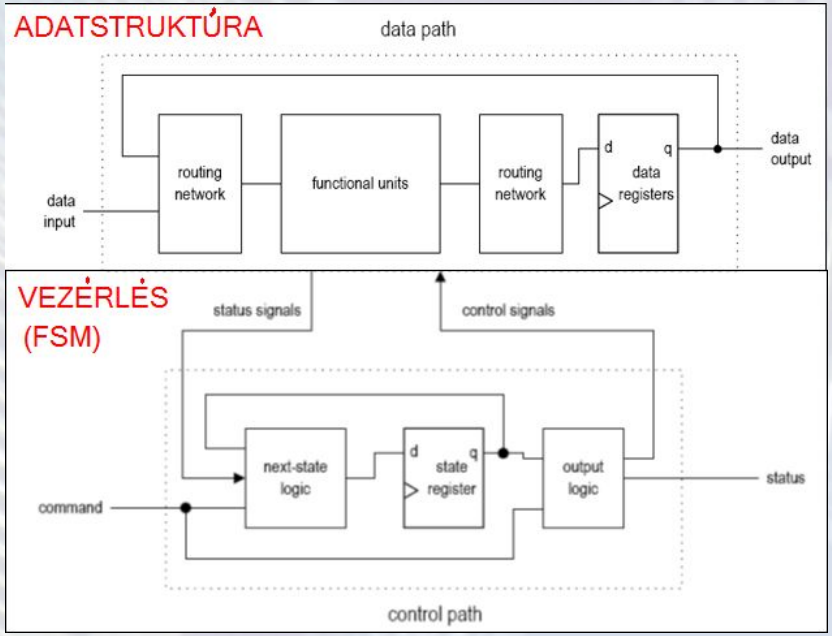
BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
MÉRÉSTECHNIKA ÉS INFORMÁCIÓS RENDSZEREK TANSZÉK

Digitális technika
VIMIAA02
8. hét

Fehér Béla, Benesóczky Zoltán
BME MIT

Processzor adatstruktúrák

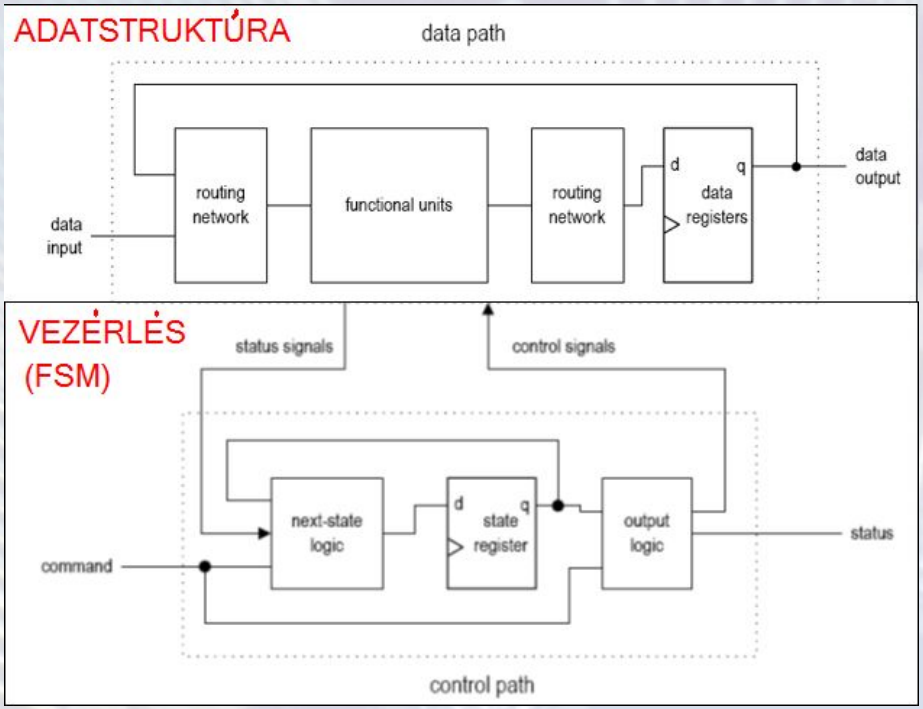
- **DIGITÁLIS RENDSZEREK ÁLTALÁNOS FELÉPÍTÉSE: ADATSTRUKTÚRA + VEZÉRLÉS**
- **Vezérlés: <u>Minden feladatra egységes általános elv</u>**
  - Az algoritmust leíró HLSM állapotdiagram alapján megtervezhető az adastruktúra és a vezérlés.
- **Adatstruktúra:**
  - ***Feladatspecifikus felépítésű***
  - ***A 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 hardwer.



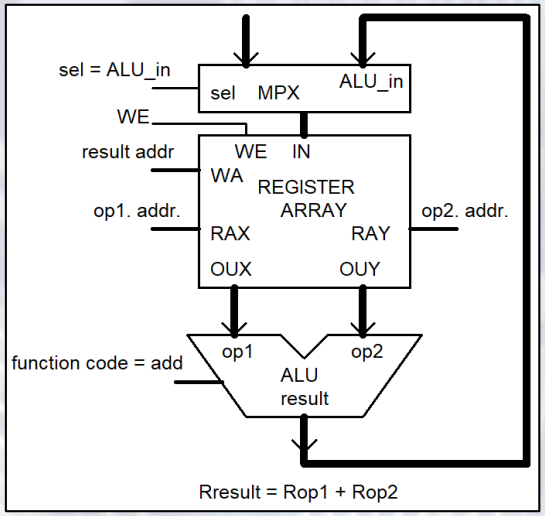
Processzor adatstruktúrák

- 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 bontjuk***, melyeket ***ugyanaz a HW-t*** végez el.
- ***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.



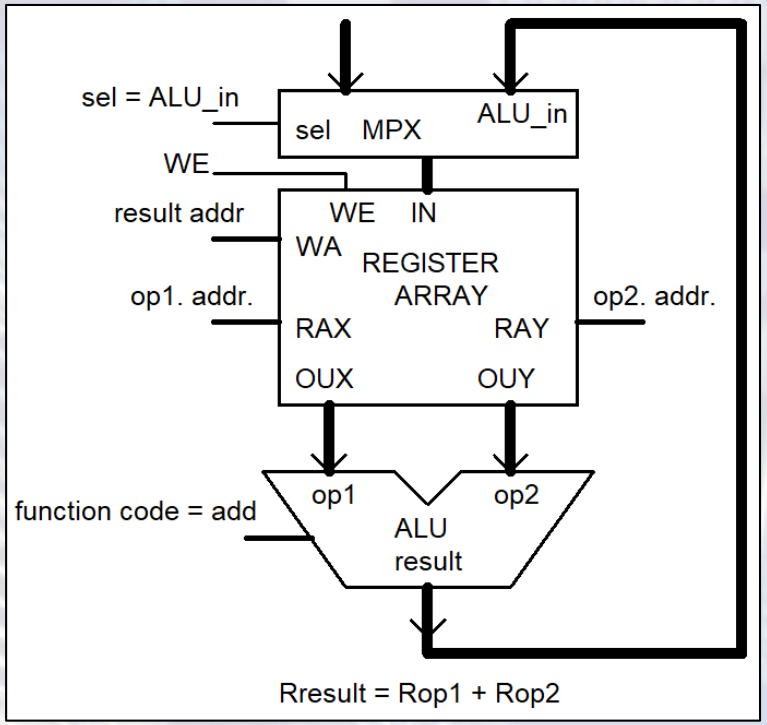
- A művelet ***operandusai egy regisztertömb-ből származnak és az eredmény is abba kerül vissza***, így felhasználható a következő művelethez.

- 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

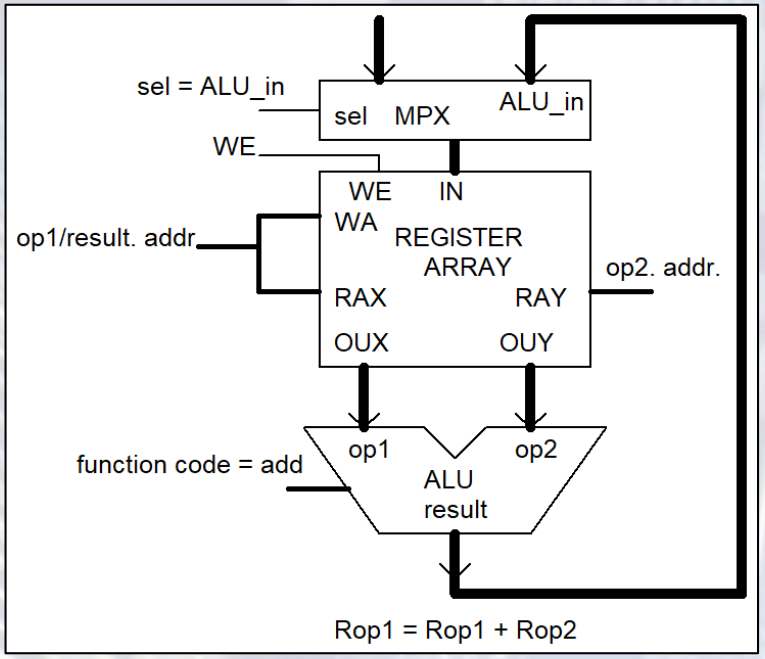
- **3 regiszter címes processzor adatstruktúra**
  - Az ***ALU (aritmetikai logikai egység)*** által végzendő ***művelet a func. code bemenetére adott kóddal állítható*** be. (Az ALU egy kombinációs hálózat)
  - A regiszter tömb RAX és RAY címző bemenetei által ***megcímzett regiszterek (operandusok) tartalma megjelenik az OUX és OUY kimeneteken***.
  - Az ***ALU művelet eredménye WE = 1 esetén beíródik a WA című regiszterbe*** az órajel felfutó élére.



