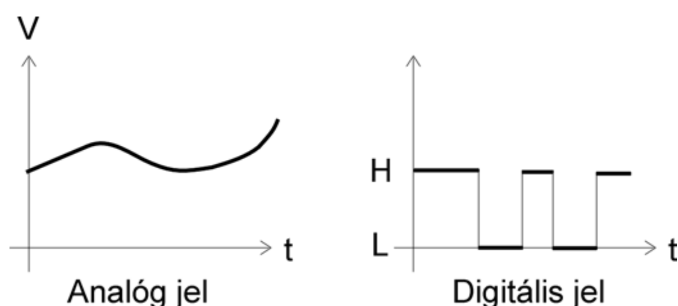
1. Bevezetés

A Pécsi Tudományegyetem Pollack Mihály Műszaki Karán tanuló műszaki informatikus hallgatók mindezidáig más oktatási intézmények által kiadott jegyzetekből és a kereskedelemben kapható drága tankönyvekből tanulhatták a digitális technikát. Bár ezekből tökéletesen elsajátíthatták a tantárgy elméleti részeit, nem volt közöttük egy olyan sem, amely a karunkon folyó képzés követelményeihez és tematikájához teljes mértékben igazodott volna. Jegyzetünk ezt a hiányt hivatott pótolni. Az írás azzal a céllal született, hogy a számítógépeket megtöltsük „élő” alkatrészekkel a hallgatók számára: tanúi lehessenek annak a fejlesztési folyamatnak, melynek során az elemi alkatrészekből fokozatosan felépül egy bonyolult mikroprocesszoros rendszer. A jegyzet természetesen tartalmazza a Digitális technika tantárgyra épülő későbbi kurzusok (Assembly programozás, Programozható vezérlések, stb.) megértéséhez szükséges elméleti alapokat is.

2. Alapfogalmak

Mindenekelőtt tisztáznunk kell a digitális jel illetve rendszer fogalmát. A jelek információtartalommal rendelkező fizikai jellemzők: például egy akkumulátor kimenetén fennálló feszültség, amelynek információtartalma a feszültség nagysága. Az informatikában *analóg* és *digitális* jelekkel találkozhatunk (1. ábra):

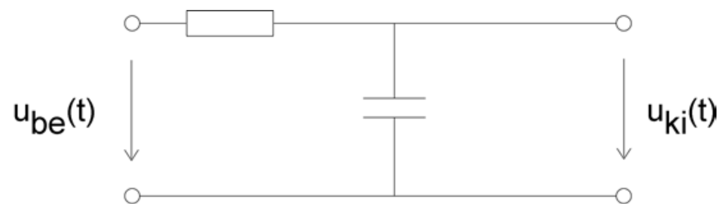
- Az *analóg jel* értékkészlete folytonos, egy adott intervallumon belül bármilyen értéket felvehet: mondjuk egy izzó fényereje bizonyos távolságból.
- A *digitális jel*ek csak kétféle értéket vehetnek föl. A jel információtartalma nem magának a fizikai jellemzőnek a nagysága, hanem az, hogy a kétféle állapot közül éppen melyikben van. Ha a Morse-ábécé szabályai szerint villogtatjuk a fenti izzót, akkor az üzenet elvolvasásakor a lámpa fényét digitális jelként értelmeztük.
  A digitális jel két állapotának elnevezése: 0 és 1, illetve L (low) és H (high).



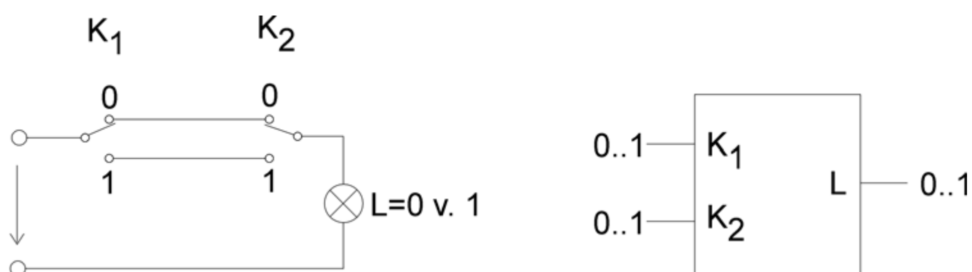
1. ábra

Az analóg és a digitális informatikai rendszereket az különbözteti meg, hogy a be- és kimenetükön, illetve a rendszeren belül előforduló jelek analóg jelek vagy digitálisak. Léteznek hibrid rendszerek is, ezekben mindkét jeltípus előfordul. A 2. ábra egy analóg rendszert mutat, míg a 3. egy digitálisat. Ahhoz, hogy a 3. ábrán látható áramkört digitális rendszernek tekintsük, a

működését logikai szempontból kell vizsgálni. Feltesszük, hogy a kapcsolók kétféle állapotban lehetnek: 0 vagy 1 helyzetben. Ha a lámpa világít, 1-es állapotban, ha nem világít, 0-s állapotban van. Digitális rendszerünknek így 2 bemenete van: K_1 és K_2, egy kimenete: *L*. Ha $K_1 = K_2 = 0$ vagy $K_1 = K_2 = 1$, akkor a lámpa ég: *L*=1, egyébként *L*=0. Ezzel definiáltuk a rendszer működését.



2. ábra



3. ábra

A digitális rendszer bemenetein (illetve kimenetein) pillanatnyilag fennálló összes érték egyetlen 2-es számrendszerbeli számmal is felírható (4. ábra). Ezt *aktuális bemeneti (illetve kimeneti) kombináció*nak nevezzük. Az összes lehetséges kombináció *n* darab vezeték esetén: 2^n.

Aktuális bemeneti kombináció: 10110
Aktuális kimeneti kombináció: 11001

4. ábra

A digitális rendszerek logikai döntést hoznak: egy bizonyos időpillanatban fennálló bemeneti jelkombináció hatására egy előre meghatározott kimeneti jelkombináció jelenik meg. A döntés kétféleképpen történhet:

- *Kombinációs hálózat*ok esetén kizárólag a bemeneti kombinációk aktuális értékei határozzák meg a kimeneti kombinációt.
- *Sorrendi (szekvenciális) hálózat*ok esetén a kimeneti kombinációt a pillanatnyi és a korábban fennállt bemeneti kombinációk határozzák meg (vagyis a rendszer emlékezettel bír).

3. Kombinációs hálózatok

3.1 A kombinációs hálózatok működésének igazságtáblás felírása

Kombinációs hálózatoknál a logikai működés legegyszerűbben az ún. igazságtábla segítségével írható fel. Ez esetben az összes lehetséges bemeneti kombinációra megadjuk a kimenet(ek) értékét, táblázatos formában. A 3. ábrán látható hálózat igazságtáblája a következő:

K_1	K_2	L
0	0	1
0	1	0
1	0	0
1	1	1

Több kimenetre is lássunk egy példát:

Be_1	Be_2	Ki_1	Ki_2
0	0	0	1
0	1	0	0
1	0	1	0
1	1	1	1

Ha bizonyos bemeneti kombinációk fennállásával nem kell számolnunk működés közben, vagy nem fontos esetükben definiálni a kimeneti kombinációt, az adott helyre vonalat vagy x-et helyezünk az igazságtáblában. Ez nem azt jelenti, hogy ilyenkor egy harmadik állapotba ugrik a kimenet; értéke 0 vagy 1, csak éppen nem lényeges, hogy melyik a kettő közül:

Be_1	Be_2	Ki
0	0	1

0	1	-
1	0	0
1	1	1

3.2 A kombinációs hálózatok működésének definiálása logikai függvényekkel

Az analóg rendszer ki- és bemenetei közötti kapcsolatot folytonos függvények adják meg (gondoljunk arra, hogy egy analóg Volt-mérő kapcsaira csatolt feszültség nagysága miképp befolyásolja a mutató kitérését). A kombinációs hálózatok ki- és bemenetei közötti összefüggések *logikai függvények*kel írhatók fel. Ehhez a matematikai alapot a *Boole-algebra* adja.

A Boole-algebra a kétértékű jelekkel végzett logikai műveletek algebrai leírását teszi lehetővé. Szabályai:

- Bármely változó lehetséges értékei: 0 vagy 1.
- Elvégezhető műveletek:

Logikai szorzás (konjunkció), ÉS kapcsolat:

$0 \cdot 0 = 0$ (1)

$0 \cdot 1 = 0$ (2)

$1 \cdot 0 = 0$ (3)

$1 \cdot 1 = 1$ (4)

Logikai összeadás (diszjunkció), VAGY kapcsolat:

$0 + 0 = 0$ (5)

$0 + 1 = 1$ (6)

$1 + 0 = 1$ (7)

$1 + 1 = 1$ (8)

Logikai tagadás (negáció):

$\overline{1} = 0$ (9)

$\overline{0} = 1$ (10)

- Alaptételek, azonosságok:

$A \cdot 0 = 0$ (11)

$$A \cdot 1 = A \quad (12)$$

$$A + 0 = A \quad (13)$$

$$A + 1 = 1 \quad (14)$$

$$A \cdot \overline{A} = 0 \quad (15)$$

$$A + \overline{A} = 1 \quad (16)$$

$$A \cdot A = A \quad (17)$$

$$A + A = A \quad (18)$$

$$\overline{\overline{A}} = A \quad (19)$$

- kommutativitás:

$$A + B = B + A \quad (20)$$

$$A \cdot B = B \cdot A \quad (21)$$

- asszociativitás:

$$A + (B + C) = A + B + C \quad (22)$$

$$A \cdot (B \cdot C) = A \cdot B \cdot C \quad (23)$$

- disztributivitás:

$$A(B + C) = AB + AC \quad (24)$$

$$A + (BC) = (A + B)(A + C)!!! \quad (25)$$

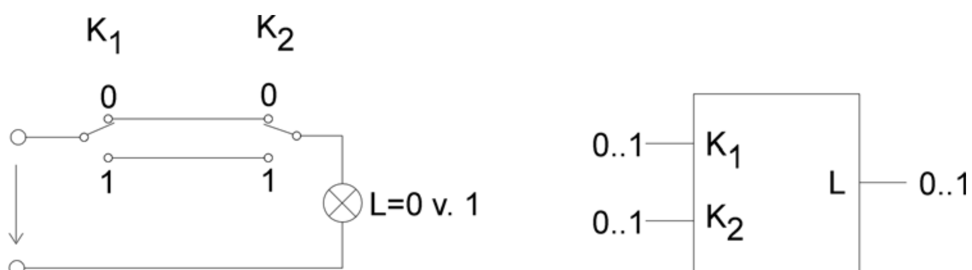
- De-Morgan azonosságok (tetszőleges számú változóra):

$$\overline{A \cdot B \cdot C} = \overline{A} + \overline{B} + \overline{C} \quad (26)$$

$$\overline{A + B + C} = \overline{A} \cdot \overline{B} \cdot \overline{C} \quad (27)$$

3.2.1 Példa: a „folyosó-kapcsolás” logikai függvényének meghatározása

Vegyük újra elő az *Alapfogalmak* részben tárgyalt kapcsolást (5. ábra)!



5. ábra

A rendszer logikai működése a következő:

$$L = 1 \text{, ha } \{K_1 = 1 \text{ ÉS } K_2 = 1\} \text{ VAGY } \{K_1 = 0 \text{ ÉS } K_2 = 0\}. \quad (28)$$

Ezt másképp is megfogalmazhatjuk:

$$L = 1 \text{, ha } \{K_1 = 1 \text{ ÉS } K_2 = 1\} \text{ VAGY } \{\overline{K_1} = 1 \text{ ÉS } \overline{K_2} = 1\}. \quad (29)$$

Az állítás első fele Boole-algebrai összefüggésekkel felírva:

$$L = K_1 \cdot K_2 , \quad (30)$$

a második fele pedig:

$$L = \overline{K_1} \cdot \overline{K_2} . \quad (31)$$

A kettő közül az egyik VAGY a másik igaz, tehát:

$$L = K_1 \cdot K_2 + \overline{K_1} \cdot \overline{K_2} . \quad (32)$$

3.2.2 Példa: logikai függvény felírása az igazságtáblából

$$Q = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C$$

A	B	C	Q
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Ha az első sornak megfelelő logikai kombináció érkezik a bemenetekre, a függvényben szereplő első szorzat eredménye 1, ezáltal a függvényé is. Ugyanez igaz a többi 1-es eredményt adó sorra is a fenti felírási mód szerint. A függvény tehát az igazságtáblának megfelelő működést adja.

3.3 Logikai függvények algebrai egyszerűsítése

Minél egyszerűbb egy kombinációs hálózat logikai függvénye, annál kevesebb áramköri elemmel tudjuk megvalósítani. A függvények egyszerűsítésének

legkézenfekvőbb módja a Boole-algebra összefüggéseit intuitív módon felhasználó algebrai egyszerűsítés. Egy függvény annál egyszerűbb, minél kevesebb a benne szereplő műveletek és változók száma.

3.3.1 Példa algebrai egyszerűsítésre

Egyszerűsítsük az alábbi függvényt!

$$Q = \overline{AB + \overline{B}C + BC} \tag{33}$$

A (25)-el, majd a (24)-el jelölt De-Morgan azonosságot felhasználva:

$$Q = \overline{AB + \overline{B}C + BC} = \overline{AB} \cdot \overline{\overline{B}C} \cdot \overline{BC} = (\overline{A} + \overline{B})(B + \overline{C})(\overline{B} + \overline{C}) \tag{34}$$

A 2. és 3. zárójel közti szorzásokat elvégezve:

$$Q = (\overline{A} + \overline{B})(B\overline{B} + B\overline{C} + \overline{B}\overline{C} + \overline{C}\overline{C}) \tag{35}$$

A Boole-algebra (15) és (17) azonosságai szerint $B\overline{B} = 0$, $\overline{C}\overline{C} = \overline{C}$, így:

$$Q = (\overline{A} + \overline{B})(B\overline{B} + B\overline{C} + \overline{B}\overline{C} + \overline{C}\overline{C}) = (\overline{A} + \overline{B})(B\overline{C} + \overline{B}\overline{C} + \overline{C}), \tag{36}$$

$\overline{C}$-at kiemelve és az azonosságokat használva:

$$Q = (\overline{A} + \overline{B})(B\overline{C} + \overline{B}\overline{C} + \overline{C}) = (\overline{A} + \overline{B})\overline{C}(B + \overline{B} + 1) = (\overline{A} + \overline{B})\overline{C} = \overline{AB} \cdot \overline{C} = \overline{AB + C} \tag{37}$$

Létezik egyszerűbb megoldási mód is:

$$Q = \overline{AB + \overline{B}C + BC} = \overline{AB + C(\overline{B} + B)} = \overline{AB + C} \tag{38}$$

Ebből is látható az algebrai egyszerűsítés erősen intuitív jellege. A későbbiekben tárgyalt grafikus egyszerűsítés kiküszöböli ezt a problémát.

3.4 Logikai kapcsolási vázlat

Miután felírtuk a logikai függvényt, a *logikai kapcsolási vázlat* megszerkesztése lesz a fizikai megvalósítás felé vezető út következő állomása. Ehhez meg kell ismerkedni a *logikai kapu*k fogalmával. Minden egyes logikai művelethez egy szimbólumot rendelünk: egy olyan egyszerű kombinációs

hálózat jelét, amely kizárólag az adott műveletet végzi el. Ezeket az alaphálózatokat nevezzük logikai kapuknak (6/1. ábra).

VAGY (OR) kapu — A, B → [1] → $X=A+B$

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

ÉS (AND) kapu — A, B → [&] → $X=A \cdot B$

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

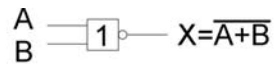
inverter — A → [1]○ → $X=\overline{A}$

A	X
0	1
1	0

6/1. ábra

Mint látni fogjuk, az alapkapuk felhasználásával bármelyik kombinációs hálózat megépíthető. A három alapműveleten kívül néhány összetettebb funkciót ellátó kaput is egyedi szimbólummal jelölnek a nagyon gyakori alkalmazásuk miatt (6/2. ábra).

NEM VAGY (NOR) kapu

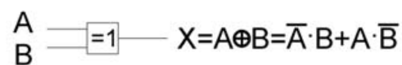


A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

NEM ÉS (NAND) kapu

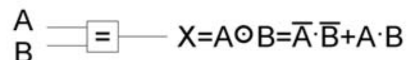
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

antivalencia (kizáró VAGY: XOR), kapu



A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

ekvivalencia (megengedő VAGY: IOR) kapu



A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

6/2. ábra

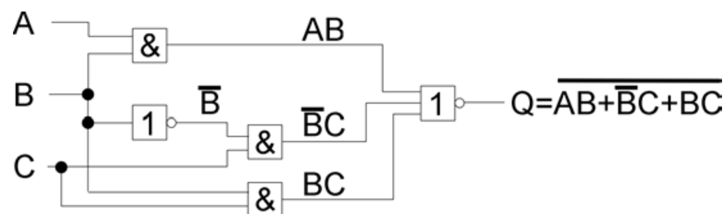
Észrevehetjük, hogy ahol kis köröcske található, ott invertálás történik. Ennek megfelelően a *NEM VAGY kapu* helyettesíthető egy VAGY kapuval és egy utána kötött inverterrel is. Hasonlóan a *NEM ÉS kapu* egy ÉS kapu és egy inverter egybeépítése. Az *antivalencia kapu* speciális feladatot lát el: a kimenetén akkor jelenik meg 1-es, ha a bemeneteire adott jelek értéke különböző (antivalens). Az *ekvivalencia kapu* ennek épp a fordítottját csinálja: azt jelzi 1-essel a kimenetén, ha a bemenetek értéke azonos (ekvivalens).

A logikai kapcsolási vázlat tulajdonképpen egy olyan, logikai kapukból összeállított „áramkör”, amelynek kimenetein a kívánt függvénynek megfelelő értékek jelennek meg. Lássunk erre egy példát!

3.4.1 Példa: logikai kapcsolási vázlat felrajzolása a logikai függvényből

Rajzoljuk fel az előző példában szereplő hálózat egyszerűsítés előtti és utáni függvényét! A bonyolultabb összefüggés:

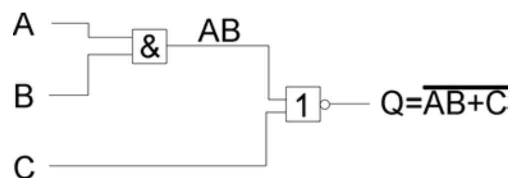
$$Q = \overline{AB + \overline{B}C + BC}\,. \tag{39}$$



7. ábra

Az egyszerűsített függvény logikai kapcsolási vázlata pedig:

$$Q = \overline{AB + C}\,. \tag{40}$$

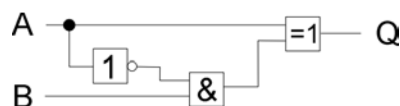


8. ábra

A megépítendő elektronikus kapcsolás és a logikai kapcsolási vázlat bonyolultsága lényegében megegyezik. A példából kitűnik a függvény-egyszerűsítés fontossága.

3.4.2 Példa: logikai függvény felírása a logikai kapcsolási vázlatból

Egy kombinációs hálózat logikai vázlata a 9. ábra szerinti. Írjuk fel a függvényt!



9. ábra

Az antivalencia kapu két bemenetén A illetve $\overline{A}B$ jelenik meg. Eszerint:

$$Q = A \oplus (\overline{A}B) \tag{41}$$

Fejtsük ki ezt a kifejezést, és próbáljuk egyszerűbb formájúra alakítani! A 6. ábra antivalencia kapura vonatkozó összefüggése alapján:

$$Q = A \oplus (\overline{A}B) = A\overline{\overline{A}B} + \overline{A} \cdot \overline{A}B \tag{42}$$

Ebből a (26) De-Morgan és egyéb azonosságokkal:

$$\begin{aligned} Q &= A \oplus (\overline{A}B) = A\overline{\overline{A}B} + \overline{A} \cdot \overline{A}B = A(A + \overline{B}) + \overline{A}B = AA + A\overline{B} + \overline{A}B = \\ &= A + A\overline{B} + \overline{A}B = A(1 + \overline{B}) + \overline{A}B = A + \overline{A}B \end{aligned} \tag{43}$$

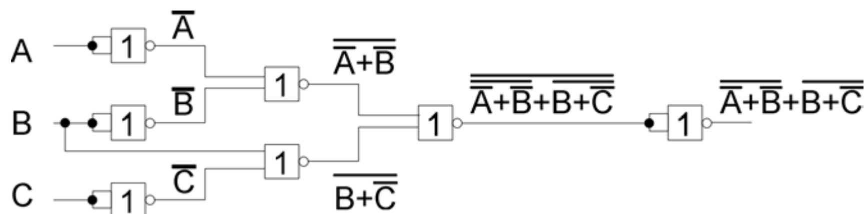
3.4.3 Példa: kombinációs hálózat megvalósítása kizárólag NOR illetve NAND kapuk felhasználásával

Bármely kombinációs hálózat megvalósítható csak NOR vagy csak NAND kapukkal is. Ennek például az az előnye, hogy az integrált áramkörök gyártóinak nem kell többféle kapu gyártástechnológiáját egyetlen chipen belül kombinálni. Az átalakítás a De-Morgan azonosságok alkalmazásával oldható meg. Felhasználjuk azt a tényt is, hogy egy invertert egy NOR vagy egy NAND kapu bemeneteinek összekötésével is meg lehet valósítani (ennek könnyű utánaszámolni). Legyen a függvény:

$$Q = AB + \overline{B}C \tag{44}$$

A NOR kapus megvalósítás:

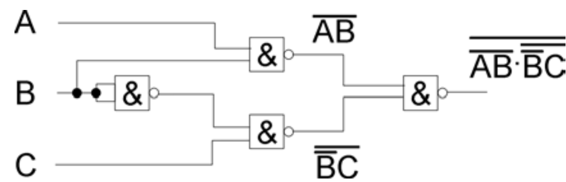
$$Q = AB + \overline{B}C + AB\overline{C} = \overline{\overline{AB}} + \overline{\overline{\overline{B}C}} + \overline{\overline{AB\overline{C}}} = \overline{\overline{A} + \overline{B}} + \overline{B + \overline{C}} \tag{45}$$



10. ábra

A NAND kapus megvalósítás:

$$Q = \overline{\overline{Q}} = \overline{\overline{AB + \overline{B}C}} = \overline{\overline{AB} \cdot \overline{\overline{B}C}} \tag{46}$$



11. ábra

3.5 Logikai függvények kanonikus (normál) alakjai

Mint láttuk, ugyanaz a logikai függvény több formában is megadható. Az egyértelműség kedvéért célszerű olyan felírási módot követni, amely esetén egy bizonyos függvény csak egyféleképpen írható le, és ha két függvény különböző, az alakjuk is biztosan különbözik. Ha mindez teljesül, a függvény kanonikus alakjáról beszélünk.

3.5.1 Diszjunktív kanonikus alak

Egy logikai függvény diszjunktív kanonikus alakban történő felírásakor az alábbi formai szabályok érvényesek:

- a függvény szorzatok összege,
- a szorzatokban valamennyi bemeneti változó negált vagy ponált alakja szerepel,
- a kimenet értéke 1, ha bármely szorzat eredménye 1.

Például egy 3 változós függvény diszjunktív kanonikus alakja a következő:

$$Q = \overline{A}\,\overline{B}C + A\overline{B}C + AB\overline{C} + ABC\ . \tag{47}$$

A fenti szorzatokat mintermeknek nevezzük. Létezik egy speciális jelölésük: m_i^n, ahol n a független változók száma, i a változókombinációt jelölő bináris szám decimális értéke. A fenti függvény felírása mintermekkel:

$$Q(A,B,C) = m_1^3 + m_5^3 + m_6^3 + m_7^3 . \quad (48)$$

A diszjunktív kanonikus alak könnyedén felírható az igazságtáblából, a … példa szerinti módszerrel.

3.5.2 Konjunktív kanonikus alak

Egy logikai függvény konjunktív kanonikus alakjának felírási szabályai a következők:

- a függvény összegek szorzata,
- az összegekben valamennyi bemeneti változó negált vagy ponált alakja szerepel,
- a kimenet értéke 1, ha minden összeg eredménye 1.

Például:

$$Q = (\overline{A} + \overline{B} + \overline{C})(\overline{A} + B + \overline{C})(\overline{A} + B + C)(A + \overline{B} + \overline{C}) . \quad (49)$$

A fenti összegeket maxtermeknek nevezzük. Jelölésük: M_i^n, ahol n a független változók száma, i a változókombinációt jelölő bináris szám decimális értéke. A fenti függvény felírása maxtermekkel:

$$Q(A,B,C) = M_1^3 \cdot M_2^3 \cdot M_3^3 \cdot M_4^3 . \quad (50)$$

A konjunktív alak is felírható az igazságtáblából. Először felvesszük a függvény negáltját diszjunktív alakban, majd ezt De-Morgan azonosságokkal összegek szorzatává alakítjuk:

A	B	C	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$\overline{Q} = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\,\overline{C} \tag{51}$$

$$\begin{aligned}\overline{\overline{Q}} &= \overline{\overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\,\overline{C}} = \overline{\overline{A}\,\overline{B}\,\overline{C}} \cdot \overline{\overline{A}B\overline{C}} \cdot \overline{\overline{A}BC} \cdot \overline{A\overline{B}\,\overline{C}} = \\ &= (A + B + C)(A + \overline{B} + C)(A + \overline{B} + \overline{C})(\overline{A} + B + C)\end{aligned} \tag{52}$$

3.6 Logikai függvények grafikus minimalizálása

Mint az korábban kiderült, az algebrai egyszerűsítés sikere nagyban függ a számítást végző gyakorlatától, vagy attól, hogy éppen mennyire tud az adott feladatra koncentrálni. A következőkben ismertetett módszer ezeket az emberi tényezőket küszöböli ki.

Tanulmányozzuk az alábbi algebrai minimalizálást! A kombinációs hálózat igazságtáblája legyen:

A	B	C	Q
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Ennek megfelelően a diszjunktív kanonikus alak:

$$Q = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C} + A\overline{B}C \,. \tag{53}$$

Egyszerűsítsük!

$$\begin{aligned} Q &= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C} + A\overline{B}C = \\ &= \overline{A}\,\overline{B}(\overline{C} + C) + A\overline{B}(\overline{C} + C) = \\ &= \overline{A}\,\overline{B} + A\overline{B} = \\ &= \overline{B}(\overline{A} + A) = \\ &= \overline{B} \end{aligned} \tag{54}$$

Láthatjuk, hogy az összevonható mintermek csak egy helyiértékben térnek el egymástól: C az egyikben ponáltan, a másikban negáltan szerepel, ezért összevonáskor kiesik. Az ilyen mintermeket *szomszédos mintermek*nek nevezzük.

Nézzük tovább! $\overline{A}\overline{B}$ és $A\overline{B}$ is csak egy helyen eltérő, ezért a fenti módszerrel egyszerűsíthető. $\overline{A}\overline{B}$ és $A\overline{B}$ elnevezése: *szomszédos term*ek. A tovább nem bontható termeket *primimplikáns*oknak nevezzük. A számítások menete tehát a következő volt::

1. lépés: Szomszédos mintermek keresése, összevonása.
2. lépés: Szomszédos termek keresése, összevonása
3. lépés: A 2. lépést ismételni, ameddig lehetséges.

Ugyanez a módszer maxtermekre is alkalmazható:

$$\begin{aligned} Q &= (A+B+C)(\overline{A}+B+C) = \\ &= (A+(B+C))(\overline{A}+(B+C)) = \\ &= A\overline{A} + A(B+C) + \overline{A}(B+C) + (B+C)(B+C) = \\ &= (A+\overline{A})(B+C) = \\ &= (B+C) \end{aligned} \quad (55)$$

Vagyis a nem azonos változót elhagyva itt is egyszerűbb alakra jutottunk.

