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

# Digitális technika VIMIAA02 9. hét

Fehér Béla, Benesóczky Zoltán
BME MIT

# Processzor utasítás rendszerek

- **A processzorok utasításkészlete az alábbi utasítás csoportokra osztható:**
  - **Adatmozgató** (MOV, LD, ST...)
  - **Aritmetikai** (ADD, SUB, MUL, DIV….)
  - **Logikai** (AND, OR, XOR, NOT…)
  - **Léptetés** (SHL, SHR, ASH…)
  - **Forgatás** (ROL, ROR,…)
  - **Feltétel vizsgálat** (CMP, TST,…)
  - **Vezérlő** (JMP, CJMP, CALL/RET,...)
  - **Egyéb** (NOP, EI/DI, HALT,...)
- **Jellemző RISC utasításkészlet: 50 – 150 utasítás**

# Processzor utasítás rendszerek

- **A RISC processzorok utasítás formátuma**
  - Általában **fix hosszúságú,** 16/32 bit
  - Felépítése **2 regiszter címes (2R) vagy 3R típusú** (utóbbi a 32 bitnél)
  - **Utasítás formátum: az utasítások egyes *bitmezőinek elhelyezkedése és jelentése***. (Bitmezők jelentése: **műveleti kód, op1reg, op2reg, konstans adat, konstans cím** *stb.*)
  - **Kevés utasítás formátumot használnak** (2-5 maximum)
    - **Egy formátumon belül minden utasítás azonosan használható**
  - **A bitmezők kiosztása** (mely sorszámú biteken van) **rögzített**
    - **Egyszerű a dekódolása, sok érték közvetlenül használható az processzor adatstruktúrájában**
  - **A RISC processzorok utasítás végrehajtási ideje azonos, (1 órajel). Párhuzamosítás:** Egy utasítás végrehajtási fázisában a következő utasítást már beolvassa. (A Mini RISC nem..)

# Processzor utasítás rendszerek

- A processzor utasításaira a program írása során a nevükkel (**mnemonik**) hívatkozunk **assembly nyelvű programozás**kor.
- A különféle processzorok ugyanazon műveletet elvégző utasításainak nevei sokszor kissé eltérőek. (Pl. adatmozgatás:MOV, LOAD, LD, STORE, ST,..)
  - A továbbiakban többnyire a MiniRISC CPU utasításait használjuk a példákban.
- **Az assembly programot a compiler lefordítja a processzor utasítkódjára**.
- A lefordított **utasítás kódokat betöltik a processzor utasítás** (kód) **memóriába**.

# Processzor utasítás rendszerek

- **A RISC processzorok címzési módjai**
  - **A címzési mód:** ***az utasításban hivatkozott információ elérési módja*** (Pl. ***Hol vannak az operandusok? (utasítás kódban, melyik regiszterben, milyen memória címen) Hol van az ugrási cím?***
  - **A címzési módok jelentősen befolyásolják az utasítás ill. programvégrehajtás hatékonyságát.**
  - **Különböző címzési módok léteznek** a ***programmemória*** és az ***adatmemória*** elérésére.
    - Eltérő igények:
      - Programmemória esetén a cél a ***következő utasítás címének meghatározása***, ez ugrásoknál, feltételes ugrásoknál, szubrutinhívásoknál (függvény/eljárás) lényeges.
      - Adatmemóriánál általánosabb igények vannak. (***Hol vannak az operandusok?)***

# Processzor utasítás rendszerek

- **Programmemória (kódmemória) címzési módjai**
- A programmemóriát a **PC** (programszámláló) címzi
- **A következő utasítás címe általában: PC = PC + 1**
- **Az ugrás vagy szubrutinhívás/visszatérés esetén a PC értékét módosítani kell.**
  - **Abszolút címzés**: A ***PC minden bitje módosul***: A teljes programmemória bármely címe szerepelhet, mint új cím. Kedvező, egyszerűen használható, de sok bitet igényel.
  - **Relatív (PC relatív): címzés** A ***PC bitjeinek csak egy része módosul.*** **Hátránya:** ***Korlátozott a következő utasítás aktuálistól mért távolsága***. A programszervezés bonyolultabb (assembler/fordító feladata), de **a programok „lokalitása”** miatt ez ritkán okoz problémát, mert **a programokban az ugrások legyakrabban kis címtávolságba történnek** (pl. ciklus szervezés).
    *Pl. Egy **16 bites PC-hez 8 bites 2-es komplemens relatív cím adódik hozzá**. Így az **ugrási tartomány PC-128-tól PC+127-ig** terjed.*

