Digitális Technika 2 házi feladat

Dudás Márton

1.
Illessszen 8085-ös mikroprocesszor alapú sínre 2764 típusú EPROM és 5516 típusú RAM memóriákat úgy, hogy az alábbi címtartományokat fedjék le:

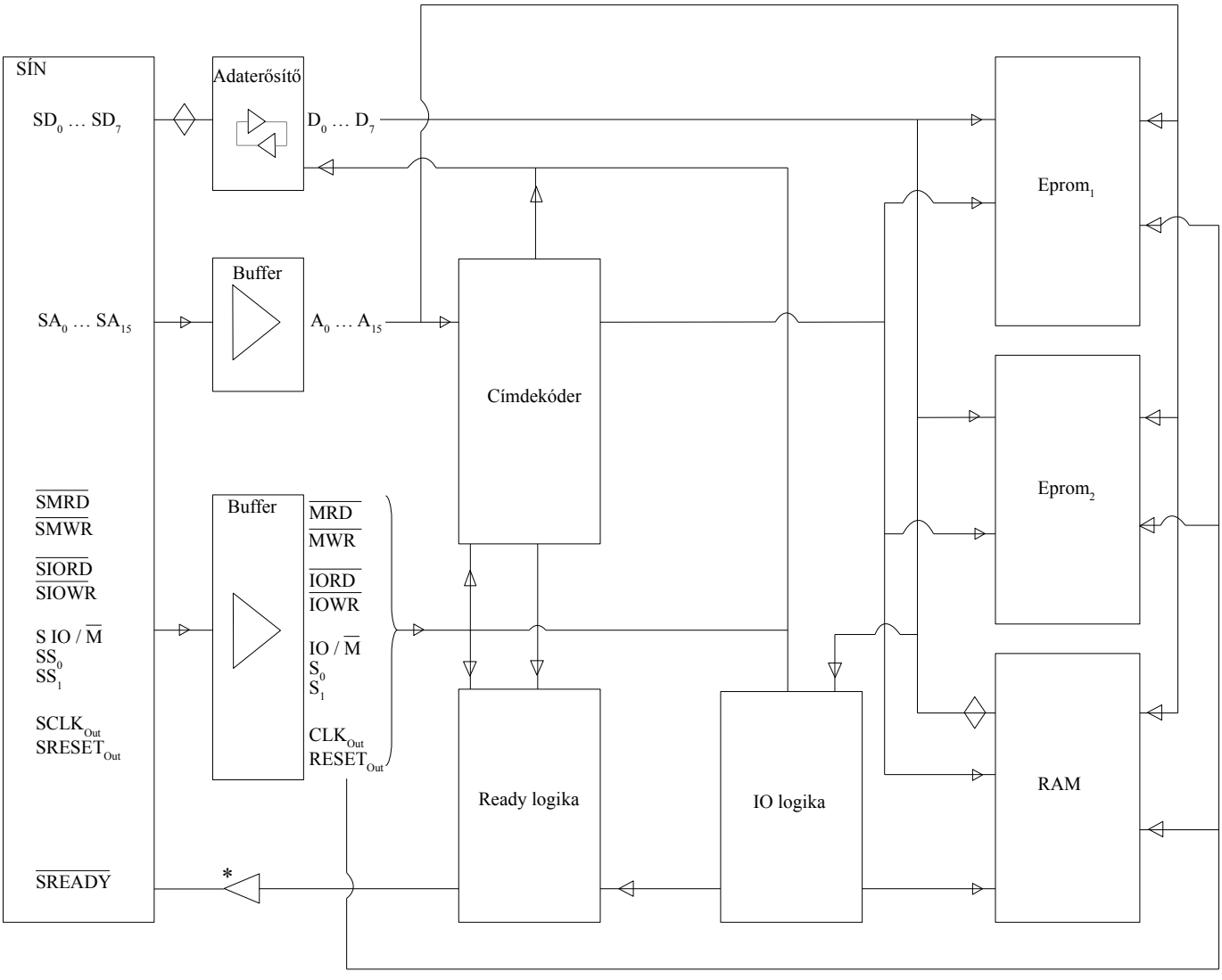
1. 0000h – 1FFFh EPROM
2. 8000h – 97FFh EPROM
3. 9800h – 9FFFh RAM

A megoldáshoz 1 darab 74LS138-as dekóder áramkört használjon minimális kiegészítő hálózattal.
A 07h IO címre írt adattal a RAM írásvédettsége változtatható legyen (ha a kiírt adat 1, akkor a RAM írásvédett, ha 0, akkor nem). Gondoskodjon róla, hogy RESET után a RAM ne legyen írásvédett.