A tapasztalatok felhasználásával a következő grafikus módszert alkalmazhatjuk:
Vegyünk egy igazságtáblát!

A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Rendezzük át ezt egy úgynevezett *Karnaugh-táblába*! (Nagyon fontos a számozás sorrendje, ezt nem szabad eltéveszteni!)

AB \ C	0	1
00	0	0
01	1	0
11	1	1
10	1	1

Tudjuk, hogy az igazságtáblázatban minden olyan sor, ahol a kimenet értéke 1, egy-egy mintermet ad meg. A táblázatra is igaz ez, méghozzá úgy, hogy a szomszédos mintermek szomszédos négyzetekben vannak. Éppen ezért nagyon könnyű felfedezni a szomszédos mintermeket, és így egyszerűsíteni is:

AB \ C	0	1
00	0	0
01	1	0
11	1	1
10	1	1

$$\overline{A}B\overline{C} + AB\overline{C} = (\overline{A} + A)B\overline{C} = B\overline{C}\ , \tag{56}$$

AB \ C	0	1
00	0	0
01	1	0
11	1	1
10	1	1

$$AB\overline{C} + A\overline{B}\,\overline{C} = A\overline{C}\ , \tag{57}$$

AB \ C	0	1
00	0	0
01	1	0
11	1	1
10	1	1

$$ABC + A\overline{B}C = AC\ . \tag{58}$$

A teljes megoldás (az algebrai módszerhez hasonlóan) a kapott termek VAGY kapcsolata:

$$Q = B\overline{C} + A\overline{C} + AC\,. \tag{59}$$

Folytassuk az egyszerűsítést a szomszédos termek összevonásával! Láttuk, hogy minden termet egy-egy kettes hurok jelöl. A szomszédos termeket szomszédos hurkok jelölik, így ezzel is könnyű dolgunk van:

AB \ C	0	1
00	0	0
01	1	0
11	1	1
10	1	1

Összevonás után:

AB \ C	0	1
00	0	0
01	1	0
11	1	1
10	1	1

$$A\overline{C} + AC = A(\overline{C} + C) = A\,. \tag{60}$$

A teljes megoldás:

AB \ C	0	1
00	0	0
01	1	0
11	1	1
10	1	1

$$Q = B\overline{C} + A\,. \tag{61}$$

A Karnaugh-tábla jellegzetessége, hogy a táblázat szélein „túlnyúlva" is szomszédos mintermeket találunk:

AB \ C	0	1
00	1	0
01	0	0
11	0	0
10	1	0

$Q = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} = \overline{B}\,\overline{C}$, vagy például: (62)

AB \ C	0	1
00	1	1
01	0	0
11	0	0
10	1	1

$Q = \overline{B}$. (63)

Egyedülálló 1-es esetén egyszerűsítésre nincs mód, ekkor a teljes minterm felírásra kerül (egyes hurok):

AB \ C	0	1
00	0	0
01	1	0
11	0	0
10	1	1

$Q = \overline{A}B\overline{C} + A\overline{B}$. (64)

Segédvonalak alkalmazásával a folyamat még átláthatóbbá tehető. Az A változó értéke a 3. és 4. sorokban 1. Rajzoljunk ide egy segédvonalat! A segédvonal tehát kijelöli azt a tartományt, ahol az A értéke 1.

AB \ C	0	1
00	0	0
01	1	0
11 (A)	1	1
10 (A)	1	1

A többi változónál is tegyük ezt meg!

AB \ C	0	1	
00	0	0	
01	1	0	B
11 (A)	1	1	B
10 (A)	1	1	
		C	

Jelöljük ki a hurkokat!

AB \ C	0	1	
00	0	0	
01	1	0	B
11 (A)	1	1	B
10 (A)	1	1	
		C	

Minden egyes hurokra nézzük meg, hogy melyik tartomány foglalja teljesen magába, melyikből marad ki teljes egészében, és melyik az, amelybe csak egy része lóg bele! Például a kettes hurok az A tartományból félig kilóg, a B-ben teljesen bent van, a C tartományon pedig teljes egészében kívül esik. Az így nyert adatok alapján a következőképpen írhatjuk fel a termeket egy bizonyos hurokra:

- Ha valamelyik tartományban teljesen bent van, a tartományhoz tartozó változó ponáltan szerepel a szorzatban.
- Ha valamely tartományból teljesen kilóg, a változó negáltan szerepel a szorzatban.

- Ha valamely tartomány csak részben tartalmazza a hurkot, az ahhoz tartozó változót elhagyjuk a szorzatból.

A kettes hurok tehát a következő termet jelöli: $B\overline{C}$

A négyes hurok teljes egészében az A tartományban van, a B és C tartományból félig kilóg. A leírt term ezért: A

A teljes megoldás:

$$Q = B\overline{C} + A. \tag{65}$$

3.6.1 4 változós Karnaugh-tábla

Vegyünk egy négy bemeneti változós igazságtáblát!

A	B	C	D	Q
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Írjuk át a következő Karnaugh-táblába!

AB \ CD	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	1	1
10	1	0	1	1

Itt is felvehető kettes, négyes, sőt nyolcas hurok is, hiszen a szomszédos négyes hurkok is összevonhatók:

AB \ CD	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	1	1
10	1	0	1	1

A három változós táblánál látottak szerint írjuk fel a termeket:

- A táblázat szélein túlnyúló négyes hurok: $A\overline{D}$
- A vízszintes négyes hurok: AB
- A nyolcas hurok: C
- A teljes megoldás tehát:

$$Q = A\overline{C}\overline{D} + AB + C \qquad (66)$$

Láthatjuk, hogy minél nagyobb hurkot találtunk, annál egyszerűbb termet kapunk. Célunk tehát a lehető legnagyobb hurkokat megkeresni. Az egy hurokban levő 1-esek száma 2, 4, 8, 16, stb., tehát 2 valahányadik hatványa lehet.

Ha tüzetesebben megvizsgáljuk a 4 változós Karnaugh táblát, akkor arra is rájöhetünk, hogy a négy sarok is szomszédos, így összevonható:

AB \ CD	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

$$Q = \overline{B}\,\overline{D} \tag{67}$$

Létezik 5, illetve 6 változós Karnaugh-tábla is, e fölött a grafikus módszer átláthatatlanná válik.

A Karnaugh-táblás egyszerűsítés folyamata tehát a következő:

1. Az igazságtábla (vagy a diszjunktív kanonikus alak) átírása Karnaugh-táblás formába.
2. Hurkok keresése a következő szempontok szerint:
   - Minden 1-est le kell fedni legalább egy huroknak. 0 nem kerülhet egyik hurokba sem.
   - Minden hurokban csak 2 valahányadik hatványának megfelelő számú egyes lehet.
   - Úgy kell minden 1-est lefedni, hogy ezt a lehető legkevesebb számú hurokkal tegyük.
   - A lehető legnagyobb hurkokat kell keresni.
   - A hurkok egymásba nyúlhatnak.
3. A termek felírása minden egyes hurokra a következő módszerrel:

- Ha valamelyik tartományban teljesen bent van, a tartományhoz tartozó változó ponáltan szerepel a szorzatban.
- Ha valamely tartományból teljesen kilóg, a változó negáltan szerepel a szorzatban.
- Ha valamely tartomány csak részben tartalmazza a hurkot, az ahhoz tartozó változót elhagyjuk a szorzatból.

3.6.2 Nem teljesen definiált logikai függvény grafikus minimalizálása

Mint az már említésre került, a kombinációs hálózatok igazságtáblájában határozatlan értékek is szerepelhetnek. Lássuk, ebben az esetben hogyan történik az egyszerűsítés! Példaként induljunk ki az alábbi igazságtáblából:

A	B	C	Q
0	0	0	0
0	0	1	-
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	-
1	1	1	1

A Karnaugh-táblába való átírás a szokott módon történik:

AB \ C	0	1
00	0	-
01	1	0
11	-	1
10	1	1

A hurkok felvétele során eldönthetjük, hogy a határozatlan értékeket 1-esként vagy 0-ként szerepeltessük, attól függően, hogyan lesz egyszerűbb a kapott eredmény. (Tehát vagy bevesszük őket a hurokba, vagy nem.)

AB \ C	0	1
00	0	-
01	1	0
11	-	1
10	1	1

$Q = B\overline{C} + A$. (68)

Példánkban az egyik értékre szükségünk volt, a másikra nem, ezért ezt nem fedtük le hurokkal. Ha ez utóbbit is bevettük volna egy újabb hurokba, avval a megoldás nem lett volna hibás, csak bonyolultabb.

3.6.3 Konjunktív alak felírása Karnaugh-táblával

Ha az összegek szorzatából álló (vagyis konjunktív) függvény felírása a célunk, hasonló módon kell eljárnunk, mint a konjunktív kanonikus alak meghatározásánál (ld. 3.5.2. fejezet): az 1-esek helyett a nullákat fedjük le hurkokkal, s ebből a függvény *negáltjának* diszjunktív alakját írjuk fel. A kapott függvényből a De-Morgan azonosságok alkalmazásával nyerjük a konjunktív formát. A gyakorlottabbak ránézésre is megállapíthatják az eredményt.

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	1	1	1
10	1	1	1	1

$\overline{Q} = \overline{A}\,\overline{B} + \overline{A}D$ (69)

$Q = \overline{\overline{A}\,\overline{B} + \overline{A}D} = \overline{\overline{A}\,\overline{B}} \cdot \overline{\overline{A}D} = (A+B)(A+\overline{D})$ (70)

Vajon egyenértékű-e ugyanazon hálózatnál a diszjunktív és konjunktív megoldás? Tegyünk egy próbát! Az igazságtábla legyen:

A	B	C	Q
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Először a legegyszerűbb diszjunktív alakot számítsuk ki!

AB \ C	0	1
00	1	1
01	1	0
11	0	0
10	0	0

$$Q = \overline{A}\,\overline{B} + \overline{A}\,\overline{C}\,, \tag{71}$$

majd a konjunktív alakot is:

AB \ C	0	1
00	1	1
01	1	0
11	0	0
10	0	0

$$\overline{Q} = A + BC \rightarrow Q = \overline{A + BC} = \overline{A} \cdot \overline{BC} = \overline{A}(\overline{B} + \overline{C})\,. \tag{72}$$

A konjunktív forma egyszerűbbnek bizonyult (3db invertálás+ 1db ÉS művelet+ 1db VAGY művelet= 5 művelet, szemben a diszjunktív alak 6 műveletével). Ez persze nem mindig van így, a tökéletességre törekvőknek érdemes mindkét egyszerűsítési módot elvégezni.

3.6.4 Nem teljesen definiált függvény konjunktív alakja

A diszjunktív alakhoz hasonlóan a határozatlan értékeket tetszőlegesen lehet 0-val vagy 1-essel behelyettesíteni:

AB \ CD	00	01	11	10
00	1	1	-	1
01	-	-	0	0
11	1	1	0	0
10	1	1	0	0

$$Q = (\overline{A} + \overline{C})(A + \overline{B}) \tag{73}$$

3.6.5 Több kimenetű kombinációs hálózatok grafikus egyszerűsítése

A kettő vagy több kimenettel rendelkező kombinációs hálózatok minimalizálásánál két utat választhatunk:

- a kimeneti függvényeket külön-külön egyszerűsítjük, ezzel gyakorlatilag független, közös bemenetekkel rendelkező alhálózatokra bontva a rendszert,
- a másik, takarékosabb mód az, hogy a függvényeket ugyan külön-külön írjuk fel, de odafigyelünk arra, hogy közös elemek is szerepeljenek bennük: ezeket a részeket a fizikai megvalósításnál elég lesz csak egyszer megépítenünk.

Világosítsuk meg a különbséget egy példán! Vegyük az alábbi rendszert:

12. ábra

A	B	C	D	P	Q
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	1
1	0	1	1	0	1
1	1	0	0	0	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	1	1	1

Először egyszerűsítsük a két függvényt külön-külön, a szokásos módon!

P

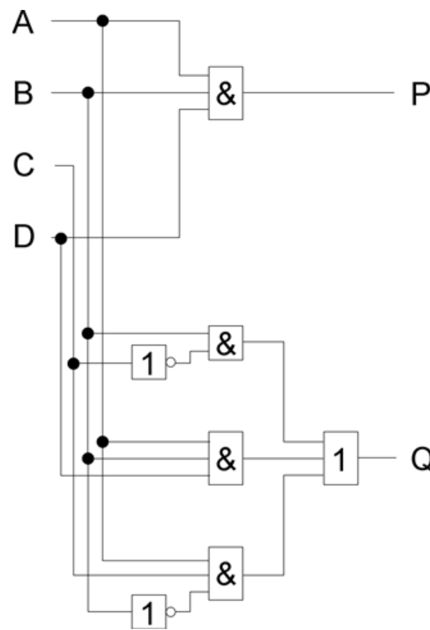
AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	1	1	0
10	0	0	0	0

$$P = ABD \tag{74}$$

Q: AB \ CD	00	01	11	10
00	0	0	0	0
01	1	1	0	0
11	1	1	1	0
10	0	0	1	1

$$Q = B\overline{C} + ACD + A\overline{B}C \tag{75}$$

Hogyan néz ki ennek a logikai vázlata? Rajzoljuk föl:



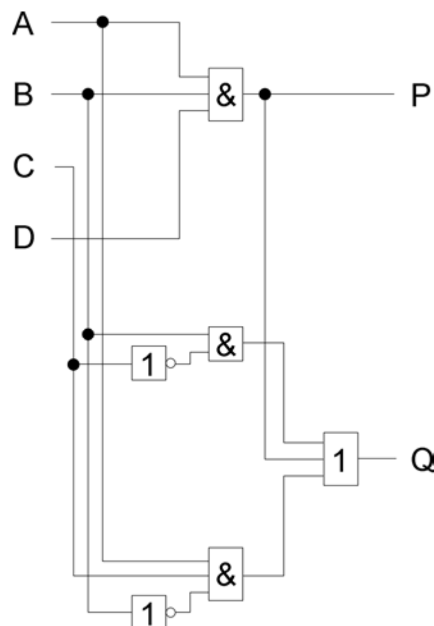
13. ábra

Ezek után egyszerűsítsük Q-t egy kicsit másképpen!

Q: AB \ CD	00	01	11	10
00	0	0	0	0
01	1	1	0	0
11	1	1	1	0
10	0	0	1	1

$$Q = B\overline{C} + ABD + A\overline{B}C \qquad (76)$$

Mint látjuk, P és Q esetében a szaggatott vonallal rajzolt hurok – így egy term – azonos. Mivel minden term egy-egy ÉS kapuval valósítható meg a logikai vázlatban, a kapcsolási rajzunk egyszerűsödik (ld. 14. ábra). Több kimenetű hálózatok esetén tehát közös hurkok keresése ajánlott.

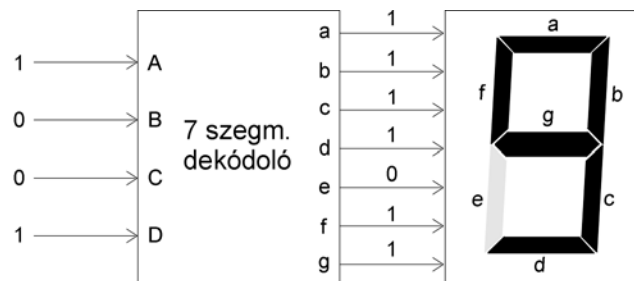


14. ábra

3.6.6 Példa grafikus egyszerűsítésre: hét szegmenses dekódoló tervezése

A 15. ábra jobb oldalán egy hét *szegmenses kijelző*t látunk: ilyennel jelenítik meg az egyes számokat a digitális órákon. A megfelelő szegmensek kigyújtásával 0-tól 9-ig az összes szám kirajzolható. A *hét szegmenses dekódoló* egy olyan digitális áramkör, amely a bemenetein bináris, kettes számrendszerbeli számokat vár, kimeneteit pedig a kijelző egyes szegmenseivel összekötve a megfelelő számot rajzolja ki (az ábra jobb oldalán). Ha például a bemenetre 1001-t, vagyis kilencet rakunk, az ’e’

kivételével az összes kimenetet magas logikai szintre állítja, így jelenítve meg a kilences számot.



15. ábra

Az áramkör igazságtáblája négy bemeneti és hét kimeneti változót tartalmaz. *D* a bemenetre adott szám legkisebb helyiértéket jelöli, *A* pedig a legnagyobbat. A táblázatot kétféleképpen értelmezhetjük:

- Egy adott kimenet oszlopában azon bemenő számoknál szerepel 1-es, ahol a szegmensek ki kell gyulladnia.
- Egy adott bemenő szám esetén azon kimeneteknél szerepel 1-es, amelyek a számot kirajzolják.

A kijelző csak 0-tól 9-ig képes számokat kiírni. Ha a bemenetre mégis ennél nagyobb szám kerülne, az áramkör nem foglalkozik vele, így az igazságtáblába definiálatlan értékek (-) kerülnek.

	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	-	-	-	-	-	-	-
11	1	0	1	1	-	-	-	-	-	-	-
12	1	1	0	0	-	-	-	-	-	-	-
13	1	1	0	1	-	-	-	-	-	-	-
14	1	1	1	0	-	-	-	-	-	-	-
15	1	1	1	1	-	-	-	-	-	-	-

A hét kimeneti függvényt egy-egy Karnaugh-táblával egyszerűsítjük:

a

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	-	-	-	-
10	1	1	-	-

$$a = A + C + BD + \overline{B}\,\overline{D} \tag{77}$$

b

AB \ CD	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	-	-	-	-
10	1	1	-	-

$$b = \overline{C}\,\overline{D} + CD + \overline{B} \tag{78}$$

c

AB \ CD	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	-	-	-	-
10	1	1	-	-

$$c = B + \overline{C} + D \tag{79}$$

d

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	-	-	-	-
10	1	1	-	-

$$d = A + \overline{B}\,\overline{D} + B\overline{C}D + C\overline{D} + \overline{B}C \tag{80}$$

e

AB \ CD	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	-	-	-	-
10	1	0	-	-

$$e = \overline{B}\,\overline{D} + \overline{B}C \qquad (81)$$

f

AB \ CD	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	-	-	-	-
10	1	1	-	-

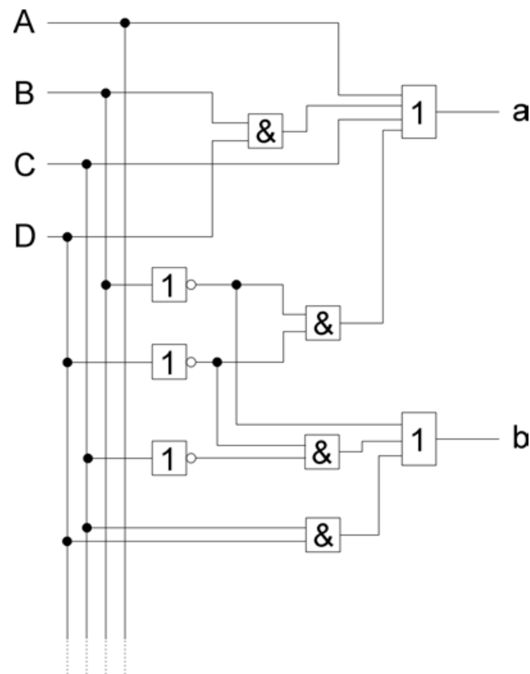
$$f = A + B\overline{C} + B\overline{D} + \overline{C}\,\overline{D} \qquad (82)$$

g

AB \ CD	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	-	-	-	-
10	1	1	-	-

$$g = B\overline{C} + A + C\overline{D} + \overline{B}C \qquad (83)$$

Gyakorlásként felrajzolhatjuk néhány szegmens logikai kapcsolási vázlatát:



16. ábra

Készíthetünk egy kicsit okosabb hét szegmenses dekódolót is: amelyik egy vonalat jeleníttet meg, hogyha 9-nél nagyobb számot adunk a bemenetére (csak *g* világít). Az igazságtáblában eltűnnek a határozatlan értékek, helyüket 0-k illetve egyesek veszik át, növelve a függvények bonyolultságát.

	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	0	0	0	0	0	0	1
11	1	0	1	1	0	0	0	0	0	0	1
12	1	1	0	0	0	0	0	0	0	0	1
13	1	1	0	1	0	0	0	0	0	0	1
14	1	1	1	0	0	0	0	0	0	0	1
15	1	1	1	1	0	0	0	0	0	0	1

a

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	0	0	0
10	1	1	0	0

$$a = A\overline{B}\,\overline{C} + \overline{A}C + \overline{A}BD + \overline{B}\,\overline{C}\,\overline{D} \qquad (84)$$

b

AB \ CD	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	0	0	0	0
10	1	1	0	0

$$b = \overline{A}\,\overline{B} + \overline{B}\,\overline{C} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD \qquad (85)$$

c

AB \ CD	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	0	0	0	0
10	1	1	0	0

$$c = \overline{A}B + \overline{B}\,\overline{C} + \overline{A}D \qquad (86)$$

És így tovább a többi kimeneti változóval…

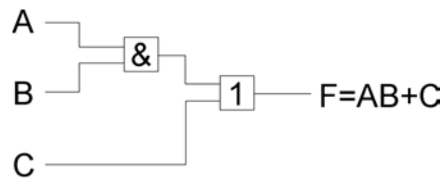
3.7 Logikai áramkörök jellemzői

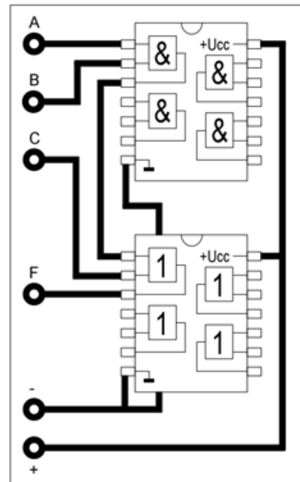
A kombinációs hálózat tervezésének eddigi lépései az alábbiak voltak:

- felírtuk az igazságtáblát,
- meghatároztuk a hálózat egyszerűsített függvényét (függvényeit),
- a függvény(ek) alapján felrajzoltuk a logikai kapcsolási vázlatot.

A logikai kapcsolási vázlat alapján akár meg is építhetjük az áramkört. Először is ki kell választanunk a megfelelő integrált áramköröket (IC-ket), és egyéb alkatrészeket. A választás többféle szempont alapján történhet: Milyen sebességgel követelünk meg a berendezésünktől? Mekkora lehet a teljesítményfelvétele? Ügyelnünk kell-e a környezetből érkező elektromos vagy egyéb zavarok hatására, stb. Az IC-ket a gyártási technológiájuk, főbb paramétereik alapján elemcsaládokba sorolják. Ha több IC-ből akarjuk összeállítani a berendezésünket, célszerű az összeset egy családból választani, mert így kompatibilitásuk (hibátlan együttműködésük) garantált. Először tehát elemcsaládot választunk, majd a családon belül keressük ki a logikai kapcsolási vázlat egyes elemeit megvalósító áramköröket. Sokszor nem tudjuk mindazt beszerezni, amire pontosan szükségünk lenne, ekkor változtatnunk kell az előzetes logikai vázlaton. Végül próbapanelen vagy nyomtatott áramköri lapon összeszereljük a kapcsolást.

A 17. ábra egy nagyon egyszerű függvény nyomtatott áramköri megvalósítására mutat példát. Két alacsony integráltságú (kevés áramköri elemet tartalmazó) IC-t használtunk fel, amelyeket katalógusból választottunk ki. Ha végigkövetjük az összeköttetéseket, felismerhetjük, hogy az elméleti és a gyakorlati kapcsolás tulajdonképpen ugyanaz; a különbség annyi, hogy a megépített áramkörben tényleges fizikai mennyiségek jelennek meg a be- és kimeneteken. Hogy mik ezek a fizikai jellemzők, és milyen értékekkel bírnak, erről szólnak a most következő fejezetek.



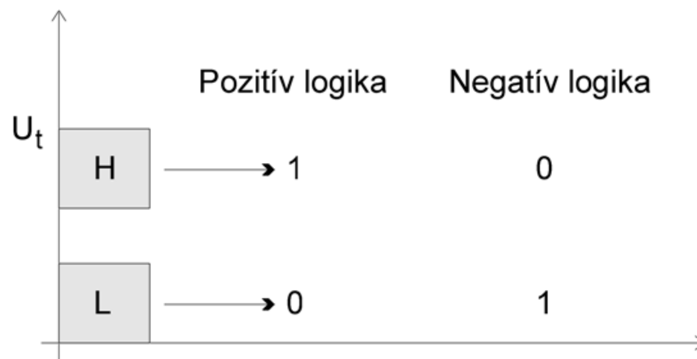


17. ábra

3.7.1 Integrált áramkörök feszültségszintjei, DC zajtávolság

Az elektronikus IC-k feszültségszintek alapján különböztetik meg a logikai (0-s vagy 1-es) értékeket. A gyártók elemcsaládonként specifikálják azokat a feszültségtartományokat, amelyeket az áramkörök logikai nullának illetve logikai egyesnek értelmeznek a bemenetükön. Az egyik tartomány a nulla Volthoz közeli: ezt *L* (low, alacsony) szintnek hívjuk. A másik tartomány a tápfeszültséghez közelít: ez a *H* (high, magas) szint. A feszültségszintek és a logikai értékek egymáshoz rendelése kétféleképpen történhet (18. ábra):

- *pozitív logika* szerint: az L szint a logikai nullát, a H szint a logikai egyest jelöli,
- *negatív logika* szerint: az L szint a logikai egyest, a H szint a logikai nullát jelöli.



18. ábra

Látható, hogy az L és H feszültségszintek között van egy közbenső tartomány is. Ha ilyen feszültségű jel érkezik a bemenetre, az IC logikai működése bizonytalan lesz.

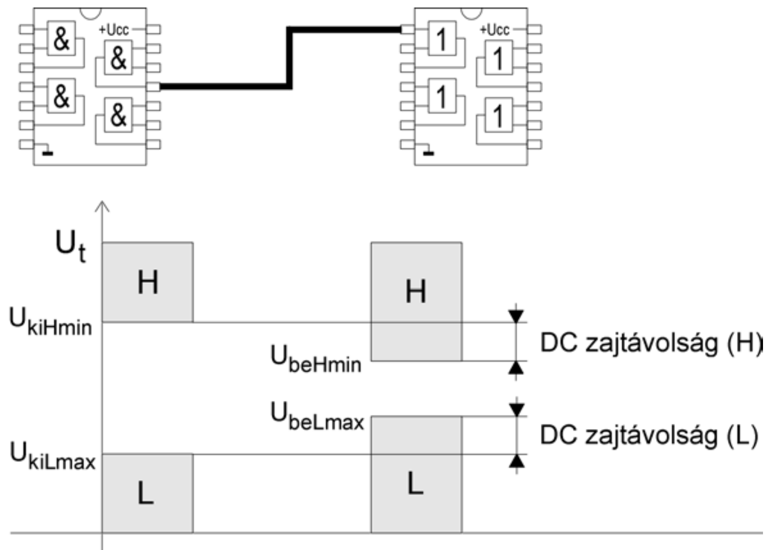
A logikai áramkörök kimenetein olyan feszültségű jelnek kell megjelennie, hogy a rájuk kapcsolt áramkörök azt *egyértelműen* L vagy H szintűnek értelmezzék. Ha ugyanazon elemcsaládból választjuk az IC-ket, akkor – normális működési feltételek között – nem fordulhat elő tévedés.

