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

Digitális technika VIMIAA02 11. hét

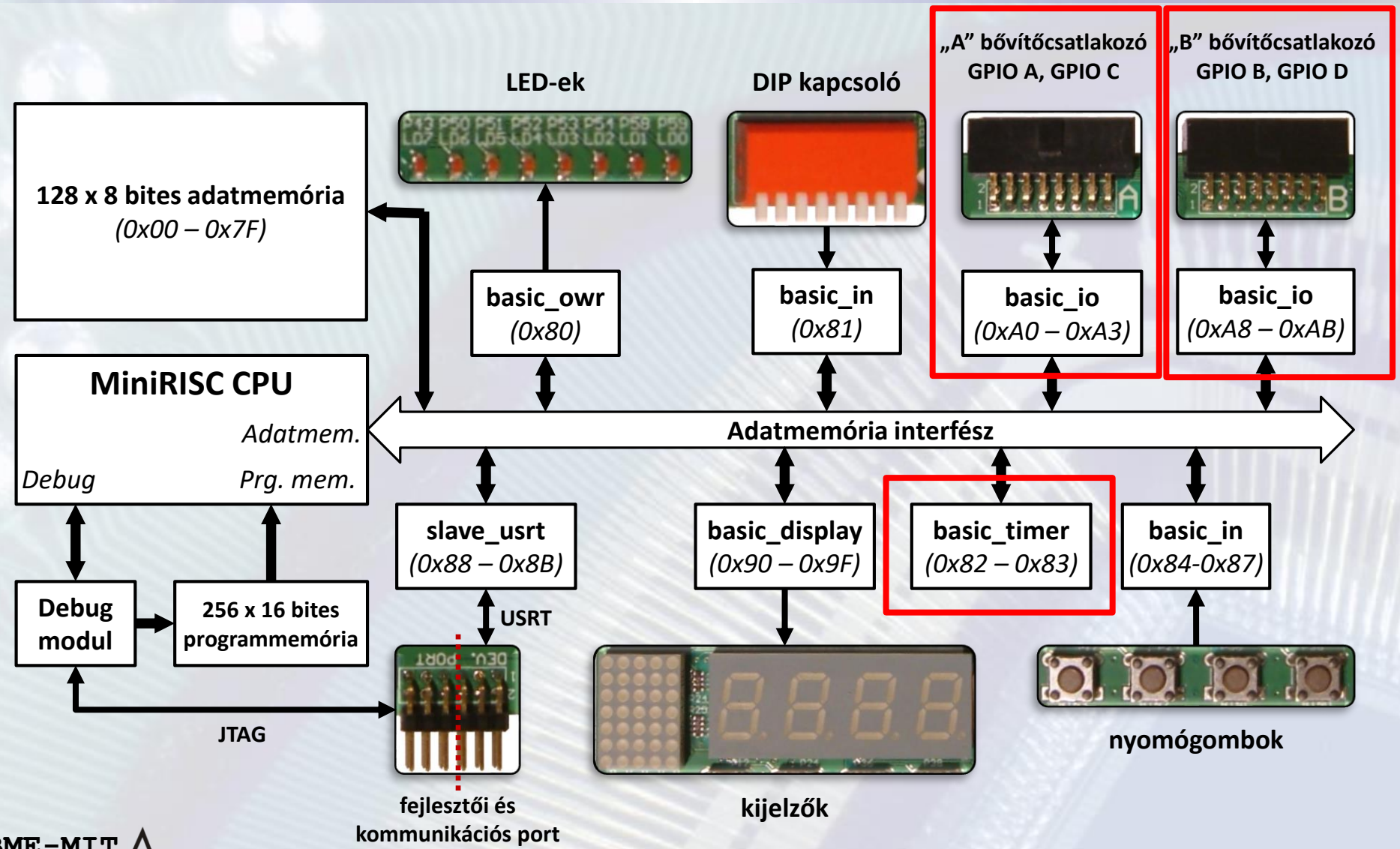
Fehér Béla, Benesóczky Zoltán
BME MIT

Mikroprocesszoros rendszer

Az előadás témái:

- A mikrokontrolerek GPIO és TIMER perifériái (MiniRISC példával)
- Interrupt I. rész

MiniRISC mintarendszer Blokkvázlat

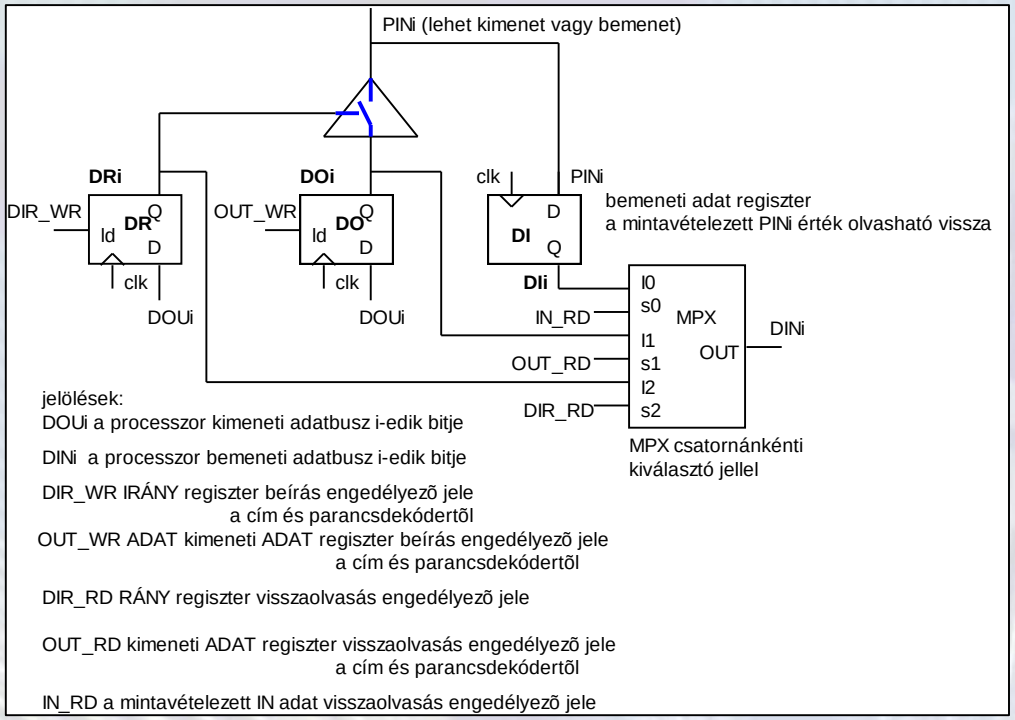


GPIO Periféria

- **A GPIO általános célú be-/kimeneti periféria.**
- **Programozhatóan lehet bemenet** (ez a default módja) és **lehet kimenet.**
- **Bemenetként** használva **a rákapcsolt logikai érték olvasható ki belőle**.
- **Kimenetként** használva **tetszőleges logikai érték adható ki a lábaira** (pin-ek).
- **A mikrokontrollerek (µC-k) szinte mindig rendelkeznek tetszőlegesen használható GPIO interfészekkel.**
  - Általában a mikrokontroller szinte minden lába használható GPIO interfészként.
  - Azonban a legtöbb lábhoz valamely belső periféria jele is hozzá van rendelve.
  - A hardver programozásával választható, hogy melyik célra kívánjuk felhasználni.