A sínen rendelekzésre álló jelek:
$SA_0 ... SA_{15}$, $SD_0 \dots SD_7$, $\overline{SMRD}$, $\overline{SMWR}$, $\overline{SIORD}$, $\overline{SIOWR}$, SIO/ $\overline{M}$, $\overline{SREADY}$, SS_0, SS_1, $SCLK_{Out}$, $SRESET_{Out}$

---

a. Rajzolja fel a memóriamodul blokkvázlatát. (Figyeljen a jelek konzisztens elnevezésére!)



b. Rajzolja fel a memóriamodulok címtérképét és a címdekóder egységet.

Címtérkép:

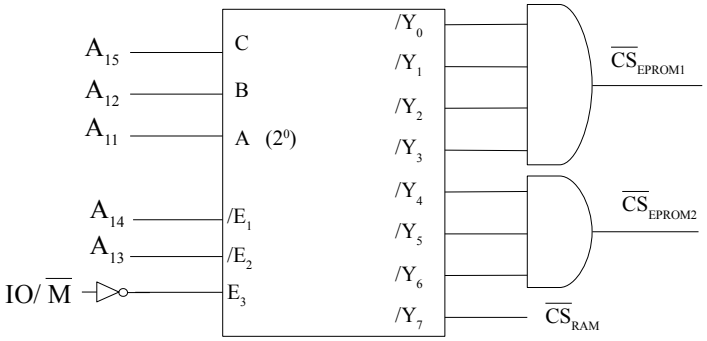
Címtartomány:	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	
9800 - 9FFF	1	0	0	1	1	RAM
9000 - 97FF	1	0	0	1	0	$EPROM_2$
8800 - 8FFF	1	0	0	0	1	
8000 - 87FF	1	0	0	0	0	
…						
1800 - 1FFF	0	0	0	1	1	$EPROM_1$
1000 - 17FF	0	0	0	1	0	
0800 - 0FFF	0	0	0	0	1	
0000 - 07FF	0	0	0	0	0	

A teljes memóriatartomány 64k. Amennyiben csak a legfelső vezetéket vennénk figyelembe, ezt a tartományt két darab, egyenként 32k-s szeletre bonthatnánk.
A memóriák esetén jelenleg 2k bitenként számolunk, ahhoz, hogy a 64k-t 2k-s szeletekre bonthassuk, 5 címvezetéket kell figyelembe vennünk. Azokban a feladatokban, ahol 1k-s szeletekre kell bontani a címtartományt, a felső 6 címvezetéket szükséges vizsgálni. Ha elég volna 4k-s szeleketre osztani a teljes tartományt, akkor csak a felső 4 címvezetéket kellene vizsgálnunk.

A feladatban szereplő EPROM chipek 8kbyte-os példányok. Jól látható, hogy a második EPROM nincs teljesen kihasználva.

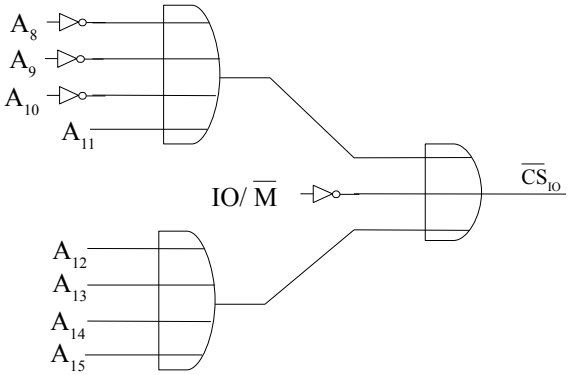
A címdekóder egység:

A memóriák címdekódere:

