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

A MiniRISC processzor

Fehér Béla, Raikovich Tamás, Fejér Attila

BME MIT

Tartalom

MiniRISC processzor - Bevezetés

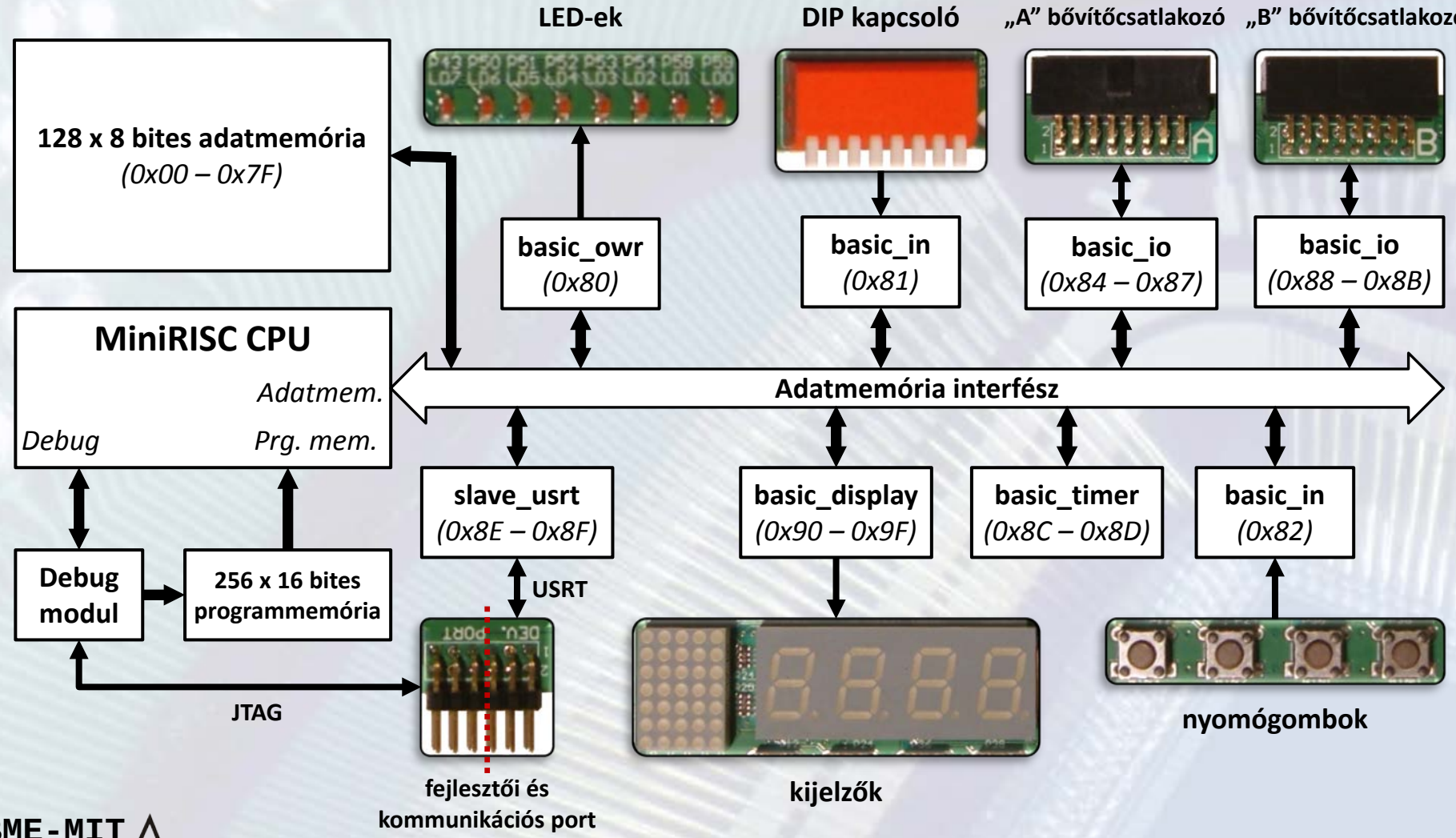
- **8 bites vezérlőegység egyszerű alkalmazásokhoz**
- **Jól illeszkedik a LOGSYS Spartan-3E FPGA kártya komplexitásához**
- **Egyszerű felépítés, kis erőforrásigény**
- **Harvard architektúra**
  - 256 x 16 bites programmemória
  - 256 x 8 bites adatmemória
- **Egyszerű RISC jellegű utasításkészlet**
  - Load/store architektúra
  - Műveletvégzés csak regisztereken
  - 16 x 8 bites belső regisztertömb

MiniRISC processzor - Bevezetés

- **Egyszerű RISC jellegű utasításkészlet**
  - Adatmozgató utasítások
  - Aritmetikai utasítások (+, -, összehasonlítás)
  - Logikai utasítások (AND, OR, XOR, bittesztelés)
  - Léptetési, forgatási és csere utasítások
  - Programvezérlési utasítások
- **Operandusok: két regiszter vagy regiszter és 8 bites konstans**
- **Abszolút és regiszter indirekt címzési módok**
- **Zero (Z), carry (C), negative (N), overflow (V) feltételbitek**
  - Feltételes ugró utasítások a teszteléshez
- **Szubrutinhívás 16 szintű hardveres verem használatával**
- **Programelágazás a teljes címtartományban**
- **Egyszintű, egyszerű megszakításkezelés**

MiniRISC mintarendszer - Bevezetés

(Egyszerűsített Blokkvázlat)

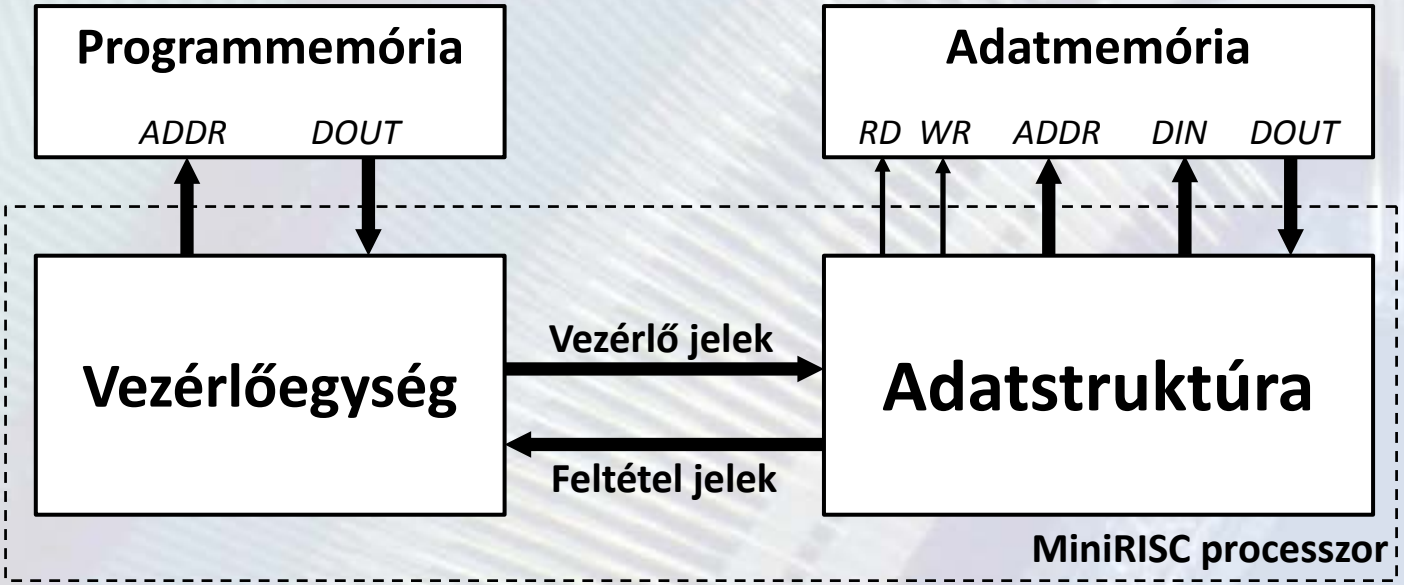


Tartalom

MiniRISC processzor - Felépítés

Felépítése követi az adatstruktúra-vezérlő szemléletet

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

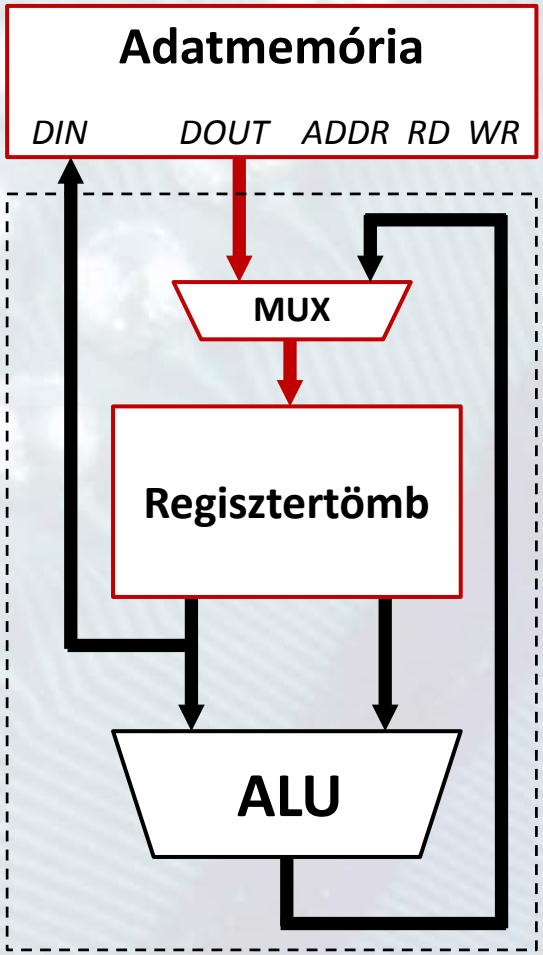


MiniRISC processzor – Felépítés
(Adatstruktúra)

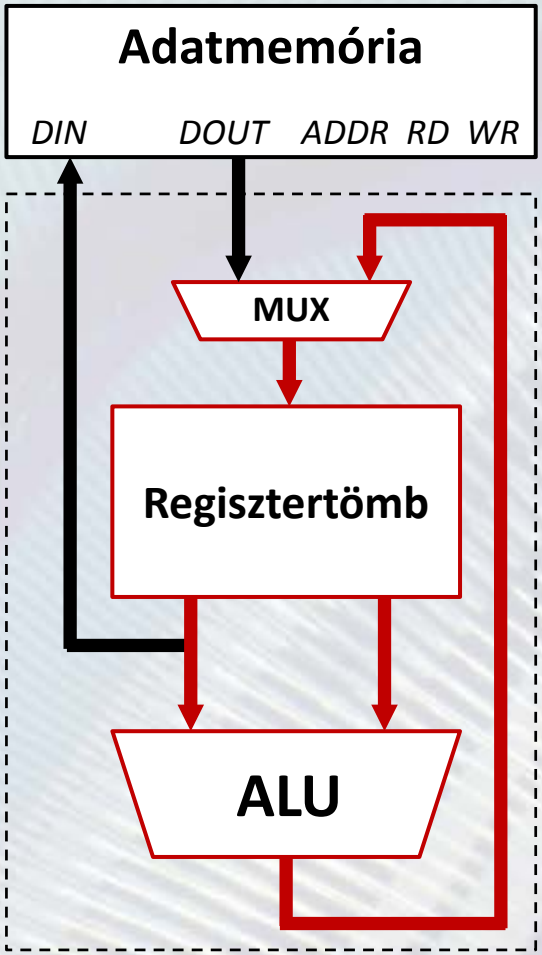
- **Az adatstruktúrában történik a műveletek végrehajtása az adatokon. Az adatfeldolgozás fő lépései a következők:**
  1. A feldolgozandó adatok betöltése
  2. Ezen adatok átalakítása
  3. Az eredmény elmentése
- **Az alap adatstruktúra ennek megfelelően:**
  - Olvassa és írja az adatmemóriát, ahol a fő adatok vannak
  - Tartalmaz regisztertömböt az adatok lokális tárolásához
  - Tartalmaz aritmetikai-logikai egységet (ALU-t) a lokális adatok átalakításához

MiniRISC processzor – Felépítés

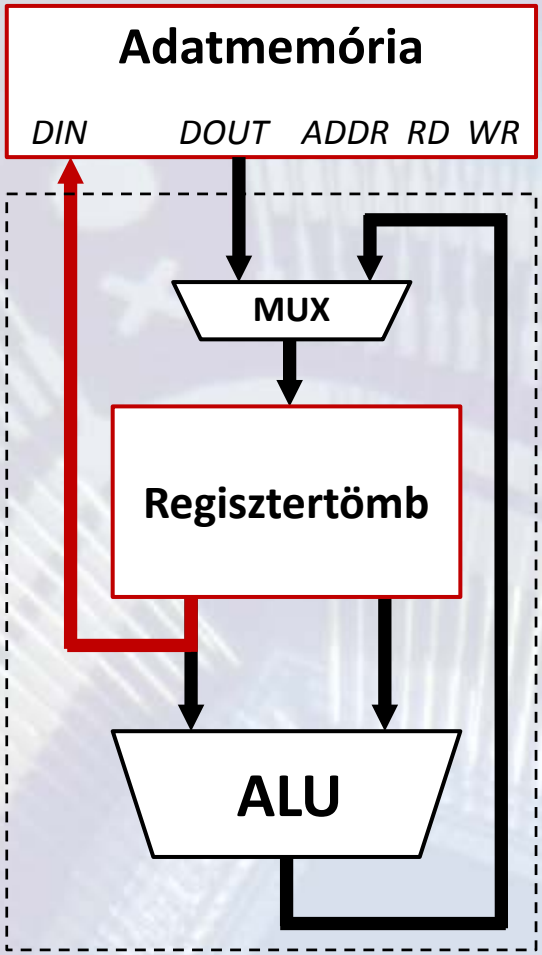
(Adatstruktúra)



Adatmemória olvasás (load)



Lokális adat átalakítása (ALU művelet)

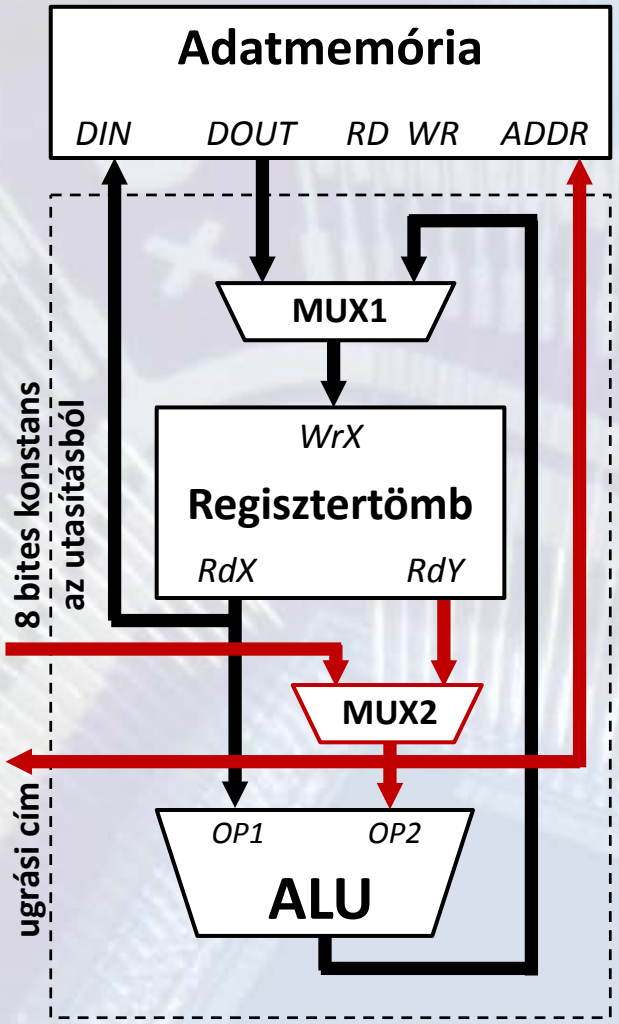


Adatmemória írás (store)

MiniRISC processzor – Felépítés

(A MiniRISC processzor adatstruktúrája)

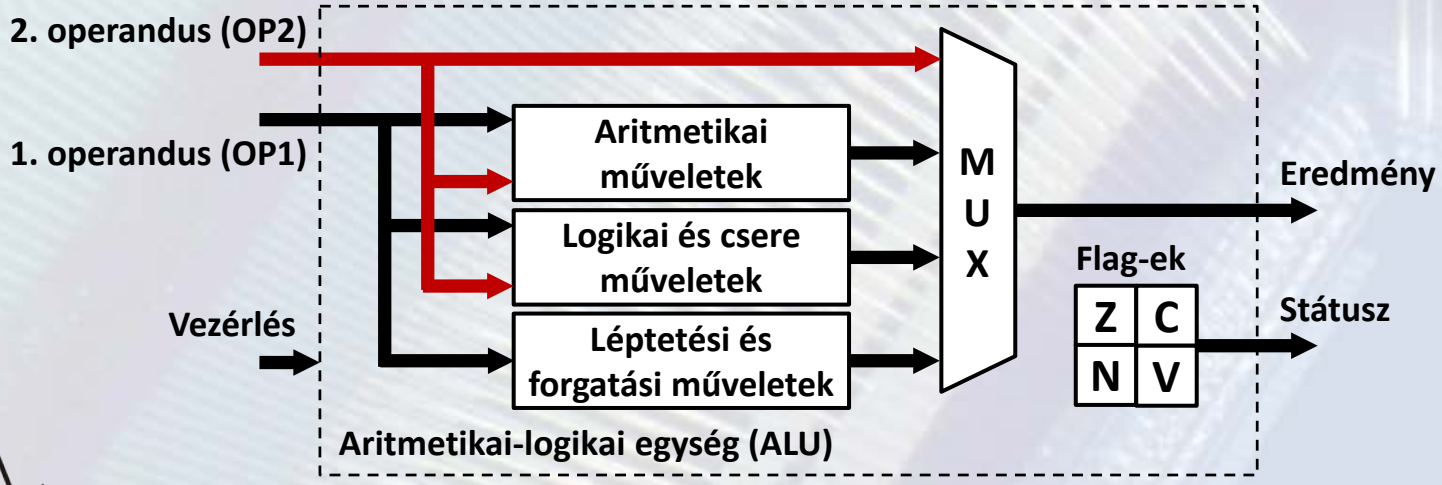
- **Az alap adatstruktúra bővített változata**
- **A regisztertömbbe írandó adatot kiválasztó multiplexer (MUX1)**
  - ALU eredmény vagy adatmemória
- **16 x 8 bites regisztertömb**
  - Két regisztercímes architektúra
- **Aritmetikai-logikai egység (ALU)**
- **A 2. ALU operandus kiválasztása (MUX2)**
  - Regiszter
  - 8 bites konstans az utasításból
- **Címzési mód kiválasztása (MUX2)**
  - Abszolút címzés: cím az utasításból
  - Indirekt címzés: cím a regiszterből



MiniRISC processzor – Felépítés

(A MiniRISC processzor adatstruktúrája - ALU)

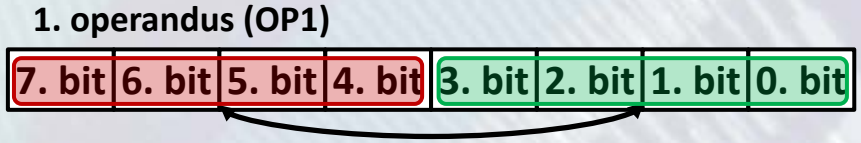
- **Az ALU műveleteket végez a lokális adatokon**
  - Adatmozgatás: nincs műveletvégzés, eredmény az OP2
  - Aritmetikai: összeadás és kivonás átvitel bittel vagy anélkül
  - Logikai: bitenkénti AND, OR, XOR
  - Léptetés és forgatás, csere
- **Státusz bitek: a műveletvégzés eredményéről adnak információt**
  - Zero (Z), carry (C), negative (N) és overflow (V) státusz bitek
  - Értéküket a feltételes ugró utasításokkal tesztelhető



MiniRISC processzor – Felépítés

(A MiniRISC processzor adatstruktúrája - ALU)

- **Logikai műveletek**
  - Bitenkénti AND, OR és XOR műveletek
- **Csere**
  - Az 1. operandus alsó és felső 4 bitjének megcserélése

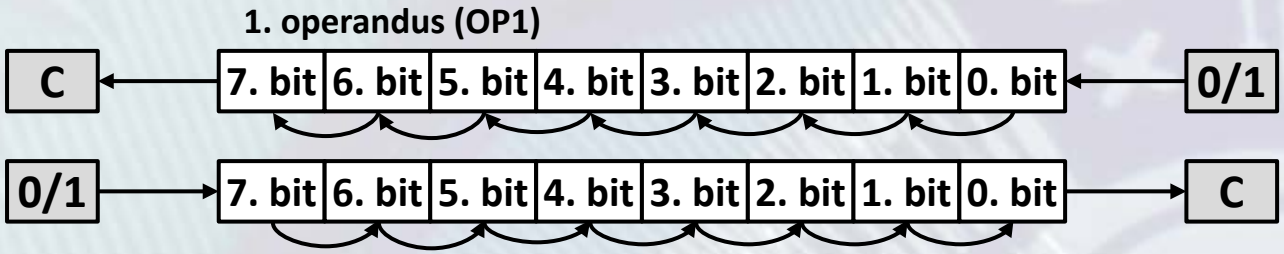


  - A logikai műveleteket elvégző blokkban van megvalósítva
    - A multiplexer negyedik bemenete felhasználható a cseréhez
    - Ugyanazon státusz bitek módosulnak (Z és N) → azonos vezérlés

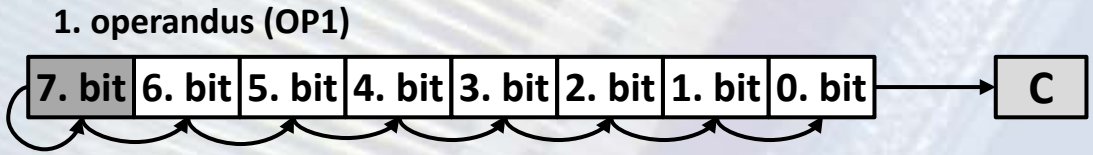
MiniRISC processzor – Felépítés

(A MiniRISC processzor adatstruktúrája - ALU)

- **Logikai shiftelés**
  - A logikai shiftelés történhet balra vagy jobbra
  - A beléptetett bit lehet 0 vagy 1, a kilépő bit a carry (C) flag-ba kerül



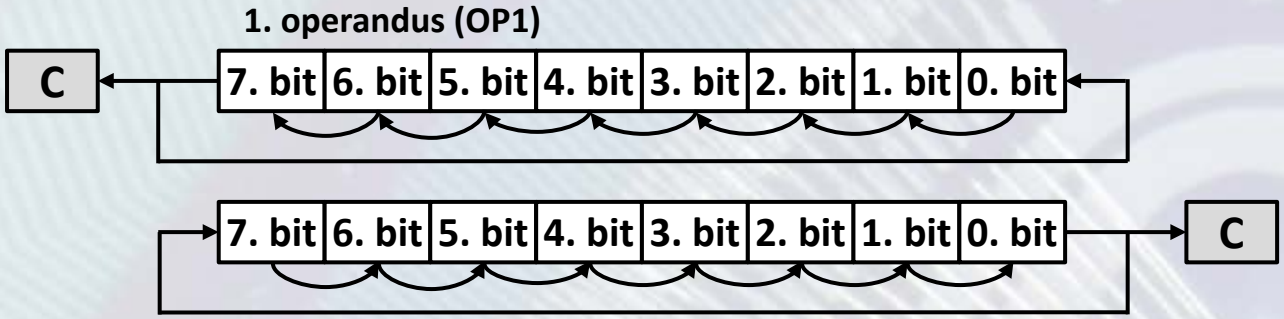
- **Aritmetikai shiftelés jobbra**
  - Előjeles számok jobbra léptetése esetén az előjel bitet (MSb) helyben kell hagyni, hogy helyes eredményt kapjunk
  - A kilépő bit a carry (C) flag-ba kerül
  - Külön balra történő aritmetikai shiftelés nincs, mert ez megegyezik a balra történő logikai shifteléssel



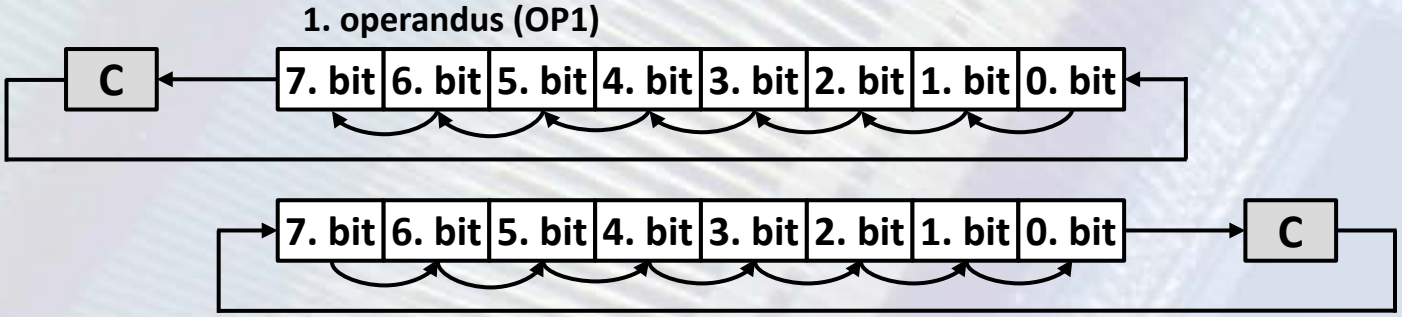
MiniRISC processzor – Felépítés

(A MiniRISC processzor adatstruktúrája - ALU)

- **Normál forgatás**
  - A forgatás történhet balra vagy jobbra
  - A kilépő bit beléptetésre kerül a másik oldalon
  - A carry (C) flag a kilépő bit értékét veszi fel



- **Forgatás a carry (C) flag-en keresztül**



MiniRISC processzor – Felépítés

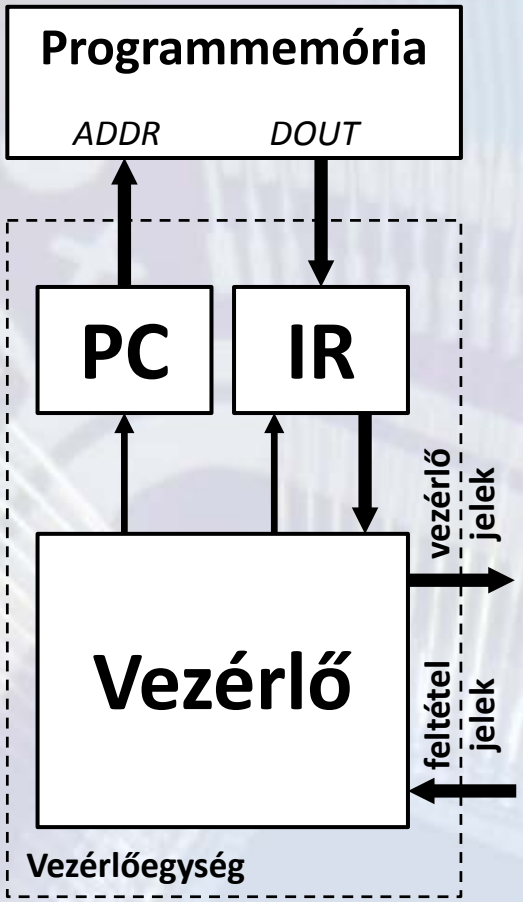
(A MiniRISC processzor adatstruktúrája - ALU)