GPIO Periféria

- **A GPIO periféria a PIN-ekkel (a µC lábaival) csatlakozik a külvilághoz. Minden PIN-hez 3 regiszter egy-egy bitje tartozik. (**Alább 1 bit felépítését mutatjuk, az összes ugyanilyen.)
- **Adat kimeneti regiszter: DO**. Ide kell írni a PIN-en megjelenítendő adatot.
- **Adat bemeneti regiszter**: **DI**. Itt lehet beolvasni a PIN-en levő adatot.
- **Irányregiszter: DR** A három állpotú meghajtó engedélyezését vezérli.

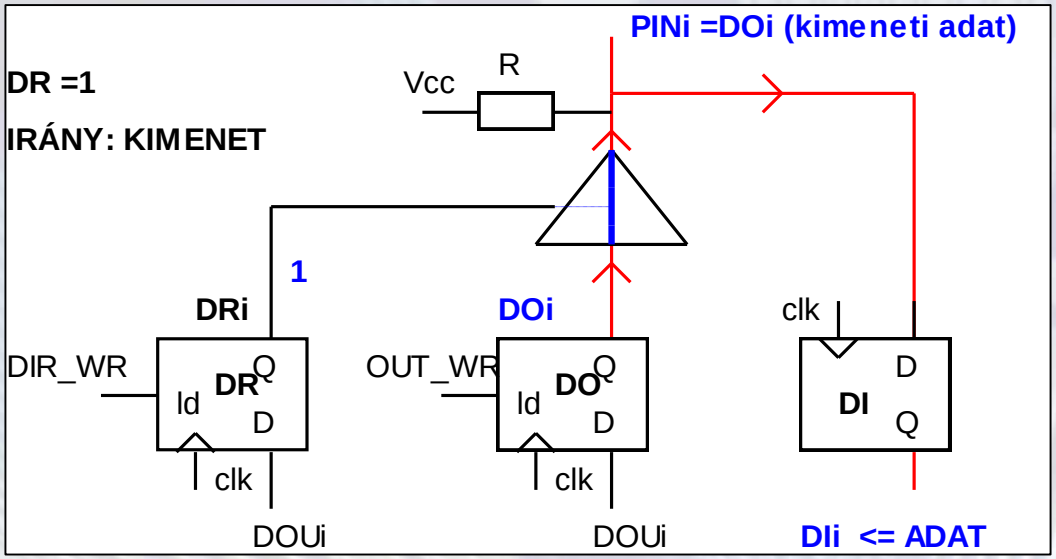


K***imenetként konfigurálás***: **DR**-be 1-et kell írni, ekkor DO a PIN-re kapcsolódik.

Bementként konfigurálás: **DR**-be 0-t kell írni ha a PIN-re kívülről kapcsolódik adat és azt akarjuk beolvasni. Ilyenkor az adatregiszter (DO) kimenete nem hajtja meg a PIN-t, mert a 3 állapotú meghajtó „nagy impededanciás” vagyis kikapcsolt állapotba kerül.

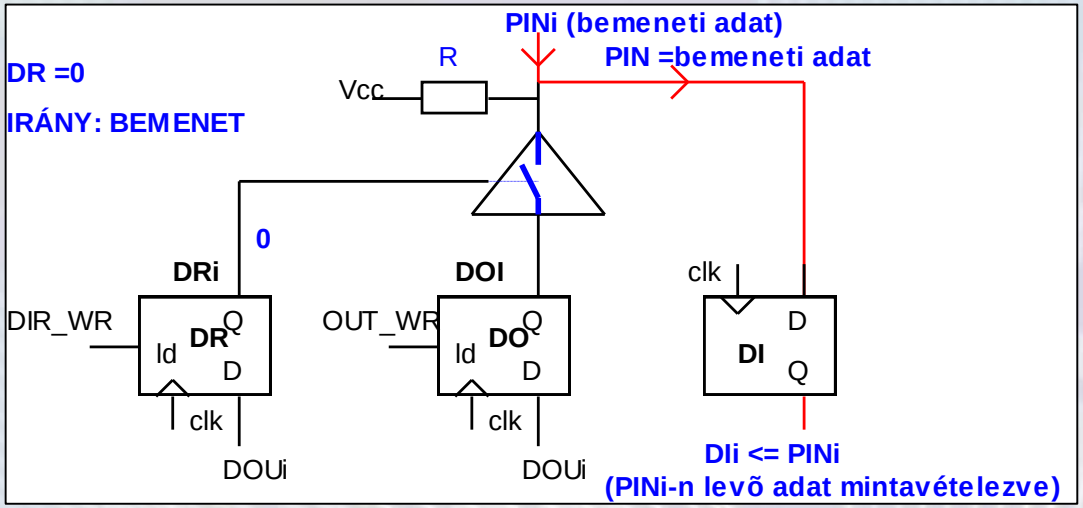
GPIO Periféria

- **A GPIO működés kimeneti módba állítva:**
- A DRi-be 1-et írva a 3 állapotú meg-hajtó a PINi-re kap-csolja a kimeneti adatregisztert (DOi-t), tehát a PIN-en ennek logikai értéke jelenik meg és DIi-ből (bemeneti adatregiszter) ez olvasható vissza.



GPIO Periféria

- **A GPIO működés bemeneti módba állítva:**
- A Dri-be 0-át írva a 3 állapotú meghajtó kikapcsolt állapotba kerül. Ilyenkor szabad a PINi-t kívülről meghajtani valamilyen logikai értékkel.



Ekkor a DIi-ből (bemeneti adatregiszter) ez a kívülről jövő logikai érték olvasható vissza. A MiniRISC GPIO PIN-jeire egy ellenállással a tápra vannak húzva. Ezért bemeneti módban, ha semmi nem hajtja meg, a bemeneti adatbit 1 értékű.

