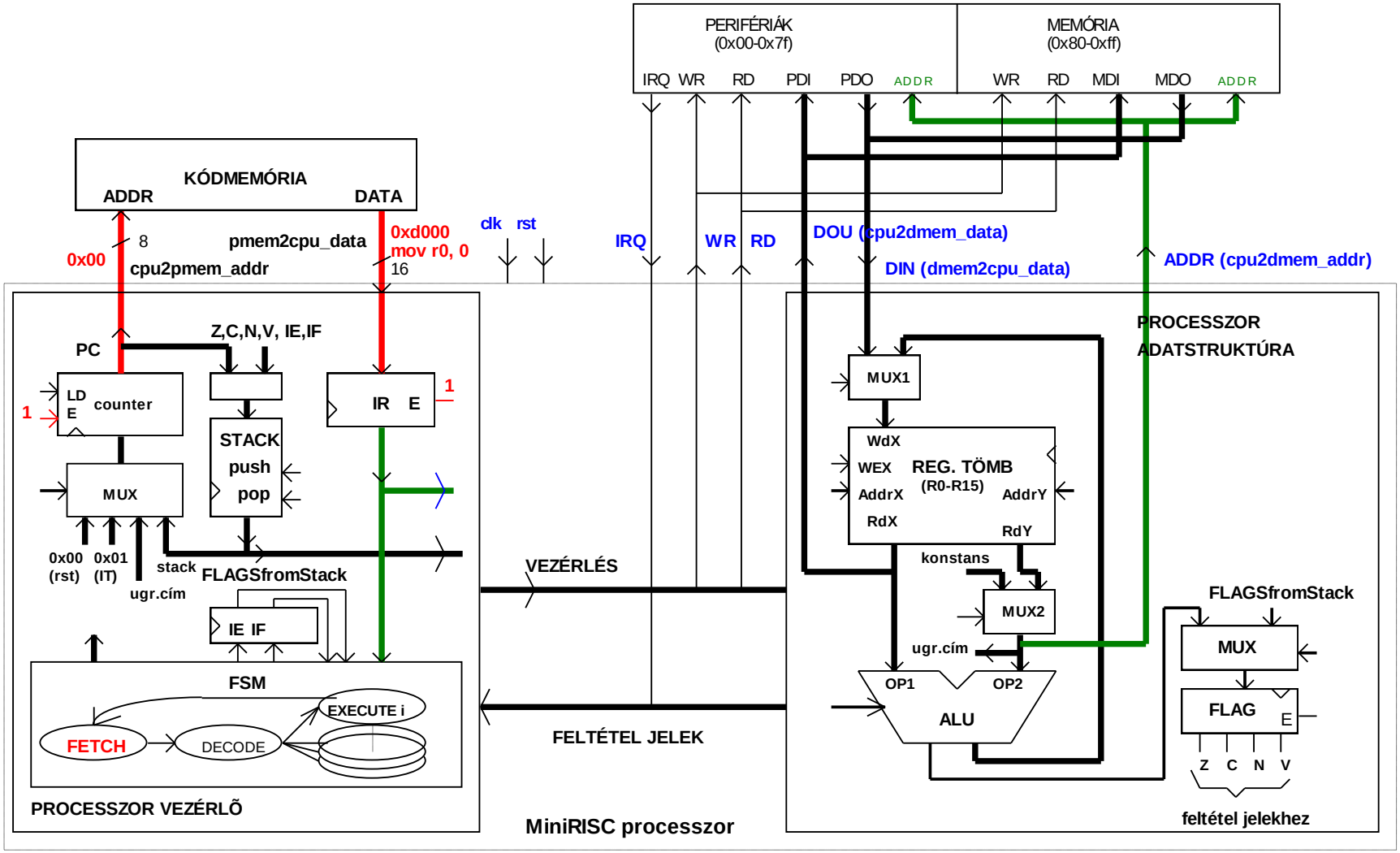


**A MiniRISC processzor belső felépítésének blokkvázlata**

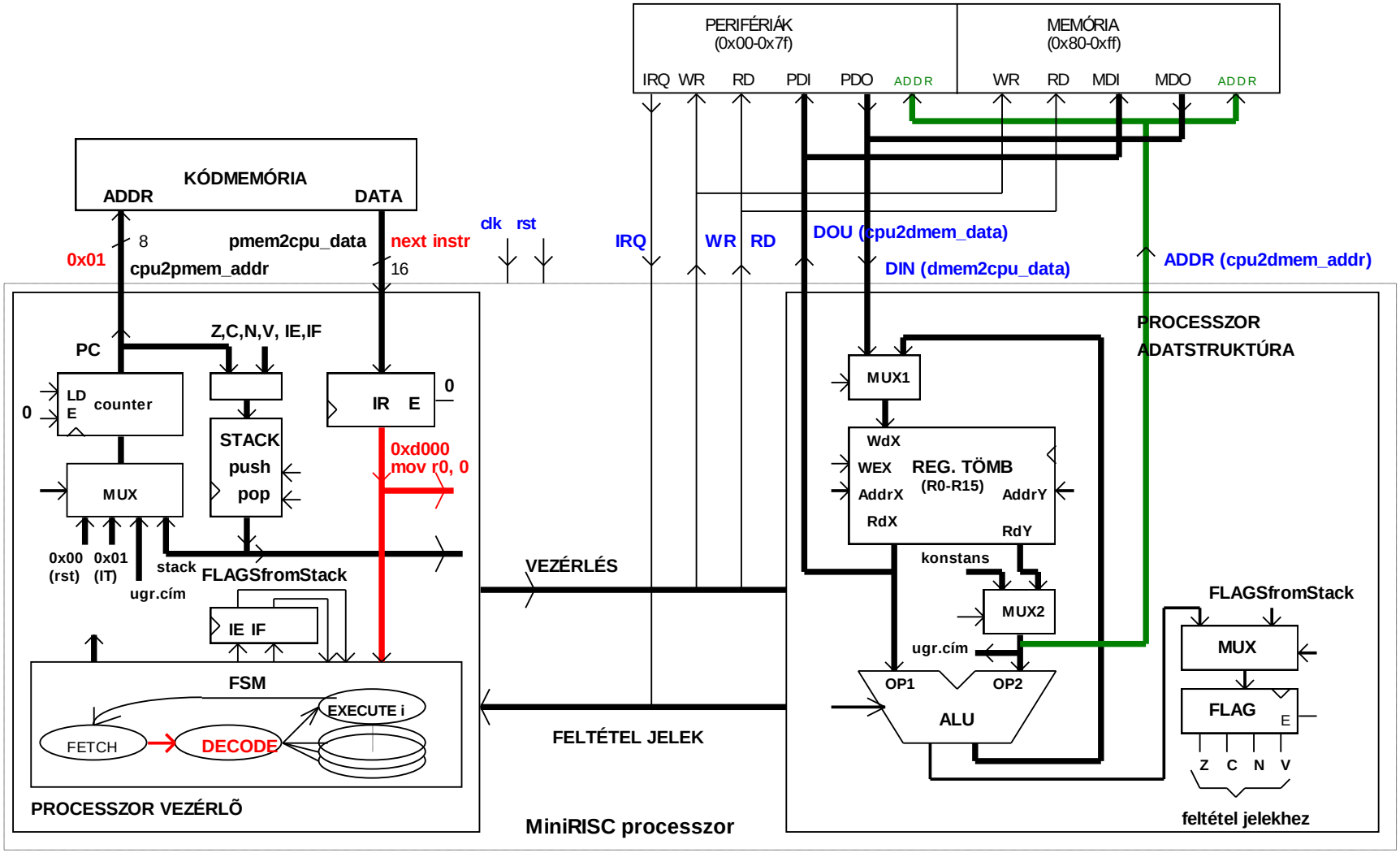
A szaggatott vonallal körülhatárolt rész a MiniRISC processzor (CPU), a többi a kódmemória, adat memória és perifériák.