- Ez a processzor adatstruktúra ***3 regisztert használ***
  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 felülírja az op1 operandust*** tartalmazó regisztert.
  - Ennél csak 2 regisztert kell megcímezni, de felülíródik az egyik operandust tartalmazó regiszter értéke.



  - Ha az ope1-re később is szükség van, több adatmozgatásra van szükség, mint a 3 regiszteres esetben.
    Rop1 = Rop1 művelet Rop2 (Pl. R2 = R2 + R3)

Processzor adatstruktúrák

- Az ALU Verilog viselkedési leírással is leírható.
- Példa 4 műveletes ALU-ra (az egyszerűség kedvéért):

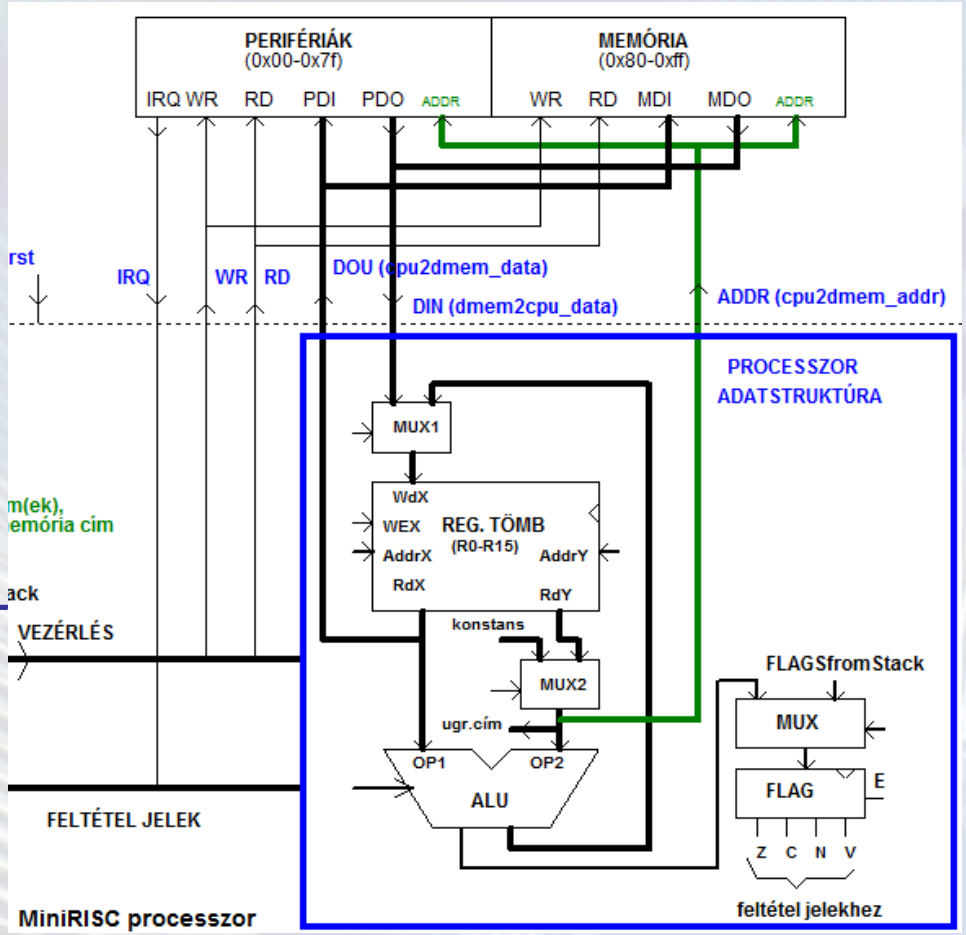
```
always@(*)
case(func_code)
2'b00: out = op1 + op2;   // összeadás
2'b01: out = op1 - op2;   // kivonás
2'b10: out = op1 & op2    // AND
2'b11: out = op1 | op2;   // OR
endcase
```

- Persze egy valódi processzornál jóval több művelet kell és ennél bonyolutabb... (lásd: MiniRISC processzor)

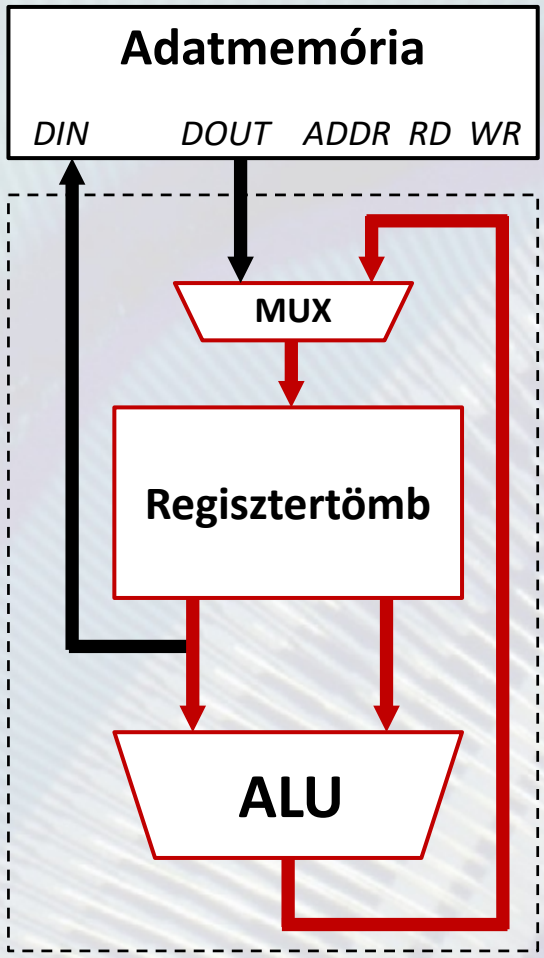
Processzor adatstruktúrák

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 válaszja ki.) ***A periféria rekeszek is memóriaként látszanak.***
- 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 ***nem adatmozgató műveletek esetén a flag-ek a FALG regiszterbe kerülnek.***



MiniRISC processzor adatstruktúra működése

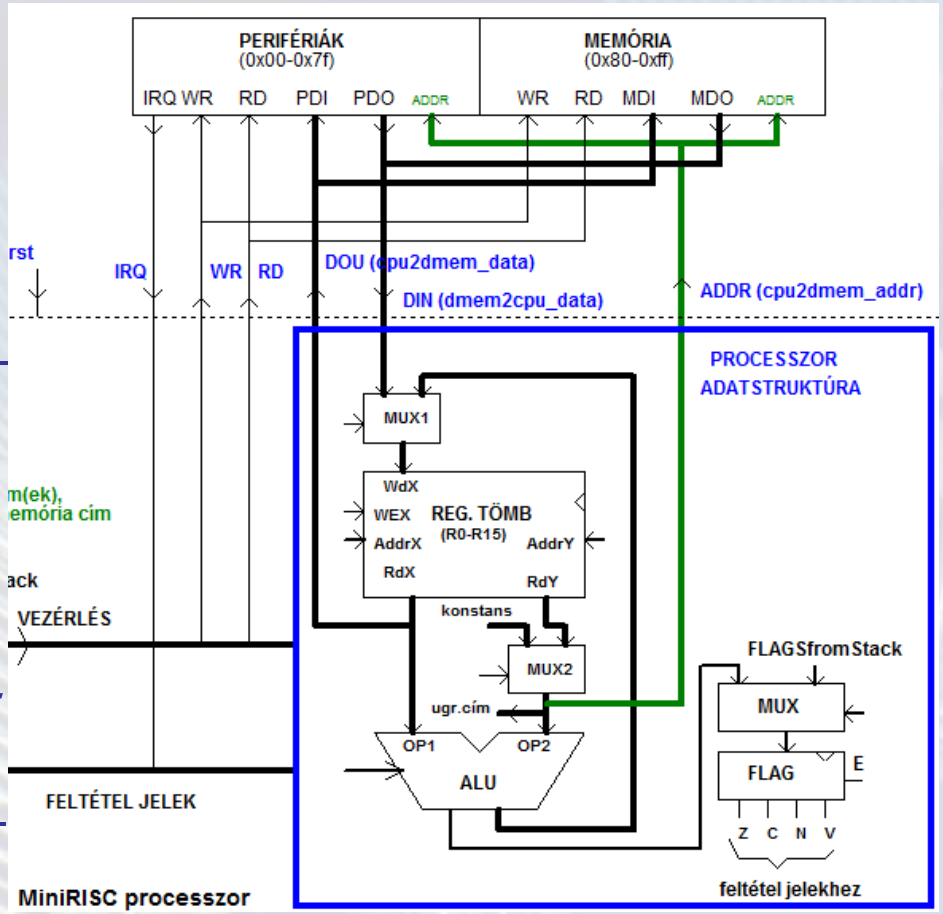


Művelet végzés lokális adatokon (ALU művelet)

