BELÉPTETŐ RENDSZER TERVEZÉSE

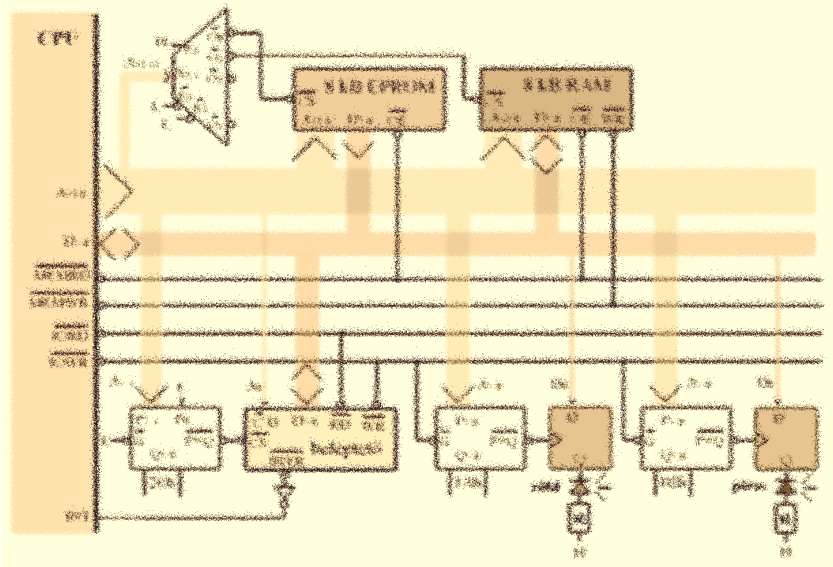
Számítógép-architektúrák

2. gyakorlat

Dr. Lencse Gábor
lencse@hit.bme.hu

Dr. Koller István
koller@hit.bme.hu

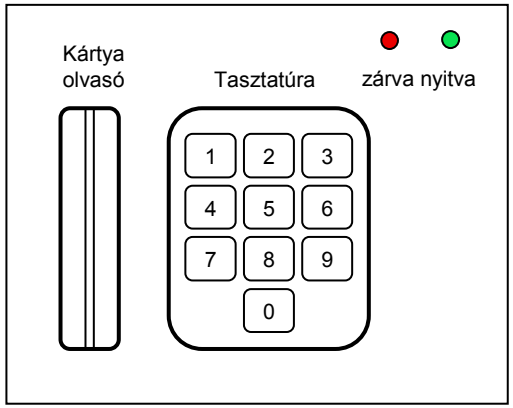
2014. szeptember 25., Budapest

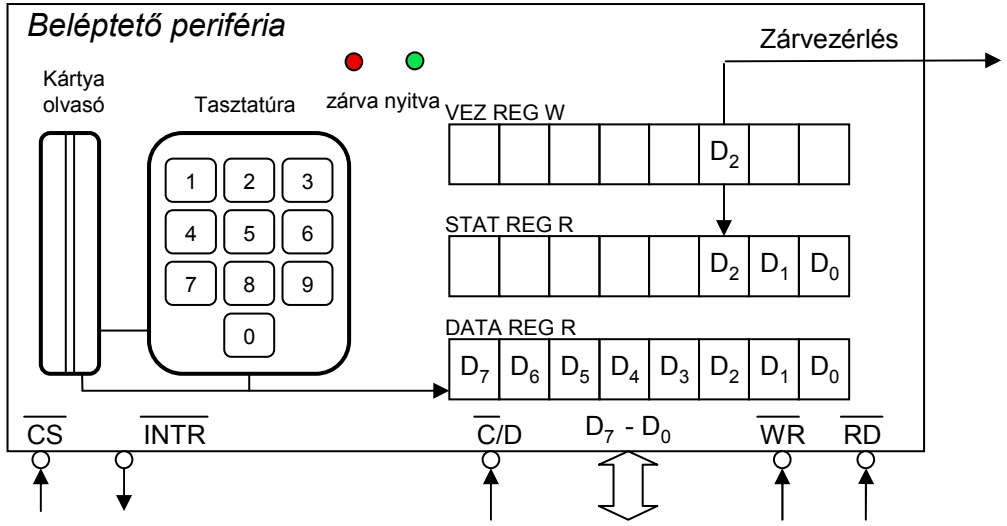


ISMÉTLÉS: A BELÉPTETŐ PERIFÉRIA MŰKÖDÉSE + A KÉSZ HARDVER

A beléptető periféria működése – 1

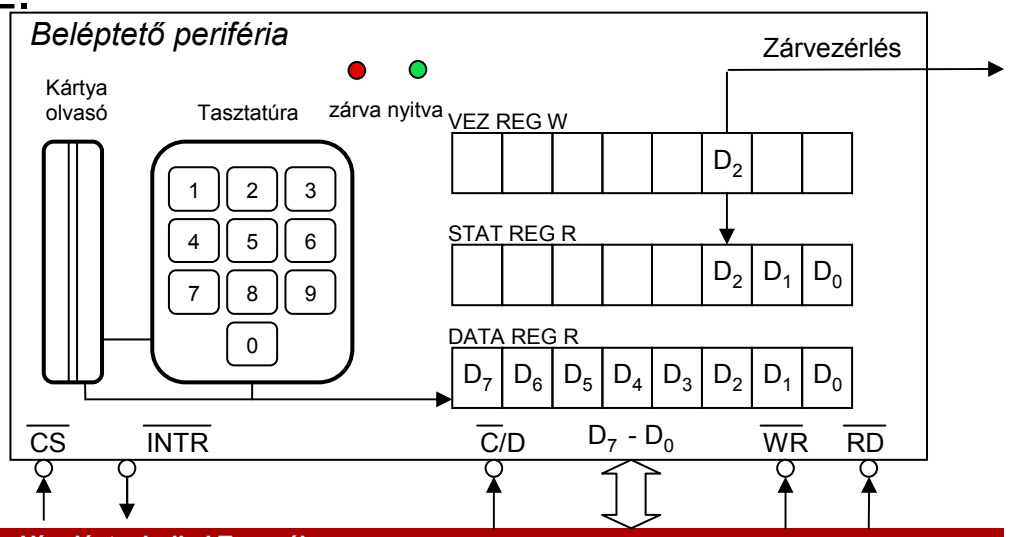
- Minden kártyához (kártyaazonosítóhoz) tartozik egy 4 decimális jegyből álló kód,
- a zárat akkor kell nyitni, ha a kártya lehúzása után megnyomott első 4 billentyű éppen a hozzá tartozó kódot adja.
- Hardverből megoldott: mindig csak lehúzás, vagy gombnyomás, egyszerre nem