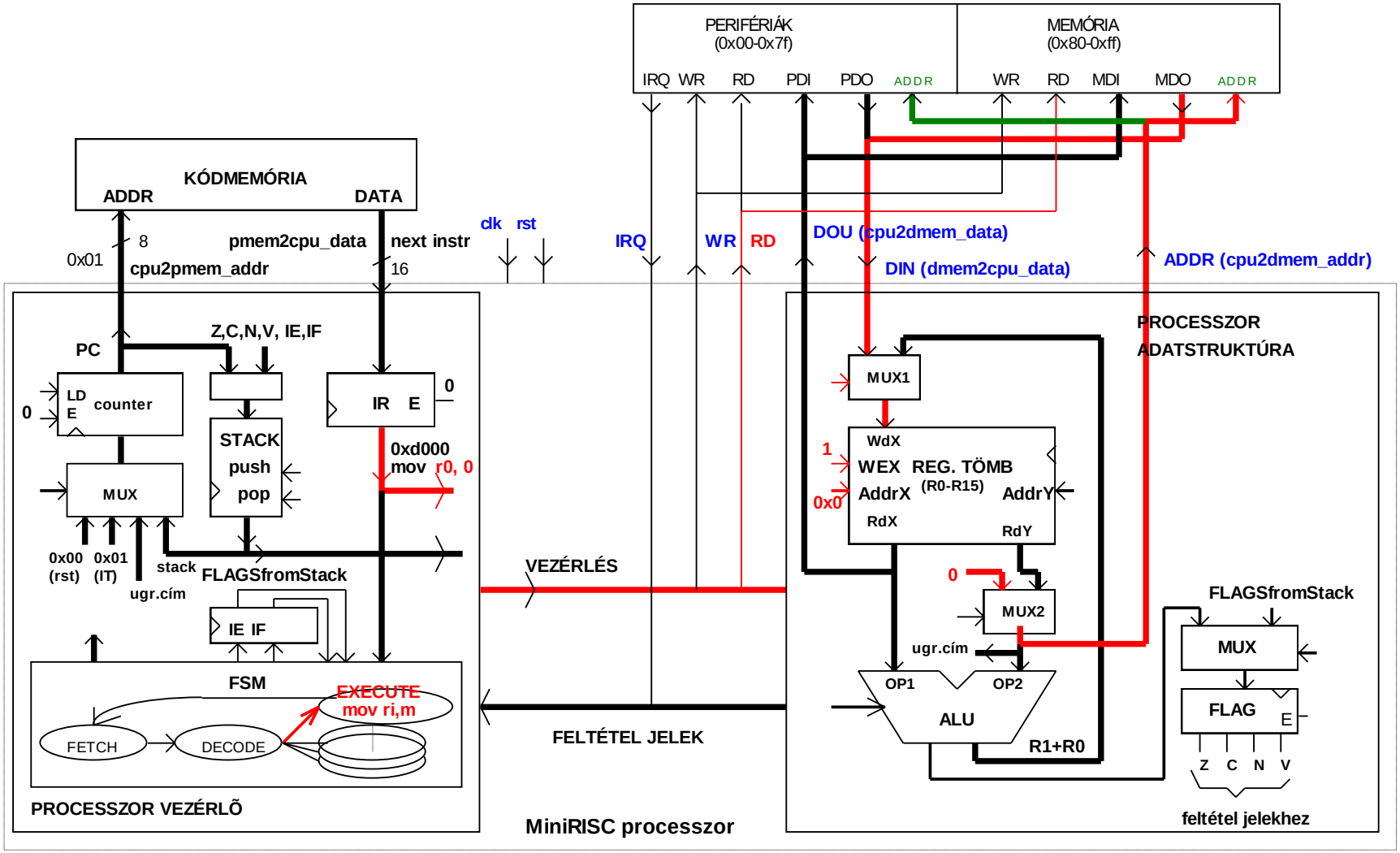
Adatmozgató utasítás, olvasás a memóriából: **FETCH** (uatsítás kód olvasás) ciklus



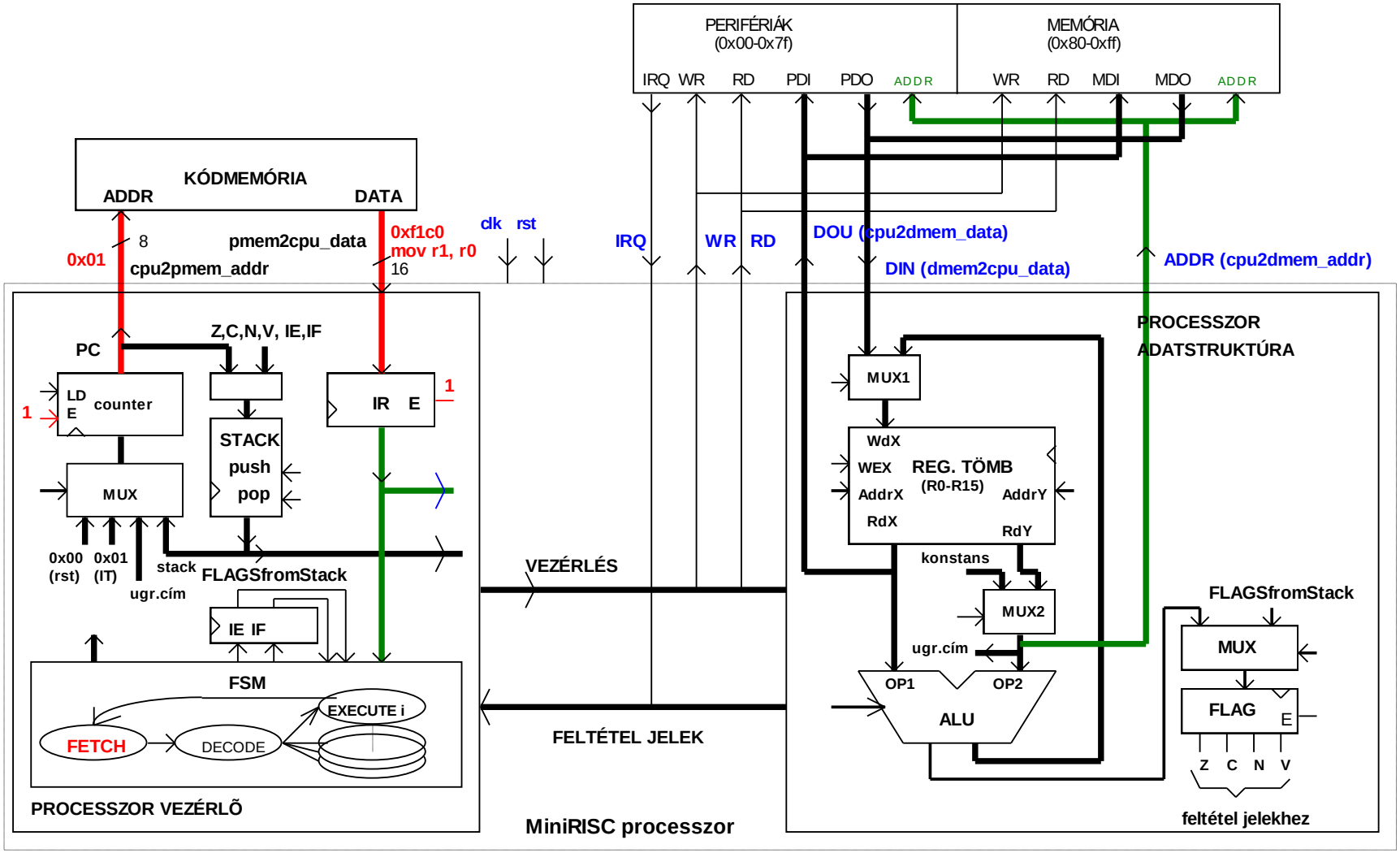
Adatmozgató utasítás, olvasás a memóriából: **DECODE** (utasítás dekódolás) ciklus



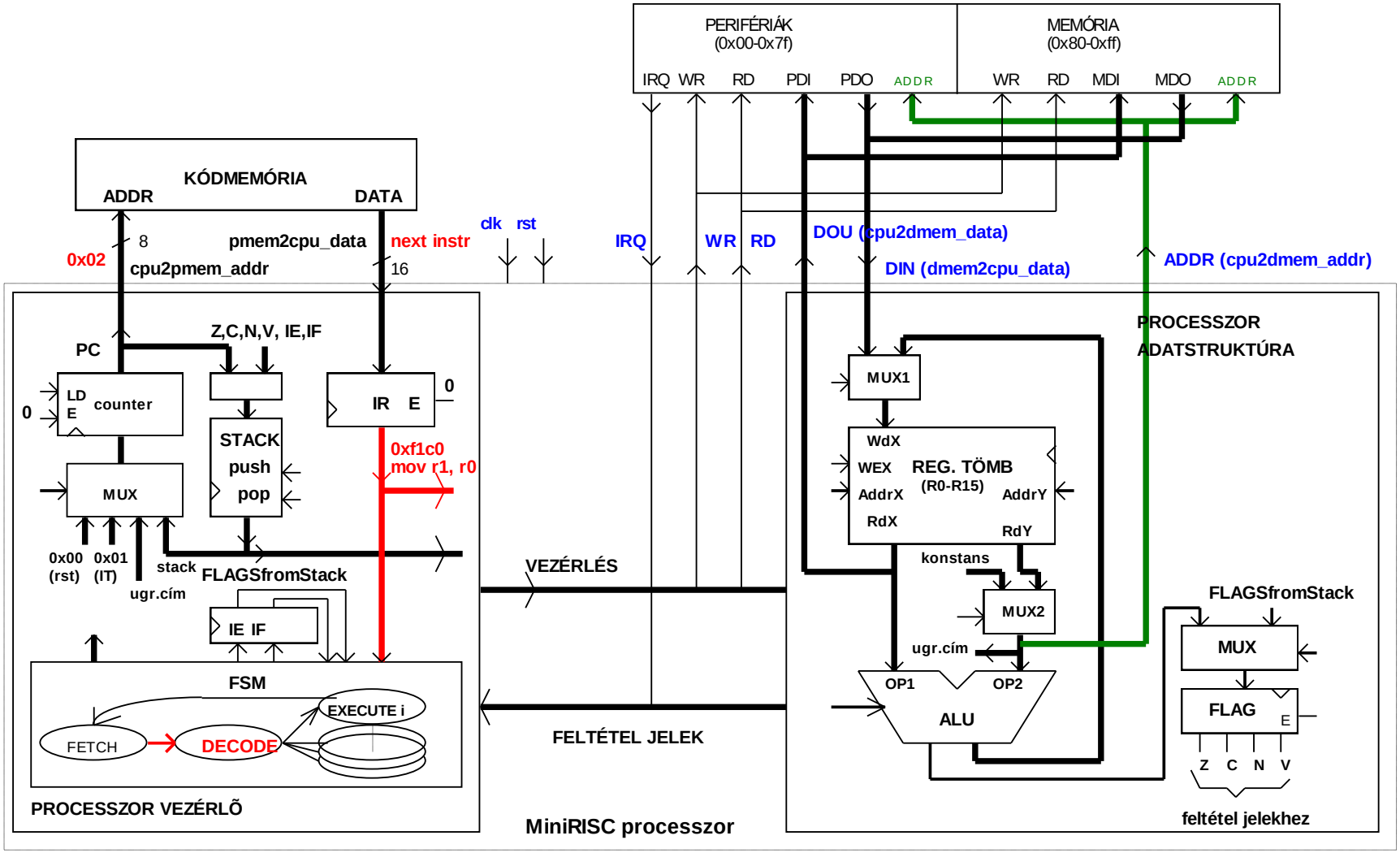
Adatmozgató utasítás, olvasás a memóriából: **EXECUTE** (utasítás végrehajtás) ciklus