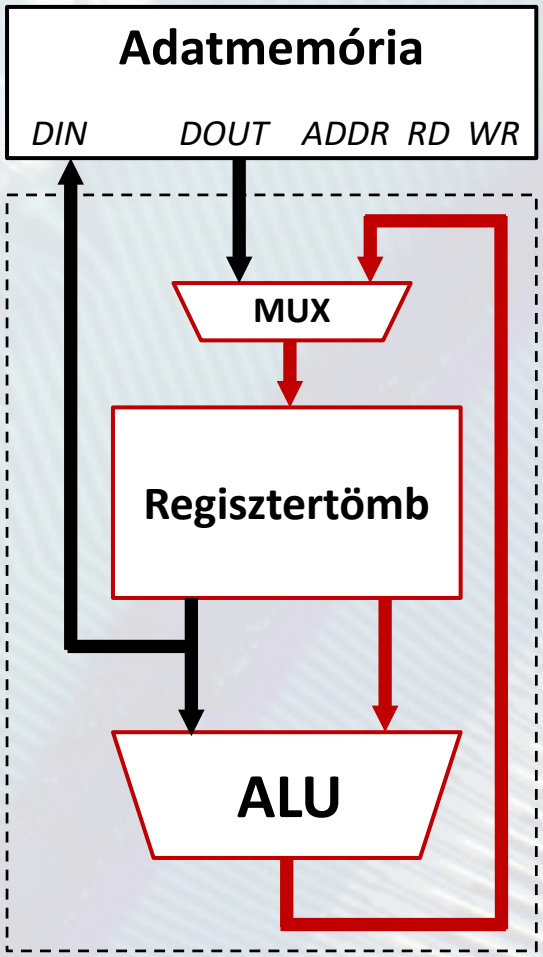
Processzor adatstruktúrák

A Mini RISC processzor adastruktúrája

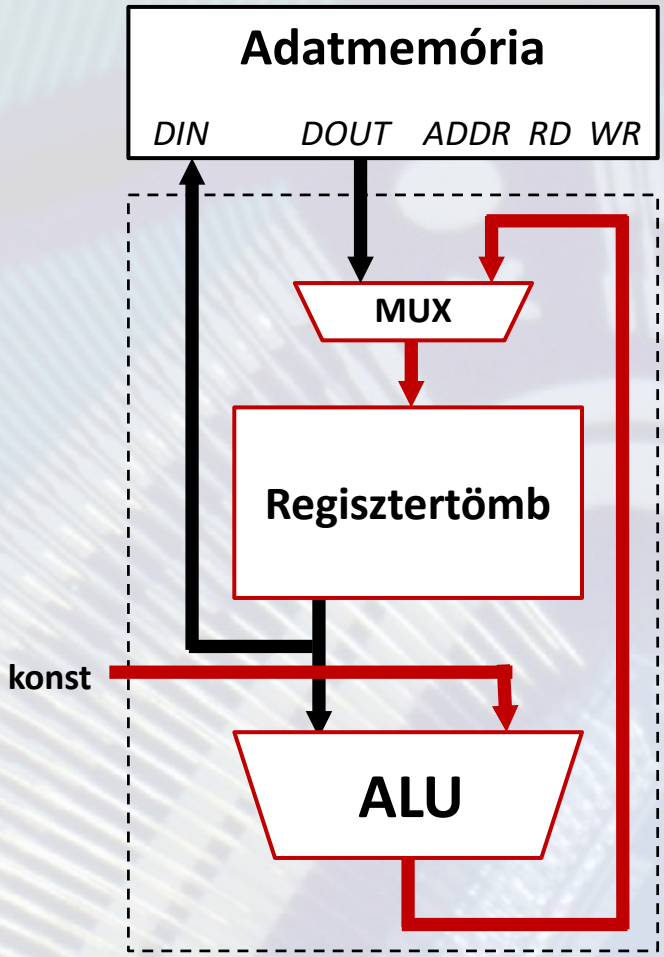
- ***Az ALU művelet egyik operandusa konstans is lehet***, nem csak egy regiszter. Ezt a vezérlő a MUX2-vel választja ki.
- ***Adatmozgatás regiszteről regiszterbe:*** A vezérlő MUX2-vel RdY-t MUX1 az ALU kimenetét választja ki és az ALU funciókód által beállított műveletet az, hogy az ALU kimenete az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