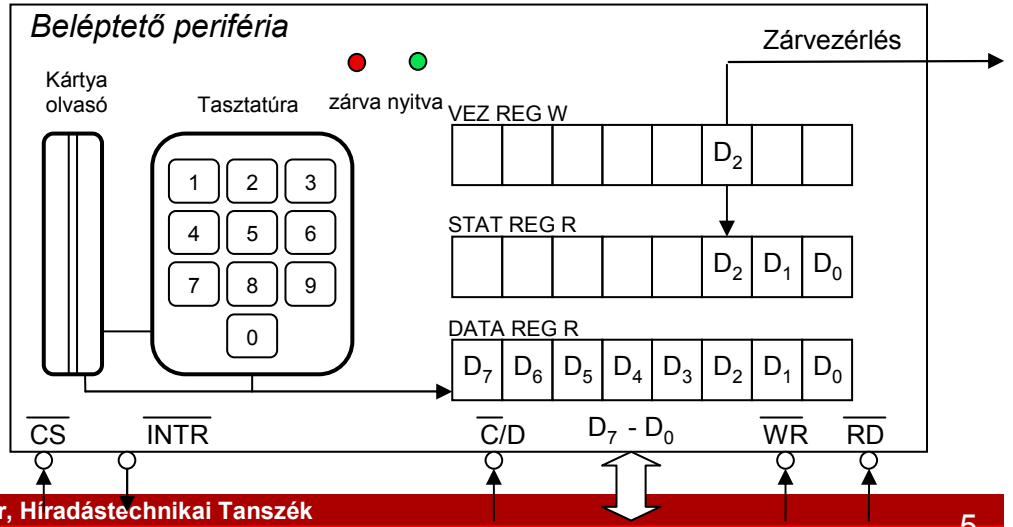
A beléptető periféria működése – 2

- Amikor a **kártyát lehúzzák**,
  - a beléptető /INTR lába alacsony szintűre vált,
  - a státuszregiszteréből olvasva a D_0 bit 1-es értékű lesz (akárhányszor is kiolvasható)
  - ezek mindaddig fennállnak, amíg az adatregiszterből ki nem olvasták a kártya azonosítóját
  - az adatregiszterből kiolvasható (csak egyszer!) a lehúzott kártya azonosítója,
  - utána rögtön /INTR magasra vált, és a státuszregiszteréből olvasva a D_0 bit értéke 0 lesz.



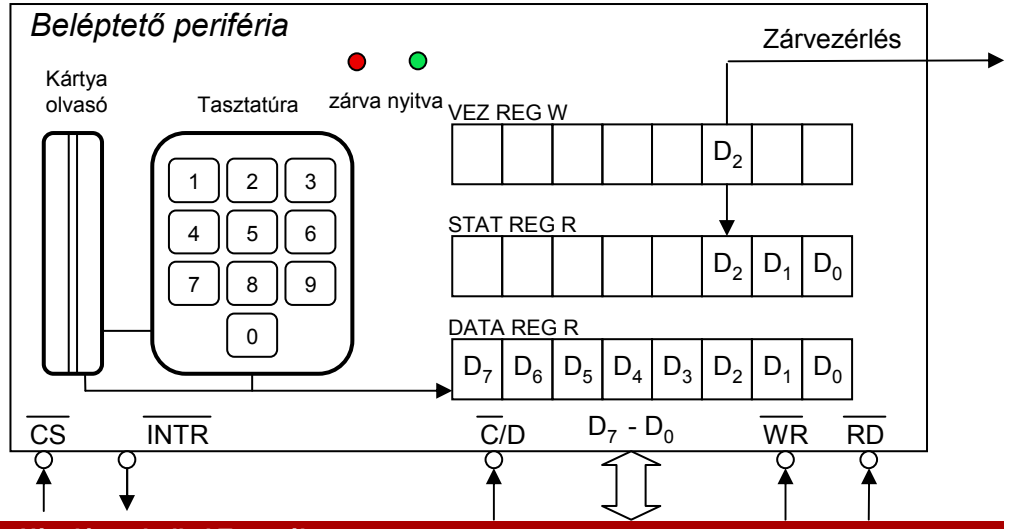
A beléptető periféria működése – 3

- Amikor egy **számjegyet beütöttek**,
  - a beléptető /INTR lába alacsony szintűre vált,
  - a státuszregiszteréből olvasva a D_1 bit 1-es értékű lesz (akárhányszor is kiolvasható)
  - ezek mindaddig fennállnak, amíg az adatregiszterből ki nem olvasták a számjegy értékét.
  - Ezután az adatregiszterből kiolvasható (csak egyszer!) a beütött számjegy értéke,
  - utána rögtön /INTR magasra vált, és a státuszregiszteréből olvasva a D_1 bit értéke 0 lesz.