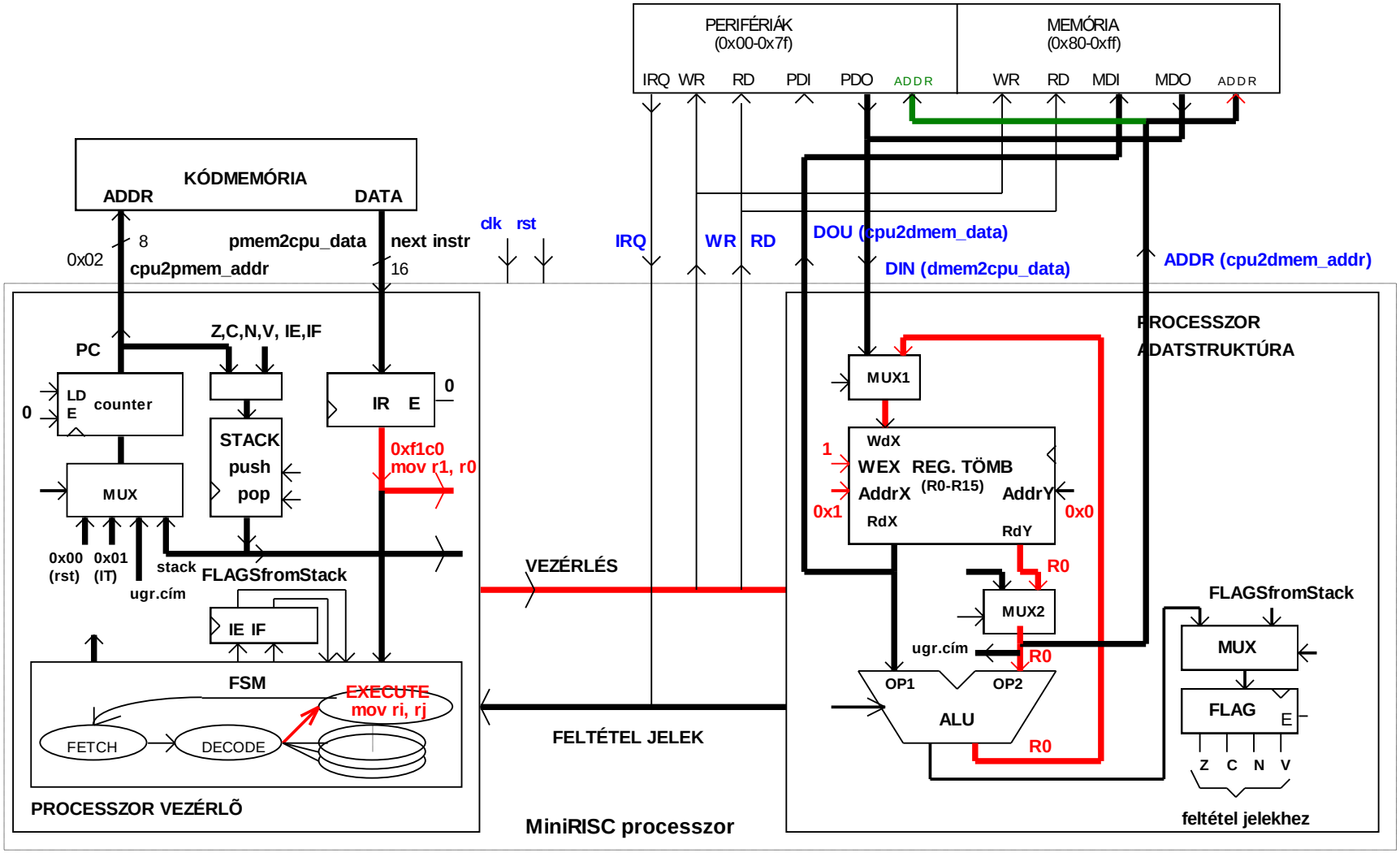
Adatmozgató utasítás, regiszterek között: **FETCH** (utasítás beolvasás) ciklus



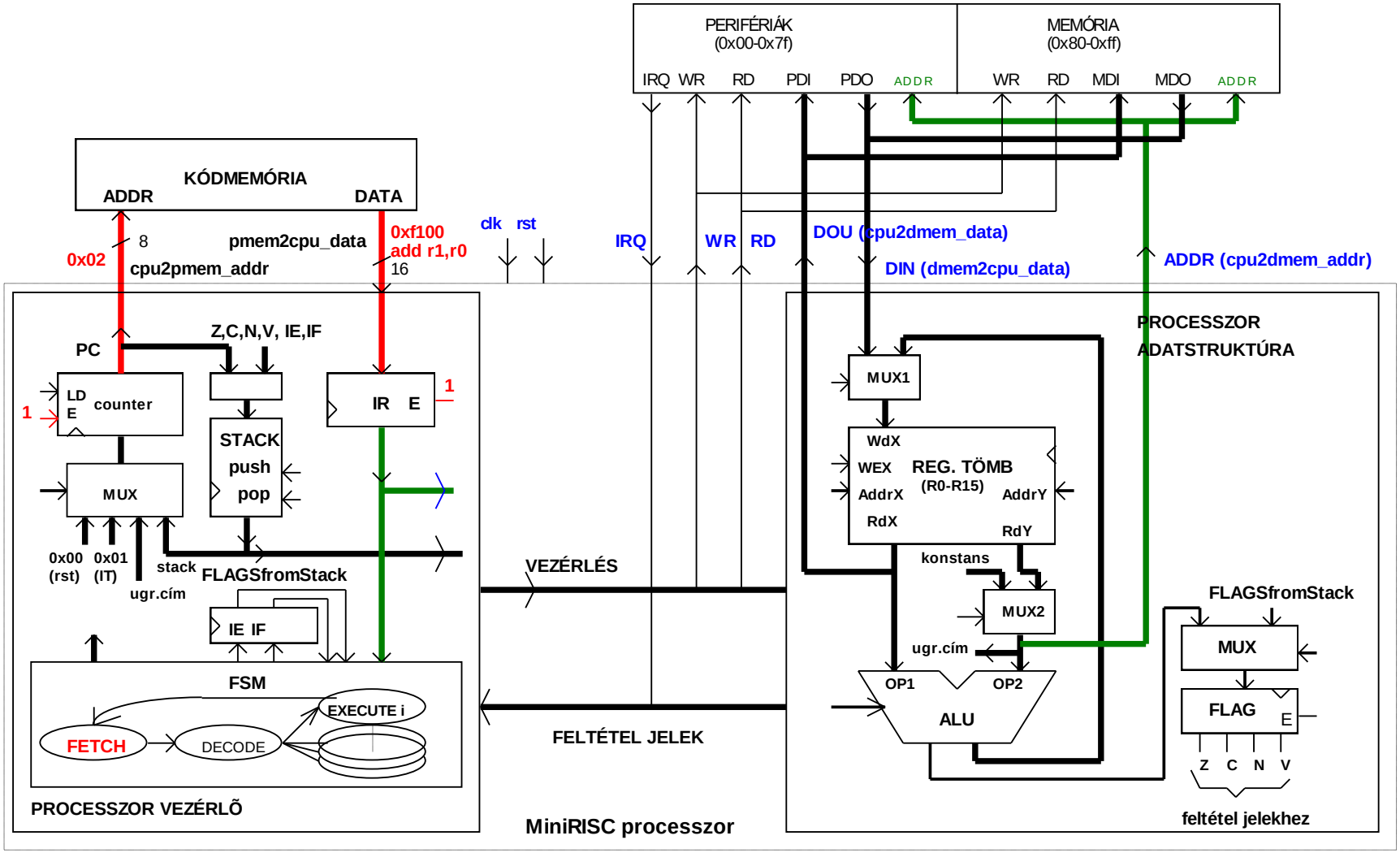
Adatmozgató utasítás, regiszterek között: **DECODE** (utasítás dekódolás) ciklus