MiniRISC processzor adatstruktúra működése



Adat mozgatás regiszterből regiszterbe

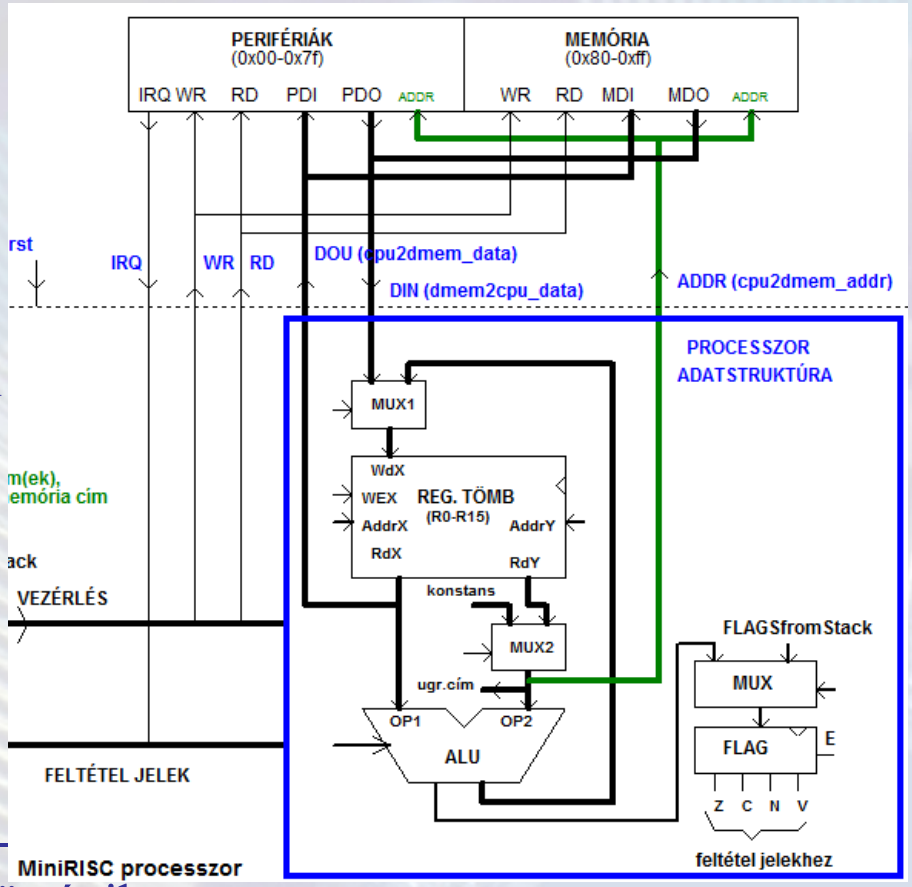


Adat mozgatás, regiszterbe konstans

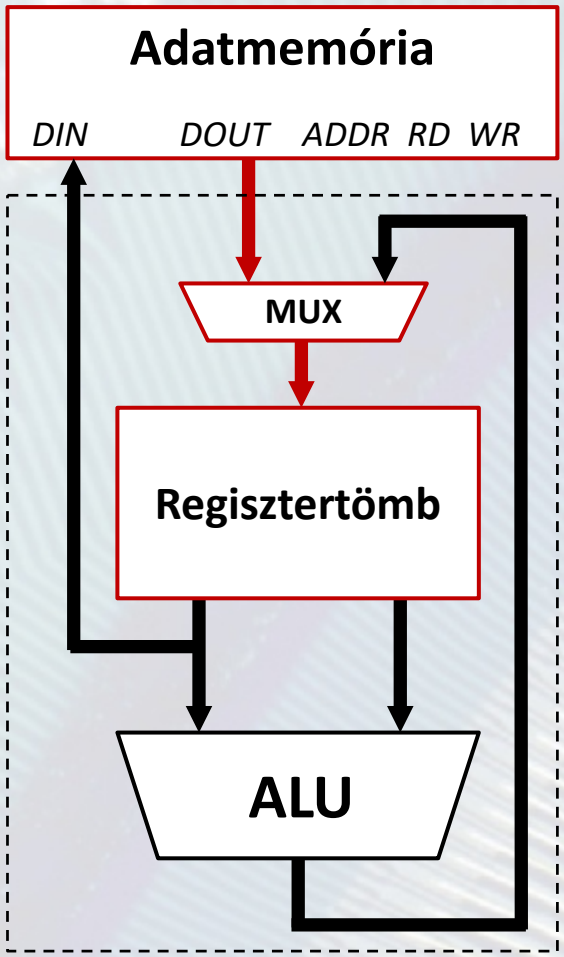
Processzor adatstruktúrák

A Mini RISC processzor adastruktúrája

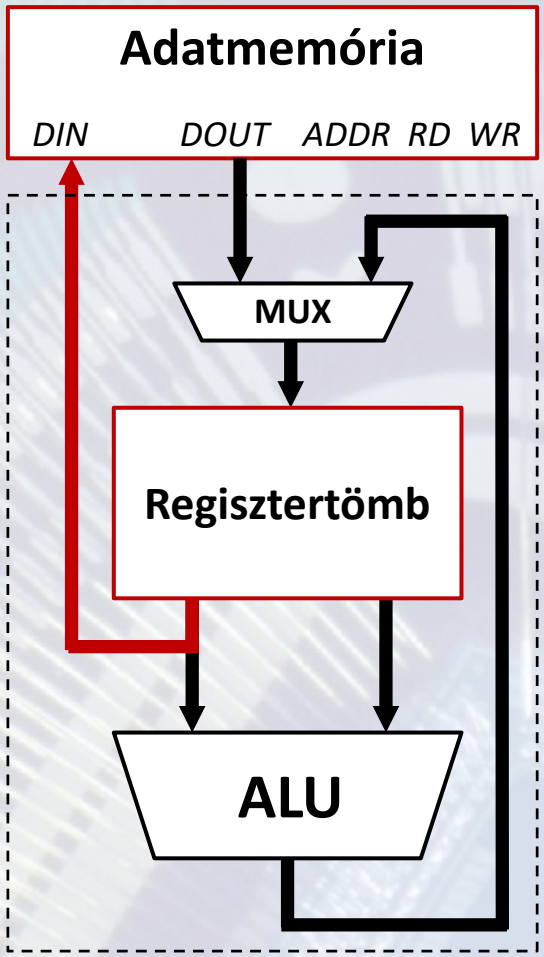
- ***Adatmozgatás regiszterből memóriába:*** Ilyenkor a regisztertömb RdX kimenete adja a processzor adat-busz kimenetét. ***A memória címe lehet konstans*** (ami a processzor utasítás kódjából származik, lásd később) és ***lehet az egyik regiszter tartalma*** (indirekt cím). Ezek közül a vezérlő MUX2-vel választ.
- ***Adatmozgatás memóriából regisz-terbe***: 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

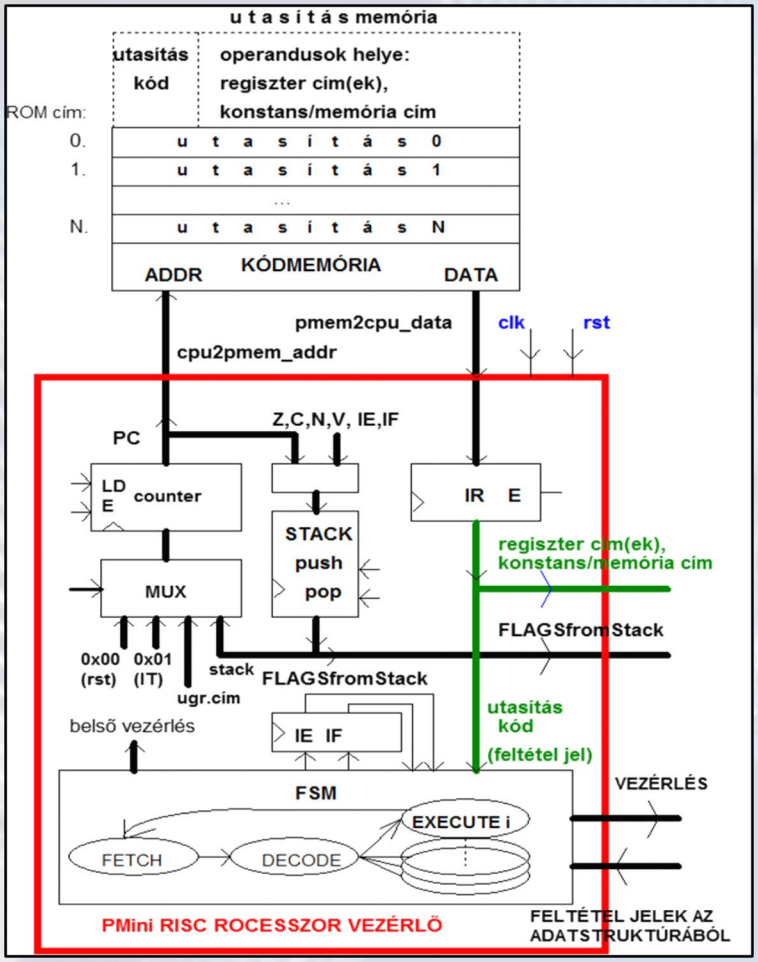
A processzor vezérlés egy konkrét megvalósítását az oktatási célú 8 bites MiniRISC processzoron mutatjuk be.

- ***A processzor adatstruktúra vezérlője egy hagyományos állapotgépből (FSM) és ezt kiegészítő saját adatstruktúrából áll.***
- ***A vezérlő utasítás vezérelt, az utasításokat az utasítás memóriából olvassa.***
- ***Az utasítás a következő információkat tartalmazza az utasítás egyes bitmezőiben:***
  A végrehajtandó művelet kódja (mit kell csinálni), operandus(ok) elhelyezkedése vagy konstans operandus értéke
- Az ***utasítás memóriát egy tölthető számláló***, a ***PC*** (Program Counter) ***címzi***.
- ***Az utasítás feldolgozása 3 fázisban*** (ciklusban) történik:
  - **FETCH** (utasítás ***beolvasása*** az IR utasítás regiszterbe),
  - **DECODE** (utasítás ***dekódolása***, utasítás kódjának felismerése)
  - **EXECUTE** (utasítás ***végrehajtása***, ***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) műveletthez 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).

A Mini RISC processzor vezérlőjének blokkvázlata



Processzor vezérlés

Processzor vezérlés, általános tulajdonságok

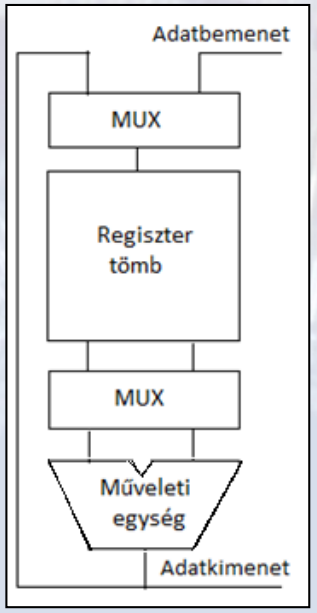
- ***Az utasítások végrehajtási sorrendje az ASM-nek megfelelően korlátozott, csak maximum 2 felé ágazás lehetséges.***
- Az ***utasítás végrehajtás normál sorrendje*** esetén egy utasítás után a következő (1-el nagyobb címen levő utasítás végrehajtása következik, ***PC = PC+1***).
- Vannak **feltétel nélküli elágazó (ugró) utasítások**. Ilyenkor egy az utasítás kódja alapján ***ismert címen*** folytatódik az utasítás végrehajtás (***PC = Ugrási cím***, ami egy konstans).
- 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.***