A vezetékeken továbbterjedő jeleket a külvilágból érkező zajok módosíthatják. Ahhoz, hogy egy zajjal terhelt jel értelmezése is hibátlanul történjen, a gyártók a kimeneti L és H tartományokat egy kicsivel „szűkebbre" veszik (19. ábra). Így, ha például a kimeneten megjelenő jel a H szint alsó határán van, akkor a rajzon jelölt *DC zajtávolság* mértékén belül ingadozó jelet is hibátlanul azonosítja a hozzá csatolt áramkör a bemenetén. A helyes működés feltételei tehát az alábbiak:

$$U_{kiH\min} < U_{beH\min}, \tag{87}$$

$$U_{kiL\max} > U_{beL\max}. \tag{88}$$

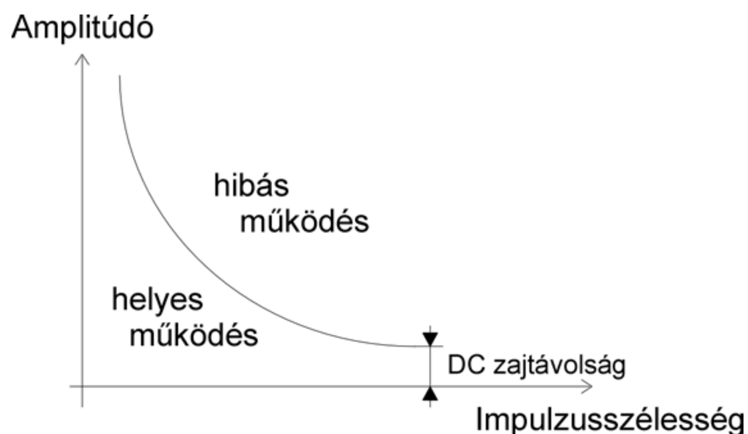
Az áramkörcsalád katalógusában megjelenő DC zajtávolság az L és H tartományoknál megengedhető zajszintek közül a kisebbik. A *DC* rövidítés a *Direct Current*-re, magyarul egyenáramra utal, mivel ez a zaj akár hosszú ideig, a jelhez állandó mértékben hozzáadódva is fennállhat.



19. ábra

3.7.2 AC (váltakozó áramú) zajtávolság

Előfordulhat, hogy egy DC zajtávolságúnál jóval nagyobb amplitúdójú impulzusszerű zaj nem okoz hibát az áramkör működésében. Ez azért lehetséges, mert a zavaró jelnek bizonyos energiatartalommal kell rendelkeznie ahhoz, hogy hatása legyen. Egy impulzus energiája az alatta levő területtel, tehát a szélességének és a magasságának a szorzatával arányos, így ha nagyon rövid ideig áll fent a jel, nagyobb amplitúdó is megengedhető (20. ábra). Ha a jel bizonyos időtartamon túl is fennáll, egyenáramú (DC) jelnek tekinthető, amplitúdóját a DC zajtávolság korlátozza.

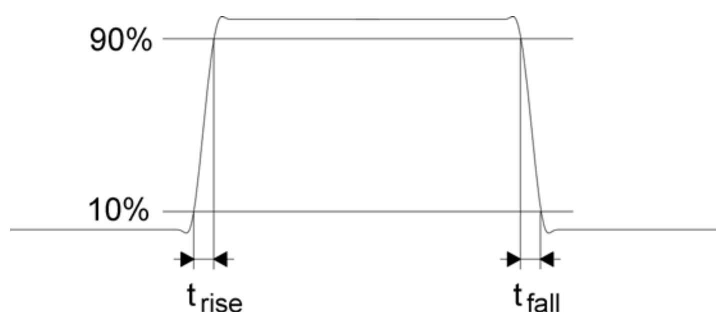


20. ábra

3.7.3 Logikai áramkörök dinamikus jellemzői

3.7.3.1 Felfutási és lefutási idő

A kimeneten megjelenő jel változási sebessége nem végtelen. A $0 \rightarrow 1$ illetve az $1 \rightarrow 0$ átmenetek jellemzői a *felfutási* és *lefutási idő*k (az angol terminológiában t_{rise} és t_{fall}). A változás kezdeténél és végénél megjelenő apró hullámok miatt a mérésüket az L és H szintek közötti feszültségkülönbség 10%-a és 90%-a között végezzük (21. ábra).

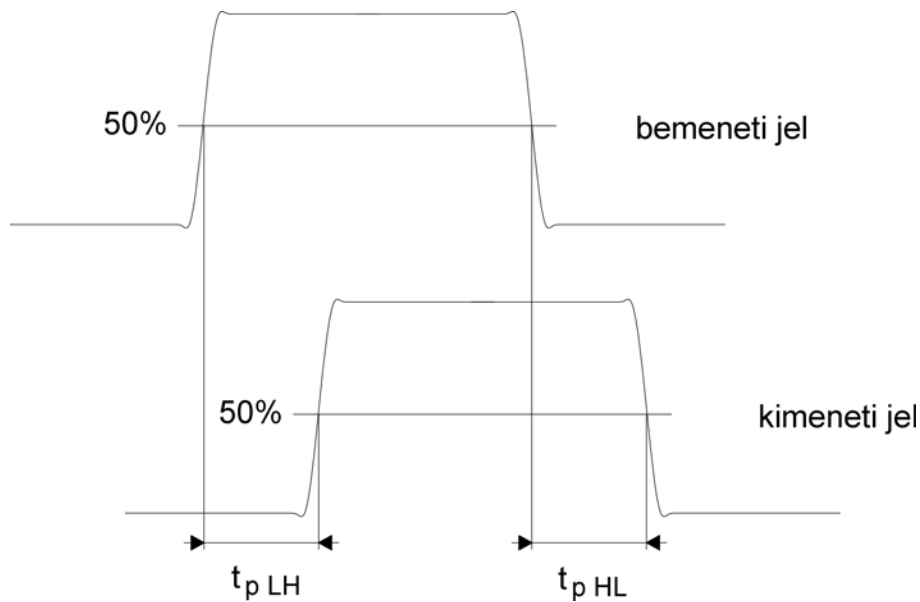


21**. ábra**

3.7.3.2 Késleltetési vagy terjedési idő

Egy adott logikai feladat elvégzéséhez mindig bizonyos időre van szüksége az áramkörnek, ezért a bemeneti jel változását csak kissé késve követi a

kimeneti jel esetleges változása. Ennek mérőszáma a *késleltetési* vagy *terjedési idő* (angolul *propagation delay*). Mérését a 22. ábra szerint végezzük. A késleltetési idő nagyságát a kimenetre kapcsolt impedancia (főként kapacitás) is befolyásolja.



22. ábra

3.7.3.3 Terhelhetőség, Fan-out

Minden egyes áramkörcsaládra meg van adva, hogy egy logikai elem kimenetére legfeljebb hány bemenet csatlakoztatható (*Fan-out*). Ezt a kimeneten leadott illetve a bemeneteken felvett áramok nagysága határozza meg. Egy adott család áramkörei például a következő áramértékeket képesek előállítani a kimenetükön az L és H szinteken:

$$I_{kiL\max} = 16mA\,, \tag{89}$$

$$I_{kiH\max} = 0{,}4mA\,, \tag{90}$$

míg a működtetésükhöz igényelt bemeneti áramok:

$$I_{beL} = 1{,}6mA\,, \tag{91}$$

$$I_{beH} = 40\mu A\,. \tag{92}$$

Ezek alapján az egy kimenetre köthető bemenetek maximális száma L és H szinten:

$$Fan - out_L = \frac{I_{kiL\max}}{I_{beL}} = 10 \text{, illetve} \quad (93)$$

$$Fan - out_H = \frac{I_{kiH\max}}{I_{beH}} = 10 \,. \quad (94)$$

Az eredő terhelhetőség a kettő közül a rosszabbik érték (jelen esetben a két szám megegyezik):

$$Fan - out = 10 \,. \quad (95)$$

A terhelhetőség növelése az ugyanolyan logikai működésű áramkörök párhuzamos kapcsolásával oldható meg.

3.7.3.4 Egység-terhelés, Fan-in

Az áramkörcsaládban előfordulhatnak a család többi tagjától eltérő bemeneti áramú elemek is. Az egység-terhelés azt adja meg, hogy az adott elem áramfelvétele hányszorosa a család többi tagja áramfelvételének.

3.7.3.5 Egyéb jellemzők

A logikai áramköröknek még sok olyan jellemzője van, amelyeket a tervezésnél figyelembe kell venni. Most a legfontosabbakat soroljuk föl:

- *Disszipáció (teljesítményfelvétel)*: az a teljesítmény, amely hővé alakul, ha a logikai áramkört 50% kitöltésű tényezőjű órajellel kapcsolgatjuk (vagyis olyan váltakozó jellel, amely az L és H szinteken ugyanannyi ideig tartózkodik). A disszipáció kisebb-nagyobb mértékben frekvenciafüggő.
- *Jósági tényező*: az átlagos késleltetési idő és a disszipáció szorzata. Két logikai áramkör közül az a jobb, amelyik ugyanolyan teljesítményfelvétel mellett gyorsabb, illetve azonos sebességnél kevesebb energiát fogyaszt: tehát kisebb a jósági tényezője.
- *A megengedett legnagyobb és legkisebb be – és kimeneti feszültségszintek, tápfeszültségszintek.*
- *Tápfeszültség-tolerancia*: a tápfeszültség legnagyobb megengedhető ingadozása százalékban kifejezve.

- A normális működéshez előírt *hőmérsékleti tartomány*.
- *Tokozás*: A lábak furatba illeszthetők, vagy felületre forraszthatók-e, a tok műanyag, esetleg hőálló kerámia, stb.

3.7.4 A TTL és CMOS elemcsaládok összehasonlítása

A logikai áramkörök két leggyakrabban használt típusa a bipoláris tranzisztorokból felépülő TTL és a térvezérlésű tranzisztorokból álló CMOS család. Mindkét típus különböző kivitelekben kapható, amelyek disszipációja és késleltetési ideje eltérő. Lássuk, mely paraméterek szólnak az egyik illetve a másik alkalmazása mellett!

A TTL áramkörök előnye (a CMOS-sal szemben) a gyorsaság. Késleltetési idejük kapunként $10ns$ körüli standard kivitelben, de léteznek $1-2ns$ terjedési idejű változatok is. Teljesítményfelvételük típustól függően néhány mW-tól néhány 10 mW-ig (standard) terjed. Stabil áramforrásra van szükségük: a polgári célra készített darabok igénye $5V \pm 5\%$, a katonai kivitel $5V \pm 10\%$-al elégszik meg.

A CMOS áramkörök lassabbak: a késleltetési idő standard kivitelben $100ns$ körüli, ami $10ns$ közeli értékekre csökkenthető (High Speed kivitel). Fő előnyük az alacsony teljesítményfelvétel: mivel a térvezérlésű tranzisztorok bemeneti ellenállása igen nagy, a CMOS áramkörök nyugalmi állapotban gyakorlatilag alig fogyasztanak energiát (néhány μW-ot). Emiatt elemes táplálású eszközökben (digitális órák, számológépek, távvezérlők, stb.) használjuk őket. Tényleges teljesítményfelvétel a logikai átkapcsoláskor történik, ezért a disszipáció teljesítményfüggő. Jellemző értéke $1\mu W / kHz$ körüli. Ez nagyjából $1MHz$ felett meghaladja a TTL áramkörök teljesítményfelvételét, így ebben a tartományban már nem célszerű az alkalmazásuk. A CMOS technológia másik nagy előnye a széles tápfeszültség-tartomány, amely az $5V \pm 5\%$-al szemben 3-tól 15V-ig(!) terjed. Ez újabb érv ahhoz, hogy elemmel működő eszközök tervezésénél elsősorban a CMOS technológiára gondoljunk.

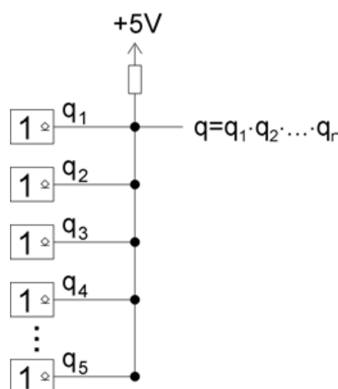
E két elemcsalád mellett természetesen még számtalan különböző logikai áramkörtípussal találkozhatunk (elég csak megnézni valamelyik nagyobb IC gyártó honlapját). A digitális technika szédületes iramú térnyerésének az alkatrész-technológia rohamos fejlődése adja az alapját.

3.7.5 Speciális kimeneti kapcsolatok TTL áramköröknél

Logikai rendszerek tervezésénél sokszor előfordul, hogy nagyon sok kimenetet kell egyetlen ÉS, esetleg VAGY kapu bemeneteire kötni. Ilyenkor rengeteg vezetéket kell egymással párhuzamosan egy kicsiny területre vezetni, ami jelentősen bonyolítja az áramkör tervezését, de a nagyszámú bemenettel rendelkező kapuk kialakítása is gondot okoz. Szerencsére a TTL áramkörcsaládok két olyan megoldást is kínálnak, amellyel kiküszöbölhetők ezek a problémák:

3.7.5.1 Nyitott kollektoros kimenetek

Léteznek olyan TTL technológiával gyártott logikai kapuk, amelyek *nyitott kollektorú (open collector) kimenet*tel rendelkeznek. Ha több ilyen kapu kimenetét a 23. ábra szerint összekötjük, a közös ponton ún. *huzalozott ÉS kapcsolat* jön létre, megspórolva így a sok bemenetű ÉS kaput, és a párhuzamos vezetékeket.

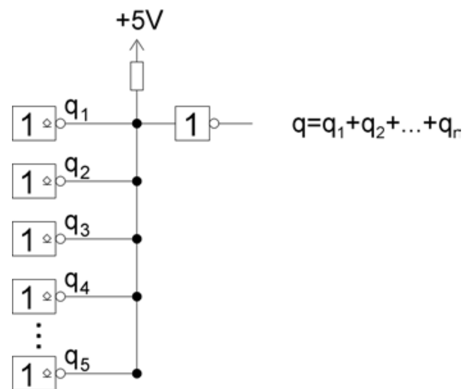


23. ábra

Mivel a De-Morgan azonosság szabályai szerint:

$$q_1 + q_2 + \ldots + q_n = \overline{\overline{q_1} \cdot \overline{q_2} \cdot \ldots \cdot \overline{q_n}}\ , \qquad (96)$$

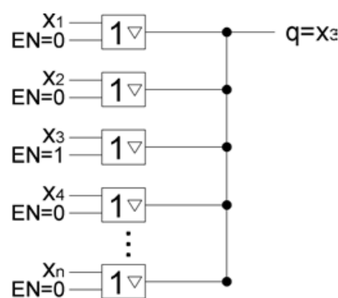
a 24. ábrának megfelelően *huzalozott VAGY* kapcsolat is előállítható.



24. ábra

3.7.5.2 Háromállapotú (tristate) kimenetek

Számítógépes rendszerekben sokszor használunk ún. *buszrendszereke*t. A buszok olyan vezetékek vagy vezetékkötegek, amelyeken több eszköz is osztozik, de egyidőben csak egyetlen, kiválasztott áramkör használhatja. Ilyenkor háromállapotú kimenetekkel rendelkező áramköröket alkalmazunk. Ezeknél az elemeknél egy járulékos vezérlőbemenettel „kikapcsolhatók” a kimenetek (nagy impedanciás módba állíthatók), megengedve egy másik eszköznek, hogy a vezetékeken adatokat továbbítson (25. ábra). Mindig ügyelni kell arra, hogy egy buszon egyszerre csak egy eszköz legyen aktív!



25. ábra

3.8 A jelterjedési idők hatása a kombinációs hálózatok működésére

A kombinációs hálózatok tervezésénél ideális áramköri elemekkel dolgoztunk, ám az előző részből kiderült, hogy a valóságban a kapuk és vezetékek jelterjedési késleltetése nem elhanyagolható. Ebben a fejezetben látni fogjuk, hogy a késleltető hatások átmenetileg hibás kimeneti kombinációkat hozhatnak létre. A hibák előfordulása a környezeti változóktól: hőmérséklet, öregedés, stb. függhet, így előzetesen nem vehetők számításba. Az ilyen véletlenszerű, rendszertelen hibajelenségeket *hazárdjelenségek*nek nevezzük. Tervezéskor arra kell figyelnünk, hogy a kombinációs hálózat működése a lehető legnagyobb mértékben független legyen a késleltetési viszonyok alakulásától.

3.8.1 A statikus hazárd

Vizsgáljuk meg ugyanazon kombinációs hálózat működését először ideális, majd bizonyos késleltetéssel rendelkező logikai kapukat feltételezve. A hálózat igazságtáblája az alábbi legyen:

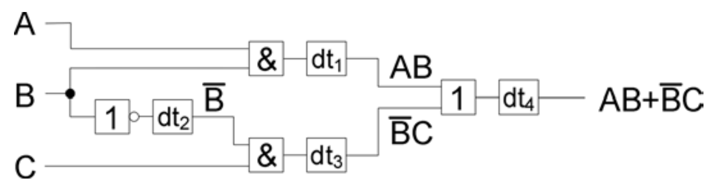
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Az egyszerűsített függvényt Karnaugh-táblával határozzuk meg:

AB \ C	0	1
00	0	1
01	0	0
11	1	1
10	0	1

$$F = AB + \overline{B}C\ . \tag{97}$$

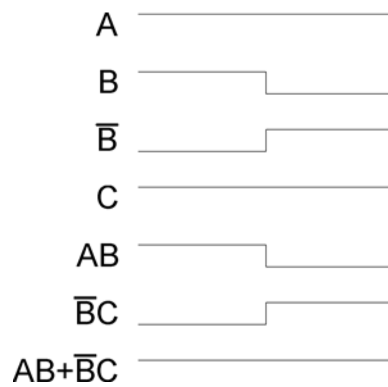
Ezek után rajzoljuk fel a kapcsolási vázlatot. A jelterjedési idők hatását modellezzük az ideális kapuk mögé kötött késleltetőelemekkel (26. ábra).



26. ábra

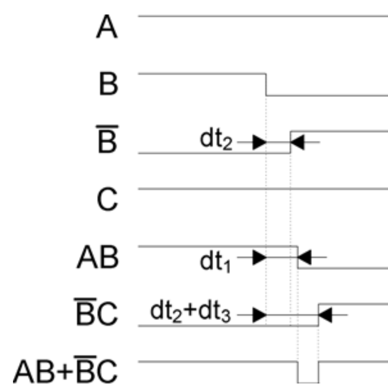
Most írjuk fel az áramkör működését egy idődiagramon úgy, hogy az A és C bemenetekre stabil 1 értéket adunk, a B bemeneten pedig $1 \to 0$ átmenet történik. Hogy könnyebb legyen követni a változásokat, a diagramba az A, B és C jeleket, B negáltját, a VAGY kapu bemenetein mért jeleket, és végül annak kimenetén mért jelet is rajzoljuk be. A VAGY kapu késleltetési idejét ne vegyük figyelembe, mert az a mérés eredményét érdemileg nem befolyásolja (csak bizonyos mértékben eltolja).

Először az ideális esetet vizsgáljuk, ahol az összes késleltetési idő zérus (27. ábra). Az eredmény a vártnak megfelelő stabil 1-es kimenet.



27. ábra

Azonban ha számításba vesszük a késleltetéseket, az áramkör nem úgy működik, ahogy azt elvárnánk: az állandó 1-es helyett $1 \to 0 \to 1$ kimeneti jelváltozást észlelünk (28. ábra). *Statikus hazárd* lépett fel a hálózatban. Általánosságban akkor beszélünk statikus hazárdról, ha valamely bemeneti kombinációról egy szomszédos bemeneti kombinációra ugrunk (vagyis *egyetlen* bemenet értékét megváltoztatjuk), és a kívánt stabil kimenet helyett egy impulzust kapunk.



28. ábra

Gondoljuk végig, hogy mi történt! Amíg a B bemenet logikai magas szinten volt, a felső ÉS kapu biztosította az 1-es értékű kimenetet, majd B változásakor az alsó kapu vette át ezt a szerepet. A baj abból adódott, hogy a felső kapu előbb „kapcsolódott ki”, mint ahogy az alsó kapu be.

Vajon felfedezhető-e, és ami a legfontosabb: kiküszöbölhető-e a hiba már a tervezés szakaszában? Vegyük elő újra a Karnaugh-táblát!

AB \ C	0	1
00	0	1
01	0	0
11	1	1
10	0	1

Tudjuk, hogy a Karnaugh-táblán felvett hurkok egy-egy ÉS kapcsolatot, végeredményben ÉS kaput eredményeznek. A mérés során az 111 bemeneti kombinációról a szomszédos 101-re váltottunk, s ha jól megfigyeljük, ezzel az egyik hurokból épp átugrottunk a másikba; vagyis könnyedén felfedezhető, ha az egyik ÉS kapu szerepét egy másik veszi át.

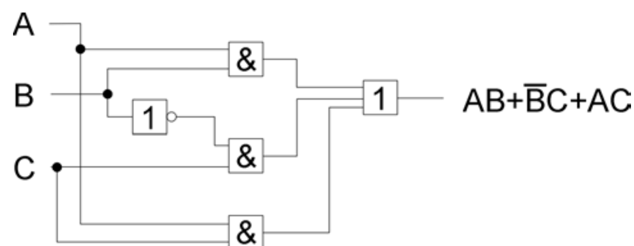
AB \ C	0	1
00	0	1
01	0	0
11	1	1
10	0	1

Mindezt végiggondolva a statikus hazárd kiküszöbölése is egyszerű: a szomszédos hurkok közötti kritikus átmenetet is le kell fednünk egy újabb hurokkal.

AB \ C	0	1
00	0	1
01	0	0
11	1	1
10	0	1

$$F = AB + \overline{B}C + AC\,. \tag{98}$$

Az egyszerűsített függvény, így a kapcsolási vázlat is kicsit bonyolultabb lett, mivel beiktattunk még egy ÉS kaput, amely biztosítja a stabil működést (29. ábra).



29. ábra

A statikus hazárdmentesítés tehát abból áll, hogy minden szomszédos hurkot összekötünk egy további hurokkal. Egy hálózat akkor és csak akkor mentes a statikus hazárdtól, ha bármely két szomszédos mintermet lefed egy-egy primimplikáns – magyarul a Karnaugh-táblában bármely két szomszédos 1-est lefed egy-egy hurok.

3.8.1.1 Példa statikus hazárdmentesítésre

Írjuk fel egy négybemenetű hálózat egyszerűsített függvényét, majd a statikus hazárdtól mentes függvényt! A rendszer igazságtáblája a következő:

A	B	C	D	P
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Az egyszerűsített függvény a Karnaugh-tábla alapján:

AB \ CD	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	1	0	0	0
10	1	0	1	1

$$Q = \overline{A}B + A\overline{C}\,\overline{D} + \overline{B}C \tag{99}$$

Hazárdmentesítve pedig:

AB \ CD	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	1	0	0	0
10	1	0	1	1

$$Q = \overline{A}B + A\overline{C}\,\overline{D} + \overline{B}C + B\overline{C}\,\overline{D} + \overline{A}C + A\overline{B}\,\overline{D} \tag{100}$$

3.8.1.2 Statikus hazárdok konjunktív hálózatokban

A Maxtermes alakban történő egyszerűsítésnél a nullákat fedtük le hurkokkal a Karnaugh-táblában, és a De-Morgan azonosságok alkalmazásával írtuk fel a végeredményt. A hurkok VAGY kapukat reprezentálnak, és elmondható, hogy a statikus hazárd lehetősége itt is fennáll: egyik hurokból a másikba átugorva stabil zérus kimenet helyett egy impulzus jelenik meg. A hazárdmentesítés a diszjunktív hálózatokéhoz hasonló. Például az alábbi grafikus egyszerűsítésnél:

AB \ C	0	1
00	1	1
01	1	0
11	0	0
10	0	1

$$Q = (\overline{A} + C)(\overline{B} + \overline{C}), \tag{101}$$

ugyanez hazárdmentesítve:

$$Q = (\overline{A} + C)(\overline{B} + \overline{C})(\overline{A} + \overline{B}). \tag{102}$$

3.8.2 A dinamikus hazárd

*Dinamikus hazárd*ról akkor beszélünk, ha egyetlen bemenet átállításakor egy egyszerű kimeneti jelváltozást várnánk, de ehelyett egy impulzussal tarkított jelváltozást tapasztalunk (30. ábra). Dinamikus hazárd csak abban a hálózatban alakulhat ki, amelynek valamelyik részéből nem küszöböltük ki a statikus hazárdot – a jelenséget ugyanis a statikus hazárd okozta impulzus idézi elő.

30. ábra

3.8.3 A funkcionális hazárd

Eddig azt vizsgáltuk, hogy mi történik a szomszédos bemeneti kombinációk közötti ugrásoknal. Most nézzük meg azt az esetet, ha két bemenetet változtatunk meg egy időben! A probléma abból fog adódni, hogy a valóságban két jel soha nem változik egyszerre, egy nagyon kicsi eltérés biztos van a változásuk időpontja között. Legyen egy kombinációs hálózat Karnaugh-táblája:

AB \ C	0	1
00	1	1
01	1	0
11	0	0
10	0	0

A kiindulási bemeneti kombináció legyen 010, majd egyszerre kapcsoljuk át B-t és C-t, 001-t előállítva! A kimenet mindkét bemeneti kombináció esetén 1-es, így elméletben végig ezen az értéken kéne maradnia. Vajon tényleg így lesz-e? A valóságban két eset lehetséges: vagy B változik előbb, vagy C:

- Amennyiben előbb B, majd C változik, a tényleges bemeneti változás: $010 \rightarrow 000 \rightarrow 001$. Az ennek megfelelő kimeneti változást kiolvashatjuk a Karnaugh-táblából, a nyíl irányában haladva: $1 \rightarrow 1 \rightarrow 1$. Nem történt hibajelenség, a kívánt értékeket kaptuk.

AB \ C	0	1
00	1	1
01	1	0
11	0	0
10	0	0

- Abban az esetben viszont, ha a változás sorrendje C majd B, a valós bemeneti változás $010 \rightarrow 011 \rightarrow 001$ lesz. Az ennek megfelelő kimeneti változás: $1 \rightarrow 0 \rightarrow 1$. Stabil 1-es helyett egy (negatív) impulzust kaptunk. A hálózatban *funkcionális hazárd* lépett fel.

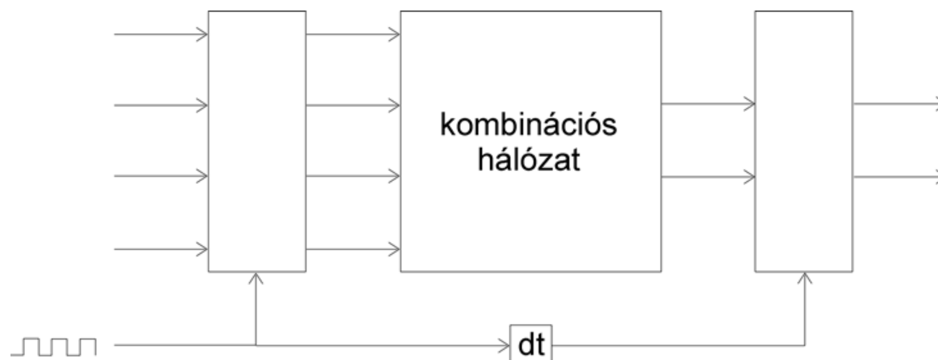
AB \ C	0	1
00	1	1
01	1	0
11	0	0
10	0	0

Hogyan lehetne kiküszöbölni a problémát? Megoldást jelenthetne, ha késleltetések beiktatásával mindig B-t engednénk előbb változni. Ám gondoljuk végig: ha visszafelé is elvégeznénk a mérést, 001 kombinációról

010-ra ugorva, éppen B korai változása okozna funkcionális hazárdot. Ráadásul a következő táblával adott hálózaton a két minterm között már el sem juthatunk úgy, hogy ne érintenénk egy 0-t:

AB \ C	0	1
00	0	1
01	1	0
11	0	0
10	0	0

A funkcionális hazárdot szinkronizációval lehet megszüntetni (31. ábra). Ilyenkor a kombinációs hálózat elé és mögé olyan szinkronizáló elemeket teszünk (ún. léptető regisztereket, róluk a későbbiekben lesz szó), amelyek csak bizonyos időközönként engedik a kombinációs hálózat bemeneteire a jeleket, megvárják, míg a hazárdok eltűnnek a hálózatból, és csak ezután jelenítik meg a kimeneteken a változásokat. (A léptetés egy periodikus órajelhez szinkronizálva történik, ezért az elnevezés.) Ez a fajta megoldás csökkenti a hálózat gyorsaságát.