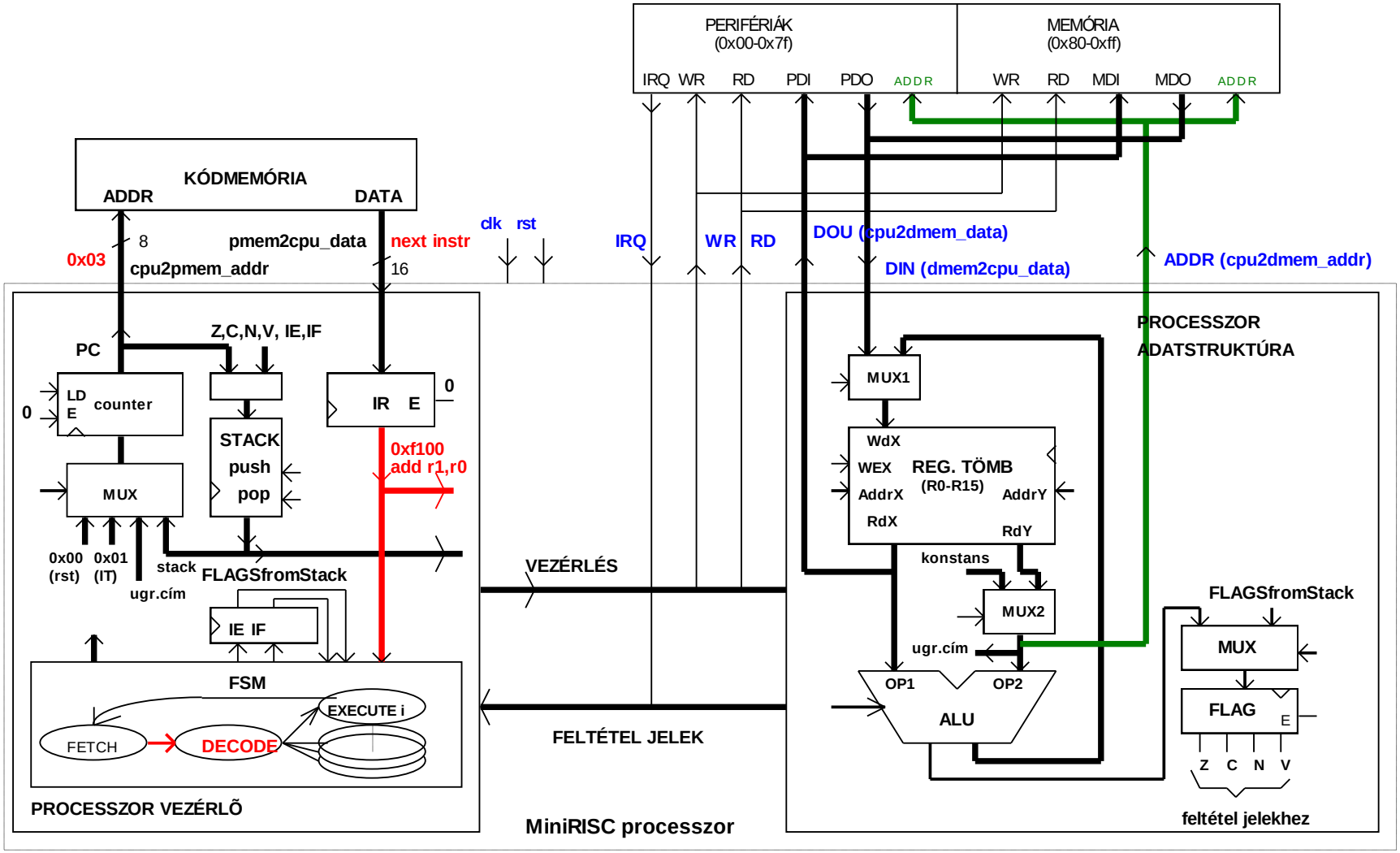
Adatmozgató utasítás, regiszterek között: **EXECUTE** (utasítás végrehajtás) ciklus



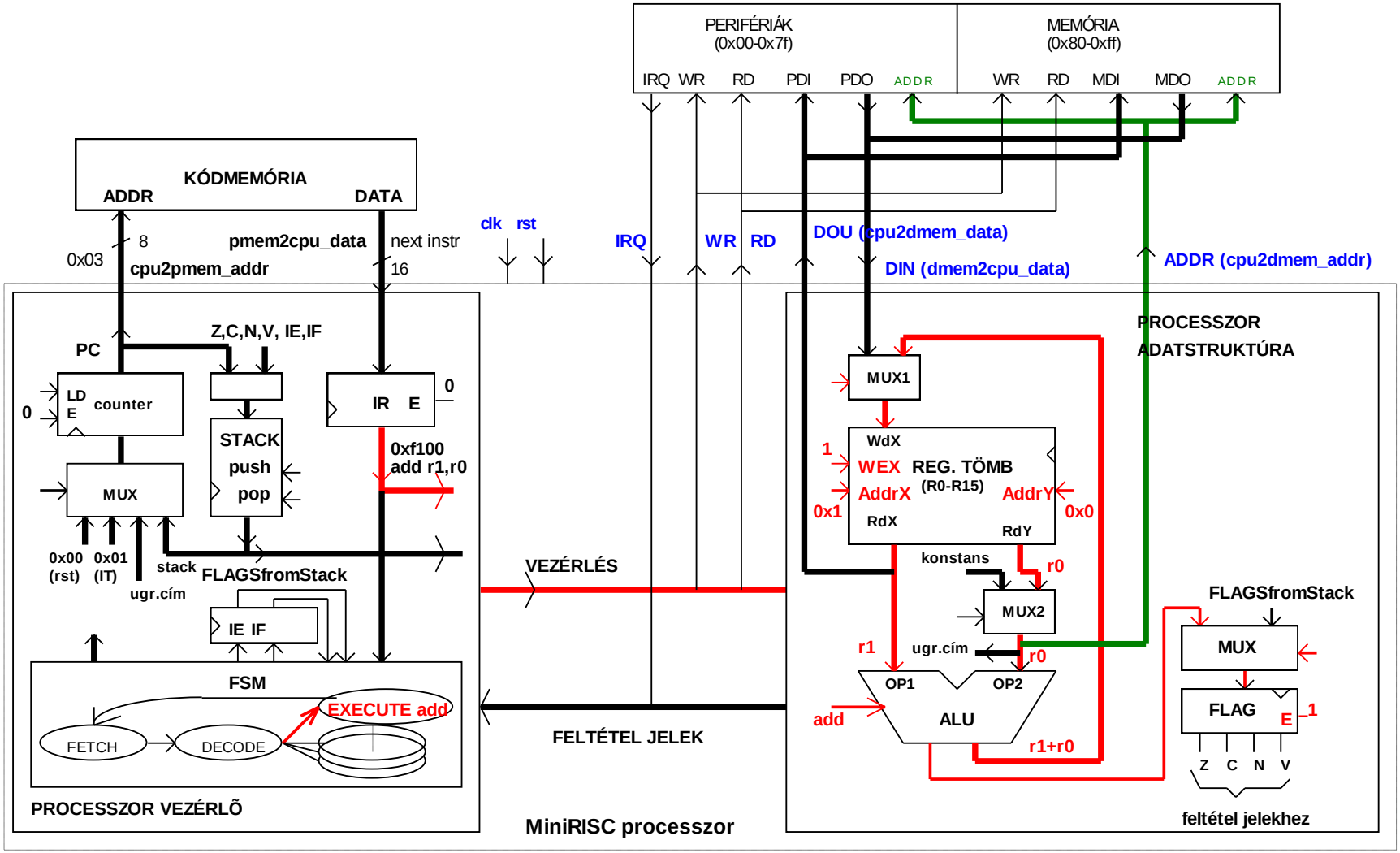
művelet végző utasítás (aritmetikai) összeadás: **FETCH** (utasítás beolvasás) ciklus



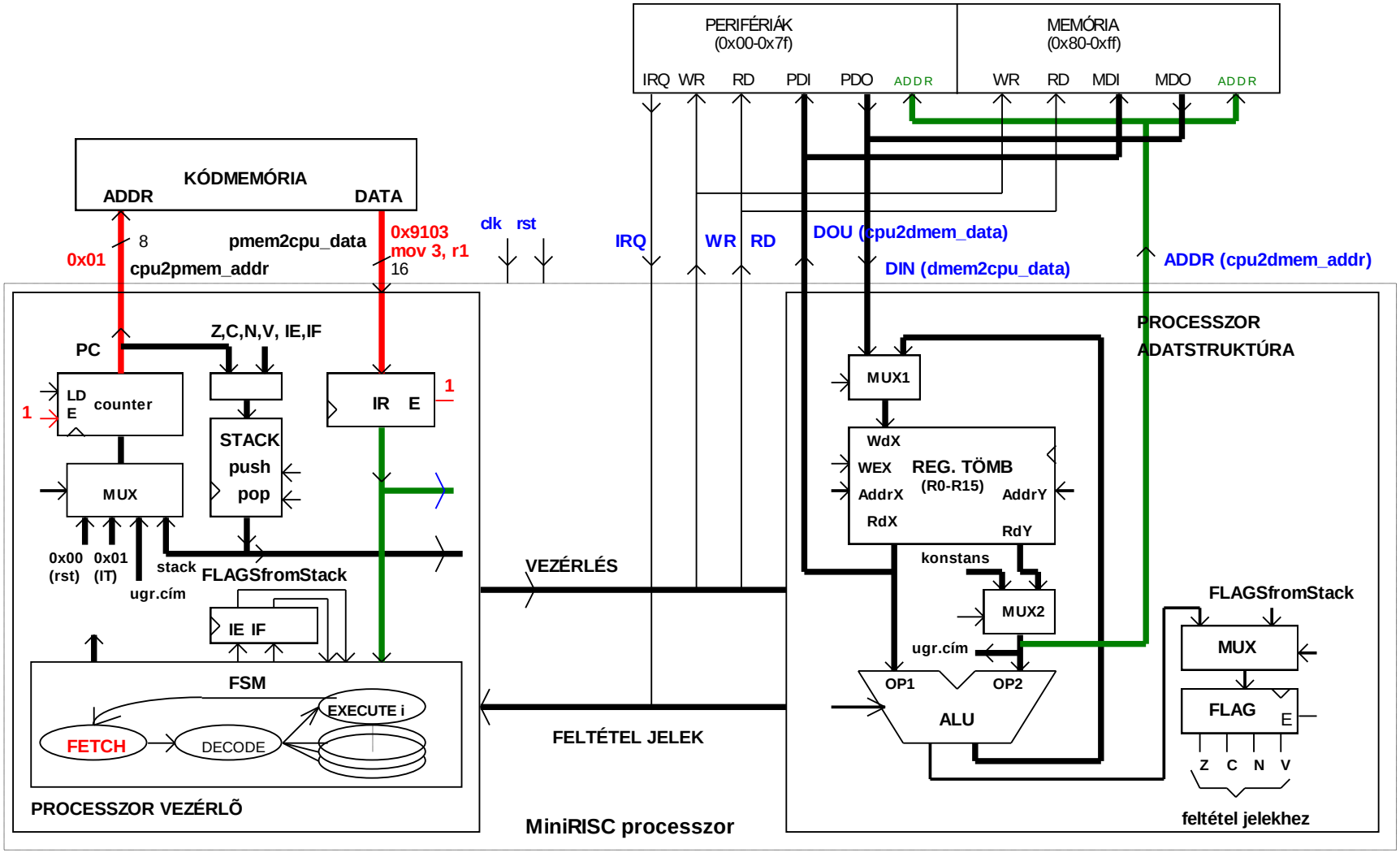
művelet végző utasítás (aritmetikai) összeadás: **DECODE** (utasítás dekódolás) ciklus