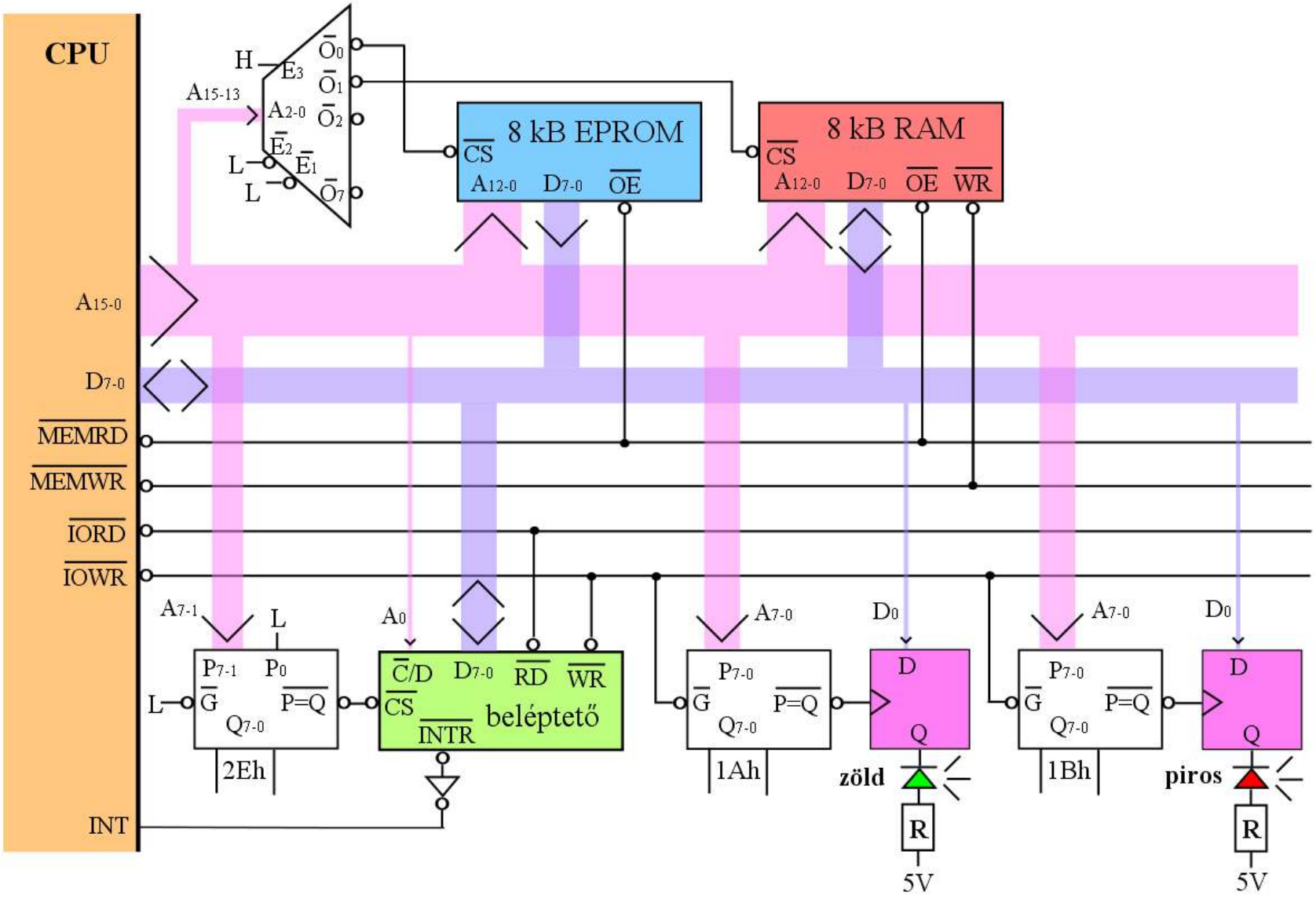


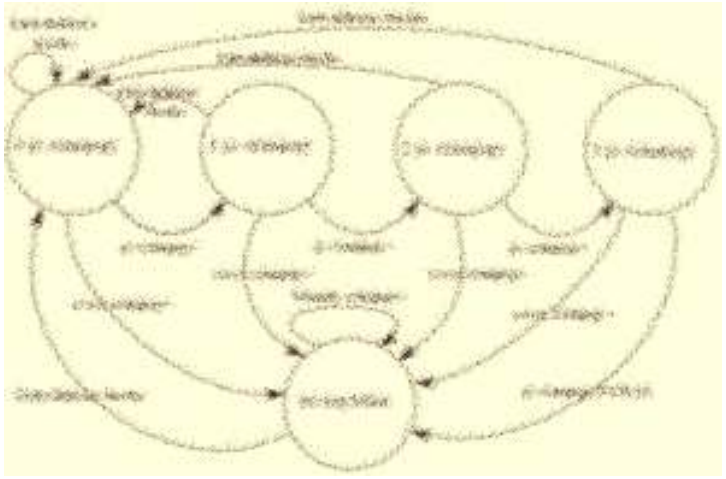
A beléptető periféria működése – 4

- A vezérlő regiszter D_2 bitjével állítható a zár állása:
  - 0: zárás parancs,
  - 1: nyitás parancs.
  - A nyitás parancs után, amint az ajtót kinyitották rögtön, de legkésőbb 10s után (timeout: ha az ajtót addig nem nyitották ki) a beléptető zárnyelv vezérlője automatikusan átmegy zárt állapotba.
- A státuszregiszter D_2 bitjéből mindig kiolvasható a zár állása: 0: zárva, 1: nyitva



A teljes logikai kapcsolási rajz

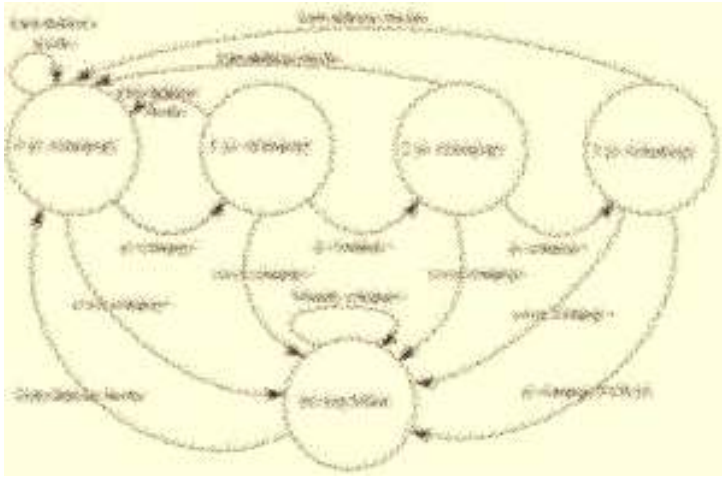




A SZOFTVER FELADAT KITŰZÉSE

A készítendő szoftver feladata

- A beléptető periféria zárja kezdetben zárt állásban van (így indul, ez nem a mi dolgunk).
- Követelmény, hogy:
  1. A LED-eknek mindenkor tükrözniük kell a zár állapotát (beleértve a nyitást, az automatikus zárást, valamit az ajtó fizikai nyitása által kiváltott zárást is).
  2. Kártyalehúzást követő helyes 4 jegyű kód megadása esetén a zárat nyitni kell (függetlenül attól, hogy éppen zárt vagy nyitott állásban van). Minden más bemeneti szekvencia esetén a zárat nem szabad kinyitni.



A SZOFTVER TERVEZÉSÉNEK MENETE (TERVEZŐI DÖNTÉSEK)

Előfeltevés, észrevétel

- Feltétel
  - Az ajtónyitó periféria biztosítja, hogy a kártyalehúzás és a számjegyleütés események közül egyszerre csak az egyik következik be – D0, D1 közül mindig csak az egyik 1-es – beépített PIC
- Észrevétel
  - Mivel a zárásról semmiféle értesítést sem kapunk, a LED-ek állapotát a D3 nyitás-zárás státus bit gyakori olvasásával kapott érték szerint állítjuk.

Tervezői döntések

1) A zár állapotát lekérdező ciklusban a LED-ek állapotát
   a) minden lekérdezés után beállíthatjuk
   b) az előző állapotukat a programban nyilvántartva elegendő csak változás esetén állítanunk
   - Döntés: A zár állapotát lekérdező ciklusban a LED-ek állapotát minden lekérdezés után beállítjuk.
     (Így az állapotukat nem kell nyilvántartanunk.)

2) ...

További döntések - 1

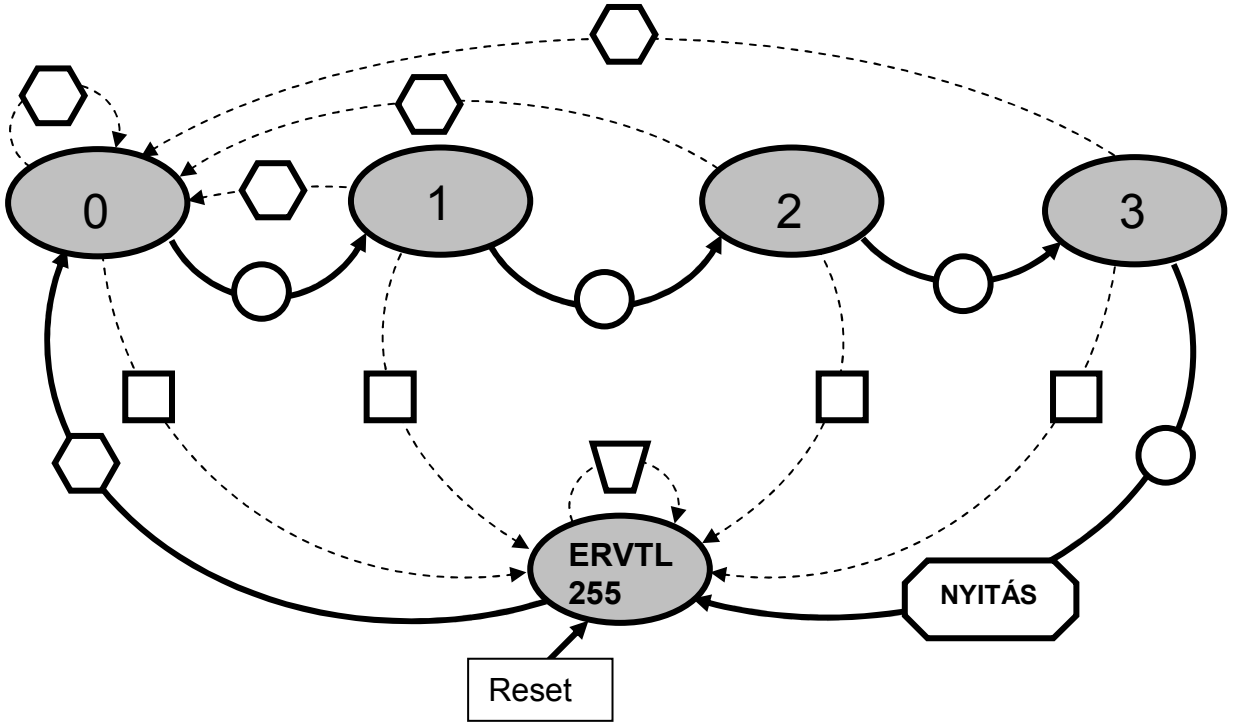
1) ....
2) A kártyalehúzás és a számjegyek leütésének figyelésére
   a) megszakítást használhatunk
   b) folytonos állapotolvasást végezhetünk

- Döntés: A kártyalehúzás és a számjegyek leütésének figyelésére megszakítást használunk.
  (Ezzel a döntéssel elestünk annak a lehetőségétől, hogy a programszámlálót használjuk a program állapotának a tárolására!)