# Processzor utasítás rendszerek

- **Programmemória címzési módjai** (közvetlen, regiszter indirekt)
  - **Közvetlen cím:** Az ***ugrási cím értéke az utasításkód része***, ez gyakran relatív címzésre korlátoz (korlátos utasítás szószélesség)
    - Az **ugrási cím** fordítási időben meghatározott, futáskor **konstans**
    - **MiniRISC**: 16 bit utasítás, 8 bit program cím: lehet abszolút is
    - Pl: jmp **addr** jmp 0b**10101010**

      Kódja: 10110000**aaaaaaaa** 10110000**10101010**

      Az ugrási cím (addr) az utasítás kód ‚a'-val jelölt bitjeiben van.

# Processzor utasítás rendszerek

- **Regiszter indirekt:** ***Az ugrási címet egy regiszter tartalmazza*** (ez általában abszolút címzésre is elegendő)
  - Az ***ugrási cím adatfüggő lehet*** (regiszterben), több utas elágazások megvalósítására. (Előtte a ***regiszterbe be kell tölteni a megfelelő utasítás címét***.)
  - **MiniRISC**: ***van regiszter indirekt címzés*** (az ugrási cím egy regiszterben van), teljes tartományra működik, mert kicsi (8 biten címezhető) a teljes kódmemória.

    Pl: jmp (**r**y) jmp (r**5**)

    Kódja: 11111011**yyyy** 11111011**0101**

    Az ***ugrási cím az ry regiszterben van***. (A regiszter sorszáma az ‚y'-al jelölt biteken található.)



- Léteznek egyéb címzési módok, nem részletezzük

# Processzor utasítás rendszerek

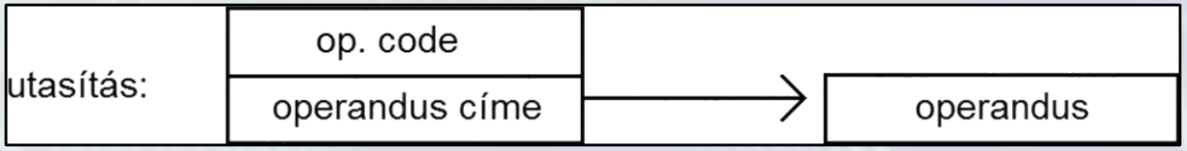
- **Adatmemória** címzése, illetve az **operandusok helyének megadása:**
  - **Közvetlen adat (adat az utasításban)**
  - **Közvetlen cím (az adat memóriacíme az utasításban)**
  - **Regiszter adat (adat a regiszterben)**
  - **Regiszter indirekt (adat címe a regiszterben)**
  - Regiszter + közvetlen ofszet című adat
  - Regiszter címzésű indexelt adat
  - Regiszter címzésű post-inkremens/pre-dekremens adat
- **Egy-egy utasításkészlet nem mindegyiket tartalmazza**
- **Léteznek egyéb speciális címzések is, ezeket nem tárgyaljuk**

# Processzor utasítás rendszerek

- **Közvetlen (immediate) adat**:
  ***Az adat az utasításkód része***
  - MiniRISC 16 bit utasítás, 8 bit adat → megoldható
    Pl: add rx, #**imm** add r3, #0b**10101010**
    kódja 0000xxxx**iiiiiiii** 0000 0011 **10101010**
    Az adat (operandus) az utasítás kód i-vel jelölt bitjeiben van.
  - Általában nem egyszerű (Pl. az utasítás szószélessége 32 bit): ***32 bites utasításban 32 bites adat nem helyezhető el***, mert akkor más információnak nem marad hely.
  - Ilyenkor gyakran használnak ***„kis értékű" konstansokat***, a közvetlen adatra egy ***8-12-16-20 bites bitmező*** áll rendelkezésre

# Processzor utasítás rendszerek

- **Közvetlen cím**: ***Az adat memóriacíme az utasítás része***




**MiniRISC** 16 bit utasítás, 8 bit adat → megoldható, mert az adatinterfész csak 256 címet tartalmaz

Pl: mov rx, **addr** mov r3, 0b**10101010**
kódja 1101xxxx**aaaaaaaa** 1101 0011 **10101010**
Az operandus címe az utasítás kód ‚a'-val jelölt bitjeiben van.

