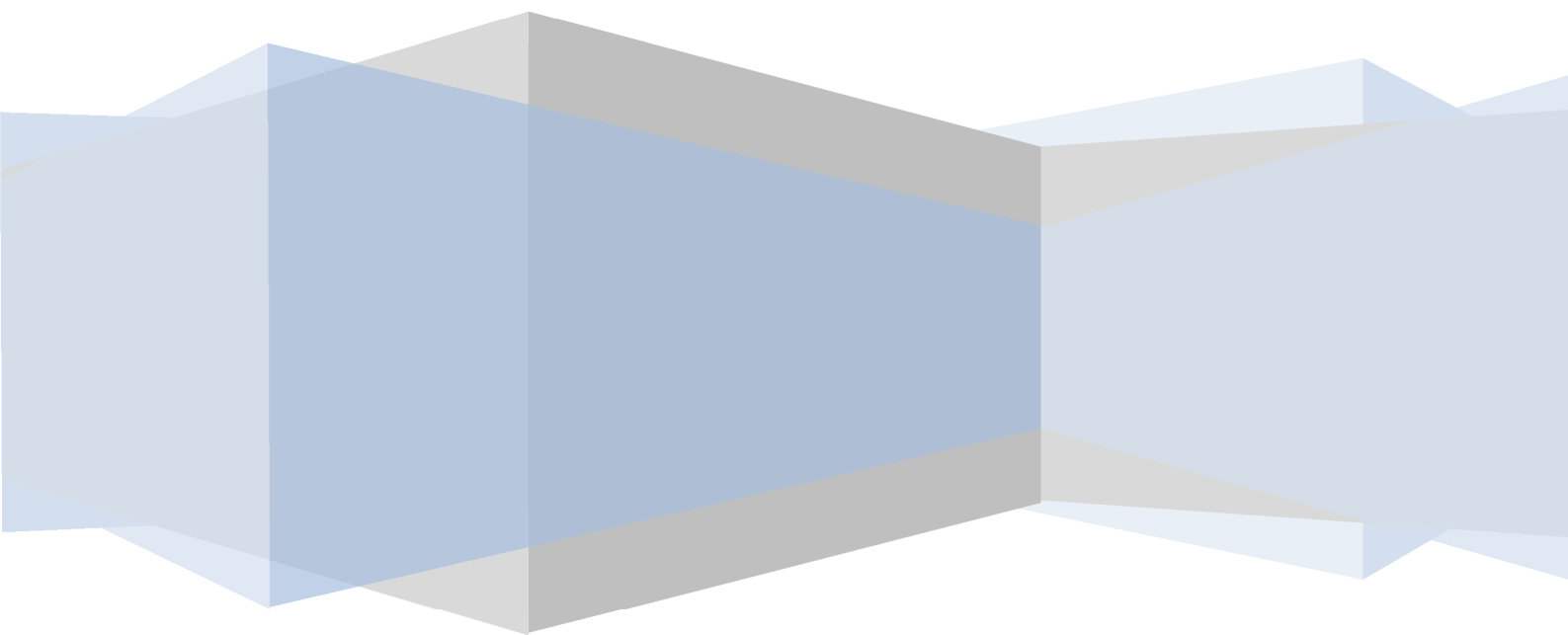


SZÁMÍTÓGÉPES RENDSZER TERVEZÉSE

ESETTANULMÁNY: BELÉPTETŐ RENDSZER



ESETTANULMÁNY: BELÉPTETŐ RENDSZER TERVEZÉSE

A tárgy első részében tanult ismeretek részbeni összefoglalására tervezzük meg egy egyszerű mikroprocesszoros rendszer hardverét, és írjuk meg működtető szoftverét!

A feladat tárgyköre és célja:

- tartalmaz memória- és perifériaillesztést, megszakításkezelést, egyszerű algoritmusok programozását
- felméri, hogy a hallgatók mennyire képesek alkalmazni a megtanult alapelveket és módszereket
- bemutatja, egy egyszerű, de az eddigieknél összetettebb szoftver megírásának módszertanát
- kísérletet tesz arra, hogy a hallgatók egy egyszerű hipotetikus (8085'-re hasonlító) processzor hipotetikus (Z80-ra hasonlító) és először látott assembly nyelvén kódolják le a kidolgozott algoritmusokat.

A hardver elemei

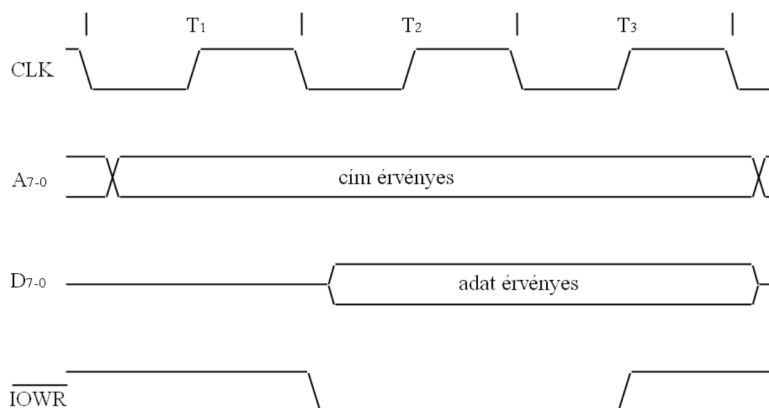
Mikroprocesszor

Adott egy hipotetikus mikroprocesszor a következő illesztési felülettel:

- A15-0 címbusz; megcímezhető 64 kB memória, de csak 256 I/O port (az A7-0 címbitekkel címezhető)
- D7-0 adatbusz
- /MEMRD, /MEMWR, /IOR, /IOWR: 0 aktív kimenő vezérlő jelek
- INT: felfutó élre érzékeny IT bemenet

Kezdetben a CPU a 0000h címről indul. Megszakításkor egy CALL 1000h utasítást hajt végre, regisztereket nem ment, a további megszakításokat letiltja.

Az első feladat (hardver tervezése) megoldásához szüksége lesz rá, ezért megadjuk még a processzor perifériára történő írásának az idődiagramját. Vegyük észre, hogy az /IOWR jel megszűnésekor az adat még kellő ideig stabil!



1. ábra: Perifériára való írás idődiagramja

A mikroprocesszor utasításaival majd a 2. feladat (szoftver tervezés és program írása) keretében ismerkedünk meg.