További döntések - 2

- Használjunk a program állapotának tárolására egy állapotváltozót, nevezzük AV-nek.
- Hordozza AV azt az értéket,
  - hogy az utolsó kártyalehúzást követően hány helyes számjegy érkezett
  - és egy másik számértékkel kódoljuk azt, ha nem volt még kártyalehúzás vagy érvénytelen számjegy jött.
- Készítsük el az állapotgráfot!

A beléptető rendszer állapotgráfja



: Az automata lehetséges állapotai. Állapotváltozás: INTERRUPT hatására

: Az elvileg helyes sorrend : Hibás sorrend

Események:

Kártyalehúzás történt.
A kártya kódja tárolva

Billentyűzés történt.
A karakter érvényes

Billentyűzés történt, de
a karakter érvénytelen

Billentyűzés történt.
(akármilyen karakter)

Hogyan tovább?

- Az állapotgráfnak megfelelő véges automatát formálisan is lekódolhatnánk, de helyette inkább gondolkozzunk egy kicsit, mert az megtérül a programozáskor! ;-)
- Az inputnak két fajtája lehetséges: kártyalehúzás vagy számjegy érkezése.
  - A **kártyalehúzás** az adott állapottól függetlenül mindig AV:=0-t eredményez, és tárolni kell a kártya azonosítóját.
  - Csak **számjegy érkezése** esetén érdekes az AV értéke. Ekkor
    - ha az AV értéke ERVTLN, akkor nincs teendőnk
    - különben szükségképpen AV eleme {0, 1, 2, 3}, ekkor
      - ha a beérkezett számjegy megfelelő, akkor az AV értéke eggyel nő;
      - ha nem megfelelő, akkor AV:=ERVTLN lesz.

    ha ezek után AV értéke 4 lett, akkor NYITÁS parancsot adunk és AV:=ERVTLN.

Adatszerkezetek

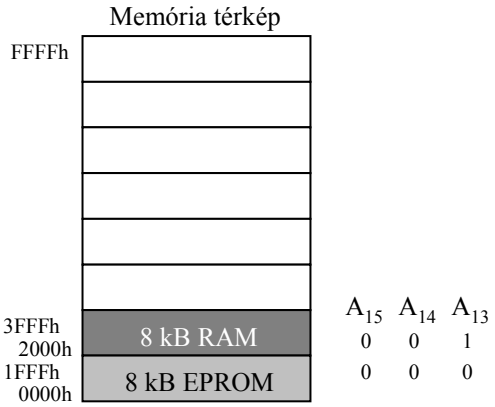
- KODTABLA:
  - 4 db 256 byte-os táblázat egymás után:
    - 0. 256 byte-os táblázat: az első várt számjegy
    - 1. 256 byte-os táblázat: a második várt számjegy
    - 2. 256 byte-os táblázat: a harmadik várt számjegy
    - 3. 256 byte-os táblázat: a negyedik várt számjegy
- AV: egy darab státusz bájt, melynek jelentése:
  - i=0-3: a legutolsó kártyaolvasás után már jött i db számjegy és az jó volt
    (Vegyük észre, hogy itt i=0 éppen úgy kezelhető mint i=1-3!)
  - legyen az értéke 255 minden egyéb esetben (ERVTLN)
- K_AZON: egy bájt, ami az utolsó kártyaazonosító értékét tárolja, de csak AV=0-3 esetén érvényes

Még egy tervezői döntés

- Igazítsuk a KODTABLA kezdetét 256 bájtos laphatárra!
  - Ez egyszerűsíti az elemeinek a címzését.
  - A mindig használható KODTABLA_KEZDOCIME+256*AV+K_AZON cím 16 biten való kiszámítása helyett
  - a KODTABLA AV-edik 256 bájtos lapján keressük a K_AZON-adik számjegyet, akkor elegendő AV-t a KODTABLA kezdőcímének magasabb helyiértékű bájtjához hozzáadni, K_AZON értékét pedig KODTABLA kezdőcímének alsó helyértékű (a laphatárra igazítás miatt 0 értékű) bájtja helyett használni.
  - 16 bites cím:

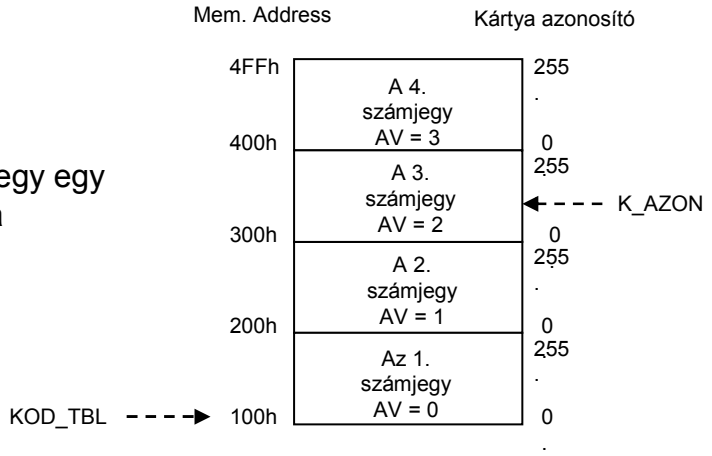
15.. 8	7.. 0
High	Low

Memória térkép



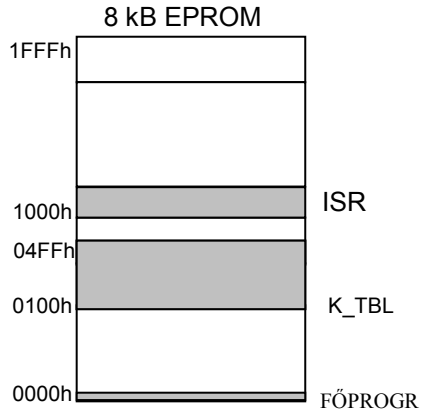
Egy leütött számjegy egy byte-on ábrázolva

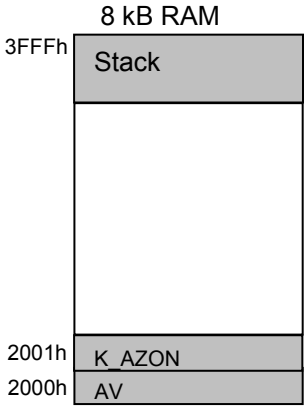
A kódtábla memória térképe:



