Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

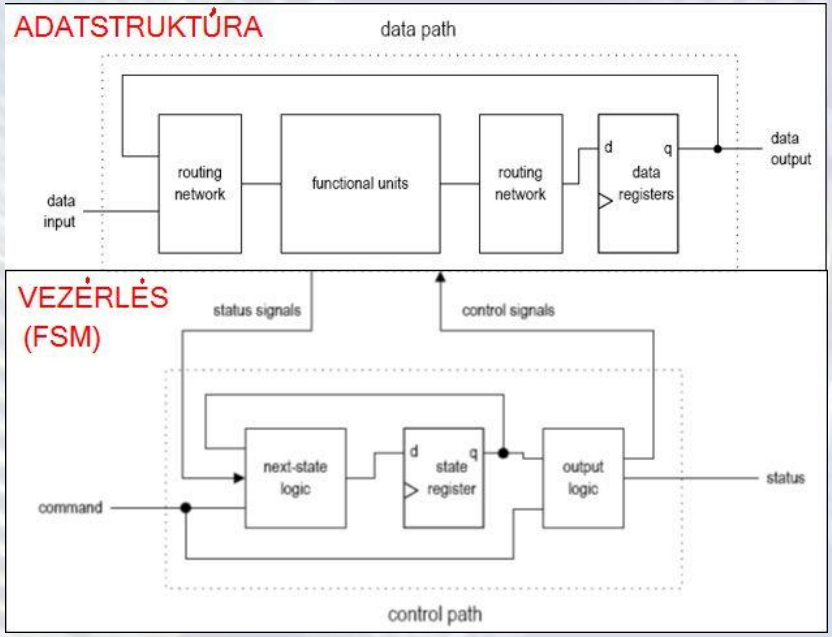
Digitális technika VIMIAA02

8. hét

Fehér Béla, Benesóczky Zoltán
BME MIT

Processzor adatstruktúrák

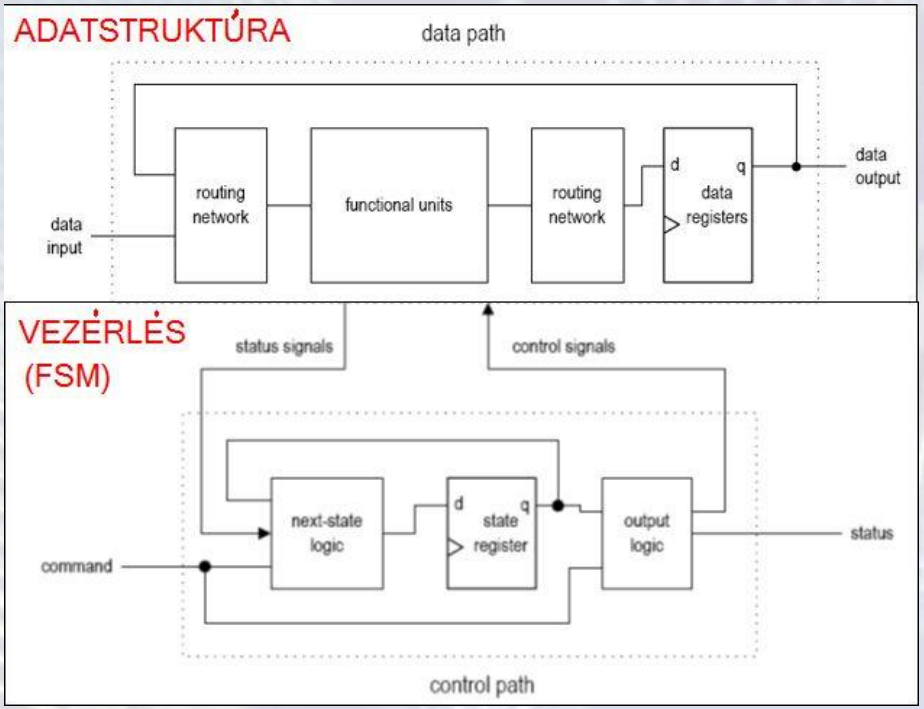
- **DIGITÁLIS RENDSZEREK ÁLTALÁNOS FELÉPÍTÉSE: ADATSTRUKTÚRA + VEZÉRLÉS**
- **Vezérlés: <u>Minden feladatra egységes általános elv</u>**
  - Az algoritmust leíró HLSM állapotdiagram alapján megtervezhető az adastruktúra és a vezérlés.
- **Adatstruktúra:**
  - ***Feladatspecifikus felépítésű***
  - ***A HW sok műveletet végez-het egyszerre, ezért gyors***
- **Vezérlő:**
  - ***Feladatspecifikus, de szisztematikusan tervezhető (FSM)***



Processzor adatstruktúrák

- **ADATSTRUKTÚRA + VEZÉRLÉS**
- **Előnye**
  - ***Gyors***, mert ***tetszőlegesen sok párhuzamos művelet végzés***re képes.
- **Hátránya**
  - ***Feladatonként új adatstruktúrát és vezérlőt kell tervezni***.

    Sok párhuzamos művelet esetén bonyolult a hardver.



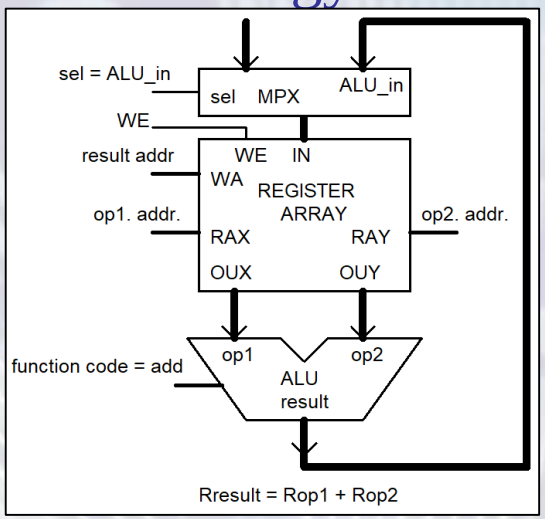
Processzor adatstruktúrák

- Lehet „***mindenre" használható adatstruktúrát*** és vezérlőt készíteni, ha ***feladjuk a párhuzamos működés előnyét***.
- **Előnye:** ***Ugyanaz a hardver szinte minden feladat elvégzésére alkalmas lesz***, nem kell faladatonként újra tervezni az egész hardvert.
- **Hátránya:** ***Egyszerre csak egy műveletet képes elvégezni, ezért lassabb***, mint az adatstruktúra-vezérlő megvalósítás.
- A ***feladatokat sok időben egymás után végzendő elemi (egyszerű) műveletre kell bontani***, melyeket ***ugyanaz a HW*** végez el a megadott sorrendben.
- ***Sokféle művelet elvégzésére alkalmas adatstruktúra*** kell, olyan művelet halmazzal, melynek ***segítségével „minden" számítási feladat megoldható***.

Processzor adatstruktúrák

A „mindenre” használható adatstruktúra:

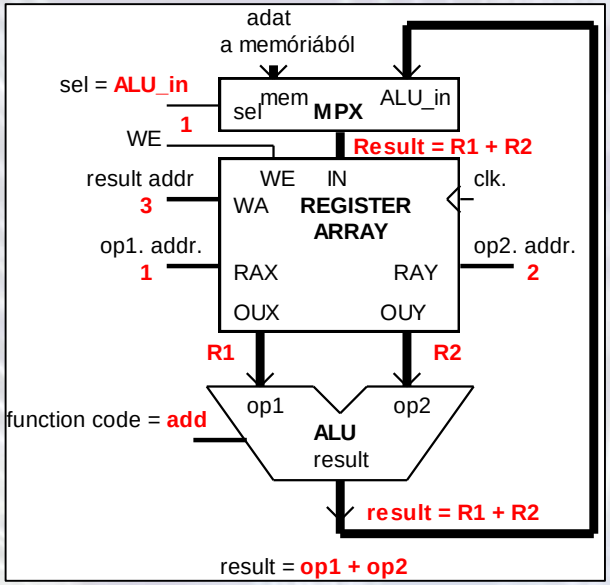
- Ez ***egyszerre egy műveletet képes elvégezni maximum 2 operanduson*** és ***beállítható***, hogy az milyen ***művelet*** legyen.
- A művelet ***operandusai egy regisztertömb-ből származnak és az eredmény is abba kerül vissza***, így felhasználható a következő művelethez. (A regisztertömb i-edik regiszterének tartalmára Ri-vel hivatkozuk.)