- **Az ALU státusz bitek a művelet eredményéről adnak információt**
- ***Zero bit (Z)***
  - Jelzi, ha az ALU művelet eredménye nulla
  - Az c adatmozgatás nem változtatja meg az értékét
- ***Carry bit (C)***
  - Aritmetikai művelet esetén jelzi, ha átvitel történt
  - Shiftelés vagy forgatás esetén felveszi a kiléptetett bit értékét
- ***Negative bit (N)***
  - Kettes komplemens előjelbit, az eredmény MSb-je (7. bitje)
  - Az egyszerű adatmozgatás nem változtatja meg az értékét
- ***Overflow bit (V)***
  - A kettes komplemens túlcsordulást jelzi
    - Az aritmetikai művelet eredménye nem fér el 8 biten
  - Detektálása: az operandusok előjel bitje azonos, de az eredmény előjel bitje ettől eltérő (a C_{in7} ***xor*** C_{out7} nem használható, mert nem lehet hozzáférni az MSb bemeneti átvitel bitjéhez)

MiniRISC processzor – Felépítés

(Vezérlőegység)

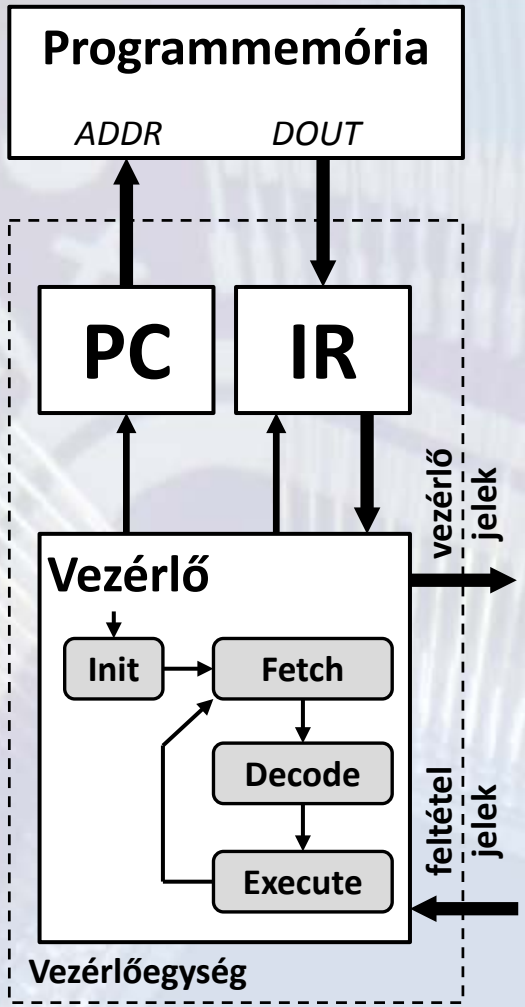
- **Példa: DMEM[3] = DMEM[0] + DMEM[1], ez négy adatstruktúra műveletet igényel:**
  1. REG[0] = DMEM[0] (load)
  2. REG[1] = DMEM[1] (load)
  3. REG[1] = REG[0] + REG[1] (ALU művelet)
  4. DMEM[3] = REG[1] (store)
- **Utasítás: a processzor által végrehajtható művelet**
- **Program: utasítások sorozata**
  - A végrehajtandó feladatot a processzor által támogatott utasításokra kell lebontani
- **A programot a programmemória tárolja**
- **A vezérlőegység beolvassa az utasításokat és végrehajtja azokat az adatstruktúrán**
  - ***Programszámláló (Program Counter, PC):*** a beolvasandó utasítás címét állítja elő
  - ***Utasításregiszter (Instruction Register, IR):*** a beolvasott utasítást tárolja



MiniRISC processzor – Felépítés

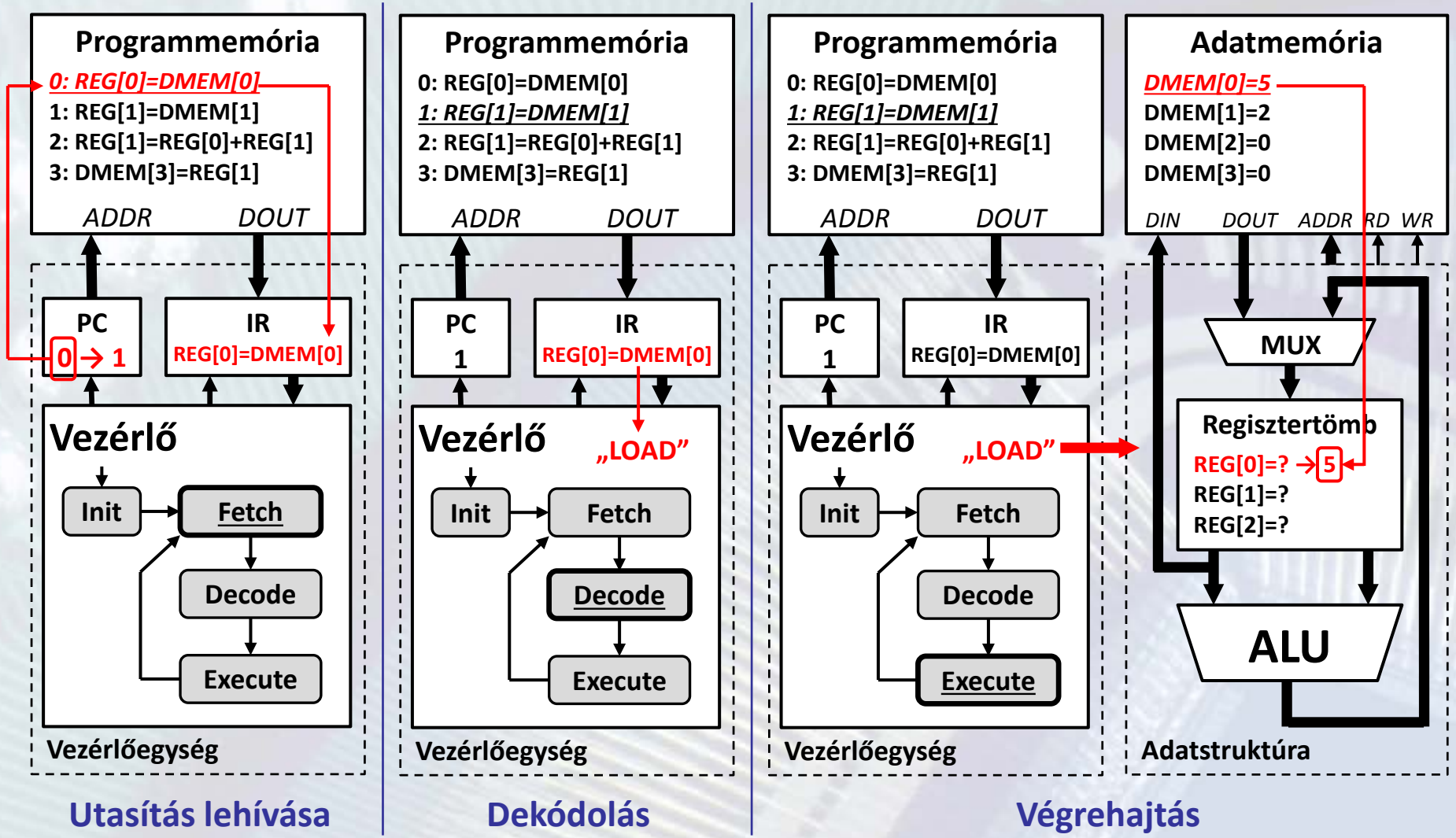
(Vezérlőegység)

- **Minden egyes utasítás végrehajtásánál a vezérlőegységnek a következő lépéseket kell elvégeznie:**
  - ***Lehívás (fetch):*** az utasítás beolvasása a programmemóriából és a PC növelése
  - ***Dekódolás (decode):*** a művelet és az operandusok meghatározása
  - ***Végrehajtás (execute):*** az utasításhoz tartozó művelet végrehajtásra kerül az adatstruktúrán
- **A vezérlő lehet például egy állapotgép**
  - A fenti lépésekhez a vezérlő állapotgép egy-egy állapota rendelhető hozzá
  - Ekkor egy utasítás végrehajtása három órajelciklust igényel



MiniRISC processzor – Felépítés

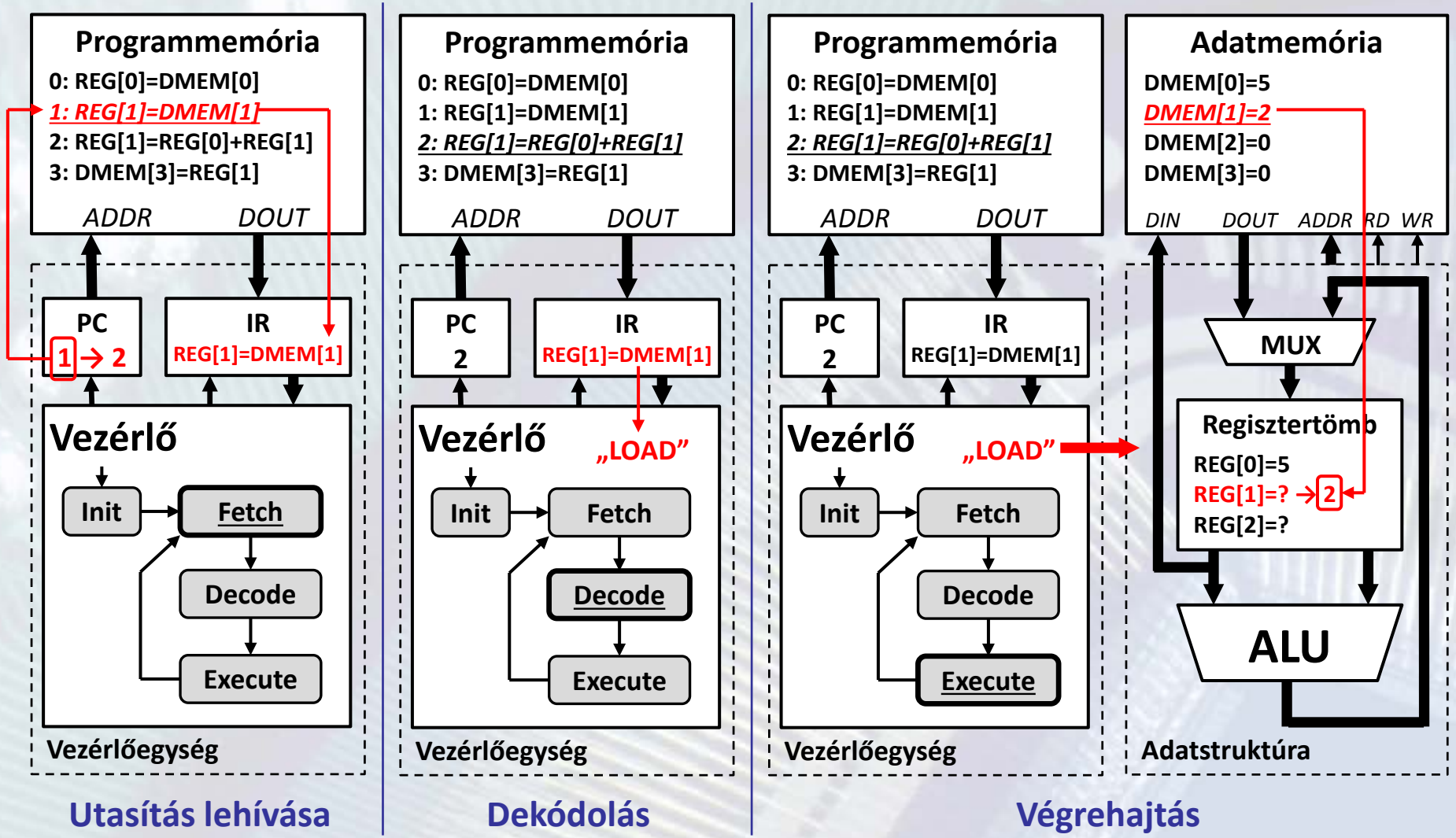
(Vezérlőegység)



Utasítás lehívása | Dekódolás | Végrehajtás

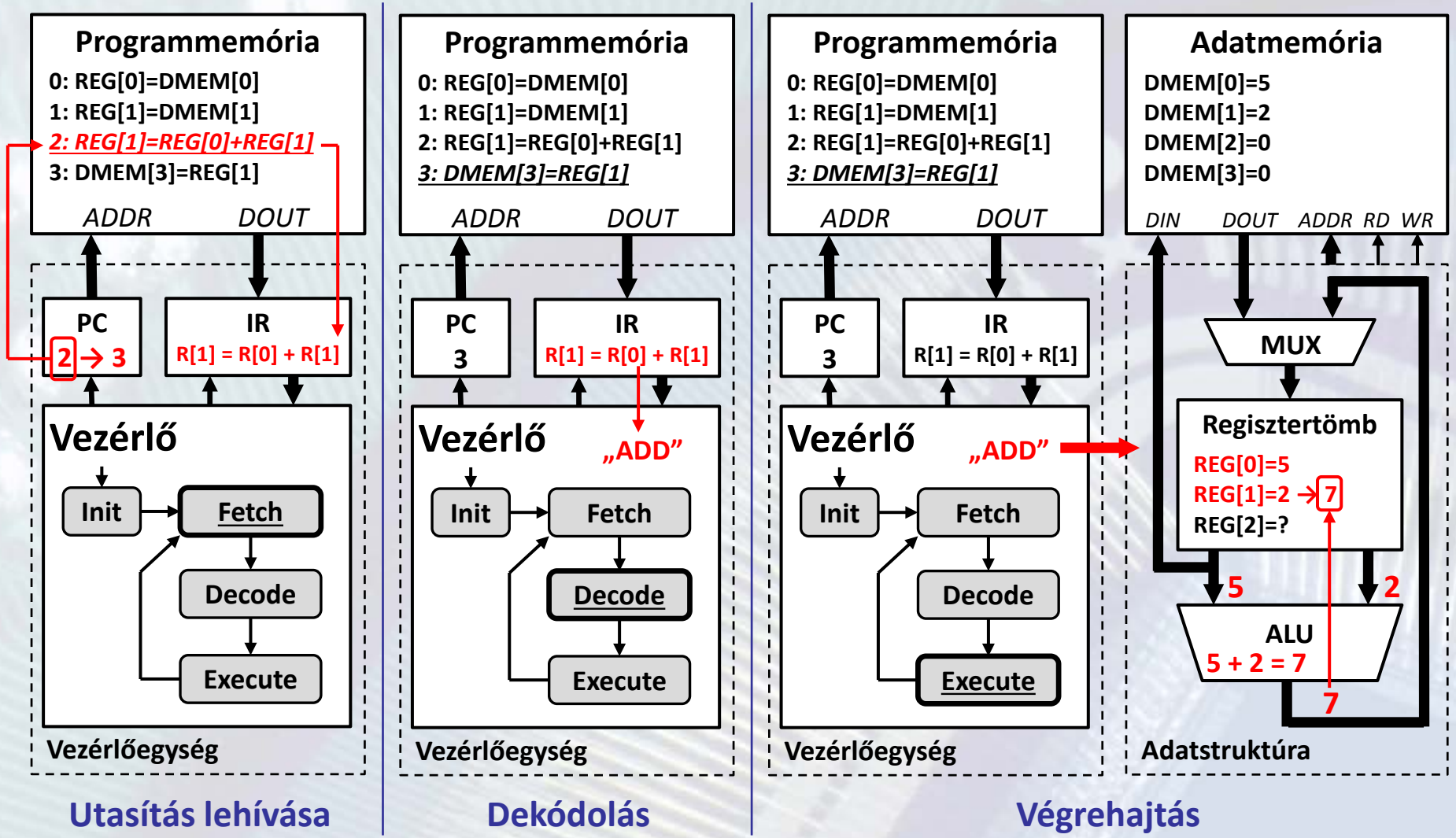
MiniRISC processzor – Felépítés

(Vezérlőegység)



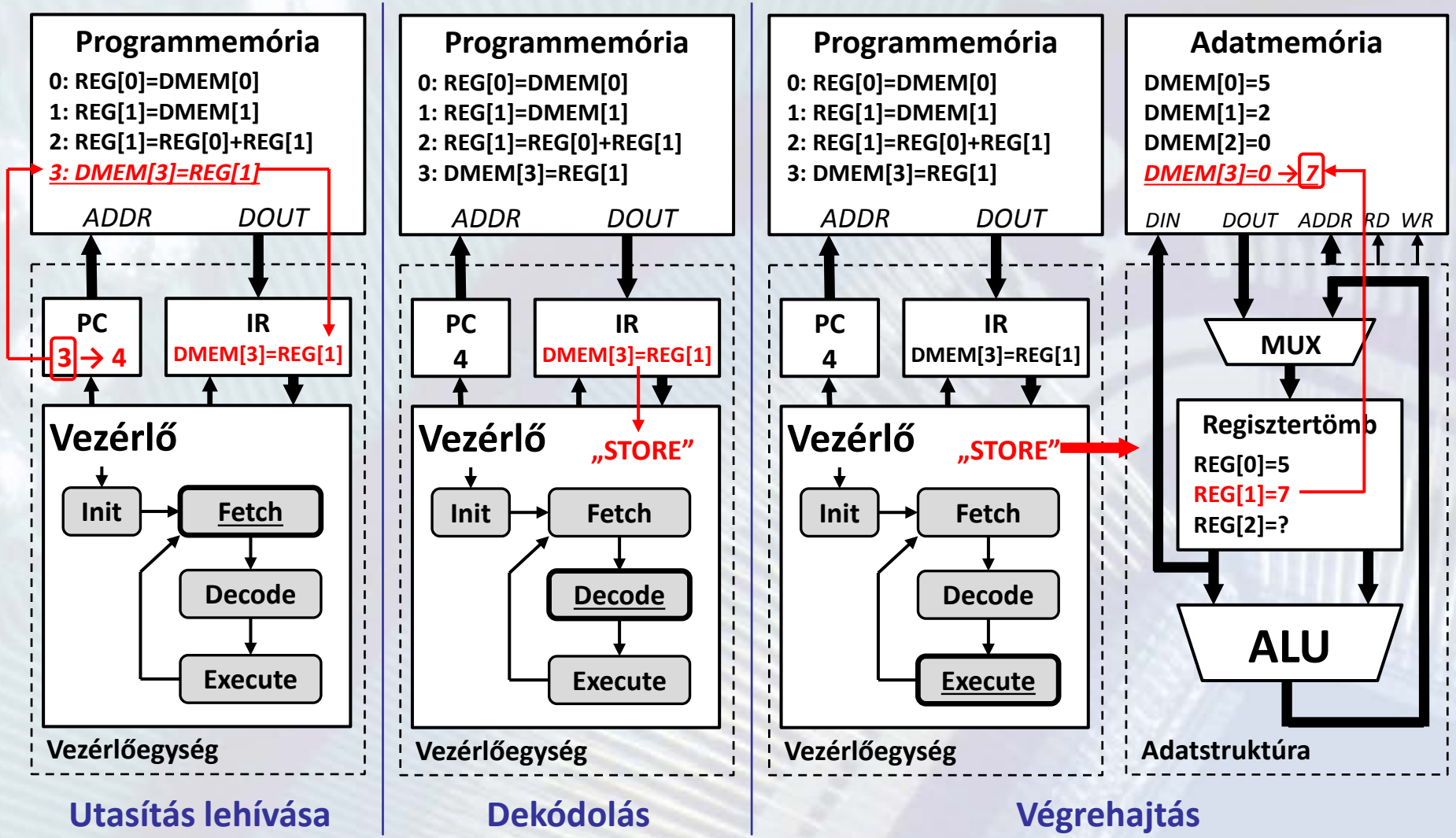
MiniRISC processzor – Felépítés

(Vezérlőegység)



MiniRISC processzor – Felépítés

(Vezérlőegység)



MiniRISC processzor – Felépítés

(Vezérlőegység)

- **A programmemória bináris kód formájában tartalmazza a végrehajtandó utasításokat (gépi kód)**
  - Magasabb szintű leírást a processzor nem tud értelmezni
- **Minden utasítás tartalmazza a művelet leírását és a művelet elvégzéséhez szükséges egyéb adatokat**

művelet (4 bit) regisztercím (4 bit) memóriacím (8 bit)

Programmemória

Cím	Művelet	Gépi kód (16 bit)	Assembly kód
0:	REG[0] = DMEM[0]	1101000000000000	MOV r0, 0x00
1:	REG[1] = DMEM[1]	1101000100000001	MOV r1, 0x01
2:	REG[1] = REG[0] + REG[1]	1111000100000000	ADD r1, r0
3:	DMEM[3] = REG[1]	1001000100000011	MOV 0x03, r1

Tartalom

MiniRISC assembler

- **A LOGSYS Spartan-3E FPGA kártyához készült MiniRISC mintarendszerre történő programfejlesztéshez egy grafikus fejlesztői környezet áll rendelkezésre (MiniRISC IDE)**
- **Az assembly nyelven megírt programok lefordítása a LOGSYS MiniRISC assemblerrel lehetséges**
  - A .NET keretrendszer 4.0 verziója szükséges a futtatáshoz (Windows 8-tól az operációs rendszer része)
- **Az assembler a processzorhoz illesztett képességű**
  - Egyszerű utasítás értelmezés
  - Abszolút kódot generál (nincs linker)
  - Nincs makrózás
  - Nincs cím aritmetika
  - Nincs feltételes fordítás
  - Hiba esetén hibaüzenettel leáll

MiniRISC assembler

(Használat)

- **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*** és fájl a LOGSYS GUI-val letölthető, a program- adatmemória tartalmát inicializáló SVF
    - ***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**

MiniRISC assembler

(Használat)

- **Forrásfájl: egyszerű szövegfájl .s kiterjesztéssel**
- **A feldolgozás soronként történik: 1 sorban 1 utasítás**
- **Az assembly forráskód sorok formátuma**
  LABEL: MNEMONIC OP1{, OP2} ; Comment
  - ***LABEL*** Azonosító, értéke az azt követő utasítás címe
  - ***MNEMONIC*** A végrehajtandó műveletre utaló mnemonik, pl. ***ADD*** – összeadás, ***JMP*** – ugrás, stb.
  - ***OP1{, OP2}*** A művelet operandusai, ***OP2*** nincs mindig (***OP1*** sincs az ***RTS***, ***RTI***, ***STI*** és ***CLI*** utasítások esetén)
  - ***; Comment*** A ';' karakter a megjegyzés kezdete, melyet az assembler figyelmen kívül hagy
- **Javaslat**
  - Minden utasításhoz írjunk megjegyzéseket
  - A formázáshoz használjuk a TAB karaktert

MiniRISC assembler
(Használat)

- **Kevés szabály van**
- **Az utasítások operandusai lehetnek**
  - Regiszter r0 – r15
  - Konstans #0 – #255 (ALU műveletekhez)
  - Memóriacím 0 – 255 (ez is konstans, csak más célra)
  - Indirekt cím (r0) – (r15)
- **Numerikus konstans**
  - Nincs prefix decimális 0 – 255
  - 0x prefix hexadecimális 0x00 – 0xFF
  - 0b prefix bináris 0b00000000 – 0b11111111
- **Karakter konstans**
  - A ′ ′ jelek közötti karakter (pl. **#′A′** – értéke 65)
  - Escape szekvenciák: **'\'', '\"', '\\', '\a', '\b', '\f', '\n', '\r', '\t', '\v'**
- **String konstans**
  - A ″ ″ jelek közötti karakterfüzér (pl. **″MiniRISC\r\n″**)
  - Csak az adat szekcióban használható

MiniRISC assembler
(Használat)

- **Assembler direktívák**
  - ***DEF:*** azonosítót rendel konstanshoz
    DEF SW 0x81 ;A DIP kapcsoló periféria címe
    - Ez nem CPU utasítás, hanem az assembler számára definiál egy helyettesítési szabályt, ami a felhasználó számára biztosítja a forráskód jobb olvashatóságát
  - ***CODE:*** a kód szekció kezdetét jelzi
    - Az utána lévő forráskód részből generált kód a prg. memóriába kerül
  - ***DATA:*** az adat szekció kezdetét jelzi
    - Az utána lévő forráskód részből generált kód az adatmemóriába kerül
    - Az adat szekcióban csak címkék és DB direktívák lehetnek
  - ***DB:*** az adatmemória inicializálása konstansokkal
    DB "MiniRISC.\r\n", 0 ;Nulla végű string
    - Csak az adat szekcióban lehet használni
    - Numerikus, karakter és string konstansok adhatók meg utána
    - Az egyes konstansokat vesszővel kell elválasztani egymástól

MiniRISC assembler
(Használat)

- **Assembler direktívák**
  - ***ORG:*** kezdőcím közvetlen előírása
    ORG memóriacím
    - Az utána lévő kódrész kezdőcímét állítja be
    - Használható a kód és az adat szekcióban is
- **Az assembler által generált LST fájlban ellenőrizhető az utasítások címe és gépi kódja, valamint a fordító által használt azonosítók értelmezése**
- **Ha a kód hibátlanul lefordult, akkor beépíthető a hardver tervbe, mint memória inicializáló adat (.HEX kimeneti fájlok) vagy letölthető a kész konfigurációra (.SVF kimeneti fájl)**

MiniRISC assembler

(Belső felépítés, működés)

- **Az assembly program jellege miatt a forrásfájl feldolgozása soronként történik, ezért az assembler belső felépítése jóval egyszerűbb, mint pl. egy C/C++ fordítóé**
- **A feldolgozás három fő lépésből áll**
  - Lexikai elemzés
  - Szintaktikai elemzés
  - Kódgenerálás

MiniRISC assembler

(Belső felépítés, működés)

- **A *lexikai elemzés* során történik a forráskód nyelvi elemekre (tokenekre) bontása**
  - Kulcsszó (utasítás, direktíva, processzor regiszter)
  - Azonosító, címke, szám, numerikus konstans, string
  - Speciális jelentéssel bíró karakter, sorvég, fájlvég, stb.
- **A nyelvi elemek véges automatákkal (FSM) felismerhetőek**
- **Érvénytelen token esetén hibajelzés**
- **Bizonyos esetekben szükséges a tokenek kiértékelése is**
- **Példa a felbontásra**

```
loop:          ;Ciklus kezdete
 mov LD, r5 ;Kiírás a LED-re
 add r5, #1 ;r5 növelése
```

Tokenek (érték a legalsó sorban, ha van)