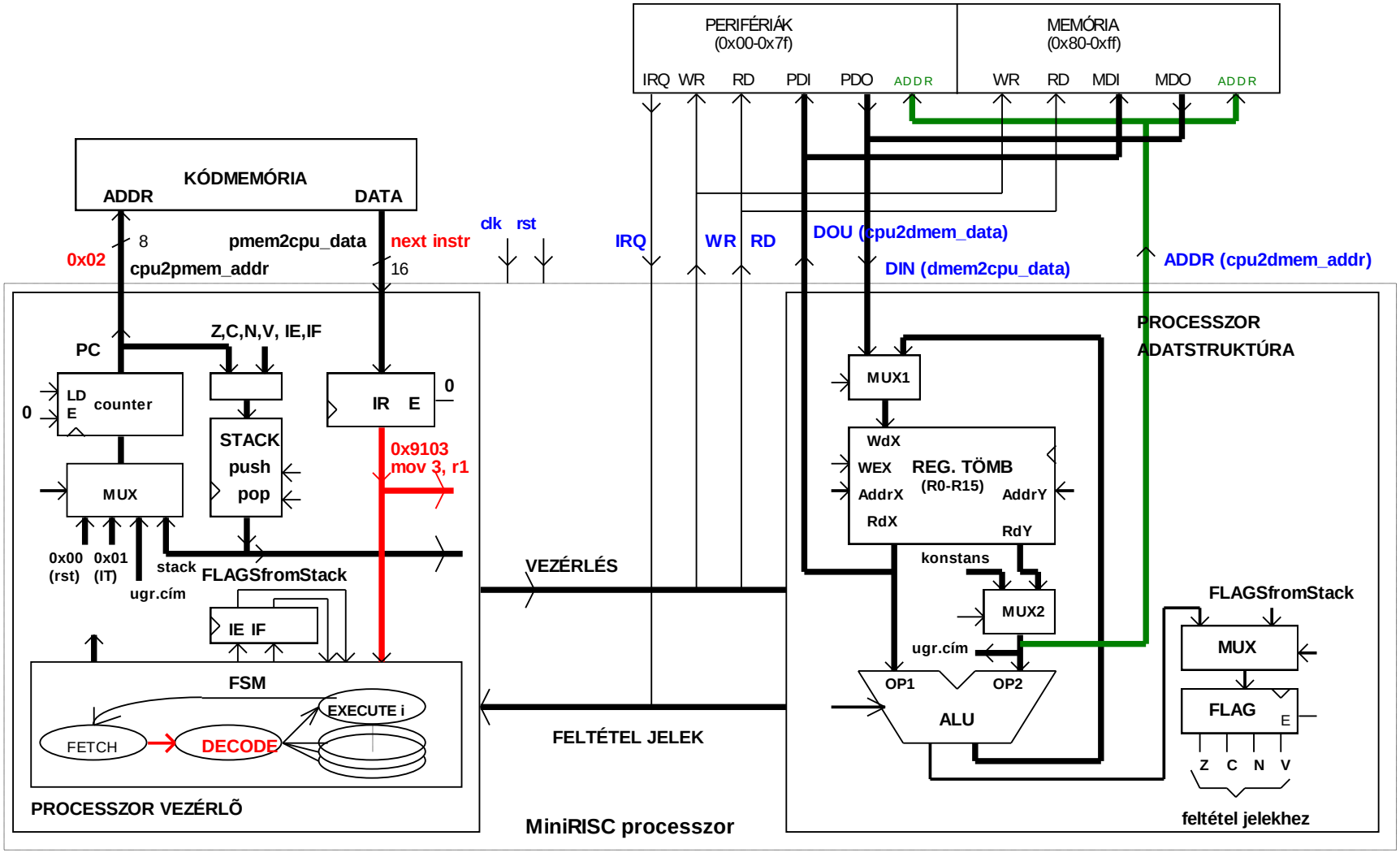
művelet végző utasítás (aritmetikai) összeadás: **EXECUTE** (utasítás végrehajtás) ciklus



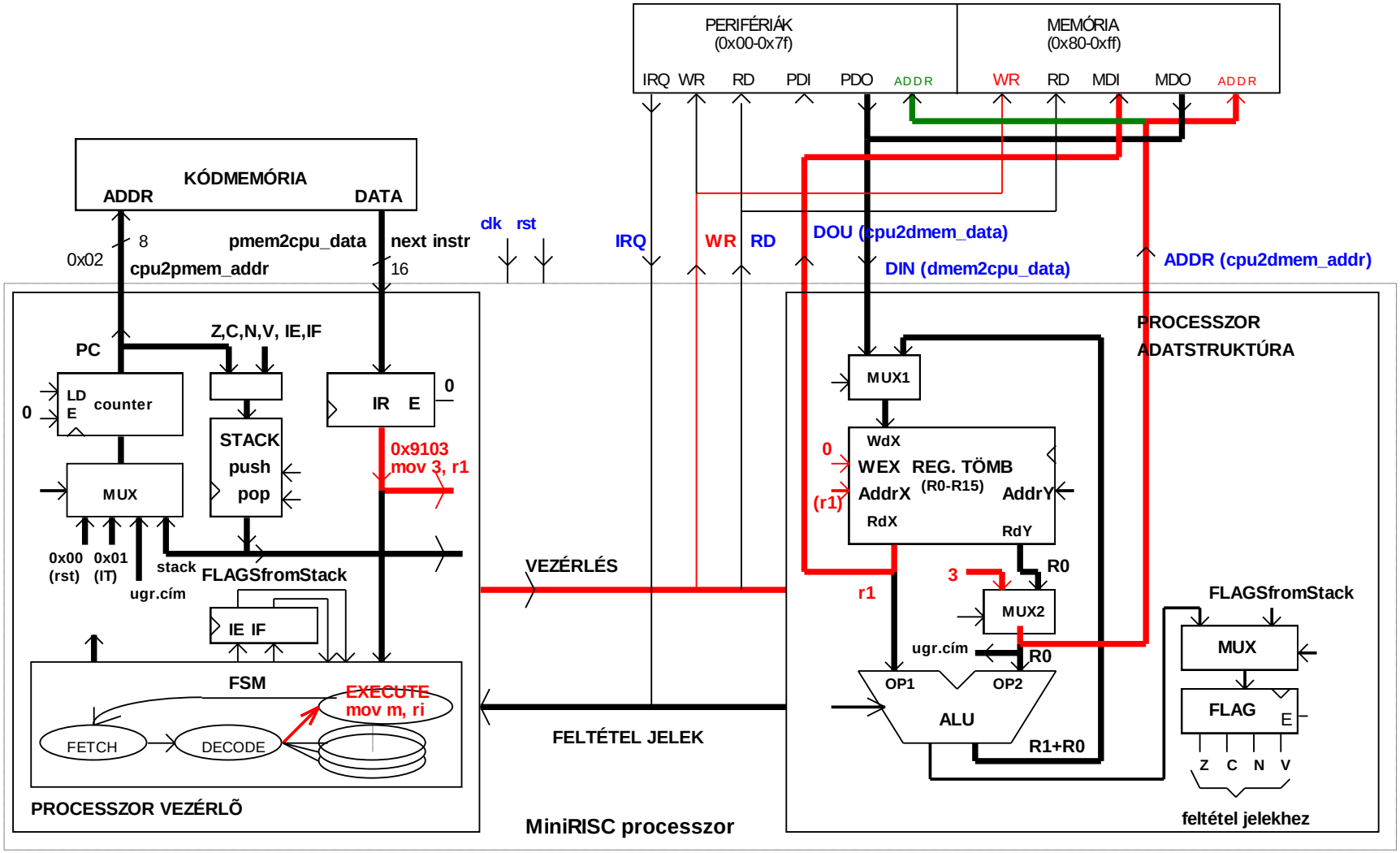
Adatmozgató utasítás, írás a memóriába: **FETCH** (utasítás beolvasás) ciklus