- Az adatstruktúra ***standard szószélességű adatokon képes elemi műveleteket végezni***. Tipikus szószélességek 8, 16, 32, 64 bit.
- Azonban ***több elemi művelettel képes a szószélességénél szélesebb adatokon is a műveleteket végezni***.

Processzor adatstruktúrák

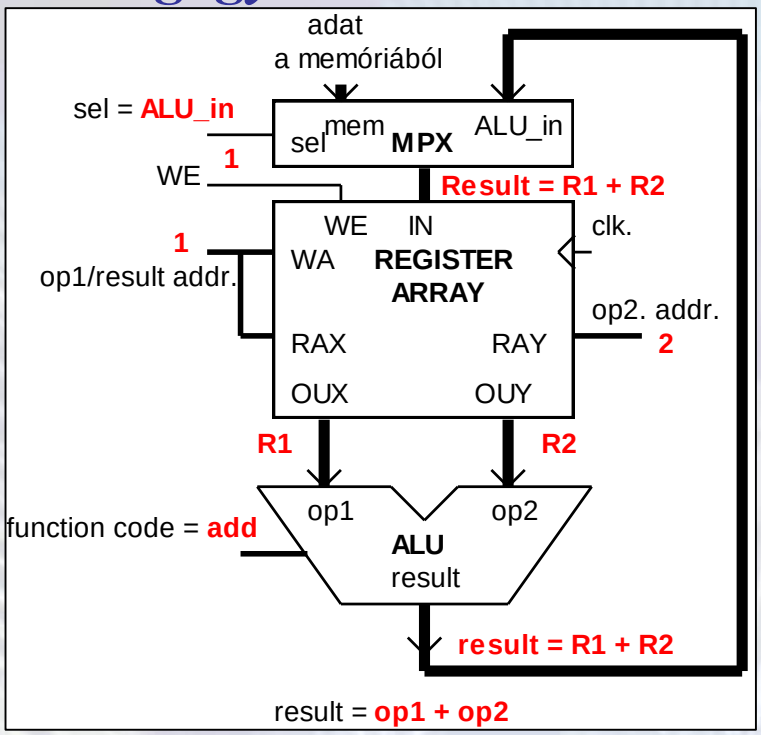
- **3 regiszter címes processzor adatstruktúra**
  - Az ***ALU (aritmetikai logikai egység)*** által végzendő ***művelet a func. code bemenetére adott kóddal állítható*** be. Ez annyi bites bemenet, ahány bittel az összes művelete megkülönböztethető. (A példában ösz-szeadás műveletet állítottunk be.)
  - A regiszter tömb RAX és RAY címző bemeneteivel ***megcímzett regiszterek (operandusok) tartalma megjelenik a regisztertömb kimenetein melyek egy-ben az ALU bemenetei.*** Az eredmény az ALU kimenetéről a MUX-on keresztül a regisztertömb bemenetére jut.



  - Az ***ALU művelet eredménye) WE = 1 esetén beíródik a WA című regiszterbe*** az órajel aktív élére. (A példá-ban a WA=3 tehát az R3 regiszterbe.) Ebben az adatstruktúrában a 2 (operandus) regiszteren végzett művelet eredménye egy 3-adik regiszterbe kerül. **Rresult = Rop1 művelet Rop2** (Pl. R3 = R1 + R2)

Processzor adatstruktúrák

- **2 regiszter címes processzor adatstruktúra**
  - Ha az ***op1 operandus olvasási címe megegyezik az eredmény címével*** (összekötjük a címbiteket), akkor ***az ALU művelet eredménye felülír-ja az op1 operandust*** tartalmazó regisztert.
  - Ennél csak 2 regisztert kell meg-címezni, de felülíródik az egyik operandust tartalmazó regiszter értéke.



  - Ha az ope1-re később is szükség van, több adatmoz-gatásra van szükség, mint a 3 regiszteres esetben.
    Rop1 = Rop1 művelet Rop2 (Pl. **R1** = **R1** + **R2**)

Processzor adatstruktúrák

- Az ALU-nak ***FLAG*** kimenete is van, aminek ***egyes bitjei plusz információkat szolgáltatnak a műveletek eredményéről***.
- **Z** (zéró): Z=1, ha az eredmény 0, **C** (átvitel): Előjel nélküli számok összeadás/kivonás átvitel bitje, **N** (negatív): N=1, ha 2'C kódban értelmezett eredmény negatív, **V** (2-es túlcsordulás) V=1, ha a 2'C kódú összadás/kivonás kivonás eredménye nem fér el az eredménybiteken.

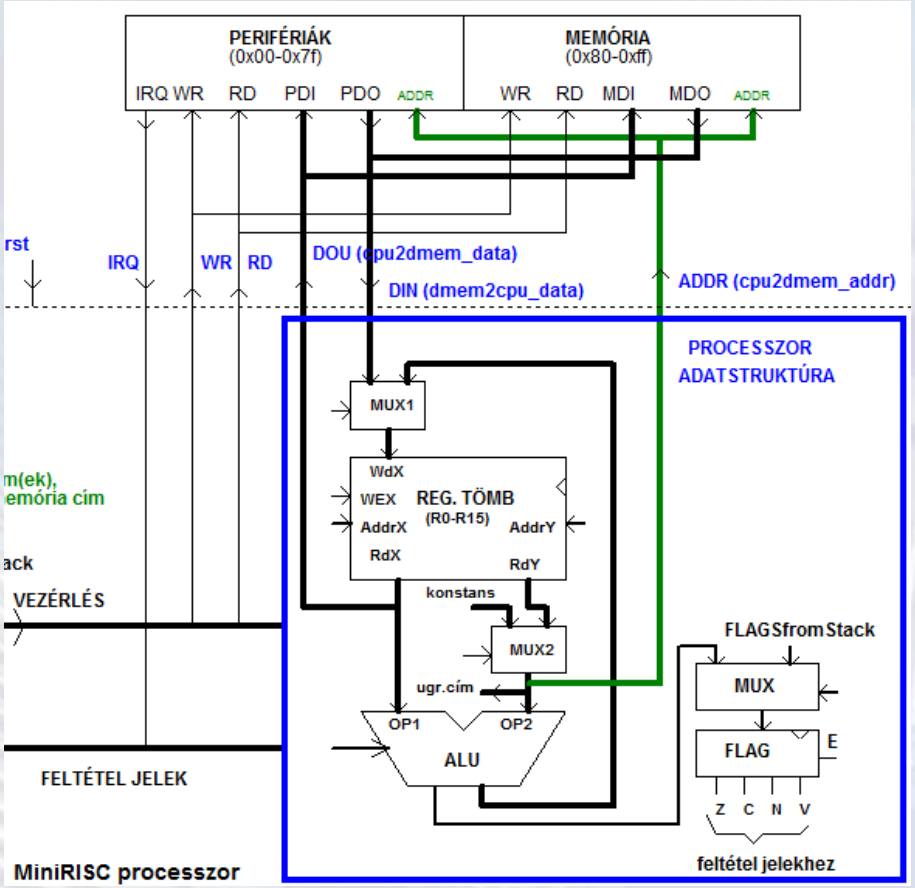
Egy egyszerű **ALU Verilog kódja**:

```
module ALU(input [7:0] op1, op2,
 input [1:0] func_code, output reg [7:0] result,
                                output  reg V, N, C, Z);
localparam f_add = 2'b00, f_sub = 2'b01,
                f_and = 2'b10,   f_or = 2'b11;
always@(*)
case(func_code)
2'b00:
  begin
  {C, result} = op1 + op2; //add
  {N, Z, V} = {result[7], (result == 8'h0),
      (op1[7]==op2[7]) & (result[7]!=op1[7])};
   end
2'b01:
    begin
    {C, result} = op1 - op2;  // kivonás
    {N, Z, V} = {result[7], (result == 8'h0),
      (op1[7]==~op2[7])&(result[7]==op2[7])};
     end
2'b10:
   begin
   result = op1 & op2;  // AND
  {Z, N, C, V} = {(result == 8'h0), 3'b0,};
    end // itt N,C,V értékét nincs értelme felhasználni
 2'b11:
    begin
    result = op1 | op2;   // OR
   {Z, N, C, V} = {(result == 8'h0), 3'b0,};
    end // itt N,C,V értékét nincs értelme felhasználni
endcase
```