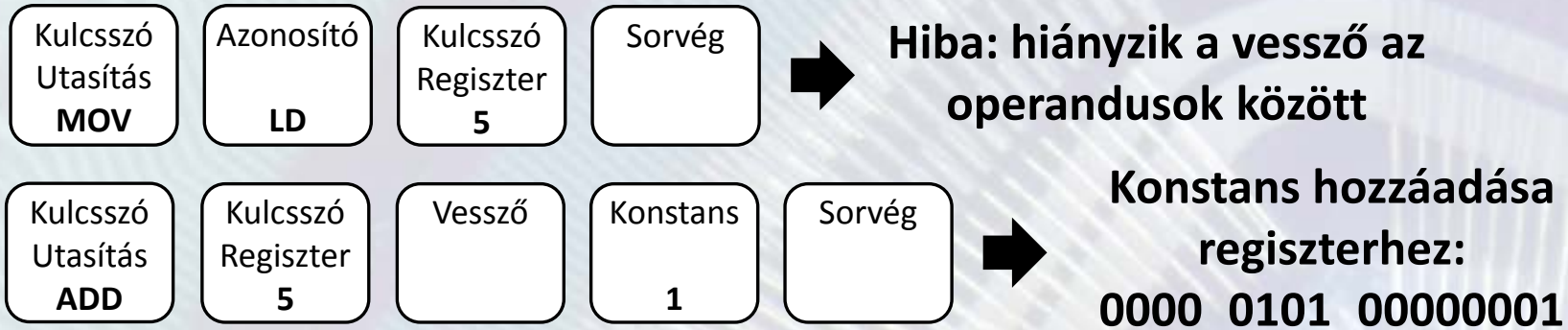
Címke	Sorvég	Kulcsszó Utasítás	Azonosító
loop		**MOV**	**LD**

Vessző	Kulcsszó Regiszter	Sorvég	Kulcsszó Utasítás	Kulcsszó Regiszter	Vessző	Konstans	Sorvég
	5		**ADD**	**5**		**1**	

MiniRISC assembler

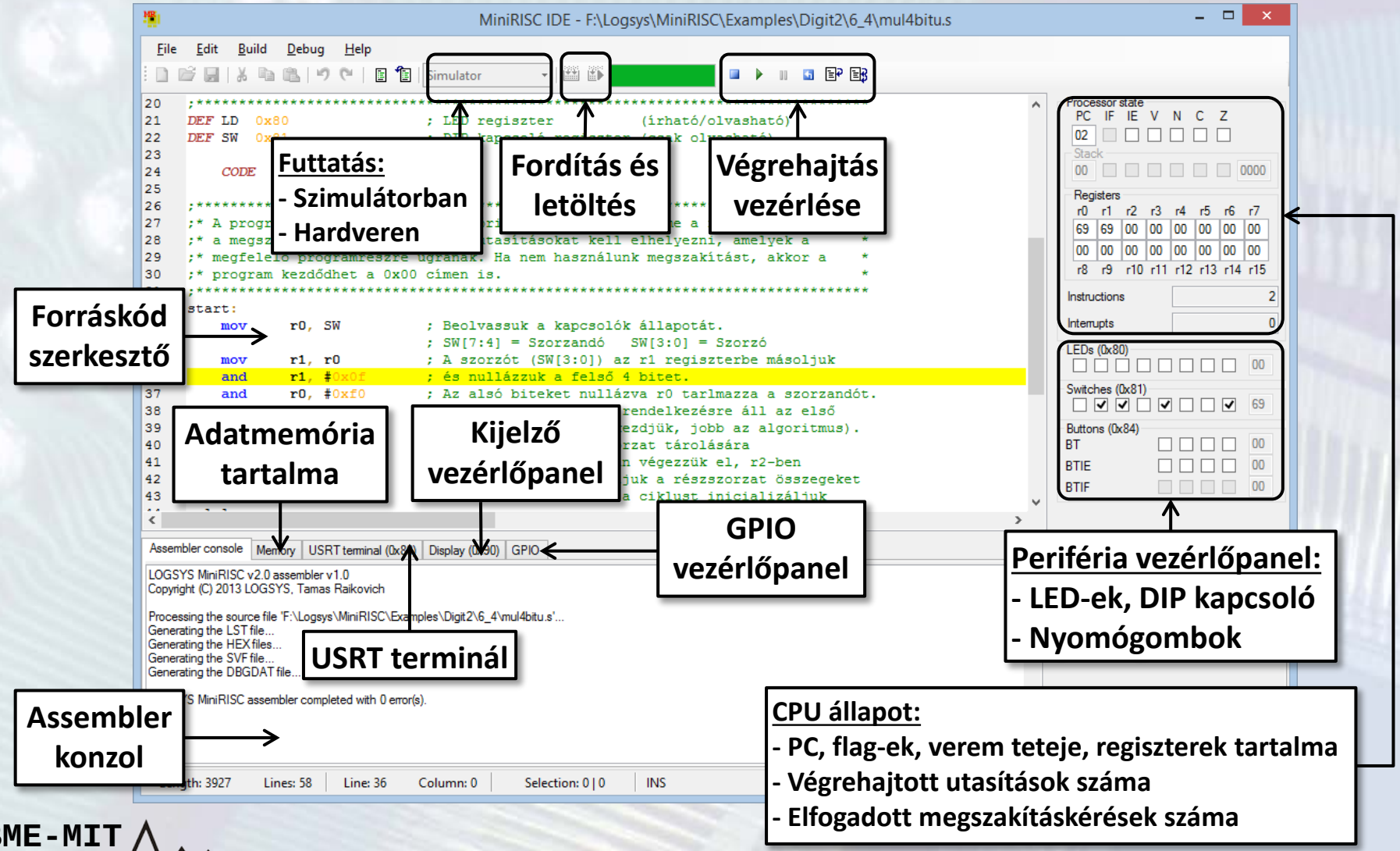
(Belső felépítés, működés)

- **A *szintaktikai elemzés* során a nyelvi elemek (tokenek) megfelelő sorrendjének vizsgálata történik**
- **Érvénytelen sorrend esetén hibajelzés**
- **Megfelelő sorrend esetén az utasítástáblázat szerinti gépi kód generálható**



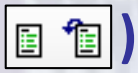
- **Az azonosítókat és a hozzájuk rendelt értékeket a *szimbólum tábla* tárolja, ebből kikereshető az azonosító értéke**
  - Egy azonosító értéke nem áll mindig elsőre rendelkezésre (pl. előre ugrás esetén) → **fordítás két menetben**

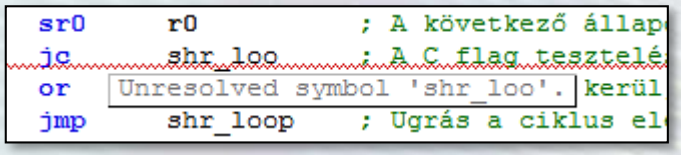
MiniRISC IDE

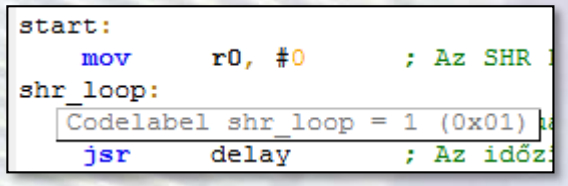


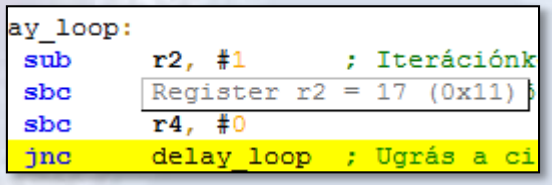
MiniRISC IDE

A forráskód szerkesztő szolgáltatásai:

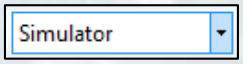
- **Szintaxis kiemelés**
- **Kivágás, másolás, beillesztés (), visszavonás, ismétlés ()**
- **Kijelölt sorok kommentezése és annak megszűntetése ()**
- **A hibás programsorok aláhúzása, illetve a hibaüzenet megjelenítése, ha az egérkurzor a hibás sorra mutat**
- **Lefordított program esetén az azonosítók értékének megjelenítése, ha az egérkurzor az azonosítóra mutat**
- **Letöltött program esetén**
  - A regiszter értékének megjelenítése, ha az egérkurzor a regiszterre mutat
  - Indirekt címzésnél a cím és a memóriában tárolt adat megjelenítése, ha az egérkurzor a regiszterre mutat







MiniRISC IDE

- **A** 

legördülő menüből kiválasztható, hogy a program
  - A szimulátorban fusson (ez mindig rendelkezésre áll)
  - A csatlakoztatott hardveren fusson (***LDCxxx*** letöltőkábel)
  - Választás csak akkor lehetséges, ha program nem fut
- **Szimulátor**
  - Órajelciklus pontossággal szimulálja az FPGA áramkörben megvalósított processzoros rendszert
  - A program végrehajtása itt lassabb, mint a hardveren
    - Kb. 400000 utasítás/s → **kb. 1,2 MHz rendszerórajel frekvencia**
  - A MiniRISC mintarendszerben lévő perifériák nagy része a szimulátorban is elérhető (ami nincs: DMA, VGA, PS/2)
- **Futtatás hardveren**
  - Ha van FPGA kártya csatlakoztatva a számítógéphez

MiniRISC IDE

- **A program a *Compile* ( , F5) gombbal fordítható le**
  - A hibaüzenetek a konzolban jelennek meg
  - A hibás programsorok pirossal aláhúzásra kerülnek
- **A lefordult program a *Download* ( , F6) gombbal tölthető le**
  - Szimulátor használata esetén is szükséges!
  - Hardveren történő futtatás esetén ha még nincs felkonfigurálva az FPGA, akkor először ez történik meg
  - A program futása megáll a 0x00 címen (reset vektor)
- **A program végrehajtásának vezérlése**
  - ***Run*** ( , F7): a program végrehajtásának folytatása
  - ***Break*** ( , F8): a program futásának felfüggesztése, a következő végrehajtandó utasítást sárga kiemelés jelzi
  - ***Step*** ( , F10): az aktuális utasítás végrehajtása, a program futása megáll a következő utasításnál

MiniRISC IDE

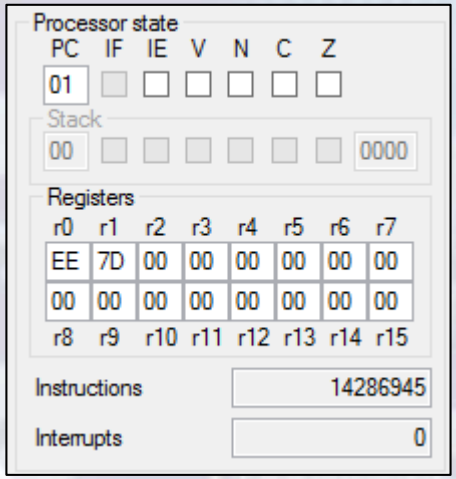
- **A program végrehajtásának vezérlése**
  - ***Auto step*** (): a ***Step*** parancs automatikus kiadása
    - A gyakoriság a Debug menüben állítható be
    - Az automatikus léptetés a ***Break*** paranccsal állítható le
  - ***Reset*** ( , F9): reset jel kiadása a processzoros rendszernek
  - ***Stop*** (): a program futtatásának befejezése
    - Ezután ismételt programletöltés szükséges
- **Töréspont elhelyezése a szerkesztőben a margóra történő kattintással lehetséges**
  - A töréspontot piros kör jelzi a margón

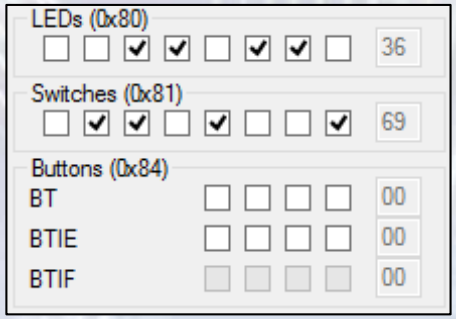
```
10        mov     r0, SW      ; A kapcsolók állapota r0-ba kerül.
11        mov     LD, r0      ; A LED-ekre kiírjuk r0 értékét.
12        jmp     start       ; Ugrás a ciklus elejére.
```

  - Töréspontra futáskor a program végrehajtása megáll
    - Lényegében egy ***Break*** parancs kiadása történik meg hardveresen

MiniRISC IDE

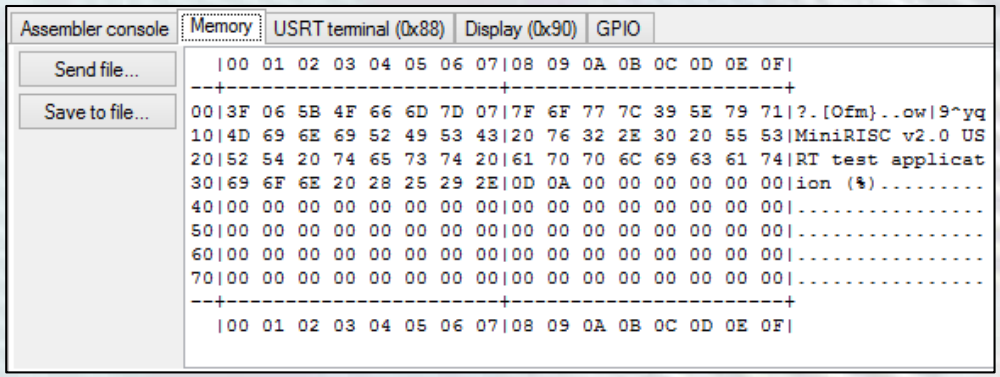
- **A processzor állapotának és az adatmemória tartalmának módosítása, illetve a perifériák vezérlése csak felfüggesztett programvégrehajtás (break állapot) esetén lehetséges**
- ***Processor state* panel:** a processzor állapota
  - Programszámláló (PC) értéke
  - Flag-ek (Z, C N, V, IE, IF) értéke (IF csak olv.)
  - Verem tetejének értéke (csak olvasható)
  - Regiszterek értéke
  - A végrehajtott utasítások száma
  - Az elfogadott megszakításkérések száma
- **Alap perifériák vezérlőpanelje**
  - A LED-ek, a kapcsolók és a nyomógombok állapotát jeleníti meg
  - Biztosítja a perifériák regiszterszintű vezérlését

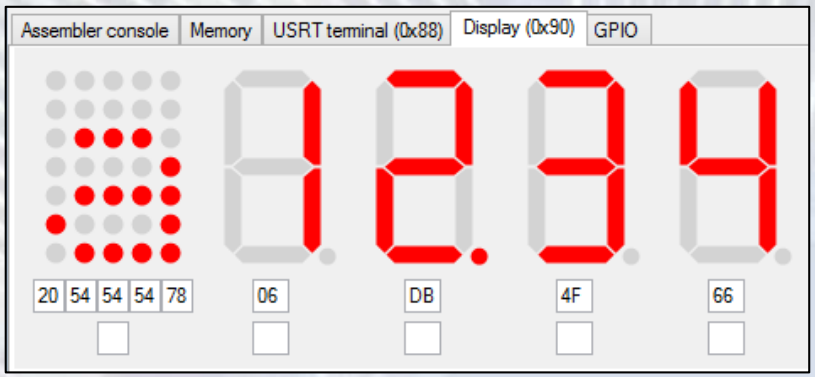




MiniRISC IDE

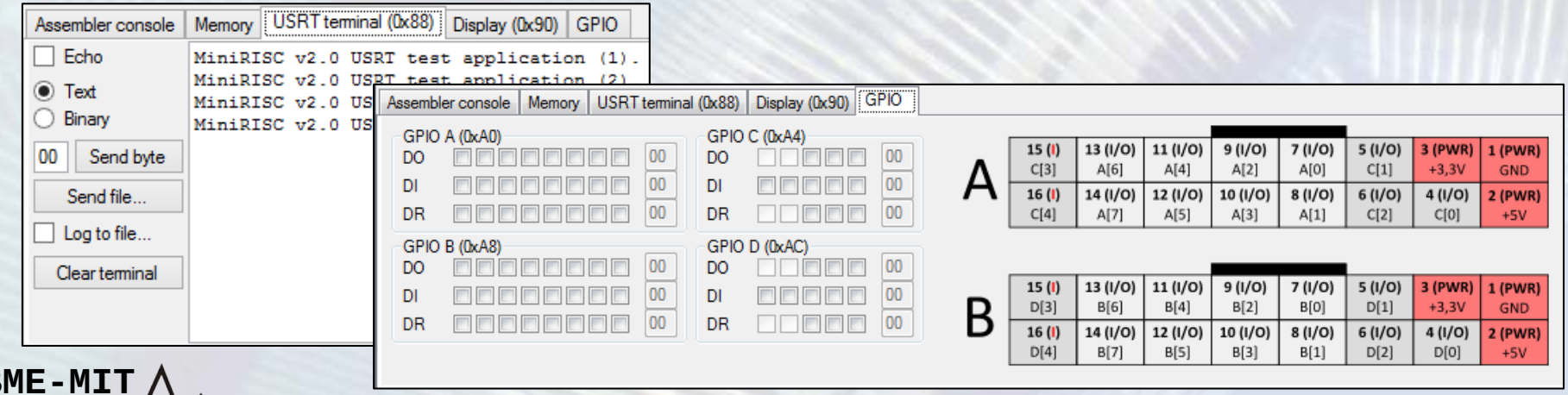
- ***Memory* ablak**
  - A 128 x 8 bites adatmemória tartalmát jeleníti meg
  - Az egyes adatbájtokra kattintva azok módosíthatók
  - A tartalom fájlból betölthető (***Send file…*** gomb), illetve fájlba elmenthető (***Save to file…*** gomb)
- ***Display* ablak**
  - A hétszegmenses és a pontmátrix kijelzők vezérlését biztosítja
  - A kijelzők szegmenseire kattintva azok ki- és bekapcsolhatók
  - A szövegdobozokban megadható az egyes szegmensek értéke, illetve a megjelenítendő karakter





MiniRISC IDE

- ***USRT terminal* ablak**
  - Soros kommunikációs lehetőséget biztosít
  - A leütött karakterek elküldésre kerülnek, a vett karakterek megjelennek a terminál ablakban
  - Lehetőség van fájl elküldésére és a vett adat fájlba mentésére is
- **GPIO ablak**
  - A GPIO perifériák állapotát jeleníti meg és ezek regiszterszintű vezérlésére szolgál (kimeneti adat, aktuális állapot, irány)
  - Ábra az FPGA kártya bővítőcsatlakozóinak lábkiosztásáról



A programfejlesztés lépései

- Gondoljuk át a problémát és készítsünk vázlatos elképzelést, hogy mit és hogyan szeretnénk megoldani
- Fogalmazzuk meg a megoldást a MiniRISC assembly utasítások szintjén és készítsük el a forráskódot
- A begépelt programot fordítsuk le a Compile (F5) paranccsal
- Az esetleges hibákat javítsuk ki
- A hibátlanul lefordult programból generált SVF fájl letölthető a Download (F6) paranccsal
- Ellenőrizzük a program működését debug módban a MiniRISC IDE szolgáltatásainak segítségével

Példaprogramok

1. példa: a DIP kapcsoló állapotának megjelenítése a LED-eken

- Nagyon egyszerű, végtelen ciklusban beolvassuk a kapcsolók állapotát (0x81) és kiírjuk a LED-ekre (0x80)
- Nem használunk megszakítást, ezért a program kezdődhet a 0x00 címen (reset vektor)

```
DEF LD   0x80              ; LED regiszter
DEF SW   0x81              ; DIP kapcsoló regiszter

    CODE

;*****************************************************
;* A program kezdete. A 0x00 cím a reset vektor.     *
;*****************************************************
start:
    mov     r0, SW         ; A kapcsolók állapota r0-ba kerül.
    mov     LD, r0         ; A LED-ekre kiírjuk r0 értékét.
    jmp     start          ; Ugrás a ciklus elejére.
```

Példaprogramok

1. példa: a DIP kapcsoló állapotának megjelenítése a LED-eken

- Az assembler által generált listafájl tartalma

```
LOGSYS MiniRISC v2.0 assembler v1.0 list file
Copyright (C) 2013 LOGSYS, Tamas Raikovich
Source file: pelda1.s
Created on : 2013.03.27. 12:51:08

S Addr  Instr   Source code
------------------------------------------------------------------------------
                DEF LD 0x80                 ; LED regiszter
                DEF SW 0x81                 ; DIP kapcsoló regiszter

                    CODE

                ;*********************************************************
                ;* A program kezdete. A 0x00 cím a reset vektor.          *
                ;*********************************************************
C 00            start:
C 00    D081        mov     r0, SW[81]      ; A kapcsolók állapota r0-ba kerül.
C 01    9080        mov     LD[80], r0      ; A LED-ekre kiírjuk r0 értékét.
C 02    B000        jmp     start[00]       ; Ugrás a ciklus elejére.
```

Tartalom

1. **Bevezetés**
2. **A MiniRISC processzoros rendszer**
   - Adatstruktúra
   - Vezérlőegység
3. **Fejlesztői környezet**
   - A MiniRISC assembler
   - A MiniRISC IDE
   - Szoftverfejlesztés
4. ***A MiniRISC processzoros rendszer – részletek***
   - **A MiniRISC processzor belső felépítése**
   - A MiniRISC processzor interfészei
   - Perifériák illesztése (példákkal)
   - A MiniRISC mintarendszer

MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

- **Utasításkészlet: a megengedett utasítások listája és ezek memóriabeli reprezentációja**
- **Egy utasítás lényegében egy bináris szám (gépi kód)**
- **Az utasítások felépítése**
  - ***Műveleti kód (opcode):*** az elvégzendő műveletet határozza meg
  - ***Operandusok (operands):*** adatok, amelyeken a műveletvégzés történik
    - Regiszter
    - Konstans

MiniRISC processzor – Felépítés

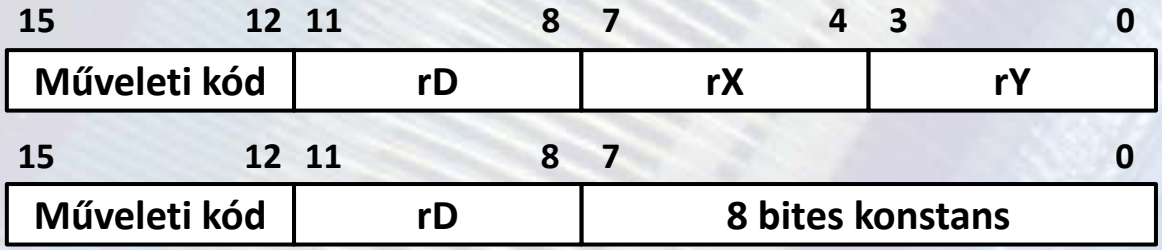
(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

- **A MiniRISC utasítások mérete 16 bites a hatékonyabb FPGA erőforrás kihasználtság érdekében (kisebb programmemória)**
- **16 regisztert tartalmazó regisztertömb**
  - ***Regiszter operandus: 4 bites cím***
  - 16 regiszter a legtöbb feladathoz elegendő
  - Több regiszter a 16 bites utasítások miatt nem lehetséges
- **8 bites adatstruktúra**
  - ***Konstans operandus: 8 bites***
  - A konstanssal való műveletvégzés gyakori, ezért az ALU műveleteknél a konstans operandusok alkalmazhatósága jelentősen csökkenti a program méretét
- **256 szavas program- és adatmemória → 8 bites memóriacím**
  - A teljes címtartomány lefedhető abszolút címzéssel és indirekt címzéssel

MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

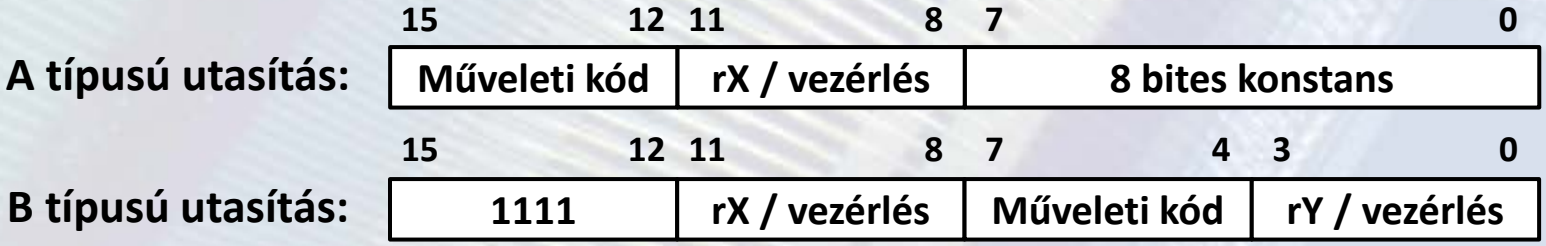
- **Hány operandus legyen a 16 bites utasításokban?**
- **Három regisztercímes architektúra**
  - A program kevesebb utasításból áll
  - Nagyméretű utasításokat igényel
  - Operandusok: 12 bit
    - Két forrásregiszter (rX, rY) és egy célregiszter (rD)
    - Egy regiszter (rD) és egy 8 bites konstans
  - Műveleti kód: 4 bit → 16 műveleti kód
    - ***A MiniRISC processzorhoz 16 műveleti kód nem elegendő!***



MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

- **Hány operandus legyen a 16 bites utasításokban?**
- **Két regisztercímes architektúra**
  - Jó kompromisszum a programban lévő utasítások száma és az utasítások mérete között
  - Az operandusok alapján két fő utasítás típus
    - Egy regiszter (rX) és egy 8 bites konstans → 12 bit A típusú utasítás
    - Két regiszter (rX és rY, rX a célregiszter)→ 8 bit B típusú utasítás
  - Műveleti kód: 4 bit + 4 bit
    - A típusú utasítás: 15 műveleti kód (a B típust az 1111 prefix jelzi)
    - B típusú utasítás: 16 műveleti kód
    - ***Összesen 31 műveleti kód, ez elegendő a MiniRISC processzorhoz***



MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

- **Egy utasítás lényegében egy bináris szám (gépi kód)**
- **A gépi kóddal nehéz dolgozni → helyette assembly kód**
- **Az assembly kód mnemonikokat használ**
  - Mnemonik: rövid szó, utal az elvégzett műveletre
    - Például: ADD – összeadás, MOV – adatmozgatás, stb.
  - A MiniRISC utasítások operandusai lehetnek:
    - Regiszter: r0 – r15
    - Konstans: #0 – #255 (ALU műveletekhez)
    - Memóriacím: 0 – 255 (ez is konstans, csak más célra)
    - Regiszter az indirekt címzéshez: (r0) – (r15)
- **Az assembly kódot az assembler fordítja le gépi kódra**

MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

Adatmozgató utasítások

- Adatmemória olvasás abszolút és indirekt címzéssel
- Adatmemória írás abszolút és indirekt címzéssel
- Konstans betöltése regiszterbe
- Adatmozgatás regiszterből regiszterbe
- Az ALU státusz biteket nem módosítják

Gépi kód	Assembly kód	Művelet	Z	C	N	V
1101xxxxaaaaaaaa	MOV rX, addr	rX ← DMEM[addr]	-	-	-	-
1111xxxx1101yyyy	MOV rX, (rY)	rX ← DMEM[rY]	-	-	-	-
1001xxxxaaaaaaaa	MOV addr, rX	DMEM[addr] ← rX	-	-	-	-
1111xxxx1001yyyy	MOV (rY), rX	DMEM[rY] ← rX	-	-	-	-
1100xxxxiiiiiiii	MOV rX, #imm	rX ← imm	-	-	-	-
1111xxxx1100yyyy	MOV rX, rY	rX ← rY	-	-	-	-

MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

Aritmetikai utasítások

- Összeadás és kivonás átvitel nélkül, valamint átvitellel
- Összehasonlítás (kivonás az eredmény tárolása nélkül)
- Operandusok: két regiszter vagy egy regiszter és egy konstans
- A műveleti kód egyes bitjeit vezérlésre használjuk fel
  - Ezáltal egyszerűsödik a vezérlő állapotgép