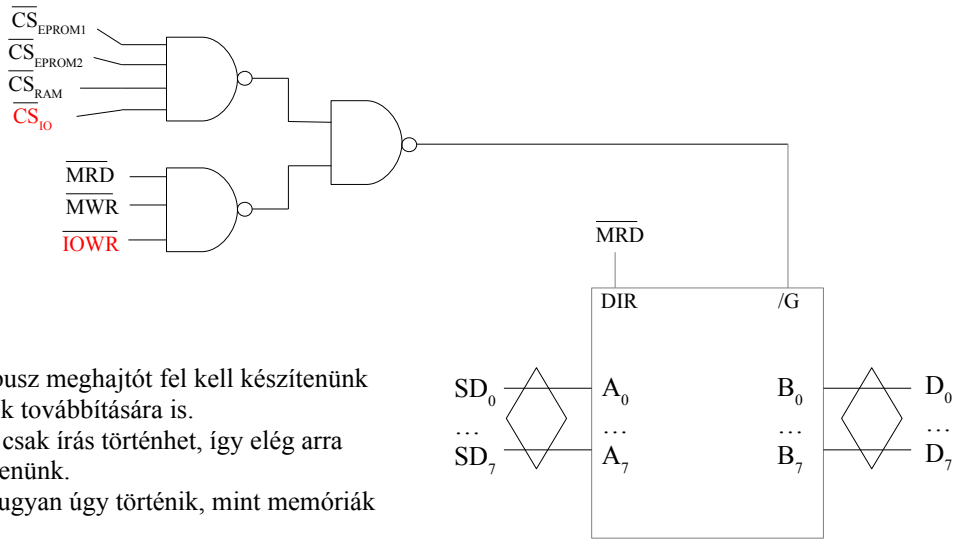


Az IO egység címdekódere:

Az IO egység címdekódere a 07h címet dekódolja.
Az IO címek az alsó és felső 8 címvezetéken duplikálva jelennek meg.
Amikor a számunkra lényeges cím jelenik meg, akkor az OR kapuk bemeneteire 0-k jutnak, így kimenetük 0-ra vált.
Az IO/ M jel segít szétválasztani az IO és memóriacímeket. Amikor IO cím van, akkor értéke 1, így negálás után pont az elvárt működést eredményezi.



c. Rajzolja meg az adatbusz meghajtó áramkör-vezérlő logikáját.

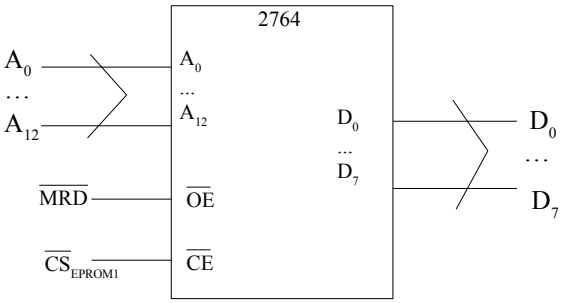


Az adatbusz meghajtót fel kell készítenünk IO adatok továbbítására is.
Jelenleg csak írás történhet, így elég arra felkészítenünk.
Ez pont ugyan úgy történik, mint memóriák esetén.

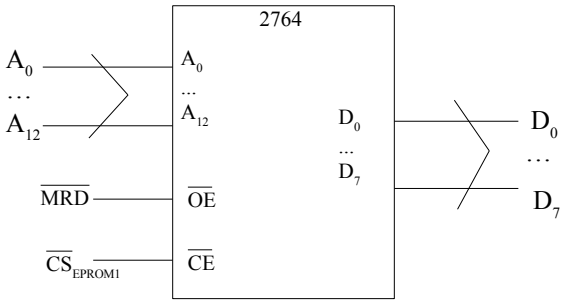
A DIR bemenetre viszont elég a $\overline{MRD}$ jelet kötni, ugyanis ez IO írás esetén is jó irányba állítja az erősítőt.
IO olvasás esetén már nem volna jó ez a megoldás.

d. Adja meg a memória-áramkörök bekötését!