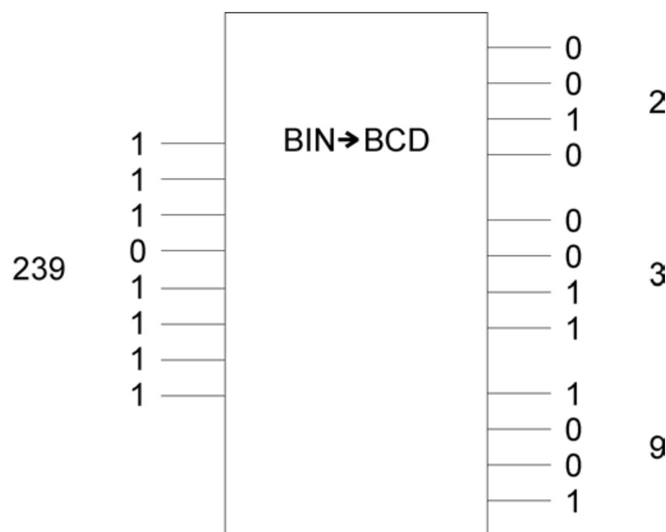
31. ábra

3.9 Néhány gyakrabban használt kombinációs hálózat

A következőkben olyan, kombinációs hálózatból felépülő áramköröket mutatunk be, amelyekkel sokszor találkozhatunk a legkülönbözőbb digitális eszközökben. Gyakori alkalmazásuk miatt IC-be integrálva, készen is megvásárolhatjuk őket.

3.9.1 Kódátalakító egységek

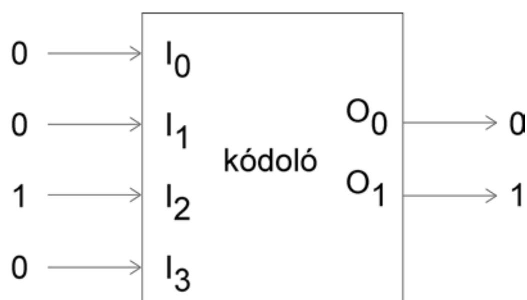
A digitális technikában többféle kódot használnak. Láttuk, hogy a számokat csak kettes számrendszerben, *bináris kód*ban tudjuk ábrázolni, tárolni vagy továbbítani. Ha mégis tízes számrendszerbeli számokkal akarunk műveleteket végezni, a *BCD (Binary Coded Decimal)* kódot választhatjuk, ahol a szám minden egyes számjegyét külön-külön, kettes számrendszerben írjuk fel. Például a 239-es szám számjegyei 2-es számrendszerben: 2=0010; 3=0011; 9=1001, ezeket egymás után írva megkapjuk a szám BCD kódját: 001000111001. A kódot újra négyes csoportokra bontva gyakorlatilag 10-es számrendszerben dolgozhatunk. A *kódátalakító egységek* egyfajta kódból egy másikba alakítanak. A 32. ábrán egy binárisból BCD kódba átalakítót láthatunk. Kódátalakító egység a már megismert hétszegmenses dekódoló is.



32. ábra

3.9.2 Kódoló

A köznapi nyelvben *kódoló*nak hívják azt a kódátalakító eszközt, melynek bemenetei közül csak az egyiken szerepelhet 1-es, míg a többi 0 (ezt *„1 az N-ből” kód*nak hívják), a kimenetein pedig annak a bemenetnek a száma jelenik meg binárisan, ahol ez az 1-es van. A 33. ábrán látható kódoló 2. bemenetére raktunk 1-est, így a kimeneteken 2, tehát 10 jelent meg (a kisebb indexű be- és kimenetek a kisebb helyiértékeket jelölik).



33. ábra

Az *n* számú kimenettel rendelkező kódolónak 2^n bemenete van. Írjuk fel a 4 bemenetű kódoló igazságtábláját! Mivel csak négy bemeneti kombinációt fogadunk el, a többi sorban nem definiáljuk a kimenetek értékét.

I_3	I_2	I_1	I_0	O_1	O_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
az összes többi kombinációra:				-	-

Ebből könnyen megtervezhetjük az áramkör logikai kapcsolási rajzát:

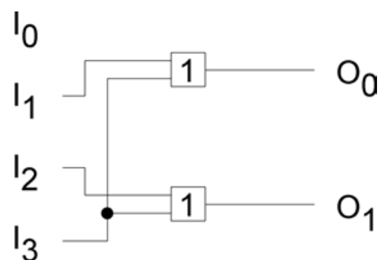
O_0 (I_3I_2 \ I_1I_0)	00	01	11	10
00	-	0	-	1
01	0	-	-	-
11	-	-	-	-
10	1	-	-	-

$$O_0 = I_1 + I_3 \tag{103}$$

O_1 (I_3I_2 \ I_1I_0)	00	01	11	10
00	-	0	-	0
01	1	-	-	-
11	-	-	-	-
10	1	-	-	-

$$O_1 = I_2 + I_3 \tag{104}$$

Mindez rajzban:

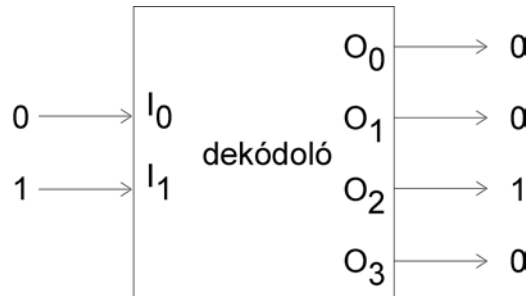


34. ábra

A kapcsolás érdekessége, hogy I_0-ra nincs is szükségünk, úgymond kizárásos alapon következtetünk az értékére.

3.9.3 Dekódoló

A *dekódoló* a kódolónak éppen a fordítottja: bemeneteire egy bináris számot vár, és a számmal megegyező sorszámú kimenetre 1-est rak, míg a többire 0-t (35. ábra).



35. ábra

Igazságtáblája:

I_1	I_0	O_3	O_2	O_1	O_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

A négy kimeneti függvény:

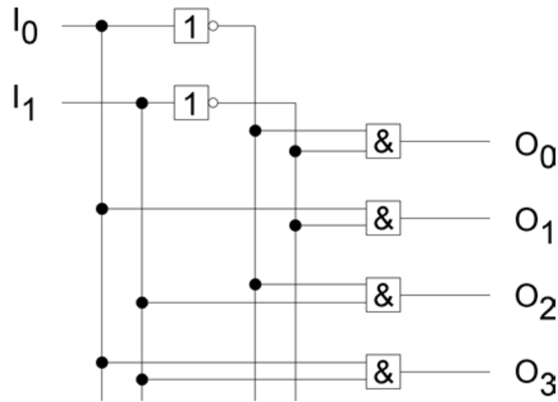
$$O_0 = \overline{I_0}\,\overline{I_1}, \tag{105}$$

$$O_1 = I_0\overline{I_1}, \tag{106}$$

$$O_2 = \overline{I_0}I_1, \tag{107}$$

$$O_0 = I_0 I_1. \tag{108}$$

Végül a logikai vázlat:

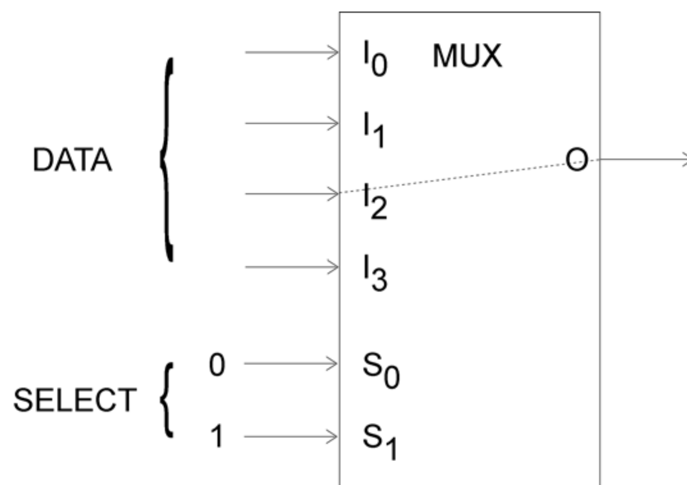


36. ábra

3.9.4 Adatút-választó eszközök

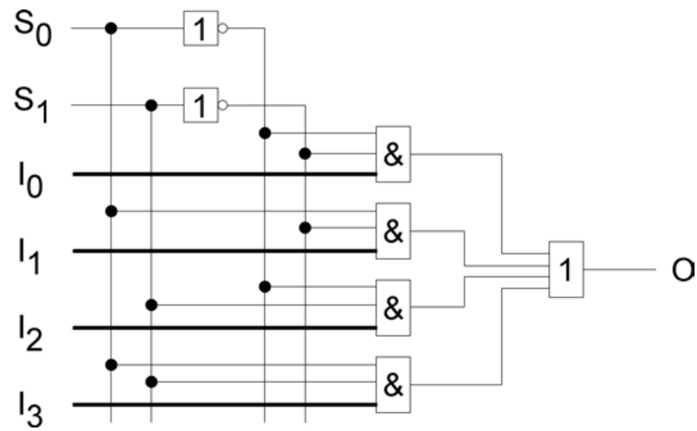
3.9.4.1 Multiplexer

A *multiplexer* (rövidítve MUX) a SELECT bemenetekre egy bináris számot vár, és az evvel megegyező sorszámú adatbemenetet összeköti a kimenettel. A 37. ábra egy négy adatbemenettel bíró eszközt mutat. *n* számú SELECT bemenet értelemszerűen 2^n darab adatbemenet közül választhat.



37. ábra

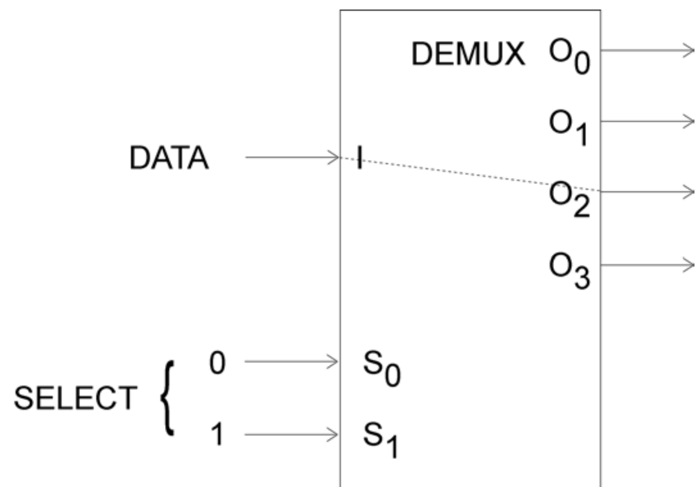
Logikai kapcsolási vázlatát a dekódoló rajzát kiegészítve nyerjük:



38. ábra

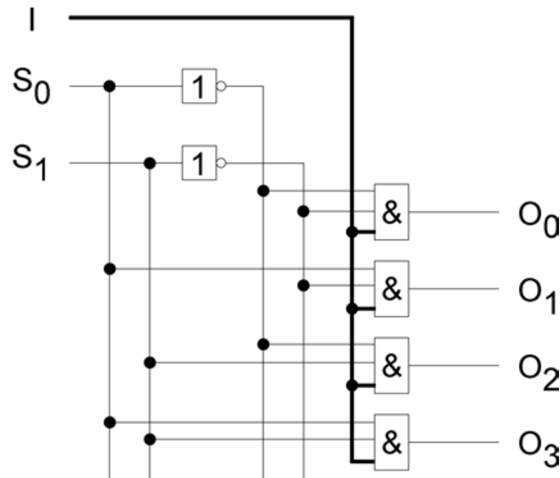
3.9.4.2 Demultiplexer

A *demultiplexer*nek egyetlen adatbemenete van, s ezt a SELECT bemenetek által kiválasztott kimenettel köti össze.



39. ábra

Megvalósítása a dekódoló kapcsolási vázlatának felhasználásával:



40. ábra

3.9.5 Bináris aritmetikát végző áramkörök

Ha kettes számrendszerbeli számokon szeretnénk matematikai műveleteket végezni, nem kell azonnal valamilyen intelligens aritmetikai egység után néznünk. Az egyszerűbb műveleteket kombinációs hálózatok is elvégezhetik. Rengeteg ilyen eszközzel találkozhatunk, a kivonótól a szorzó áramkörökig – ezek közül mutatunk most be egyet a példa kedvéért.

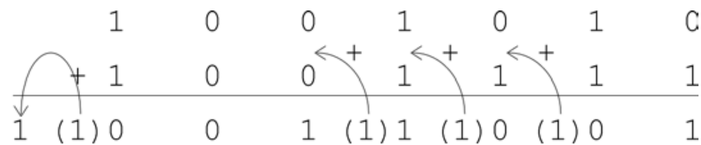
3.9.5.1 1 bites teljes összeadó

Legelőször is nézzük meg, hogyan adunk össze két bináris számot! A feladatot a tízes számrendszerben végzett összeadás analógiájára oldhatjuk meg. Ha papíron végezzük a műveletet, jobbról balra haladva sorra összeadjuk az egyes számjegyeket, és ahol kilencnél nagyobb eredményt kapunk, ott hozzáadjuk a maradék egyest az egyel nagyobb helyiértékű számjegyekhez (41. ábra).

```
    2   5    6
          +
+   3   1    7
--------------
    5   7 (1)3
```

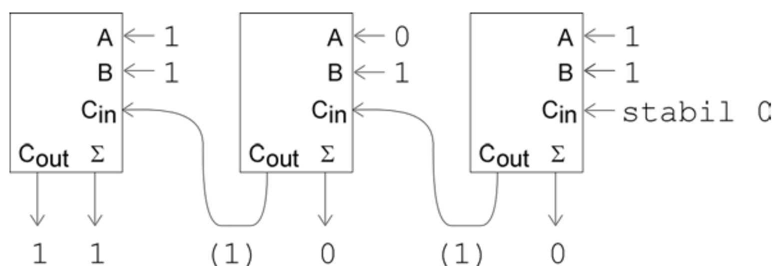
41. ábra

Kettes számrendszerben pontosan ugyanígy járunk el, csak a felhasználható számjegyek a 0 és az 1, és akkor történik a következő helyiértékre átvitel, ha az eredmény nagyobb, mint 1 (42. ábra).



42. ábra

Az *egy bites teljes összeadó* két bináris szám egy-egy bitjét (a 42. ábra egy oszlopát) adja össze úgy, hogy figyelembe veszi az egyel kisebb helyiértékről érkező átvitelt, és ha nála is keletkezik átvitel, akkor továbbítja azt. Az átvitel fogadására és továbbítására egy-egy be- illetve kimenetet használ. Mindennek az az eredménye, hogy több egy bites teljes összeadót összekapcsolva – ha a 43. ábrán látható elrendezést követjük –, két bármilyen hosszú bináris számot össze tudunk adni.



43. ábra

Ahogy a rajz is mutatja, az egy bites teljes összeadó egy 3 bemenetű és 2 kimenetű kombinációs hálózattal valósítható meg. A Σ kimenet akkor 1, ha egyetlen, vagy mindhárom bemenet 1-es (vö. a 42. ábrával). Átvitel pedig akkor keletkezik (C_{out}=1) , hogyha egynél több bemenet 1-es. Ezek alapján felírhatjuk az igazságtáblát:

A	B	Cin	Σ	Cout

0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A két kimeneti függvényt grafikus egyszerűsítéssel próbáljuk meghatározni:

Σ

AB \ C_{in}	0	1
00	0	1
01	1	0
11	0	1
10	1	0

Bár erről még nem volt szó, az ilyen sakktábla-szerű Karnaugh-tábla antivalencia kapukat jelez:

$$\Sigma = A \oplus B \oplus C_{in} . \quad (109)$$

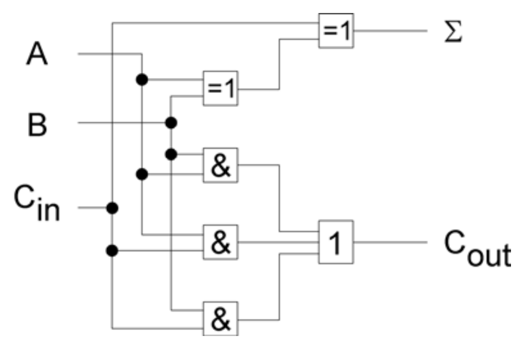
A másik kimenetre:

C_{out}

AB \ C_{in}	0	1
00	0	0
01	0	1
11	1	1
10	0	1

$$C_{out} = AB + AC_{in} + BC_{in} . \quad (110)$$

A függvényeket megvalósító logikai kapcsolási vázlat a 44. ábrán látható.



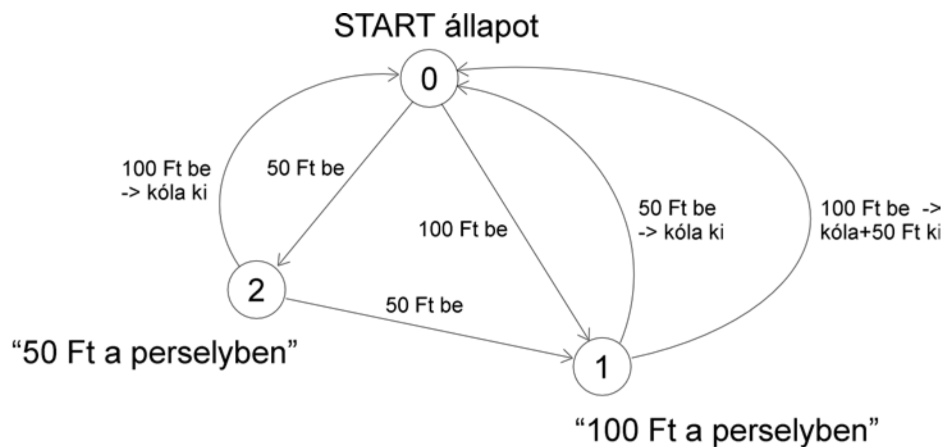
44. ábra

4. Szekvenciális (sorrendi) hálózatok

A kombinációs hálózatok után most a logikai rendszerek másik fő osztályáról lesz szó: a *szekvenciális*, más néven *sorrendi hálózatok*ról. Tudjuk, hogy a kombinációs hálózatoknál a kimenetek értékét mindig az *aktuálisan* fennálló bemeneti kombináció határozza meg. Sorrendi hálózatoknál a kimeneti kombinációt a bemenetek *aktuális* értékei, valamint a *korábban fennállt* értékei is befolyásolják. Magyarul az áramkör „emlékszik" arra, hogy korábban mit kapcsoltunk a bemenetekre. Ilyen emlékező áramkörök létrehozásához egyáltalán nem szükségesek bonyolult memóriaelemek, sőt a logikai alapkapukon kívül sokszor nem is kell hozzájuk más! Látni fogjuk, hogy kombinációs hálózatok egyszerű visszacsatolásával is megoldható a feladat.

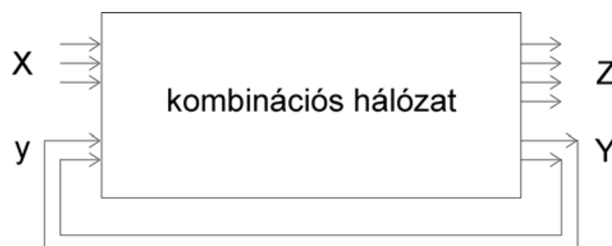
A sorrendi hálózat működési folyamata a következő: bekapcsoláskor ún. start állapotban van, a hálózatnak „előélete" nincs, várja, hogy történjen valamilyen esemény. Miután változás következik be a bemeneteken, a rendszer egy új állapotba ugrik. Innen további változások hatására újabb állapotokba ugorhatunk, vagy akár vissza is térhetünk egy korábbi helyzetbe. Ha egy sorrendi hálózat egy bizonyos állapotban van, akkor az egyértelműen megadja, hogy mi történt vele az előzőekben. Vegyünk egy példát! Egy italautomata a bekapcsolás után várja, hogy pénzt helyezzenek a bedobó nyílásba. Ez a start állapot (45. ábra). Miután valaki bedobott 100 forintot, a gép egy újabb állapotba ugrik (jelöljük ezt 1-essel). Ha az automata ebben az állapotban van, az azt jelenti, hogy 100 forint van a perselyben. Ebbe az állapotba közvetve is eljuthatunk: ha a bekapcsolás után 50 forintot dobunk be (2-es állapot), majd ismét 50 forintot, szintén az 1-es állapotba kerülünk. Ha mindezek után további 50 forintot helyezünk a nyílásba, már 150 forint van a perselyben, a gép kiadhatja a kólát, és visszatérhet a start állapotba, várva az újabb pénzérméket. Ha már 200 forintot dobtunk be, az automatának kólán kívül egy ötvenest is kell adnia. A példából jól látszik, hogy mindegyik állapot egyértelműen meghatározza a

hálózat előéletét (vagyis, hogy hány forint van a perselyben), függetlenül attól, hogy milyen úton jutottunk oda.



45. ábra

Hogyan csinálhatunk ezek alapján egy kombinációs hálózatból sorrendi hálózatot? A megoldást a 46. ábrán láthatjuk: a kimenetek egy csoportján mindig az aktuális állapot sorszámát jelenítjük meg (jelölése: Y, és *szekunder kombináció*nak nevezzük), és ezt visszacsatoljuk a bemenetekre, hogy a hálózat lássa, hogy éppen melyik állapotban van. A kombinációs hálózat kimeneteit így már nem csak a környezetből kapott bemenetek aktuális értéke (X), hanem a hálózat aktuális állapota (Y) is befolyásolhatja. Tehát megtervezhetjük, hogy egy bizonyos állapotban (y) egy bizonyos bemeneti kombinációra (X) mi legyen a környezet felé átadandó kimenetek értéke (Z), és melyik legyen a következő állapot (Y).

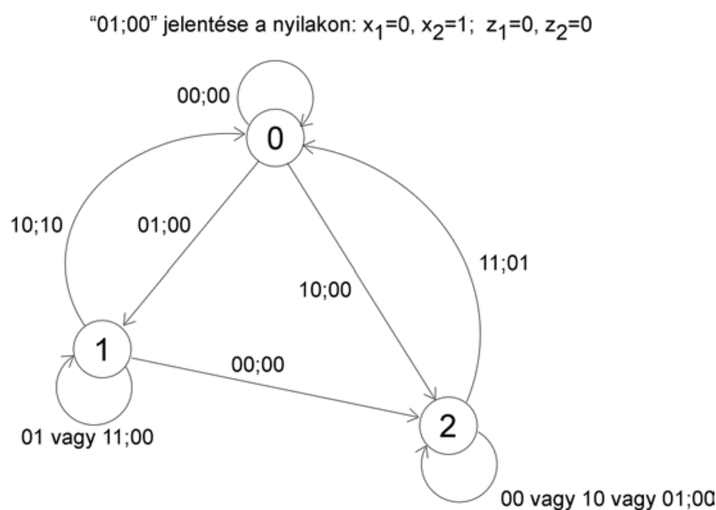


46. ábra

A sorrendi hálózatok állapotterét kétféleképpen definiálhatjuk: állapotgráffal vagy állapottáblával.

4.1 Állapotgráfos leírás

A 47. ábrán egy két bemenetű (x_1, x_2) és két kimenetű (z_1, z_2) szekvenciális hálózat *állapotgráf*ját látjuk. Az állapotokat sorszámozott körök jelölik, köztük pedig nyilak mutatják a lehetséges állapotváltásokat. A nyilakon fel van tűntetve, hogy milyen bemeneti kombináció esetén haladunk rajtuk, és hogy ekkor mi legyen a kimenetek értéke. Természetesen azt is meg lehet adni, hogy a rendszer bizonyos bemeneti kombinációra ugyanabban az állapotban maradjon: ezt önmagukba visszatérő hurokkal jelezzük. (Tulajdonképpen a 45. ábra is egy állapotgráfot mutat, csak a be- és kimeneti változókat nem adtuk meg egzaktul.)



47. ábra

4.2 Állapottáblás felírás

Az *állapottábla* táblázatos formában mutatja meg, hogy a *y* állapotból a különböző bemeneti kombinációk hatására mely *Y* állapotokba ugrunk. A kimenetek alakulását ugyanebbe, de külön táblázatba is írhatjuk. Az előző pontban állapotgráffal fölírt hálózat állapottáblája a következő:

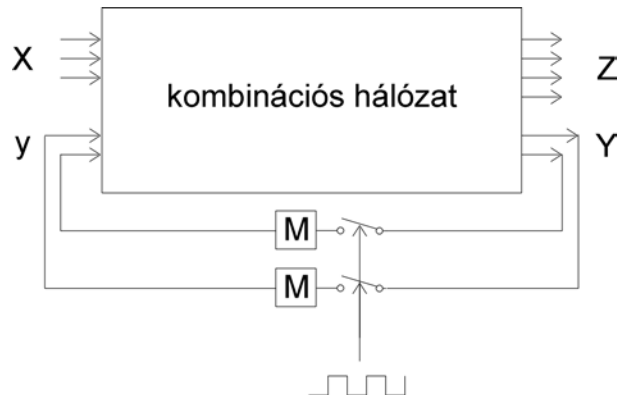
Y	x_1x_2			
y	00	01	10	11
0	0	1	2	-
1	2	1	0	1
2	2	0	2	2

z_1z_2	x_1x_2			
y	00	01	10	11
0	00	00	00	-
1	00	00	10	00
2	00	00	00	01

48. ábra

4.3 Aszinkron és szinkron sorrendi hálózatok

Ha egy kombinációs hálózatot az iméntiek szerint egyszerűen visszacsatolunk, *aszinkron sorrendi hálózathoz* jutunk, mert az aktuális állapot a bemenő jelek hatására bármelyik pillanatban megváltozhat.
Ha viszont a visszacsatolt jeleket csak bizonyos időközönként engedjük vissza a bemenetre – mondjuk egy külső órajel mindegyik periódusában csak egyszer –, akkor az állapotváltozások is csak ebben az ütemben történhetnek. Ilyenkor *szinkron sorrendi hálózat*ról beszélünk. A szinkron hálózatoknak több előnyük is van: egyrészt nem engedik a hazárdokat visszacsatolódni a bemenetekre, hogy hibás állapotváltozásokat idézzenek elő, másrészt nem kell foglalkoznunk az olyan instabil állapotokkal, amelyekből azonnal továbbugrik a rendszer, esetleges oszcillációt okozva. Hátrányuk viszont, hogy az órajel ütemére csökken a sebességük. A 49. ábra egy szinkron sorrendi hálózatot mutat. A kapcsolók mögé egyszerű tároló-(memória-)elemeket kell tennünk, hogy az *y* bemeneteken nyitott kapcsolóállásnál is fennmaradjon az aktuális állapot száma.



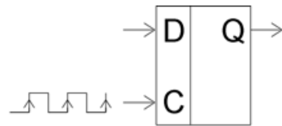
49. ábra

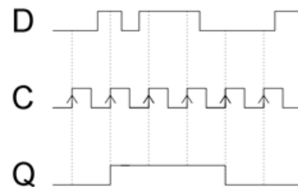
4.4 Elemi sorrendi hálózatok (flip-flopok)

*Flip-flop*oknak, magyarul *tároló*knak vagy *billenőkörök*nek nevezzük azokat a gyakran használt egyszerű sorrendi hálózatokat, amelyeknek mindössze két állapotuk van (0 és 1), egyetlen kimenettel rendelkeznek, és jellemzőjük még, hogy magát a kimenetet csatoljuk vissza a *y* bemenetre, így az aktuális állapot és kimeneti érték mindig megegyezik. Léteznek aszinkron és szinkron flip-flopok is, ez utóbbiakat az órajel-bemenetükről ismerhetjük fel. A „tároló" elnevezés arra utal, hogy többnyire egyetlen bit tárolására használjuk őket. A tokozott, készen vásárolható flip-flop áramköröknél rendszerint a kimenet negáltja is kivezetésre kerül.