Beléptető periféria

Adott egy beléptető periféria, ami tartalmaz:

- egy kártyaolvasót: a belépőkártyák azonosítója 0-255 egész szám
- egy 0-9 számjegyeket tartalmazó billentyűzetet (más gomb, pl. törlés nincs)
- egy relé vezérlő egységet (nyitott vagy zárt állásba vezérelhető a zár nyelve; az ajtó zárt állásban is becsukódik, de csak nyitott állásban nyitható ki)
- a processzor felé való illesztési felületet

A periféria illesztési felülete a CPU felé

A periféria be és kimenetei a következők:

- D7-0: kétirányú adatbusz
- /CS, RD és WR: 0 aktív bemenő vezérlő jelek,
- (/C)/D (Code=0/Data=1): választó bemenet, hatása:
 - (/C)/D=0 érték esetén: D7-0 olvasáskor státusz, íráskor zárvezérlés
 - (/C)/D=1 érték esetén: az adatregiszterből az utolsó eseményhez tartozó számérték (utoljára lehúzott kártya azonosítója vagy utoljára megnyomott számjegy) olvasható ki, az adatregiszter írása hatástalan.
- INTR: alacsony logikai szinttel jelző esemény kimenet

A beléptető periféria (/C)/D választó bemenetét az A0 címbitre kötve tehát az eszköz a rendszerben 2 db periféria címet foglal el: az alacsonyabb címen a vezérlő/státusz regiszter, a magasabb címen az adatregiszter található.

A periféria működése

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. (Időzítéssel nem foglalkozunk!) A feladat megoldásához felhasználjuk a beléptető periféria alábbi funkcióit:

- Amikor a kártyát lehúzzák, a beléptető INTR lába alacsony szintűre vált, és a státuszregiszteréből olvasva a D0 bit 1-es értékű lesz (akárhányszor is kiolvasható): ezek mindaddig fennállnak, amíg az adatregiszterből ki nem olvasták a kártya azonosítóját. Ezután adatregiszterből kiolvasható (csak egyszer!) a lehúzott kártya azonosítója, utána rögtön INTR magasra vált, és a státuszregiszteréből olvasva a D0 bit értéke 0 lesz.
- Amikor egy számjegyet beütöttek, a beléptető INTR lába alacsony szintűre vált, és a státuszregiszteréből olvasva a D1 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 D1 bit értéke 0 lesz.
- A vezérlő regiszter D2 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 D2 bitjéből mindig kiolvasható a zár állása: 0: zárva, 1: nyitva

További rendelkezésre álló elemek

A feladat megoldásához adott még: 8kB ROM, 8kB RAM, 3/8 dekóder, 3db 8 bites címkomparátor, 1 piros és 1 zöld LED nyitó irányban 2V feszültségeséssel és 5-15mA

áramigénnyel, 2db 250 Ohm-os ellenállás, 2db D tároló. A D tárolók (a tanult módon) az órajel felfutó élere tárolják el a D bemenet logikai értékét.

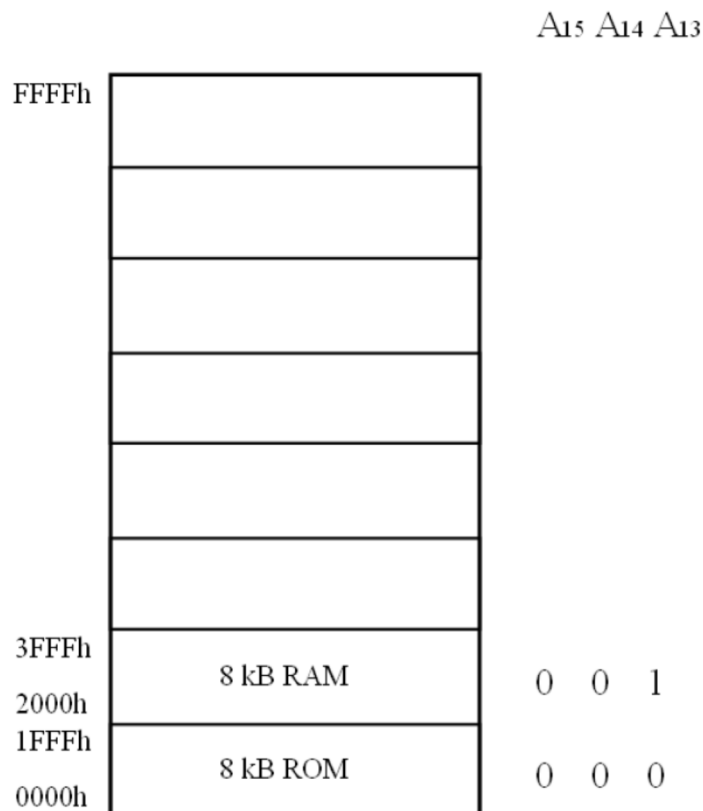
1. Feladat: a hardver megtervezése

Illesszünk a hipotetikus mikroprocesszorhoz 8kB EPROM-ot a 0000h címre, 8kB RAM-ot a 2000h címre, a beléptető perifériát a 2Eh báziscímre (így 2Eh/2Fh címeken érhető el), valamint egy zöld és egy piros LED-et az 1Ah és az 1Bh portcímekre (D tárolók és ellenállások segítségével) úgy, hogy az adatbusz D0 bitjével lehessen az állapotukat vezérelni!

A megoldás menete

Memóriák illesztése

Készítsünk memóriatérképet! A memóriatérkép mellett megadtuk az A15-13 címbitek értékét is; ebből (is) világosan látható, hogy egy 3/8-as dekóderrel az EPROM és a RAM illesztése könnyen megoldható: a 0000h-1FFFh címeken elérhető EPROM esetén A15-13=000, a 2000h-3FFFh címeken elérhető RAM esetén A15-13=001, tehát az 3/8-as dekóder /O0, illetve /O1 kimenetére kell őket kötnünk. Mindkét memória mérete 8kB, így megcímezésükhöz 13 bitre (A12-0) van szükség. Az adatbusz bekötésénél ügyeljünk arra, hogy az EPROM-ba nem tudunk beleírni!



2. ábra: Memóriatérkép

Beléptető periféria illesztése

A beléptető perifériát címkomparátorral a 2Eh báziscímre illesztjük: a címkomparátor P7-1 bemenetére a címbusz A7-1 bitjeit kötjük, de az A0 címbit helyett P0-ra fixen logikai 0-t (L) kötünk, Q7-0 bemenetére pedig az 1Eh értéket kapuzzuk. Így a két perifériacím (2Eh és 2Fh)

bármelyikének megszólítása esetén a komparátor kimenete aktív lesz. A címkomparátor /G engedélyező bemenetét fixen logikai 0-ra (L) kötjük. (Meghajthatnánk az /IOR és /IOWR jelekkel egy ÉS kapun keresztül, ami 0 aktív logikában VAGY kapuként funkcionál, de időzíteni megfontolásból most nem ezt tesszük.) A címkomparátor 0 aktív /(P=Q) kimenetével engedélyezzük a beléptető perifériát annak CS bemenetén. Az A0 címbitet rákötjük a beléptető