EPROM$_1$



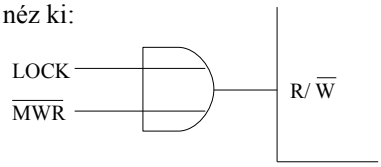
EPROM$_2$



Figyeljünk a címvezetékek számára és a vezérlő bemenetek helyes elnevezésére és bekötésére.

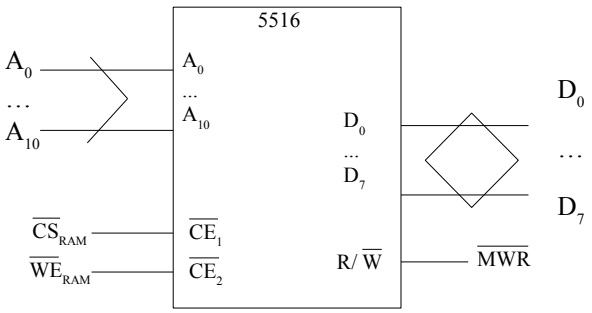
A RAM $\overline{WE}$ jelét az IO egységnél tervezzük meg.
Ez szolgál majd a RAM írásvédettségének elérésére.

Erre van egy másik módszer is, mely a következő módon néz ki:



Ekkor a RAM, ha írásvédett (ekkor legyen a LOCK jel 1-es értékű), nem tud írás műveletet végrehajtani, mivel a LOCK jel az OR kapu kimenetét fixen 1-en tartja.
Használhatjuk ezt az egyszerű megoldást is.

RAM



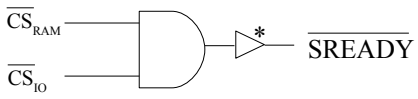
e. Rajzolja fel a READY logikát a következő paraméterek figyelembevételével:
a RAM memóriák READY logikája 0 WAIT állapotot,
az EPROM memóriák READY logikája kizárólag olvasásra 0 WAIT állapotot
iktasson közbe a műveletvégzés közben!

Itt fontos megjegyezni, hogy az IIT oldalán elérhető egy remek dokumentum, mely mindenféle READY logika megtervezésében segít bennünket.

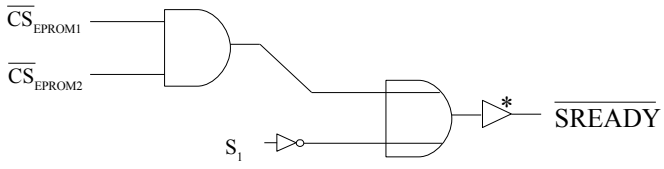
A 0 wait ebben a feladatban meglehetősen egyszerűen megvalósítható.
Az EPROM-ok olvasásra kért WAIT-je az egyedüli trükkös rész, ehhez figyelembe kell vennünk az S_1 státuszjelző bitet. Ready logikáról bővebben egyéb dokumentumokban lehet olvasni.

Ne feledkezzünk meg arról, hogy az IO egység is ad readyt!

RAM és IO egység READY logikája:



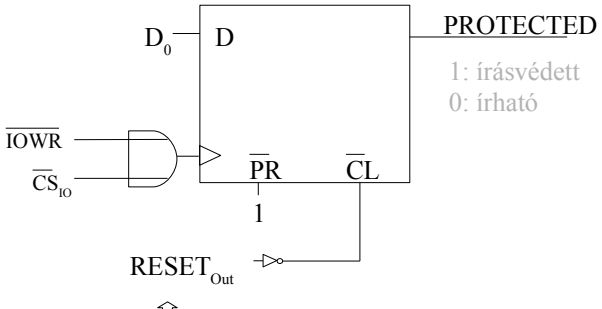
EPROM-ok READY logikája:



f. Tervezze meg a feladatban kért I/O egységet (dekódoló, flip-flop)!