4.4.1 D tároló

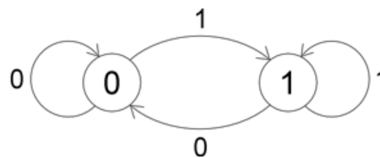
A *D tároló* szinkron elemi sorrendi hálózat, áramköri jele az 50. ábrán látható. Működése a következő: a C bemenetre kapcsolt órajel felfutó élekor (és csak akkor) a kimenet felveszi a D bemenet értékét, és egészen a következő felfutó élig megőrzi azt (függetlenül attól, hogy a D bemeneten történt-e közben változás). Az idődiagramon jól követhető ez a folyamat.





50. ábra

Más megközelítésben úgy is mondhatnánk, hogy a D tároló állapota – a szinkron működésnek megfelelően – az órajel ciklusában követi a bemenet változásait. A flip-flop állapotgráfja is ezt mutatja:



51. ábra

Az ennek megfelelő állapottábla pedig a következő:

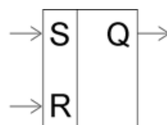
Y

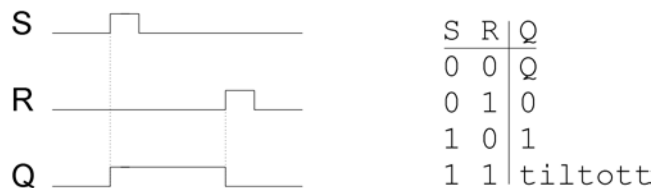
y \ D	0	1
0	0	1
1	0	1

52. ábra

4.4.2 S-R tároló

Az *S-R flip-flop* bemenetei a Set (beírás) és Reset (törlés) szavak első betűit viselik (53. ábra). Funkciójuk a nevüknek megfelelő: ha az S bemenetre 1-est rakunk, a kimenet 1-be íródik, az R bemenet 1-esbe állításakor pedig a kimenet értéke 0 lesz. Ha mindkét bemenet 0, a kimenet értéke változatlan marad. A két bemenetre egyszerre 1-est rakni nem szabad, ez tiltott kombináció. Mindezt az idődiagram melletti táblázatban foglaltuk össze. Az S-R tároló aszinkron működésű, azonnal reagál a változásokra.

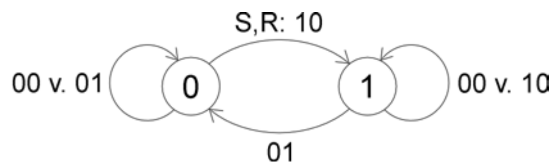




S	R	Q
0	0	Q
0	1	0
1	0	1
1	1	tiltott

53. ábra

A tároló állapotgráfja:



54. ábra

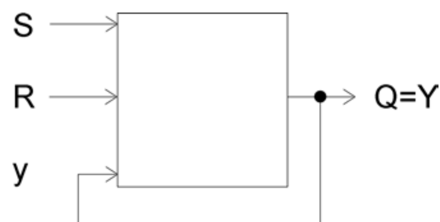
Az 55. ábra az S-R flip-flop állapottábláját mutatja.

Y

y \ SR	00	01	11	10
0	0	0	-	1
1	1	0	-	1

55. ábra

A táblázat olyan formában került felírásra, hogy segítségével bepillantást nyerhessünk a szekvenciális hálózatok tervezésének folyamatába. Ha jobban megnézzük, az S és R változók a Karnaugh-tábláknál megismert sorrendben szerepelnek benne: vagyis ez nem más, mint egy *S, R* és *y* bemenetű, *Y* kimenetű kombinációs hálózat Karnaugh-táblája. A fejezet bevezető részében említettük, hogy a flip-flopoknál magát a kimenetet csatoljuk vissza az *y* bemenetre: az S-R tárolónál ez az 56. ábra szerint alakul. Az ábrán látható négyzet egy egyszerű kombinációs hálózatot rejt, S, R és y bemenetekkel, Q=Y kimenettel: vagyis épp a Karnaugh-táblában szereplő változókkal. Mindezek alapján csak el kell végeznünk a grafikus egyszerűsítést, és abból már fel is rajzolhatjuk az S-R tároló logikai kapcsolási vázlatát.



56. ábra

Az állapottábla Karnaugh-táblaként való egyszerűsítése:

Y: y \ SR	00	01	11	10
0	0	0	-	1
1	1	0	-	1

(S: 11, 10 oszlopok; R: 01, 11 oszlopok; y: 1 sor)

$$Y = Q = S + y\overline{R}, \qquad (111)$$

Ezek szerint a tároló kapcsolási rajza:

57. ábra

4.4.2.1 Az S-R tároló megvalósítása NOR illetve NAND kapukkal

Tudjuk, hogy minden kombinációs hálózat megvalósítható kizárólag NOR vagy NAND kapuk felhasználásával is. Ha az S-R flip-flop áramkörét is e két formára alakítjuk, sokkal tetszetősebb eredményt kapunk. Nézzük először a NOR kapus megoldást! A 111. összefüggésből a De-Morgan azonosságok egyikének (26) segítségével megkaphatjuk Q negáltját:

$$Y = Q = S + y\overline{R} = S + \overline{\overline{y} + R}\text{, így} \tag{112}$$

$$\overline{Q} = \overline{S + \overline{\overline{y} + R}}\,. \tag{113}$$

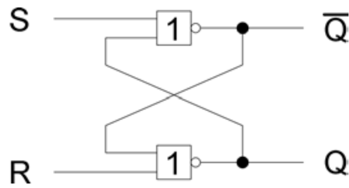
Q átalakított függvényét ugyanilyen módon állíthatjuk elő, csak előzőleg ne Q-t, hanem $\overline{Q}$-ot írjuk fel a Karnaugh-táblából:

Y (y \ SR)	00	01	11	10
0	0	0	-	1
1	1	0	-	1

$$\overline{Y} = \overline{Q} = R + \overline{y}\,\overline{S} = R + \overline{y + S}\,, \tag{114}$$

$$Q = \overline{R + \overline{y + S}}\,. \tag{115}$$

Az így nyert függvények logikai vázlata egy két kapuból álló, szimmetrikus áramkör:



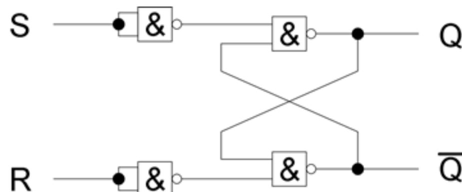
58. ábra

A NAND kapus változathoz is a 112. és 114. egyenletet alakítjuk át, most a 27-sel jelölt De-Morgan szabályt alkalmazva:

$$Y = Q = S + y\overline{R} = \overline{\overline{S + y\overline{R}}} = \overline{\overline{S} \cdot \overline{y\overline{R}}} \text{, illetve} \tag{116}$$

$$\overline{Y} = \overline{Q} = R + \overline{y}\overline{S} = \overline{\overline{R + \overline{y}\overline{S}}} = \overline{\overline{R} \cdot \overline{\overline{y}\overline{S}}} \,. \tag{117}$$

Az eredményül kapott függvényalakok egy-egy negációt is tartalmaznak. E műveleteket, mint tudjuk, összekötött bemenetű NAND kapukkal is elvégeztethetjük (59. ábra).

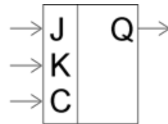


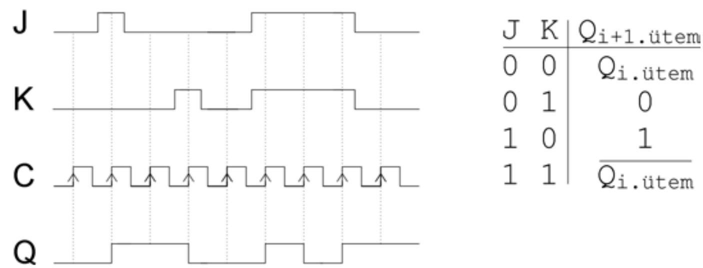
59. ábra

4.4.3 J-K tároló

A *J-K tároló* működése megegyezik az S-R tárolóéval, azzal a különbséggel, hogy itt az 11 bemeneti kombináció is engedélyezett: ekkor a tároló állapota, s így a kimenet is a negáltjára változik. Ez aszinkron hálózatnál oszcillációt eredményezne, hiszen ha mindkét bemenetre 1-est kötnénk – bármilyen rövid időre is –, a hálózat őrült sebességgel ugrálna az egyik állapotból a másikba. A J-K flip-flop ezért szinkron működésű: a C bemenetre érkező órajel felfutó élére kapuzza be az adatokat, csak ekkor változik az állapota

(és a kimenet). Tartós 11 bemeneti kombinációra tehát az órajel ütemében váltakozik a kimenet. Az áramkör J-vel jelzett lába a Set (beíró) láb, a K pedig a Reset (törlés).

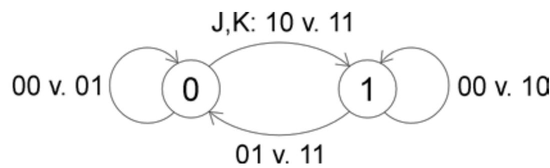




J	K	$Q_{i+1.\text{ütem}}$
0	0	$Q_{i.\text{ütem}}$
0	1	0
1	0	1
1	1	$\overline{Q_{i.\text{ütem}}}$

60. ábra

A flip-flop állapotgráfja:



61. ábra

Állapottáblája:

Y: y \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

62. ábra

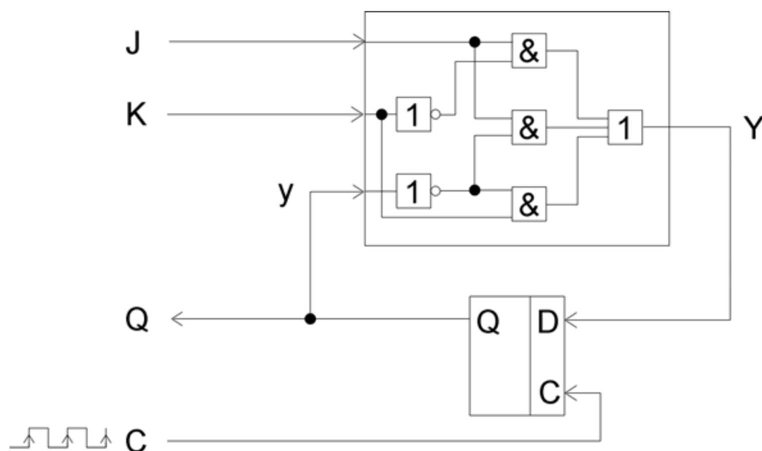
Az állapottábla alapján az S-R tárolóéhoz hasonlóan a J-K flip-flop kapcsolási rajzát is megtervezhetjük. A táblázatot Karnaugh-táblaként

értelmezve az Y kimeneti függvény (ügyelve a statikus hazárd kiküszöbölésére is):

Y: y \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$Y = J\overline{y} + J\overline{K} + K\overline{y}\,. \qquad (117)$$

A tároló szinkron működésű, ezért a visszacsatoló ágba egy D flip-flopot helyezünk, amely csak az órajel ütemében engedi vissza a y bemenetre a következő állapot számát: *Y*-t (ld. 63. ábra). Mivel a J-K tároló *Q* kimenete mindig az aktuális állapottal (*y*) egyenlő, nem a következő állapottal (*Y*), a D tároló „mögül” kell kivezetnünk.

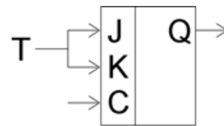


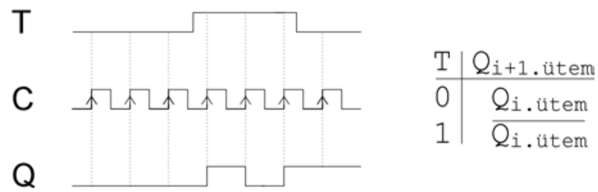
63. ábra

4.4.4 T tároló

T tárolót úgy nyerünk, ha a J-K tároló J és K bemeneteit összekötjük és elnevezzük T-nek. A flip-flop beíró és törlő funkciója így elveszik; ha T=0, a

kimenet megtartja értékét, ha viszont T-t 1-re állítjuk, Q a negáltjára változik az órajel ütemében (64. ábra).

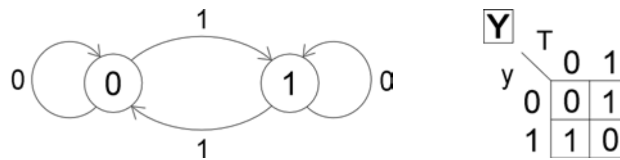




T	$Q_{i+1.\text{ütem}}$
0	$Q_{i.\text{ütem}}$
1	$\overline{Q_{i.\text{ütem}}}$

64. ábra

A T tároló állapotgráfja és állapottáblája:

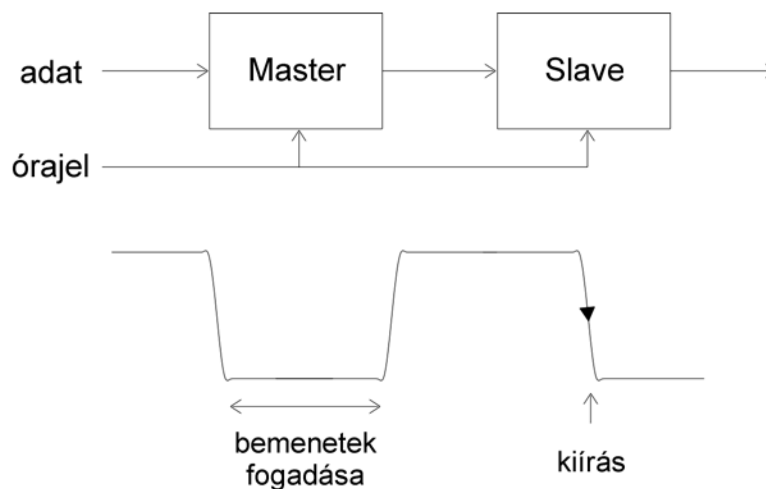


Y (y \ T)	0	1
0	0	1
1	1	0

65. ábra

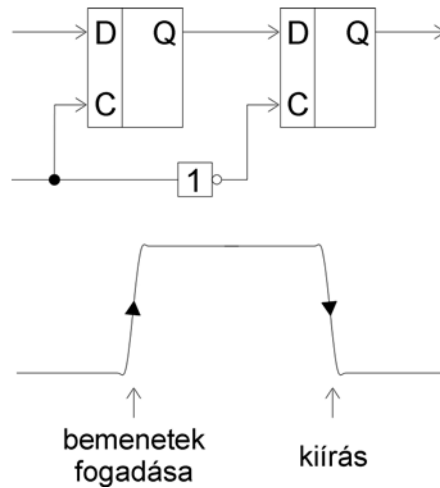
4.4.5 Master-Slave flip-flopok

Az eddigiekben megismert szinkron tárolók *élvezérelt működésű*ek voltak: az órajel felfutó élére „léptek működésbe", ekkor fogadták a bemeneteiken az adatokat, és azonnal meg is jelenítették a változást a kimenetükön. Bizonyos alkalmazások megkövetelik, hogy ezt a két fázist elkülönítsük. A *Master-Slave flip-flopok* két fő egységből állnak: a Master (mester) egység kapuzza be az adatokat a bemenetről, majd továbbküldi azokat a Slave (szolga) egységnek, amely a kimenetre írást intézi. A fázisok elkülönítése érdekében a mester az órajel aljában, 0-s szinten olvassa a bemeneteket, a szolga pedig a lefutó él megjelenésekor helyezi a kimenetekre az új értékeket (66. ábra).



66. ábra

Léteznek még *élvezérelt Master-Slave tárolók* is. Ezeknél a Master egység a felfutó élre, a Slave pedig a lefutó élre lép működésbe. Ilyen típusú tárolót könnyedén készíthetünk, mégpedig oly módon, hogy egy hagyományos élvezérelt flip-flop mögé kötünk egy másik, az órajel negáltjával működő élvezérelt D tárolót (67. ábra). A továbbiakban egyébként, ha flip-flopokról ejtünk szót a jegyzetben, az egyszerű élvezérelt típust értjük alatta – ha mégsem, akkor arra külön felhívjuk a figyelmet.

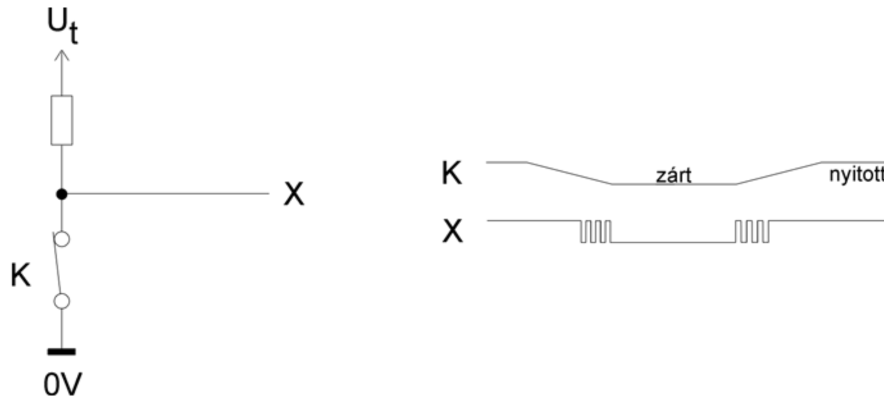


67. ábra

4.4.6 A tárolók jellegzetes alkalmazásai

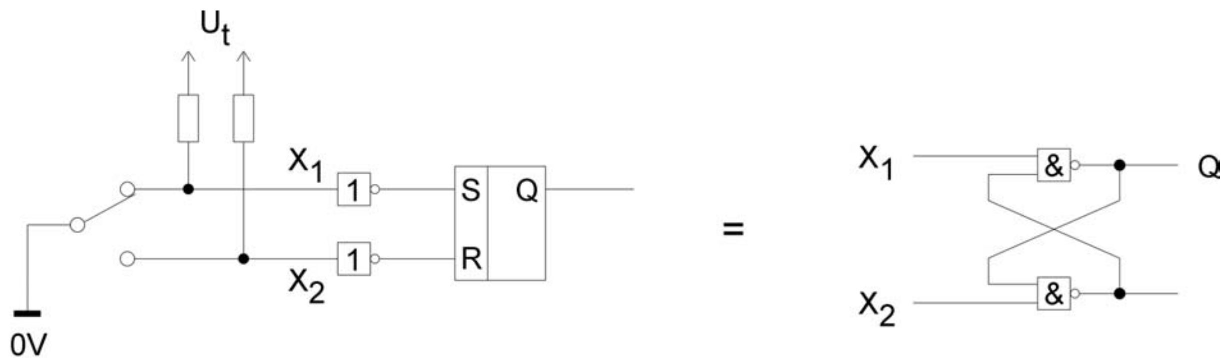
4.4.6.1 Pergésmentesítés

Ha digitális áramkörünkben nyomógombbal vagy kapcsolóval szeretnénk a logikai 0-nak illetve 1-nek megfelelő feszültségszintek között választani, a 68. ábra szerinti áramkört alkalmazhatjuk. A kapcsoló nyitott állásánál az *x* pont a tápfeszültséghez közeli, logikai magas szintre kerül. Ha viszont zárjuk a kapcsolót, 0 potenciálra, logikai alacsony szintre állítjuk (az ellenállás ilyenkor a telep rövidre zárását akadályozza meg). Az áramkör azonban nem tökéletes: a kapcsoló nyitásánál illetve zárásánál ugyanis apró szikrák jelennek meg az elváló vagy egymáshoz közelítő felületek között. Ezek olyan nagy energiájú impulzusokat okoznak, hogy az élvezérelt eszközök tévesen többszöri ki-bekapcsolást érzékelnek. A jelenséget pergésnek vagy prellezésnek nevezzük.



68. ábra

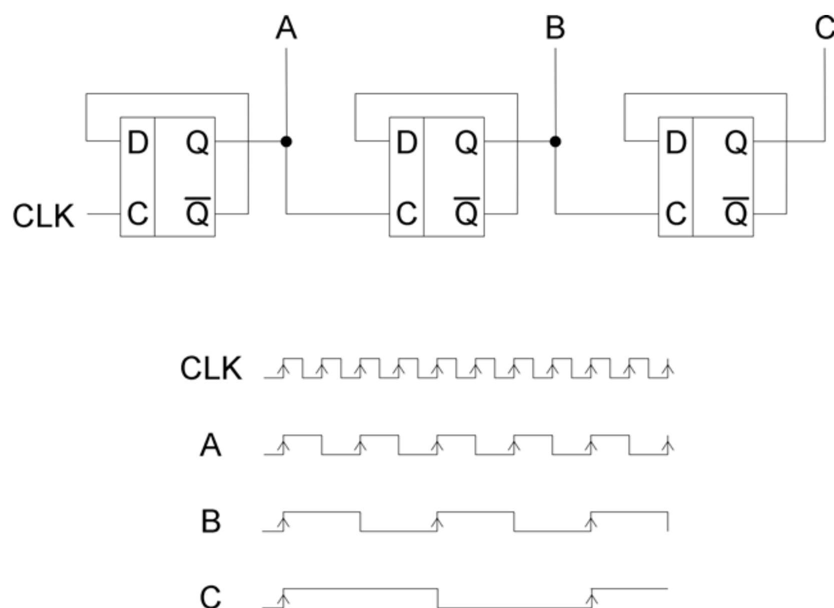
A prellezést a 69. ábrán bemutatott ötletes megoldással küszöbölhetjük ki. Ugyan a kétállású kapcsolóban is keletkeznek szikrák, ám ezeket egy S-R tárolóval „megszelídítjük”: már az első impulzus átbillenti a flip-flopot, így a többinek már nincs hatása a tároló működésére, ezért nem juthatnak el a kimenetig.



69. ábra

4.4.6.2 Frekvenciaosztás

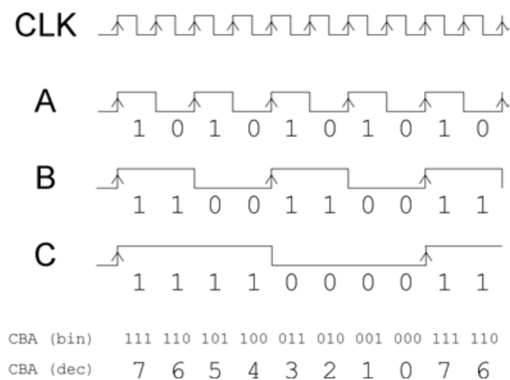
Ha megépítjük a 70. ábrán látható elrendezésű kapcsolást, az *A* ponton olyan váltakozó jelet mérhetünk, amelynek frekvenciája éppen a fele a CLK bemenetre kötött órajel frekvenciájának. A D tároló $\overline{Q}$ kimenetét visszakötöttük az adatbemenetre, így a beérkező órajel minden felfutó élekor a negáltjára vált a flip-flop kimenete. Mindez azt eredményezi, hogy amíg a CLK jel fel, majd lefut, addig az *A* jel csak egyszer változik – vagyis fele akkora lesz a frekvenciája. A második illetve harmadik tároló is kettes osztást végez, a *B* jel ezért a CLK órajelhez képest már csak negyedakkora, a *C* jel pedig 8-ad akkora frekvenciájú lesz, és így tovább. *n* darab tároló frekvenciaosztása: 2^n .



70. ábra

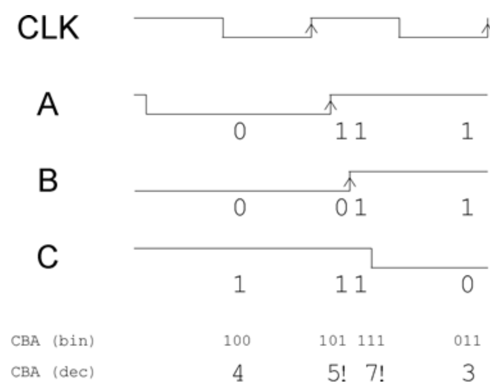
4.4.6.3 Aszinkron számlálók

Írjuk fel a frekvenciaosztó áramkör egymást követő kimenetei kombinációit az időben előre haladva! A változók sorrendje legyen CBA. A 71. ábráról (amely nem más, mint a 70. ábrából kiragadott idődiagram) olvashatjuk le az eredményt:



71. ábra

Láthatjuk, hogy a frekvenciaosztó egyben számlál is, méghozzá kettes számrendszerben 7-től 0-ig lefelé, s azután újrakezdi. Az áramkör teljes neve: *modulo 8-as aszinkron lefelé számláló*. Modulo 8-as azért, mert 8 értéket tud megkülönböztetni, aszinkron, mert a tárolók nem szinkronban „billennek”: az órajel végigfut az elsőtől kezdve az utolsóig, egymás után hozva működésbe őket. Ennek sajnos az az eredménye, hogy minden egyes számláláskor rövid ideig hibás kombináció jelenik meg a kimeneteken. A 72. ábrán a 4 és 3 közötti hibás átmenetet láthatjuk.



72. ábra

Kiküszöbölhetjük a hibát, ha a kimeneteket egyszerre működésbe lépő tárolókkal szinkronizáljuk (73. ábra). A kimeneti flip-flopok csak akkor válthatnak, ha a számlálón már végighaladt az órajel – ezért iktattuk be a *dt* késleltetést.

73. ábra

Amennyiben nem a Q kimenetekről, hanem $\overline{Q}$-ról visszük tovább az órajelet, modulo 8-as aszinkron *felfelé számlálót* kapunk (74. ábra). A változás abban áll, hogy a negációk miatt az előző szint kimenetének a *lefutó* élénél lép működésbe a második, majd ugyanígy a harmadik tároló. Természetesen itt is fennáll az előbbi probléma: kijavítása is a megismert módon történik.

74. ábra

4.4.6.4 Szinkron számlálók

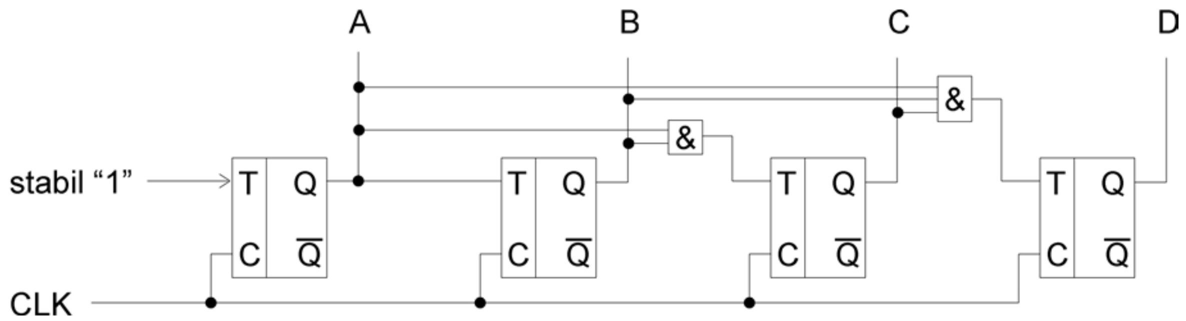
Az aszinkronitásból eredő hibát *szinkron számálók* alkalmazásával is kiküszöbölhetjük. Ezeknél az eszközöknél a külső órajel az összes flip-flopot egyszerre vezérli. Vegyük példaként a *szinkron felfelé számlálót*! Ennek tervezésénél a következő felismerésből indulhatunk ki: egy tetszőleges bináris számot úgy tudunk növelni egyel, hogy megváltoztatjuk (0-ról 1-re vagy fordítva)

- a legkisebb számjegyét,
- továbbá azokat a számjegyeit, amelyekre igaz, hogy a náluk kisebb helyiértékeken mindenütt 1-es áll (ld. a 75. ábrát).