GPIO Periféria

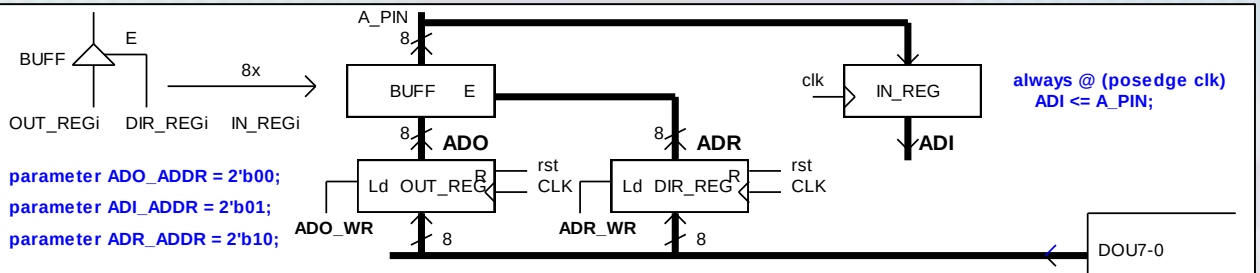
- **A Mini RISC GPIO A illesztés blookvázlata és Verilog kódja:**
  (A Verilog kódban a változó definíciók nem szerepelnek.)

GPIO A felépítése

parameter BASE_ADDR = 8'HA0;

A7	A6	A5	A4	A3	A2	A1	A0		
1	0	1	0	0	0	0	0	R/W	**ADO**
1	0	1	0	0	0	0	1	R	**ADI**
1	0	1	0	0	0	1	0	R/W	**ADR**

parameter ADO_ADDR = 2'b00;
parameter ADI_ADDR = 2'b01;
parameter ADR_ADDR = 2'b10;



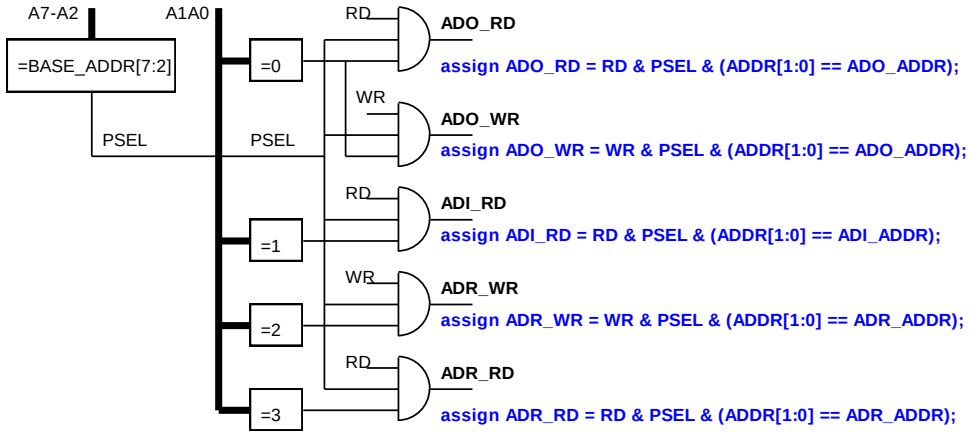

```
always @ (posedge clk)
  ADI <= A_PIN;
```

```
always @ (posedge clk)
if(rst) ADO <= 0x00;
else
  if(ADO_WR) ADO <= DOUT;
```

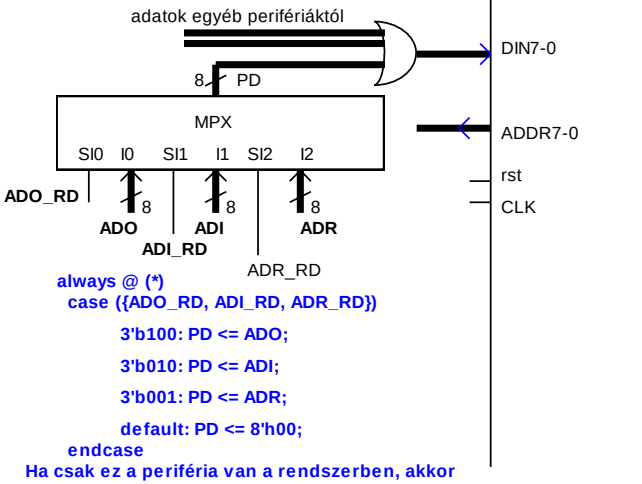
```
always @ (posedge clk)
if(rst) ADR <= 0x00;
else
  if(ADR_WR) ADR <= DOUT;
```

PROCESSZOR

assign PSEL = (ADDR >>2) == (BASE_ADDR >> 2) ;
másképp is lehet: assign PSEL = ADDR[7:2] == BASE_ADDR[7:2];




```
assign ADO_RD = RD & PSEL & (ADDR[1:0] == ADO_ADDR);
assign ADO_WR = WR & PSEL & (ADDR[1:0] == ADO_ADDR);
assign ADI_RD = RD & PSEL & (ADDR[1:0] == ADI_ADDR);
assign ADR_WR = WR & PSEL & (ADDR[1:0] == ADR_ADDR);
assign ADR_RD = RD & PSEL & (ADDR[1:0] == ADR_ADDR);
```




```
always @ (*)
  case ({ADO_RD, ADI_RD, ADR_RD})
      3'b100: PD <= ADO;
      3'b010: PD <= ADI;
      3'b001: PD <= ADR;
      default: PD <= 8'h00;
  endcase
```

Ha csak ez a periféria van a rendszerben, akkor nem kellenek DIN előtt az OR kapus adat bemenetek a teljes OR hálózat elhagyható. Ekkor a MUX a PD bemenet helyett közvetlenül a CPU DIN bemenetére csatlakozik. Tehát a PD <= .. -ot DIN <= ----re kell cserélni. Vizsga ZH-n is ez lesz.