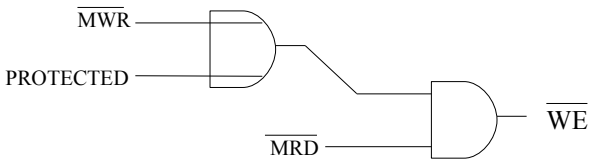
Az IO címet már dekódoltuk, ezt a jelet $\overline{CS}_{IO}$-nak neveztük eddig.
Az a feladatunk, hogy a D_0-s adatbitet beolvassuk, és eltároljuk, amikor a megadott IO címre írás történik.
A tárolást egy flip-flop fogja megvalósítani. A beolvasásnál figyeljünk arra, hogy mikor olvassuk be az adatot. Használjuk a segédlet időidagramjait. Az adatot akkor kell beolvasnunk, amikor az $\overline{IOWR}$ jel eltűnik, mert ekkor biztosan stabil. Amikor az $\overline{IOWR}$ jel megjelenik, az adatvezeték éppen tranziensben lehet, így rossz adatot olvashatnánk be.

Az adatot beolvasó és tároló flip-flop:



⇧
Resetre a RAM biztosan írható lesz.

A memória írásvédettségét megvalósító hálózat:



Amennyiben a PROTECTED jel 1-es, a RAM nem lesz engedélyezve írás művelet esetén.

2. Készítse el a következő assembly szubrutint, amellyel a RAM memória tesztelhető.

a. Írjon KITOLT szubrutint, amely a DE regiszterpárban egy kezdőcímet, a BC regiszterpárban egy hossz értéket kap, az így meghatározott memóriablokkot kitölti úgy, hogy minden byte a saját címe alsó 8 és felső 8 bitjének kizáró vagy kapcsolatát negálva tartalmazza.

```
;KITOLT szubrutin:
;BC és DE regiszterpárokban vesz át értékeket.
;Nem rontja el egyetlen regiszter értékét sem.
;RAM memóriát ír.

KITOLT:
PUSH PSW        ;Mentés a STACK-re
PUSH B
PUSH D
                ; Ellenőrizzük, hogy nem 0 darab memóriarekeszt kell-e írnunk.
MOV A, B        ; B-t vigyük az akkuba, mert csak azzal tudunk műveletet végezni.
ORA C           ; B + C = 0 akkor és csak akkor, ha B és C is 0. Így ellenőrizhetjük, hogy egy
                ; regiszterpárban 0 érték van-e.
JZ EXIT         ; Ha 0 hosszú memóriaterületet kell ellenőriznünk, kilépünk.

CIKL:           ; A kitöltést végző ciklus.
MOV A,E         ; Alsó 8 bit mozgatása az akkuba.
XRA D           ; Az alsó és felső 8 bit kizáró vagy kapcsolatát így megoldottuk.
CMA             ; Negálás.
STAX D          ; A DE regiszterpár által mutatott memóriaterületre írjuk az adatot.
INX D           ; A címet növeljük egyel, akár a pointereket szoktuk c-ben.
DCX B           ; A darabszámot csökkentjük 1-gyel, mert egy memóriarekeszt megtöltöttünk.
MOV A,B         ; Az elején látott módszerrel ellenőrizzük, hogy végeztünk-e, vagy van még
ORA C           ; megtöltendő memóriarekesz.
JNZ CIKL        ; Ha nem végeztünk, kezdjük elölről.

EXIT:           ; Végeztünk.
POP D           ; Visszaállítjuk a regisztereinket, és felszabadítjuk a STACK-et.
POP B
POP PSW
RET             ; Visszatérünk a szubrutinból, ami olyan, mint c-ben egy függvény.
```