0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

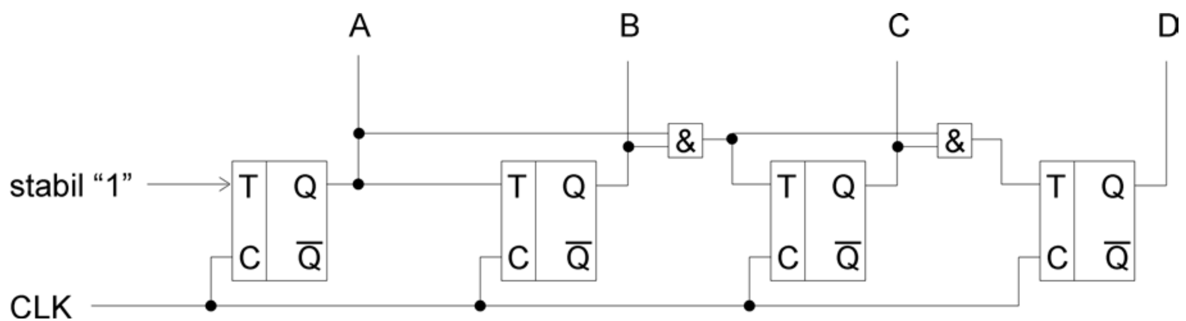
75. ábra

Tudjuk, hogy a T tároló akkor változtatja meg az állapotát, ha az órajel felfutó élekor 1-es van az adatbemenetén – márpedig a szinkron felfelé számláló megépítésénél nekünk pontosan erre van szükségünk. Az első T flip-flop adatbemenetére stabil 1-est kötünk, így minden órajel ciklusban váltani fog (legkisebb helyiérték; ld. 76. ábra). A többi T tároló adatbemenetére pedig a náluk előrébb álló egységek kimenetének ÉS kapcsolatát vezetjük – így ha mindegyik egyes, a flip-flop váltani fog.



76. ábra

A kapcsolás hátránya, hogy az *n-ik* szinten *n-1* bemenetű ÉS kapu kell, mert mindegyikükhöz külön-külön vezettük el a kimeneteket (ezt *párhuzamos átvitel*nek nevezzük). A problémát *soros átvitel* alkalmazásával küszöbölhetjük ki (77. ábra): itt felhasználjuk az alacsonyabb szinteken álló ÉS kapuk eredményét is. A soros átvitel hátránya, hogy az ÉS kapukon keresztüli jelterjedést végig kell várnunk, és csak utána folytathatjuk a számolást. (A jelterjedés viszont nem okoz az aszinkron számlálóknál tapasztalt hibás kimeneti kombinációt, mert a tárolók egyszerre lépnek működésbe.)



77. ábra

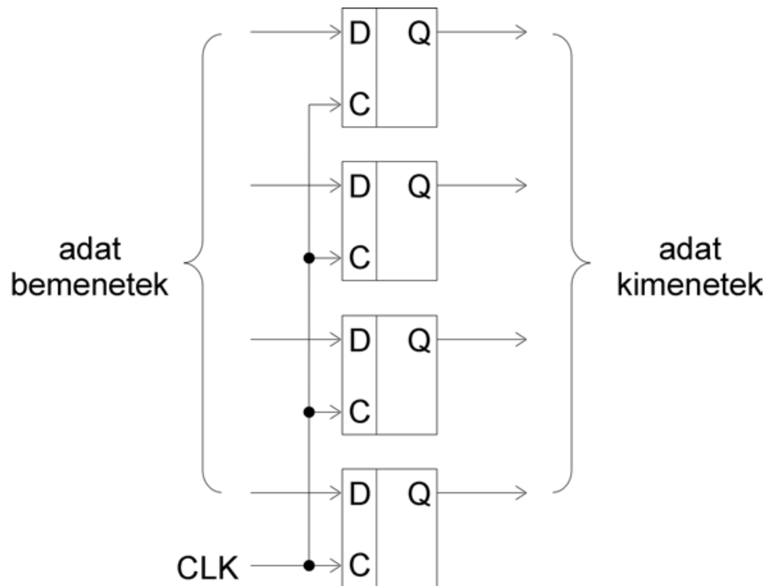
4.4.6.5 Még néhány szó a számlálókról

Az előző pontokban megismert eszközökön kívül még nagyon sokféle számlálót lehet akár tokozott formában beszerezni, vagy megépíteni. A bináris mellett még decimális vagy BCD kódúak közül is válogathatunk. Léteznek 8, 16 bitesek és kaszkádolhatók is, amelyekből többet láncra fűzve szinte bármekkorára növelhetjük a kapacitást. Némelyiket utasíthatjuk, hogy lefelé vagy fölfelé számoljon. A *beírható, vagy tölthető (loadable) számlálók*nál külön erre a célra fenntartott bemeneteken beírhatunk egy számot, ahonnan a számlálás kezdődjön. A beírást egy vezérlőbemenettel engedélyezzük. Ezt kétféleképpen tehetjük meg:

- Szinkron beírás esetén a szám betöltése – ugyanúgy, mint a számlálás – csak az órajellel szinkronban történhet. Egy LOAD/COUNT bemenettel szabályozzuk, hogy a számláló az adott ütemben betöltődjön, vagy számoljon.
- Aszinkron beírású számlálóknál a kívánt számot az órajeltől függetlenül, bármikor betölthetjük. Ha a LOAD bemenet felfutó jelet érzékel, a szám azonnal beíródik.

4.4.6.6 Regiszterek

A *tároló regiszterek* több bites számok tárolására alkalmasak. A 78. ábrán egy 4 bites regisztert látunk, négy darab közös órajelű, párhuzamosan kötött D tárolóból kialakítva. Működése a D flip-flopot ismerve nem szorul különösebb magyarázatra: az órajel felfutó élénél elraktározza az adatbemenetekre adott számot, amelyet a kimeneteken bármikor leolvashatunk. A számot a következő órajel-felfutásig tárolja, amikor is újat olvas be.

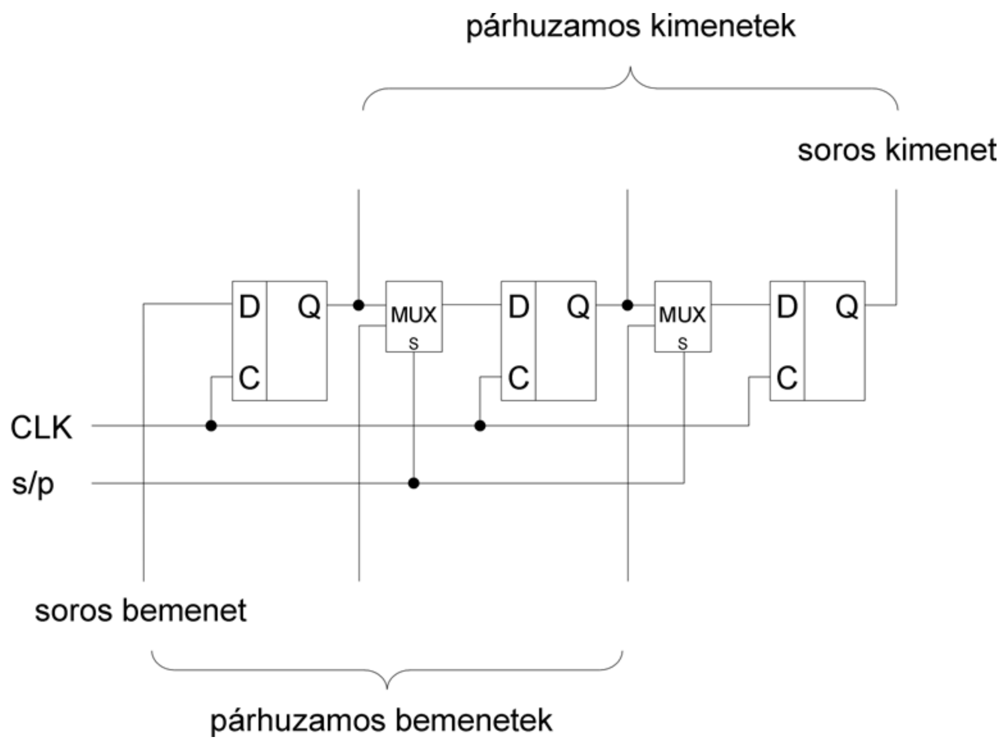


78. ábra

Vannak regiszterek, amelyek a tárolt szám jegyeit egy vagy több helyiértékkel jobbra vagy balra el tudják léptetni: ezeket *léptető vagy shift regiszterek*nek hívjuk. A léptető regiszterekkel megtehetjük, hogy a számjegyeket egyetlen kimeneten, sorra egymás után olvassuk ki (az órajel ütemében). Ezt *soros kiolvasás*nak nevezzük. Léteznek *soros beírás*ú léptető regiszterek is, ezeket egyetlen bemeneten tölthetjük fel a számjegyekkel. A 79. ábra egy univerzális léptető regisztert mutat: a számot tetszés szerint sorosan és párhuzamosan is be tudjuk írni, illetve ki tudjuk olvasni. Működése a következő:

- Ha az *s/p (soros/párhuzamos)* bemeneten 1-et állítunk be, akkor az órajel felfutó élére a párhuzamos bemenetekről töltődnek be a flip-flopokba az adatok (mert a multiplexerek az alsó adatbemenetüket kötik össze a kimenetükkel). Ilyenkor a párhuzamos kimeneteken azonnal ki is olvashatjuk a számot.
- Az *s/p* 0-ra állításával a multiplexerek az egymást követő tárolók ki- és bemeneteit kötik össze, így az órajel felfutó élekor az első flip-flopban tárolt számjegy átkerül a másodikba, a másodikban levő a harmadikba, vagyis jobbra léptetjük a számot. Többszöri órajel-ütemre a számjegyek sorra kipotyognak a „soros kimenet”-en, de közben ugyanígy újabb számjegyeket olvasunk be a „soros bemenet”-en. A

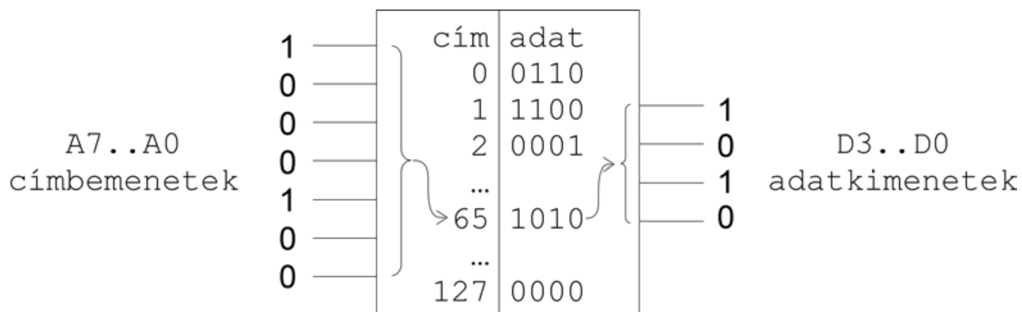
sorosan beolvasott számot természetesen bármikor párhuzamosan is kiolvashatjuk.



79. ábra

5. Memóriák

Míg a D flip-flop egyetlen bit, egy regiszter pedig egyetlen bináris szám tárolására alkalmas, addig a *memóriák*ban már több ilyen számot is megőrizhetünk. Az adatokat számozott rekeszekben tároljuk (80. ábra). Egy adott rekesz tartalmát úgy tudjuk kiolvasni, hogy a *címbemenetek*re (*Address pins*) adjuk annak sorszámát (természetesen bináris formában), mire az *adatkimenetek*en (*DATA pins*) az ún. *hozzáférési idő* elteltével megjelenik a rekeszben tárolt szám.



80. ábra

Az egy rekeszben tárolható bitek száma adja a memória *szóhosszúságá*t. A memória *kapacitás*a a tárolható szavak számát jelöli. Ha n darab címvezetékünk van, a memória 2^n rekeszt tartalmaz. A tárolás jellege alapján kétféle típust különböztetünk meg:

- *ROM (Read Only Memory), csak olvasható memória*: ezeket az eszközöket a memóriát tartalmazó rendszer fejlesztése során töltjük fel adatokkal, a rendszer működtetésénél már csak olvasunk belőlük. A beírt adatok az áramellátás megszűnése után is megmaradnak (*nem-illanó memóriák*), akár évekig is. A beíráshoz sokszor külön erre a célra használatos (ún. égetőáramköröket) használunk. Vannak ROM-ok, amelyek csak egyszer írhatók, újraírásukra nincs lehetőség (*OTP: One Time Programmable devices*). Az OTP áramköröket nagy sorozatban gyártott eszközökben alkalmazzák, miután teljesen lezárult a

rendszerfejlesztés folyamata. Az *EPROM*-ok a beírás után a chip tetejére irányuló UV fénnyel kitörölhetők, és újra felhasználhatók. Az *EEPROM*-ok elektromosan írhatók és törölhetők is. Az újraírhatóság a prototípusok fejlesztésénél elengedhetetlen. Az EEPROM-ok egyik fajtája a manapság divatos *FLASH memória*, amely technológiai jellemzői szerint ROM, ám alkalmazását tekintve inkább már a következő kategóriába sorolható:

- *RWM (Read-Write Memory), írható-olvasható memóriák*: ezeknek a típusoknak tetszés szerint írhatjuk és olvashatjuk bármelyik rekeszét. Kikapcsolás után az adatok elvesznek, ezért *illanó memóriák*nak is hívjuk őket. Az adatvezetékeik kétirányúak: ki- és bemenetek is egyben. A beírás folyamatát általában egy $\overline{WE}$ (Write Enable) bemenettel vezéreljük: gyakorlatilag evvel döntjük el, hogy írni vagy olvasni akarjuk a címbemeneteken kiválasztott rekesz tartalmát. Kétféle technológiával készítenek ilyen eszközöket: a *statikus memóriák*ban sok-sok flip-flop tárolja az adatokat. A *dinamikus memóriák* apró, mátrix-alakban elhelyezett kondenzátorok segítségével raktározzák el a biteket. Ha egy kondenzátor fel van töltve, az adott bit 1-es, egyébként 0. Mivel ezek a kondenzátorok nagyon kicsik, gyakran ki kell olvasni őket, majd ugyanazt az adatot visszaírni (frissíteni), különben elszivárognak a töltések, és elveszik a tárolt információ. A korszerű dinamikus memóriák egy speciális áramkört is tartalmaznak, amely időről időre automatikusan elvégzi a frissítést.

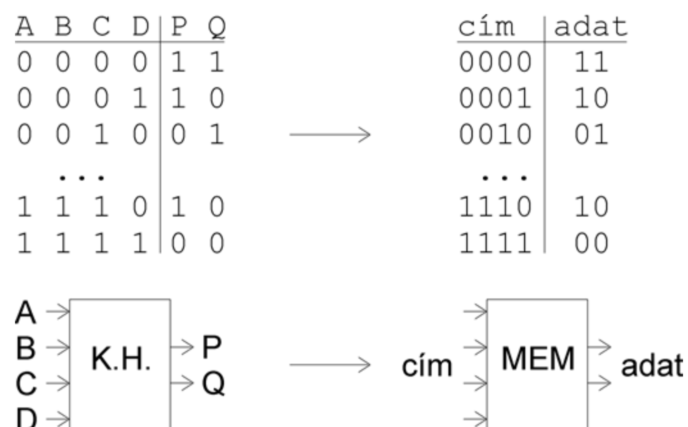
A ROM-ok „ellenpárjaként” a köztudatban nem a RWM, hanem a *RAM (Random Access Memory), véletlen hozzáférésű memória* szerepel. A valóságban azonban egy RAM is lehet ROM: a rövidítés ugyanis azt takarja, hogy a memória hozzáférési ideje állandó, bármelyik, véletlenszerűen kiválasztott rekeszt is olvassuk. Ezzel szemben a *SAM (Serial Access Memory), soros hozzáférésű memória* legutolsó rekeszéhez csak az összes előtte álló adat kiolvasása után férhetünk hozzá.

Az eddigiekben csak a *hely szerint címzett memóriák*ról ejtettünk szót, ahol a kívánt adatot a címe alapján választottuk ki. Ritkán, de találkozhatunk még *tartalom szerint címzett (asszociatív) memóriák*kal is, amelyek a tárolt

információ egy része alapján választják ki a megfelelő adatot (az információ egy részéről asszociálnak az egészre).

5.1 Kombinációs hálózatok megvalósítása programozható logikai elemek felhasználásával

Vessünk újra egy pillantást a kombinációs hálózatokra! Egy adott bemeneti kombinációra az igazságtábla ugyanazon sorában feltüntetett kimeneti kombináció a válasz. Beadunk egy számot, mire egy másik számot várunk a kimeneteken. Ha jól belegondolunk, a memóriák feladata is teljesen ugyanez: minden egyes cím bevitelekor egy előzőleg betöltött adat jelenik meg. Vagyis ha egy memóriát egy vele azonos számú be- és kimenettel rendelkező kombinációs hálózat igazságtáblája szerint töltünk fel, akkor ez a memória helyettesítheti a kombinációs hálózatot (81. ábra).



81. ábra

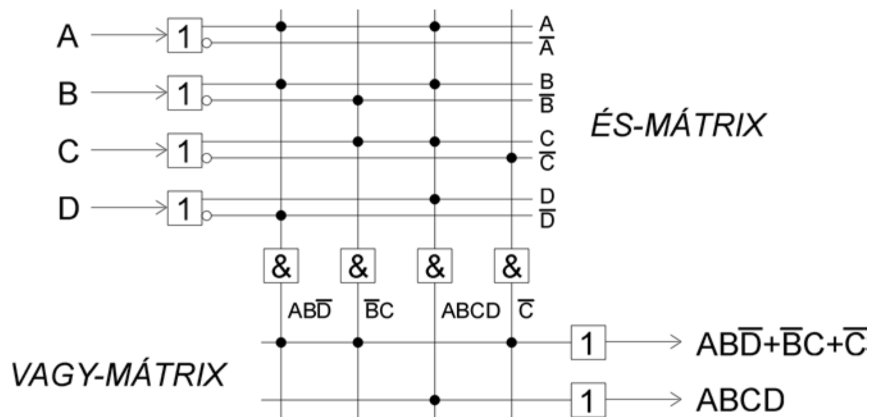
A memóriaelemmel történő megvalósítás előnyei:

- könnyen átprogramozható, így a fejlesztési szakaszban nem kell újraépítenünk egy apró változtatásnál az egész áramkört,
- nem igényel függvény-egyszerűsítést,
- nem fordulhat elő benne statikus és dinamikus hazárd (bár itt is van funkcionális hazárd, amit szinkronizációval szüntethetünk meg).

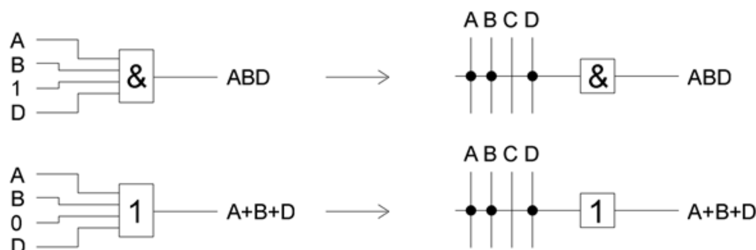
A megoldásnak hátrányai is vannak:

- Egy memória-áramkör sokkal lassabb, mint a logikai kapukból összeállított kombinációs hálózat,
- speciális időzítési feltételekkel fogadhat csak jeleket (pl. a címnek bizonyos ideig stabilnak kell lennie, hogy előálljon a kimenet),
- míg a függvény-egyszerűsítéssel kapott megoldás esetleg csak néhány kapuból állna, a memóriába a teljes igazságtáblázatot be kell programozni: n darab bemenethez mindenképp be kell szereznünk egy 2^n kapacitású memóriát,
- ezen kívül a memóriaelem a legtöbb esetben drágább is.

A két változat előnyeit egyesíti a *programozható logikai eszközök*kel történő megvalósítás. Ezek tulajdonképpen olyan előre kialakított kombinációs hálózatok, amelyeknek az összeköttetéseit programozhatjuk. Angol elnevezésük az *FPLA (Field Programmable Logic Array)*. *PLA*-nak hívják a gyárban előre beprogramozott (a felhasználó által nem változtatható) eszközöket. A memóriákhoz hasonlóan EPROM, EEPROM típusú FPLA-k is kaphatók.
A 82. ábra egy FPLA áramkör belső elrendezését mutatja. A rajzban a sok párhuzamos vezeték eltűntetése végett egyszerűsített jelöléseket alkalmaztunk: a kapuk bemeneteit egyetlen szimbolikus vezetékkel jelöltük, amelyre több változót is kötöttünk. Ez a valóságban több bemenetet jelöl, mindegyiken egy-egy változóval (ld. az ábra magyarázó részét). A kis körök mutatják az összeköttetések helyét: ezeket mi határozhatjuk meg, tetszőleges diszjunktív alakú függvényeket létrehozva.



Az ábrán az alábbi egyszerűsített jelöléseket használtuk:



82. ábra

Az FPLA-knál mind az ÉS-mátrix, mind a VAGY-mátrix a felhasználó által programozható. Léteznek olyan egyszerűbb áramkörök is, amelyeknél csak az ÉS-mátrix változtatható, a VAGY-mátrixot a gyártás során rögzítették. Az ilyen eszközöket *PAL*-nak (*Programmable Array Logic*) nevezzük. Elterjedésük oka, hogy olcsóbb berendezésekkel programozhatók, mint az FPLA-k. Felhasználásuk során katalógusból kell kiválasztanunk a nekünk megfelelő VAGY-mátrixú áramkört.

Az FPLA-k előnyei:

- A memória-áramkörökhöz hasonlóan könnyen átprogramozhatók,
- annyi programozott összeköttetés is elegendő az esetükben, amennyivel az egyszerűsített függvényt elő tudjuk állítani (míg a memóriáknál az összes variációt be kellett programozni),
- gyorsabbak, mint a memória-áramkörök (hiszen egyszerű kombinációs hálózatok).

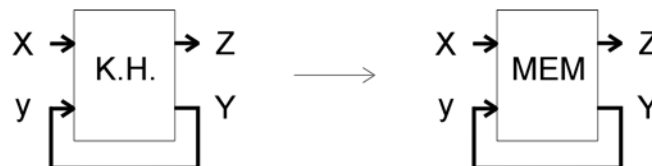
Hátrányaik:

- A kombinációs hálózathoz hasonlóan egyszerűsítést igényelnek,
- nem mentesek a hazárdoktól.

6. A mikroprocesszoros rendszerek alapjai

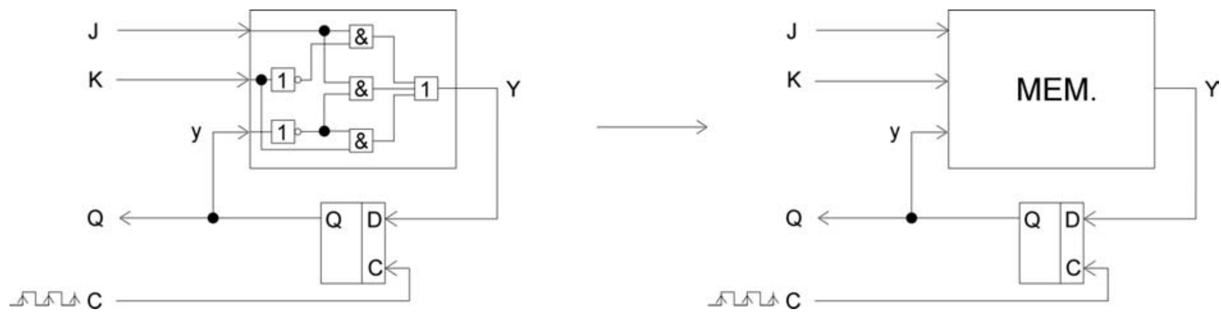
Tekintsük át az eddig tanultakat! Először is definiáltuk a digitális jel illetve rendszer fogalmát. Megismertük az ilyen elven működő hálózatok matematikai alapjait, a logikai kapukat, majd ezekből kombinációs hálózatokat terveztünk. Visszacsatoltunk néhány kimenetet, s ezzel máris emlékező áramkörökhöz jutottunk: a sorrendi hálózatokhoz. Röviden összefoglaltuk a memóriák működését, végül megnéztük azt, hogyan lehet programozható eszközöket felhasználni a kombinációs hálózatok tervezésénél. A következőkben a megkezdett úton haladunk tovább: a már megismert elemekből egy olyan egyszerű automatát építünk, amely némi jóindulattal már számítógépnek nevezhető, s megtudhatjuk, hogy milyen fejlesztési lépések megtétele után lesz belőle valódi mikroprocesszor.

Az előző fejezetben láthattuk, hogy a kombinációs hálózatok megvalósíthatók memória-áramkörökkel is. Folytassuk a gondolatmenetet: mivel a sorrendi hálózatok is a kombinációs hálózatok alapjaira építkeznek, bennük is használhatunk memóriákat (83. ábra). Ezzel programozható szekvenciális hálózatokhoz jutunk.



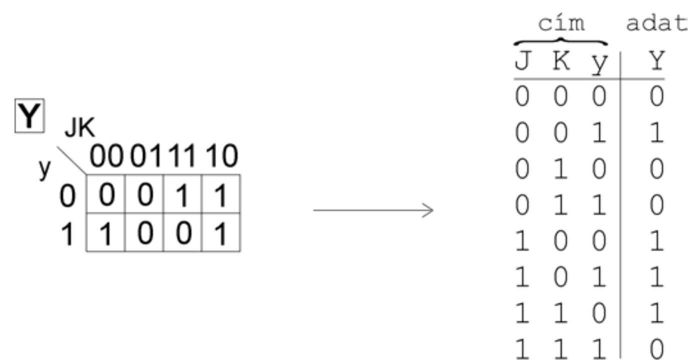
83. ábra

Vegyük például a J-K tárolót! A logikai kapcsolási vázlatát már ismerjük: a benne szereplő kombinációs hálózatot kell memóriára cserélnünk (84. ábra). Bár a gigabyte-ok korszakában furcsának tűnhet, de léteznek 8 szavas 1 bit szóhosszúságú memóriák is.