GPIO Periféria

- **A GPIO periféria használata:**
  - Párhuzamos módban, együtt vezérelve a 8 bitet
  - Függetlenül, bitenként megadva és használva az egyes jeleket bemenetként/kimenetként
- **Értékadás GPIO periféria kimenetének**
  - A MiniRISC Load/Store működési elve miatt 2 utasításonként adható ki új érték/változtatható a GPIO láb értéke (2*187,5ns = 375ns)

```
mov r0, #konst
mov ADO, r0
```

GPIO Periféria

- **A GPIO lábakon tetszés szerinti protokoll/időzítési szekvencia lejátszható (maximum a fenti sebességgel)**
  - Ezt hívjuk „**bit-banging**” módnak, azaz a processzor a programból, a megfelelő időzítéssel, dinamikusan változtatja a lábakon megjelenő kimeneti értékeket, illetve mintavételezi és beolvassa a bemeneti értékeket.
  - Az időzítés nem mindig pontos, kisebb ingadozás, „jitter” felléphet

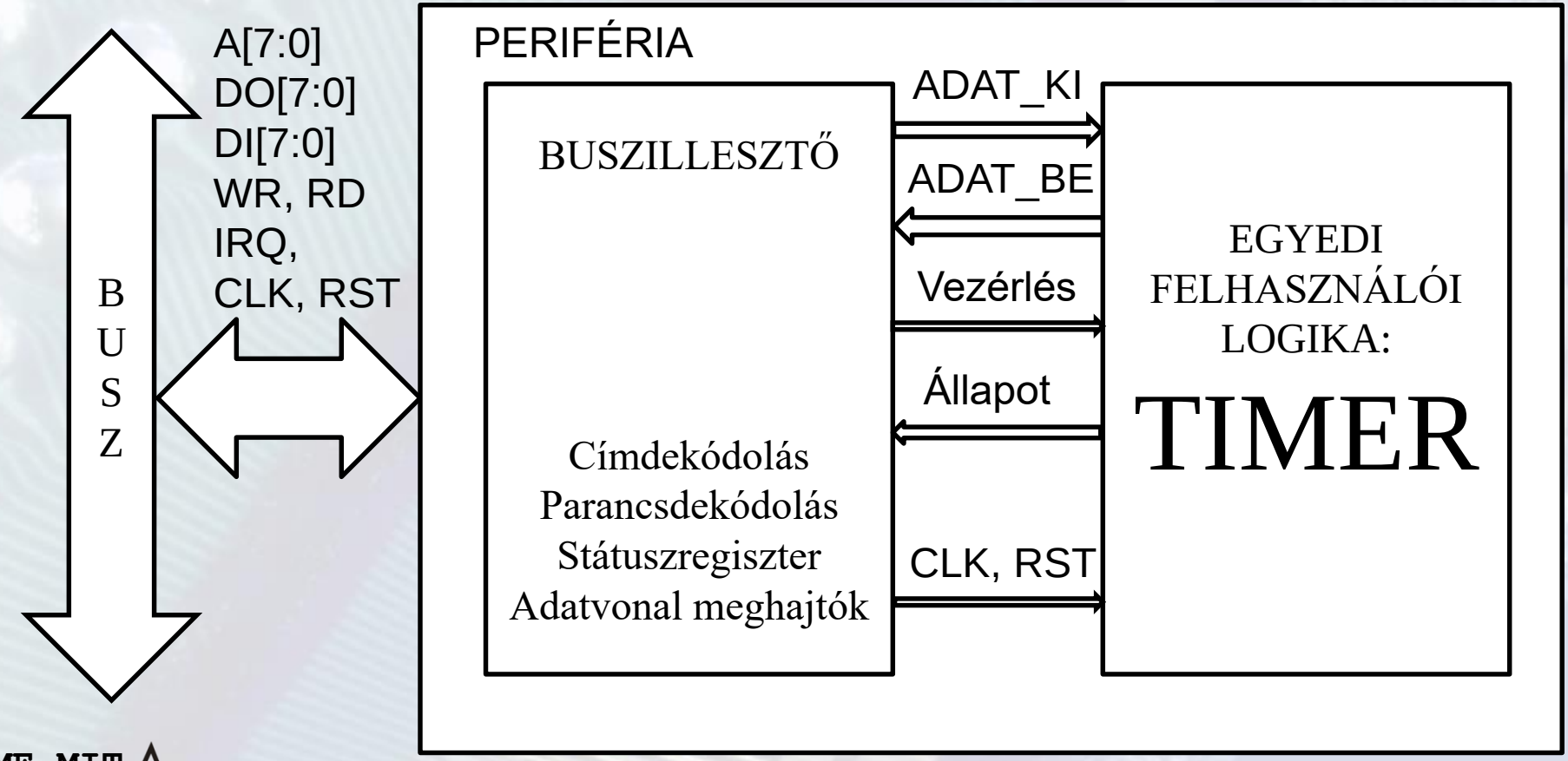
GPIO Periféria

- **Mintapélda:** Programozzuk fel a GPIO A-t, hogy a ***felső 7 bitje kimenet***, ***a legkisebb bit bemenet*** legyen. Ha a legkisebb biten 1 van, akkor folyamatosan kapcsolgassuk 1-be majd 0-ba felső 7 bitjet, egyébként a kimenet legye 0.

```
        mov r0, #0
        mov ADO, r0     ;adatregiszter kezdetben 0
        mov r0, #0b11111110
        mov ADR, r0     ;felső 7 bit kimenet, ADR[0] bemenet
loop:   mov r0, ADI                 ;ADI[7:0] beolvasása
        tst   r0, #0b00000001       ;ADI[0] bemenet tesztelése
        jz loop                     ;vissza, amíg 0
        mov r0, #0b11111110
        mov ADO, r0                 ;ADO[7:1] = 1111111
        mov r0, #0b00000000         ;(az ADO[0] nem jut ki, mert bemenet)
        mov ADO, r0                 ;ADO[7:1] = 0000000
        jmp loop                    ;vissza a ADI[0] figyelésére
```

Timer periféria

Időzítő periféria (TIMER) a processzoros időzítések támogatására



Timer periféria

- **Probléma:** A processzorral programozottan tudunk időzítést beállítani (tervezett hosszúságú ciklussal), de ez alatt a processzor nem tud hasznos munkát végezni. Pl.

```
DEF N 100
Delay:mov r0, #N
loop:   sub r0, #1
        jnz loop
        ...
```