b. Írjon ELLENOR szubrutint, amely a DE regiszterpárban egy kezdőcímet, a BC regiszterpárban egy hossz értéket kap, és ellenőrzi, hogy a memóriablokk rekeszei a KITOLT szubrutin által beírt értékeket tartalmazzák-e?
A szubrutin CY=1-gyel jelezze, ha hibát talált. Ilyenkor a DE regiszterpár az utolsó (legmagasabb memóriacímű) megtalált hiba címét, a HL regiszterpár pedig a hibásnak talált byte-ok darabszámát tartalmazza. Ha nincs hiba, CY=0, HL=0 és DE a memóriablokk első elemére mutat.

```
;ELLENOR szubrutin:
;BC és DE regiszterpárokban vesz át értékeket.
;Elrontja a DE és HL regiszterpárok értékeit, állítja a CY flag értékét.

ELLENOR:
PUSH PSW            ;Mentés a STACK-re
PUSH B
LXI H, 0h           ; A HL-t kinullázzuk, mert még biztosan nincs hiba.
PUSH H              ; Elrakjuk a STACK-re, itt tároljuk majd a hibák számát.
PUSH D              ; A DE-t is a STACK-re rakjuk, ez lesz az utolsó hibás cím.

MOV A, B            ; Ellenőrizzük, hogy nem 0 darab memóriarekeszt kell-e írnunk.
ORA C
JZ VEGE             ; Ha 0 hosszú memóriaterületet kell ellenőriznünk, kilépünk hiba nélkül.

CIKL:
MOV A, E            ; A jó adat újbóli előállítása.
XRA D
CMA
MOV H, A            ; A jó adatot ideiglenesen a H-ba tesszük.
LDAX D              ; Az akkuba beolvassuk a DE által mutatott memóriaterület tartalmát.
XRA H               ; A beolvasott értéket összehasonlítjuk a H tartalmával, ami a jó adatot tartalmazza.
JNZ FALSE           ; Ha nem 0-t kapunk, akkor a kettő nem egyezik, hibát találtunk.
JZ TRUE             ; Ha 0-t kapunk, nem találtunk hibát, mehetünk tovább.

FALSE:
POP H               ; A STACK tetején a DE van, így annak a tartalma
POP H               ; elvész, amikor újra ugyan oda felhozzuk mostmár a HL tartalmát. Ez gyakori trükk.
INX H               ; Hibák számának növelése
PUSH H              ; Hibák számát újra mentjük
PUSH D              ; Az aktuális cím az utolsó hibás cím, ezt újra mentjük.
JMP TRUE            ; Mostmár jó minden, ugorhatunk tovább.

TRUE:
INX D               ; A már jól ismert rutin. DE-t növeljük
DCX B               ; BC-t csökkentjük
MOV A,B             ; Megnézzük végeztünk-e
ORA C
JNZ CIKL            ; Ha nem végeztünk, elölről kezdjük

VEGE:
POPD                ; Előveszünk mindent a STACK-ről, kivéve az akkut, mert azt még használjuk.
POP H
POP B
MOV A,H             ; Megnézzük, hogy a HL 0 értéket tartalmazza-e.
ORA L
JNZ SETCARRY1       ; Ha nem, akkor hiba volt, CY=1-et kell állítanunk.
JZ SETCARRY0        ; Ha igen, akkor nincs hiba, CY=0;

SETCARRY1:
POP PSW             ; Akku felhozatala
STC                 ; CY = 1
RET                 ; Visszatérés

SETCARRY0:
POP PSW             ;Akku felhozatala
STC                 ; Csak 1-be tudjuk állítani CY-t
CMC                 ; De utána negálhatjuk!
RET                 ; Visszatérés
```

c. Írjon programrészletet, amely a processzor SID bemenetén fellépő 0 → 1 átmenet hatására a KITOLT és ELLENOR szubrutinok segítségével ellenőrzi a RAM területet. A teszt indulását és befejeződését a SOD kimeneten egy-egy 1ms idejű impulzussal jelezze. Az időzítés meghatározásánál vegye figyelembe, hogy a program a feladatban meghatározott EPROM memóriában fut!
Figyeljen arra, hogy a tesztelés alatt a RAM terület ne legyen írásvédett.