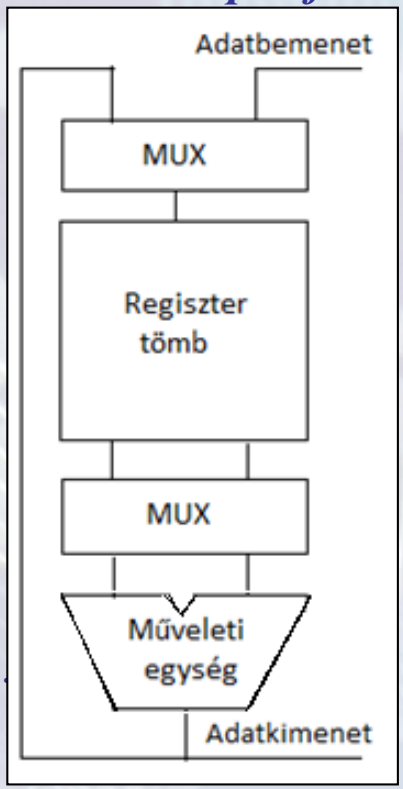
Általános tulajdonságok:

- ***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, HALT,...)
- **Minden művelet a szabványos adatméreten**
  - 8 / 16 / 32 / 64 bit, az adott rendszer jellemzője
  - **Nagyobb adatméret:** Átvitelbittel kiterjesztett műveletvégzés

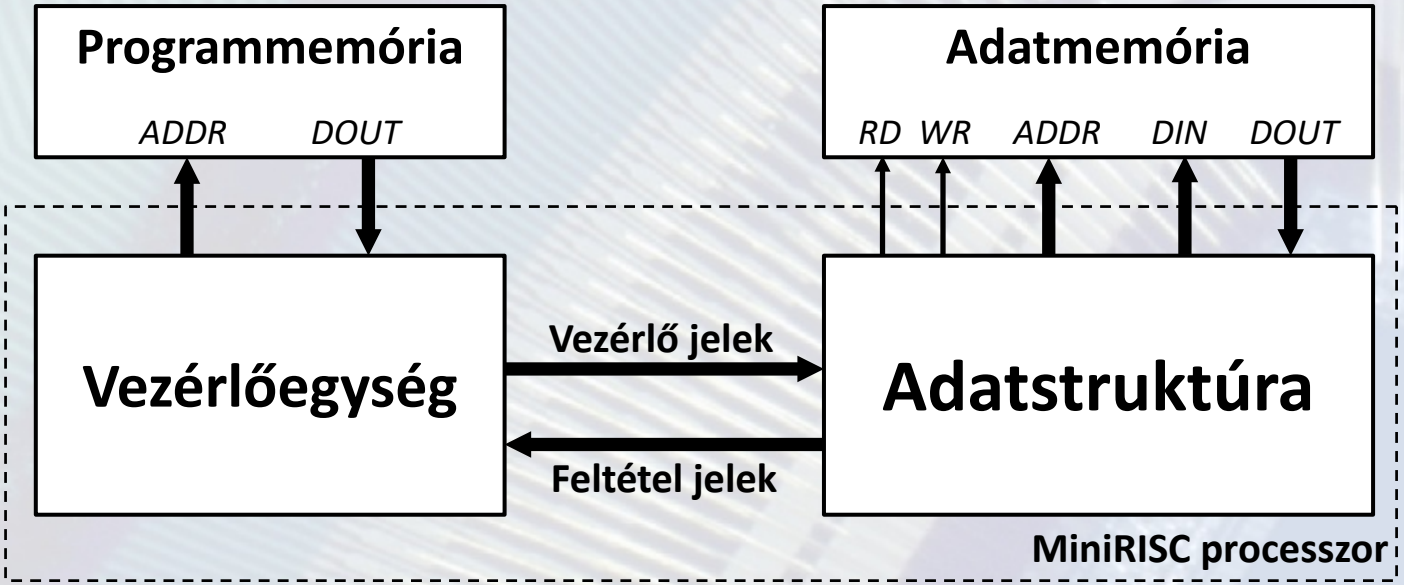


Összegzés

- **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).