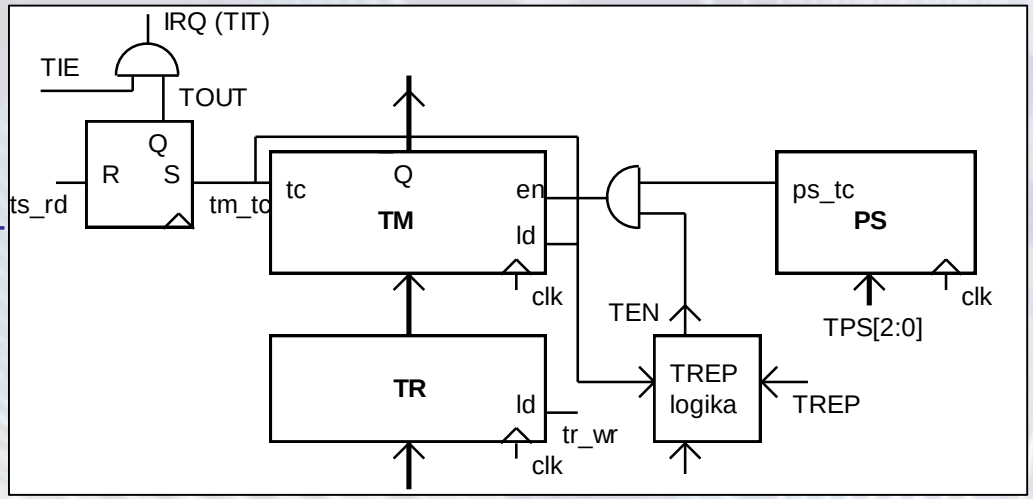
A loop ciklus hossza: **N*Un*Tu**
Un: ciklus törzs utasításainak száma (itt 2)
Tu: 1 MiniRSIC utasítás végrehajtási ideje
(3*1/(16*10E6)) sec = 187 ns

Timer periféria

- **Probléma:** A processzorral programozottan tudunk időzítést beállítani (tervezett hosszúságú ciklussal), de ez alatt a processzor nem tud hasznos munkát végezni.
- A idő múlását számolgató utasításciklusok alatt végezhetne hasznos feladatokat is a processzor.
- Erre használható a **TIMER** (időzítő) periféria.
- Az időzítő perifériák a leggyakoribb perifériák a processzoros rendszerekben.
- Az időzítő perifériák legfontosabb funkcionális eleme a **számláló**.

Timer periféria

- A MinRISC processzor TIMER perifériájában egy ***programozható előosztó*** (**PS**) kimenete léptet (engedélyez) egy ***8 bites tölthető, változtatható modulusú lefele számlálót*** (**TM**). Felépítése az alábbi egyszerűsített blokvázlaton látható.
- Amikor a TM lefele számláló eléri a 0 értéket (tm_tc) , bebillenti a **TOUT** jelző bitet és betölti a **TR *kezdőérték-regiszter*** értékét a TM számláló-ba. Ha a **TC *parancsregiszter* TIE** bitje engedélyezi, az IRQ kimeneten interruptot kér és **TIT** jelzést ad. A TOU és TIT jelzések kiolvashatók a ***TS státuszregiszterből***. (TR-be felprogramozáskor írjuk be a kezdőértéket.)



- A **PS** programozható előosztó a TM számlálót a TC parancsregiszter **TPS[2:0]** bitjeivel beállítható számú órajel leteltekor lépteti. A beállítható értékek:

TPS	000	001	010	011	100	101	110	111
PS	1	16	64	256	1024	4096	16384	65536

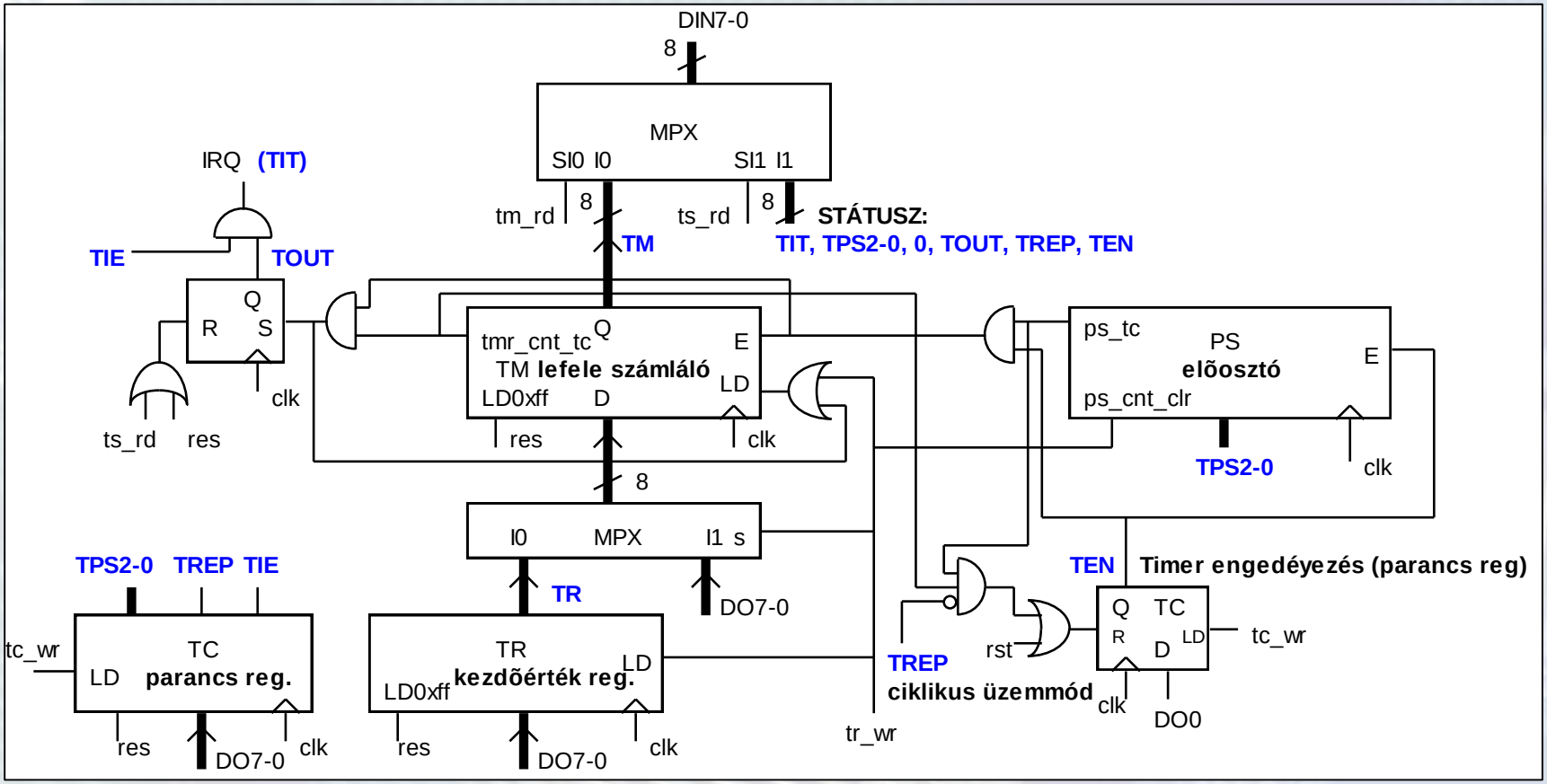
Timer periféria

- **A TIMER periféria üzemmódjai:**
  - **Egyszeres lefutási mód:** az időzítés végén jelez és leáll.
    Beállítása: **TC** parancsregiszterben TREP = 0.
  - **Periodikus mód:** az időzítés végén jelez és automatikus újraindul (a TR-be írt kezdőértékről).
    Beállítása: TC parancsregiszterben TREP = 1.
  - A számlálóregiszter TM aktuális értéke időzítés közben bármikor visszaolvasható.
  - A beállítható teljes időzítési tartomány

$$T = (TR+1)*PS*T_{clk} \quad (62{,}5ns – 1{,}048s)$$

Timer periféria

- **A TIMER részletesebb blokkvázlata:**



A cím és parancs dekódert nem rajzoltuk fel.