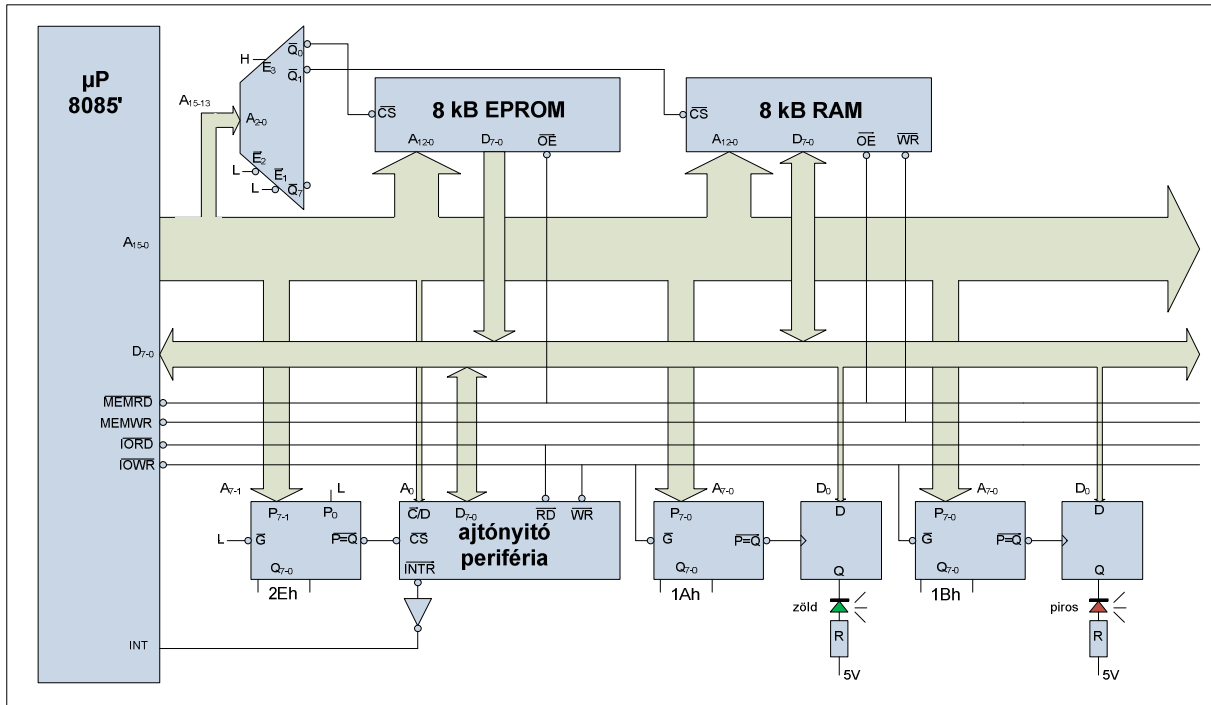
periféria (/C)/D bemenetére; az /IOR, /IOWR, és D7-0 jeleket pedig értelemszerűen a beléptető periféria /RD, /WR, és D7-0 lábaira kötjük. A periféria /INTR kimenetét inverzre keresztül kötjük a CPU felfutó élre érzékeny INT lábára.

LED-ek meghajtása

A LED-ek meghajtásához fontoljuk meg a következőket! TTL alkatrészekről lévén szó, a D tároló kimenetének áram terhelhetősége logikai 1 értéknél bizonyosan 1mA alatt van, logikai 0-nál viszont 20mA környékén, tehát közvetlenül a D tároló kimenetéről csak logikai 0 esetén tudjuk biztosítani a LED-ek meghajtásához a szükséges áramot. Ha az egy LED-en nyitó irányban eső feszültség 2V, és figyelembe vesszük, hogy a D tároló kimenetén nem pontosan 0V a feszültség, valamint az 5V névleges értékű logikai 1 sem pontosan 5V, akkor a 250 Ohm-os áramkorlátozó ellenálláson még kb. 2V feszültségeséssel számolhatunk, ami 8mA áramot biztosít; ez a LED-ek számára megfelelő. Tehát a LED-eket a D tárolók kimenetéről 250-ohm ellenálláson keresztül logikai 1 ("H") szintre kötjük. (Vigyázzunk, hogy a LED-ek rajzjelének állása a pozitív áramirányt kövesse a H értéktől a D tároló kimentet felé!) Ne felejtsük el, hogy ezzel a megoldással azt is eldöntöttük, hogy a LED-ek kigyújtásához 0-t kell a D tárolókba írunk, kioltásához pedig 1-et!

D tárolók illesztése

A D tárolókat az 1Ah és az 1Bh portcímekekre illesztjük címkomparátorokkal. (A címkomparátorok P7-0 bemenetére a címbusz A7-0 bitjeit kötjük, Q7-0 bemenetére pedig az 1Ah illetve 1Bh értékeket kapuzzuk.) A címkomparátorok engedélyezéséhez a /G engedélyező bemenetükre a CPU /IOWR kimenetét kötjük, mivel a tárolókat csak írunk kell. A címkomparátorok 0 aktív /(P=Q) kimenetét a D tároló felfutó élre érzékeny órajel bemenetére kötjük; így az /IOWR megszűnésekor a felfutó él hatására a kiválasztott D tároló eltárolja a D bemenetére kötött adatbusz D0 bitjének ekkor még stabilan tartott értékét (lásd 1. ábra).



3. ábra: Logikai kapcsolási rajz

2. Feladat: a szoftver megtervezése és megírása assembly nyelven

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, valamint 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 tervezés menete

Megfontolások, tervezői döntések

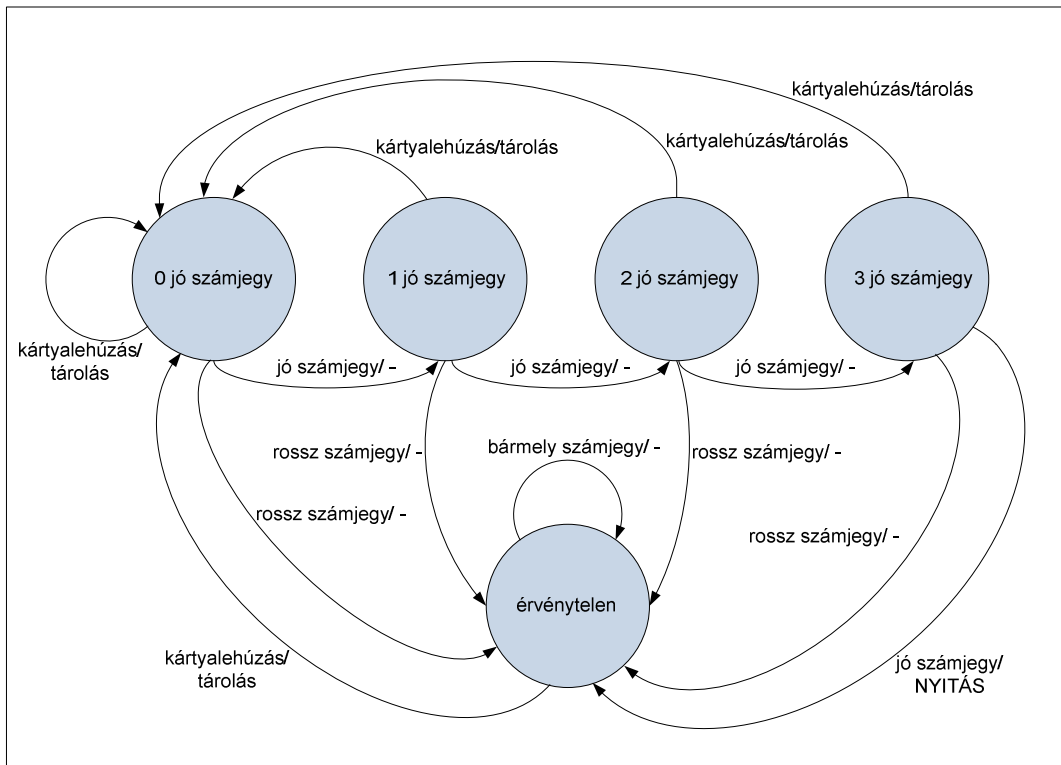
Vegyük észre, hogy mivel a zárásról semmiféle értesítést sem kapunk, az 1. követelményt csak úgy tudjuk kielégíteni, ha folytonosan kiolvassuk a zár állapotát és (szükség esetén) beállítjuk a LED-eket. Az már tervezői döntés, hogy ebben a ciklusban folytonosan állítjuk-e LED-ek állapotát vagy csak változás esetén, hasonlóan az is, hogy a ciklusban csak a nyitottról zártra állítással foglalkozunk és (kártyalehúzással és helyes kóddal kiváltott) nyitáskor végezzük el a LED-ek állítását vagy inkább minden LED állítást a ciklusban végzünk. Most válasszuk azt, hogy ebben a ciklusban végzünk el minden LED állítást és állapotváltozástól függetlenül mindig beállítjuk a LED-eket a zár beolvasott állásának megfelelően!