Gépi kód	Assembly kód	Művelet	Z	C	N	V
0000xxxxiiiiiiii	ADD rX, #imm	rX ← rX + imm	+	+	+	+
0001xxxxiiiiiiii	ADC rX, #imm	rX ← rX + imm + C	+	+	+	+
0010xxxxiiiiiiii	SUB rX, #imm	rX ← rX – imm	+	+	+	+
0011xxxxiiiiiiii	SBC rX, #imm	rX ← rX – imm – C	+	+	+	+
1010xxxxiiiiiiii	CMP rX, #imm	rX – imm	+	+	+	+

Átvitel kiválasztása (0: átvitel nélkül, 1: átvitellel)

Művelet kiválasztása (0: összeadás, 1: kivonás)

1, ha az aritmetikai művelet eredményét nem kell tárolni

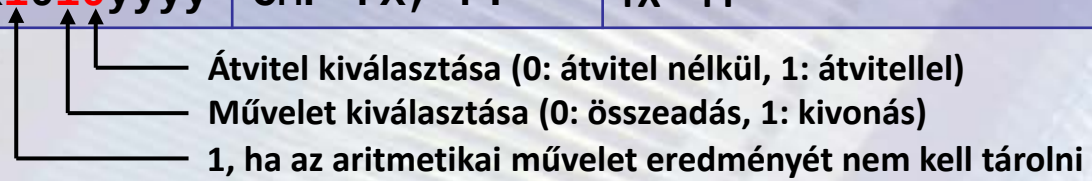
MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

Aritmetikai utasítások

- Összeadás és kivonás átvitel nélkül, valamint átvitellel
- Összehasonlítás (kivonás az eredmény tárolása nélkül)
- Operandusok: két regiszter vagy egy regiszter és egy konstans
- A műveleti kód egyes bitjeit vezérlésre használjuk fel
  - Ezáltal egyszerűsödik a vezérlő állapotgép

Gépi kód	Assembly kód	Művelet	Z	C	N	V
1111xxxx0000yyyy	ADD rX, rY	rX ← rX + rY	+	+	+	+
1111xxxx0001yyyy	ADC rX, rY	rX ← rX + rY + C	+	+	+	+
1111xxxx0010yyyy	SUB rX, rY	rX ← rX – rY	+	+	+	+
1111xxxx0011yyyy	SBC rX, rY	rX ← rX – rY – C	+	+	+	+
1111xxxx1010yyyy	CMP rX, rY	rX – rY	+	+	+	+



MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

Logikai és csere utasítások

- Bitenkénti AND, OR és XOR
- Bittesztelés (bitenkénti AND az eredmény tárolása nélkül)
- Az A típusú 0111 a csere kódja (a B típusú 0111 a shiftelés kódja)
- Operandusok: két regiszter vagy egy regiszter és egy konstans
- A műveleti kód egyes bitjeit vezérlésre használjuk fel

Gépi kód	Assembly kód	Művelet	Z	C	N	V
0100xxxxiiiiiiii	AND rX, #imm	rX ← rX & imm	+	-	+	-
0101xxxxiiiiiiii	OR rX, #imm	rX ← rX | imm	+	-	+	-
0110xxxxiiiiiiii	XOR rX, #imm	rX ← rX ^ imm	+	-	+	-
0111xxxx00000000	SWP rX	rX ← {rX[3:0], rX[7:4]}	+	-	+	-
1000xxxxiiiiiiii	TST rX, #imm	rX & imm	+	-	+	-

Művelet kiválasztása (00: AND, 01: OR, 10: XOR, 11: csere)

1, ha a logikai művelet eredményét nem kell tárolni

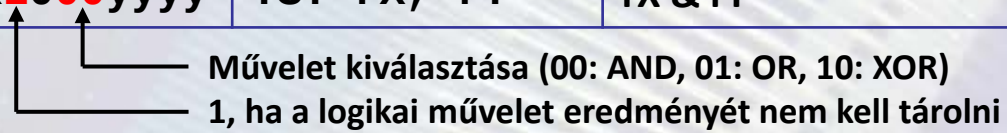
MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

Logikai és csere utasítások

- Bitenkénti AND, OR és XOR
- Bittesztelés (bitenkénti AND az eredmény tárolása nélkül)
- Az A típusú 0111 a csere kódja (a B típusú 0111 a shiftelés kódja)
- Operandusok: két regiszter vagy egy regiszter és egy konstans
- A műveleti kód egyes bitjeit vezérlésre használjuk fel

Gépi kód	Assembly kód	Művelet	Z	C	N	V
1111xxxx0100yyyy	AND rX, rY	rX ← rX & rY	+	-	+	-
1111xxxx0101yyyy	OR rX, rY	rX ← rX | rY	+	-	+	-
1111xxxx0110yyyy	XOR rX, rY	rX ← rX ^ rY	+	-	+	-
1111xxxx1000yyyy	TST rX, rY	rX & rY	+	-	+	-



MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

Léptetési és forgatási utasítások

- Normál és aritmetikai shiftelés, normál forgatás
- Forgatás a carry (C) flag-en keresztül
- Operandusok: egy regiszter (rX) → az rY regisztercím vezérlésre használható, ezért csak egy műveleti kódra van szükség

Gépi kód	Assembly kód	Művelet	Z	C	N	V
1111xxxx01110000	SL0 rX	rX ← {rX[6:0], 0}	+	+	+	-
1111xxxx01110100	SL1 rX	rX ← {rX[6:0], 1}	+	+	+	-
1111xxxx01110001	SR0 rX	rX ← {0, rX[7:1]}	+	+	+	-
1111xxxx01110101	SR1 rX	rX ← {1, rX[7:1]}	+	+	+	-
1111xxxx01111001	ASR rX	rX ← {rX[7], rX[7:1]}	+	+	+	-

1111xxxx01111001

- Irány kiválasztása (0: balra, 1: jobbra)
- Művelet kiválasztása (0: léptetés, 1: forgatás)
- A beléptetett bit értéke shiftelésnél
- A léptetés típusa (0: normál, 1: aritmetikai)

MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

Léptetési és forgatási utasítások

- Normál és aritmetikai shiftelés, normál forgatás
- Forgatás a carry (C) flag-en keresztül
- Operandusok: egy regiszter (rX) → az rY regisztercím vezérlésre használható, ezért csak egy műveleti kódra van szükség

Gépi kód	Assembly kód	Művelet	Z	C	N	V
1111xxxx01110010	ROL rX	rX ← {rX[6:0], rX[7]}	+	+	+	-
1111xxxx01110011	ROR rX	rX ← {rX[0], rX[7:1]}	+	+	+	-
1111xxxx01110110	RLC rX	rX ← {rX[6:0], C}	+	+	+	-
1111xxxx01110111	RRC rX	rX ← {C, rX[7:1]}	+	+	+	-

Irány kiválasztása (0: balra, 1: jobbra)
Művelet kiválasztása (0: léptetés, 1: forgatás)
A beléptetett bit forgatásnál (0: a kilépő bit, 1: carry flag)
A léptetés típusa (0: normál, 1: aritmetikai)

MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

Programvezérlési utasítások

- Feltétel nélküli ugrás abszolút és indirekt címzéssel (JMP)
- Feltételes ugrás abszolút és indirekt címzéssel (Jxx)
  - Az ALU státusz bitek értékének tesztelésére használhatók
- Operandusok: egy regiszter (rY) vagy egy konstans
  - Az rX regisztercím nem használt → a műveletet választja ki
  - Egy műveleti kód elegendő a programvezérlési utasításokhoz

Gépi kód	Assembly kód	Művelet	Z	C	N	V
10110000aaaaaaaa	JMP addr	PC ← addr	-	-	-	-
11110000 1011yyyy	JMP (rY)	PC ← rY	-	-	-	-
10110001aaaaaaaa	JZ addr	PC ← addr, ha Z=1	-	-	-	-
11110001 1011yyyy	JZ (rY)	PC ← rY, ha Z=1	-	-	-	-
10110010aaaaaaaa	JNZ addr	PC ← addr, ha Z=0	-	-	-	-
11110010 1011yyyy	JNZ (rY)	PC ← rY, ha Z=0	-	-	-	-

MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

Programvezérlési utasítások

- Feltétel nélküli ugrás abszolút és indirekt címzéssel (JMP)
- Feltételes ugrás abszolút és indirekt címzéssel (Jxx)
  - Az ALU státusz bitek értékének tesztelésére használhatók
- Operandusok: egy regiszter (rY) vagy egy konstans
  - Az rX regisztercím nem használt → a műveletet választja ki
  - Egy műveleti kód elegendő a programvezérlési utasításokhoz

Gépi kód	Assembly kód	Művelet	Z	C	N	V
10110011aaaaaaaa	JC addr	PC ← addr, ha C=1	-	-	-	-
1111001111011yyyy	JC (rY)	PC ← rY, ha C=1	-	-	-	-
10110100aaaaaaaa	JNC addr	PC ← addr, ha C=0	-	-	-	-
1111010011011yyyy	JNC (rY)	PC ← rY, ha C=0	-	-	-	-
10110101aaaaaaaa	JN addr	PC ← addr, ha N=1	-	-	-	-
1111010111011yyyy	JN (rY)	PC ← rY, ha N=1	-	-	-	-

MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

Programvezérlési utasítások

- Feltétel nélküli ugrás abszolút és indirekt címzéssel (JMP)
- Feltételes ugrás abszolút és indirekt címzéssel (Jxx)
  - Az ALU státusz bitek értékének tesztelésére használhatók
- Operandusok: egy regiszter (rY) vagy egy konstans
  - Az rX regisztercím nem használt → a műveletet választja ki
  - Egy műveleti kód elegendő a programvezérlési utasításokhoz

Gépi kód	Assembly kód	Művelet	Z	C	N	V
10110110aaaaaaaa	JNN addr	PC ← addr, ha N=0	-	-	-	-
1111011011011yyyy	JNN (rY)	PC ← rY, ha N=0	-	-	-	-
10110111aaaaaaaa	JV addr	PC ← addr, ha V=1	-	-	-	-
1111011111011yyyy	JV (rY)	PC ← rY, ha V=1	-	-	-	-
10111000aaaaaaaa	JNV addr	PC ← addr, ha V=0	-	-	-	-
1111100011011yyyy	JNV (rY)	PC ← rY, ha V=0	-	-	-	-

MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Utasításkészlet)

Programvezérlési utasítások

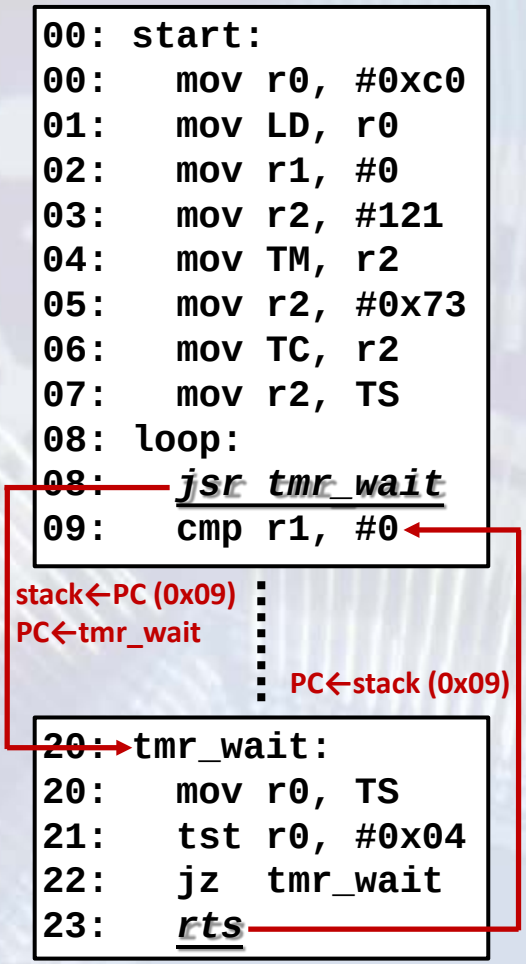
- Szubrutinhívás abszolút és indirekt címzéssel (JSR)
- Visszatérés szubrutinból (RTS) és megszakításból (RTI)
- Megszakítások engedélyezése (STI) és tiltása (CLI)
- Operandusok: egy regiszter (rY) vagy egy konstans
  - Az rX regisztercím nem használt → a műveletet választja ki
  - Egy műveleti kód elegendő a programvezérlési utasításokhoz

Gépi kód	Assembly kód	Művelet	Z	C	N	V
10111001aaaaaaaa	JSR addr	verem ← PC ← addr	-	-	-	-
1111100111011yyyy	JSR (rY)	verem ← PC ← rY	-	-	-	-
1011101000000000	RTS	PC ← verem	-	-	-	-
1011101100000000	RTI	{PC, Z, C, N, V, IE, IF} ← verem	+	+	+	+
1011110000000000	CLI	IE ← 0	-	-	-	-
1011110100000000	STI	IE ← 1	-	-	-	-

MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Szubrutinhívás)

- **Szubrutin**
  - A kód egy adott feladatot végrehajtó része
  - A kód többi részétől viszonylag független
  - Többször felhasználható → csak egy példány szükséges egy adott szubrutinból
  - A szubrutinhíváshoz és a visszatéréshez külön utasítások állnak rendelkezésre
- **Szubrutin meghívása: *JSR utasítás***
  - A következő utasítás címét (a visszatérési címet) elmenti a ***verembe (stack)***
  - A programszámlálóba betölti a szubrutin első utasításának a címét
- **Visszatérés a szubrutinból: *RTS utasítás***
  - A programszámlálóba betölti a visszatérési címet a veremből



MiniRISC processzor – Felépítés

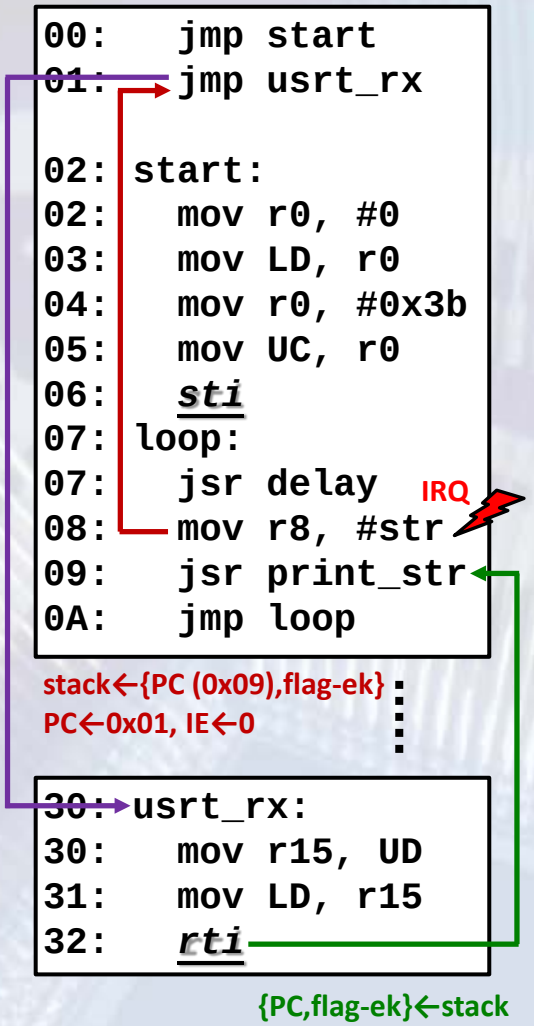
(A MiniRISC processzor vezérlőegysége - Megszakítás)

- **Az utasítások végrehajtása a programozó által meghatározott sorrendben történik**
  - Események kezelése lekérdezéssel → lassú
  - Sok esetben gyorsabb reagálás kell → megszakítás
- **Megszakítás (interrupt)**
  - Külső jelzés a processzornak kiszolgálási igényre
  - A processzor az aktuális utasítás végrehajtása után elfogadhatja
- **Megszakítással kapcsolatos bitek a MiniRISC vezérlőegységében**
  - ***IE (Interrupt Enable) bit:*** a megszakítások engedélyezése
    - IE=0: a megszakítások kiszolgálása tiltott (CLI utasítás)
    - IE=1: a megszakítások kiszolgálása engedélyezett (STI utasítás)
  - ***IF (Interrupt Flag) bit:*** a processzor állapotát jelzi
    - IF=0: normál programvégrehajtás
    - IF=1: megszakításkérés kiszolgálása van folyamatban
    - Értéke csak a debug interfészen keresztül olvasható ki

MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Megszakítás)

- **A MiniRISC megszakítási rendszere**
  - Aktív magas megszakításkérő bemenet (IRQ)
  - Egyszerű megszakítási rendszer
    - A perifériától csak jelzés jön, a kérés azonosítása a programban történik
    - A megszakításkezelő rutin címe fix 0x01
- **Megszakítások kiszolgálása a MiniRISC esetén**
  - Alapvetően a szubrutinhívásra hasonlít
  - Ha IE=1 és IRQ=1, akkor az aktuális utasítás végrehajtása után
    - A visszatérési cím és a flag-ek (ALU státusz bitek, IE, IF) elmentésre kerülnek a ***verembe (stack)***
    - A megszakítás vektor (0x01) betöltésre kerül a programszámlálóba és az IE bit törlődik
- **Visszatérés a megszakításból: *RTI utasítás***
  - A PC-be betöltésre kerül a visszatérési cím és visszaállításra kerülnek a flag-ek a veremből



MiniRISC processzor – Felépítés

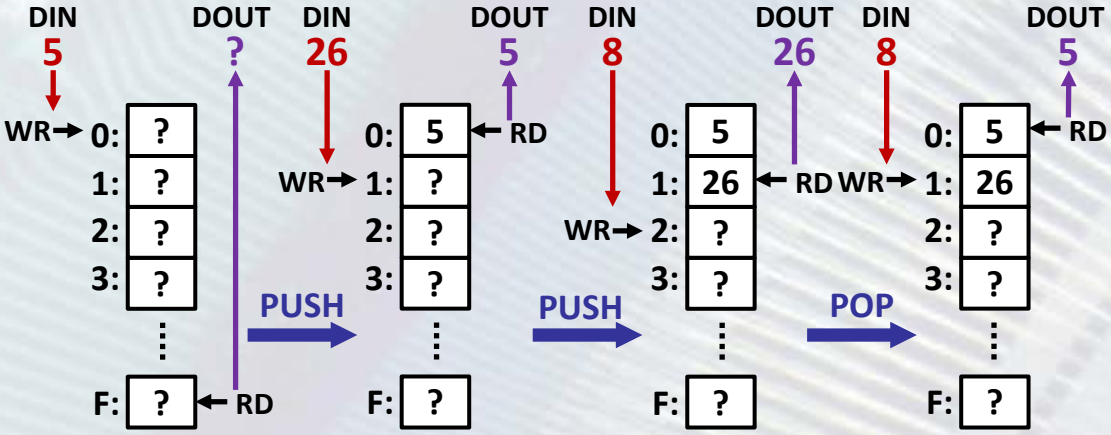
(A MiniRISC processzor vezérlőegysége - Verem)

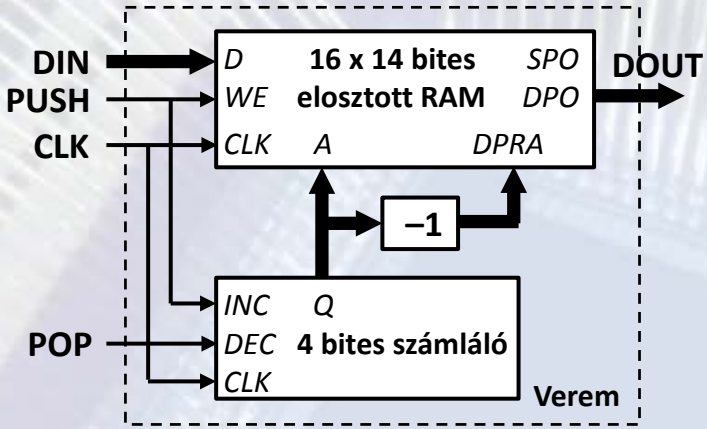
- **A *verem (stack)* egy LIFO (Last-In-First-Out) adattároló**
  - A legutoljára beírt adat olvasható ki először
  - Két művelet értelmezhető rajta
    - ***Push:*** adat ráhelyezése a verem tetejére (tele verem → túlcsordulás)
    - ***Pop:*** adat levétele a verem tetejéről (üres verem → alulcsordulás)
- **A verem típusa lehet**
  - A processzorban lévő belső hardveres verem
  - A memóriában megvalósított külső verem
    - Az SP (Stack Pointer) regiszter tárolja a verem tetejének címét
- **A MiniRISC processzorban 16 szavas belső hardveres verem van**
  - 16 szintű szubrutinhívás és megszakítás kiszolgálás lehetséges
    - Elmentésre kerülnek: programszámláló (PC), flag-ek (Z, C, N, V, IE és IF)
  - RTS utasítás: visszaállítja az elmentett PC értéket
  - RTI utasítás: visszaállítja az elmentett PC értéket és a flag-eket

MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége - Verem)

- **A hardveres verem egy lehetséges megvalósítása FPGA-val**
  - PC: 8 bit, flag-ek: 6 bit → 16 x 14 bites elosztott RAM
    - A push művelet engedélyezi az írást
  - 4 bites kétirányú írási címszámláló
    - A push művelet növeli, a pop művelet csökkenti az értékét
  - Olvasási cím = írási cím – 1
- **A túlcsordulás és az alulcsordulás nincs hardveresen kezelve, ezek elkerülése a programozó feladata!**





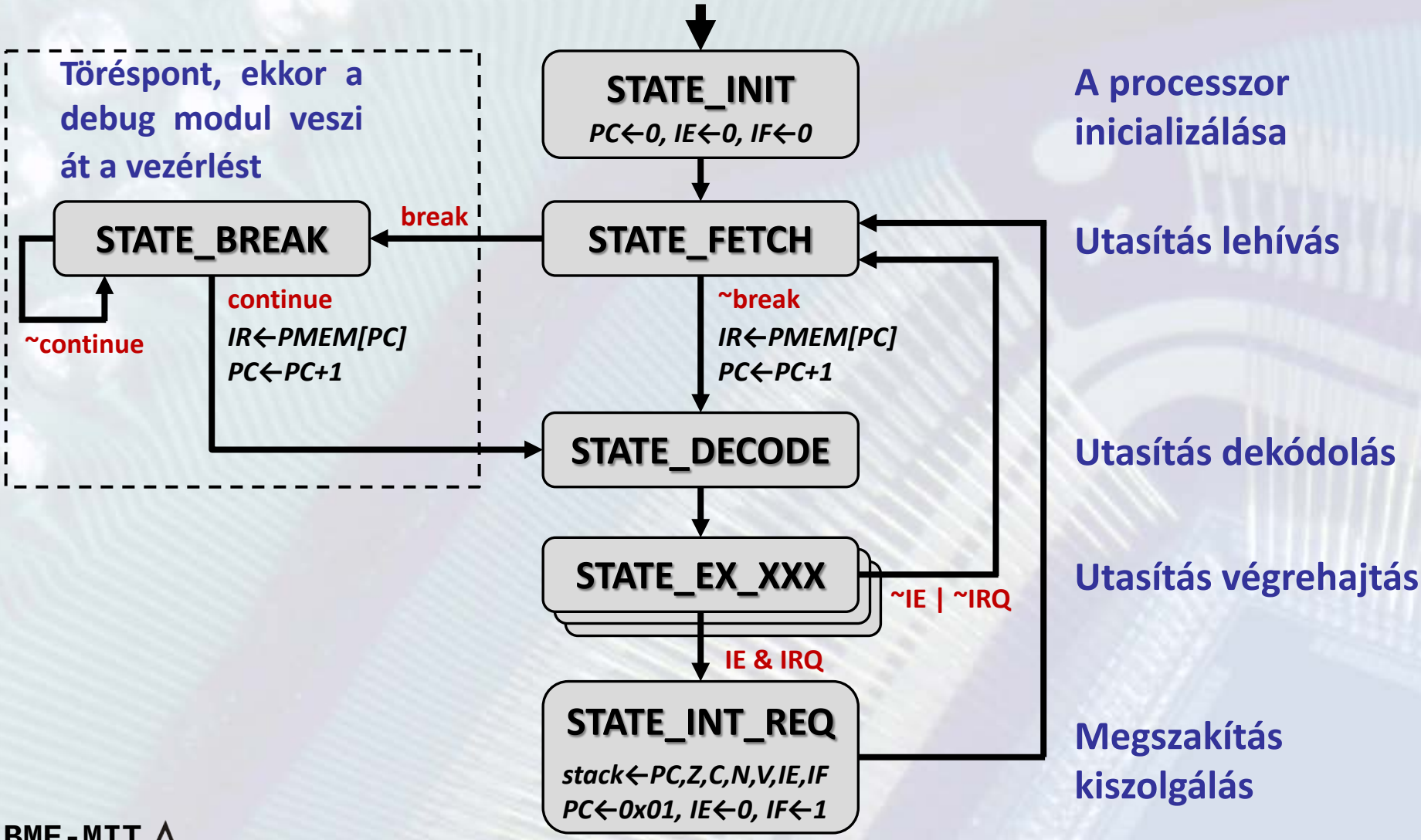
MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége – IR és PC)

- **Utasításregiszter (Instruction Register, IR)**
  - 16 bites tölthető regiszter
  - Az utasítás lehívási (fetch) fázisban kerül betöltésre a programmemóriából beolvasott utasítás
- **Programszámláló (Program Counter, PC)**
  - 8 bites tölthető és engedélyezhető számláló
  - A lehívandó utasítás címét állítja elő
  - Töltés
    - Processzor inicializálása: betöltődik a reset vektor (0x00)
    - Megszakítás kiszolgálás: betöltődik a megszakítás vektor (0x01)
    - Ugrás és szubrutinhívás: betöltődik az ugrási cím
    - Visszatérés (RTS, RTI): betöltődik a visszatérési cím a veremből
  - Engedélyezés
    - Az utasítás lehívási (fetch) fázisban eggyel növekszik az értéke

MiniRISC processzor – Felépítés

(A MiniRISC processzor vezérlőegysége – Vezérlő állapotgép)