Timer periféria

A TIMER egység programozói felülete

- **TR (0x82): kezdőérték-regiszter (csak írható)**
  - A TR-be írt érték az időzítés kezdetekor beíródik a TM számlálóba ami innen számol lefele 0-ig. A TM lefele számláló modulusa: TR + 1 (TR, TR-1, ...1, 0)
  - ***A TR-be íráskor ugyanaz az érték TM-be is beíródik.***
- **TM (0x82): számlálóregiszter**
  - A lefele számláló aktuális értéke innen olvasható ki.
- **TC (0x83) Parancsregiszter (írható), TS(0x83) státuszregiszter (olvasható):**

	D7	D6	D5	D4	D3	D2	D1	D0
parancs regiszter TC:	TIE	TPS2	TPS1	TPS0	-	-	TREP	TEN
státus regszter TS:	**TIT**	TPS2	TPS1	TPS0	0	**TOUT**	TREP	TEN

- **TIE (**interrupt engedéylezés), **TPS[2:0] (**előoszás megadása), **TREP** (0: egyszeri jelzés 1: ciklikus jelzés)**, TEN: (**működés engedélyezés**)**

TPS2-0:	000	001	010	011	100	101	110	111
előosztás:	1	16	64	256	1024	4096	16384	65536
Periódusidő (frekvencia) ha fclk: 16MHz	1/16 us (16MHz)	1 us (1MHz)	4 us (250kHz)	16 us (62500Hz)	64 us (15625Hz)	256 us (3906.25Hz)	1024 us (976.5625Hz)	4096 us (244.141 Hz)

- A státuszregiszterből részben a parancsregiszter bitjei olvashatók vissza, új bit a **TOUT**, ami az időzítés lejártának állapotát jelzi és a **TIT** amely az interrupt kérést jelzi. **A státuszregiszter olvasása törli a TOUT és TIT jelzést.**

Timer periféria

A Timer egység programozói felületete:

- 4 interfész regiszter, ebből kettő-kettő azonos címen
- A Báziscím 0x82, a regisztercímek 0x82 és 0x83.

Időzítő	TR	kezdőállapot regiszter	0x82	WR	0xFF	Az időzítő számláló kezdeti értéke					
	TM	számláló regiszter	0x82	RD	0xFF	Az időzítő számláló aktuális értéke					
	TC	parancs regiszter	0x83	WR	0x00	TIE	TPS[2:0] - előosztás	-	-	TREP	TEN
	TS	státusz regiszter	0x83	RD	0x00	TIT	TPS[2:0] - előosztás	0	TOUT	TREP	TEN

Timer periféria

- **A TIMER periféria használata**

Példa: 0.5sec-os időzítés felprogramozása

Üzemmód: nincs megszakítás, előosztás 65536, ismétléses üzemmód, működés engedélyezés

TC: TIE, TPS[2:0],0,0,TREP, TEN

0 111 0 0 1 1

1/2= 1/16e6 *65536*(N+1) -> N = 16e6/(2*65536)-1 = **121,** ezt kell **TR**-be írni!

A TOUT státuszbit tesztelése:

TS: TIT, TPS[2:0],0,**TOUT**,TREP, TEN

0 000 0 **1** 0 0

Kimaszkoljuk (bitenkénti ÉS kapcsolatba hozzuk) kiolvasott **TOUT** értékét a 0b00000100 = 0x04 maszkkal

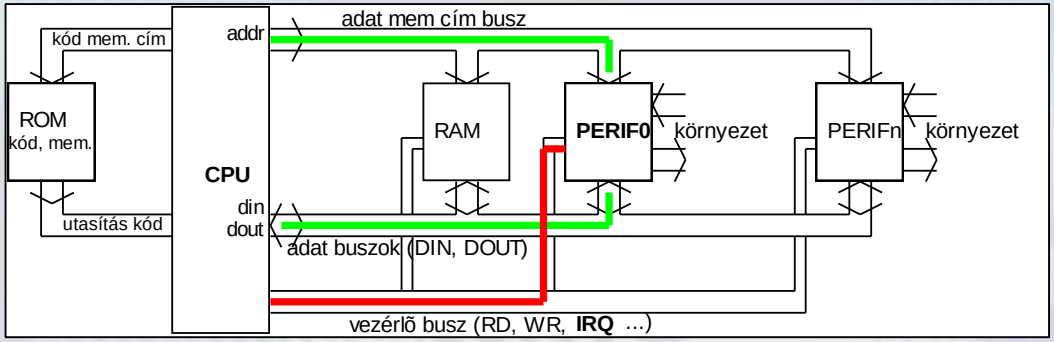
Ha az eredmény nem 0, akkor lejárt az időzítés, egyébként maradunk a ciklusban. Az TS olvasása egyben a TOUT jelzést is törli. Mivel az idő lejártáig a processzor nem tud a várakozásnál hasznosat cselekedni, az alábbi megoldásnál célszerűbb az interruptos periféria kezelés használata (lásd később).

```
DEF TR_INI 121              ;kezdőérték
DEF TC_INI  0b01110011      ;TC-be írandó
DEF   TOUT 0b00000100       ;TOUT bit maszkja

        mov r2, #TR_INI
        mov TR, r2
        mov r2, #TC_INI
        mov TC, r2
tm_loop:
        mov r2, TS          ;státusz olvasás
        tst r2, #TOUT       ;TOUT tesztelés
        jz tm_loop          ;vissza, amíg nem járt le az idő
```

Periféria kezelési módszerek

- A periféria kezelés során információt kell átvinni a periféria és a processzor (és a memória) között.