Éljünk azzal az előfeltevéssel is, hogy a kártyalehúzás és a számjegyleütés események közül egyszerre csak az egyik következik be! (A rendszer működése kellően gyors a felhasználóhoz képest.)

Beléptető megszakítással

A kártyalehúzást és a számjegyek leütését lekérdezni is képesek vagyunk, és megszakítást is használhatunk. Tegyük most az utóbbit!

Mivel a megszakítások használatát választottuk, elesünk annak a lehetőségétől, hogy a programszámlálót használjuk a program állapotának a tárolására, ezért kézenfekvő, hogy használjunk egy állapotváltozót, nevezzük AV-nek. Hordozza AV azt az értéket, hogy az utolsó kártyalehúzást követően hány helyes számjegy érkezett (lehetséges értékek 0-tól 4-ig) és egy másik számértékkel kódoljuk azt, ha nem volt még kártyalehúzás vagy érvénytelen számjegy jött. Könnyen belátható, hogy csak egy ilyen "érvénytelen" állapotra van szükség, ugyanis csak egyfajta esetben lehet folytatás: ha kártyalehúzás jön. Az is nyilvánvaló, hogy az AV=4 esetet sem szükséges megkülönböztetnünk az érvénytelentől, mert ha egyszer már elvégeztük a nyitást, a továbbiakban ez az állapot nem különbözik az érvénytelentől (ismét kártyalehúzásra kell várnunk). Vizsgáljuk meg az alábbi állapotgráfot!



4. ábra: Állapotgráf

A fenti állapotgráfnak megfelelő véges automatát formálisan is leködölhatnánk, de helyette inkább gondolkozzunk egy kicsit, mert az megtérül a programozáskor! ;-) Az inputnak két fajtája lehetséges: kártyalehúzás vagy számjegy érkezése.

- Látható, hogy 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 amennyiben AV értéke 0, 1 vagy 2 és a korábban tárolt kártyaazonosító és az adott AV mellett a beérkezett számjegy megfelelő, akkor az AV értéke eggyel nő; ha nem megfelelő, akkor $AV:=\text{érvénytelen}$ lesz. Amennyiben $AV=3$, akkor is az előbbihez hasonlóan kell eljárunk, annyi különbséggel, hogy ha megfelelő számjegy jött, akkor NYITÁS parancsot adunk és $AV:=\text{érvénytelen}$ lesz. A programban kezeljük ezért inkább együtt a 0, 1, 2 és 3 állapotokat, és csak a végén vizsgáljuk meg, hogy vajon AV értéke 4 lett-e! Természetesen amennyiben AV értéke érvénytelen, akkor bármilyen számjegy jött is, az AV értéke változatlan marad, tehát a programnak ezzel az esettel nem is kell foglalkoznia!

Adatszerkezetek

- KODTABLA: 4 db 256 byte-os táblázat egymás után: az i. táblázat j. karaktere a j. kártyaazonosítóhoz tartozó i. számjegy; $j=0-255$, $i=0-3$
- 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

- K_AZON: egy bájt, ami az utolsó kártyaazonosító értékét tárolja, de csak AV=0-3 esetén érvényes

További tervezői döntés a következő apró trükk: 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 így, ha 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 helyiértékű (a laphatárra igazítás miatt 0 értékű) bájtja helyett használni.

Az egyes programrészek működése nagy vonalakban

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

Az IT rutin menti a használt regisztereket, majd:

- Ha kártyalehúzás jött, akkor az azonosítót tárolja és AV:=0;
- Különböző 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 (csak a periférián) és AV:=255;

Végül az IT rutin visszatölti a mentett regiszterek értékét, engedélyezi a megszakítást és visszatér.

A program működése folyamatábrával kifejezve

Rövid, egyszerű assembly nyelvű programok 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 címeikkel és nyilakkal jelezzük:

- A <-- (HL): Az A regiszterbe (akkumulátor) betöltjük a HL regiszterpár (értéke) által megcímezett memóriarekesz tartalmát.
- port5Bh <-- A: Az akkumulátor értékét kiírjuk az 5Bh számú portra.

A folyamatábra így természetesen processzorfüggő. A folyamatábra elkészítéséhez tehát előbb megadjuk az hipotetikus processzor regiszterkiosztását.

A hipotetikus processzor és az assembly nyelv és fordító használata

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 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

- Akkumulátor címzés: az egyik operandus és a művelet eredménye implicit módon 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 adatszámítás (immediate): közvetlenül az utasítás kódja után, az utasítás részeként szerepel az operandus

```
LD     B, 10             ; B ← 10
```