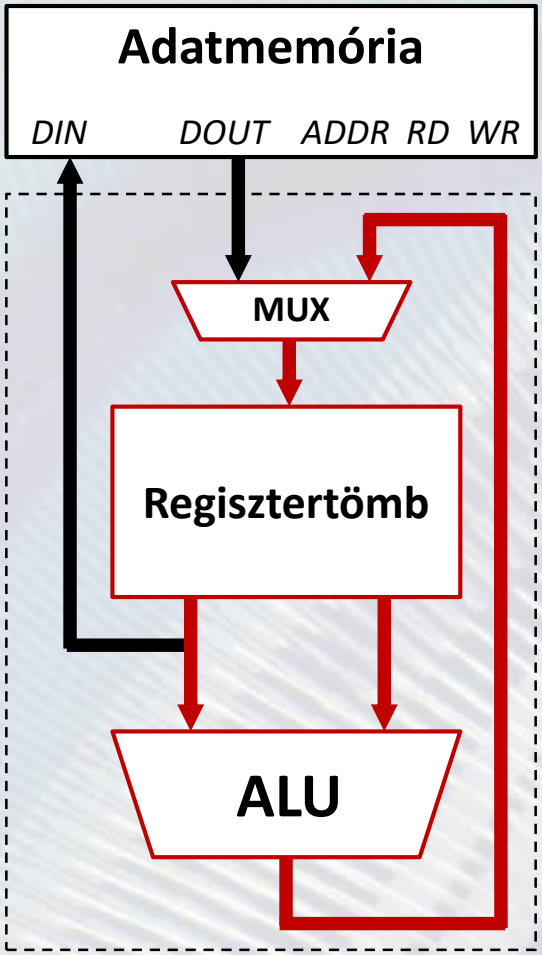
Processzor adatstruktúrák

A processzor adatstruktúra egy konkrét megvalósítását az oktatási célú Mini RISC 8 bites processzoron mutatjuk be. (Ez 2 regiszter címes.)

- Az adatsruktúra (kékkel körberajzolt rész) regisztertömbjébe ***az adatok a kívülről kapcsolódó periféria és memória rekeszekből***, vagy ***az ALU eredmény kimenetéről*** jöhetnek. (A vezérlő MUX1-el választja ki.) ***A periféria rekeszek is memóriaként látszanak, csak más a címtartomány.***
- Az ALU használatát igénylő ***műveletek esetén*** az ALU kimenetén a processzor vezérlő által beállított funkciónak megfelelő művelet eredménye megjelenik, visszaíródik az op1-et tartalmazó regiszterbe és bizonyos ***műveletek*** (***aritmetika, logikai stb.) esetén a flag-ek a FALG regiszterbe kerülnek.***



MiniRISC processzor adatstruktúra működése

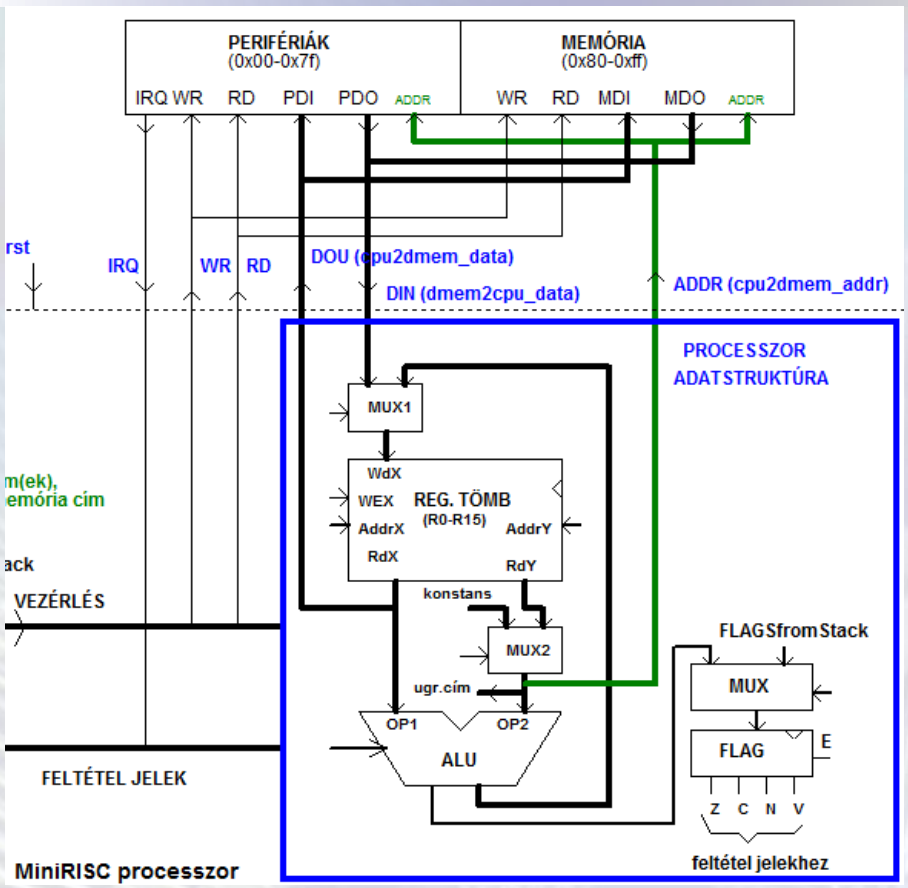


Művelet végzés lokális adatokon (ALU művelet)

Processzor adatstruktúrák

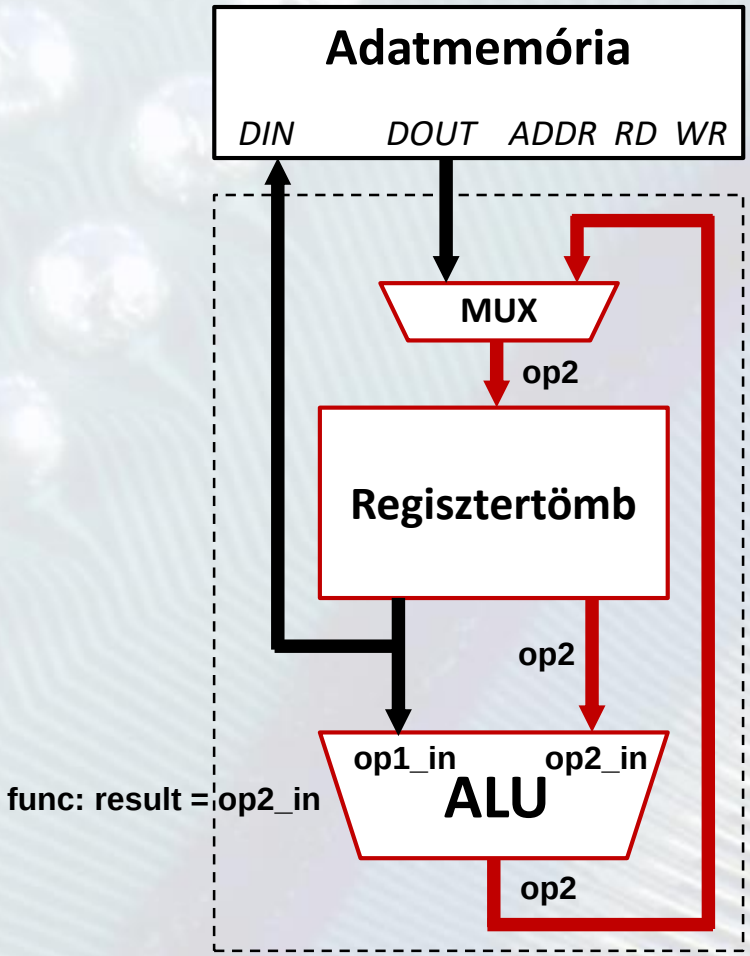
A Mini RISC processzor adastruktúrája

- ***Adatmozgatás regiszteről regiszterbe:***
  A vezérlő MUX2-vel RdY-t MUX1 az ALU kimenetét választja ki. Az ALU funciókód által beállított művelet az, hogy az ALU kimenete az OP2 legyen, vagyis a mozgatandó adat.
- ***Regiszterbe konstans adatmozgatás:*** Az ALU művelet egyik operandusa***konstans is lehet***, nem csak egy regiszter.