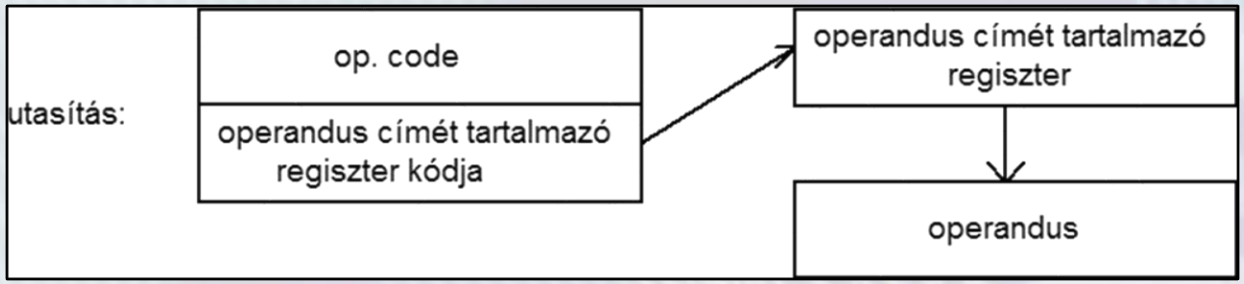
- Általában ugyanazok a korlátozások érvényesek, mint a közvetlen adatnál, bár beágyazott rendszerekben a memóriaméret korlátos (nem kell 32 címbit)

# Processzor utasítás rendszerek

- **Regiszter adat**: ***Az adatot regiszter tartalmazza*** (***a regiszter címe az utasítás része***)
  - MiniRISC 16 bit utasítás:
    2 regiszter címes architektúra → max. 2 regiszterben lehet adat
  - Ez a leggyakoribb címzési mód
  - RISC processzorokban minden regiszter egyenértékű, egyformán minden utasításban használhatók. (Vannak processzorok, ahol ez nem igaz.)
    Pl: add **rx**, **ry** add r**3**, r**1**
    kódja 1111**xxxx**0000**yyyy** 1111 **0011** 0000 **0001**
    Az adatok az rx és ry regiszterekben vannak. (***Az rx és ry regiszterek sorszáma az utasítás kódban található***.)

# Processzor utasítás rendszerek

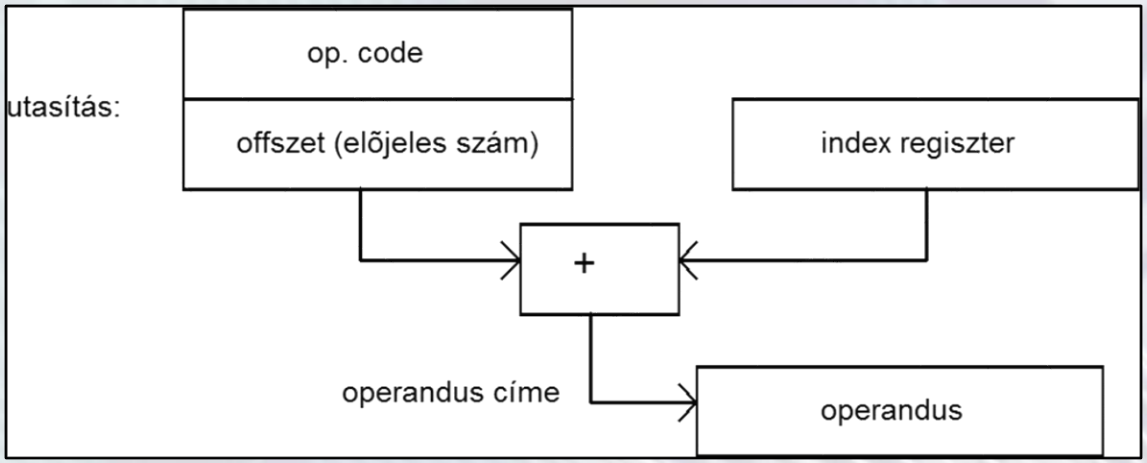
- **Regiszter (indirekt) címzésű adat: Az adat *memóriacímét* regiszter tartalmazza**




- A MiniRISC utasításkészlet tartalmazza
  Pl: mov rx, **(ry)** mov r3, (r**1**)
  kódja 1111xxxx1100**yyyy** 1111 0011 1101 **0001**
  Az operandus (adat) címe az ry regiszterben van.
  (Az rx és ry regiszterek sorszáma az utasítás kódban található.)
- Hatékony címzés ***adatvektorok/tömbök eléréséhez, mert a regiszter tartalma módosítható.***

# Processzor utasítás rendszerek

- **Regiszter + közvetlen offset adat:** ***Az adat címe egy regisztertartalom (bázis/index regiszter) és az utasításból származó közvetlen érték (offset) összege***




Pl: AVR mikrokontrollernél:

| Mnemonik | kód formátuma | Rd | offset |
|---|---|---|---|
| LDD Rd, Y+q | 10q0 qq0d dddd 1qqq | ddddd | qqqqqq |

Az adat címe: Y regiszter tartalma + q

# Processzor utasítás rendszerek

- **Regiszter címzésű indexelt adat**: ***Az adat címe két regiszter tartalmának összeadásából adódik***
  Az egyik a bázis cím a másik az index érték
  **adat címe = báziscím_reg + index_reg**

- **Regiszter címzésű post-inkremens/pre-dekremens címzés**: Az adat elérése után/előtt a címregiszter tartalma automatikusan módosul. Hasznos pl. tömb másolásánál.