- Direkt memóriacímzés: az operandus memóriabeli címét adjuk meg zárójelben (azért kell a zárójelpár, hogy a közvetlen adatszámítástól 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]
```

Felhasznált mnemonikok

(A használt mnemonikok a 8085 és a Z80 assembly nyelvére emlékeztetnek.)

- 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)
- 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

Felhasznált fordítói direktívák

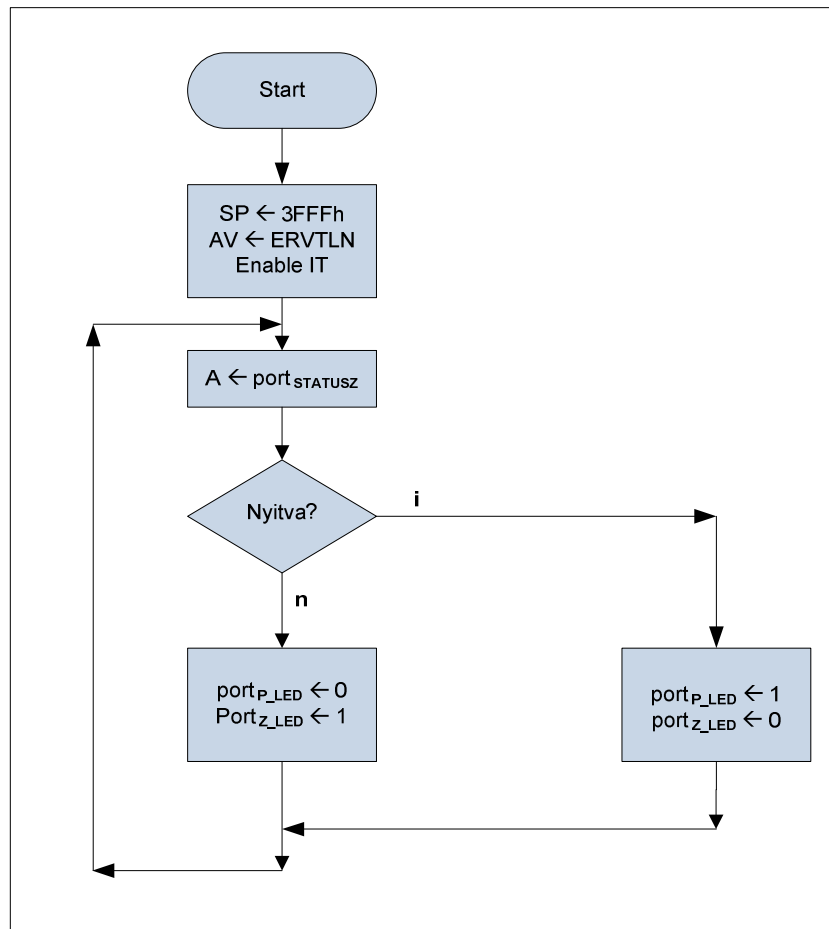
- EQU: szimbólumhoz érték rendelése
- ORG (origin): program memóriabeli kezdőcímének megadása
- \$: az utasítás elhelyezési számláló aktuális értéke
- DB (define byte): adatok elhelyezése a gépi kódba az elhelyezés számláló értéke szerinti helyre
- END: a fordítás befejezése

További elemek:

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

A főprogram

Az alábbi folyamatábra a főprogram működését írja le.

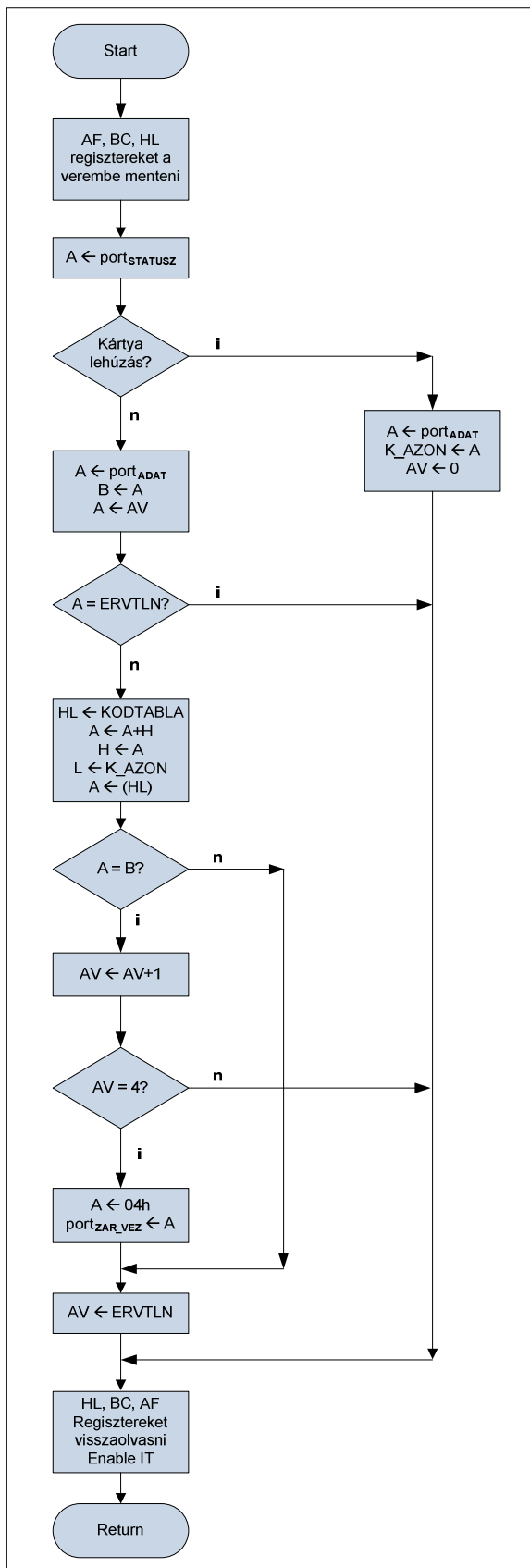


5. ábra: A főprogram működése

Amint a folyamatábrán is látható, a program végtelen ciklusban fut. Kártyalehúzás, valamint számjegy leütése esetén megszakítás keletkezik, ekkor a főprogram működését megszakítva a megszakítási rutin kerül meghívásra.

A megszakítási rutin

A megszakításokat kiszolgáló eljárás működését az alábbi folyamatábra írja le.



6. ábra: A megszakítási rutin működése

A beléptető program kódja

Főprogram