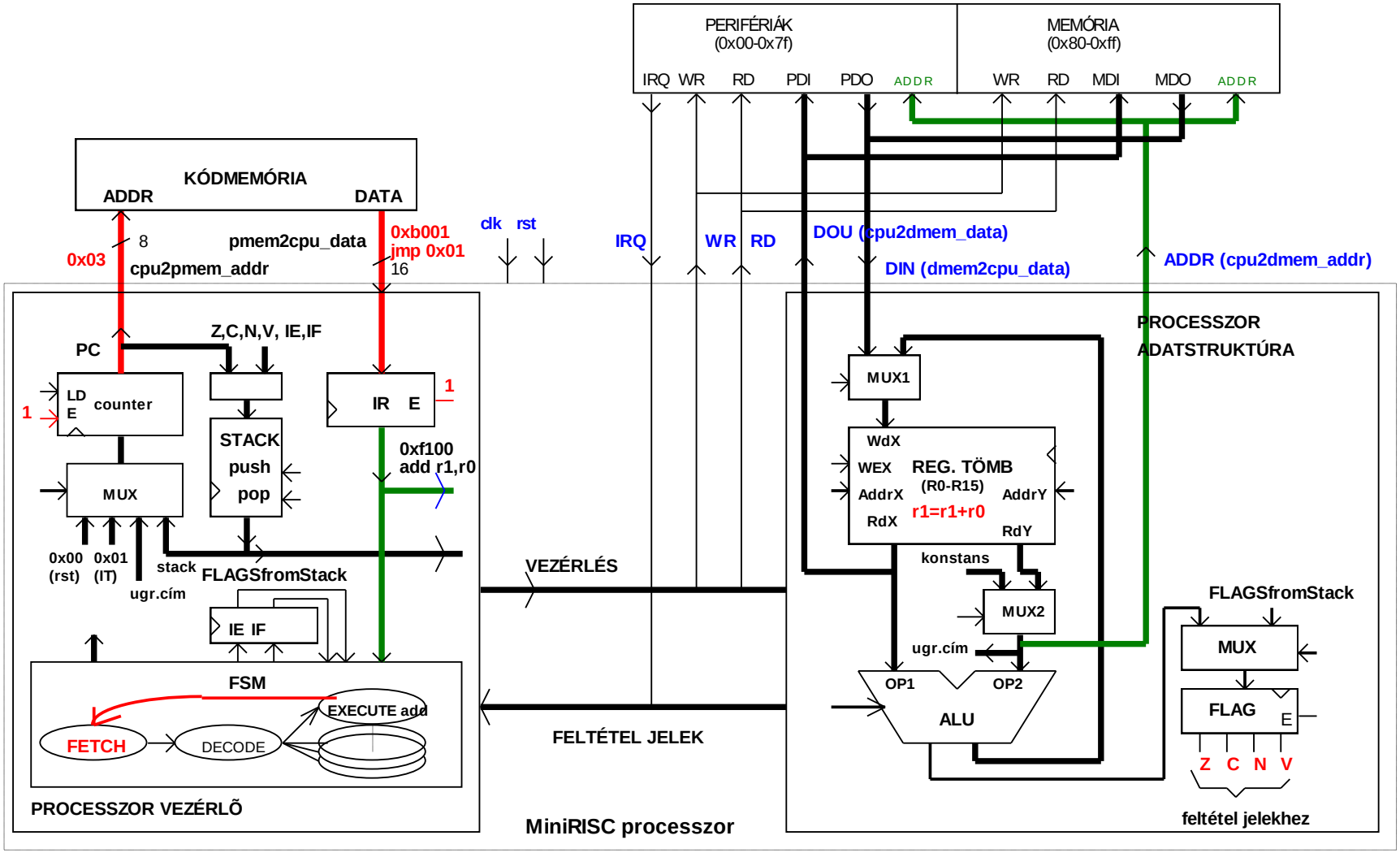
Adatmozgató utasítás, írás a memóriába: **DECODE** (utasítás dekódolás) ciklus



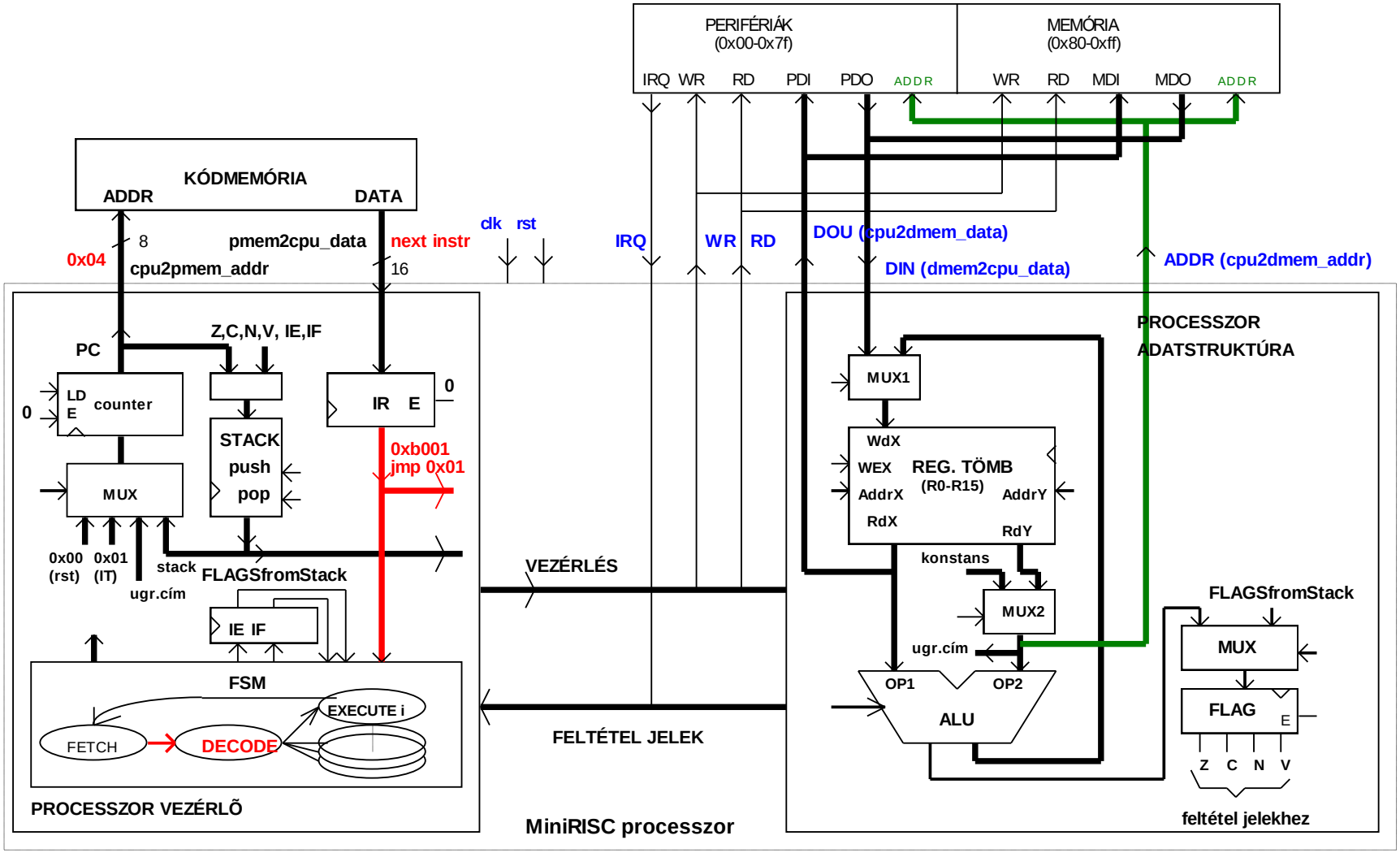
Adatmozgató utasítás, írás a memóriába: **EXECUTE** (utasítás végrehajtás) ciklus



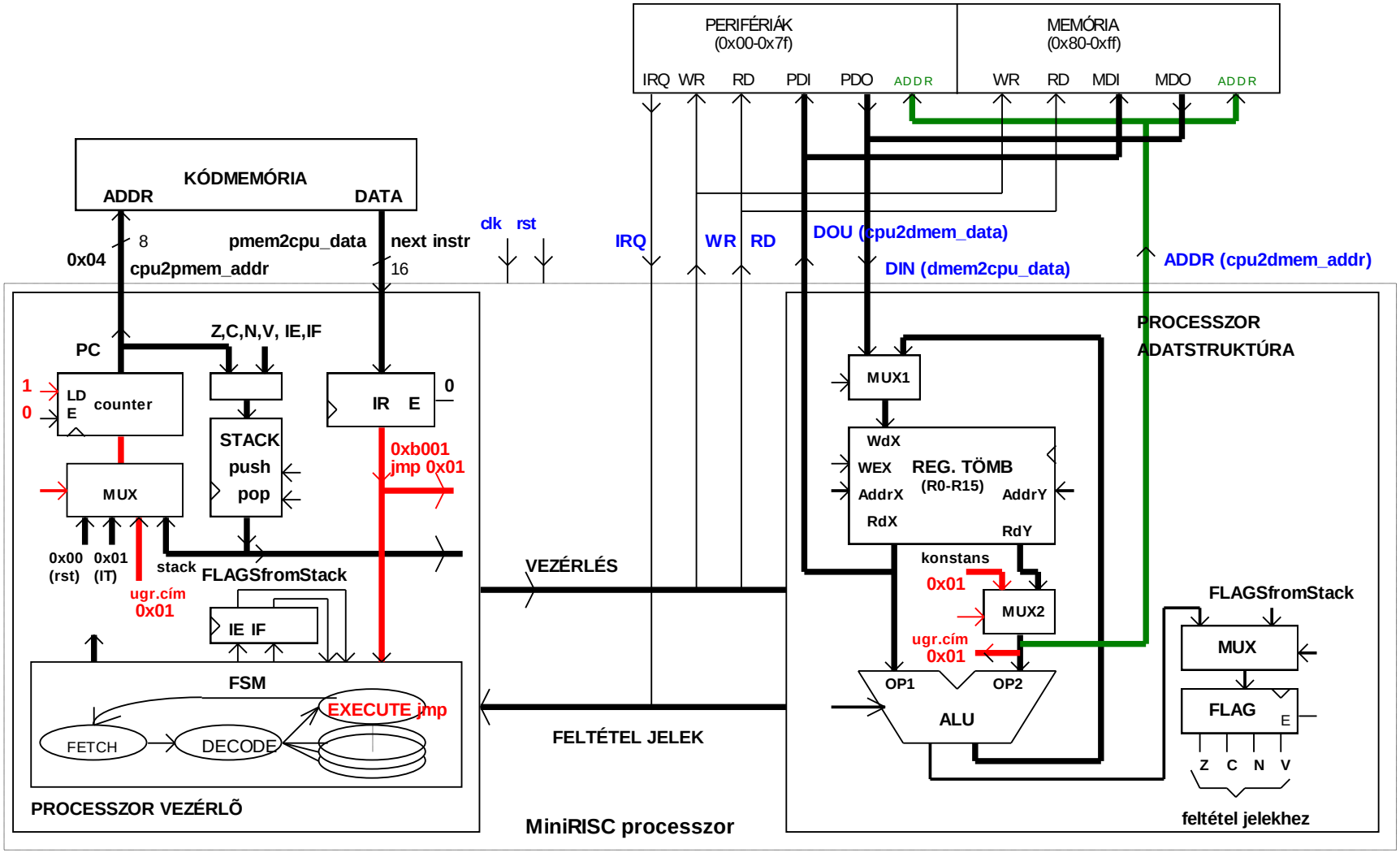
vezérlés átadó utasítás (ugró), direkt ugrás: **FETCH** (utasítás beolvasás) ciklus