Az adat címe = KOD_TBL + 256* AV + K_AZON 256 = 100h

Base Addr High byte Low byte



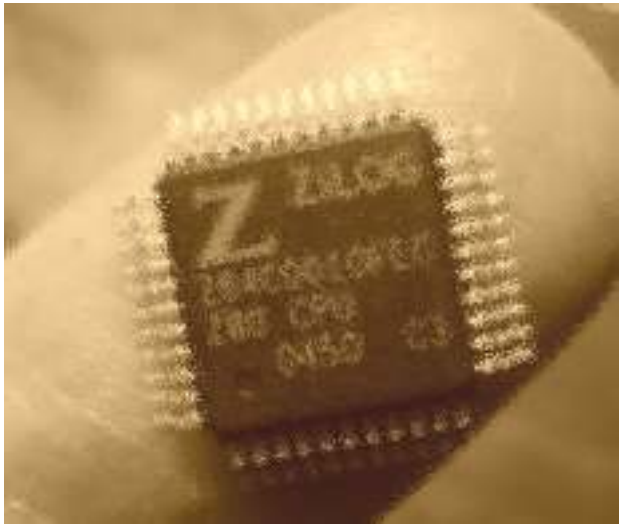


A főprogram működése

- A főprogram
  - beállítja a verem helyét
  - az állapotváltozó értékét: ERVTLN
  - engedélyezi a megszakításokat
  - végtelen ciklusban
    - kiolvassa a zár állapotát
    - megjeleníti a LED-eken.

A megszakítási rutin működése

- Az IT rutin menti a használt regisztereket, majd:
  - Ha kártyalehúzás jött, akkor az azonosítót tárolja és AV:=0;
  - Különben számjegy jött (mert másért nem lett volna IT), tehát:
    - ha AV=255, akkor a számjegyet eldobjuk, állapot változatlan,
    - ellenkező esetben csakis 0-3 állapot lehet, így: ellenőrizzük, hogy jó-e a számjegy,
      - ha nem: AV:=255;
      - ha igen: AV++; amennyiben az állapot 4 lett, akkor: NYITÁS és AV:=255;
- Végül az IT rutin visszatölti a mentett regiszterek értékét, engedélyezi a megszakítást és visszatér.



A HIPOTETIKUS PROCESSZOR UTASÍTÁSKÉSZLETE

Regiszterkiosztás

- A hipotetikus processzornak a következő regiszterei vannak: A, F, B, C, D, E, H, L regiszterek 8 bitesek, és az SP regiszter 16 bites az alábbi értelmezésekkel:
  - A - akkumulátor: kitüntetett regiszter, az aritmetikai és logikai műveletek egyik operandusaként használjuk, és a műveletek eredménye is benne képződik; valamint az IN (perifériáról való bevitel) és OUT (perifériára való kivitel) utasítások implicit operandusa is ez.
  - F - jelzőbitek regisztere: a jelzőbitek közül csak a Z-t használjuk, értéke pontosan akkor 1, ha az utolsó aritmetikai vagy logikai művelet eredménye 0. Az A regiszterrel együtt az AF 16 bites regiszterpárt alkotja, ami verem műveleteknél együtt kezelhető.
  - BC, DE, HL - regiszterpárok: együtt 16 bites regiszterként használhatók, de az egyes regiszterek külön-külön is használhatók.
  - SP - veremmutató

Címzési módok – 1

- Akkumulátor címzés: az egyik operandus és a művelet eredménye az akkumulátor (de nem nevezzük meg). Ezzel találkozunk az aritmetikai és logikai műveleteknél valamint az IN/OUT utasításoknál.

```
AND     B       ; A ← A&B
OUT     38h     ; port38h ←  A
```

- Regisztercímzés: operandusként regisztert vagy regiszterpárt adunk meg.

```
LD      A, B    ; A ← B
```

- Közvetlen adatcímzés (immediate): közvetlenül az utasítás kódja után, az utasítás részeként szerepel az operandus

```
LD      B, 10    ; B ← 10
```

Címzési módok – 2

- Direkt memóriacímzés: az operandus memóriabeli címét adjuk meg zárójelben (azért kell a zárójelpár, hogy a közvetlen adatcímzéstől meg tudjuk különböztetni)

```
LD          (3000h), D      ; MEM[3000h] ← D
```

- Indirekt címzés: a címet tartalmazó regiszterpár zárójelben megadva szerepel (itt is azért kell a zárójelpár, hogy a regisztercímzéstől meg tudjuk különböztetni).

```
LD          A,(HL)          ; A ← MEM[HL]
```

Utasítás készlet, mnemonikok – 1

- LD: adatmozgatás: hova, mit
- PUSH / POP: regiszter vagy regiszterpár értékének mentése a verembe / visszatöltése a veremből
- IN / OUT: a megadott portról bevitel vagy oda kivitel
- AND: bitenkénti logikai ÉS művelet a megadott operandus és az akkumulátor között
- ADD: 8 bites összeadás a megadott operandus és az akkumulátor között
- INC: a megadott regiszter értékének növelése 1-gyel
- CMP: a jelzőbitek állítása az
  "akkumulátor tartalma mínusz a megadott érték"
  művelet eredménye szerint
  (de az akkumulátor tartalma nem változik meg)

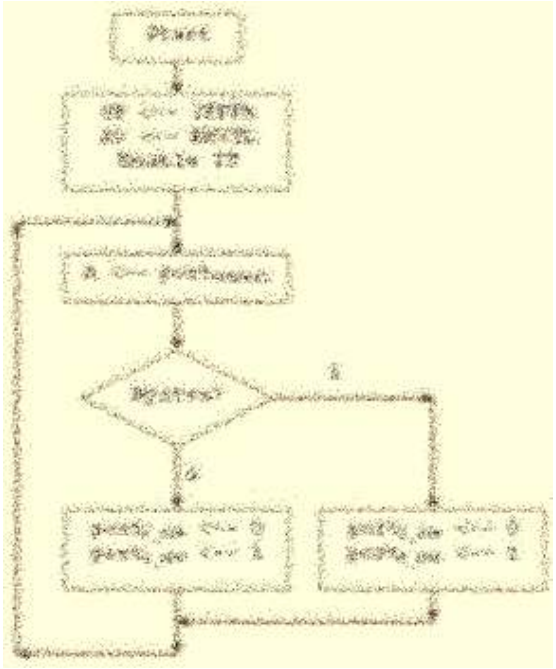
Utasítás készlet, mnemonikok – 2

- JMP: feltétel nélküli ugrás (a megadott címre)
- JZ / JNZ: ugrás, ha az előző aritmetikai vagy logikai művelet eredménye 0 / nem 0
- CALL: szubrutin hívása
- RET: szubrutinból vagy megszakításból való visszatérés
- EI: megszakítások engedélyezése

Fordítói direktívák

- EQU: szimbólumhoz érték rendelése
- ORG (origin): program memóriabeli kezdőcímének megadása
- DB (define byte): adatok elhelyezése a gépi kódba az elhelyezés számláló értéke szerinti helyre
- END: a fordítás befejezése

- További elemek:
  - Címkék: sor elején kezdődnek, kettősponttal zárulnak
  - Megjegyzések: pontosvesszőtől (”;”) az adott sor végéig