```

; hardver címek
Z_LED      EQU    1Ah      ; zöld LED portcíme
P_LED      EQU    1Bh      ; piros LED portcíme
STATUSZ    EQU    2Eh      ; beléptető periféria státusz regiszterének portcíme
ZAR_VEZ    EQU    2Eh      ; beléptető periféria zárvezérlő regiszterének portcíme
ADAT       EQU    2Fh      ; beléptető periféria adatregiszterének portcíme
;
; egyéb konstansok
HOSSZ      EQU    04       ; a belépőkártyákhoz tartozó kód hossza
KLE        EQU    0000001b ; maszk a kártyalehúzás teszteléséhez
SZJ        EQU    0000010b ; maszk a számjegy beütésének teszteléséhez
ZARALL     EQU    00000100b ; maszk a zár állapotának kiolvasásához
NYISD      EQU    00000100b ; a nyitás parancs kódja
ERVTLN     EQU    255      ; az érvénytelen állapot kódja
LAP_MER    EQU    100h     ; 256 byte, a belepőkártyák kódtáblájának laphatárra helyezéséhez
;
; változók címei
AV         EQU    2000h     ; az állapotváltozót itt taroljuk (a RAM-ban)
K_AZON     EQU    2001h     ; a kártyaazonosítót itt tároljuk
; A kódtáblát majd a főprogram után az EPROM-ban helyezzük el az értékek definiálásával!
;
; itt kezdődik a főprogram
                ORG    0000h     ; a programkód elhelyezése a 0000h kezdőcímtől
                LDI    SP,3FFFh   ; veremmutató beállítása a RAM tetejére
                LDI    A,ERVTLN   ; az érvénytelen állapot kódja
                LD     AV,A       ; állapotváltozó beállítása
                EI                ; megszakítások engedélyezése
;
; most jön a program főciklusa
FOCIKL:        IN     STATUSZ     ; a beléptető periféria állapotának kiolvasása
                ANI    ZARALL     ; zárállapot vizsgálata
                JNZ    NYITVA     ; ha a D2 bit 1-es volt: nyitva van
; ha nem ugrott el, akkor tudjuk, hogy zárva van!
                LDI    A,0        ; kigyújtás
                OUT    P_LED      ; piros LED világítson
                LDI    A,1        ; kioltás
                OUT    Z_LED      ; zöld LED ne világítson
                JMP    FOCIKL     ; végtelen ciklus
NYITVA:        LDI    A,0        ; kigyújtás
                OUT    Z_LED      ; zöld LED világítson
                LDI    A,1        ; kioltás
                OUT    P_LED      ; piros LED ne világítson
                JMP    FOCIKL     ; végtelen ciklus
;
; A kódtáblát laphatárra helyezzük az alábbi apró trükkal:
; (a fenti programkód rövidsége miatt nyilván "ORG 0100h"-val ekvivalens)
                ORG    ($+LAP_MER-1) AND NOT (LAP_MER-1)
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
;

```

Megszakítási rutin

; most jön a megszakítási rutin			
	ORG	1000h	; az IT rutin elhelyezése az 1000h címtől
; minden olyan regiszter értékét elmentjük, amit használni fogunk			
	PUSH	AF	; az A regiszter és a flag-ek értékének mentése a verembe
	PUSH	BC	; a BC regiszterpár értékének mentése a verembe
	PUSH	HL	; a HL regiszterpár értékének mentése a verembe
	IN	STATUSZ	; mi volt a megszakítás oka?
	ANI	KLE	; kártyalehúzás jött?
	JNZ	LEHUZ	; igen
; biztos, hogy számjegy volt, nem mert kártyalehúzás!			
	IN	ADAT	; számjegy kiolvasása (kötelező, mert különben az IT fennmarad)
	LD	B,A	; a számjegy mentése
	LD	A,AV	; állapotváltozó betöltése
	CPI	ERVTLN	;
	JZ	IT_VEGE	; ha érvénytelen, akkor nincs további teendők
; az AV értéke szükségképpen a [0,HOSSZ-1] intervallumba esik (lásd később: HOSSZ tesztelése)			
; a várt számjegy a KODTABLA+LAP_MER*AV+K_AZON címen van, készítjük el a címet!			
	LD	HL,KODTABLA	; kódtábla kezdőcíme, a laphatárra helyezés miatt L érteke 0
	ADD	H	; A:=A+H, túlcordulás tudjuk, hogy nem lesz!
	LD	H,A	; A KODTABLA-ban az (AV). lapon keressük a ...
	LD	L,K_AZON	; ... (K_AZON) értéket
	LD	A,(HL)	; a várt számjegy kiolvasása indirekt címzéssel
	CMP	B	; fent B-be mentettük a beérkezett számjegyet
	JNZ	ROSSZJ	; hibás a beütött számjegy
; a beütött számjegy helyes volt, ezért AV 1-gyel nő:			
	LD	A,AV	
	INC	A	
	LD	AV,A	
	CPI	HOSSZ	; elértük-e már a kódsorozat hosszát?
	JNZ	IT_VEGE	; még nem
; ha már elértük: nyitunk + AV:=ERVTLN			
NYITAS:	LDI	A,NYISD	; nyitás parancs kódja
	OUT	ZARVEZ	; parancs kiadása
ROSSZJ:	LDI	A, ERVTLN	; az új állapot: érvénytelen
	LD	AV,A	; állapot eltárolása
	JMP	IT_VEGE	; ugrás az IT rutin végére