84. ábra

Tudjuk, hogy a tároló állapottáblája és a kombinációs hálózatának Karnaugh-táblája megegyezik (ld. a 4.4.3. fejezetet). Ha ezt visszaalakítjuk igazságtáblává, azokat az adatokat kapjuk, amelyeket a memóriába kell tölteni:



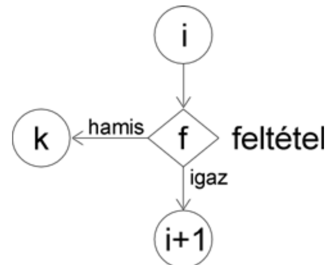
Y: y \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

cím			adat
J	K	y	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

85. ábra

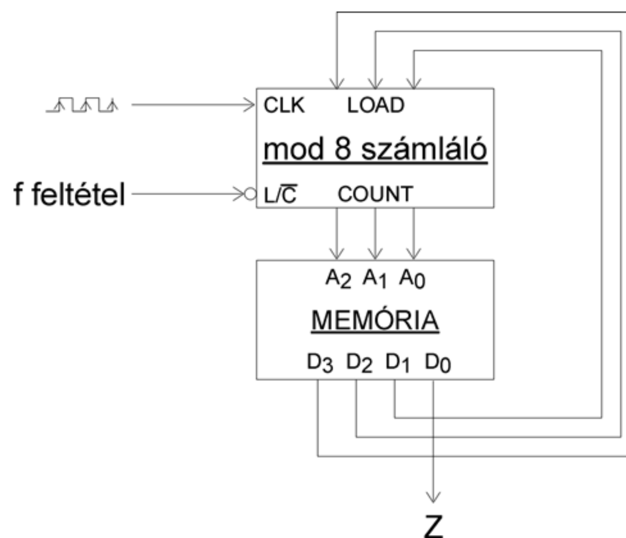
A tábla egyetlen sora azt adja meg, hogy a flip-flop egy bizonyos állapotból (*y*) adott bemeneti kombináció (*J, K*) hatására mely állapotba ugorjon (*Y*).
Bár a táblázat csak nyolc soros, az utólagos ellenőrzés vagy hibakeresés nagy koncentrációt igényel, annak ellenére, hogy a J-K tároló a legegyszerűbb, elemi sorrendi hálózatok egyike. A probléma az, hogy a szekvenciális hálózatok állapottere (-gráfja) túl bonyolult. Ahhoz, hogy hatékonyan programozható automatákat tervezhessünk, a következő szigorítást kell tennünk: bármely *i*-ik állapotból csak 2 irányba ugorhassunk tovább: ha egy bizonyos feltétel teljesül, akkor az *i*+1-ik állapotba, ha nem teljesül, akkor egy általunk választott *k* állapotba (86. ábra). A feltétel a

bemenetek valamely kombinációját vagy kombinációit jelentheti (pl. ha x_1=0 és x_2=1, akkor a feltétel igaz: $f = \overline{x_1} x_2$).



86. ábra

Készítsünk egy olyan egyszerű rendszert, amelyik így működik! A 87. ábrán egy számlálót és egy memóriát kötöttünk sorba. A számláló modulo 8-as, tehát 0-tól 7-ig számlál. *Szinkron* LOAD bemenetekkel is rendelkezik: ha az órajel felfutó élénél az $L/\overline{C}$ vezérlőbemeneten 1-est talál, akkor a LOAD bemenetekről betölti az ott lévő számot. Ha 0-t, akkor eggyel továbbszámol. A memória 8 szavas, 4 bit szóhosszúsággal. A D_0 adatkimenet legyen az automatánk *Z* kimenete. A számláló kimenetei a memória címbemeneteire vannak kötve: így mindig az a sorszámú rekesz van kiválasztva, amelyik számon a számláló éppen áll.



87. ábra

Definiáljuk úgy a rendszert, hogy az automata mindig abban az állapotban van, amelyik számnál a számláló tart. Az i-ik állapotban a számláló a memória i-ik rekeszét jelöli ki, ennek a rekesznek a tartalma jelenik meg a memória kimenetein.

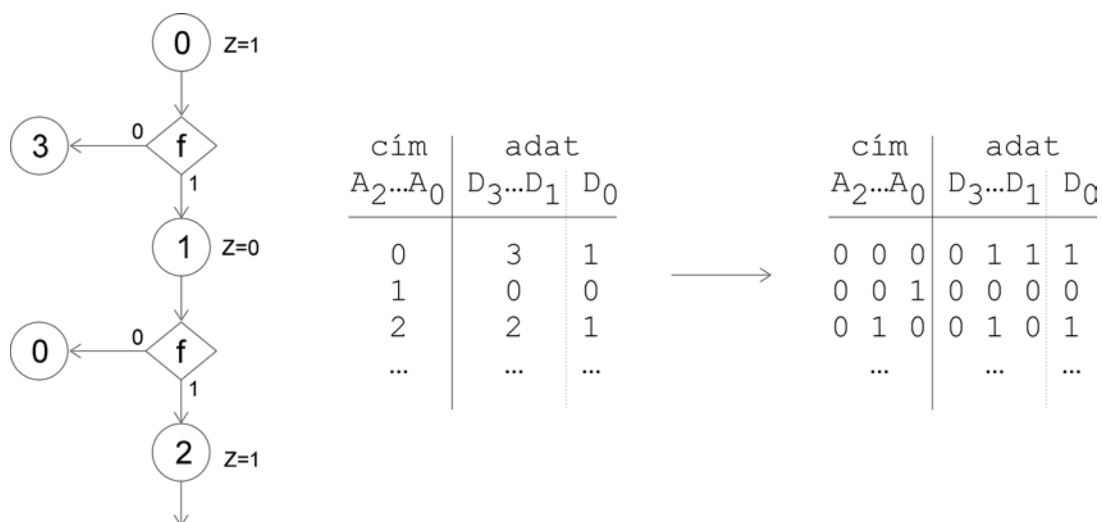
Ha a feltétel igaz, vagyis 1-es (ekkor az $L/\overline{C}$ bemenetre a negáció miatt 0 kerül), a számláló az órajel ütemében számol, az automata rendre az eggyel nagyobb sorszámú állapotokba ugrik.

Amennyiben az i-ik állapotban vagyunk, és az órajel felfutó élekor hamis a feltétel (f=0), a LOAD bemenetek aktiválódnak: a memória i-ik rekeszének felső 3 bitje ezeken keresztül visszatöltődik a számlálóba. Az automata tehát az i-ikből olyan állapotba ugrik, amilyen számot az i-ik rekesz $D_3 \ldots D_1$ bitjeibe programoztunk (egy bináris szám formájában).

Ezek alapján az automata programozása a következő módon történik:

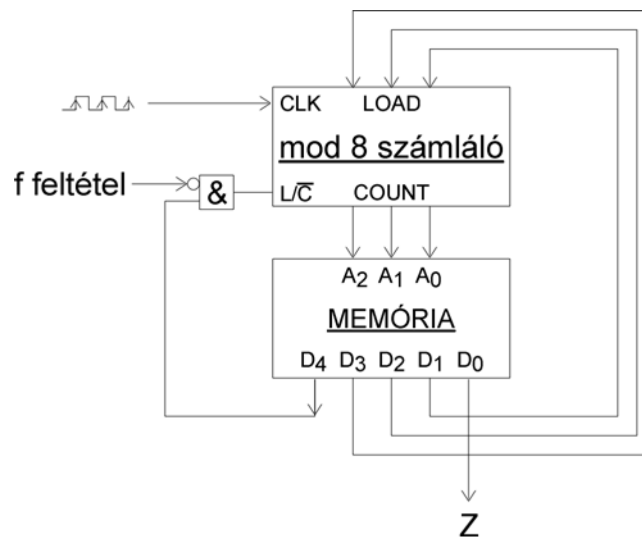
- Az i-ik rekesz legalsó (0-ik) bitjébe beírjuk azt, hogy mi jelenjen meg a kimeneten az i-ik állapotban,
- az i-ik rekesz felső három bitjébe ($D_3 \ldots D_1$) annak az állapotnak a sorszámát írjuk, amelybe ugrani akarunk, hogyha nem teljesül az f feltétel.

Erre a 88. ábrán láthatunk egy példarészletet:



88. ábra

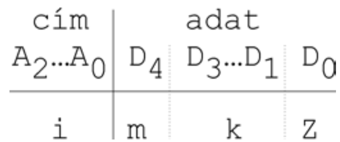
Most fejlesszük tovább a rendszerünket úgy, hogy eldönthessük: egy adott állapotban meg akarjuk vizsgálni a feltételt vagy sem. Az eredmény a 89. ábrán látható. A memória szóhossza immár 5 bites. A legfelső adatbitnek és az f feltétel negáltjának ÉS kapcsolatát vezetjük az $L/\overline{C}$ bemenetre. Ez arra való, hogyha egy adott állapotban D_4=0, akkor az f feltétel semmiképpen sem juthat át az ÉS kapun, nem állíthatja le a számlálást, vagyis mindenképp a következő állapotba ugrunk.



89. ábra

Az automata programozása tehát a következőképp zajlik:

- Az i-ik rekesz legalsó (0-ik) bitjébe beírjuk azt, hogy mi jelenjen meg a kimeneten az i-ik állapotban (*Z*-vel jelölve a 90. ábrán),
- az i-ik rekesz legfelső (4-ik) bitjébe 0-t írunk, ha az i-ik állapotban nem akarjuk megvizsgálni a feltételt, 1-et pedig akkor, ha meg akarjuk vizsgálni (*m* az ábrán),
- az i-ik rekesz $D_3 \ldots D_1$ bitjeibe annak az állapotnak a sorszámát írjuk, amelybe ugrani akarunk, ha a vizsgálat során nem teljesül az f feltétel (*k* az ábrán).



90. ábra

Rendszerünk működését egy kicsit más szemszögből is nézhetjük. Azt is mondhatjuk, hogy a memóriába egy programot töltöttünk: minden egyes rekeszben egy-egy utasítás van. A berendezés elindul a 0-ik programsorról, és egymás után végrehajtja az utasításokat (lefuttatja a programot). A gépünk kétféle utasítást ismer:

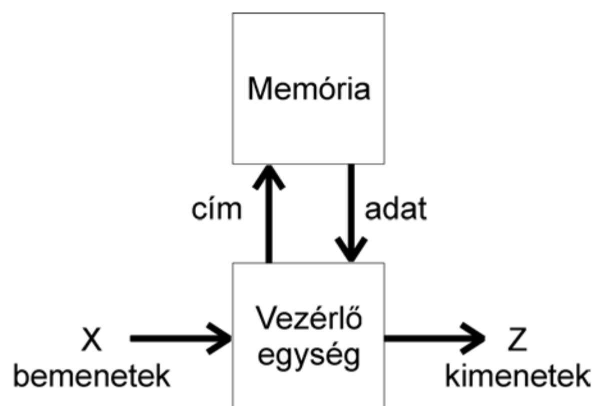
- ha m=0, akkor arra utasítjuk, hogy egyszerűen írjon ki egy számot a Z kimenetre (kiíró utasítás). A program végrehajtása ilyenkor a következő programsorról folytatódik.
- Ha m=1, vizsgálja meg az f feltételt, és az eredménytől függően a következő, vagy a k-ik sorszámú programsorra ugorjon. Z kiírása ekkor is megtörténik (feltételes ugró utasítás kiírással).

Gyakorlatilag egy primitív számítógépet alkottunk, amely még a programciklusok kezelésére is képes! Nem kell lebecsülnünk a művünket, a modern számítógépek programjai is többé-kevésbé a most megismert formájúak:

- A programok utasításokból állnak, amelyeket szekvenciálisan hajt végre a számítógép.
- Az utasítások két fő részből tevődnek össze: a *műveleti kód*ból és az *operandusok*ból. A műveleti kód mondja meg konkrétan, hogy mit végezzen a gép. Ez esetünkben az m-el jelölt bit. A komolyabb rendszerek persze többféle utasítást ismernek: n bites műveleti kód esetén az utasítások lehetséges száma 2^n. Minden műveleti kódhoz tartozhat egy vagy több operandus is: ezek az utasítás elvégzéséhez szükséges adatok. A mi számítógépünk „feltételes ugró és kiíró” utasításának két operandusa van: k, amely ugrás esetén megadja a következő utasítás címét, és Z, amely a kimenetre írt adatot tartalmazza. A sima kiíró utasítás egyetlen operandussal rendelkezik:

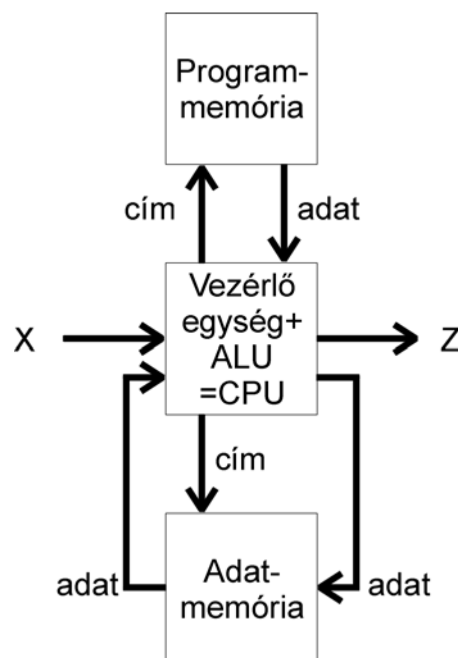
Z-vel. Ilyenkor is ki kell töltenünk a *k*-val jelölt biteket, de tartalmuk lényegtelen, nem tekinthetők operandusnak.

A számítógépünket könnyedén továbbfejleszthetnénk, hogy mondjuk négy utasítást ismerjen. A műveleti kód részére két bitet kellene elkülönítenünk, amelyeket tetszőlegesen felhasználhatnánk: az egyiket akár közvetlenül összeköthetnénk egy csengővel, így a gép „csengető utasítás"-sal is rendelkezne. Utasíthatnánk a gépünket a kimeneten való műveletvégzésre is: Z mögé egy logikai kaput illeszthetnénk, amely megfelelő műveleti kód esetén negálná azt. Nagyobb kapacitású memória beszerzésével hosszabb programot írhatnánk, és a kimenetek illetve a rajtuk elvégezhető műveletek számát is növelhetnénk. Számítógépünk bővítését a végtelenségig folytathatnánk. Írjuk föl inkább most szerzett tapasztalataink alapján egy általános többutasításos rendszer vázlatát!
Az eszköz két részből áll: a memóriából és a *vezérlő egység*ből (91. ábra). Mint látjuk, a vezérlő egységbe érkeznek a bemenetek. Szándékosan nem „feltételek"-et írtunk, mint az előző rendszernél, mert a feltételek a bemenetek további feldolgozásával, csoportosításával állnak elő. Az általánosított rendszerben a kimenetek is a vezérlő egységen keresztül távoznak, nem közvetlenül a memóriából, így lehetőségünk van az előző bekezdésben említett utólagos feldolgozásukra. Az általunk készített egyszerű rendszer vezérlő egysége a számlálón kívül egyetlen ÉS kapu volt.



91. ábra

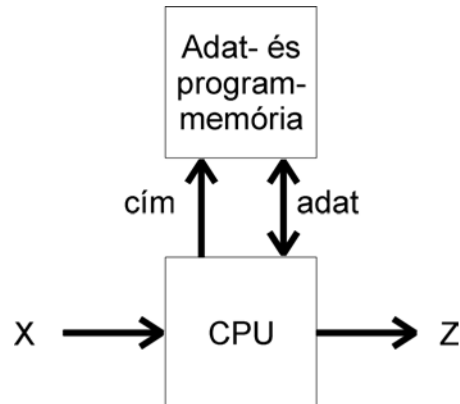
A rendszer hatékonysága nagy mértékben növelhető, ha a kimenetek egy részét egy *adatmemóriá*ban tárolni tudjuk, és a bemenetre visszacsatolva a későbbiekben föl tudjuk használni (92. ábra). A programmemória típusa ROM, hiszen innen az előzőleg betöltött programot olvassuk a futtatás közben. Az adatmemória RWM, mert a fő feladata az ideiglenes adattárolás: egyetlen rekesz tartalma akár többször is megváltozhat a működés során. Tovább növelhetjük a hatékonyságot, ha a sűrűn használt aritmetikai és logikai funkciók ellátását külön erre a feladatra tervezett egységre bízzuk (*ALU=Aritmetic Logic Unit*). A vezérlési, aritmetikai és logikai, illetve egyéb feldolgozási műveleteket elvégző központi egységet *CPU*-nak (*Central Processing Unit, központi feldolgozó egység*) hívjuk.



92. ábra

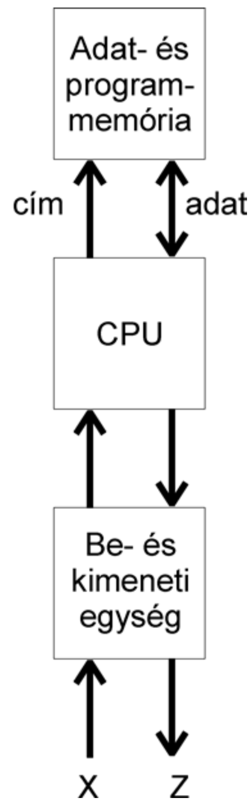
A külön adat- és programmemóriával rendelkező számítógépek a *Harvard architektúra* szerint épülnek fel. A Harvard architektúra előnye az, hogy az adatok memóriába írásával vagy olvasásával egy időben már be is tölthetjük a következő utasítást a programmemóriából, felgyorsítva a rendszer működését. Neumann János, a modern számítógépek szülőatyja a következő ötletet vetette fel: legyen az adat- és programmemória egyben, ezzel

egyszerűsödik a struktúra, és a program saját magán is képes változtatásokat eszközölni (*Neumann-architektúra, 93. ábra*). Manapság mindkét megoldás használatos.



93. ábra

A be- és kimeneti jelek megfelelő elektronikai illesztésére (így a CPU elektronikai védelmére), valamint a beérkező adatok ideiglenes tárolására célszerű be- és kimeneti egységeket (I/O: Input/Output devices) alkalmazni (94. ábra).



94. ábra

Ha a be- és kimeneti egységhez különböző perifériákat (megjelenítő, adatbeviteli és egyéb berendezéseket) csatolunk, ahhoz az eszközhöz jutunk, amelyet a mindennapokban *számítógép*nek hívunk. A számítógép központi vezérlőegységét *mikroprocesszor*nak vagy *mikrovezérlő*nek nevezzük. A mikroprocesszor kezdetben nem volt más, mint az a chip, amely a CPU-t tartalmazta. A mai mikrovezérlőkbe sokszor korlátozott kapacitású program- és adatmemóriát, valamint I/O és más egységeket is integrálnak, aszerint, hogy milyen célra tervezték őket. Az asztalunk alatt zümmögő számítógépek csak egy kis szeletét jelentik a mikroprocesszoros rendszereknek. Napjainkban a riasztórendszerektől a mosógépeken át az autókig mindenütt számítógépekkel találkozunk: ezek tervezői számtalan, kisebb-nagyobb tudású mikrovezérlő közül válogathatnak, sőt akár bélyeg nagyságú komplett számítógépek is a rendelkezésükre állnak.

6.1 A mikroprocesszoros rendszerek főbb egységeinek feladatai

A 94. ábrán bemutatott struktúra ismeretében összegezzük a három fő modul működését és feladatait! Kezdjük a legutóbb tárgyalttal: az I/O egységgel, majd a memóriát és a CPU-t vegyük sorra.

6.1.1 Be- és kimeneti egység

A korszerű be- és kimeneti egységek sokféle, az adatok fogadásával és továbbításával, valamint előzetes feldolgozásával és tárolásával kapcsolatos feladatot ellátnak:

- A be- és kimeneti jelek megfelelő feszültség- és áramszintjeit biztosítják a környezet felé, az esetleges időzítési, ütemezési feltételeknek megfelelően.
- Többféle kommunikációs protokollt ismernek. A digitális adatátvitelben rengeteg szabály van, amelyek az adatok biztonságos, ellenőrizhető átvitelét segítik elő. Szabványos adatátviteli protokollok használatával megkönnyíthetjük számítógépünk és mások által gyártott eszközök összekapcsolódását.
- A be- és kiviteli egységek általában egyszerre több eszközzel tarthatnak kapcsolatot: ezek mindegyike egy-egy kapuhoz, vagyis a külvilág számára írható és/vagy olvasható regiszterhez csatlakozik. Ezekben a regiszterekben tárolhatjuk a beérkező illetve kimenő jeleket, amíg a CPU vagy a számítógépünkhöz csatolt külső eszköz ki nem olvassa, fel nem dolgozza azokat. A kapuk általában a memóriarekeszekhez hasonlóan egyedi címmel rendelkeznek, írásuk és olvasásuk a memóriaelemekéhez hasonló.
- Nagyszámú regiszter vagy memória alkalmazásával lehetőség nyílik a folyamatosan beérkező adatok tömbszerű tárolására az I/O egységen belül, így a CPU-nak nem kell állandóan a kimenetekre figyelnie. Ezt a folyamatot *pufferelés*nek hívjuk.

6.1.2 Memória

A memóriák utasításokat és/vagy adatokat tárolnak. Az utasítások egymás után hajtódnak végre, ily módon programot (software) alkotnak. Az utasítások műveleti kód része megadja, hogy mit végezzen el a CPU (adjon össze két számot, írjon be vagy olvasson ki a memóriából egy bizonyos adatot, stb.). A műveleti kódot az operandusok követik. Az operandusok azok az adatok, amelyek a műveletek végrehajtásához kellenek. Ezek lehetnek számok, amelyeket össze akarunk adni, vagy memóriacímek: velük azokat a rekeszeket jelölhetjük ki, ahol az összeadandó számokat találjuk. A különböző utasításokhoz változó számú operandus tartozhat. A műveleti kód és a hozzá tartozó operandusok vagy ugyanabban a memóriarekeszben foglalnak helyet (ekkor nagyobb szóhosszúságú memóriára van szükségünk), vagy egymás utáni memóriarekeszekben (ekkor nagyobb kapacitású memória kell).

6.1.3 CPU

A CPU két fő részből áll: a *vezérlőegység*ből (*CU=Controll Unit*), ez dolgozza fel a memóriából érkező utasításokat, és vezérli ezeknek megfelelően a számítógép különböző részeit, valamint az *ALU*-ból, amely az aritmetikai és logikai műveleteket végzi. Az ALU mellett minden esetben feltűnik egy ún. *akkumulátor-regiszter* is (esetleg valamilyen más néven, pl. *work-regiszter*). Az akku szerepe a memóriában tárolt operandusok számának csökkentése és az aritmetikai, logikai műveletek végrehajtásának egyszerűsítése. Vegyük például az összeadást: ennek az utasításnak három operandusra van szüksége: két számra, amelyeket összead, továbbá egy memóriacímre, ahova az eredmény kerül. Elvben tehát az összeadó utasítást három operandusnak kellene követnie a memóriában, amelyeket a vezérlő egység három memória-hozzáférési ciklussal és ideiglenes adattárolással – vagyis meglehetősen

bonyolult módon – dolgozna fel. Ennek elkerülésére – gyakorlatilag valamennyi mikrovezérlő esetén, az összes aritmetikai és logikai utasításhoz hasonlóan – az alábbi módszert kell követni:

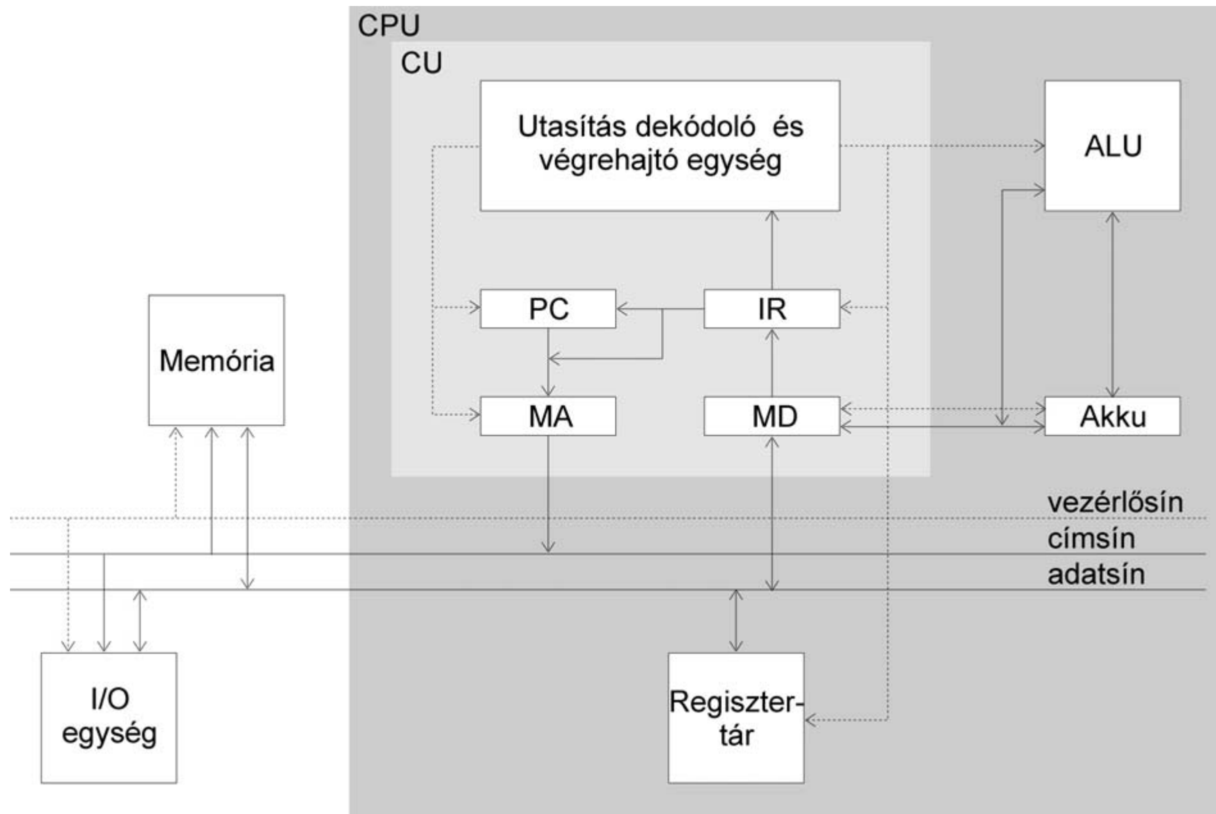
1. Az egyik összeadandó számot előzőleg be kell tölteni az akkumulátor-regiszterbe (erre egy külön utasítás való).
2. Az összeadó utasítást egyetlen operandus követi a memóriában: a másik összeadandó szám. Az ALU az utasítás végrehajtásakor ezt a számot és az akkuban találtat adja össze.
3. Az eredmény az akkumulátor-regiszterbe kerül: ha szükségünk van rá, innen kell kiolvasnunk (egy újabb utasítással).

A CPU-ban több olyan regiszter is található, amelyek az akkuhoz hasonlóan speciális feladatokat látnak el (néhánnyal a későbbiekben megismerkedünk). A *speciális célú regiszterek* mellett *általános célú regiszterek*et is találhatunk: ezeket – mivel írásuk és olvasásuk sokkal gyorsabb a memóriákénál - az adatok gyors, ideiglenes tárolására használhatja a programozó. A regiszterek többnyire tömbösítve, ún. *regisztertár*akban helyezkednek el.

A CPU a hozzá csatlakoztatott memóriaegységekkel, be- és kimeneti egységekkel, regisztertárakkal *cím- adat-* illetve *vezérlősínek*en keresztül tart kapcsolatot. Ezek gyakorlatilag több párhuzamos vezetéket jelentenek, melyeken az adatok, a memóriák egyes rekeszeit kiválasztó címek, és egyéb vezérlőjelek utaznak. A síneken rendszerint több eszköz osztozik, de egyszerre csak egy használhatja őket: ezt a központi egység választja ki az éppen végrehajtott folyamatnak megfelelően.

A 95. ábrán egy egyszerű CPU belső felépítését és kapcsolatait láthatjuk. A processzor egy 8 bites memóriához csatlakozik, amely a programok és az adatok tárolását is végzi (Neumann-architektúra). A CPU 256 utasítást ismer, a műveleti kód így 8 bites, kitölti a memória szóhosszát. Az operandusok is 8 bitesek, és sorra a műveleti kód utáni memóriarekeszekben helyezkednek el. A folyamatos vonallal húzott nyilak cím- és adatutakat jelölnek, míg a vezérlőjelek a szaggatottan jelölt utakon haladnak. Észrevehető, hogy a vezérlőjelek mindig az *utasítás dekódoló és végrehajtó* egységből indulnak ki: ez az egység irányítja, ütemezi az összes többi működését. Némileg leegyszerűsítve annyit csinál, hogy az adatokat

vagy címeket a megfelelő útvonalon vezeti végig, hogy a kívánt helyre eljussanak, majd ha szükséges, beindítja az ALU valamelyik műveletvégző áramkörét.