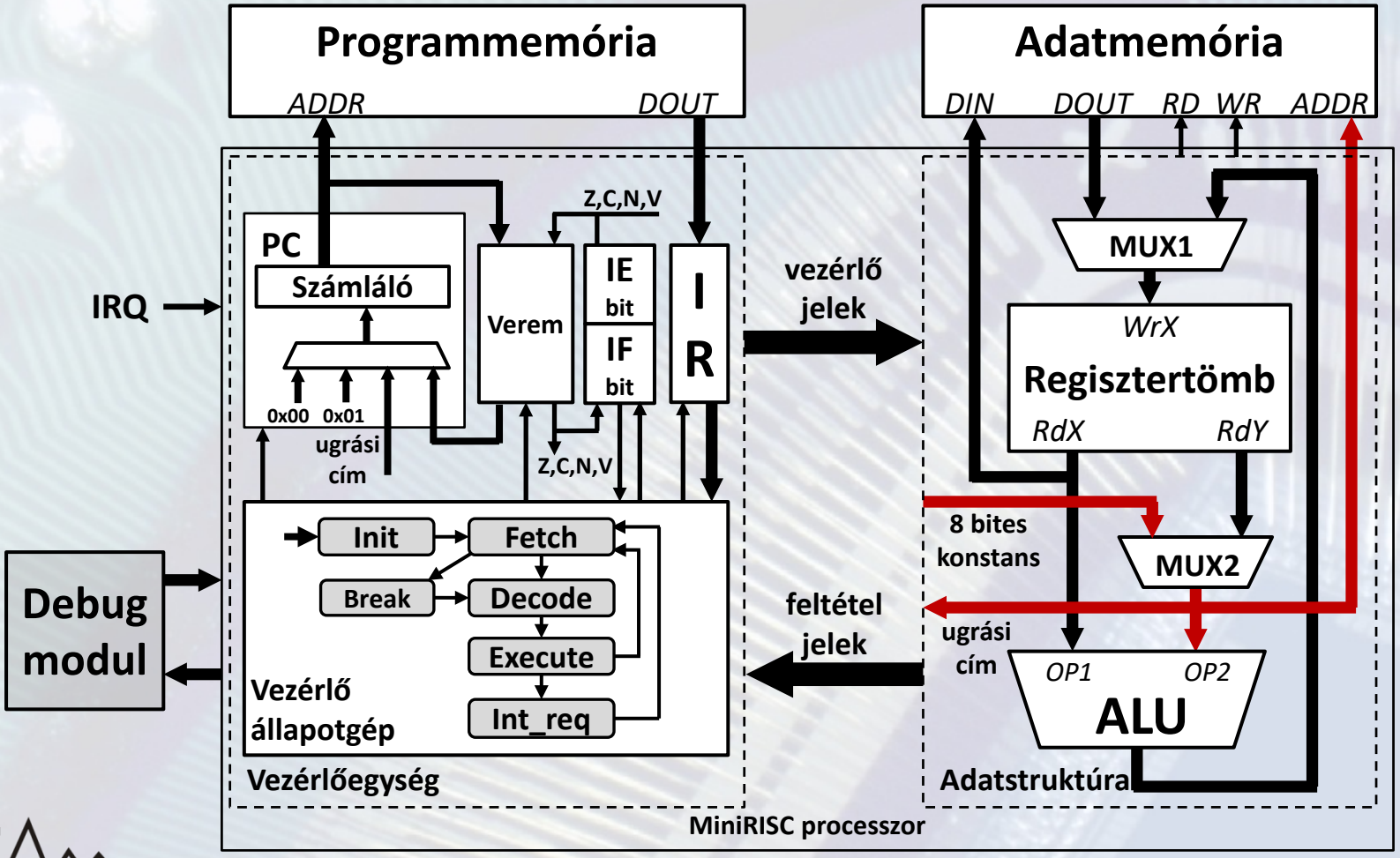
MiniRISC processzor

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



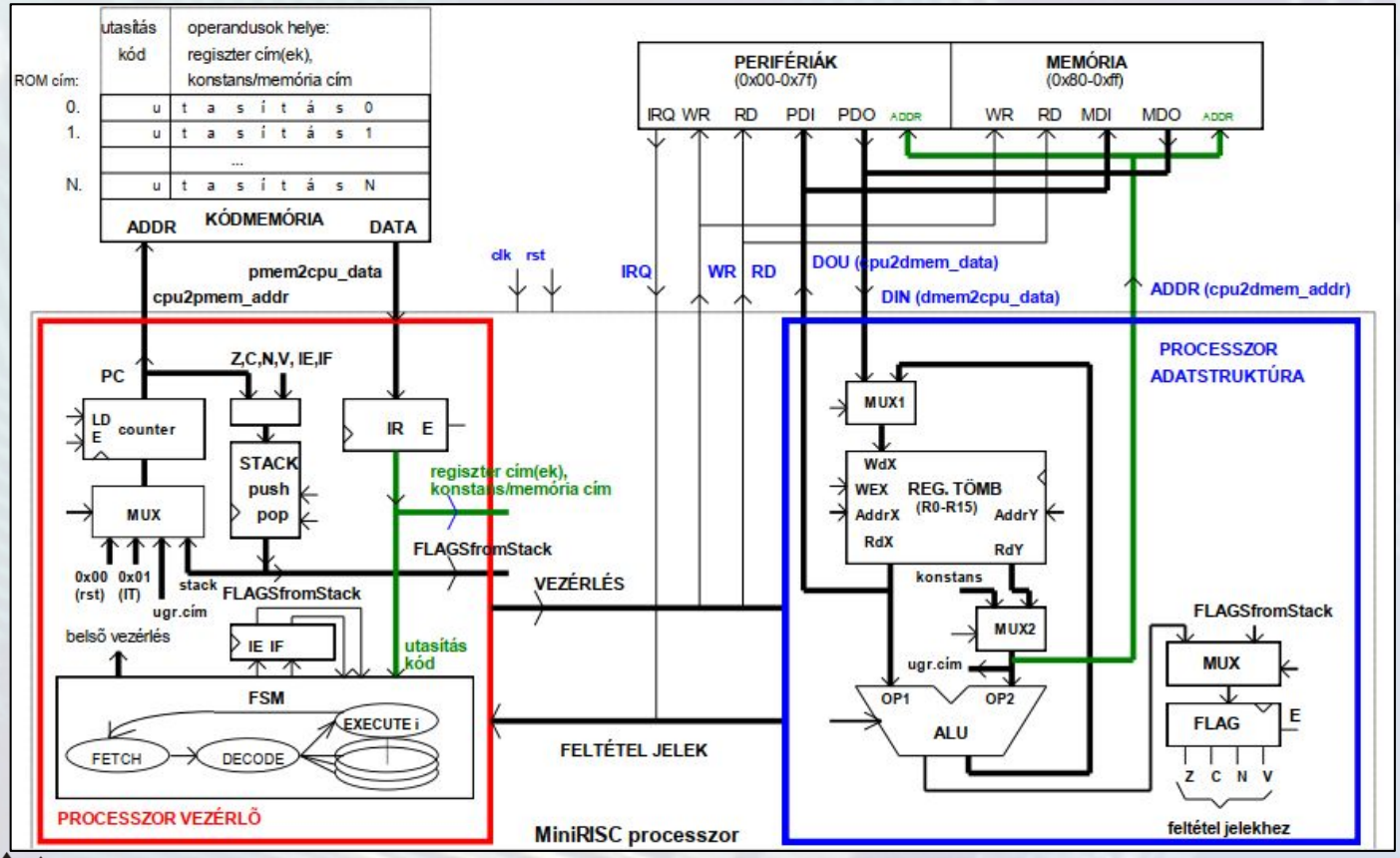
MiniRISC processzor blokkvázlata

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



Kiegészítő jegyzetek
A Mini RISC utasítás feldolgozás

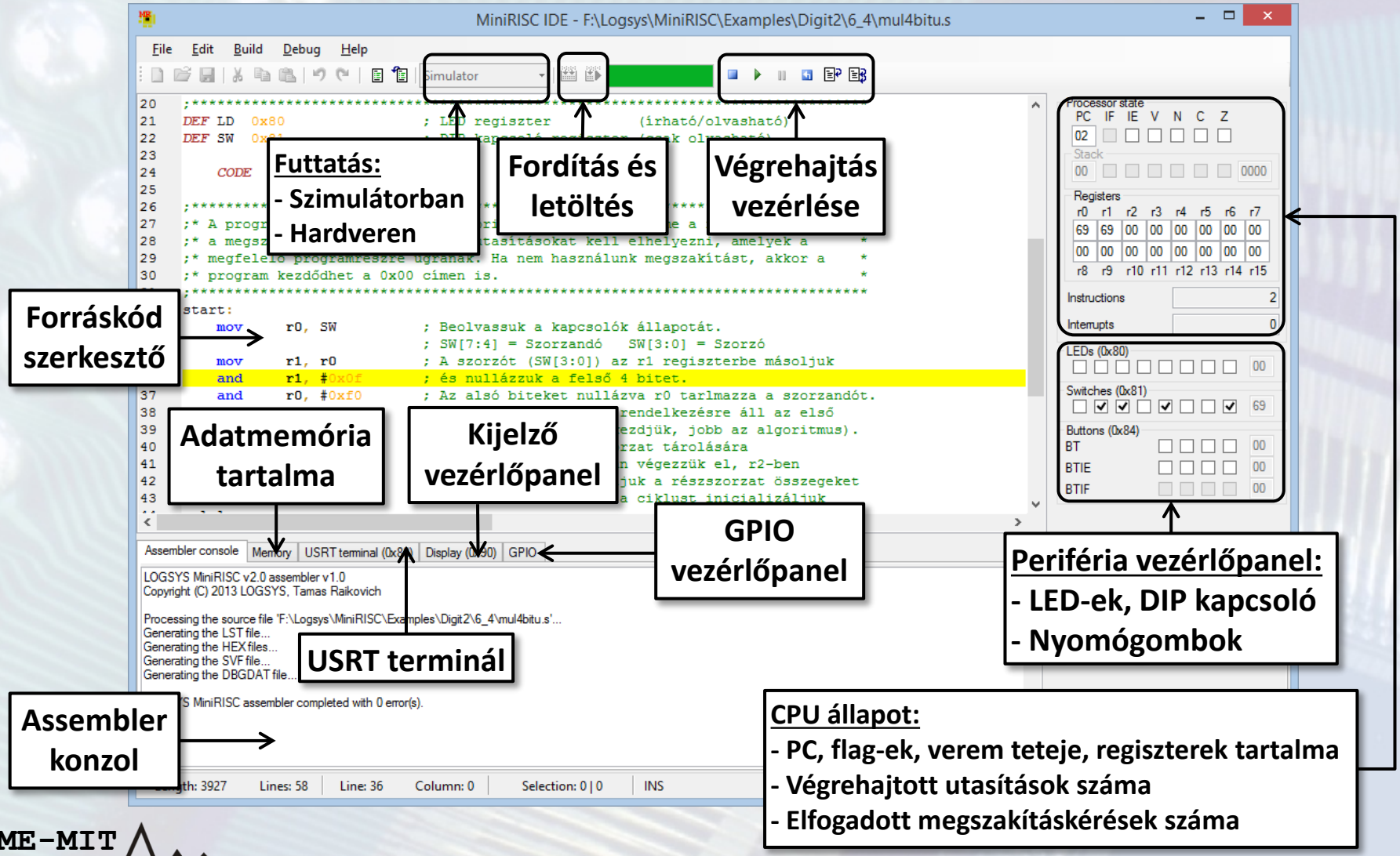
- **A Mini RISC utasítás feldolgozás rajzos magyarázata az azonos nevű kiegészítő jegyzetben található.**



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

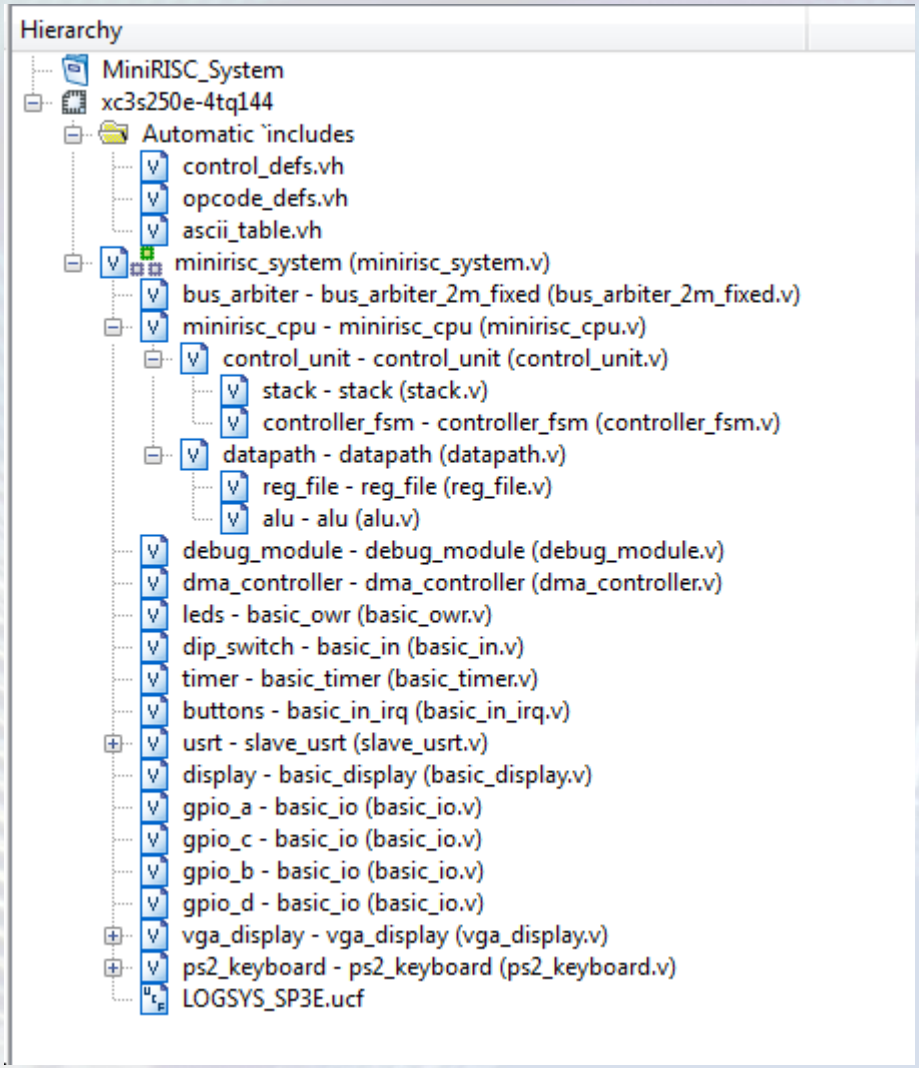


Digitális Technikai
8. EA vége

A további diák a MiniRISC vezérlő Verilog HDL megvalósításának részleteit ismertetik a LOGSYS Spartan3E250 kártyán, demonstrálják a HW működését egy szimulációval és bemutatják a MiniRISC assembler program tulajdonságait és használatát parancssorból