; kártyalehúzás feldolgozása			
LEHUZ:	IN	ADAT	; belépőkártya kódjának kiolvasása
	LD	K_AZON,A	; kártyakód eltárolása
	LDI	A,0	; ez lesz az új állapot
	LD	AV,A	; állapotváltozó beállítása
; Az IT rutin végén 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		; ez a direktíva jelzi, hogy vége a fordításnak

Önálló feladat: Beléptető állapotolvasással

A hallgatók önállóan gondolják végig, hogy 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! Javasolom, hogy először mindenki készítse el a saját megoldását és csak utána olvassa el az alábbi minta megoldást!

Az előbbi minta megoldásban az IT rutin már tartalmazza mind a két esemény (kártyalehúzás, számjegy érkezése) feldolgozását; készítsünk ezekből szubrutinokat! A főprogram végtelen ciklusában mentsük el a beolvasott állapotot a B regiszterbe (ez egy utasítás beszúrását jelenti), majd a jelzőbitek beállítása után a B regiszterből az állapotot visszatöltve vizsgáljuk meg, hogy fennáll-e a kártyalehúzás illetve a számjegy érkezése, amelyik igen, ahhoz hívjuk meg a megfelelő szubrutint.

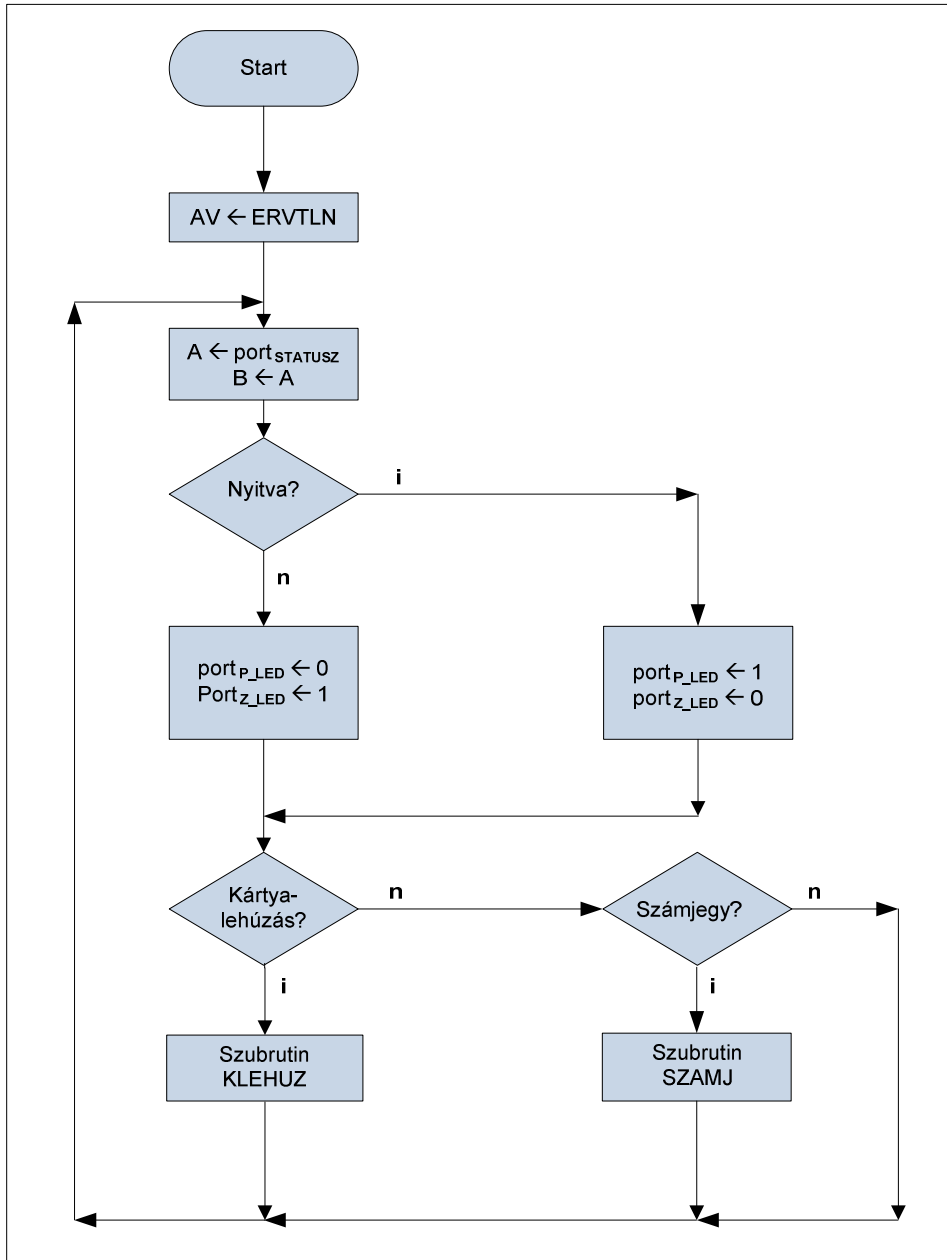
Megjegyzések:

- Egyáltalán nem okoz gondot, hogy a LED-ek állítása megelőzi a kártyalehúzás illetve a számjegy érkezése események vizsgálatát, hiszen ha a zár állapota esetleg változik, a következő ciklusban a LED-ek úgymint követni fogják.
- Természetesen az állapotolvasást akár többször is elvégezhetnénk, az állapot elmentése csak egy lehetséges megoldás.
- Továbbra is élünk azzal az előfeltevéssel, hogy a kártyalehúzás és a számjegy leütése események közül egyszerre csak az egyik következhet be.
- Egy kicsit bonyolultabb feladatnál már érdemes új folyamatábrát rajzolni. Most, mivel a kártyalehúzás és a számjegy leütése események feldolgozását külön szubrutinokba szerveztük.

A módosított beléptető program

Módosított főprogram

; hardver címek			
Z_LED	EQU	1Ah	; zöld LED portcíme
P_LED	EQU	1Bh	; piros LED portcíme
STATUSZ	EQU	2Eh	; beléptető periféria státusz regiszterének portcíme
ZAR_VEZ	EQU	2Eh	; beléptető periféria zárvezérlő regiszterének portcíme
ADAT	EQU	2Fh	; beléptető periféria adatregiszterének portcíme
;			
; egyéb konstansok			
HOSSZ	EQU	04	; a belépőkártyákhoz tartozó kód hossza
KLE	EQU	00000001b	; maszk a kártyalehúzás teszteléséhez
SZJ	EQU	00000010b	; maszk a számjegy beütésének teszteléséhez
ZARALL	EQU	00000100b	; maszk a zár állapotának kiolvasásához
NYISD	EQU	00000100b	; a nyitás parancs kódja
ERVTLN	EQU	255	; az érvénytelen állapot kódja
LAP_MER	EQU	100h	; 256 byte, a belépőkártyák kódtáblájának laphatárra helyezéséhez
;			
; változók címei			
AV	EQU	2000h	; az állapotváltozót itt taroljuk (a RAM-ban)
K_AZON	EQU	2001h	; a kártyaazonosítót itt tároljuk
; A kódtáblát majd a programrészek után az EPROM-ban helyezük el az értékek definiálásával!			
;			
; itt kezdődik a főprogram			
	ORG	0000h	; a programkód elhelyezése a 0000h kezdőcímtől
	LDI	SP,3FFFh	; veremmutató beállítása a RAM tetejére
	LDI	A,ERVTLN	; az érvénytelen állapot kódja
	LD	AV,A	; állapotváltozó beállítása
; a megszakítások engedélyezésére most nincs szükség			
;			
; most jön a program főciklusa			
FOCIKL:	IN	STATUSZ	; a beléptető periféria állapotának kiolvasása
	MOV	B,A	; a periféria állapotának elmentése
	ANI	ZARALL	; zárállapot vizsgálata
	JNZ	nyitva	; ha a D2 bit 1-es volt: nyitva van
; ha nem ugrott el, akkor tudjuk, hogy zárva van!			
	LDI	A,0	; kigyújtás
	OUT	P_LED	; piros LED világítson
	LDI	A,1	; kioltás
	OUT	Z_LED	; zöld LED ne világítson
	JMP	TOV1	; további vizsgálatokra van szükség...
NYITVA:	LDI	A,0	; kigyújtás
	OUT	Z_LED	; zöld LED világítson
	LDI	A,1	; kioltás
	OUT	P_LED	; piros LED ne világítson
TOV1:	MOV	A,B	; periféria elmentett állapotának visszatöltése
	ANI	KLE	; kártyalehúzás jött?
	JZ	TOV2	; ha nem: további vizsgálat...
	CALL	KLEHUZ	; igen: meghívjuk a szubrutint
	JMP	FOCIKL	; ekkor már számjegy nem jöhetett...
TOV2:	MOV	A,B	; periféria elmentett állapotának visszatöltése
	ANI	SZJ	; számjegy jött?
	JZ	FOCIKL	; ha nem: folytatjuk a főciklust
	CALL	SZAMJ	; igen: meghívjuk a szubrutint
	JMP	FOCIKL	; folytatjuk a főciklust
;			

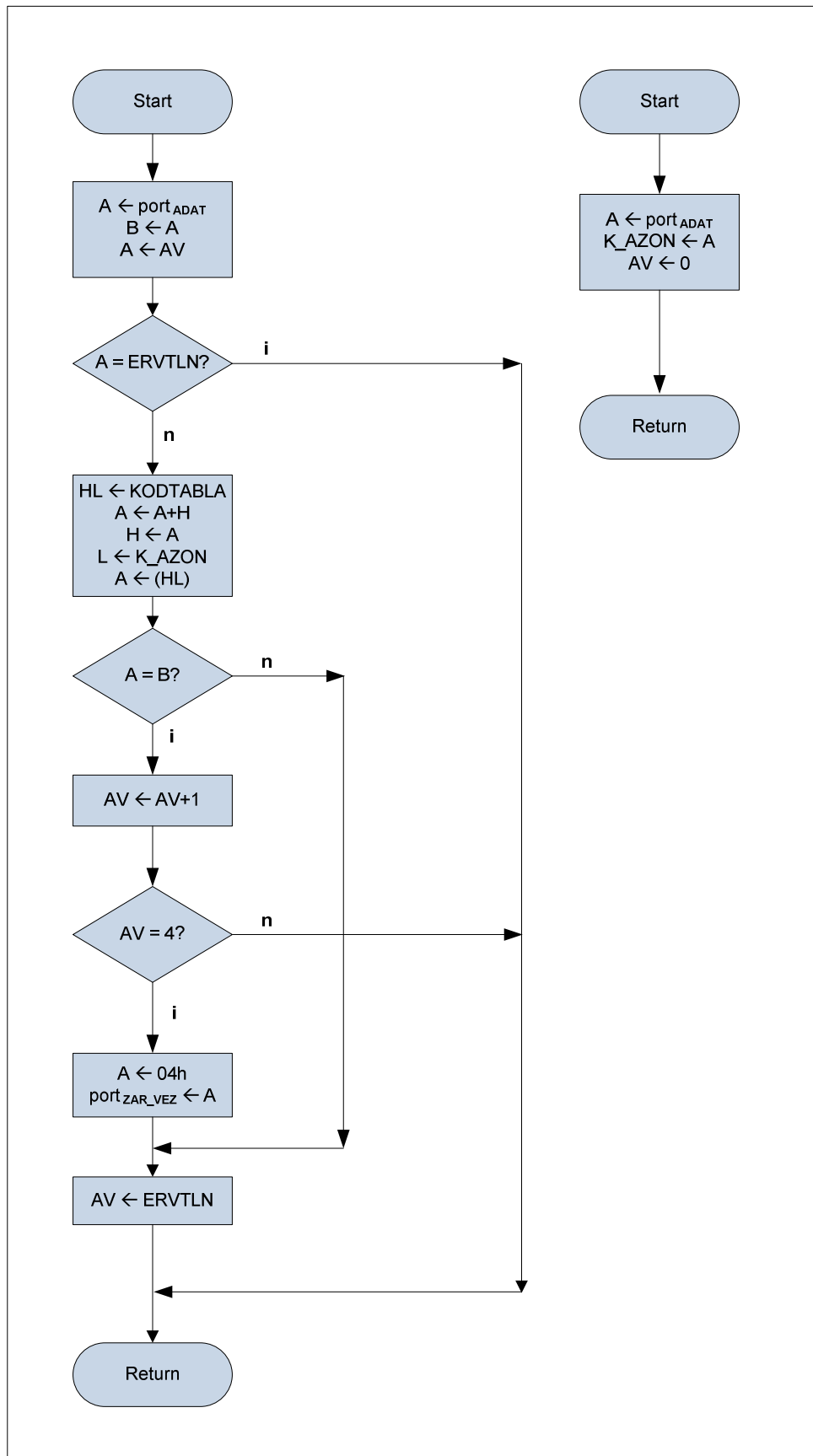


Módosított periféria kezelés - szubrutinok