MiniRISC processzor – Felépítés

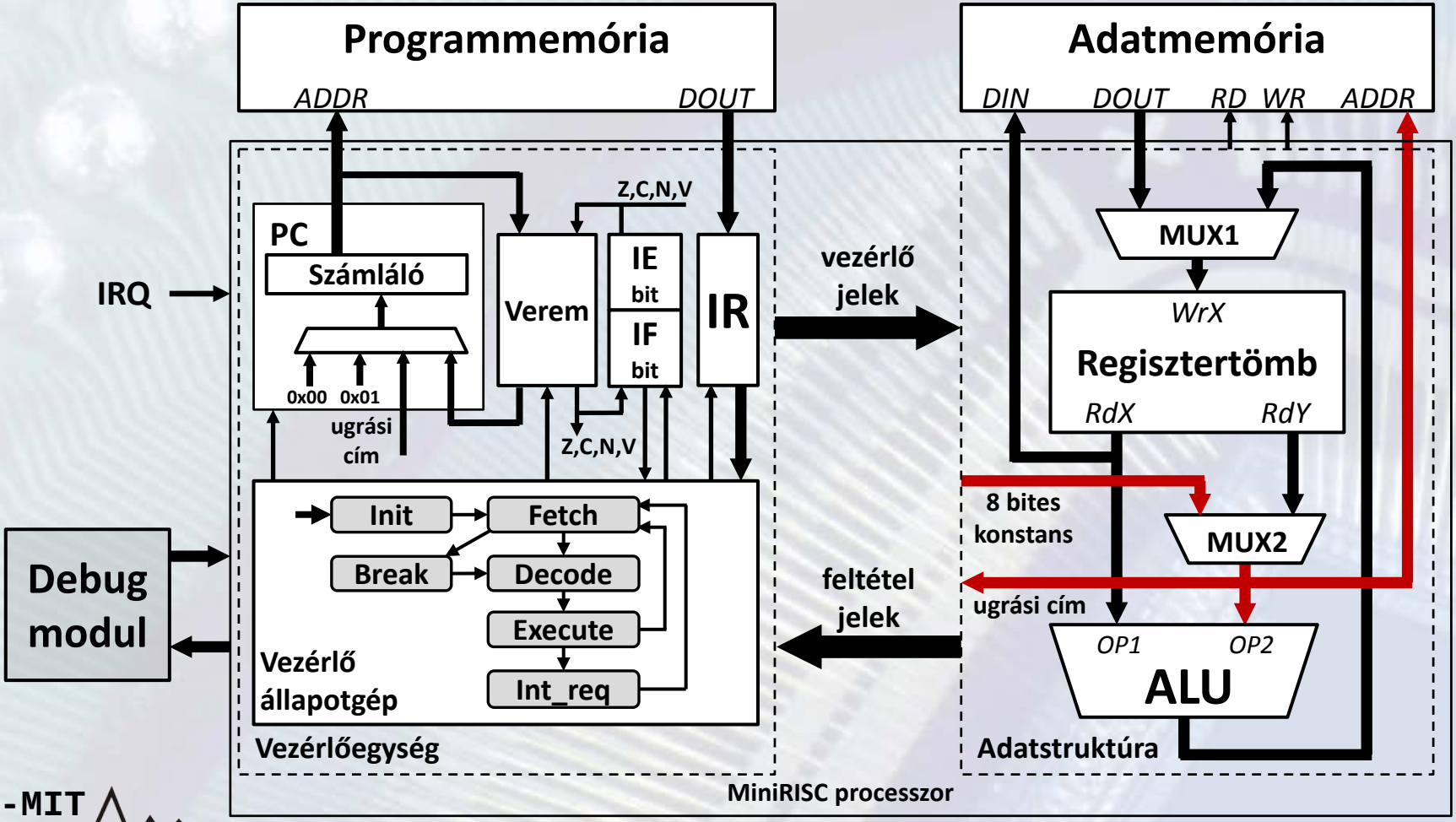
(A MiniRISC processzor vezérlőegysége – Vezérlő állapotgép)

- **Töréspont állapot (*STATE_BREAK*)**
  - A fejlesztést segítő nyomkövetési és hibakeresési célokat szolgál
  - A debug modul veszi át a vezérlést (részletek később)
  - Fetch fázisban ***break*** jelzés → utasítás végrehajtás leáll, BREAK állapot
  - BREAK állapotban ***continue*** jelzés → utasítás végrehajtás folytatása
- **Utasítás végrehajtás**
  - Egy-egy állapot utasítás csoportonként elegendő, mert az utasítások egyes bitjeit közvetlenül felhasználjuk vezérlésre
  - ***STATE_EX_LD:*** adatmemória olvasás végrehajtása
  - ***STATE_EX_ST:*** adatmemória írás végrehajtása
  - ***STATE_EX_MOV:*** konstans betöltés, másolás regiszterből regiszterbe
  - ***STATE_EX_ARITH:*** aritmetikai műveletek végrehajtása
  - ***STATE_EX_LOGIC:*** logikai és csere műveletek végrehajtása
  - ***STATE_EX_SHIFT:*** léptetési és forgatási műveletek végrehajtása
  - ***STATE_EX_CTRL:*** programvezérlési utasítások végrehajtása
  - ***STATE_EX_NOP:*** nincs műveletvégzés (nem használt műveleti kódok)

MiniRISC processzor – Felépítés

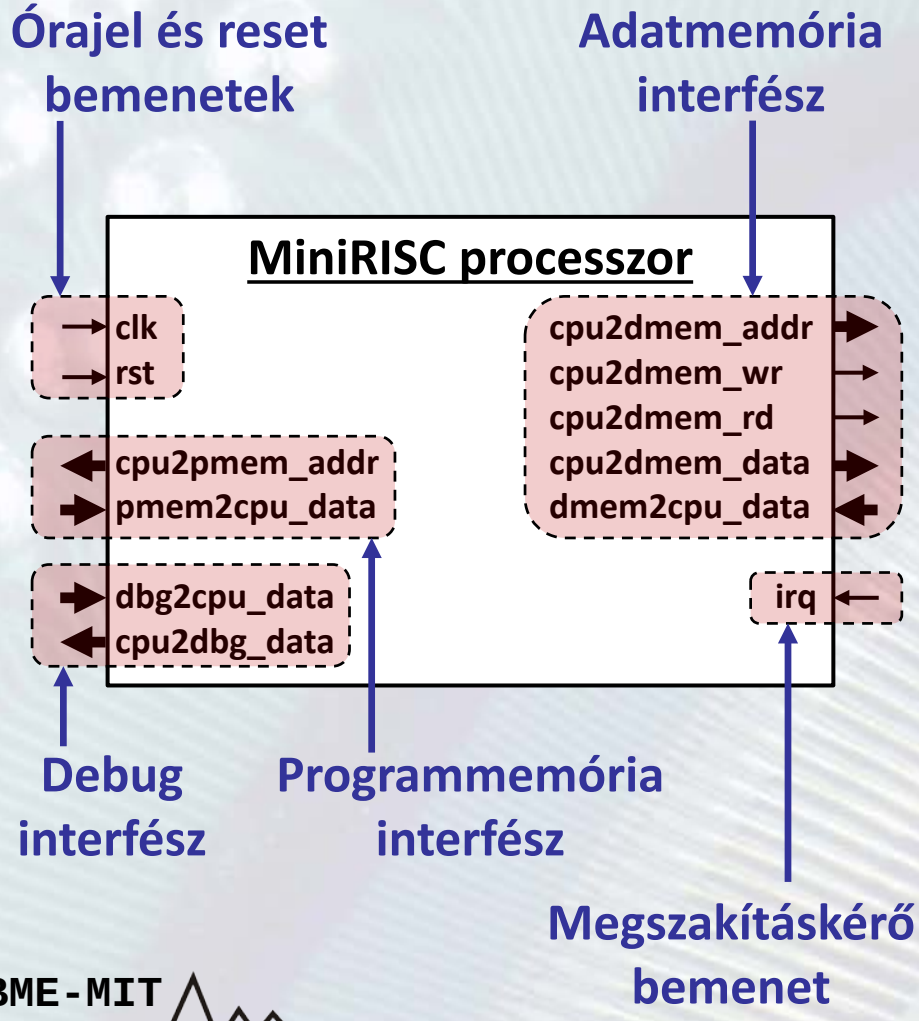
(A MiniRISC processzor blokkvázlata)

A részletes felépítést lásd a Verilog forráskódban



Tartalom

MiniRISC processzor – Interfészek



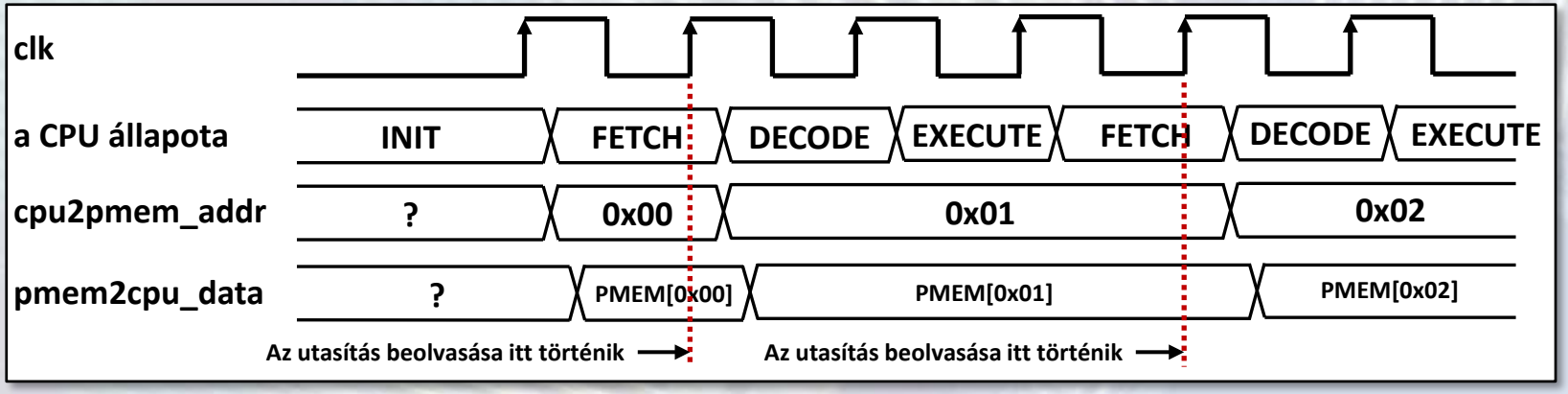
A Verilog modul fejléce:

```
module minirisc_cpu(
   //Órajel és reset.
   input  wire        clk,
   input  wire        rst,

   //A programmemóriával kapcsolatos jelek.
   output wire [7:0]  cpu2pmem_addr,
   input  wire [15:0] pmem2cpu_data,

   //Az adatmemóriával kapcsolatos jelek.
   output wire [7:0]  cpu2dmem_addr,
   output wire        cpu2dmem_wr,
   output wire        cpu2dmem_rd,
   output wire [7:0]  cpu2dmem_data,
   input  wire [7:0]  dmem2cpu_data,

   //Megszakításkérő bemenet (aktív magas).
   input  wire        irq,

   //Debug interfész.
   input  wire [22:0] dbg2cpu_data,
   output wire [47:0] cpu2dbg_data
);
```

MiniRISC processzor – Interfészek

(Órajel, reset, megszakításkérés, programmemória interfész)

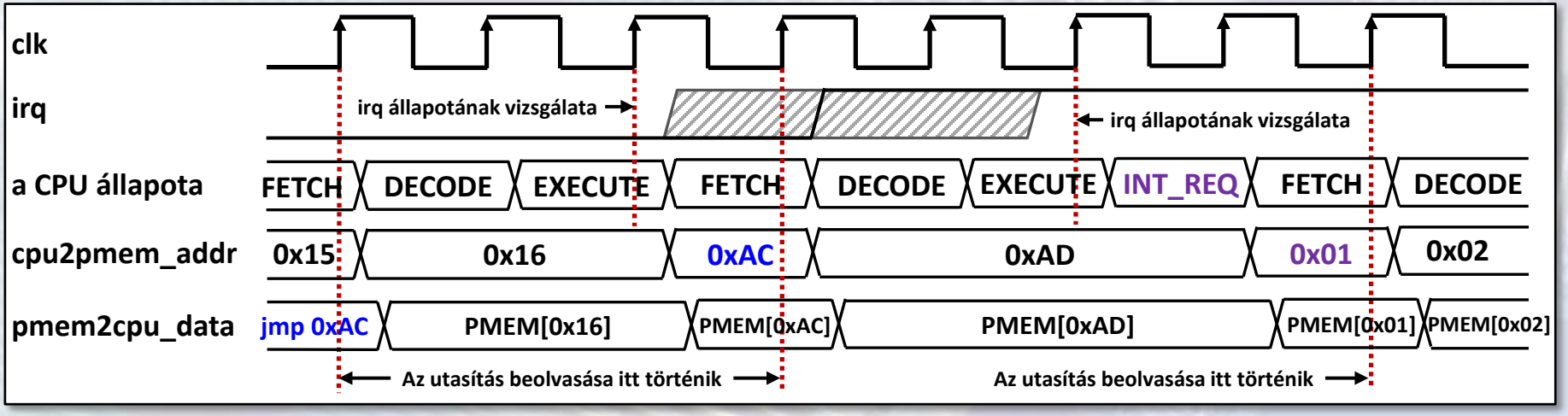
- **Órajel és reset**
  - ***clk:*** rendszerórajel (16 MHz), minden tároló a ***clk*** felfutó élére működik
  - ***rst:*** aktív magas reset jel, amely alapállapotba állítja a processzort
- **Megszakításkérés**
  - ***irq:*** aktív magas megszakításkérő bemenet
  - Több megszakítást kérő periféria esetén VAGY kapcsolat szükséges
- **Programmemória interfész**
  - ***cpu2pmem_addr:*** 8 bites cím a programmemória számára
  - ***pmem2cpu_data:*** a programmemória 16 bites adatkimenete
  - Az utasítások a lehívási (fetch) fázisban kerülnek beolvasásra



MiniRISC processzor – Interfészek

(Programmemória interfész)

- **Programmemória interfész**
  - Ugrás vagy szubrutinhívás esetén a programszámláló értéke módosul(hat) a végrehajtási (execute) fázisban
  - Megszakításkérés kiszolgálása esetén az INT_REQ állapotban a programszámlálóba betöltődik a megszakítás vektor (0x01)
  - A fenti két esetben a következő lehívási (fetch) fázisra éppen időben megjelenik az új cím a programmemória címbuszán



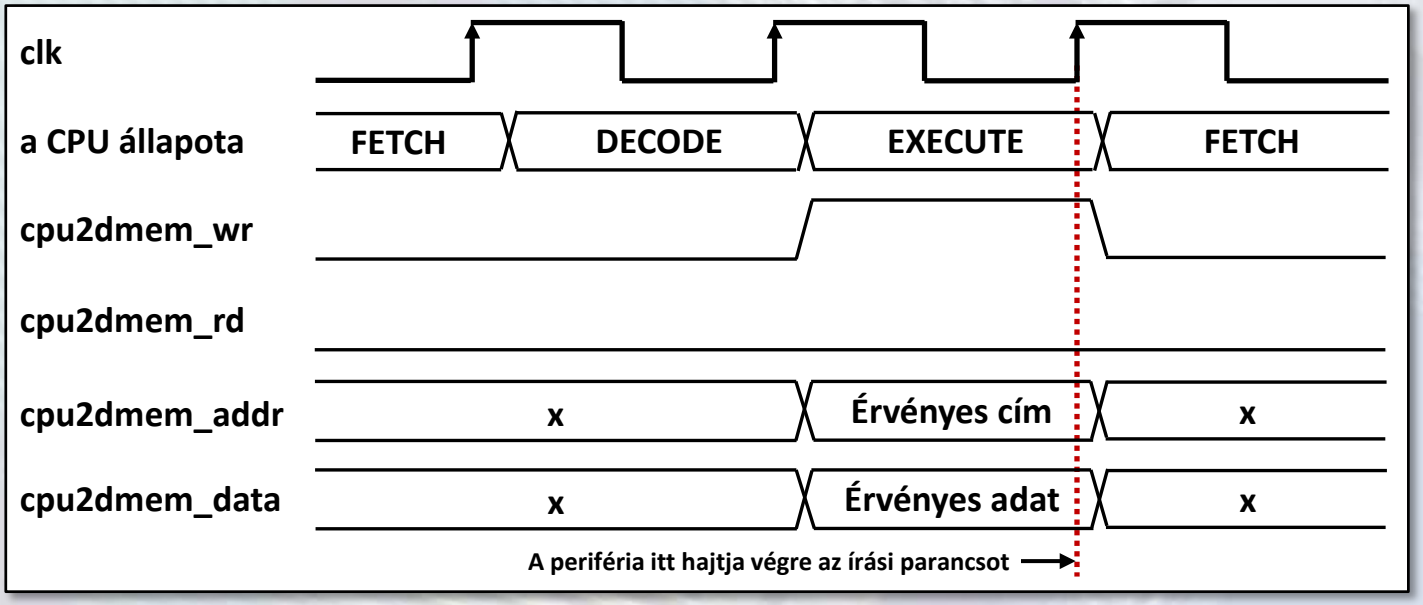
MiniRISC processzor – Interfészek
(Adatmemória interfész)

- **Adatmemória interfész**
  - Egyszerű szinkron busz, parancsok érvényesítése a ***clk*** felfutó élére
  - ***cpu2dmem_addr:*** 8 bites címbusz
  - ***cpu2dmem_wr:*** aktív magas írás engedélyező jel
  - ***cpu2dmem_rd:*** aktív magas olvasás engedélyező jel
  - ***cpu2dmem_data:*** 8 bites írási adatbusz (CPU → periféria)
  - ***dmem2cpu_data:*** 8 bites olvasási adatbusz (periféria → CPU)
- **Az FPGA-n belül nincsenek háromállapotú meghajtók, ezért szükséges a két külön adatbusz (a meghajtók nem megfelelő vezérlése esetén fellépő rövidzárlat tönkreteheti az áramkört)**
- **A MiniRISC processzornak nincsen külön periféria I/O interfésze, ezért a perifériákat memóriába ágyazott módon (az adatmemória interfészen keresztül) kell illeszteni a processzorhoz**
- **Több periféria illesztése esetén, ha az inaktív perifériák 0 értékkel hajtják meg az olvasási adatbuszt, akkor elegendő az olvasási adatbuszokat VAGY kapuval összekapcsolni és nem szükséges multiplexer**
  - Elosztott busz multiplexer funkció

MiniRISC processzor – Interfészek

(Adatmemória interfész – Írási buszciklus)

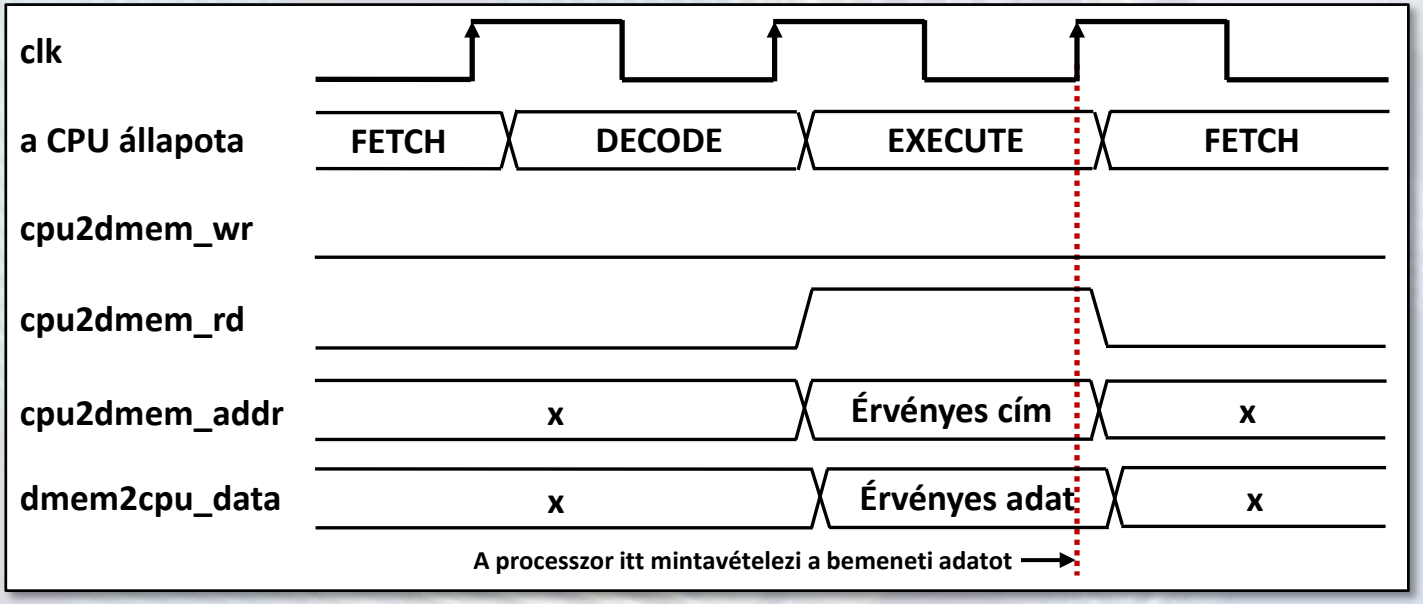
- **Az adatmemória interfész írási (write) ciklusa**
  - Az írási ciklust a végrehajtási (execute) fázisban 1 órajelciklus ideig aktív ***cpu2dmem_wr*** jel jelzi
  - Az írási ciklus alatt a ***cpu2dmem_addr*** címbuszon a cím stabil
  - Az írási ciklus alatt a ***cpu2dmem_data*** írási adatbuszon az adat stabil, melyet a kiválasztott periféria az órajel következő felfutó élére mintavételez



MiniRISC processzor – Interfészek

(Adatmemória interfész – Olvasási buszciklus)

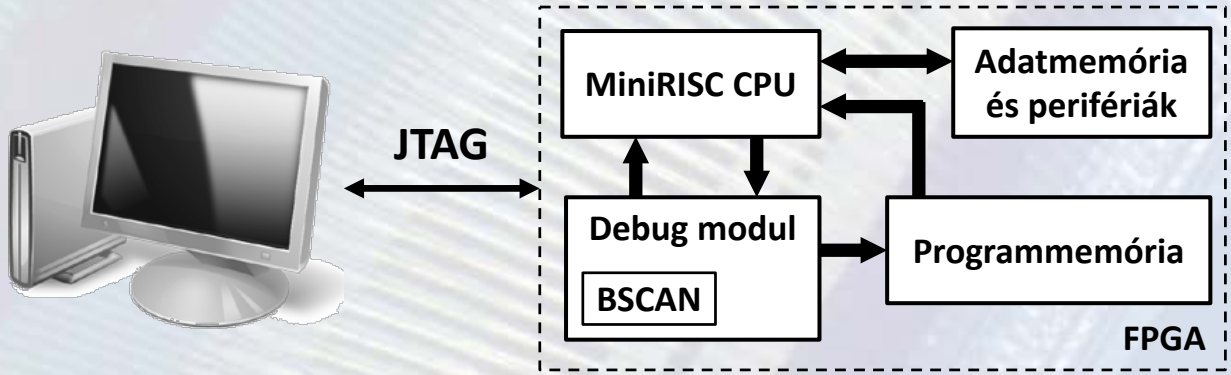
- **Az adatmemória interfész olvasási (read) ciklusa**
  - Az olvasási ciklust a végrehajtási (execute) fázisban 1 órajelciklus ideig aktív ***cpu2dmem_rd*** jel jelzi
  - Az olvasási ciklus alatt a ***cpu2dmem_addr*** címbuszon a cím stabil
  - Az olvasási ciklus alatt a kiválasztott periféria a ***dmem2cpu_data*** olvasási adatbuszra kapuzza az adatot, a többi periféria ezalatt inaktív nullával hajtja meg az adatkimenetét



MiniRISC processzor – Interfészek

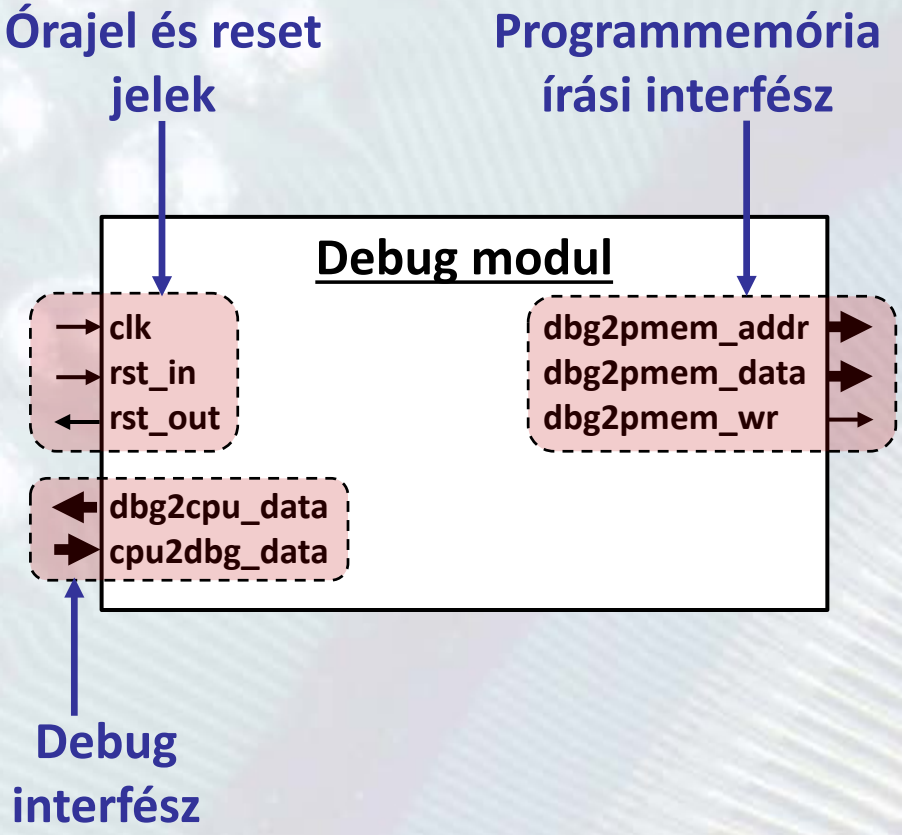
(Debug modul, debug interfész)

- **Debug modul: a fejlesztést, nyomkövetést és hibakeresést segíti**
  - Reset jel kiadása a processzoros rendszernek
  - Program letöltése (programmemória írás)
  - A regisztertömb, a PC, valamint a flag-ek írása és olvasása
  - Az adatmemória írása és olvasása
  - Töréspont elhelyezése tetszőleges programmemória címre
    - Töréspontra futáskor az utasítások végrehajtása felfüggesztődik
  - A program futásának megállítása és folytatása
- **A debug modul és a PC-n futó MiniRISC fejlesztői környezet közötti kommunikáció a JTAG interfészen keresztül történik**



MiniRISC processzor – Interfészek

(Debug modul, debug interfész)



A Verilog modul fejléce:

```
module debug_module(
   //Órajel és reset.
   input  wire         clk,
   input  wire         rst_in,
   output wire         rst_out,

   //A programmamória írásához
   //szükséges jelek.
   output wire [7:0]  dbg2pmem_addr,
   output wire [15:0] dbg2pmem_data,
   output wire         dbg2pmem_wr,

   //Debug interfész.
   output wire [22:0] dbg2cpu_data,
   input  wire [47:0] cpu2dbg_data
);
```

MiniRISC processzor – Interfészek

(Debug modul, debug interfész)

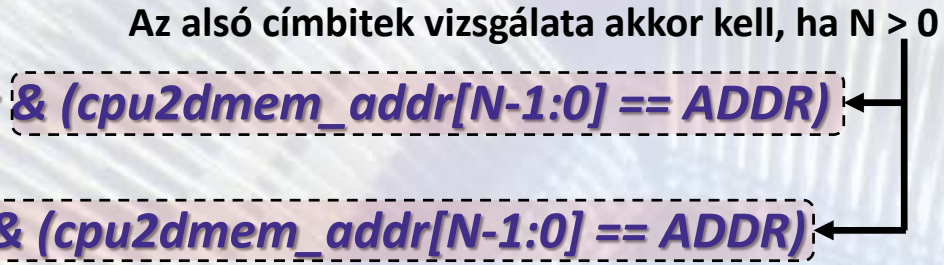
- **Interfész a MiniRISC processzor és a debug modul között**
  - ***dbg2cpu_data:*** jelek a debug modultól a processzor felé
  - ***cpu2dbg_data:*** jelek a processzortól a debug modul felé
  - Ha nincs debug modul a rendszerben, akkor a processzor ***dbg2cpu_data*** bemenetére kössünk konstans nullát
- **A debug modul egyéb vonalai**
  - Órajel és reset
    - ***clk:*** rendszerórajel (16MHz)
    - ***rst_in:*** külső reset jel (pl. reset nyomógomb)
    - ***rst_out:*** reset jel a processzoros rendszer számára
  - Programmemória írási interfész
    - ***dbg2pmem_addr:*** 8 bites címbusz
    - ***dbg2pmem_data:*** 16 bites írási adatbusz
    - ***dbg2pmem_wr:*** a programmemória írását engedélyező jel