MiniRISC processzor Verilog HDL

- **A Verilog HDL projekt kicsit „összetettebb” az eddig megismerteknél, de egy teljes processzoros rendszer esetén ez természetes**
- **A fő komponensek:**
  MiniRISC CPU:
  vezérlés + adatstruktúra
  Perifériák:
  Külső, belső egységek
- **És ez még csak a HW…**



MiniRISC processzor implementáció

- **A szintézis szerint a MiniRISC processzor kb. az FPGA 50%-át foglalja el**

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	641	4,896	13%	
Number of 4 input LUTs	2,321	4,896	47%	
Number of occupied Slices	1,414	2,448	57%	
Number of Slices containing only related logic	1,414	1,414	100%	
Number of Slices containing unrelated logic	0	1,414	0%	
Total Number of 4 input LUTs	2,432	4,896	49%	
Number used as logic	1,624			
Number used as a route-thru	111			
Number used for Dual Port RAMs	604			
Number used for 32x1 RAMs	64			
Number used as Shift registers	29			
Number of bonded IOBs	68	108	62%	
Number of RAMB16s	10	12	83%	
Number of BUFGMUXs	1	24	4%	
Number of BSCANs	1	1	100%	
Average Fanout of Non-Clock Nets	4.15			

- **Az időzítési adatok alapján a szükséges 16MHz-es sebesség bőven teljesül**

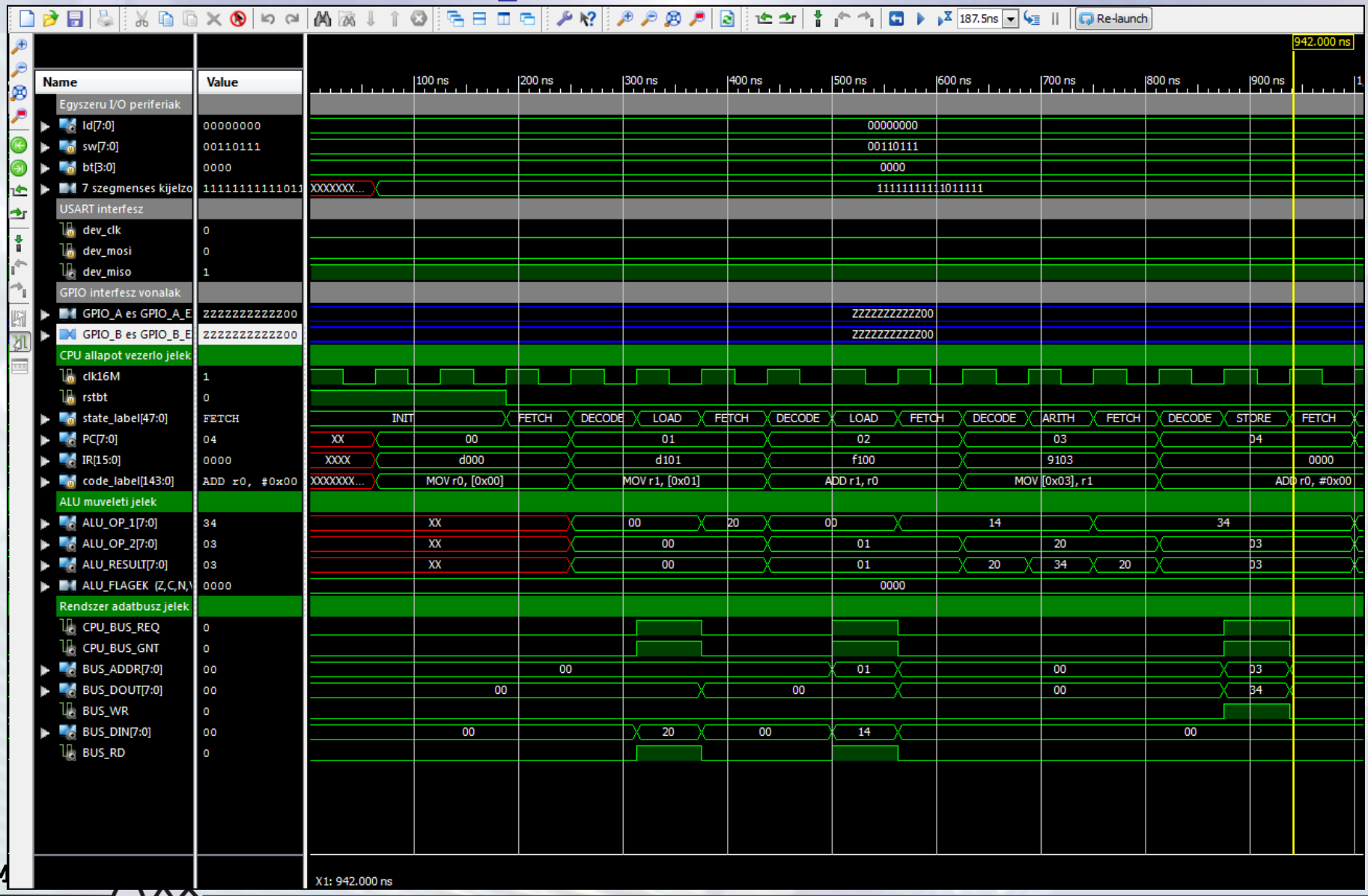
```
Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 14.545ns (Maximum Frequency: 68.752MHz)
   Minimum input arrival time before clock: 8.353ns
   Maximum output required time after clock: 7.094ns
   Maximum combinational path delay: No path found
```

MiniRISC processzor szimuláció

- **A szimulációt támogató fájlok**
  - A minirisc_system_TF.v szimulációs tesztkörnyezet
  - A MiniRISC_alap.wcfg szimulációs hullámforma ablak konfigurációs fájl
- **A szimuláció menete:**
  - Az aktuális code.hex és data.hex fájlok bemásolása
  - Az ISE átkapcsolása szimulációs módba
  - A minirisc_system_TF.v fájl kiválasztása
  - Az ISim szimulátor indítása viselkedési módban
  - Szimulációs idő beállítása 187,5ns-ra == 1 utasítás

MiniRISC processzor szimuláció



MiniRISC processzor szimuláció

- **Megfigyelhető a belső működés, tetszőleges részletességgel (de erre ritkán vagyunk kíváncsiak)**
  - Vezérlő állapota, PC, IR tartalma
  - A FETCH–DECODE–EXEC ciklus
  - Utasításkód alapján
    - az utasítás mnemonik megjelenítése, disassembler funkció, (bináris kódból utasítás neve, paramétereinek értelmezése), de címkék, azonosítók/változónevek nincsenek
  - ALU operandusok és a művelet eredménye
  - Flagek értéke (Z,C,N,V)
  - Busz adatátvitel paraméterei, iránya és időzítése
  - Elemi perifériák (LD, SW, BT) értékei

MiniRISC processzor szimuláció

- **A valódi programok működése jellemzően sokkal hosszabb futásidőt igényel**
- **Az időDiagram itt nem áttekinthető, nem jelent segítséget, viszont minden részletében megfigyelhető a rendszer, ha szükséges**
- **Esetleg speciális perifériáknál SW-HW együttes fejlesztése**
- **Fontosabb egy jó programozói modell, ami a processzor műveleti szintjét, belső állapotát jeleníti meg (PC, program forráskód, regiszterek, memória)**

MiniRISC assembler

- **A MiniRISC processzor felhasználói programjait alacsonyszintű (gépközeli) programnyelven készíthetjük el és fordíthatjuk futtatható bináris kódra.**
- **Az assembly nyelven megírt programok lefordítása a LOGSYS MiniRISCv2-as assemblerrel lehetséges**
- **Konkrét programfejlesztési részletek az előadás végén**