  Pl: AVR mikrokontrollernél:
  LD Rd, **X+**
  Az adatot betölti az X regiszterben levő címről, majd inkrementálja X tartalmát (a címet).

# MiniRISC utasításkészlete

**A MiniRISC processzor diasorozat 1 - 59 diája**

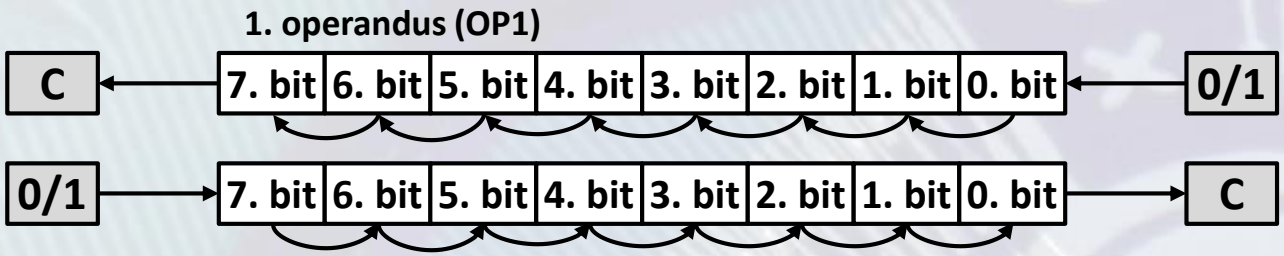
**(nem ismertetjük részletesen, de kérjük mindenki nézze át!)**

**A továbbiakban néhány érdekesebb utasítás működését ismertetjük.**

# MiniRISC utasításkészlete

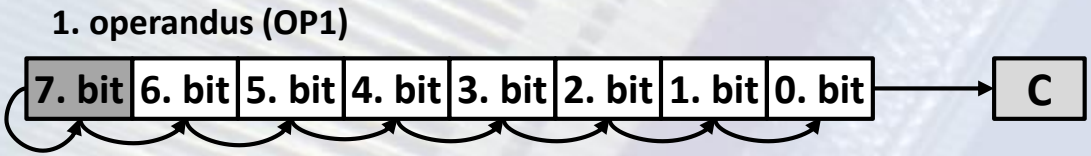
## Logikai shiftelés

- 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

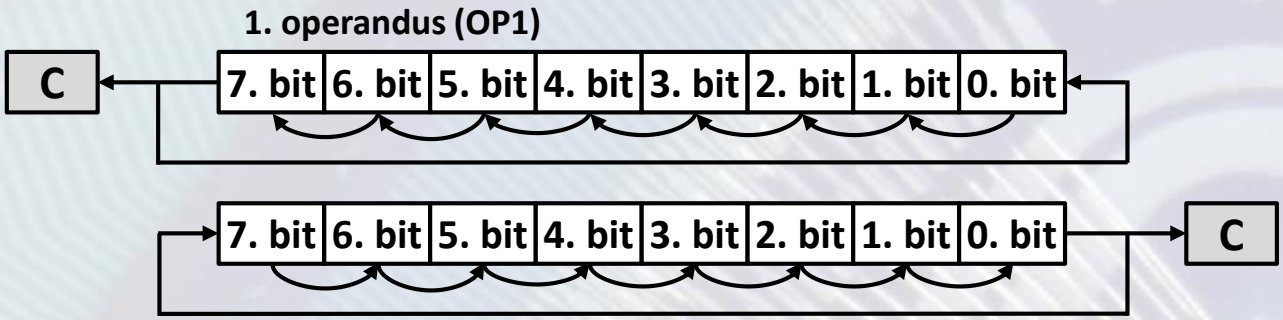
- Előjeles számok jobbra léptetése esetén az előjel bitet (MSb) helyben kell hagyni, hogy helyes eredményt kapjunk
- A kilépő bit a carry (C) flag-ba kerül
- Külön balra történő aritmetikai shiftelés nincs, mert ez megegyezik a balra történő logikai shifteléssel