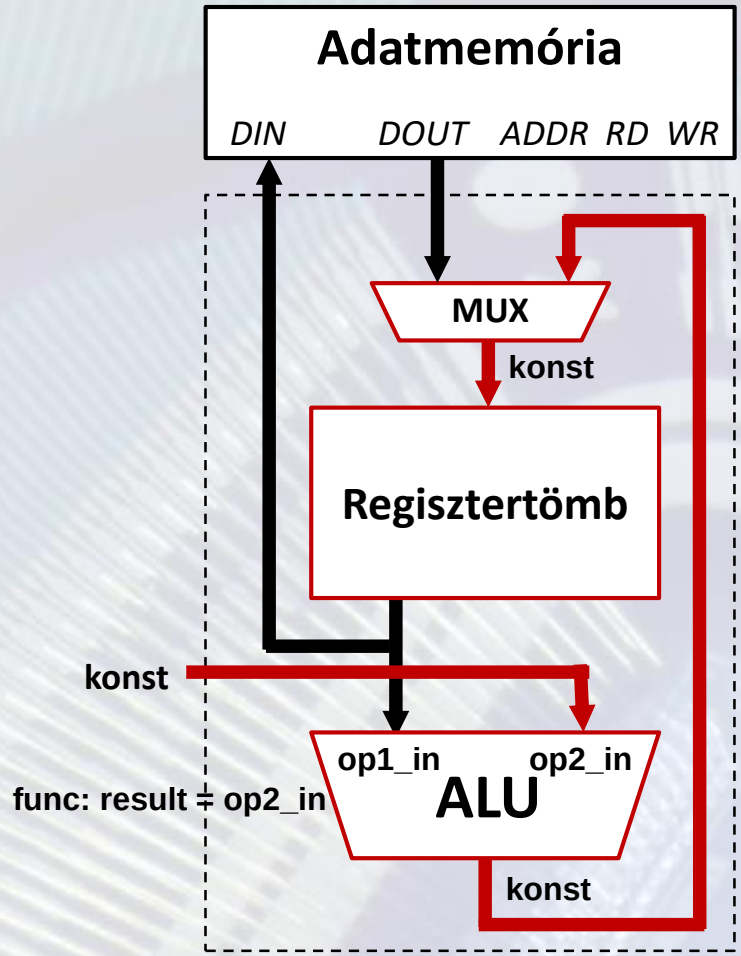


- Ezt a vezérlő a MUX2-vel választja ki. (A konstans az utasításkódból származik, lásd később.) Az előzőhöz hasonló, csak MUX2-vel a konstans lesz kiválasztva.
-