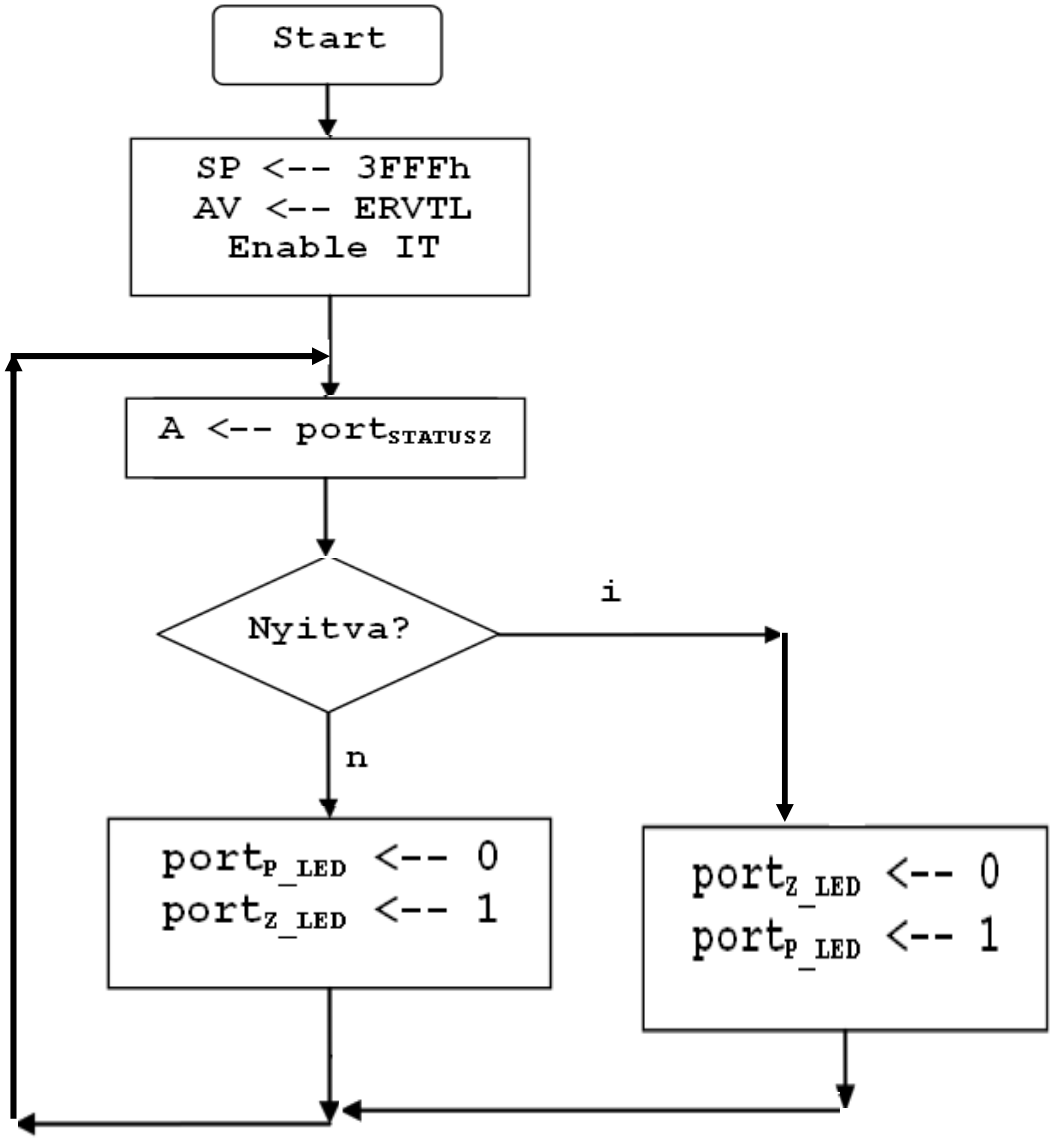


FOLYAMATÁBRÁK RAJZOLÁSA

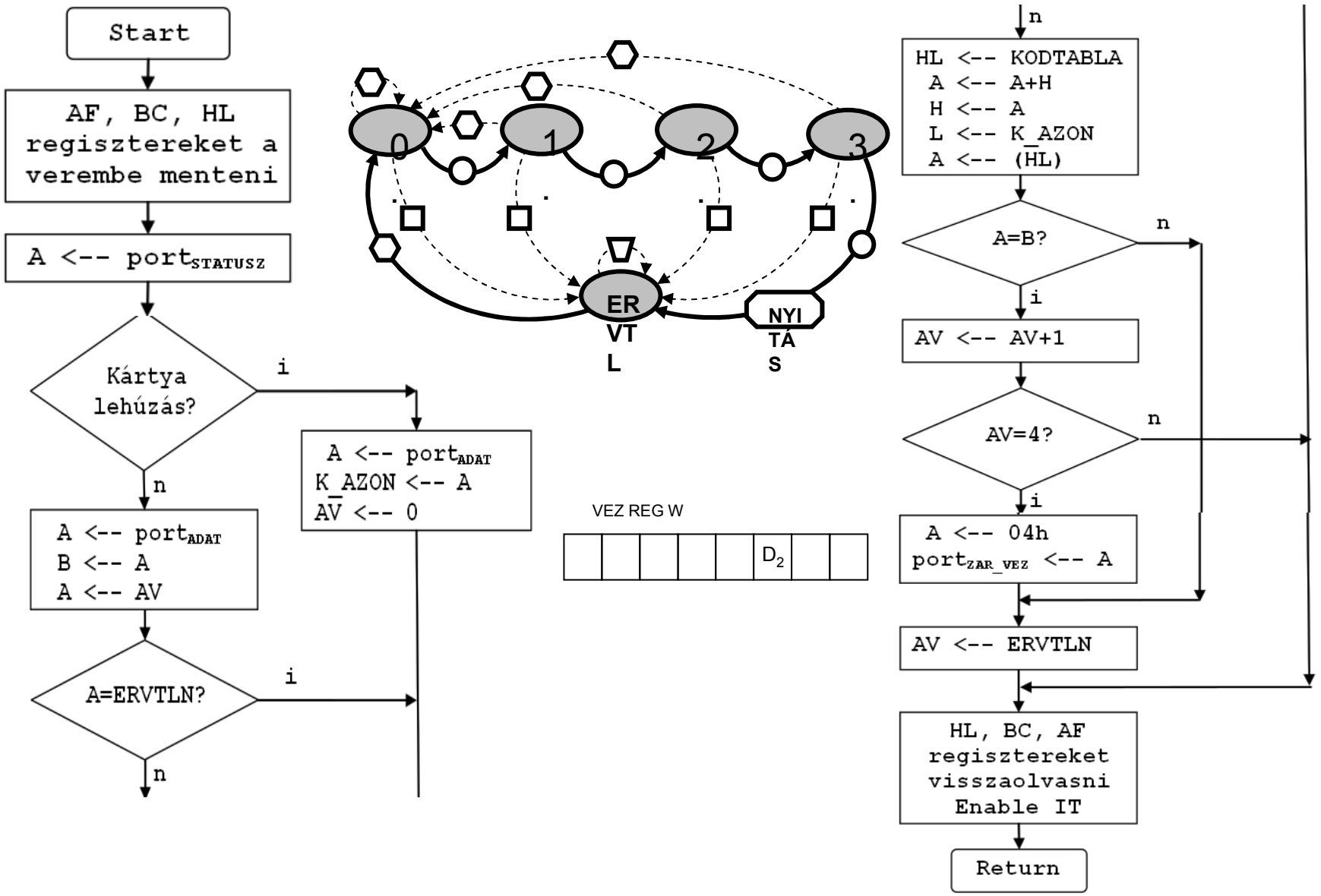
Miért és hogyan használjuk?