- Az adatátvitel célszerű módja jelentősen függ a perifériák tulajdonságaitól.
- **Passzív perifériák**
  Mindig kész a működésre, bármikor írható ill. olvasható. Pl. LED display regisztere.
- **Aktív perifériák**
  Nincs mindig kész állapotban ezért, ha elkészül jelzést ad (státusregiszterben, és ha kell interrupt-tal).

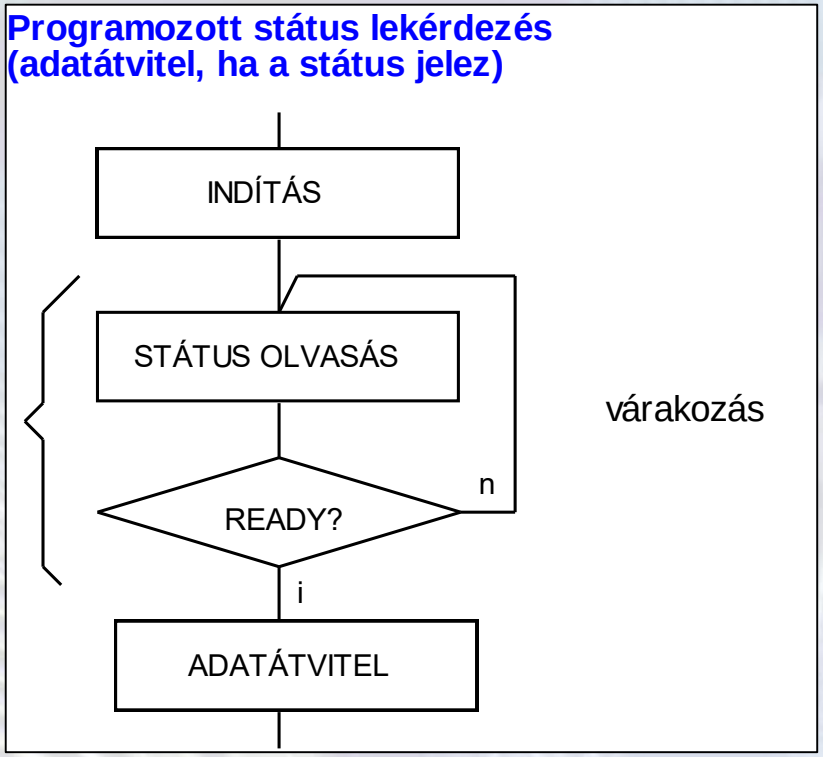
Periféria kezelési módszerek: Programozott státus lekérdezés

- **Aktív perifériák kezelési módszerei**
- **Programozott státus lekérdezés (polling)**
  Akkor célszerű, ha a periféria indítást igényel és az indítás után "rövid idő múlva" készül el, (pl. gyors A/D konverter).
- A processzornak a periféria indítása után meg kell várnia, hogy az elkészüljön.
- ***Ha a periféria a processzorhoz képest elég gyors*** (pl. néhányszor 10 utasítás alatt elkészül), vagy a processzornak nincs fontosabb teendője, ***akkor megengedhető***, hogy a program egy ciklusban addig várjon, míg a státusregiszterből kész jelzést kap, s csak utána folytatja a periféria műveletet (pl. olvassa ki az adatot).

Periféria kezelési módszerek: Programozott státus lekérdezés

- **Programozott státus lekérdezés tipikus folyamatábrája**

- A periféria indítása után addig olvassuk és teszteljük a státusregiszter megfelelő bitjét, amíg az kész jelzést nem ad. Ekkor kimeneti periféria esetén kiírjuk az adatregiszterbe az adatot. Bemeneti periféria esetén beolvassuk az adatregiszter tartalmát, majd feladattól függően feldolgozzuk, vagy eltároljuk a memóriában.

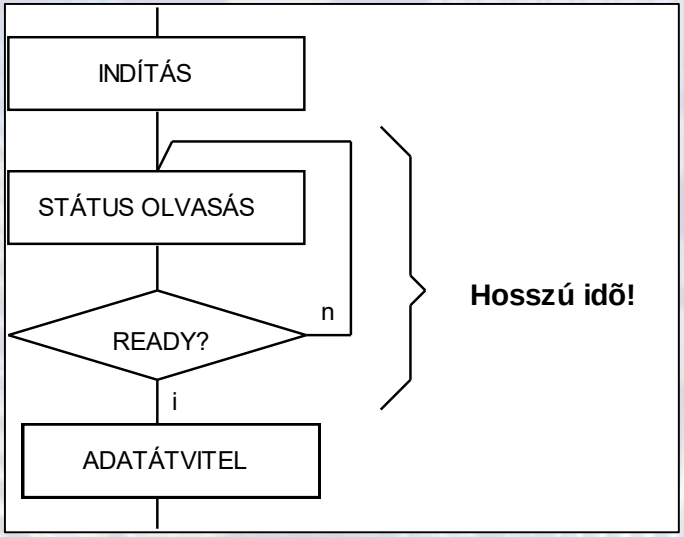


Interruptos periféria kezelés I.

- **Hogyan lehet hatékonyan kezelni az alábbi tulajdoságú perifériákat?**
  - A periféria indítást igényel és az indítás után “hosszú idő múlva” készül el (pl. lassú A/D konverter)
  - A periféria véletlenszerű időpontban termel adatot (pl: kommunikációs periféria adat vétel).

Interruptos periféria kezelés I.

- A státus információra (periféria kész) mindenképpen szükség van. Azonban ***a lekérdezéses várakozási ciklus nagyon hosszú lehet*** (ezalatt a uP nem tud más hasznosat végezni), ezért nem hatékony.



- Rossz a CPU kihasználtsága!
- Több periféria esetén adat veszhet el, mert az adat felülíródhat, mielőtt kiolvassa a CPU!
- Más megoldás kell: **interrupt**
- **A periféria külön vezetéken jelzi, hogy kiszolgálási igénye van. Ezt nevezzük interrupt kérésnek.**
- Az megszakításkor megoldandó feladatokat az **interrupt rendszer** látja el.

Interruptos periféria kezelés I.