Tartalom

MiniRISC processzor – Perifériaillesztés

A perifériaillesztési feladat lépései

- **A periféria típusa alapján az igények felmérése**
  - Regiszterek száma és használati módja (írható, olvasható)
    - Parancs, státusz, üzemmód, stb. regiszterek
  - Esetleg FIFO vagy kisebb memória blokk
- **A báziscím kijelölése, a címtartomány használatának megtervezése**
- **A címdekódolás kialakítása**
  - ***psel = ((cpu2dmem_addr >> N) == (BASEADDR >> N))***
  - A címtartomány mérete 2^N byte
- **Írás engedélyező jelek**
  - ***xxx_wr = psel & cpu2dmem_wr & (cpu2dmem_addr[N-1:0] == ADDR)***
- **Olvasás engedélyező jelek**
  - ***xxx_rd = psel & cpu2dmem_rd & (cpu2dmem_addr[N-1:0] == ADDR)***
  - A kimeneti MUX vezérlése: egy időben csak egy MUX kimenetén van érvényes adat, a többi periféria kimenetének értéke inaktív nulla
  - Használni kell még, ha az olvasás állapotváltozást okoz (pl. FIFO)



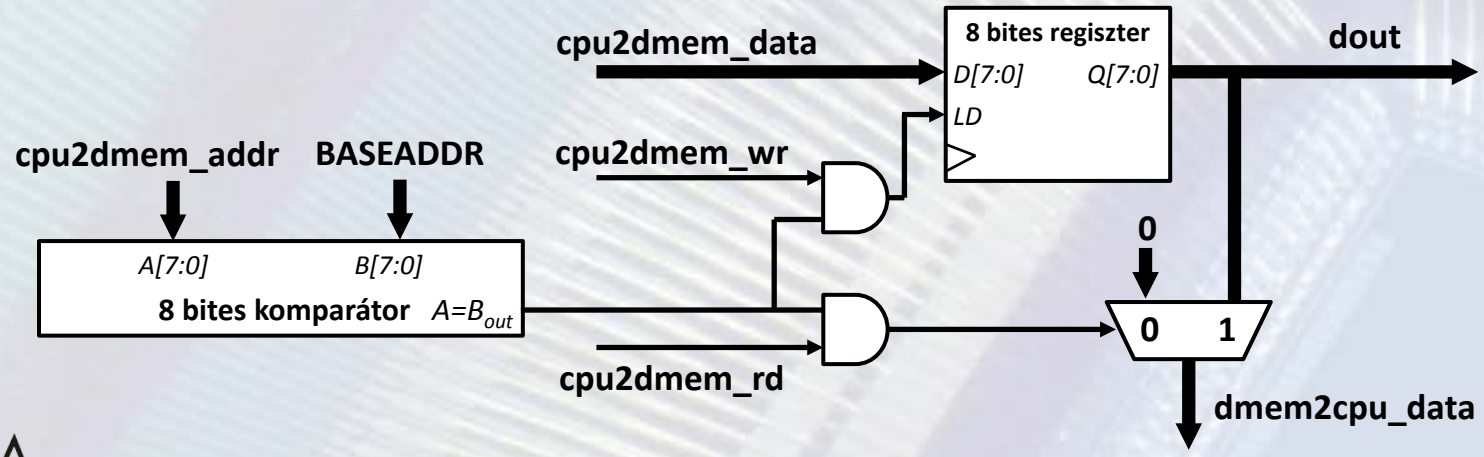
MiniRISC processzor – Perifériaillesztés

(1. példa – Specifikáció)

Feladat: 8 darab LED illesztése a processzoros rendszerhez, az állapot legyen visszaolvasható

- **Egyszerű, egy 8 bites írható és olvasható regiszter szükséges**
- **Adatregiszter: BASEADDR + 0x00, 8 bites, írható/olvasható**
  - Az OUT_i bit hajtja meg az i-edik LED-et

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W



MiniRISC processzor – Perifériaillesztés

(1. példa – Megvalósítás Verilog nyelven)

Feladat: 8 darab LED illesztése a processzoros rendszerhez

```
module basic_owr #(
   //A periféria báziscíme.
   parameter BASEADDR = 8'hff
) (
   //Órajel és reset.
   input  wire       clk,
   input  wire       rst,

   //Adatmemória interfész.
   input  wire [7:0] cpu2dmem_addr,
   input  wire       cpu2dmem_wr,
   input  wire       cpu2dmem_rd,
   input  wire [7:0] cpu2dmem_data,
   output reg  [7:0] dmem2cpu_data,

   //Kimenő adat.
   output reg  [7:0] dout
);

//A periféria kiválasztó jele.
wire psel = (cpu2dmem_addr == BASEADDR);
```

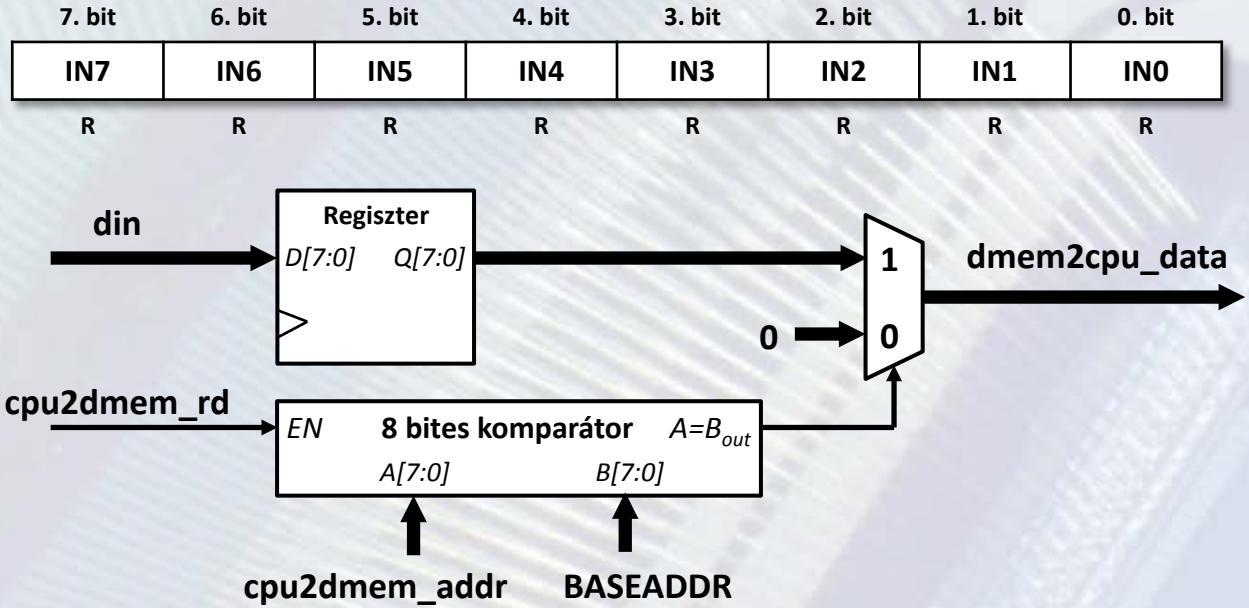
```
//Az adatreg. írás engedélyező jele.
wire dreg_wr = psel & cpu2dmem_wr;

//Az adatreg. olvasás engedélyező jele.
wire dreg_rd = psel & cpu2dmem_rd;

//Adatregiszter.
always @(posedge clk)
   if (rst)
      dout <= 8'd0;
   else
      if (dreg_wr)
         dout <= cpu2dmem_data;

//Az olvasási adatbusz meghajtása.
always @(*)
   if (dreg_rd)
      dmem2cpu_data <= dout;
   else
      dmem2cpu_data <= 8'd0;

endmodule
```

MiniRISC processzor – Perifériaillesztés

(2. példa – Specifikáció)

Feladat: 8 darab kapcsoló illesztése a processzoros rendszerhez

- **Egyszerű, egy 8 bites csak olvasható regiszter szükséges, amely folyamatosan mintavételezi a 8 kapcsoló állapotát**
- **Adatregiszter: BASEADDR + 0x00, 8 bites, csak olvasható**
  - Az IN_i bit az i-edik kapcsolón beállított értéket veszi fel



MiniRISC processzor – Perifériaillesztés

(2. példa – Megvalósítás Verilog nyelven)

Feladat: 8 darab kapcsoló illesztése a processzoros rendszerhez

```
module basic_in #(
   //A periféria báziscíme.
   parameter BASEADDR = 8'hff
) (
   //Órajel és reset.
   input  wire       clk,
   input  wire       rst,

   //Adatmemória interfész.
   input  wire [7:0] cpu2dmem_addr,
   input  wire       cpu2dmem_rd,
   output reg  [7:0] dmem2cpu_data,

   //Bejövő adat.
   input  wire [7:0] din
);

//A periféria kiválasztó jele.
wire psel = (cpu2dmem_addr == BASEADDR);
```

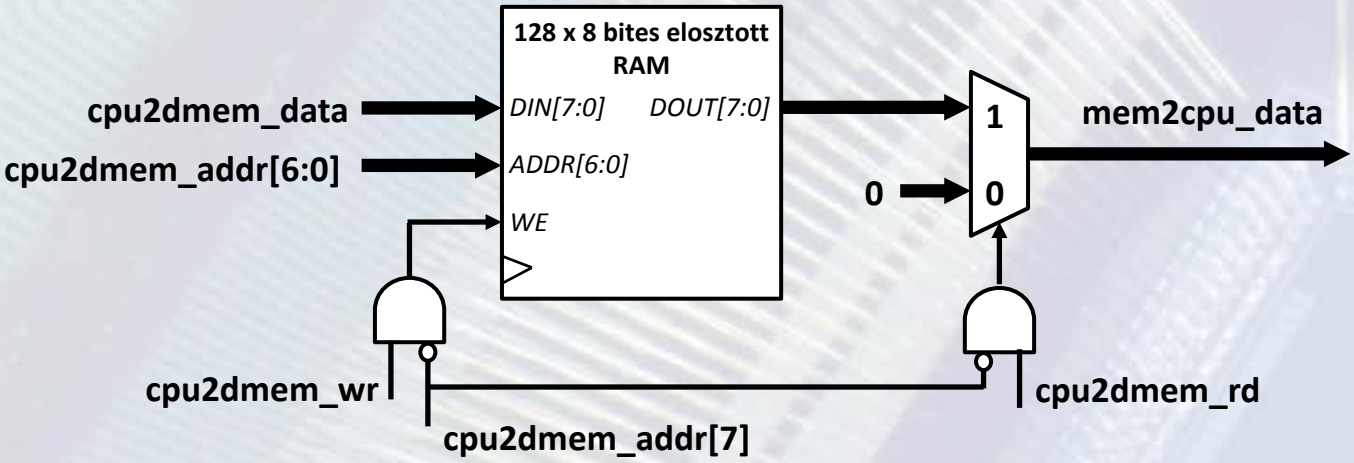
```
//Az adatreg. olvasás engedélyező jele.
wire in_reg_rd = psel & cpu2dmem_rd;

//Adatregiszter.
reg [7:0] in_reg;

always @(posedge clk)
   if (rst)
      in_reg <= 8'd0;
   else
      in_reg <= din;

//Az olvasási adatbusz meghajtása.
always @(*)
   if (in_reg_rd)
      dmem2cpu_data <= in_reg;
   else
      dmem2cpu_data <= 8'd0;

endmodule
```

MiniRISC processzor – Perifériaillesztés

(3. példa – Specifikáció)

Feladat: 128 x 8 bites memória illesztése a processzoros rendszerhez

- **A memóriának 7 címbitje van ($128 = 2^7$)**
  - Címtartomány: **00000000 (0x00) – 01111111 (0x7F)**
- **A címdekóder logika megvalósítása**
  - A címbusz alsó 7 bitje kapcsolódik a memóriához
  - A legfelső címbitet (MSb) használjuk a címdekódoláshoz
    - Ha ez 0, akkor a memória ki van választva



MiniRISC processzor – Perifériaillesztés

(3. példa – Megvalósítás Verilog nyelven)

Feladat: 128 x 8 bites memória illesztése a processzoros rendszerhez

```
wire       memsel   = ~cpu2dmem_addr[7];     //A memória kiválasztó jele.
wire [6:0] mem_addr = cpu2dmem_addr[6:0];    //A memória címbusza.
wire       mem_wr   = memsel & cpu2dmem_wr;  //A memória írás engedélyező jele.
wire       mem_rd   = memsel & cpu2dmem_rd;  //A memória olvasás engedélyező jele.

//128 x 8 bites elosztott RAM deklarálása.
(* ram_style = ”distributed” *)
reg [7:0] mem [127:0]

//Az elosztott RAM írási portja.
//Az elosztott RAM írása szinkron művelet.
always @(posedge clk)
   if (mem_wr)
      mem[mem_addr] <= cpu2dmem_data;

//Az elosztott RAM olvasási portja és az olvasási adatbusz meghajtása
//(ezt kell VAGY kapuval rákötni a processzor olvasási adatbuszára).
//Az elosztott RAM olvasása aszinkron művelet.
wire [7:0] mem2cpu_data = (mem_rd) ? mem[mem_addr] : 8’d0;
```

MiniRISC processzor – Perifériaillesztés

(4. példa – Specifikáció)

- **Feladat: 4 nyomógomb illesztése a MiniRISC processzorhoz**
  - Pergésmentesítés 200 Hz-en történő mintavételezéssel
  - Megszakításkérés, ha egy bemenet értéke megváltozott
- **A perifériában kialakítandó regiszterek**
  - Egy 8 bites adatregiszter, amelyből beolvasható a bemenetek aktuális állapota
  - Egy 8 bites megszakítás engedélyező regiszter
    - Minden bemenethez egy-egy megszakítás engedélyező bit
  - Egy 8 bites megszakítás flag regiszter
    - Minden bemenethez egy-egy megváltozást jelző bit
  - A regiszterek felső 4 bitje nincs használatban
- **A periféria 4 byte méretű címtartományt kap**
  - 3 byte kellene csak, de 2^N méretet könnyebb dekódolni

MiniRISC processzor – Perifériaillesztés

(4. példa – Regiszterkészlet)

- **Adatregiszter**
  - BASEADDR + 0x00, 8 bites, csak olvasható
  - Az IN_i bit az i-edik bemenet értékét adja meg

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
0	0	0	0	IN3	IN2	IN1	IN0
R	R	R	R	R	R	R	R

- **Megszakítás engedélyező (Interrupt Enable, IE) regiszter**
  - BASEADDR + 0x01, 8 bites, írható és olvasható
  - Az IE_i bit engedélyezi az i-edik bemenetre a megszakításkérést

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
0	0	0	0	IE3	IE2	IE1	IE0
R	R	R	R	R/W	R/W	R/W	R/W

- **Megszakítás flag (Interrupt Flag, IF) regiszter**
  - BASEADDR + 0x02, 8 bites, írható és olvasható
  - Az IF_i bit jelzi az i-edik bemenet megváltozását, 1 beírásával törölhető

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
0	0	0	0	IF3	IF2	IF1	IF0
R	R	R	R	R/W1C	R/W1C	R/W1C	R/W1C

MiniRISC processzor – Perifériaillesztés

(4. példa – Megvalósítás Verilog nyelven)

A modul fejléce

- ***BASEADDR*** **paraméter: a periféria (néggyel osztható) báziscíme**

```
module btn4_in_irq #(
   parameter BASEADDR = 8'hff                  //A periféria báziscíme
) (
   //Órajel és reset.
   input  wire             clk,                //Órajel
   input  wire             rst,                //Reset jel

   //A busz interfész jelei.
   input  wire [7:0]       cpu2dmem_addr,      //Címbusz
   input  wire             cpu2dmem_wr,        //Írás engedélyező jel
   input  wire             cpu2dmem_rd,        //Olvasás engedélyező jel
   input  wire [7:0]       cpu2dmem_data,      //Írási adatbusz
   output reg  [7:0]       dmem2cpu_data,      //Olvasási adatbusz

   //Megszakításkérő kimenet.
   output reg              irq,

   //A nyomógombok aktuális értéke.
   input  wire [3:0]       btn_in
);
```

MiniRISC processzor – Perifériaillesztés

(4. példa – Megvalósítás Verilog nyelven)

Címdekódolás

- A periféria kiválasztó jelének előállítása
- A regiszterek írás engedélyező jeleinek előállítása
- A regiszterek olvasás engedélyező jeleinek előállítása

```
//A periféria kiválasztó jele.
//4 bájtos címtartomány -> kettővel kell jobbra shiftelni.
wire psel    = ((cpu2dmem_addr >> 2) == (BASEADDR >> 2));

//A regiszterek írás engedélyező jelei.
//Írható regiszterek: IE regiszter és IF regiszter.
wire ie_wr   = psel & cpu2dmem_wr & (cpu2dmem_addr[1:0] == 2'b01);
wire if_wr   = psel & cpu2dmem_wr & (cpu2dmem_addr[1:0] == 2'b10);

//A regiszterek olvasás engedélyező jelei.
//Olvasható regiszterek: mindegyik.
wire dreg_rd = psel & cpu2dmem_rd & (cpu2dmem_addr[1:0] == 2'b00);
wire ie_rd   = psel & cpu2dmem_rd & (cpu2dmem_addr[1:0] == 2'b01);
wire if_rd   = psel & cpu2dmem_rd & (cpu2dmem_addr[1:0] == 2'b10);
```

MiniRISC processzor – Perifériaillesztés

(4. példa – Megvalósítás Verilog nyelven)

Az órajel osztó és az adatregiszter

```
//A 200 Hz-es engedélyező jelet előállító számláló.
//A modulus: 16*10^6 Hz / 200 Hz = 80000 -> 79999 - 0.
reg  [16:0] clk_div;
wire        clk_div_tc = (clk_div == 17'd0);

always @(posedge clk)
   if (rst || clk_div_tc)           //Reset vagy végállapot:
      clk_div <= 17'd79999;         //a kezdőállapot betöltése
   else
      clk_div <= clk_div - 17'd1;   //Egyébként lefele számlálás

//A bemenet mintavételezése. Ez egyben az adatregiszter is.
reg [3:0] dreg;

always @(posedge clk)
   if (rst)
      dreg <= 4'd0;                 //Reset: az adatregiszter törlése
   else
      if (clk_div_tc)
         dreg <= btn_in;            //A bemenet mintavételezése
```

MiniRISC processzor – Perifériaillesztés

(4. példa – Megvalósítás Verilog nyelven)

Az IE regiszter és a bemeneti változás detektálása

```
//A megszakítás engedélyező regiszter.
reg [3:0] ie_reg;

always @(posedge clk)
   if (rst)
      ie_reg <= 4'd0;                    //Reset: az IE reg. törlése
   else
      if (ie_wr)
         ie_reg <= cpu2dmem_data[3:0]; //Az IE regiszter írása

//Az adatregiszter előző értékének tárolása a változás detektálásához.
//A változás detektálása XOR kapuval lehetséges (0->1 vagy 1->0).
reg  [3:0] dreg_prev;
wire [3:0] dreg_changed = dreg ^ dreg_prev;

always @(posedge clk)
   if (rst)
      dreg_prev <= 4'd0;                 //Reset: törlés
   else
      dreg_prev <= dreg;                 //A korábbi érték tárolása
```

MiniRISC processzor – Perifériaillesztés

(4. példa – Megvalósítás Verilog nyelven)

Az IF regiszter és az IRQ kimenet meghajtása

```
//A megszakítás flag regiszter.
reg [3:0] if_reg;
integer i;

always @(posedge clk)
   for (i = 0; i < 4; i = i + 1)              //FOR ciklus az indexeléshez
      if (rst)
         if_reg[i] <= 1'b0;                   //Reset: törlés
      else
         if (dreg_changed[i])
            if_reg[i] <= 1'b1;                //Bemeneti változás
         else
            if (if_wr && cpu2dmem_data[i])    //Törlés 1 beírásával
               if_reg[i] <= 1'b0;

//A megszakításkérő kimenet meghajtása. Megszakítást kell kérni
//ha valamelyik IF bit 1 (bemeneti változás) és a hozzá tartozó
//IE bit 1 (engedélyezett megszakítás).
always @(posedge clk)
   irq <= |(if_reg & ie_reg);
```

MiniRISC processzor – Perifériaillesztés

(4. példa – Megvalósítás Verilog nyelven)

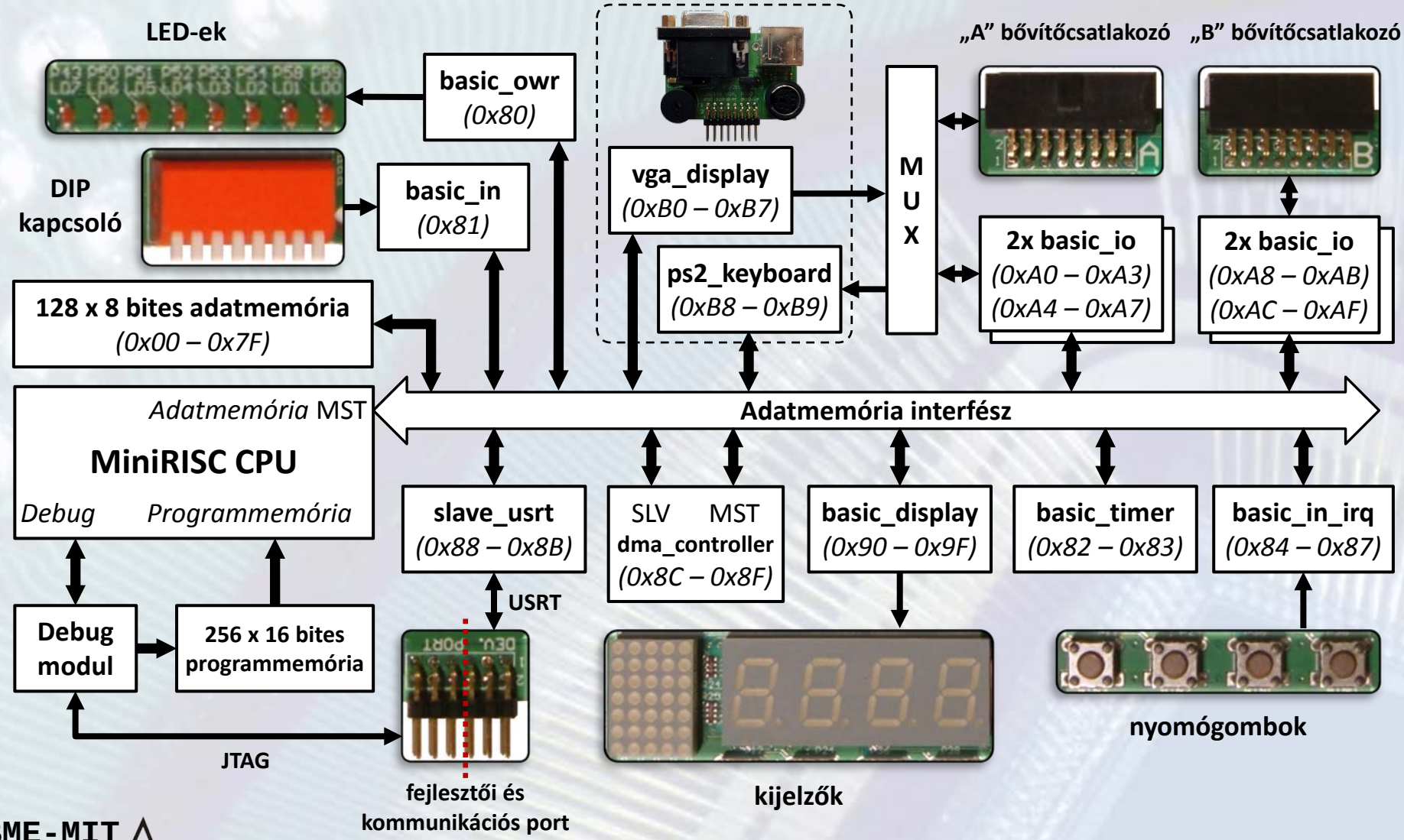
Az olvasási adatbusz meghajtása

- A regiszterek felső 4 bitje nincs használatban, nullát adnak vissza
- Ha nincs egy regiszter sem kiválasztva olvasásra, akkor az olvasási adatbuszra az inaktív nulla érték kerül

```
//Az olvasási adatbusz meghajtása.
wire [2:0] dout_sel;

assign dout_sel[0] = dreg_rd;
assign dout_sel[1] = ie_rd;
assign dout_sel[2] = if_rd;

always @(*)
   case (dout_sel)
      3'b001 : dmem2cpu_data <= {4'd0, dreg};     //Adatregiszter
      3'b010 : dmem2cpu_data <= {4'd0, ie_reg};   //IE regiszter
      3'b100 : dmem2cpu_data <= {4'd0, if_reg};   //IF regiszter
      default: dmem2cpu_data <= 8'd0;             //Inaktív 0 érték
   endcase

endmodule
```

MiniRISC mintarendszer

(Blokkvázlat)



MiniRISC mintarendszer

(Az adatmemória interfész címkiosztása)

Címtartomány	Méret	Periféria	Funkció
0x00 – 0x7F	128 byte	adatmemória	128 x 8 bites memória
0x80	1 byte	basic_owr	LED-ek illesztése
0x81	1 byte	basic_in	DIP kapcsoló illesztése
0x82 – 0x83	2 byte	basic_timer	időzítés
0x84 – 0x87	4 byte	basic_in_irq	nyomógombok illesztése
0x88 – 0x8B	4 byte	slave_usrt	soros kommunikáció
0x8C – 0x8F	4 byte	dma_controller	DMA vezérlő
0x90 – 0x9F	16 byte	basic_display	hétszegmenses és pontmátrix kijelzők illesztése
0xA0 – 0xA3	4 byte	basic_io (GPIO A)	„A” bővítőcsatlakozó illesztése
0xA4 – 0xA7	4 byte	basic_io (GPIO C)	
0xA8 – 0xAB	4 byte	basic_io (GPIO B)	„B” bővítőcsatlakozó illesztése
0xAC – 0xAF	4 byte	basic_io (GPIO D)	
0xB0 – 0xB7	8 byte	vga_display	VGA monitor illesztése
0xB8 – 0xB9	2 byte	ps2_keyboard	PS/2 billentyűzet illesztése
0xBA – 0xFF	70 byte	szabadon felhasználható a későbbi bővítéshez	

MiniRISC mintarendszer

(Perifériák – basic_owr és basic_in)

- ***basic_owr:* 8 bites visszaolvasható kimeneti periféria**
  - Egyszerű kimenet, alapértéke 0x00
  - Adatregiszter: BASEADDR + 0x00, írható és olvasható
    - Az adatregiszter OUT_i bitje beállítja az i-edik kimeneti bit értékét