- Rövid, egyszerű assembly nyelvű programok (néhányszor tíz sor) megírása előtt általában célszerű (és elegendő is) folyamatábrát készíteni.
- A folyamatábrába olyan lépéseket szoktunk írni, amit vagy közvetlenül egy gépi utasítással vagy legfeljebb 2-3 utasítással el tudunk végezni. Ezeket kifejezhetjük szavakkal, de még jobb, ha formálisan írjuk le őket (mert így egyértelmű); például az adatmozgatásokat regiszterekkel, memória/port címekkel és nyilakkal jelezzük:
  - A <-- (HL): Az A regiszterbe (akkumulátor) betöltjük a HL regiszterpár (értéke) által megcímzett memóriarekesz tartalmát.
  - $port_{5Bh}$ <-- A: Az akkumulátor értékét kiírjuk az 5Bh számú portra.
- A folyamatábra így természetesen processzorfüggő.

A főprogram folyamatábrája



Az IT rutin folyamatábrája



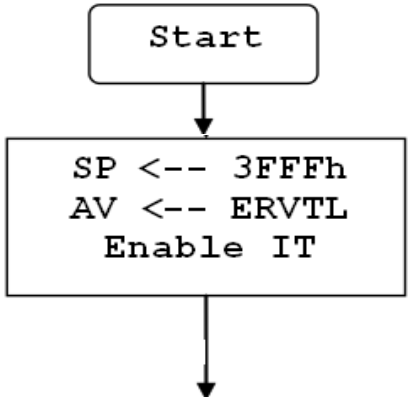
ASSEMBLY NYELVŰ PROGRAM ÍRÁSA

Assembly nyelvű program – 1

```
; hardver címek
Z_LED    EQU     1Ah             ; zöld LED portcíme
P_LED    EQU     1Bh             ; piros LED portcíme
STATUSZ  EQU     2Eh             ; periféria státusz reg. portcíme
ZAR_VEZ  EQU     2Eh             ; periféria zárvez. reg. portcíme
ADAT     EQU     2Fh             ; periféria adat reg. portcíme
;
; egyéb konstansok
HOSSZ    EQU     04              ; belépőkártyákhoz kódjának hossza
KLE      EQU     00000001b       ; maszk kártyalehúzás teszteléséhez
SZJ      EQU     00000010b       ; maszk számjegy teszteléséhez
ZARALL   EQU     00000100b       ; maszk a zár állapot kiolv.-hoz
NYISD    EQU     00000100b       ; a nyitás parancs kódja
ERVTLN   EQU     255             ; az érvénytelen állapot kódja
LAP_MER  EQU     100h            ; 256 byte
;
; változók címei
AV       EQU     2000h           ; állapotváltozót itt tároljuk
K_AZON   EQU     2001h           ; kártyaazonosítót itt tároljuk
```

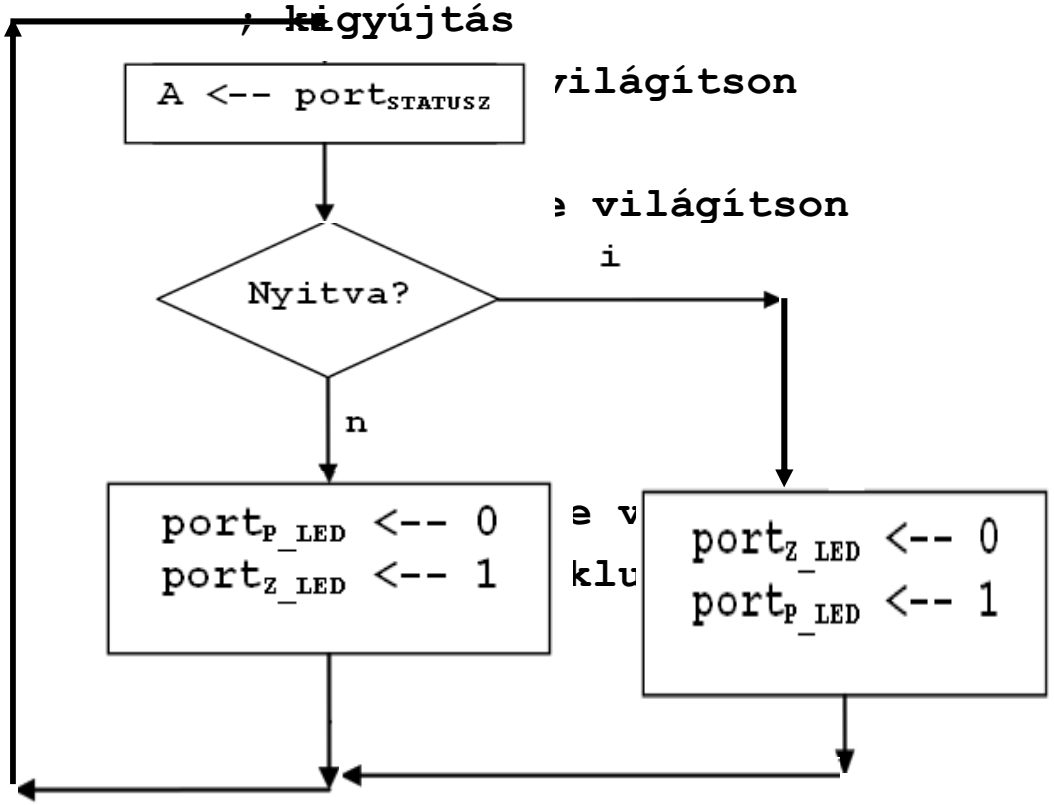
Assembly nyelvű program – 2

```
; A kódtáblát majd a főprogram után az EPROM-ban helyezzük el...
;
; itt kezdődik a főprogram
        ORG     0000h           ; programkód elhelyezése a 0 címtől
        LD      SP,3FFFh        ; veremmutató beáll. a RAM tetejére
        LD      A,ERVTLN        ; az érvénytelen állapot kódja
        LD      (AV),A          ; állapotváltozó beállítása
        EI                      ; megszakítások engedélyezése
```



Assembly nyelvű program – 3

```
; most jön a program főciklusa
FOCIKL: IN      STATUSZ         ; periféria állapotának kiolvasása
        AND     ZARALL          ; zárállapot vizsgálata
        JNZ     nyitva          ; ha a D2 bit 1-es volt: nyitva van
; ha nem ugrott el, akkor tudjuk, hogy zárva van!
        LD      A,0             ; kigyújtás
        OUT     P_LED           rilágítson
        LD      A,1
        OUT     Z_LED           e világítson
        JMP     FOCIKL          i
NYITVA: LD      A,0
        OUT     Z_LED
        LD      A,1
        OUT     P_LED           e v
        JMP     FOCIKL          klu
;
```