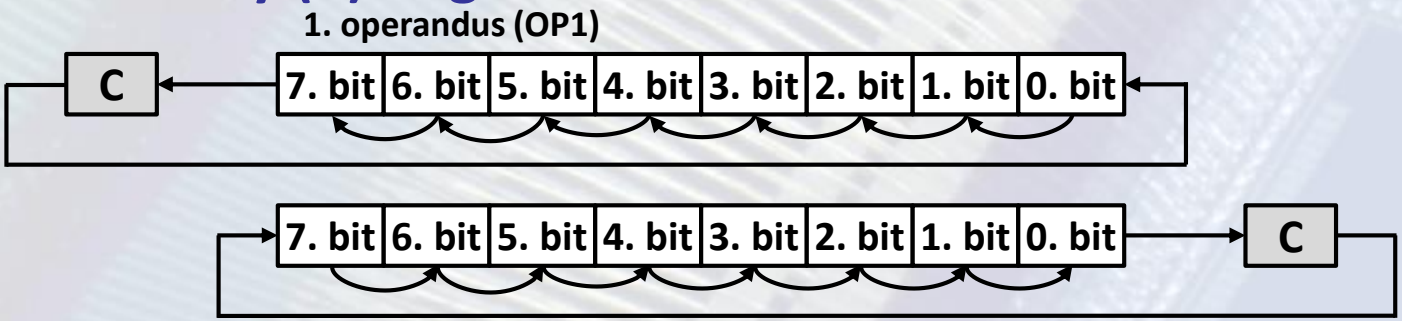
# MiniRISC utasításkészlete

## 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

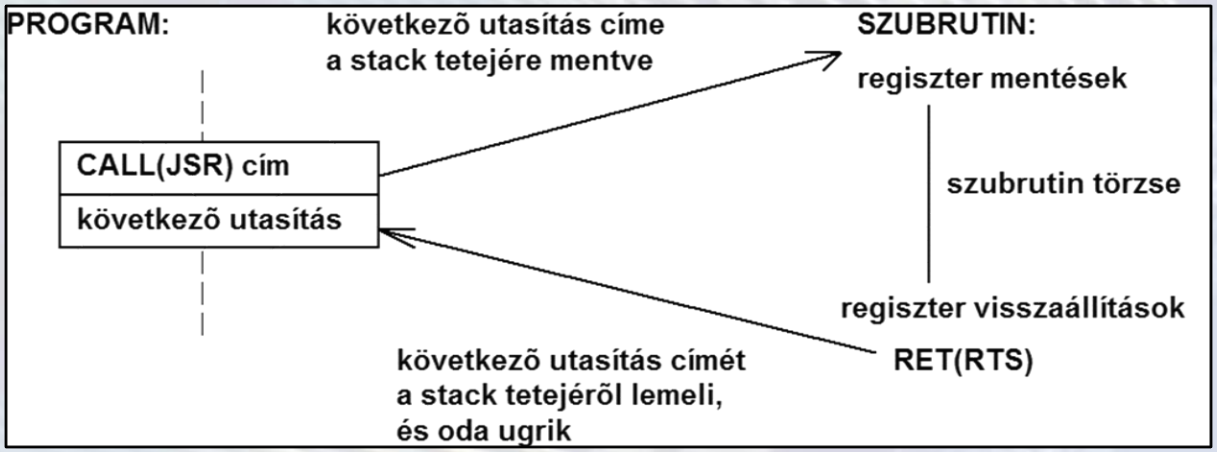



## Forgatás a carry (C) flag-en keresztül



# MiniRISC utasításkészlete

- **Szubrutin**
- A szubrutin hívó utasítás egy speciális ugró utasítás, amely az ugrás előtt a stack-re menti az utána következő utasítás címét. Elnevezése  többnyire **CALL**, de a MiniRISC-nél **jsr.**
- Az ugrási címen egy valamilyen feladatot megvalósító program van elhelyezve. Ennek utolsó utasítása a szubrutinból visszatérő utasítás, neve többnyire **RET**, a MiniRISC-nél **rts.** Ez is speciális ugró utasítás, mely a stack tetejéről kiveszi a CALL/jsr által elmentett címet és oda ugrik, vagyis a szubrutin hívó utasítást követő utasításra.