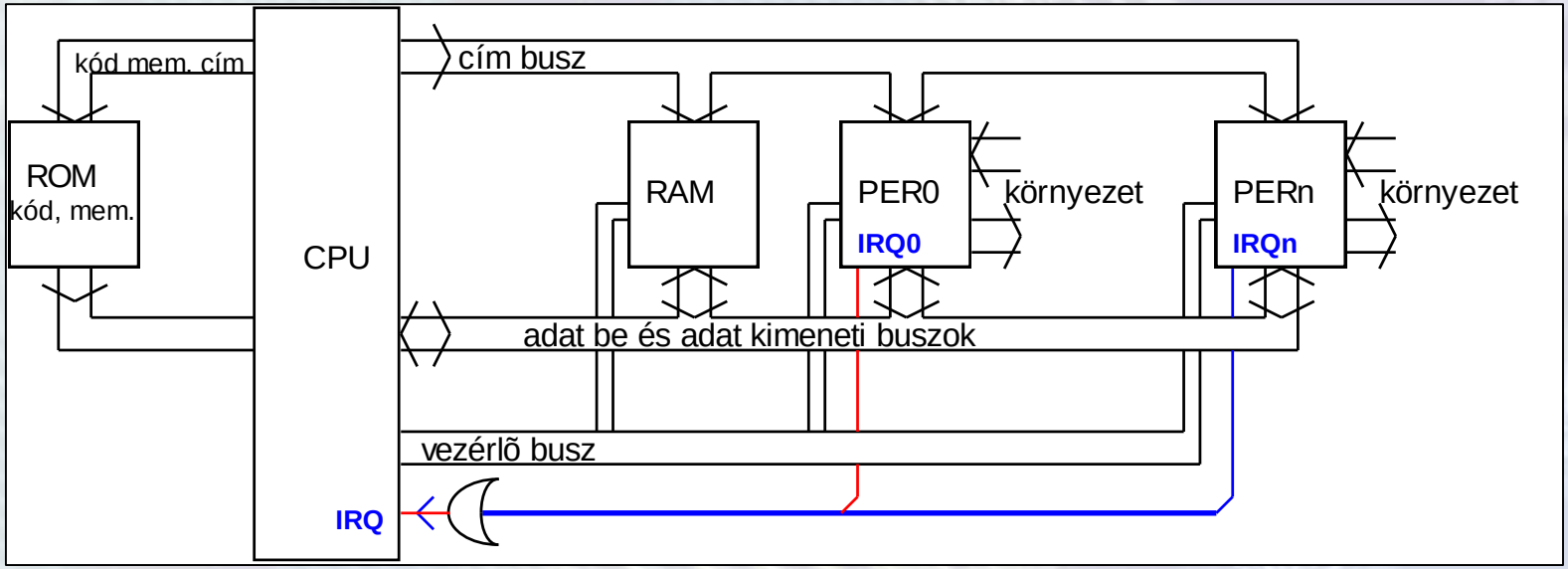
- **Megszakítást kiváltó események:**
  A hardverhez kapcsolódó események jelzése
  - **Normál működés közbeni** kiszolgálási kérések
    - I/O periféria események
    - Időzítő által generált ütemező jelek
    - A megszakítás használatának előnye az egyszerűbb programszervezés
  - **Hibás működés hatására** adott jelzések
    - Memória hiba
    - Tápfeszültség probléma

A megszakítás használatának előnye, az egyébként nagyon ritka eseményekre történő megbízható gyors reakció

Interruptos periféria kezelés I.

Interrupt (megszakítás) végrehajtásának lépései

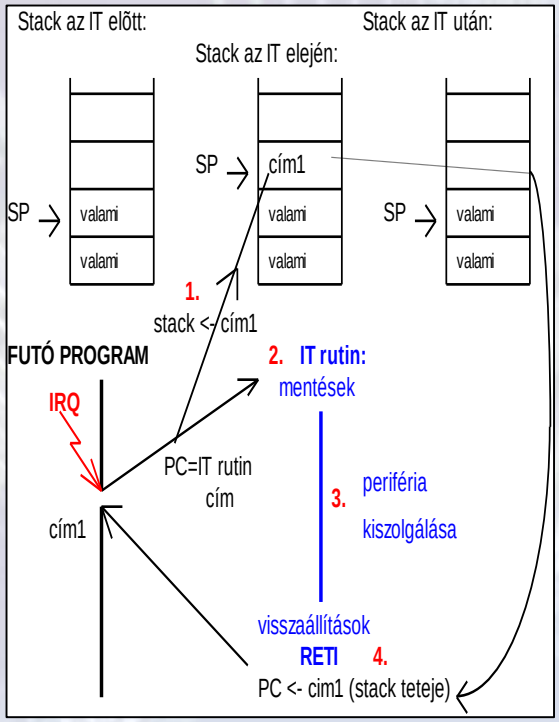
- Ha a perifériának kiszolgálásra van szüksége, ezt maga ***jelzi a CPU-nak*** (vagy interrupt vezérlőnek) egy ***külön jelvezetéken***. Pl. A MiniRISC esetén az alábbi ábra szerint.



Interruptos periféria kezelés I.

Interrupt végrehajtásának lépései

- **Ha az interrupt kérés elfogadásának feltételei teljesülnek, akkor a következők történnek:**
  - A CPU ***az aktuális utasítás végrehatása után elmenti a következő utasítás címét*** (és esetleg egyéb információkat) ***a stack-re*** (1.). (Hasonló történik szubrutin híváskor, de ott csak a következő utasítás címe mentődik el.)
  - ***Elugrik az ISR*** (Interrupt Service Routine, inerrupt kiszolgáló rutin) ***kezdőcímére*** (2.) és ***elmenti a használt regisztereket***.
  - ***Végrehatja az ISR-t*** (3.), amely elvégzi a periféria kezelését.
  - ***Az IT rutin mindig az IT-ből visszatérő utasítással fejeződik be***, (neve: ***rti***, ***reti***) ami ***visszamenti a használt regisztereket***, ***leveszi a stack tetejéről a visszatérési címet és oda ugrik*** (4.) (folytatja a megszakított programot) és más is történik (lásd később).



A konkrét interrupt rendszertől függően egyebek is történhetnek, de ezeket az IT rendszerek leírásánál részletezzük.

Interruptos periféria kezelés I.

- **Megszakításrendszerek feladatai:**
  - Megszakítási események **jelzése**
  - Megszakításkérés **prioritásának kezelése** (több IT kérés..)
  - Megszakításkérések **engedélyezése / tiltása**
  - Megszakítás **forrás azonosítása** (ki érte)
  - Megszakítás kérés **kiszolgáló szubrutin kezdőcímének meghatározása** (hol kezdődik az IT rutin)
  - Megszakításkérés **kiszolgáló szubrutin** (ISR, Interrupt Service Routine) **végrehajtása**, a szükséges **állapot mentés elvégzése**
  - Visszatéréskor a **program eredeti állapotának visszaállítása**

Folytatjuk a következő előadáson...

Digitális technika
11. EA vége