```

; most jönnek a szubrutinok
;
; számjegy érkezésének feldolgozása
SZAMJ:      IN      ADAT          ; számjegy kiolvasása
            LD      B,A          ; a számjegy mentése
            LD      A,AV        ; állapotváltozó betöltése
            CPI     ERVTLN      ;
            JZ      SZ_VEG      ; ha érvénytelen, akkor nincs további teendők
; az AV értéke szükségképpen a [0,HOSSZ-1] intervallumba esik (lásd később: HOSSZ tesztelése)
; a várt számjegy a KODTABLA+LAP_MER*AV+K_AZON címen van, készítsük el a címet!
            LD      HL,KODTABLA  ; kódtábla kezdőcíme, a laphatárra helyezés miatt L érteke 0
            ADD     H           ; A:=A+H, túlcsordulás tudjuk, hogy nem lesz!
            LD      H,A         ; A KODTABLA-ban az (AV). lapon keressük a ...
            LD      L,K_AZON    ; ... (K_AZON) értéket
            LD      A,(HL)      ; a várt számjegy kiolvasása indirekt címzéssel
            CMP     B           ; fent B-be mentettük a beérkezett számjegyet
            JNZ     ROSSZJ      ; hibás a beütött számjegy
; a beütött számjegy helyes volt, ezért AV 1-gyel nő:
            LD      A,AV
            INC     A
            LD      AV,A
            CPI     HOSSZ      ; elértük-e már a kódsorozat hosszát?
            JNZ     SZ_VEGE    ; még nem
; ha mar elértük: nyitunk + AV:=ERVTLN
NYITAS:     LDI     A,NYISD     ; nyitás parancs kódja
            OUT    ZARVEZ      ; parancs kiadása
ROSSZJ:     LDI     A, ERVTLN   ; az új állapot: érvénytelen
            LD     AV,A        ; állapot eltárolása
SZ_VEG:     RET              ; visszatérés a szubrutinból
; kártyalehúzás feldolgozása
KLEHUZ:     IN      ADAT          ; belépőkártya kódjának kiolvasása
            LD      K_AZON,A     ; kártyakód eltárolása
            LDI     A,0          ; ez lesz az új állapot
            LD      AV,A         ; állapotváltozó beállítása
            RET              ; visszatérés a szubrutinból
;
; A kódtáblát laphatárra helyezzük az alábbi apró trükkel:
ORG ($+LAP_MER-1) AND NOT (LAP_MER-1)
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
;
            END              ; ez a direktíva jelzi, hogy vége a fordításnak

```



További kérdések önálló feldolgozásra

1. Hogyan módosítaná a hardvert, ha a 2000h címre nem 8 kB, hanem 16 kB RAM-ot kellene illesztenie?
2. Milyen megoldást választana, ha a feladat úgy szólna, hogy a LED-eknek akkor kell világítani, ha vezérlésükhöz használt perifériacím 0. bitjére utoljára logikai 1 értéket írtunk és nem világítani, ha 0 értéket írtunk?
3. Hogyan tudná egyszerűsíteni a hardvert, ha előre tudjuk, hogy a piros és a zöld LED állapota mindig egymással ellentétes?
4. A megszakításos és az állapotolvasásos megoldás közül melyiket biztosan nem lehet megvalósítani RAM nélkül? Miért?
5. Hogyan tudná a másikat megvalósítani RAM nélkül? (Mit, hova helyezne el?)
6. Hogyan módosítaná a programot, ha a KODTABLA nem illeszkedne laphatárra? (A kritikus számításához használjon bites műveleteket, a mnemonikokat értelemszerűen állapítsa meg!)
7. 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? (Legelőször a j=0 kártyaazonosítóhoz tartozó 0., 1., 2. és 3 számjegy, aztán a j=1-hez tartozó 4 számjegy, ... végül a j=255-höz tartozó 4 számjegy szerepelne benne.)