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

  - A LED-ek illesztéséhez használjuk
- ***basic_in:* 8 bites bemeneti periféria**
  - Egyszerű bemenet folyamatos mintavételezéssel
  - Adatregiszter: BASEADDR + 0x00, csak olvasható
    - Az adatregiszter IN_i bitje az i-edik bemenet állapotát tükrözi

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
IN7	IN6	IN5	IN4	IN3	IN2	IN1	IN0
R	R	R	R	R	R	R	R

  - A DIP kapcsoló illesztéséhez használjuk

MiniRISC mintarendszer

(Perifériák – basic_in_irq)

***basic_in_irq:* 8 bites bemeneti periféria pergésmentesítéssel és megszakításkéréssel**

- **Adatregiszter**
  - BASEADDR + 0x00, 8 bites, csak olvasható
  - Az IN_i bit az i-edik bemenet értékét adja meg

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
IN7	IN6	IN5	IN4	IN3	IN2	IN1	IN0
R	R	R	R	R	R	R	R

- **Megszakítás engedélyező (Interrupt Enable, IE) regiszter**
  - BASEADDR + 0x01, 8 bites, írható és olvasható
  - Az IE_i bit engedélyezi az i-edik bemenetre a megszakításkérést

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
IE7	IE6	IE5	IE4	IE3	IE2	IE1	IE0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- **Megszakítás flag (Interrupt Flag, IF) regiszter**
  - BASEADDR + 0x02, 8 bites, írható és olvasható
  - Az IF_i bit jelzi az i-edik bemenet megváltozását, 1 beírásával törölhető

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
IF7	IF6	IF5	IF4	IF3	IF2	IF1	IF0
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C

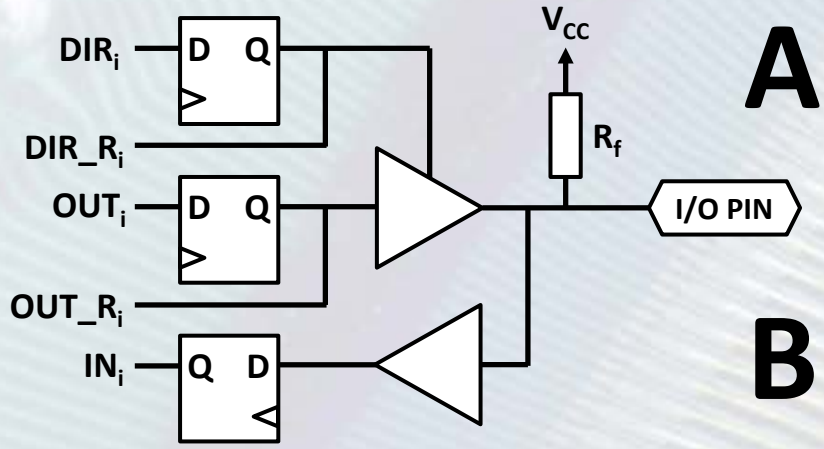
- **A nyomógombok illesztéséhez használjuk (a regiszterek felső 4 bitje nem használt)**

MiniRISC mintarendszer

(Perifériák – basic_io)

***basic_io:* 8 bites GPIO (General Purpose I/O) periféria**

- **A GPIO vonalak egyenként konfigurálhatók**
  - Kimenet, bemenet, kétirányú I/O vonal
- **A GPIO vonalak felhúzó ellenállást tartalmaznak**
  - Ha a láb bemenetként nincs vezérelve, akkor értéke magas lesz
- **A *basic_io* perifériák az FPGA kártya „A” és „B” bővítőcsatlakozóit illesztik a rendszerhez a lenti ábrák szerinti lábkiosztással**



A:

15 (I) C[3]	13 (I/O) A[6]	11 (I/O) A[4]	9 (I/O) A[2]	7 (I/O) A[0]	5 (I/O) C[1]	3 (PWR) +3,3V	1 (PWR) GND
16 (I) C[4]	14 (I/O) A[7]	12 (I/O) A[5]	10 (I/O) A[3]	8 (I/O) A[1]	6 (I/O) C[2]	4 (I/O) C[0]	2 (PWR) +5V

B:

15 (I) D[3]	13 (I/O) B[6]	11 (I/O) B[4]	9 (I/O) B[2]	7 (I/O) B[0]	5 (I/O) D[1]	3 (PWR) +3,3V	1 (PWR) GND
16 (I) D[4]	14 (I/O) B[7]	12 (I/O) B[5]	10 (I/O) B[3]	8 (I/O) B[1]	6 (I/O) D[2]	4 (I/O) D[0]	2 (PWR) +5V

MiniRISC mintarendszer

(Perifériák – basic_io)

basic_io: **8 bites GPIO (General Purpose I/O) periféria**

- **3 regisztert tartalmaz → 4 bájtos címtartomány**
- **Kimeneti adatregiszter: BASEADDR + 0x00, írható és olvasható**
  - Az OUT_i bit hajtja meg az adott I/O vonalat, ha az kimenet

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- **Bemeneti adatregiszter: BASEADDR + 0x01, csak olvasható**
  - Az IN_i bit az adott I/O vonal állapotát tükrözi

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
IN7	IN6	IN5	IN4	IN3	IN2	IN1	IN0
R	R	R	R	R	R	R	R

- **Irányvezérlő regiszter: BASEADDR + 0x02, írható és olvasható**
  - Az I/O vonal iránya: $DIR_i = 0$ → bemenet, $DIR_i = 1$ → kimenet

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
DIR7	DIR6	DIR5	DIR4	DIR3	DIR2	DIR1	DIR0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

MiniRISC mintarendszer

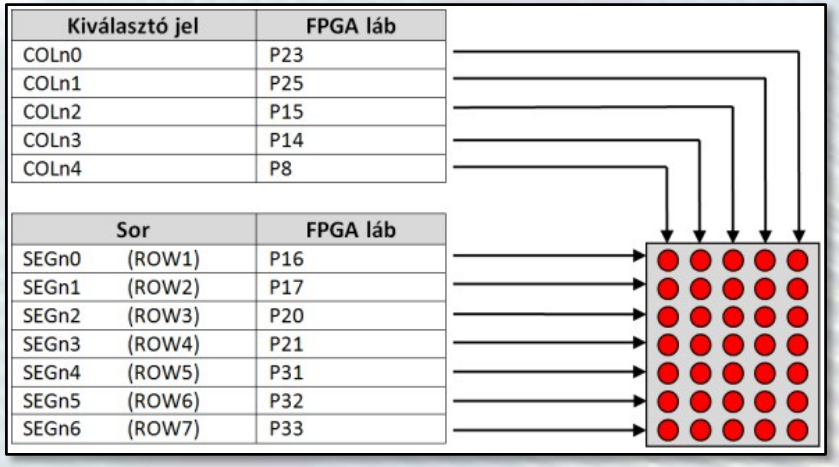
(Perifériák – basic_display)

***basic_display:* a kijelzőket vezérlő periféria**

- **A kijelzők bekötése az FPGA kártyán**
  - Közös kimeneti adatvonalak a megjelenítéshez (aktív alacsony)
    - A hétszegmenses kijelzőnél az adatvonalak egy digitet hajtanak meg
    - A pontmátrix kijelzőnél az adatvonalak egy oszlopot hajtanak meg
  - Egyedi digit- és oszlopkiválasztó jelek (aktív alacsony)
- **A kijelzők vezérlése időmultiplexelt módon történik**
  - A periféria ezt hardveresen megoldja

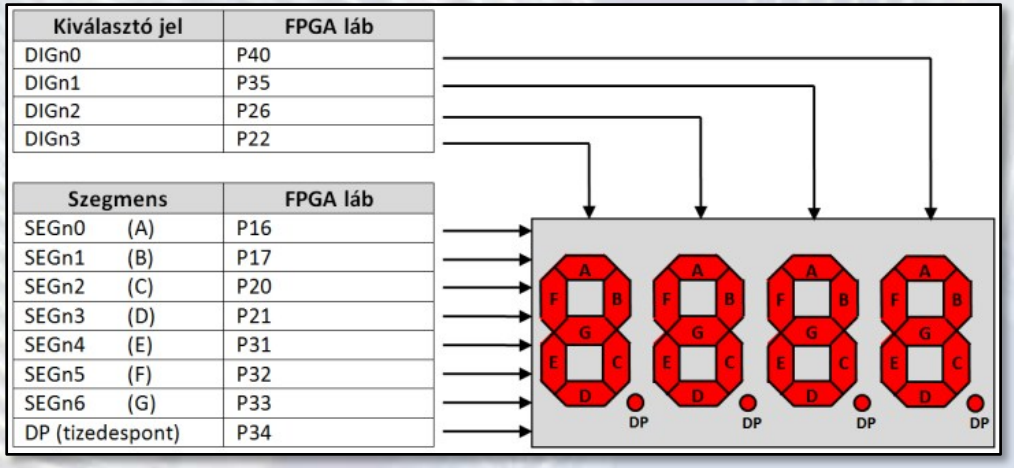
Kiválasztó jel	FPGA láb
COLn0	P23
COLn1	P25
COLn2	P15
COLn3	P14
COLn4	P8

Sor		FPGA láb
SEGn0	(ROW1)	P16
SEGn1	(ROW2)	P17
SEGn2	(ROW3)	P20
SEGn3	(ROW4)	P21
SEGn4	(ROW5)	P31
SEGn5	(ROW6)	P32
SEGn6	(ROW7)	P33



Kiválasztó jel	FPGA láb
DIGn0	P40
DIGn1	P35
DIGn2	P26
DIGn3	P22

Szegmens		FPGA láb
SEGn0	(A)	P16
SEGn1	(B)	P17
SEGn2	(C)	P20
SEGn3	(D)	P21
SEGn4	(E)	P31
SEGn5	(F)	P32
SEGn6	(G)	P33
DP (tizedespont)		P34

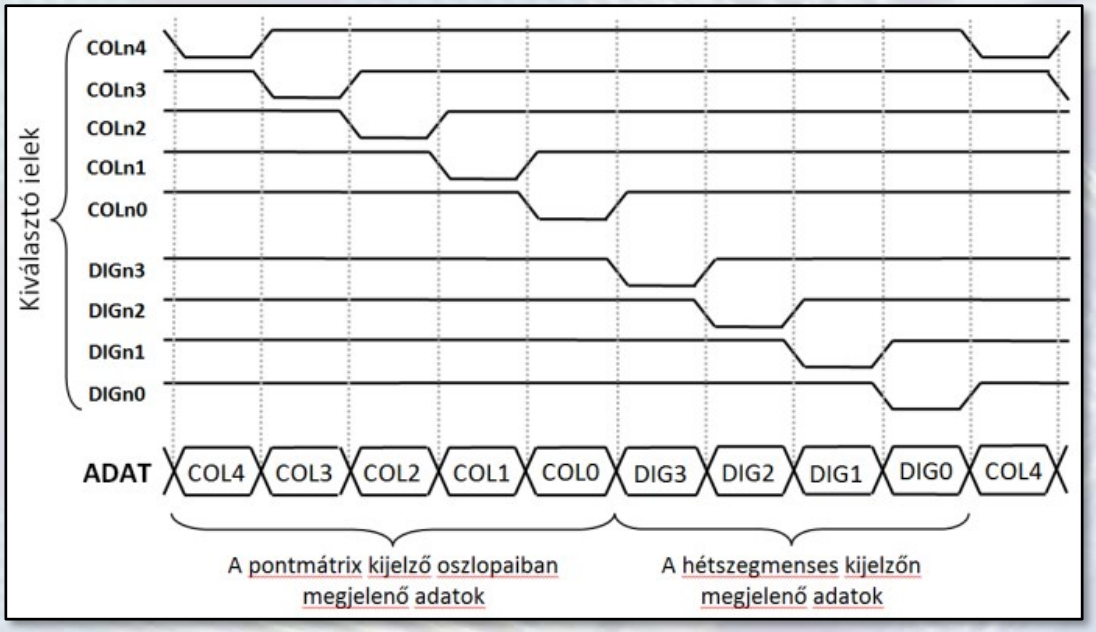


MiniRISC mintarendszer

(Perifériák – basic_display)

***basic_display:* a kijelzőket vezérlő periféria**

- **A kijelzők vezérlése időmultiplexelt módon történik**
  - 4 digit és 5 oszlop → 9 fázisból álló ciklus szükséges
  - Minden egyes fázisban aktiválódik az adott digithez vagy oszlophoz tartozó kiválasztó jel és az adatvonal felveszi a megjelenítendő mintához tartozó értéket



MiniRISC mintarendszer

(Perifériák – basic_display)

basic_display: **a kijelzőket vezérlő periféria**

- **9 regisztert tartalmaz → 16 bájtos címtartomány**
- **4 adatregiszter a hétszegmenses kijelző 4 digitjéhez (DIG0 – DIG3)**
  - BASEADDR + (0x00 – 0x03), írható és olvasható
  - Egy-egy bit tartozik a digitek egy-egy szegmenséhez

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
DP	G	F	E	D	C	B	A
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- **5 adatregiszter a pontmátrix kijelző 5 oszlopához (COL0 – COL4)**
  - BASEADDR + (0x04 – 0x08), írható és olvasható
  - Egy-egy bit tartozik az oszlopok egy-egy pontjához
    - A 7. bit nincs felhasználva a vezérléshez

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
-	ROW7	ROW6	ROW5	ROW4	ROW3	ROW2	ROW1
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- **Reset hatására az adatregiszterek értéke 0x00 lesz**

MiniRISC mintarendszer

(Perifériák – basic_timer)

***basic_timer:* időzítő periféria**

- **Felépítés: előosztó és egy 8 bites lefele számláló**
  - A számláló az előosztó által meghatározott ütemben számlál
- **Számláló kezdőállapot regiszter (TR)**
  - BASEADDR + 0x00, csak írható
  - A számláló kezdőállapota az időzítés mértékét határozza meg

7. bit	6. bit	5. bit	4. Bit	3. bit	2. bit	1. bit	0. bit
TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
W	W	W	W	W	W	W	W

- **Számláló regiszter (TM)**
  - BASEADDR + 0x00, csak olvasható
  - Az időzítő számlálójának aktuális értéke

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
TM7	TM6	TM5	TM4	TM3	TM2	TM1	TM0
R	R	R	R	R	R	R	R

MiniRISC mintarendszer

(Perifériák – basic_timer)

basic_timer: időzítő periféria

- Parancs regiszter (TC): BASEADDR + 0x01, csak írható

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
TIE	TPS2	TPS1	TPS0	-	-	TREP	TEN
W	W	W	W	n.a.	n.a.	W	W

- Státusz regiszter (TS): BASEADDR + 0x01, csak olvasható

7. bit	6. bit	5. bit	4. Bit	3. bit	2. bit	1. bit	0. bit
TIT	TPS2	TPS1	TPS0	0	TOUT	TREP	TEN
R	R	R	R	R	R	R	R

Bit	Funkció
TEN	Engedélyező bit (0: a működés tiltott, 1: a működés engedélyezett)
TREP	Működési mód kiválasztása (0: egyszeri, 1: ismétlődéses)
TOUT	Időzítési periódus lejárt jelzőbit
TPS[2:0]	Az előosztás (PS) mértékét beállító bitek 0 : nincs előosztás 1 – 7 : $2^{2\cdot(TPS+1)}$ előosztás (16, 64, 256, 1024, 4096, 16384 vagy 65536)
TIE / TIT	Időzítő megszakítás engedélyezés és jelzés bitek

MiniRISC mintarendszer

(Perifériák – basic_timer)

basic_timer: **időzítő periféria**

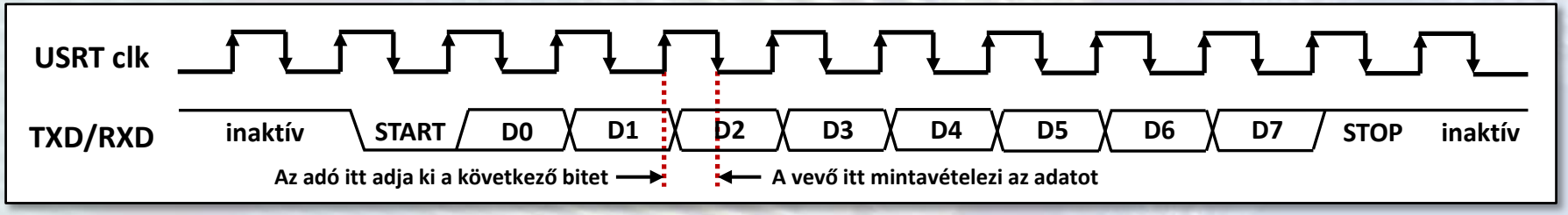
- **Az időzítő periódusideje:** $T = (TR + 1) \cdot PS \cdot T_{CLK}$
  - TR az időzítő számláló kezdőállapota
  - PS az előosztás (1, 16, 64, 256, 1024, 4096, 16384 vagy 65536)
  - T_{CLK} a rendszerórajel periódusideje (16 MHz → T_{CLK}=62,5 ns)
- **A maximálisan beállítható periódusidő 1,048576 s**
- **A paraméterek (TR, TPS, TREP) beállítása után az időzítő a TEN bit beállításával indítható**
- **Egyszeri üzemmód (TREP=0) esetén a számláló a nulla végállapot elérésekor leáll, ismétlődéses üzemmód (TREP=1) esetén újból betöltődik a számlálóba a TR regiszter értéke**
- **Az időzítési periódus leteltekor a TOUT bit beállítódik, amit a státusz regiszter kiolvasásával törölhetünk**
- **Ha a megszakításkérés engedélyezett (TIE=1), akkor a TOUT bit egyúttal aktiválja a megszakításkérést is**

MiniRISC mintarendszer

(Perifériák – slave_usrt)

***slave_usrt:* USRT soros kommunikációt biztosító periféria**

- **USRT adatátvitel**
  - Keretezett formátum:
    - 1 START bit, melynek értéke mindig 0
    - 8 adatbit (D0 – D7)
    - 1 STOP bit, melynek értéke mindig 1
  - USRT órajel: az adatátviteli sebességet határozza meg
    - A master egység adja ki a slave egység felé
  - Adás: a bitek kiadása az USRT órajel felfutó élére történik
  - Vétel: a mintavételezés az USRT órajel lefutó élére történik
    - Csak a kerethiba mentes (STOP bit = 1) karakterek kerülnek tárolásra



MiniRISC mintarendszer

(Perifériák – slave_usrt)

slave_usrt: **USRT soros kommunikációt biztosító periféria**

- **Vezérlő regiszter (UC): BASEADDR + 0x00, írható és olvasható**

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
0	0	0	0	RXCLR	TXCLR	RXEN	TXEN
R	R	R	R	W	W	R/W	R/W

Bit	Mód	Funkció
TXEN	R/W	0: az USRT adó tiltott 1: az USRT adó engedélyezett
RXEN	R/W	0: az USRT vevő tiltott 1: az USRT vevő engedélyezett
TXCLR	W	1 beírásának hatására az adási FIFO törlődik
RXCLR	W	1 beírásának hatására a vételi FIFO törlődik

- **Adatregiszter (UD): BASEADDR + 0x03, írható és olvasható**
  - Írás: adat írása az adási FIFO-ba (ha TXNF=1)
  - Olvasás: adat olvasása a vételi FIFO-ból (ha RXNE=1)

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

MiniRISC mintarendszer

(Perifériák – slave_usrt)

slave_usrt: **USRT soros kommunikációt biztosító periféria**

- **FIFO státusz regiszter (US): BASEADDR + 0x01, csak olvasható**

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
0	0	0	0	RXFULL	RXNE	TXNF	TXEMPTY
R	R	R	R	R	R	R	R

- **Megszakítás eng. regiszter (UIE): BASEADDR + 0x02, írható és olvasható**
  - A FIFO státusz megszakítások engedélyezhetők/tilthatók vele
  - A megszakításkérés az engedélyezett események fennállásáig aktív

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
0	0	0	0	RXFULL	RXNE	TXNF	TXEMPTY
R	R	R	R	R/W	R/W	R/W	R/W

Bit	Jelentés	
TXEMPTY	0: az adási FIFO-ban van adat	1: az adási FIFO üres
TXNF	0: az adási FIFO tele van	1: az adási FIFO nincs tele
RXNE	0: a vételi FIFO üres	1: a vételi FIFO-ban van adat
RXFULL	0: a vételi FIFO nincs tele	1: a vételi FIFO tele van

MiniRISC mintarendszer

(Perifériák – vga_display)

vga_display: **VGA monitor illesztését biztosító periféria (1024x768 @ 60 Hz)**

- **A használatához szükség van a *LOGSYS VGA, PS/2 és hangszóró modul* csatlakoztatására az FPGA kártya „A” bővítőcsatlakozójára**
  - A VGA interfészről részletek a modul dokumentációjában
- **Grafikus (256 x 192 pixel) és karakteres (32 x 24 karakter) üzemmód**
- **Vezérlő/státusz regiszter (VC/VS): BASEADDR + 0x00, írható és olvasható**

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
VBLANK	IRQ	0	0	0	INCSEL	MODE	VEN
R	R	R	R	R	R/W	R/W	R/W

Bit	Mód	Funkció
VEN	R/W	0: a VGA periféria tiltott 1: a VGA periféria engedélyezett Ha a VGA periféria engedélyezett, akkor GPIO A és C tiltódik
MODE	R/W	0: grafikus üzemmód 1: karakteres üzemmód
INCSEL	R/W	0: X irányú írás/olvasás 1: Y irányú írás/olvasás
IRQ	R	a VGA periféria megszakításkérő kimenetének állapota
VBLANK	R	0: nincs vertikális visszafutás 1: vertikális visszafutás

MiniRISC mintarendszer

(Perifériák – vga_display)

vga_display: **VGA monitor illesztését biztosító periféria (1024x768 @ 60 Hz)**

- **Megszakítás engedélyező regiszter (VIE): BASEADDR + 0x01, írható és olvasható**
  - A vertikális visszafutás megszakítás engedélyezése/tiltása

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
0	0	0	0	0	0	0	VBLIE
R	R	R	R	R	R	R	R/W

- **Megszakítás flag regiszter (VIF): BASEADDR + 0x02, írható és olvasható**
  - A VBLIF a vertikális visszafutás kezdetekor beállítódik és a vertikális visszafutás végén, illetve 1 beírásának hatására törlődik

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
0	0	0	0	0	0	0	VBLIF
R	R	R	R	R	R	R	R/W1C

- **Adatregiszter (VD): BASEADDR + 0x03, írható és olvasható**
  - Grafikus mód: pixel színe

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
0	0	0	0	0	pixel B	pixel G	pixel R
R	R	R	R	R	R/W	R/W	R/W

  - Karakteres mód: 1. hozzáférés → ASCII kód, 2. hozzáférés → színek és villogás

	7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
1.	a karakter ASCII kódja							
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
2.	villogás	-	háttér B	háttér G	háttér R	karakter B	karakter G	karakter R
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

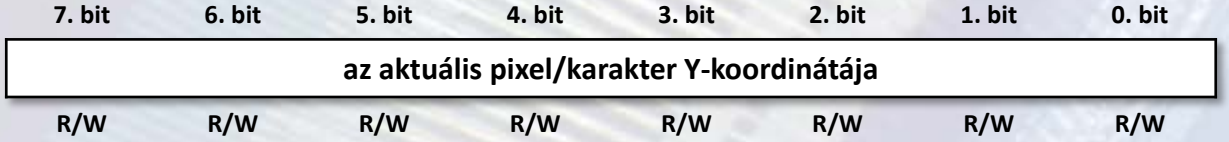
MiniRISC mintarendszer

(Perifériák – vga_display)

vga_display: **VGA monitor illesztését biztosító periféria (1024x768 @ 60 Hz)**

- **X-koordináta regiszter (VX): BASEADDR + 0x04, írható és olvasható**
  - Értéke grafikus módban 0 – 255, karakteres módban 0 – 31
  - X irányú írás/olvasás esetén (INCSEL=0) minden adatregiszter hozzáférésnél (karakteres módban minden második) értéke eggyel növekszik, átfordulás esetén az Y-koordináta értéke eggyel növekszik

- **Y-koordináta regiszter (VY): BASEADDR + 0x05, írható és olvasható**
  - Értéke grafikus módban 0 – 191, karakteres módban 0 – 23
  - Y irányú írás/olvasás esetén (INCSEL=1) minden adatregiszter hozzáférésnél (karakteres módban minden második) értéke eggyel növekszik, átfordulás esetén az X-koordináta értéke eggyel növekszik



MiniRISC mintarendszer

(Perifériák – ps2_keyboard)

ps2_keyboard: **PS/2 billentyűzet illesztését biztosító periféria**

- **A használatához szükség van a *LOGSYS VGA, PS/2 és hangszóró modul* csatlakoztatására az FPGA kártya „A” bővítőcsatlakozójára**
  - A PS/2 interfészről részletek a modul dokumentációjában
- **Vezérlő/státusz regiszter (KBC/KBS): BASEADDR + 0x00, írható/olvasható**

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
FIFONE	ASCII	IRQ	FIFOCLR	IE	LANGSEL	MODE	PEN
R	R	R	W	R/W	R/W	R/W	R/W

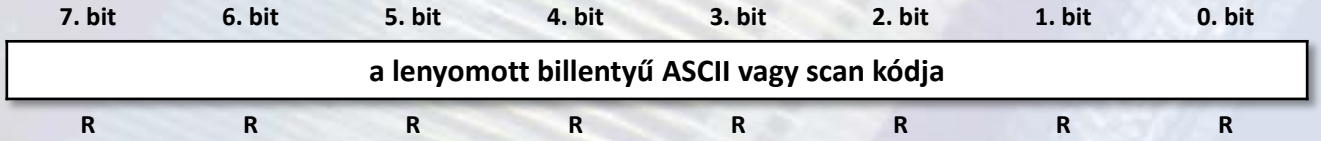
Bit	Mód	Funkció
PEN	R/W	0: a PS/2 periféria tiltott 1: a PS/2 periféria engedélyezett Ha a PS/2 periféria engedélyezett, akkor GPIO A és C tiltódik
MODE	R/W	0: ASCII mód 1: scan kód mód
LANGSEL	R/W	0: angol billentyűzetkiosztás 1: magyar billentyűzetkiosztás
IE	R/W	0: FIFONE megszakítás tiltott 1: FIFONE megszakítás engedélyezett
FIFOCLR	W	1 beírásának hatására a vételi FIFO törlődik
IRQ	R	a PS/2 periféria megszakításkérő kimenetének állapota
ASCII	R	0: az aktuális adat scan kód 1: az aktuális adat ASCII kód
FIFONE	R	0: a vételi FIFO üres 1: a vételi FIFO-ban van adat

MiniRISC mintarendszer

(Perifériák – ps2_keyboard)

***ps2_keyboard:* PS/2 billentyűzet illesztését biztosító periféria**

- **Adatregiszter (KBD): BASEADDR + 0x01, csak olvasható**
  - Ha FIFONE=1, akkor a lenyomott billentyű ASCII vagy scan kódja (scan kódkészlet 2) olvasható ki az adatregiszterből
  - ***ASCII módban (MODE=0)*** csak a nyomtatható karakterek esetén kerül ASCII kód a vételi FIFO-ba (ASCII=1), nem nyomtatható karakterek esetén scan kód kerül a vételi FIFO-ba (ASCII=0)
  - ***Scan kód módban (MODE=1)*** mindig scan kód kerül a vételi FIFO-ba (ASCII=0)
  - Kibővített billentyűk esetén a scan kód 7. bitjének értéke mindig 1 lesz (a 0xE0 prefix nem kerül be a vételi FIFO-ba)
    - Pontosabban KBD[7:4] ≥ 9 esetén kibővített a billentyű, mert a nem kibővített F7 billentyű scan kódja 0x83



MiniRISC mintarendszer

(Közvetlen memória hozzáférés)

- **A programozott adatátvitel nem hatékony, pl. memória másolás**

```
memcpy:
  mov  r0, (r8) ; Olvasás a forráscímről.
  mov  (r9), r0 ; Írás a célcímre.
  add  r8, #1   ; A mutatók növelése.
  add  r9, #1
  sub  r10, #1  ; Az adatméret csökkentése.
  jnz  memcpy   ; Ugrás, ha van még adat.
  rts           ; Visszatérés a hívóhoz.
```

➡ **6 utasítás, azaz *18 órajelciklus kell 1 bájt másolásához***

- **A hatékonyság növelhető, ha az adatátvitel külső vezérlőegység (hardver) irányításával történik → közvetlen memória hozzáférés (Direct Memory Access, DMA)**
  - Általános célú (központi, centrális) DMA vezérlő
  - Dedikált vezérlő logika a perifériákban (bus master képesség)
- **A MiniRISC mintarendszerben egy általános célú DMA vezérlő periféria található, amely a teljes adatmemória címtartományhoz képes hozzáférni**

MiniRISC mintarendszer

(Közvetlen memória hozzáférés)

A DMA működési folyamata a MiniRISC mintarendszer esetén:

1. **Az átviteli paraméterek beállítása és az átvitel indítása a programból (üzemmód, forráscím, célcím, adatméret)**
2. **A DMA vezérlő adatmemória hozzáférést kér. Ha megkapja, akkor beolvassa és tárolja a forráscímről az adatot.**
3. **A DMA vezérlő adatmemória hozzáférést kér. Ha megkapja, akkor beírja a célcímre az adatot. A mutatók növelése (ha engedélyezett) és a hátralévő adatméret csökkentése. Ugrás a 2. pontra, ha van még adat.**
4. **„DMA adatátvitel kész” megszakítás jelzés a CPU-nak**

Optimális esetben *1 bájt másolása 2 órajelciklust* igényel. A DMA átvitel alatt a CPU hasznos tevékenységet is tud végezni.