MiniRISC processzor adatstruktúra működése



Adat mozgatás regiszterből regiszterbe

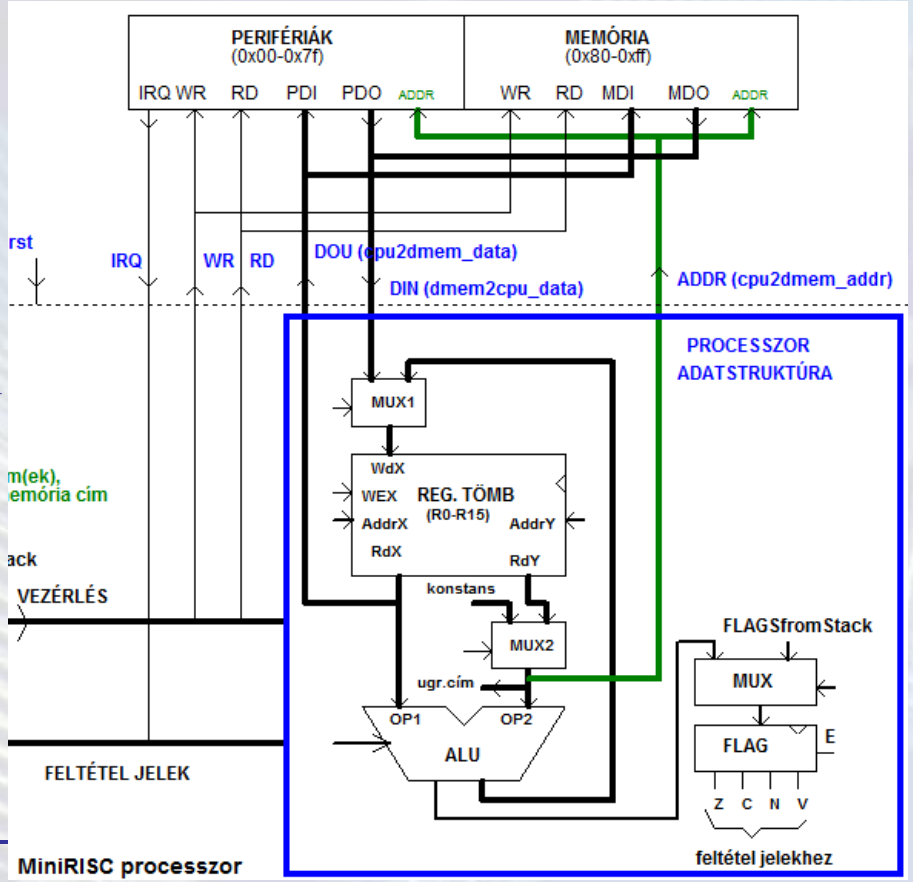


Adat mozgatás, regiszterbe konstans

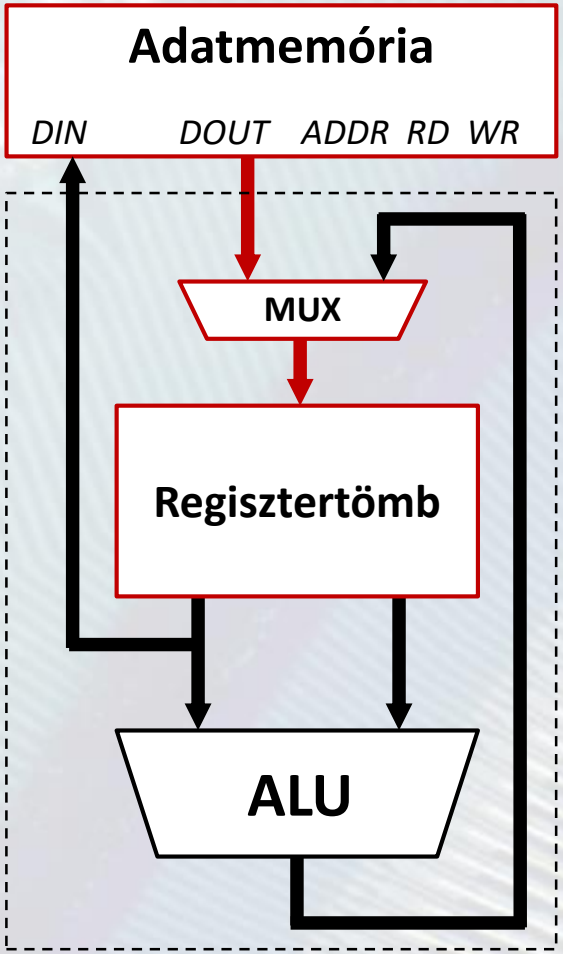
Processzor adatstruktúrák

A Mini RISC processzor adastruktúrája

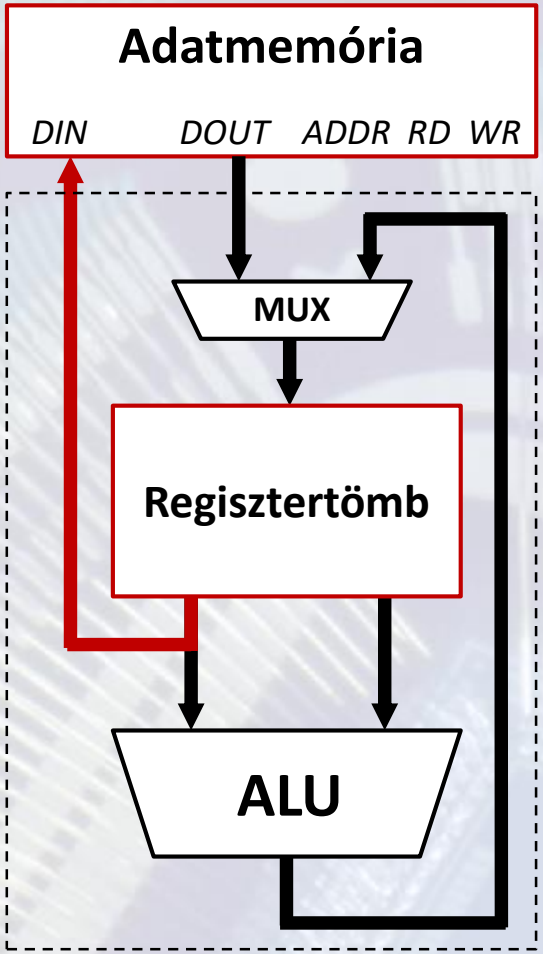
- ***Adatmozgatás regiszterből memóriába:*** Ilyenkor a regisztertömb RdX kimenete adja a processzor adat-busz kimenetét. ***A memória címe lehet konstans*** (ami a processzor utasítás kódjából származik, lásd később) és ***lehet az egyik regiszter tartalma*** (indirekt cím). Ezek közül a vezérlő MUX2-vel választ.
- ***Adatmozgatás memóriából regiszterbe***: A vezérlő a MUX1-el a memória/periféria adat kimenetét választja ki a regisztertömb be-menetére. A memória megcím-zése a regiszterből memóriába moz-gatás címzésével azonos módon történik.



MiniRISC processzor adatstruktúra működése



Adatmemória/periféria olvasás (load)



Adatmemória/periféria írás (store)

Processzor vezérlés

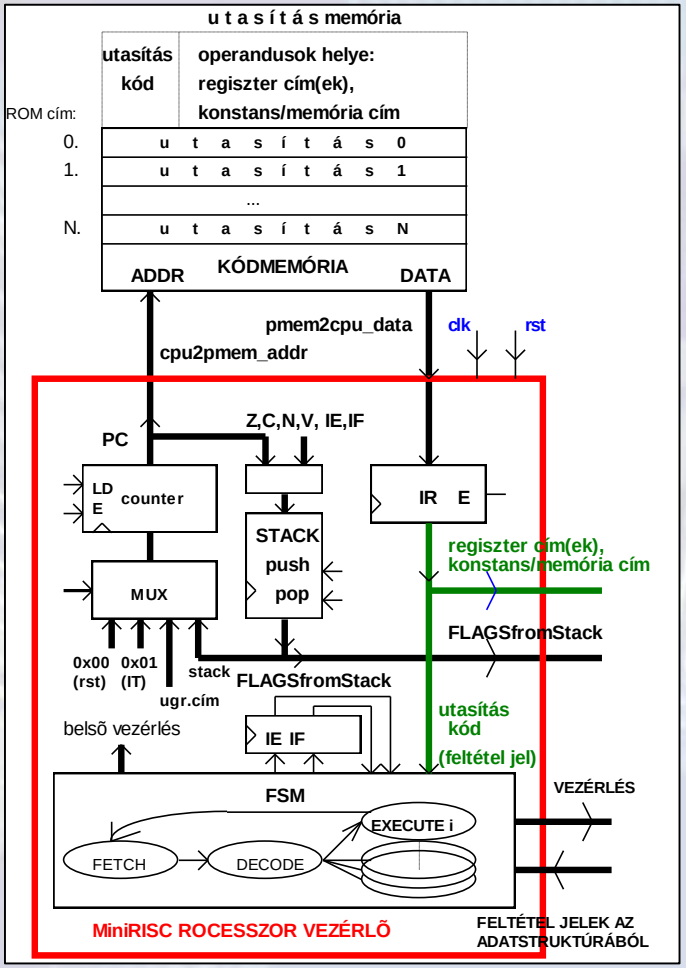
A processzor vezérlés egy konkrét megvalósítását az oktatási célú 8 bites MiniRISC procesz-szoron mutatjuk be.

- A processzor adatstruktúra vezérlője (pirossal keretezve) egy hagyományos ***állapotgépből (FSM) és ezt kiegészítő saját adatstruktúrából áll***.
- A ***vezérlő*** utasítás vezérelt, ***az utasításokat az utasítás memóriából olvassa***. (A MiniRISC utasítás memóriája 16 bit szószélességű.)
- Az ***utasítás*** a következő információkat tartalmazza az egyes bitmezőiben:

A végrehajtandó ***művelet kódja*** (mit kell csinálni) és az ***operandus(ok) elhelyezkedése vagy konstans operandus értéke***.

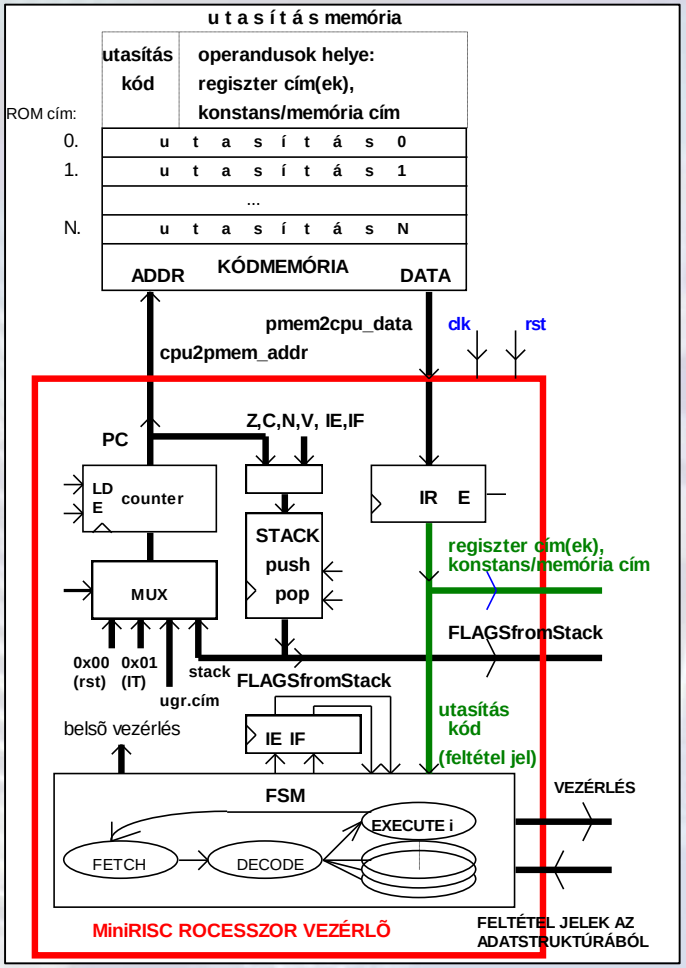
- ***Az utasítás memóriát egy tölthető számláló, a PC*** (Program Counter) ***címzi.***