# MiniRISC utasításkészlete

- ***Szubrutinként célszerű megírni az olyan részfeladatot megvalósító programrészt, amelyet  többször végre kell hajtani a program különböző részeiben***.
- Így a programrészt csak egyszer kell leírni. ***A kódmemóriában is csak egyszer szerepel***. Viszont ***tetszőleges alkalommal felhasználható***. Ha használni akarjuk, csak le kell írni a megfelelő címet tartalmazó hívó utasítást.
- ***Ha paraméterei is vannak*** a szubrutinnak (bemenő adatok és/vagy kimenő adatok). Ezeket ***átadhatjuk regiszterekben, vagy memória területen***.
- Bizonyos esetekben célszerű lehet a szubrutin által használt regiszterek egy részét a szubrutin elején elmenteni, a végén visszamenteni.

```
00: start:
00:    mov r0, #0xc0
01:    mov LD, r0
02:    mov r1, #0
03:    mov r2, #121
04:    mov TM, r2
05:    mov r2, #0x73
06:    mov TC, r2
07:    mov r2, TS
08: loop:
08:    jsr tmr_wait
09:    cmp r1, #0
```

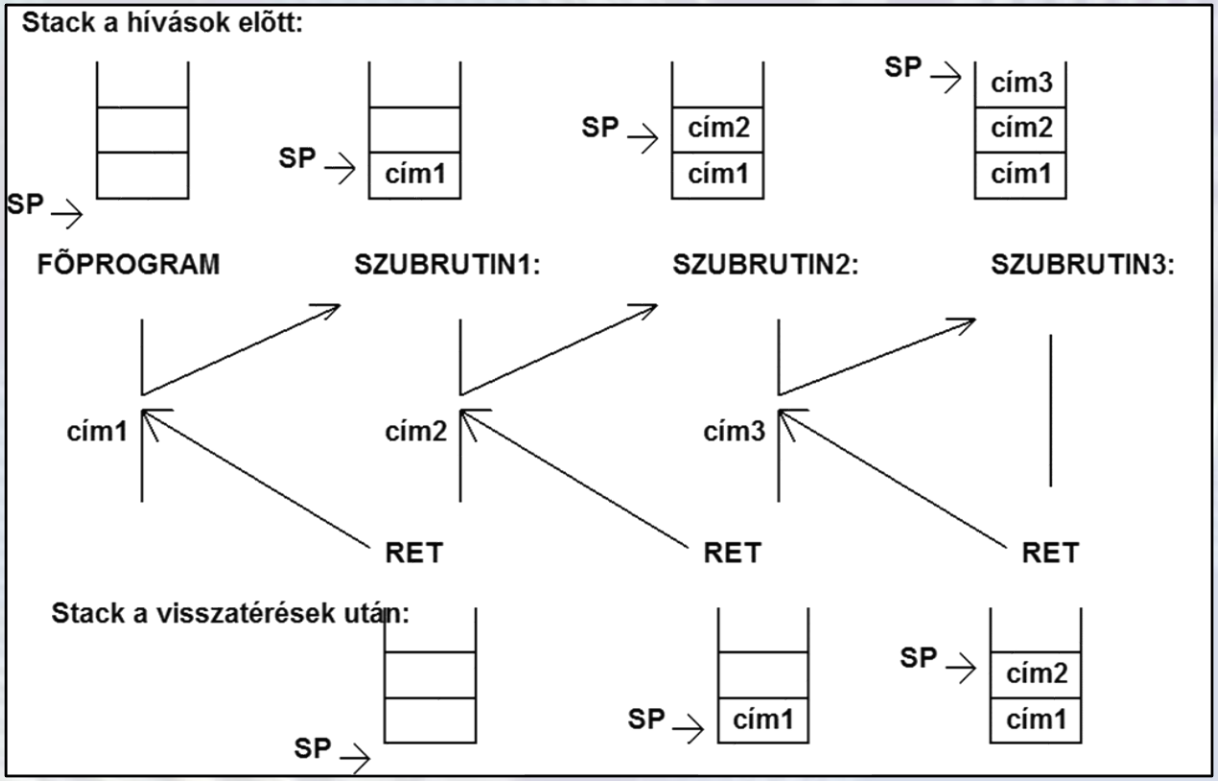
stack←PC (0x09)
PC←tmr_wait

PC←stack (0x09)

```
20: tmr_wait:
20:    mov r0, TS
21:    tst r0, #0x04
22:    jz  tmr_wait
23:    rts
```

# MiniRISC utasításkészlete

- **Egymásba ágyazott szubrutin hívások**
- A szubrutin hívásokat a stack terület által korlátozott mértékben egymásba lehet ágyazni, vagyis egy-egy szubrutinból újabbakat lehet hívni. Ezt a mechanizmust a stack teszi lehetővé.