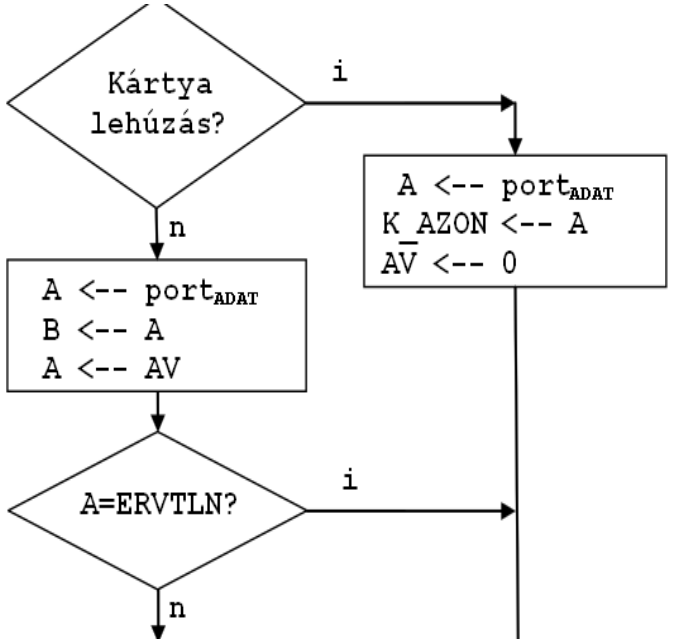


Assembly nyelvű program – 4

```
; A kódtáblát laphatárra helyezzük
        ORG 0100h
KODTABLA: DB    1, 4, 5, 2, 3, 4, 7, 8, 8, 0, 2, 3, 6, 2, 4, 9
        DB      4, 4, 3, 1, 7, 8, 0, 1, 3, 6, 4, 1, 4, 7, 6, 5
        DB      ...
; összesen 64 sor, soronként 16 értékkel,
; azaz 64x16=1024=4x256 db számjegy
;
```

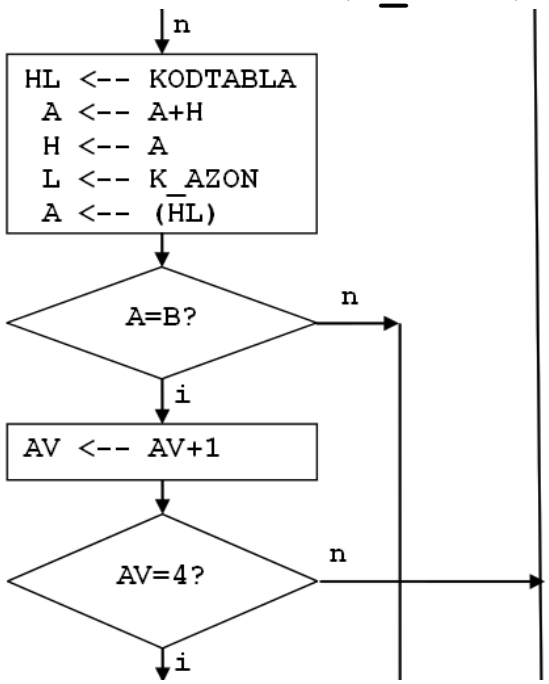
Assembly nyelvű program – 5

```
; most jön a megszakítási rutin
        ORG     1000h           ; az IT rutin elhelyezése
; minden olyan regiszter értékét elmentjük, amit használni fogunk
        PUSH    AF              ; az A regiszter és a flag-ek,
        PUSH    BC              ; a BC regiszterpár,
        PUSH    HL              ; a HL regiszterpár mentése
        IN      STATUSZ         ; mi volt a megszakítás oka?
        AND     KLE
        JNZ     LEHUZ
; biztos, hogy számjegy volt
        IN      ADAT
        LD      B,A
        LD      A,(AV)
        CMP     ERVTLN
        JZ      IT_VEGE
```

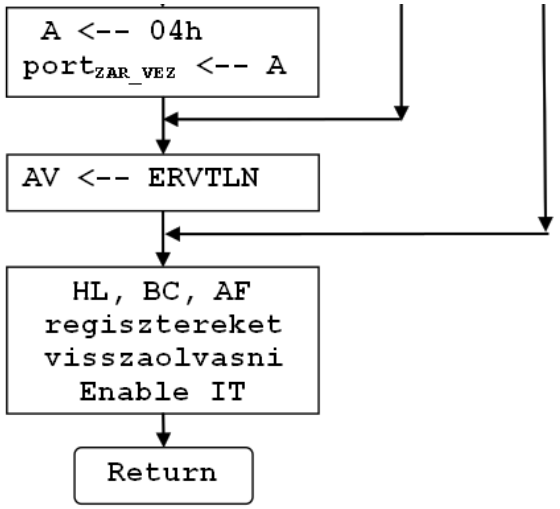


Assembly nyelvű program – 6

```
; az AV értéke szükségképpen a [0,HOSSZ-1] intervallumba esik!
; a várt számjegy a KODTABLA+LAP_MER*AV+K_AZON címen van:
        LD      HL,KODTABLA     ;kódtábla kezdőcíme,(L=0 laphatár)
        ADD     H               ; A:=A+H
        LD      H,A             ; A KODTABLA-ban az (AV). lapon
        LD      L,(K_AZON)      ; keressük a (K_AZON). értéket
        LD      A,(HL)                                  asása
        CMP     B                                       t számjegy
        JNZ     ROSSZJ                                  egy
; a beütött számjegy helyes volt
        LD      A,(AV)
        INC     A
        LD      (AV),A
        CMP     HOSSZ                                   zat hosszát?
        JNZ     IT_VEGE
```



Assembly nyelvű program – 7




```
; ha mar elértük: nyitunk + AV                                    a
NYITAS: LD      A,NYISD                                          a
        OUT     ZARVEZ
ROSSZJ: LD      A,ERVTLN                                         nytelen
        LD      (AV),A
        JMP     IT_VEGE                                          égére
; kártyalehúzás feldolgozása
LEHUZ:  IN      ADAT                                             ak kiolvasása
        LD      (K_AZON),A      ; kártyakód eltárolása
        LD      A,0             ; ez lesz az új állapot
        LD      (AV),A          ; állapotváltozó beállítása
; a regisztereket fordított sorrendben kell visszatölteni!
IT_VEGE: POP    HL
        POP     BC
        POP     AF
        EI
; Az IT engedélyezésének hatása egy utasítást késik!
        RET                     ; visszatérés az IT rutinból
        END                     ; vége a fordításnak
```

ALTERNATÍV MEGOLDÁSOK VIZSGÁLATA

Önálló hallgatói munka – 1

- A hallgatók önállóan gondolják végig:
  - miben különbözne a program, ha nem használnánk megszakítást, hanem helyette állapotolvasással kérdeznénk le, hogy történt-e kártyalehúzás, illetve érkezett-e számjegy!
  - Módosítsák ennek megfelelően a programot önállóan!

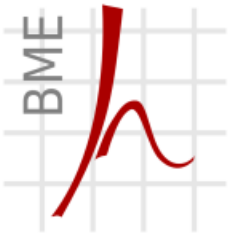
FIGYELEM! Először mindenki készítse el a saját megoldását és csak utána nézze meg a tárgy jegyzetében szereplő minta megoldást!

Önálló hallgatói munka – 2

- Gondolkozzanak el önállóan az alábbi kérdéseken!
  1. A megszakításos és az állapotolvasásos megoldás közül melyiket biztosan nem lehet megvalósítani RAM nélkül? Miért?
  2. Hogyan tudná a másikat megvalósítani RAM nélkül? (Mit, hova helyezne el?)
  3. Hogyan módosítaná a programot, ha a KODTABLA nem illeszkedne laphatárra? (A kritikus számításhoz használjon 8 bites műveleteket, a mnemonikokat értelemszerűen állapítsa meg!)
  4. Hogyan módosulna a fenti számítás akkor, ha a KODTABLA (mint kétdimenziós tömb) indexeit felcserélnénk, azaz a 4db 256 bájt méretű tömb helyett 256 db 4 bájtos tömb lenne
  5. Hogyan valósíthatnánk meg a megalkotott állapotgráf szisztematikus lekódolását, és ez mekkora munkát jelentene!

Kérdések?

KÖSZÖNÖM A FIGYELMET!



Híradástechnikai Tanszék