A MiniRISC processzor vezérlőjének blokkvázlata



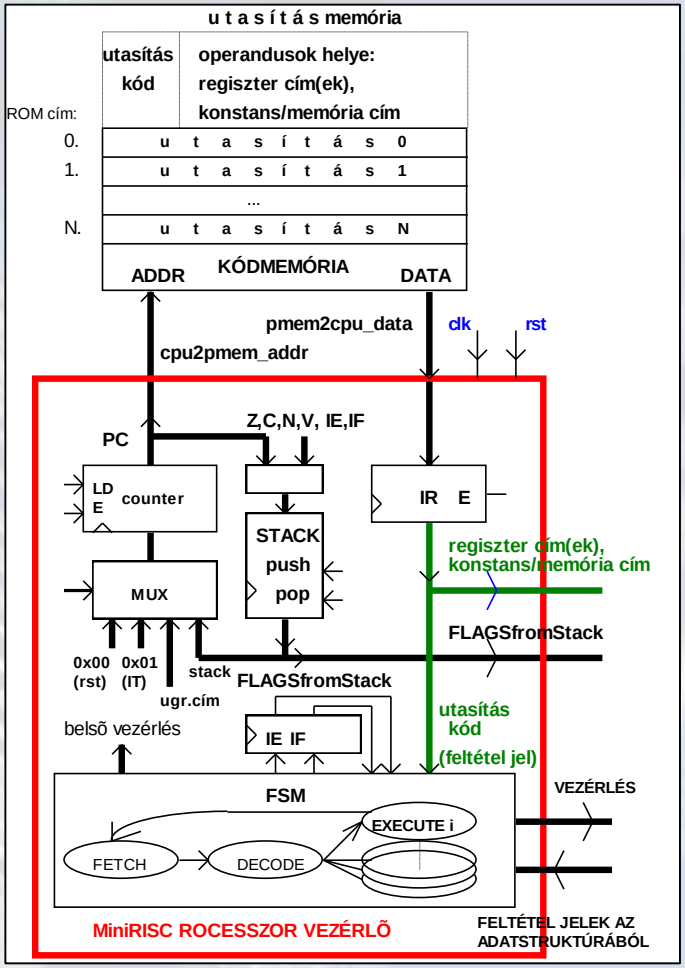
Processzor vezérlés

- **Az utasítás feldolgozása** (ciklusban) történik:*3 fázisban*
  - **FETCH** (utasítás ***beolvasás)***. Utasítás kód betöltése az IR reg.-be.
  - **DECODE** (utasítás ***dekódolás)*** utasítás kódjának felismerése). ***Az utasítás kód alapján*** az utasítás adatsruktúra általi ***végrehajásához szükséges vezérlő kimeneteket előállító EXECUTEi állapotba ugrás***. Annyi ilyen állapot van, ahány különféle vezérlés szükséges a különböző utasítások végrehajtásához.
  - **EXECUTE** (utasítás ***végrehajtás***). A ***művelet elvégzéséhez szükséges vezérlőjelek kiadása a processzor adatstruktúrának***.



Processzor vezérlés

- A **FETCH** ciklus alatt ***az FSM a FETCH állapotban van***. A PC által ***megcímzett utasításkód megjelenik az utasítás memória adat kimenetén. A következő órajelre az utasításkód beíródik az IR utasítás regiszterbe és a PC számol 1-et (PC = PC+1).*** Ugyanekkor ***az FSM állapotgép átlép a DECODE állapotba***.
- A **DECODE** ***állapotban*** az állapotgép ***az utasítás műveleti kód bitmezőjét figyeli*** (feltétel bemenete) és az ennek ***megfelelő (i-edik) utasításhoz tartozó EXECUTEi állapotba lép*** a következő órajel hatására.
- Az **EXECUTEi** ***állapotban*** a vezérlő ***kiadja az adatstruktúrának a művelet elvégzéséhez tartozó vezérlőjeleket.***
- ***A művelet a következő órajel hatására végrehajtódik és az FSM újra a FETCH állapotba lép*** (ha nem észlel interruptot, lásd később).



Processzor vezérlés

Processzor vezérlés, általános tulajdonságok

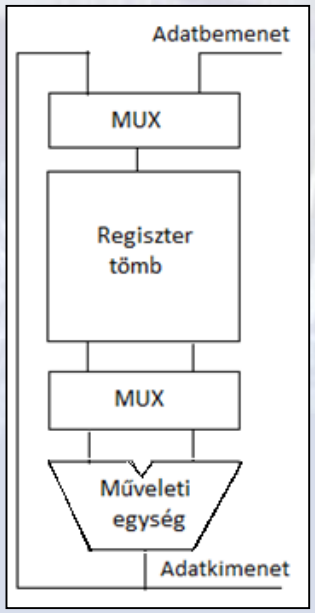
- ***Az utasítások végrehajtási sorrendje az ASM-nek megfelelően korlátozott, csak maximum 2 felé ágazás lehetséges***.
- Az ***utasítás végrehajtás normál sorrendje*** esetén egy utasítás után a következő (1-el nagyobb címen levő utasítás végrehajtása következik, ***PC = PC+1***).
- Vannak **feltétel nélküli elágazó (ugró) utasítások**. Ilyenkor egy az utasítás kódja alapján ***ismert címen*** folytatódik az utasítás végrehajtás (***PC = ugrási cím*,** ami egy konstans, vagy reg. adat).
- Léteznek **feltételes elágazó utasítások**. Ilyenkor ***a feltétel teljesülése esetén ugrik (PC = Ugrási cím), egyébként a következő utasítást hajtja végre (PC = PC+1)***. A ***feltétel*** általában az ***ALU flag kimenetei***ből származik. A ***MiniRISC esetén a flag-ek***: **Z** (eredmény 0), **C** (átvitel bit), **N** (2-es komplemens eredmény negatív), **V** (2-es komplemens túlcsordulás)

Processzor műveletvégzés

Regisztertömb alapú adatstruktúrát ***ki kell egészíteni be/kimeneti interfésszel (memória, periféria), hogy az adatokat be lehessen vinni, az eredményt ki lehessen juttatni.***

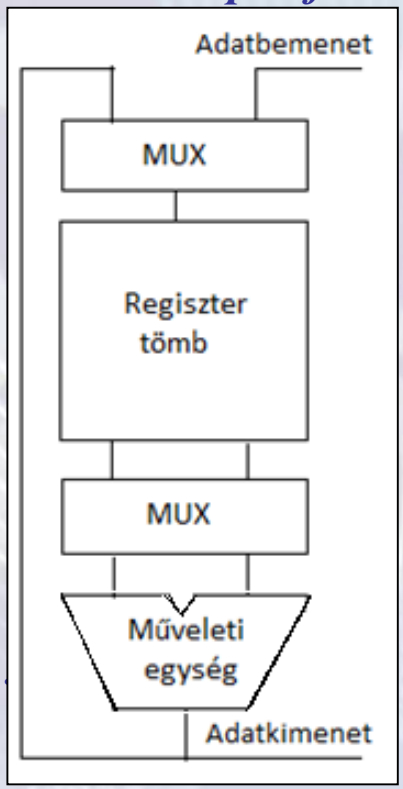
Általános tulajdonságok:

- ***Minden adatot először beírunk a regisztertömbbe*** (**LOAD**)
- ***Műveletet csak a regiszter adatokon végzünk (pl. R1 = R1 + R2)***, de ***létezik közvetlen adat a programkódból (pl. R1 = R1 + KONSTAS)***
- A részeredményeket ***visszaírjuk*** a regisztertömbbe
- A végeredményt ***kiírjuk*** memóriába/perifériába (**STORE**)
- **Ezt hívjuk LOAD/STORE felépítésnek**
- **A regisztertömb mérete**
  - ***Szószélesség***: 8/16/32/64 bit, ***Mélység***:16/32/64 regiszter
  - Több regiszter → több reg. címbit (utasítás méret nő)
  - Több regiszter → kevesebb extra adatmozgatás (gyorsabb)
  - A 32 bites utasításméret jó kompromisszum



Processzor műveletvégzés

- **Az adatstruktúra műveleti egysége az ALU (Aritmetikai Logikai Egység)**
- **Műveleti képességek (utasítás csoportok)**
  Az egyes ***utasítások elnevezése (mnemonikja) sokszor processzor család specifikus*** (ugyanazon műveletnek más és más nevet adnak)
  - ***Adatmozgató*** (MOV, LD, ST...)
  - ***Aritmetikai*** (ADD, SUB, MUL, DIV….)
  - ***Logikai*** (AND, OR, XOR, NOT…)
  - ***Léptetés (shiftelés)*** (SHL, SHR, ASH…)
  - ***Forgatás, a shiftelés kilépő bitje visszajön belépő bitként*** (ROL, ROR,…)
  - ***Feltétel vizsgálat*** (CMP, TST,…)
  - ***Vezérlő (ugró, szubrutin hívó)*** (JMP, CJMP, CALL/RET,...
  - ***Egyéb*** (NOP, EI/DI,...)