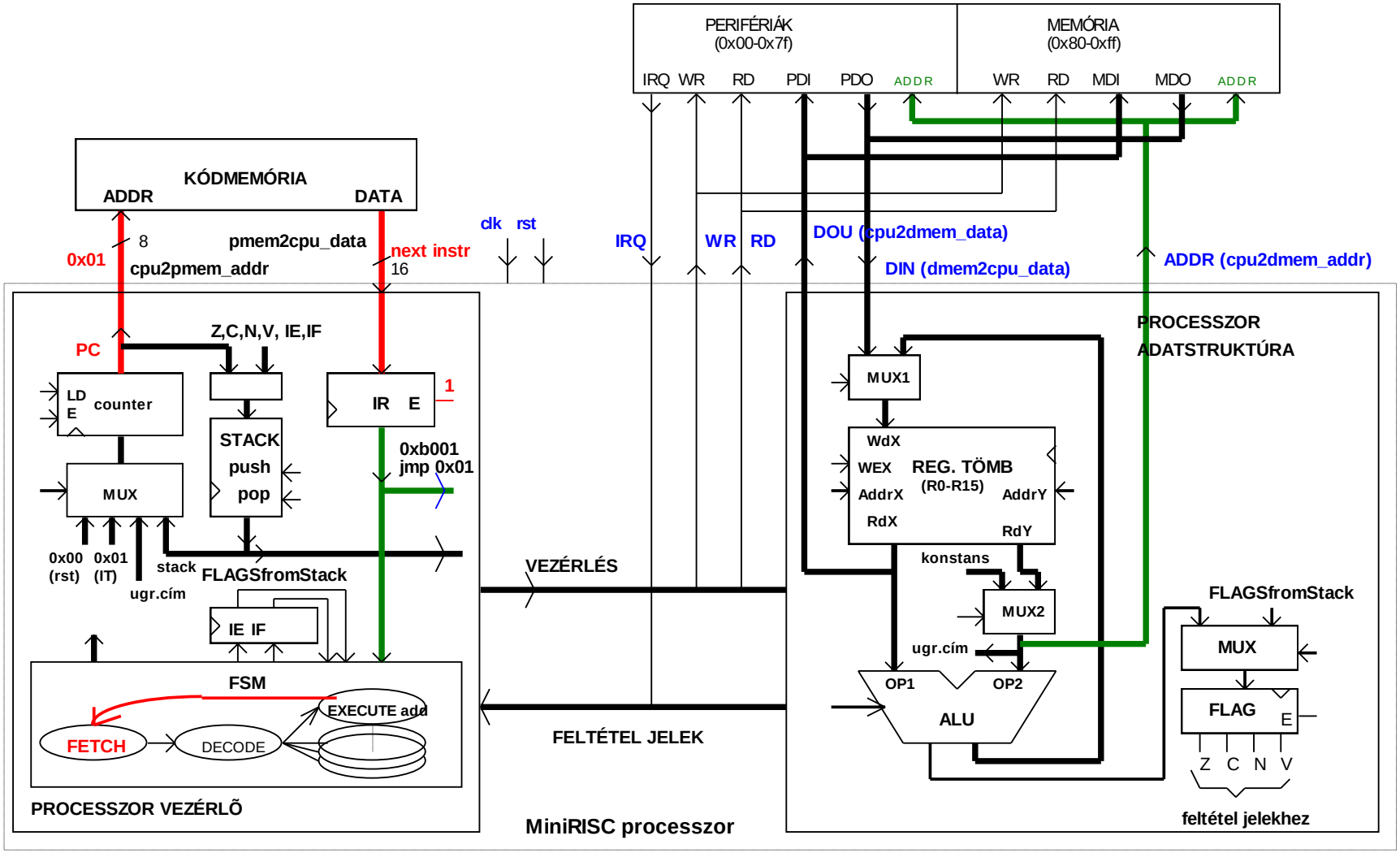
vezérlés átadó utasítás (ugró), direkt ugrás: **DECODE** (utasítás dekódolás) ciklus



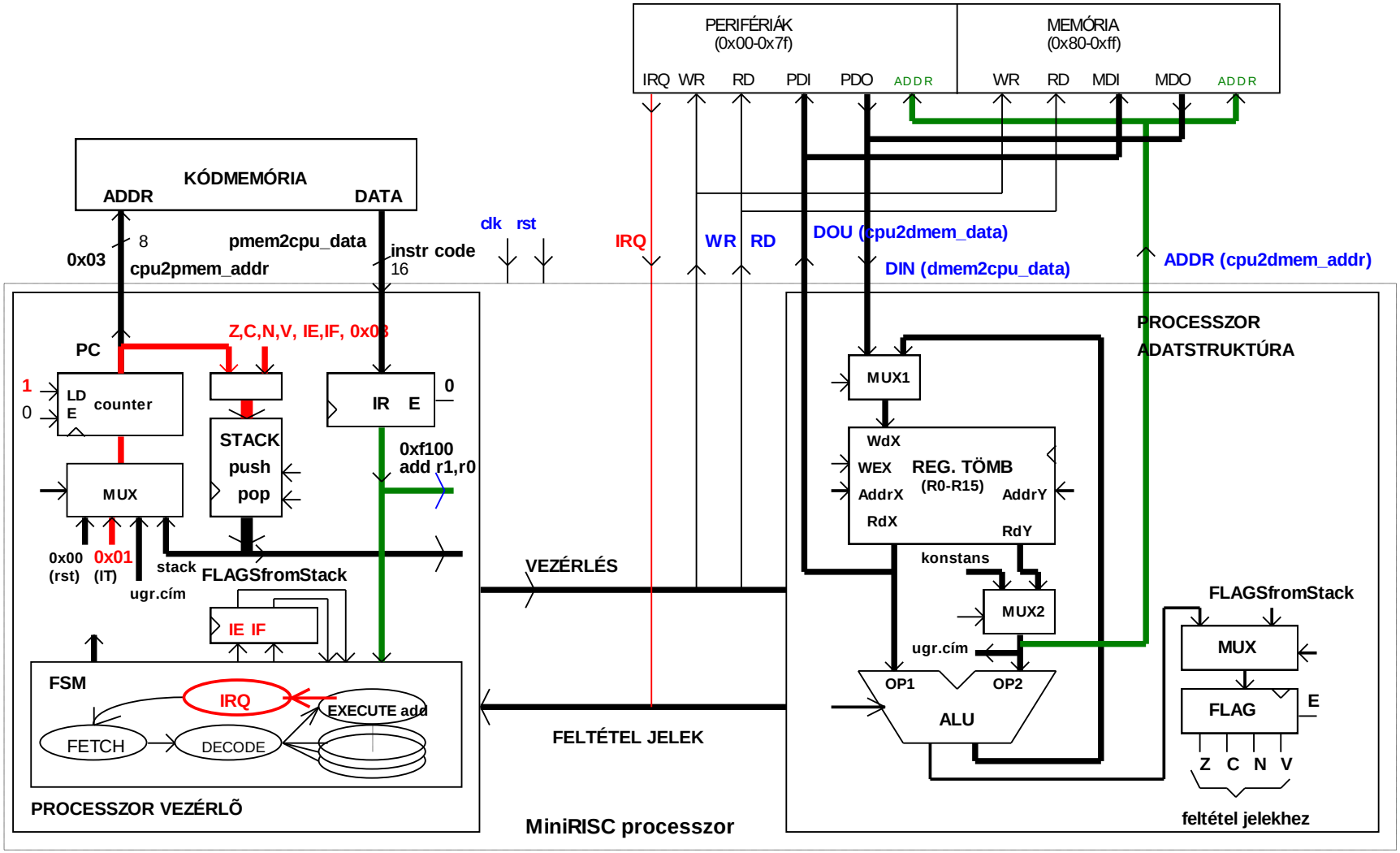
vezérlés átadó utasítás (ugró), direkt ugrás: **EXECUTE** (utasítás végrehajtás) ciklus