MiniRISC kódgenerálás az assemblerrel

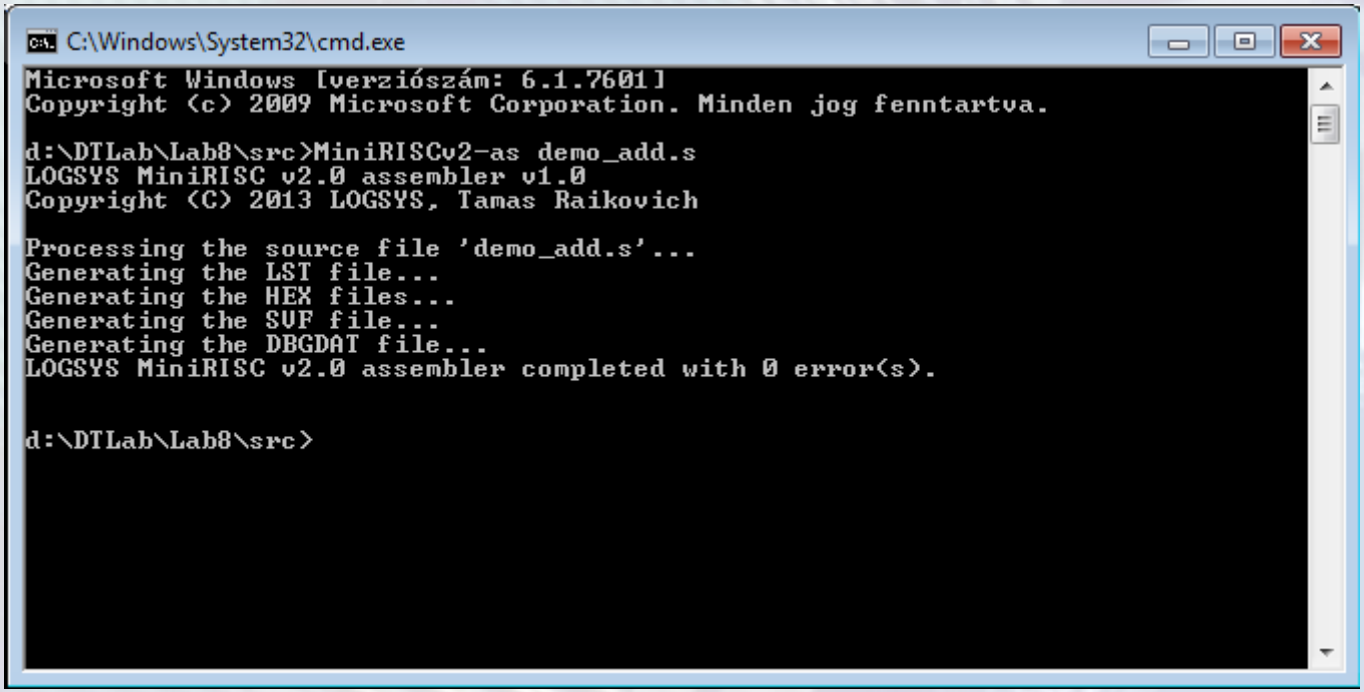
- **Futtatása parancssorból: *MiniRISCv2-as filename.s***
  - A ***filename.s*** assembly forrásfájlt beolvassa és ha nincs benne hiba, lefordítja és generálja a következő fájlokat
    - ***filename.lst*** assembler listafájl címekkel, címkékkel, azonosítókkal, utasításkódokkal
    - ***code.hex*** szöveges fájl a Verilog kódból történő programmemória inicializáláshoz
    - ***data.hex*** szöveges fájl a Verilog kódból történő adatmemória inicializáláshoz
    - ***filename.svf*** a LOGSYS GUI-val letölthető, a program- és adatmemória tartalmát inicializáló SVF fájl
    - ***filename.dbgdat*** a debug információkat tartalmazó fájl
  - Az esetleges hibaüzenetek a konzolon jelennek meg
- **A MiniRISC assembler a MiniRISC IDE integrált részét képezi, a parancssorból való futtatása ezért nem gyakori, de egy példa kedvéért bemutatjuk.**

MiniRISC kódgenerálás az assemblerrel

- **Futtatása parancssorból:**

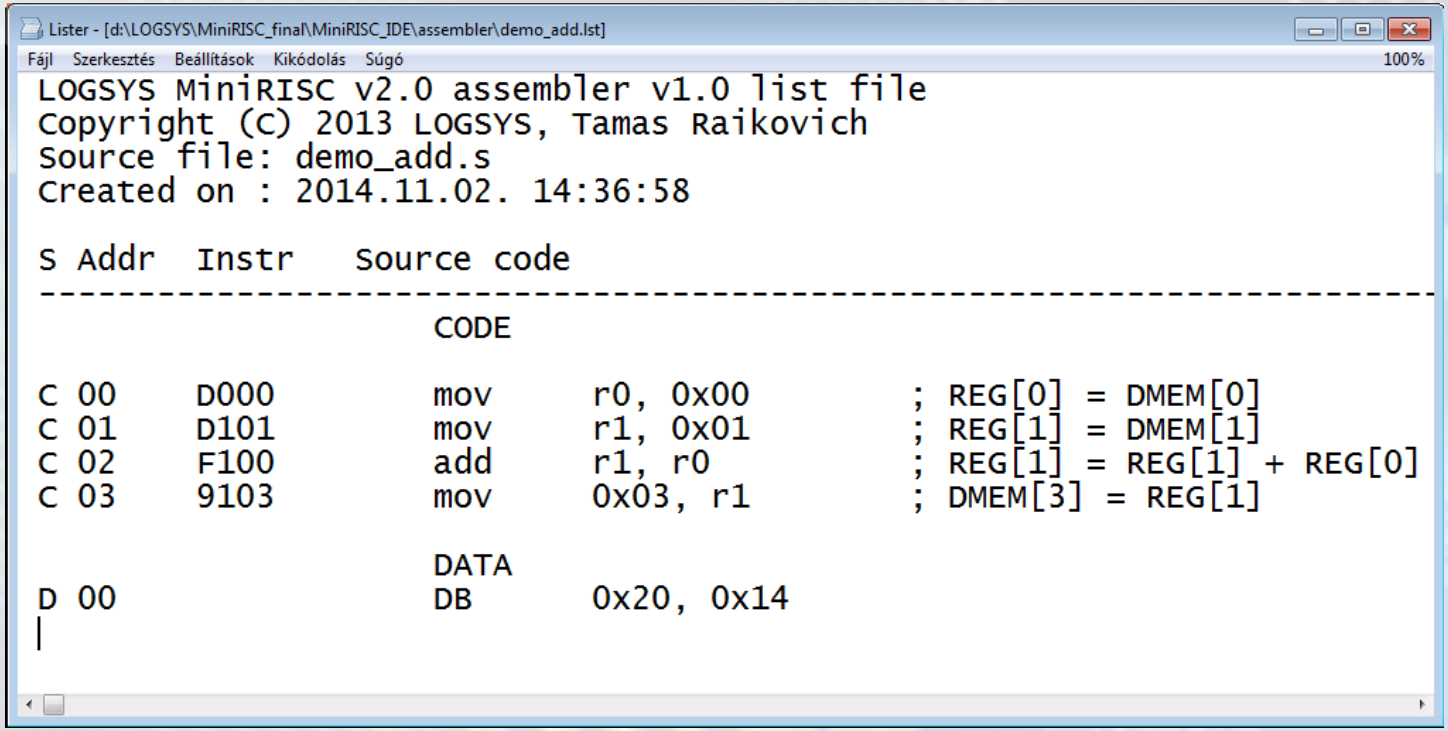
  MiniRISCv2-as filename.s

  – *Forrásprogram: demo_add.s*



MiniRISC kódgenerálás az assemblerrel

- **Fordítás eredménye:**
  - Generált lista fájl: demo_add.lst



MiniRISC kódgenerálás az assemblerrel

- **A generált memória tárgykód fájlok:**
  - Programmemória kódfájl: code.hex
    - 256 db 16 bites utasításkód
  - Adatmemória kódfájl: data.hex
    - 128 db 8 bites adat
  - A kész kódfájlokat elérhetővé tesszük a Verilog HDL projekt számára (átmásoljuk az ISE projekt könyvtárba)

```
initial
        $readmemh("code.hex", prg_mem, 0, 255);
```

```
initial
        $readmemh("data.hex", data_mem, 0, 127);
```

code.hex

```
D000
D101
F100
9103
0000
0000
....
....
....
0000
0000
0000
0000
0000
0000
0000
0000
0000
0000
0000
```

data.hex

```
20
14
00
00
..
..
..
00
00
00
00
```