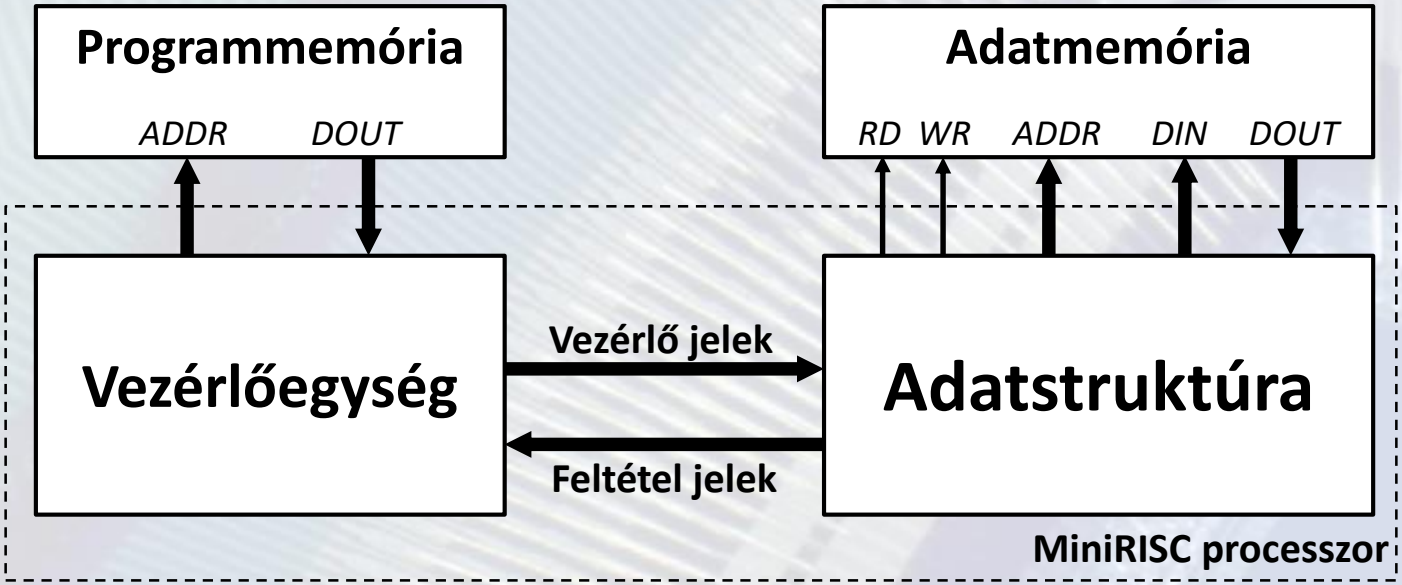
- **Minden műveletet szabványos adatméreten képes elvégezni**
  - 8 / 16 / 32 / 64 bit, az adott rendszer jellemzője (Pl. A MiniRISC 8 bites)
  - **Nagyobb adatméret több utasítással:** Átvitelbittel kiterjesztett műveletvégzés

Összegzés

- **Tetszőleges digitális rendszer:**
  - Általános rendszerterv: ***feladat specifikus adatstruktúra + vezérlés***
- **Processzoros rendszerek:**
  - ***Általános műveletek elvégzérése képes processzor adatstruktúra*** + ASM alapú egyszerűsített ***processzor vezérlőegység***
  - ***Az adatstruktúra vezérlésének indirekt megadása*:**
    - A PC által címzett program memóriában levő utasításba van kódolva, hogy annak végrehajtásához a proc. adatstruktúra mely vezérő jeleit kell aktivizálni
    - PC (programmemória cím)
      → programtár olvasás (aktuális utasítás)
      → dekódolt ***vezérlő jelek származtatása*** és végrehajtás
    - Továbblépés: PC = PC +1 vagy (feltételes) ugrás
- **Egységesített adatméret (szószéleség)**
- **Be/kimeneti interfészek**
- **LOAD/STORE működés:** külső vagy memória adatokat regiszterbe kell tölteni (LOAD) használat előtt, műveletek elvégzése, majd a regiszterben keletkező eredményt ki kell íratni a memóriába/perifériába (STORE).

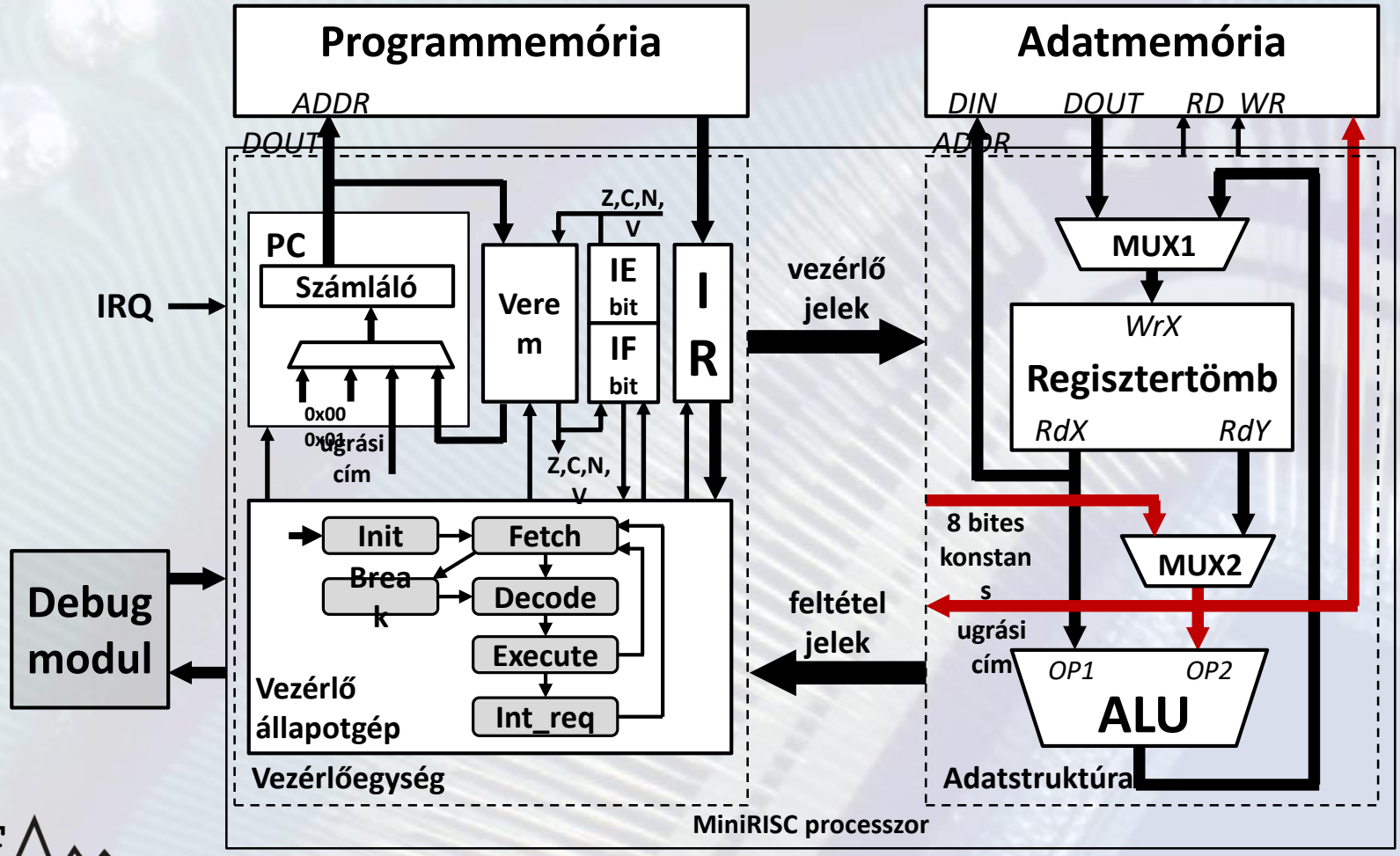
MiniRISC processzor

- **Felépítése követi az adatstruktúra-vezérlő szemléletet**
- **Vezérlő:** az utasítások beolvasása, feldolgozása és ennek megfelelően az adatstruktúra vezérlése
- **Adatstruktúra:** műveletek végrehajtása az adatokon