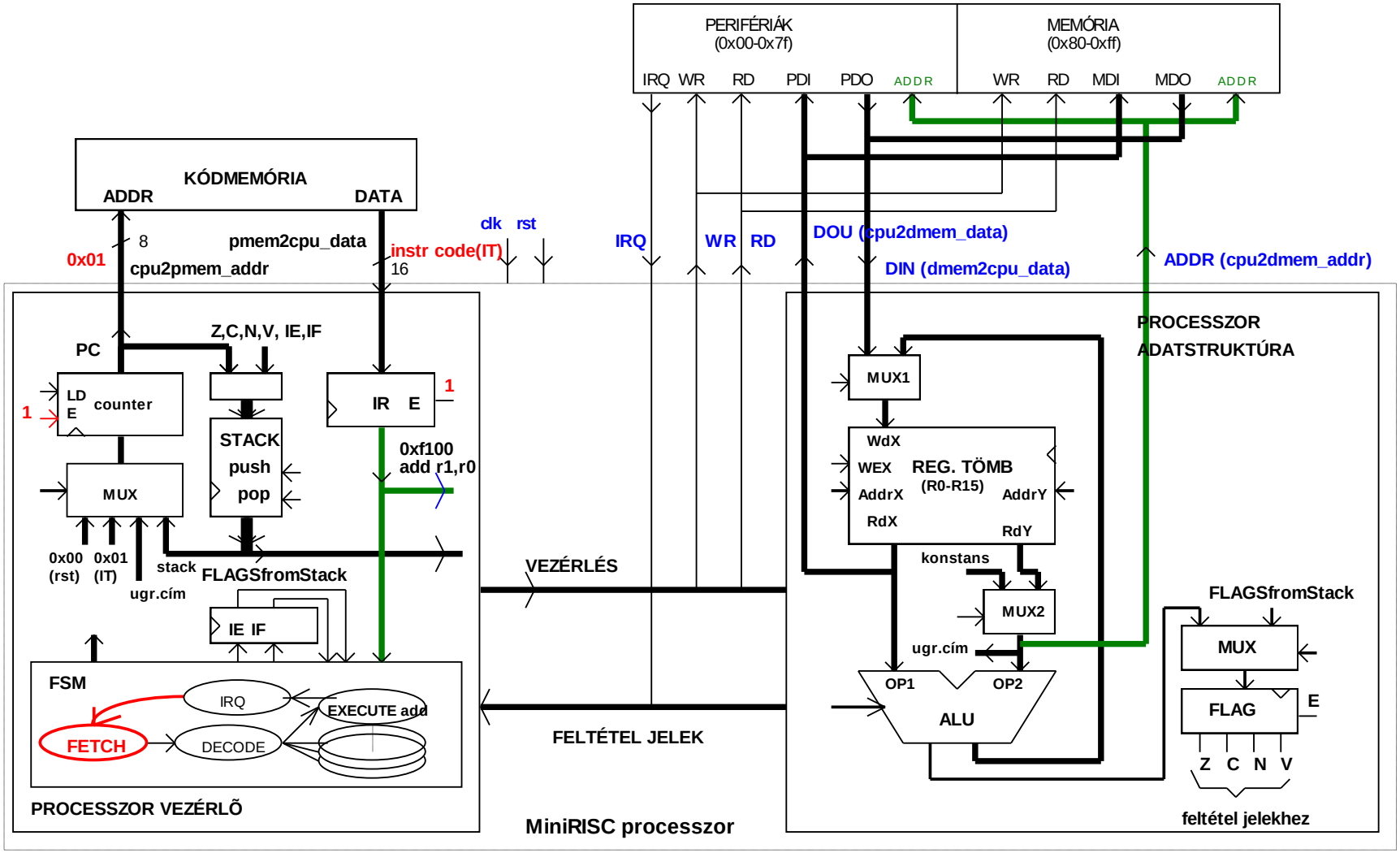
Ugrás végrehajtása uán a PC tartalma (a következő utasítás címe) az ugrási cím



Interrupt kérés hatására**: interrupt elfogadás** ciklus:PC és FLAG-ek STACK-be, IE = 0, IF =1, PC = 1 (IT program kezdőcíme)



**interrupt elfogadás** ciklusból a FETCH-be lépéskor a PC-be az interrupt első utasításának címe kerül (Mini RISC processzornál ez mindig 0x01)



## *Minta program*

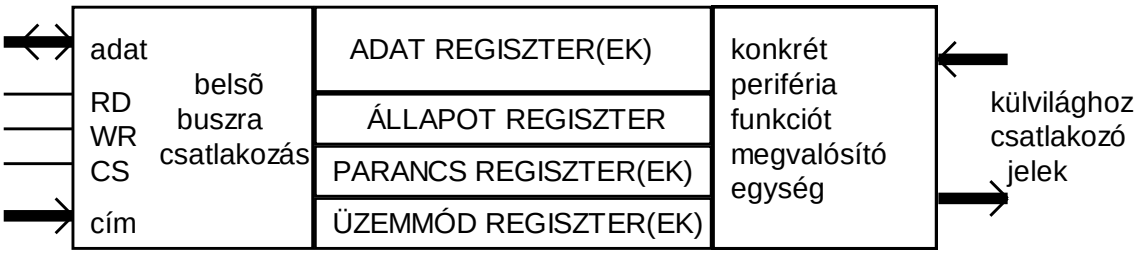
A memóriában levő 2 adat beolvasása, összeadása, az eredmény kiírása a memóriába.

```
   CODE

   mov  r0, data1        ; REG[0] = DMEM[0]
   mov  r1, data2        ; REG[1] = DMEM[1]
   add   r1, r0          ; REG[1] = REG[1] + REG[0]
   mov  result, r1       ; DMEM[3] = REG[1]
loop:
   jmp loop

   DATA
data1:
   DB 0x20
data2:
   DB 0x14
result:
   DB 0x00
```

# *A perifériák*




Egy periféria 4 fajta regiszterrel rendelkezhet, melyek írása, olvasása a RAM-hoz hasonló vezérlő jelekkel történik.

**Üzemmód regiszter:** A periféria működési módjának beállítására szolgál. Egy-egy periféria egy adott feladatcsoport megoldását teszi lehetővé konfigurálható üzemmódok segítségével (pl. egy timer egységgel időzítési, számlálási, impulzus generálási feladatok oldhatók meg).

**Parancs regiszter**: A periféria valamely működését lehet vele kezdeményezni. Pl. A/D konverzió indítása. Sokszor az üzemmód és parancs reigsztert összevonják és vezérlő regiszternek nevezik.

**Állapot (státus) regiszter:** A periféria állapotáról ad információkat. Az állapotregiszter bitjei ún. megszakítást is kérhetnek, ha az engedélyezett.

**Adat regiszter:** Ide kell beírni itt lehet kiolvasni az adatot.

## Periféria kezelési módszerek

- A perifériák egy része jelzést ad, ha elkészült a feladattal (új adat keletkezett vagy új adat fogadására kész).

- A jelzés észrevételére és az adat kezelésére 3 féle alapmódszer létezik. Hogy ezek közül melyiket célszerű alkalmazni, az a periféria és a CPU relatív sebességétől függ.

# A perifériákat sebesség szerint 3 kategóriába sorolhatjuk:

**a.** A CPU sebességénél gyorsabban termel adatokat vagy gyorsabban kell kapnia adatokat, mint amit CPU képes lekezelni. (Ezzel most nem foglalkozunk részletesebben.)

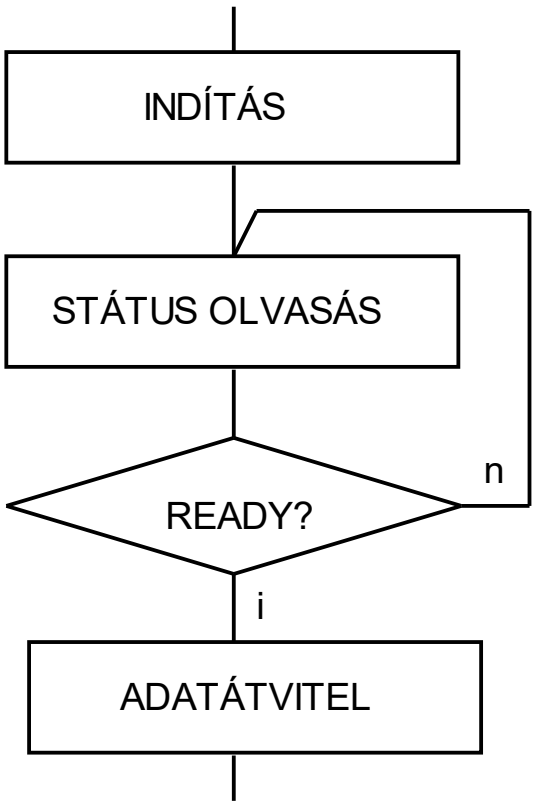
**b.** A periféria az indítása után elég gyorsan (néhányszor 10 utasításnyi időn belül) elkészül, a CPU képes lekezelni.

**c.** A periféria az indítása után nagyon soká készül el. (A CPU annak elkészültéig nagyon sok utasítást képes végrehajtani.)

# Programozott lekérdezéses periféria kezelés

- A b. esetben CPU a periféria státus regiszterét figyelve veheti észre, hogy a perifériának kiszolgálási igénye van.

- Mivel a periféria olyan sebességgel termeli az adatot, hogy a státus figyelő ciklusban nem tölt sok időt a CPU, így nem romlik a kihasználtsága.



***Interruptos periféria kezelés***

- A c. esetben az a célszerű, ha a periféria aktívan jelzi a CPU-nek a kiszolgálási igényét, mert pl. programozott lekérdezésnél feleslegesen sok időt töltene a lekérdezési ciklusban.

- Ezt interrupt (IT) kéréssel teheti meg.

**Az IT hatása:**

- A CPU a következő utasítás címét a stack-re menti.

- Többnyire letiltja a további IT-ket.

- Elugrik az IT rutin elejére.

- Végrehajtaja az IT rutin utasításait.

- Az IT rutin utolsó utasítása RETI, aminek hatására előveszi a stak-ről az oda mentett címet és ennek alapján visszaugrik a megszakított program következő utasítására.

- Az IT rutin feladata a periféria jelzés hatására az IT rutinban gyorsan elvégezni a feladatok időkritikus részét (pl. adat elvétele).

- A többi kapcsolódó (nem időkritikus) feladatot a főprogramra kell hagyni.

## Az IT rutin címéne meghatározása, IT rendszerek

## Egyszerű IT rendszer:

- Minden IT rutin ugyazon a címen kezdődik.
- Az IT kérő periféria beazonosítása a státusregiszterek lekérdezésével történik