– Ki kell találnunk a megfelelő maszkoló konstansokat, amelyek segítségével beolvashatjuk a SID bemenetet és amelyek segítségével állíthatjuk a SOD kimenetet. Használjuk a segédletet!
– Gondoskodnunk kell arról, hogy a RAM ne legyen írásvédett.

```
IOADDRESS   EQU   07h       ; Az IO címünk
ADDRESS     EQU   9800h     ; A RAM kezdőcíme.
PCS         EQU   800h      ; Ha a STACK-nek helyet akarnánk hagyni, akkor itt kisebb érték kéne!
RIMMASK     EQU   80h
SIMMASK     EQU   40h

DI                          ; Megszakítás tiltása a teszt alatt.
PUSH PSW                    ; Mentés
PUSH B
PUSH D
PUSH H
MVI A, SIMMASK              ; SOD = 0 beállításához akkuba visszük a megfelelő értéket
SIM                         ; Majd beállítjuk az IT maszkot.

WAIT:
RIM                         ; Beolvassuk az IT maszkot amiből minket csak a SID értéke érdekel,
ANI RIMMASK                 ; így AND kapcsolatba hozzuk a megfelelő konstanssal
JNZ WAIT                    ; Ha SID = 1, akkor nem jöhet 0 → 1 átmenet, így megvárjuk, hogy nullázódjon
JZ NULL                     ; Ha a SID = 0, akkor a következő már a 0 → 1 átmenet lesz.

NULL:
RIM                         ; Újra beolvassuk a SID-et.
ANI RIMMASK
JZ NULL                     ; Addig várunk, amíg nem jön a 0 → 1 átmenet.
JNZ START                   ; Ha megjött a 0 → 1 átmenet, indulunk.

START:
MVI A, 0h
OUT IOADDRESS               ; Kiírjuk az akku értékét, ami 0, így a RAM nem írásvédett
LXI B, PCS                  ; A megfelelő regiszterekbe töltjük a megfelelő adatokat
LXI D, ADDRESS
CALL IMPULSE                ; Ezt mindjárt megírjuk, az 1ms impulzus, ami jelzi a teszt elejét
CALL KITOLT                 ; Ezt már megírtuk, hívjuk meg.
CALL ELLENOR                ; Ezt is megírtuk már.
CALL IMPULSE                ; 1ms impulzus, jelzi a teszt végét.
EI                          ; megszakítást újra engedélyezzük

…
```

Az impulzust generáló szubrutin:
Egy egyszerű ciklus, ami lefoglalja a processzort 1ms ideig, hogy eközben a SOD kimenet értéke ne változhasson.
A SOD kimenetet már korábban 0-ba állítottuk.
Így most 1-be kell állítanunk, 1-en kell tartanunk 1ms ideig, majd újra 0-ba kell állítanunk és vissza kell térnünk.

```
;IMPULSE:
;Nem vesz át értéket és nem ront el regisztert. A SOD kimeneten 1ms idejű impulzust ad.


W1MS        EQU   128          ; Ezt ki fogjuk számolni, hogy miért ennyi.
SIMMASK0    EQU   80h          ; SOD-ot 0-ra állító konstans.
SIMMASK1    EQU   0C0h         ; SOD-ot 1-re állító konstans.


IMPULSE:
PUSH PSW                       ; Mentünk.
PUSH B
LXI B, W1MS                    ; BC-be írjuk a megfelelő konstanst.
MVI A, SIMMASK1                ; Az akkuba írjuk a megfelelő konstanst a SOD 1-re állításához.
SIM                            ; A SOD innentől 1, indul az impulzus.

CIKL:                          ; Egy 1ms ideig futó ciklus, mely annyit csinál, hogy megnézi, végzett-e?
DCX B                          ; BC-t csökkentjük.
MOV A,B                        ; Elértük a 0-t?
ORA C
JNZ CIKL                       ; Ha nem, akkor még futnia kell a ciklusnak.

MVI A, SIMMASK0                ; Letelt az 1ms, a SOD-ot újra 0-ra kell állítanunk.
SIM
POP B                          ; STACK felszabadítása, értékek visszatöltése
POP PSW
RET                            ; Visszatérés
```

Miért annyi az a konstans amennyi?
Egy fázis ideje ezen processzorban 325ns.
Ha kiszámoljuk, hogy ez ciklus hány fázis, akkor azt kapjuk, hogy jelen példában 24. Amennyiben WAIT fázist kérne az EPROM, gépi ciklusonként a WAIT fázisok számát hozzá kéne adnunk.
Az utasítások gépi ciklusainak és fázisainak száma megtalálható a segédletben.

Ha tudjuk, hogy egy ciklus mennyi ideig tart (24 * 325ns) és tudjuk, hogy 1ms ideig akarjuk, hogy fusson, akkor csak el kell osztanunk a kettőt egymással, és megkapjuk a megfelelő konstans értéket.

C = 1ms / (24 * 325ns) ~ 128

128-szor kell tehát lefutnia a ciklusunknak.