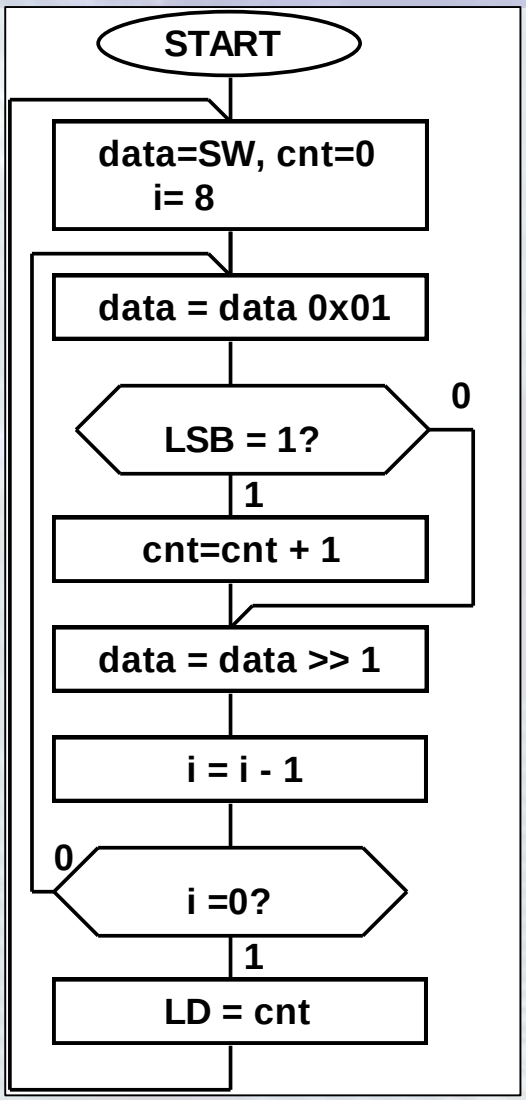
# Assembly programozás

- Az assembly nyelvű programozás során a magas szintű nyelvektől eltérően csak a processzor utasításkészletét használhatjuk.
- Ez az utasításkészlet a magas szintű nyelvektől eltérően csak egyszerű műveletek elvégzésére képes.
- Minden utasításnak van egy elnevezése „mnemonik-ja” amely néven hivatozhatunk rá.
- Minden komplexebb feladatot ezek felhasználásával kell megvalósítanunk. Ez ugyanazon feladat esetén sokkal több utasítást használatát és programozói munkát igényel, mint a magas szintű nyelvek esetében.

# Assembly programozás

- A program megtervezésére és grafikus jellegű leírására a 7. előadáson ismertetett ASM állapot és feltétel dobozát használjuk, azonban az állapotnévre és állapotkódra itt nincs szükség.
- Példa: Tervezzünk programot amely megszámolja az SW kapcsolóról beolvasott adatban 1-esek számát és kírja a LED-ekre (LD).
- Folyamatábra a következő oldalon.

# Assembly programozás




Adat beolvasása, cnt bit számláló és i ciklus számláló inicializálása.

Adat LSB maszkolása, hogy a z flag beálljon.

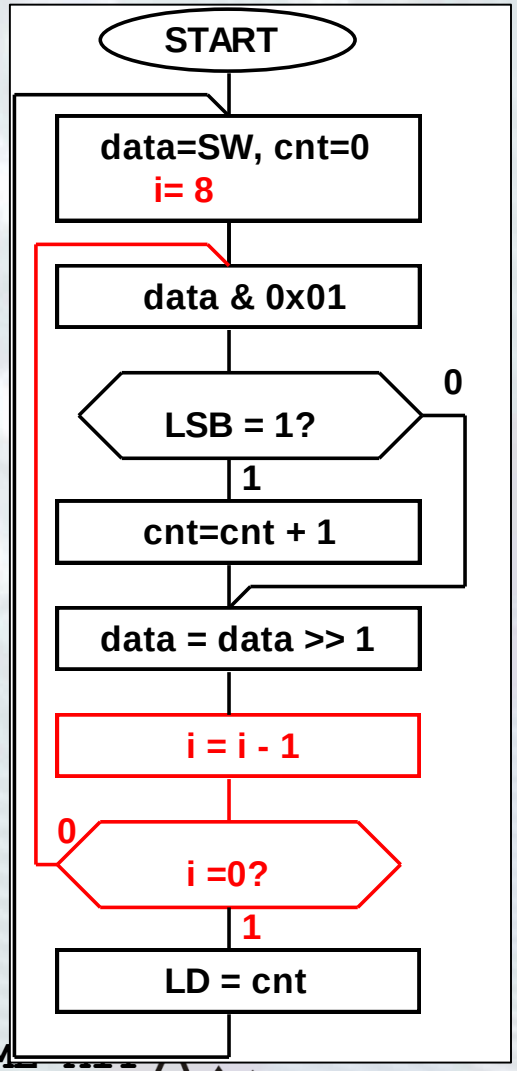
Bit számolás, ha LSB 1.