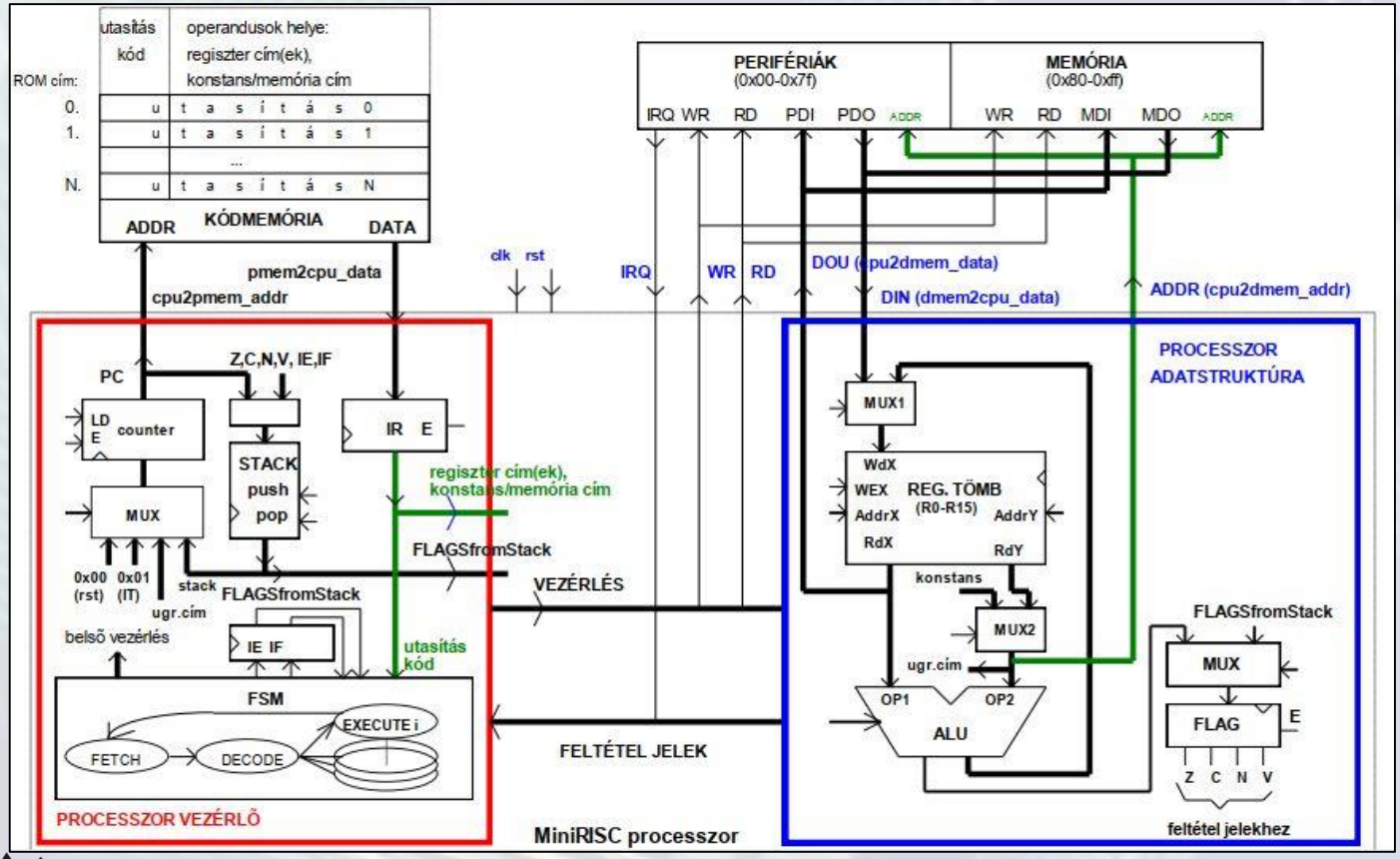
MiniRISC processzor blokkvázlata

- **A processzor egyszerűsített blokkvázlata (adatstruktúra + vezérlő a programmemóriával és adatmemóriával)**



Kiegészítő jegyzetek
A Mini RISC utasítás feldolgozás

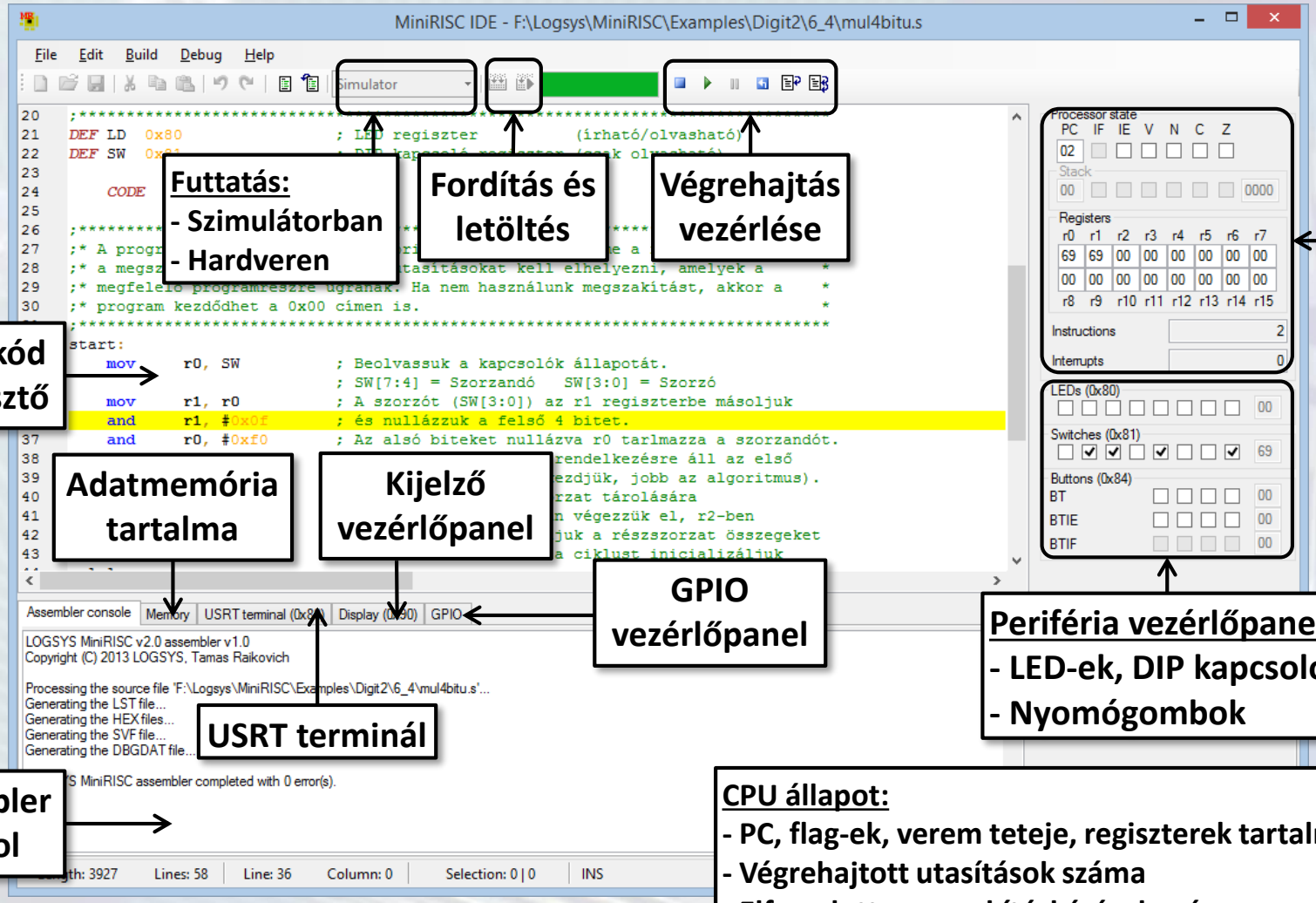
- **A Mini RISC utasítás feldolgozás rajzos magyarázata az azonos nevű kiegészítő jegyzetben található.**



Kiegészítő jegyzetek
Mini RISC IDE

A Mini RISC fejlesztő környezet (Mini RISC IDE) leírása a Mini RISC CPU jegyezet 32. oldalán kezdődik. Elolvasása a laborok processzoros részéhez feltétlenül szükséges.

MiniRISC IDE



Digitális Technikai
8. EA vége