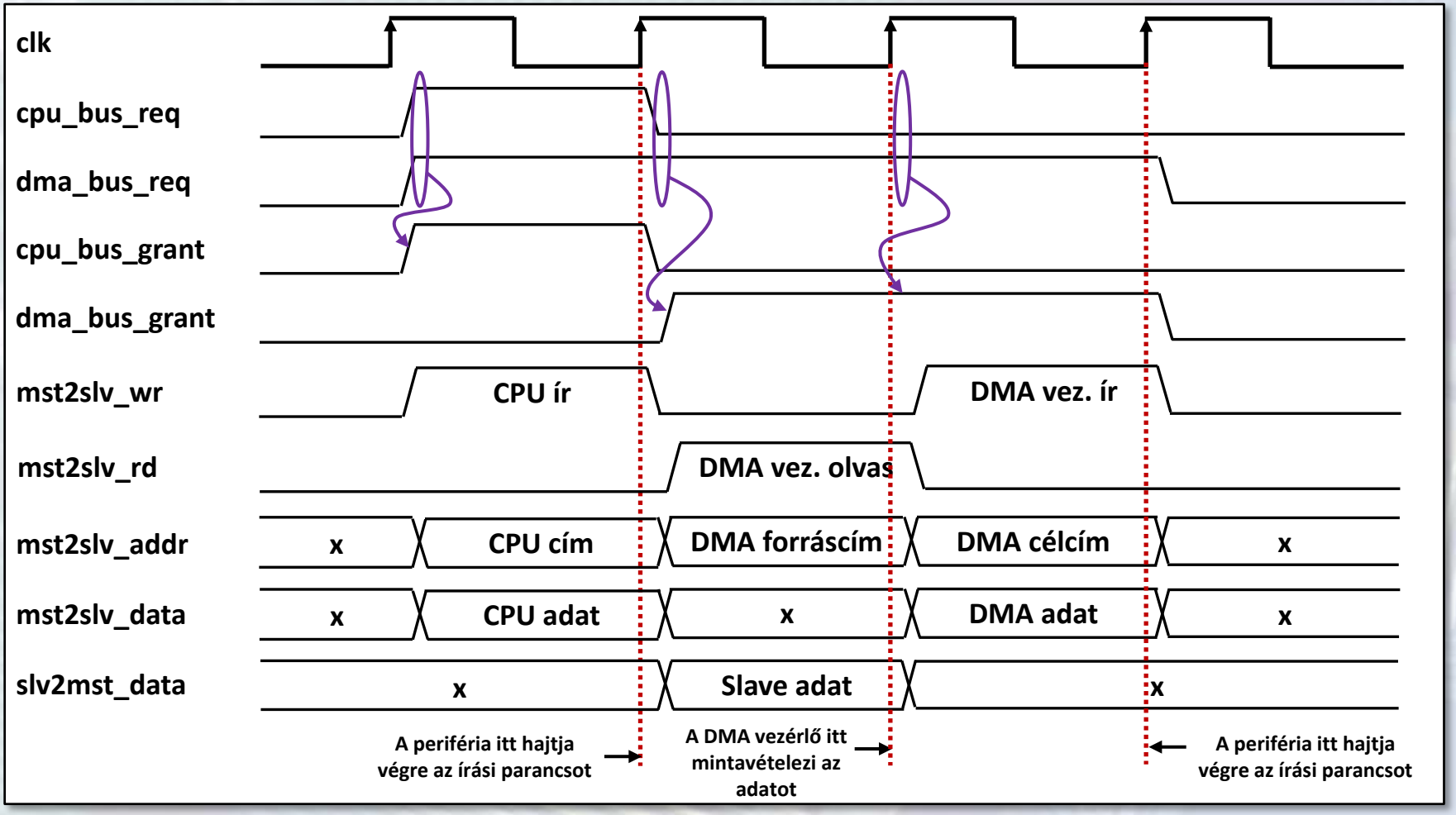
MiniRISC mintarendszer

(Közvetlen memória hozzáférés)

- **A buszra kapcsolódhatnak**
  - **Master egységek:** csak ezek kezdeményezhetnek átvitelt
  - **Slave egységek:** végrehajtják a master egység kérését
- **A MiniRISC mintarendszerben master: CPU, DMA vezérlő**
  - A CPU csak master interfésszel rendelkezik
  - A DMA vez.-nek van slave interfésze is (felprogramozáshoz)
- **Több master egység esetén előfordulhatnak egyidejű kérések**
  - A ***busz arbiter*** valamilyen algoritmus szerint (pl. fix vagy körforgásos prioritás, stb.) kiválaszt egy master egységet
    - A MiniRISC mintarendszerben fix prioritás van, a CPU-é a nagyobb
  - ***xxx_bus_req*** jel: busz hozzáférés kérése
  - ***xxx_bus_grant*** jel: busz hozzáférés megadása
  - Csak a kiválasztott master egység hajthatja meg a buszt inaktív nullától különböző értékkel (elosztott busz MUX)

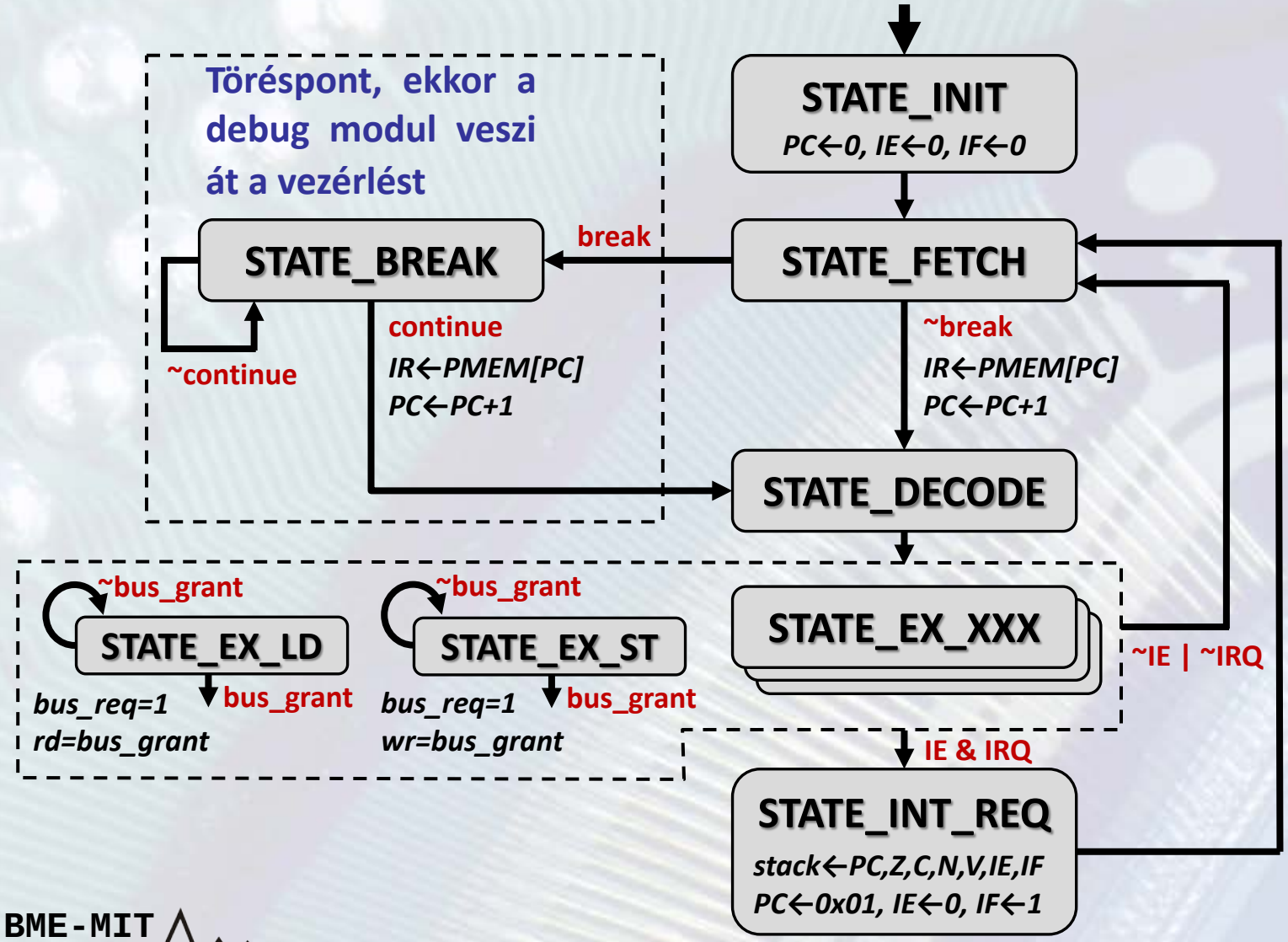
MiniRISC mintarendszer

(Közvetlen memória hozzáférés – Adatbusz arbitráció)



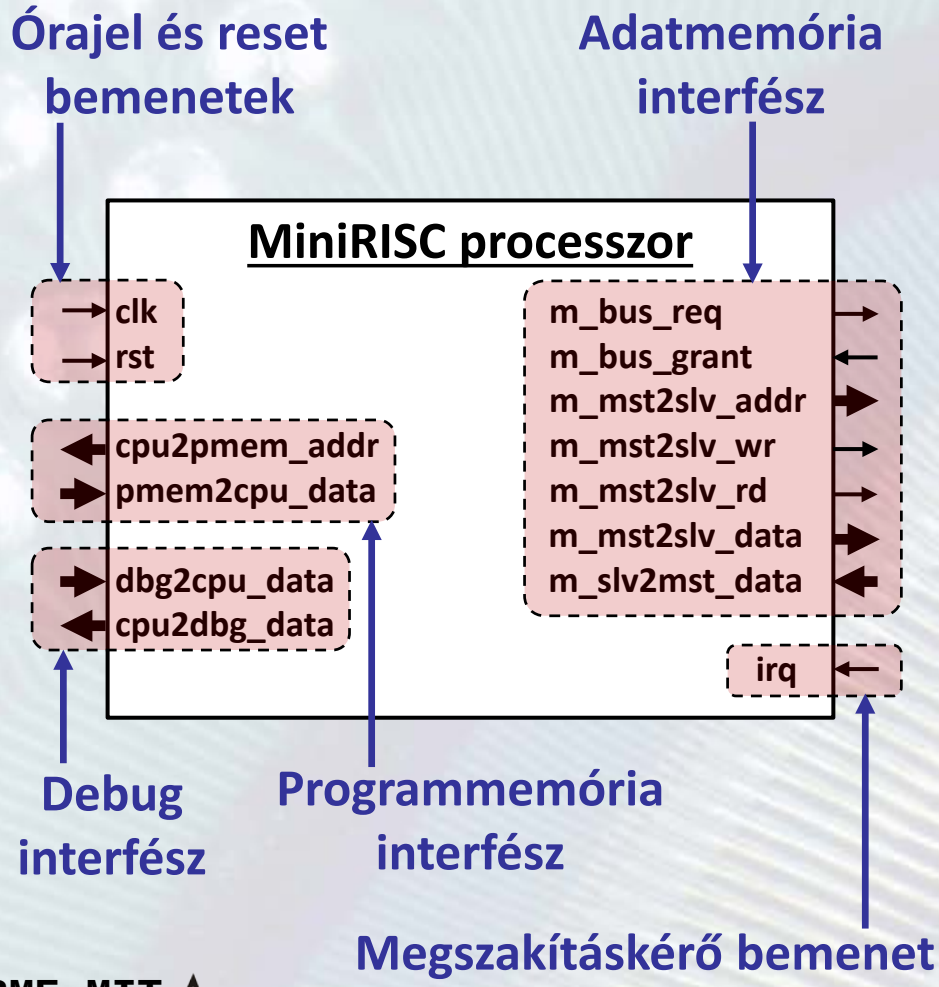
MiniRISC mintarendszer

(Közvetlen memória hozzáférés – Módosított CPU vezérlő FSM)



MiniRISC mintarendszer

(Közvetlen memória hozzáférés – Módosított CPU interfész)



A Verilog modul fejléce:

```
module minirisc_cpu(
   //Órajel és reset.
   input  wire         clk,
   input  wire         rst,

   //A programmemóriával kapcsolatos jelek.
   output wire [7:0]   cpu2pmem_addr,
   input  wire [15:0]  pmem2cpu_data,

   //Az adatmemóriával kapcsolatos jelek.
   output wire         m_bus_req,
   input  wire         m_bus_grant,
   output wire [7:0]   m_mst2slv_addr,
   output wire         m_mst2slv_wr,
   output wire         m_mst2slv_rd,
   output wire [7:0]   m_mst2slv_data,
   input  wire [7:0]   m_slv2mst_data,

   //Megszakításkérő bemenet (aktív magas).
   input  wire         irq,

   //Debug interfész.
   input  wire [22:0]  dbg2cpu_data,
   output wire [47:0]  cpu2dbg_data
);
```

MiniRISC mintarendszer

(Perifériák – dma_controller)

dma_controller: **centrális DMA vezérlő periféria**

- **Lehetővé teszi az adatmemórián belüli hatékony adatátvitelt**
  - 1 bájt másolása 2 órajelcilkus alatt: memória olvasás, majd írás
- **Megszakításkérés az adatátvitel befejeződésekor**
- **Vezérlő/státusz regiszter (DC/DS): BASEADDR + 0x00, írható/olvasható**

7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
BUSY	IRQ	IF	0	0	IEN	DINC	SINC
R	R	R/W1C	R	R	R/W	R/W	R/W

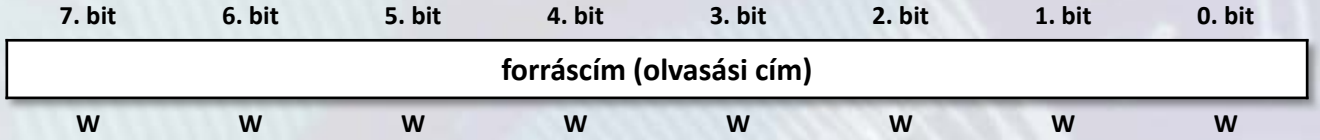
Bit	Mód	Funkció
SINC	R/W	0: forráscím növelés tiltott 1: forráscím növelés engedélyezett
DINC	R/W	0: célcím növelés tiltott 1: célcím növelés engedélyezett
IEN	R/W	0: „adatátvitel befejeződött” megszakítás tiltott 1: „adatátvitel befejeződött” megszakítás engedélyezett
IF	R/W1C	az „adatátvitel befejeződött” megszakítás flag (a jelzés 1 beírásával törölhető)
IRQ	R	a DMA vezérlő megszakításkérő kimenetének állapota
BUSY	R	0: nincs adatátvitel 1: adatátvitel van folyamatban

MiniRISC mintarendszer

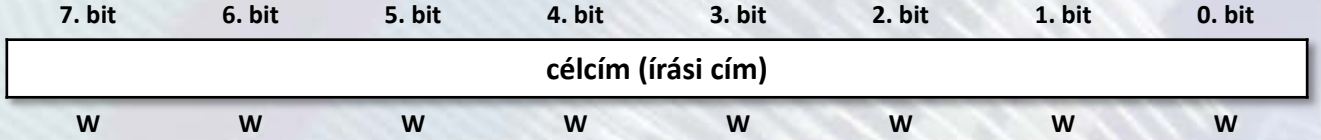
(Perifériák – dma_controller)

dma_controller: **centrális DMA vezérlő periféria**

- **Forráscím regiszter (DSA): BASEADDR + 0x01, csak írható**
  - Értéke minden bájt átvitele után növekszik, ha SINC=1



- **Célcím regiszter (DDA): BASEADDR + 0x02, csak írható**
  - Értéke minden bájt átvitele után növekszik, ha DINC=1



- **Adatméret regiszter (DTS): BASEADDR + 0x03, csak írható**
  - A DMA adatátvitel mérete bájtokban
  - Írása indítja el a DMA átvitelt

Példaprogramok

2. példa: 8 bites jobbra léptető Johnson számláló

- **Johnson számláló: shiftregiszter, melynek LSb-je vagy MSb-je inverterrel vissza van csatolva, a kezdőállapota 0x00**
- **A számláló állapotát a LED-eken jelenítsük meg**
- **A jobbra léptetéshez használhatjuk az SR0 utasítást**
  - MSb-be 0, a C flag-ba a kiléptetett bit (LSb) kerül
  - Ha a C flag értéke 0, akkor az MSb-be egyet kell írni
- **Az állapotváltás történjen kb. fél másodpercenként (2 Hz)**
  - Az időzítés legyen szoftveres, az időzítő szubrutinnak a meghívása után kb. 0,5 s múlva kell visszatérnie
  - 16 MHz-es rendszerórajel, utasításonként 3 órajelciklus
    - 0,5 s alatt $\frac{16 \cdot 10^6 Hz}{3 \cdot 2 Hz} = 2666667$ utasítás hajtódik végre
- **Nem használunk megszakítást, ezért a program kezdődhet a 0x00 címen (reset vektor)**

Példaprogramok

2. példa: 8 bites jobbra léptető Johnson számláló

```
DEF LD  0x80                ; LED regiszter

    CODE

;*****************************************************
;* A program kezdete. A 0x00 cím a reset vektor.     *
;*****************************************************
start:
    mov     r0, #0          ; Az SHR kezdőállapotának beállítása.
shr_loop:
    mov     LD, r0          ; Az aktuális állapot megjelenítése.
    jsr     delay           ; Az időzítő szubrutin meghívása.
    sr0     r0              ; A következő állapot beállítása.
    jc      shr_loop        ; A C flag tesztelése. Ugrás, ha C=1.
    or      r0, #0x80       ; Az MSb-be 1 kerül, ha C=0.
    jmp     shr_loop        ; Ugrás a ciklus elejére.
```

Példaprogramok

2. példa: 8 bites jobbra léptető Johnson számláló

```
;*********************************************************
;* Az időzítő szubrutin. 24 bites számlálót használunk és  *
;* a C flag értékét teszteljük, amely a -1 elérésekor lesz *
;* először 1. Így iterációnként 4 utasítás szükséges csak. *
;* A számláló kezdőállapota ezért 2666667/4 - 1 = 666665.  *
;*********************************************************
delay:
    mov     r4, #0x0A   ; A kezdőállapot beállítása, r4 az
    mov     r3, #0x2C   ; MSB, r2 az LSB.
    mov     r2, #0x29   ; 666665 -> 0x0A2C29
delay_loop:
    sub     r2, #1      ; Iterációnként a számláló értékét
    sbc     r3, #0      ; eggyel csökkentjük.
    sbc     r4, #0
    jnc     delay_loop  ; Ugrás a ciklus elejére, ha C=0.
    rts                 ; Visszatérés a hívóhoz.
```

Példaprogramok

2. példa: 8 bites jobbra léptető Johnson számláló (LST fájl)

```
S Addr  Instr   Source code
-------------------------------------------------------------------------------
                DEF LD 0x80                     ; LED regiszter

                    CODE

C 00            start:                          ; A program kezdete. A 0x00 cím a reset vektor.
C 00    C000        mov     r0, #0x00           ; Az SHR kezdőállapotának beállítása.
C 01            shr_loop:
C 01    9080        mov     LD[80], r0          ; Az aktuális állapot megjelenítése.
C 02    B907        jsr     delay[07]           ; Az időzítő szubrutin meghívása.
C 03    F071        sr0     r0                  ; A következő állapot beállítása.
C 04    B301        jc      shr_loop[01]        ; A C flag tesztelése. Ugrás, ha C=1.
C 05    5080        or      r0, #0x80           ; Az MSb-be 1 kerül, ha C=0.
C 06    B001        jmp     shr_loop[01]        ; Ugrás a ciklus elejére.

C 07            delay:                          ; Az időzítő szubrutin kezdete.
C 07    C40A        mov     r4, #0x0A           ; A számláló kezdőállapotának beállítása,
C 08    C32C        mov     r3, #0x2C           ; r4 az MSB, r2 az LSB.
C 09    C228        mov     r2, #0x29           ; 666665 -> 0x0A2C29
C 0A            delay_loop:
C 0A    2201        sub     r2, #0x01           ; Iterációnként a számláló értékét
C 0B    3300        sbc     r3, #0x00           ; eggyel csökkentjük.
C 0C    3400        sbc     r4, #0x00
C 0D    B40A        jnc     delay_loop[0A]      ; Ugrás a ciklus elejére, ha C=0.
C 0E    BA00        rts                         ; Visszatérés a hívóhoz.
```

Példaprogramok

3. példa: USRT kommunikáció

- **1 másodpercenként küldjük el a ”MiniRISC CPU” stringet az USRT interfészen keresztül**
  - A stringet az adatmemóriában tárolhatjuk
    - A karakterek eléréséhez indirekt címzést használunk
    - A string végét a nulla karakter jelzi
  - Az időzítést a ***basic_timer*** periféria segítségével végezzük lekérdezéses módon
    - Ismétlődéses mód és 65536-os előosztás szükséges
    - A számláló kezdőállapota: $TR = \frac{16 \cdot 10^6 Hz}{65536 \cdot 1Hz} - 1 = 243$
- **A vett karaktereket jelenítsük meg a LED-eken**
  - A vétel kezelése történjen megszakításos módon

Példaprogramok

3. példa: USRT kommunikáció

```
DEF LD   0x80              ; LED regiszter
DEF TM   0x8c              ; Időzítő számláló regiszter
DEF TC   0x8d              ; Időzítő parancs regiszter
DEF TS   0x8d              ; Időzítő státusz regiszter
DEF UC   0x8e              ; USRT kontroll regiszter
DEF US   0x8e              ; USRT státusz regiszter
DEF UD   0x8f              ; USRT adatregiszter

    CODE

;*************************************************************
;* A program kezdete.                                        *
;*************************************************************
    jmp     start          ; 0x00 a reset vektor.
    jmp     rx_irq         ; 0x01 a megszakítás vektor.

start:
    mov     r0, #0         ; A LED-ek törlése.
    mov     LD, r0
```

Példaprogramok

3. példa: USRT kommunikáció

```
    mov     r0, #0x3b   ; Az USRT periféria inicializálása: FIFO-k törlése,
    mov     UC, r0      ; RX megszakítás, RX és TX engedélyezés (0011_1011).
    mov     r0, #243    ; Az időzítő inicializálása. A kezdőállapot
    mov     TM, r0      ; beírása a TM időzítő regiszterbe.
    mov     r0, #0x73   ; Az időzítő konfigurálása: 65536-os előosztás,
    mov     TC, r0      ; ismétléses mód, engedélyezés (0111_0011).
    mov     r0, TS      ; Az időzítő esetleges TOUT jelzésének törlése.
    sti                 ; A megszakítások engedélyezése.

send_str:
    jsr     tmr_wait    ; Az időzítő szubrutin meghívása (1s várakozás).
    mov     r1, #str    ; A mutatót a string első karakterére állítjuk.
send_ch:
    mov     r0, US      ; Az USRT adási FIFO állapotának ellenőrzése.
    and     r0, #0x40   ; Várakozás, amíg az adási FIFO tele van (TXNF=0).
    jz      send_ch
    mov     r0, (r1)    ; A string következő karakterének beolvasása.
    cmp     r0, #0      ; A nulla végkarakter ellenőrzése.
    jz      send_str    ; Ugrás, ha a string végére értünk.
    mov     UD, r0      ; A karakter beírása az USRT adási FIFO-ba.
    add     r1, #1      ; A mutatót a következő karakterre állítjuk.
    jmp     send_ch     ; Elküldjük a következő karaktert.
```

Példaprogramok

3. példa: USRT kommunikáció

```
;*****************************************************
;* Várakozás az időzítő TOUT jelzésére.              *
;*****************************************************
tmr_wait:
    mov     r8, TS      ; Beolvassuk az időzítő státusz
    tst     r8, #0x04   ; regisztert a TOUT bit vizsgálatához.
    jz      tmr_wait    ; Várakozás, ha a TOUT bit még mindig 0.
    rts                 ; Visszatérés a hívóhoz.

;*****************************************************
;* Az USRT vételi megszakítás kezelése.              *
;*****************************************************
rx_irq:
    mov     r15, UD     ; A vett karakter beolvasása és
    mov     LD, r15     ; kiírása a LED-ekre.
    rti                 ; Visszatérés a megszakításból.
```

Példaprogramok

3. példa: USRT kommunikáció

```
    DATA

;*************************************************************
;* Az adatmemória inicializálása.                            *
;*************************************************************
;Az elküldendő string.
str:
    DB      "MiniRISC CPU\r\n", 0x00
```

A stringgel inicializált adatmemória tartalma:

Instruction set | Peripheral address map | Memory | USRT terminal (0x8E) | Display (0x90)

```
  |00 01 02 03 04 05 06 07|08 09 0A 0B 0C 0D 0E 0F|
--+-----------------------+-----------------------+
00|4D 69 6E 69 52 49 53 43|20 43 50 55 0D 0A 00 00|MiniRISC CPU....
10|00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00|................
20|00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00|................
30|00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00|................
40|00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00|................
50|00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00|................
60|00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00|................
70|00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00|................
--+-----------------------+-----------------------+
  |00 01 02 03 04 05 06 07|08 09 0A 0B 0C 0D 0E 0F|
```