Következő bit az LSB-be.

Ciklus számláló csökkentése.

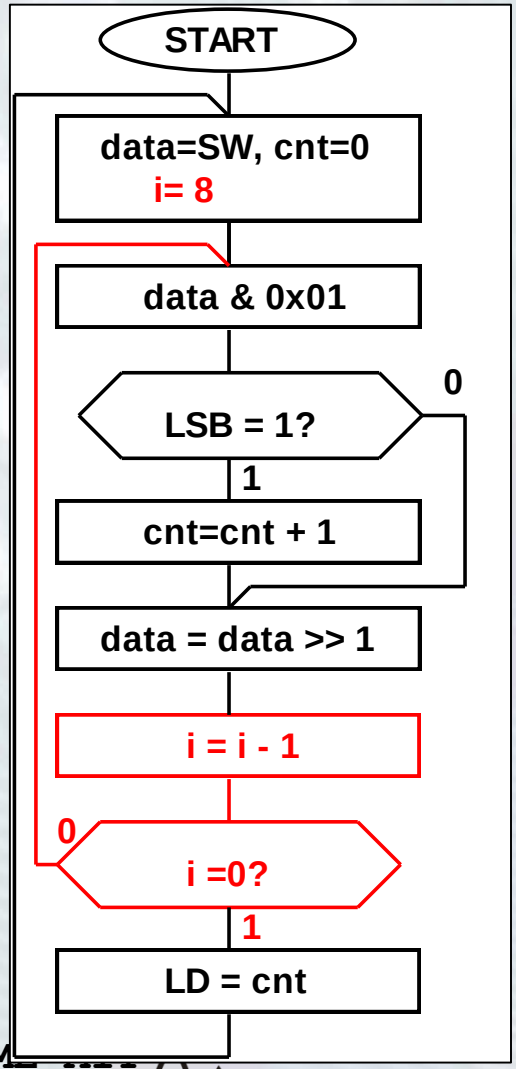
Ha nem járt le a ciklus szám, vissza, egyébként kijelzés és kezjük elölről.

# Assembly programozás




```
DEF LD 0x80
DEF SW 0x81
start:    mov r0, SW          ;r0 az adat
          mov r1, #0          ;r1 a bitek száma
          mov r2, #8          ;r2 a ciklus számláló
loop:     tst r0, #0b00000001 ;z flag beáll az LSB alapján

          jz no_add           ;ugrás, ha nem kell számolni

          add r1, #1          ;bit számlálás

no_add:   shr0 r0             ;következő bit az LSB-be

          sub r2, #1          ;ciklus számláló csökkentése.

          jnz loop            ;vissza, ha még van bit
          mov LD, r1          ;eredmény kiíratás
          jmp start           ;vissza a kezdésre
```

# Assembly programozás




- A programokban gyakran előfordul, hogy egy utasítás sorozatot megadott számúszor ismételni kell.
- Ennek megvalósítását nevezzük ciklus szervezésnek.
- Az ábrán és az előző oldal assembly program kódjában kiemeltük a ciklus szervezéshez tartozó részeket.
- A ciklusba lépés előtt inicializálni kell a ciklus számlálót.
- A ciklus törzsében van az ismételendő utasítássorozat.
- A ciklus végén ellenőrizni kell, hogy megfelelő számúszor végrehajtódott-e a ciklus.
  Ha ciklus számláló kezdőértékének a ciklus számot adjuk, akkor az ellenőrzés előtt azt csökkenteni kell és a 0 érték jelzi, ha vége. Ha a ciklus számlálót az elején 0-ra inicializáljuk (i=0), akkor az ellenőrzés előtt növelni kell és az ellenőrzésnél az igényelt cikusszámmal kell összehasonlítani (i=N?). Az előbbi megoldás assembly nyelven 1-el kevesebb utasítással megoldható, mert a dekrementálás maga beállítja a z flag-et. A második esetben ehhez még egy cmp r2, #8 utasítás is kell.

# Assembly programozás

Az Mini RISC processzor assembly nyelvű programozásához a MiniRISC CPU című kiegészítő jegyzet 3. fejezetében találhatók további információk.

# Digitális technika
# 10. EA vége