95. ábra

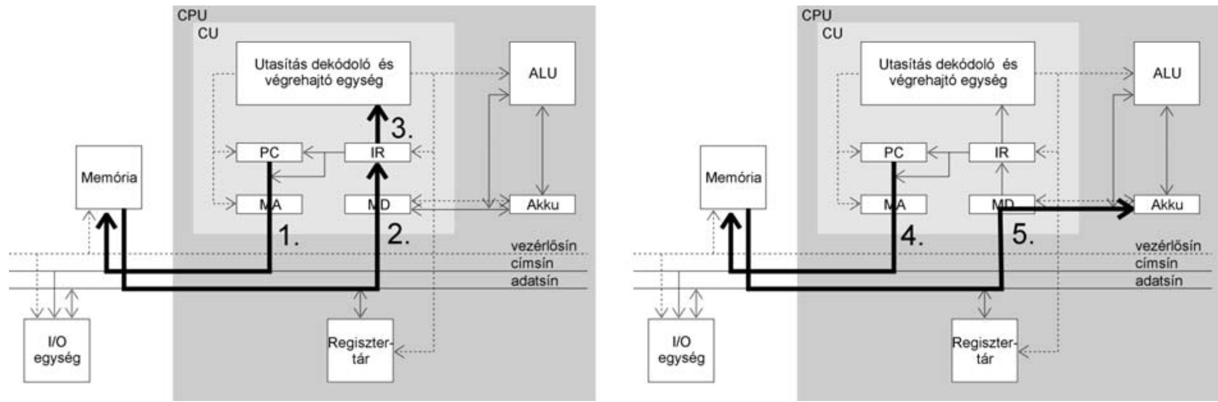
A CU (Central Unit) az utasítás dekódoló és végrehajtó egységen kívül még néhány speciális célú regisztert tartalmaz:

- *PC, Program Counter Register, programszámláló regiszter*: mindig a következő utasítás címét tartalmazza, innen tudja a CPU, hogy éppen hol tart a program. Minden utasításvégrehajtás során egyel nő az értéke.
- *IR, Instruction Register, utasítás regiszter*: ide érkeznek meg egymás után a műveleti kód és az operandusok a memóriából, hogy az utasításvégrehajtó és dekódoló egység kiolvassa és feldolgozza őket. Ugrás esetén innen kerül a következő utasítás címe a PC-be, memóraíráskor illetve -olvasáskor ebből a regiszterből jut el a kívánt cím a memóriához (az MA regiszteren keresztül).

- *MA: Memory Address Register, memória címregiszter*: az MA és MD regiszterek tartják a közvetlen kapcsolatot a memóriával. Az MA-ból jut a memória bemeneteire a kiválasztott rekesz címe (adatírás, -olvasás, utasításbeolvasás esetén).
- *MD: Memory Data Register, memória adatregiszter*: a memóriából kiolvasott adat közvetlenül ide kerül, illetve a memóriába innen töltjük az adatokat.

Most nézzük, hogyan hajtja végre ez a CPU azt az utasítást, amikor egy adott számot írunk az akkumulátor regiszterbe. Ennek az utasításnak egyetlen operandusa van: a beírandó szám. Az utasítás így két memóriahelyet foglal: egyet a műveleti kód, az utána levőt pedig az operandus. Onnan induljunk el, hogy a CPU már befejezte az előző utasítás végrehajtását, a programszámláló regiszter az általunk vizsgált utasítás címét tartalmazza.

1. lépés: A PC-ből az MA-n keresztül a memória bemeneteire jut az utasítás címe (1-el jelölve a 96. ábrán). Amint a memória adatvezetékein megjelenik a rekesz tartalma (vagyis a műveleti kód), az MD-n keresztül az IR-be kerül (2).
2. lépés: A PC értéke eggyel nő (így az operandus címére mutat).
3. lépés: Az utasítás dekódoló és végrehajtó egység beolvassa a műveleti kódot (3), és értelmezi azt.
4. lépés: Az operandus címe a PC-ből a memória bemeneteire jut (4), majd a szám az MD-n keresztül az akkuba (5).
5. lépés: A PC értéke megint eggyel nő, vagyis a következő utasításra mutat: elkezdődhet annak a végrehajtása.



96. ábra

6.2 A gépi kódú programozásról

A bináris formában leírt, közvetlenül a memóriába tölthető utasítássor a *gépi kódú program*. Minden egyes műveleti kód egy bitsorozat, amelyet a CPU értelmezni tud (pl. egy 00110010 kód arra utasíthatja, hogy adjon össze két számot, a 00110011 pedig kivonást jelent). Természetesen az operandusok vagy azok címei is binárisan vannak tárolva a memóriában. Ez a forma a programozó szempontjából átláthatatlan, ezért a program írásakor a műveleti kódokat szavakkal (*mnemonikok*kal) helyettesítjük. Például az

ADD A,5D
INC A

utasítássor azt jelenti (egy bizonyos fajtájú processzornál), hogy adja össze az 5 decimális számot az akkumulátor tartalmával, majd inkrementálja (növelje 1-el) az akkumulátorban tárolt eredményt. Az ilyen formában leírt program az *assembly program*. Az Assembly programot egy egyszerű fordítóprogram (az *assembler*) egyszerű behelyettesítésekkel gépi kódúvá alakítja, amely már a memóriába tölthető.

A mikrovezérlők általában nem túl sok utasítást ismernek. A gépi kódú programban nem találhatjuk meg még az olyan, magas szintű nyelvekben megszokott struktúrákat sem, mint a ciklusok vagy függvények. Az assembly programozás nagy odafigyelést és türelmet kíván: egyszerű, elemi utasításokból kell bonyolult programot írni, mintha LEGO kockákból próbálnánk összerakni az országházat. Az alapvető gépi kódú utasítások a következők:

- bináris aritmetikai utasítások (összeadás, kivonás, stb.),
- logikai műveletek (AND, OR, stb.),
- regiszterműveletek (jobbra-balra léptetés, inkrementálás, dekrementálás),
- bitműveletek (egyetlen bit beállítása valamely memóriarekeszben vagy regiszterben),
- adatátviteli utasítások (a memória és a regiszterek, illetve az I/O egység között),

- feltételes illetve feltétel nélküli ugró utasítások.

Ebből kell tehát gazdálkodnunk. Tudnunk kell, hogy a gépi kód használatát nem kerülhetjük ki: mivel a processzorok csak ezt tudják feldolgozni, a magas szintű nyelveken megírt programokat a fordítók (több lépésen keresztül) gépi kódúra alakítják.

Mivel az assembly programozással külön tantárgy foglalkozik, jegyzetünkben nem tárgyaljuk részletesebben.

6.3 Váratlan események kezelése

A program futása során előfordulhatnak olyan váratlan események, melyek hatására a processzornak meg kell szakítania a feladat végrehajtását és az újonnan kialakult helyzettel kell foglalkoznia. A váratlan esemény kiváltója lehet:

- maga a program: például 0-val való osztás esetén, vagy ha egy memóriarekeszbe vagy regiszterbe túl nagy számot akarunk tölteni (túlcsordulás). A software-es váratlan események elnevezése: *exception (kivétel)*.
- Váratlan eseményt okozhat valamelyik hardware elem is: a számítógépünkhöz kapcsolt perifériák egyike egy előre definiált módon (pl. egy külön erre a célra fenntartott bemeneten) jelzi, hogy meg akarja szakítani a program futását, mert fontosabb közlendője akadt (pl. adatot helyezett a bemeneti kapura, amelynek nem szabad elvesznie). A hardware-es váratlan eseményeket *interrupt*nak (*megszakítás*nak) nevezzük.

Az esemény bekövetkezési helyét tekintve két típust különböztethetünk meg:

- *Szinkron váratlan események* azok, amelyek a program futása szempontjából jól meghatározható helyen következhetnek be. (Nem biztos, hogy bekövetkeznek, ezért váratlanok.) Ilyen pl. a nullával való osztás, és a többi software-es esemény.

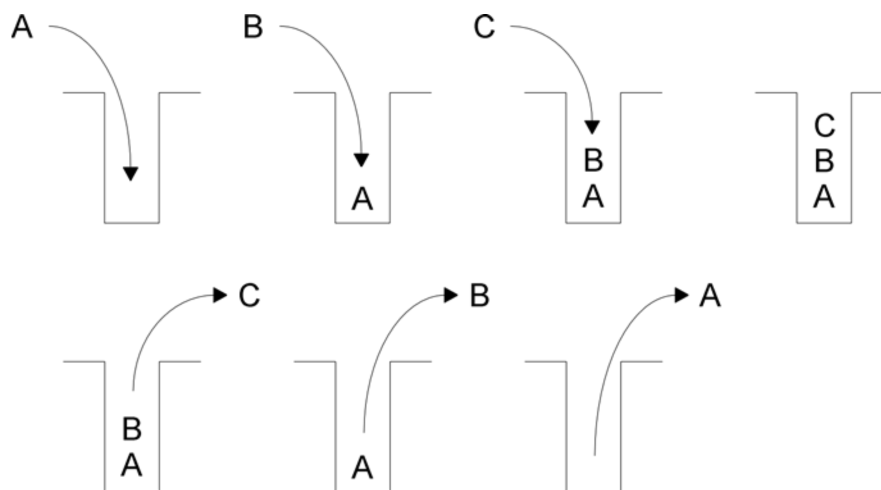
- *Aszinkron váratlan események*: a program futása szempontjából előre kiszámíthatatlan helyen következnek be. A hardware-es események tartoznak ide.

A váratlan eseményekre a programozónak fel kell készülnie: minden lehetséges megszakításra egy-egy programocskát, *szubrutint* kell írnia, amely annak bekövetkezésére reagál. (A kivételek kezelését csak a bonyolultabb mikroprocesszorok támogatják. Legegyszerűbben úgy védekezhetünk ellenük, ha hibátlan programot írunk.) Vegyünk például egy épületriasztó-rendszert. A vezérlő számítógép egy önmagába visszatérő ciklust futtat: várakozik. Egy mozgásérzékelő megszakíthatja ezt a folyamatot: ekkor elindít egy programrutint, amely először is megnézi, hogy élesítve van-e a rendszer, és ha igen, riasztást küld a központnak. Az épületbe belépők személyi kódjukat egy számbillentyűzetbe írják: ez is megszakítást idéz elő, de itt már egy másik program kezd futni, amely például kinyit egy elektromos ajtót.

A váratlan eseményeket úgy kell feldolgozni, hogy a megszakított program ne vegye észre, hogy futása közben bármi is történt. Ezt egy speciális memóriastruktúra: a *verem,* más néven *stack* használatával lehet elérni. A verem elnevezése jól tükrözi a működését: ez ugyanis egy olyan memóriatartomány, amelybe úgy lehet adatokat elhelyezni, hogy mindig csak azt az adatot tudjuk kiolvasni belőle, amelyik legfölül van (97. ábra). Egy program végrehajtása közben a következőképp lehet egy másik programot „feltűnésmentesen” lefuttatni:

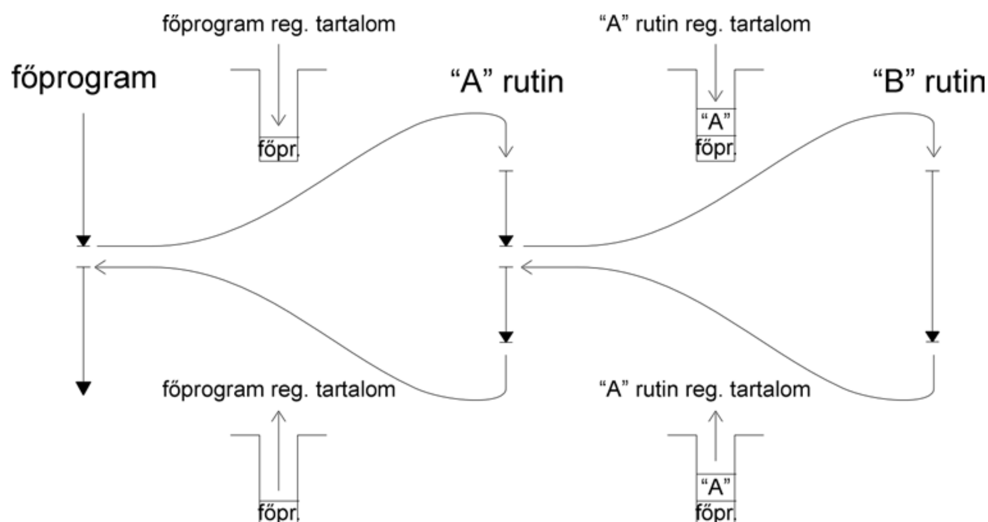
1. lépés: A processzor befejezi az aktuálisan végrehajtott utasítást.
2. lépés: A programszámláló és a regiszterek tartalmát a veremben tárolja.
3. lépés: Futtatja a megszakítást kiszolgáló szubrutint (amely szabadon dolgozhat a regiszterekkel).
4. lépés: Visszatölti a veremből a programszámláló és a regiszterek tartalmát, így az eredetileg futó program nem vehet észre semmit.
5. lépés: *Megszakítás esetén* a program végrehajtását a következő utasítástól folytatja. *Ha kivétel történt*, azt maga a program, vagyis a legutóbb végrehajtott utasítás okozta. Éppen ezért, miután a

szubrutinnal megpróbáltuk kijavítani a hibát, a processzor újra megkísérli végrehajtani a hibát okozó utasítást. Ha ez most sem sikerül, *végzetes kivétel*lel állunk szemben, a program terminálódik.



97. ábra

A verem használata azt is lehetővé teszi, hogy egy megszakítást kiszolgáló szubrutin futását egy nála fontosabb esemény megszakítsa (98. ábra). Az eredetileg futó program regiszterkörnyezetét megszakításkor a verembe töltjük. Amikor a kiszolgáló rutin („A”) futását is megszakítjuk, a regiszterek aktuális tartalmát a verem tetejére helyezzük. A második („B”) rutin befejeztekor az első („A”) környezetét állítjuk helyre azzal, hogy a verem tetejéről visszatöltjük a regiszterek tartalmát. Miután ez a rutin is lefutott, a főprogram regiszterkörnyezetét állítjuk helyre.



98. ábra

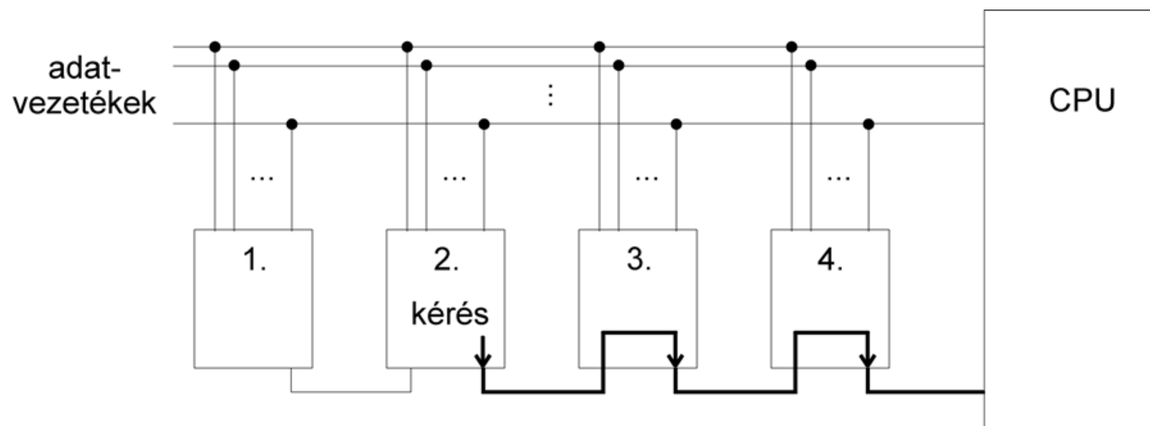
A CPU nem feltétlenül hajtja végre a megszakításokat. A perifériák először *megszakítási kérelmet* küldenek a CPU-nak. A CPU dönt arról, hogy ezt elfogadja vagy sem, s csak ha elfogadta, akkor ugrik a kiszolgáló rutinra. A kérelem elfogadása többféle tényezőtől függ:

- Letiltható-e a megszakítás? *A maszkolható megszakítások* egy erre a célra fenntartott regiszter egy-egy bitjének beállításával tilthatók vagy engedélyezhetők. A *nem maszkolható megszakítások* kiszolgálását nem lehet megtiltani.
- Ha éppen egy megszakítás kiszolgálása közben vagyunk, engedélyezzük-e, hogy egy másik váratlan esemény megszakítsa a folyamatot? Ez a kérdés úgy oldható meg, hogy minden megszakításhoz egy számot rendelünk, amely megadja a fontosságát, vagyis a *proiritás*át. A nagyobb prioritású esemény megszakíthatja a kisebbet, de fordítva ez nem következhet be. A prioritási sorrend bármikor, akár ideiglenes is újradefiniálható.
- Mi történjen egyszerre bekövetkező megszakításkérelmek esetén? A kiszolgálás sorrendje ebben az esetben a CPU fejlettségétől függ: véletlenszerű, vagy prioritásos lehet.

A CPU másik feladata a megszakítási kérelem keletkezési helyének meghatározása. Minden perifériához tartozik egy cím, ahol az őt kiszolgáló

rutin kezdődik. A kérdés az, hogy melyik eszköz kérte a megszakítást, vagyis melyik rutin induljon el. A kiválasztás megoldható *software-es mód*on: egy program bizonyos időközönként sorra lekérdezi a szóba jöhető eszközöket, hogy akarnak-e megszakítást (*polling, lekérdezéses módszer*). Csak nagyon egyszerű esetekben alkalmazzák, helyette különböző *hardware-es módszerek* terjedtek el:

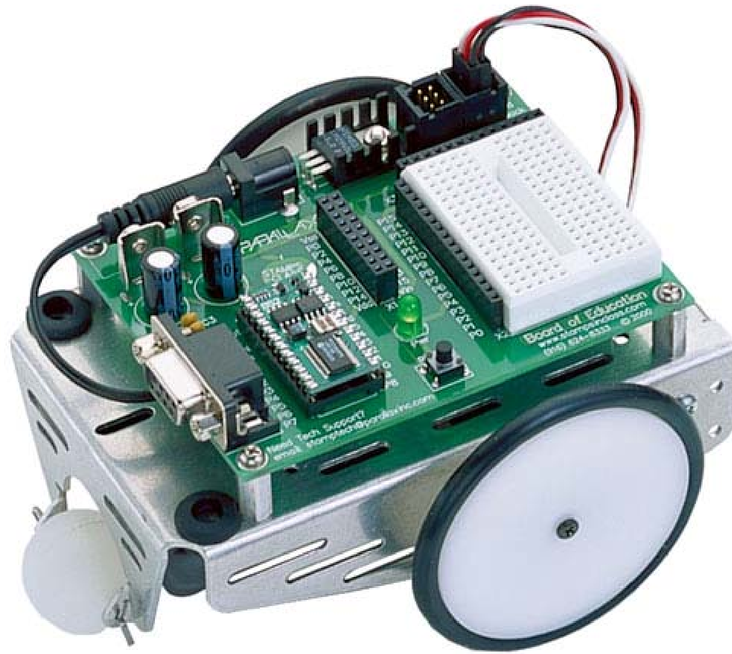
- A legegyszerűbb (ám legköltségesebb) megoldás az, ha mindegyik eszköz külön megszakításkérő vonallal rendelkezik. Ezen a periféria egy megszakításkérő jelet küld a CPU-nak, amely így egyértelműen be tudja azonosítani. Azt elfogadást egy másik vezetéken visszaküldött jellel közli a CPU.
- *Egyetlen megszakítási vonal esetén* az eszközök láncba vannak fűzve (99. ábra). Ezen a láncon keresztül jut el a CPU-ig a megszakítás-kérés. A CPU visszaküld egy azonosítást kérő jelet, amelyet a megszakítást kérő eszköz nyel el (a többi továbbküldi). Ezután az eszköz az adatvonalokon azonosítja magát a következő módszerek valamelyikével:
    - Az adatvonalakon eljuttatja a CPU-nak az őt kiszolgáló rutin címét.
    - A rutint hívó teljes utasítást elküldi.
    - Azt a címet küldi el, ahol a rutint hívó utasítás található. (Ez azért jó, mert ha ezek az utasítások a memória elején vannak, akkor nem kell hosszú címeket küldeni.)
    - Egy sorszámot küld el, amely a memória elején kialakított *megszakítási vektortáblában* kijelöl egy rekeszt. A tábla rekeszeiben helyezkednek el a kiszolgáló rutinok címei. Ezt a módszert *vector interrupt-nak* nevezzük. Ha a táblázatot a CPU tartalmazza, akkor *autovector interrupt*ról beszélünk.



99. ábra

7. Esettanulmány: mikrovezérlővel irányított autonóm működésű robot

Miután áttekintettük a mikroprocesszoros rendszerek működésének és felépítésének elméleti alapjait, jegyzetünk utolsó fejezetében egy gyakorlati példa kerül bemutatásra. A Parallax cég főként oktatási célokra fejlesztette ki a 100. ábrán látható apró robotot. A szerkezetet egy bélyeg nagyságú, BASIC nyelven programozható számítógép vezérli. A mozgását két egyenáramú szervomotor biztosítja, moduláris felépítése pedig lehetővé teszi, hogy a legkülönfélébb kiegészítőket – az elektronikus iránytűtől a rádió adó-vevőn át a kameráig – helyezzük működésbe rajta szinte egyetlen mozdulattal. A robot ismertetését a rendszer magjával, a mikrovezérlővel kezdjük, majd bemutatjuk, hogy ebből a „körülményesen” programozható, viszonylag kevés funkciójú eszközből hogyan varázsoltak a fejlesztők egy könnyen kezelhető, fejlett kommunikációs képességekkel bíró mikroszámítógépet. A tervezők nem álltak meg ennél a pontnál: olyan hordozóáramkört készítettek, amely jól definiált, egységes felületet nyújt a különböző perifériák illesztéséhez, ezzel ösztönözve más gyártókat ötletes kiegészítők gyártására.



100. ábra

7.1 Mikrovezérlő-szint

A robot lelke a Microchip cég által gyártott *PIC16C57c* jelű mikrovezérlő. A CMOS technológiájú eszköz 40 MHz-es órajellel működik, egyetlen utasítást 100 ns alatt hajt végre, vagyis másodpercenként 10 millió utasítást dolgoz fel. A chipnek 28 lába van, ebből 20 használható fel tetszőlegesen ki- vagy bemenetként (2 darab 8 bites és egy 4 bites portba rendezve). Felépítése a Harvard architektúrát követi: 2048 szavas, 12 bit szóhosszúságú beépített programmemóriával és mindössze 72 szavas, 8 bites adatmemóriával rendelkezik. Az ALU is 8 bites: a 0..255 tartományba tartozó számokat ismeri, összeadni, kivonni, léptetni és logikai műveleteket végezni képes rajtuk. (Bár ez szegényesnek tűnhet, a mikroprocesszoros alkalmazások jelentős részében elegendő. Ha mégsem, rengeteg nagyobb tudású, ám drágább mikrovezérlő áll még a tervezők rendelkezésére). Gépi kódban (assemblyben) programozható, az utasítások száma 33. A fent említett aritmetikai és logikai műveleteken kívül adatmozgató, bitkezelő, ugró, valamint különböző speciális regiszterek kezelésére alkalmas utasításokat

ismer. A külvilággal csak kezdetleges módon tud kommunikálni: a kimenetként definiált I/O lábak logikai szintjét állíthatjuk 1-re vagy 0-ra, illetve leolvashatjuk a bemenetekre érkező jelek pillanatnyi értékét. A mikrovezérlő speciális funkciókkal is bír, ilyen például a beépített független működésű számláló, vagy az ún. *Watchdog,* „házőrző” áramkör, amely a program nem várt elakadása esetén azonnal újraindítja a rendszert.

7.2 Mikroszámítógép-szint

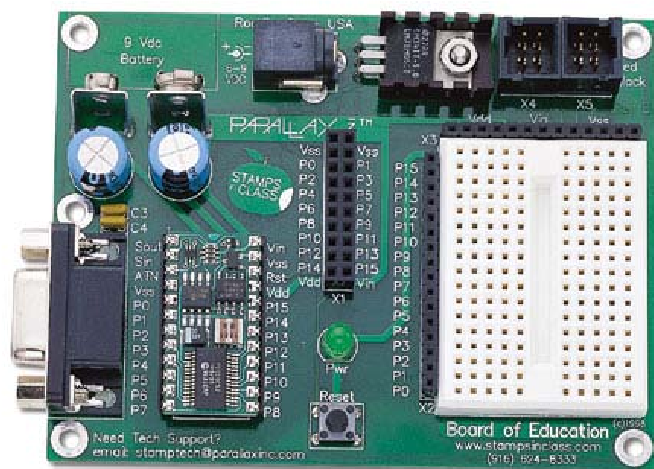
A bemutatott mikrovezérlővel ugyan nagyon sokféle feladat megoldható, de ijesztő a gondolat, hogy szerény kommunikációs képességeivel, assembly nyelven programozva próbáljunk kapcsolatot teremteni mondjuk egy intelligens képfeldolgozó áramkörrel. A Parallax fejlesztői el kívánták gördíteni a fejlődés útjából ezt az akadályt: a mikrovezérlőt néhány áramkörrel kiegészítve egy olyan bélyegnagyságú számítógépet hoztak létre, melynek programozása magas szintű, felhasználó-közeli nyelven történik, jelentősen lecsökkentve a fejlesztésre fordítandó időt és energiát, továbbá ismer néhány szabványos, összetett adatátviteli protokollt, hogy az intelligens eszközökkel való kommunikáció akár egyetlen utasítás kiadásával megoldható legyen. A roboton található számítógép a *BASIC Stamp II.* nevet viseli, egyrészt a programozási nyelvre: a BASIC-re, másrészt az eszköz bélyeg nagyságú méretére utalva (ld. a 101. ábrán, a jobboldalon jól látható a körömnyi nagyságú mikrovezérlő is). Ismeri a szinkron és aszinkron soros, illetve párhuzamos adatátviteli módokat: egyetlen utasítással egész szövegeket, parancsokat küldhetünk egy másik eszköznek, és elolvashatjuk annak válaszát. Képes akár ultrahang-frekvenciáig terjedő modulált szinuszos jelek vagy impulzussorozatok kiadására, le tudja olvasni egy RC kör időállandóját, hogy segítségünkre legyen az analóg jelek bevitelénél. Programozása egy PC-n futó alkalmazással történik szabványos soros porton keresztül. Kezeli a feltételes elágazásokat és ciklusokat is. Ezen kívül fejlett *hiba-visszakereső (debugging) módszer*ek állnak a fejlesztők rendelkezésére.

101. ábra

Az intelligenciának azonban ára van: a BASIC Stamp II. csak nagyjából 4000 utasítást dolgoz fel másodpercenként (szemben a belsejében működő mikrovezérlő 10 millió utasításával!). Programmemóriája kb. 500 utasítás tárolására elegendő, adatmemóriája 32 byte-os (ha viszont ennél jóval nagyobbra lenne szükségünk, akár rádiókapcsolat segítségével a számítógépre is tudunk adatokat menteni).
Most már csak az van hátra, hogy a külső eszközök fizikailag is könnyen csatlakoztathatók legyenek. Ezt biztosítja a következő pontban tárgyalt hordozó áramkör.

7.3 Rendszer-szint

A Basic Stamp II-höz sokféle alaplapot kínálnak. A roboton egy Board Of Education (BOE) elnevezésű, oktatásban hasznos funkciókkal rendelkező tábla kapott helyet (102. ábra).



102. ábra

A tábla baloldalán egy RS-232-es csatlakozó található, ezen keresztül kommunikál a mellette elhelyezett Basic Stamp II. a PC-vel. Egy darab 9V-os elem, vagy 4 darab másfél Voltos ceruzaelem adhatja az energiát, az 5V TTL szintet feszültség-stabilizátor IC állítja elő (fent, középen). Az IC mellet hat darab egyenáramú szervo motor csatlakoztatásához alakítottak ki konnektorokat. A tábla közepén lévő csatlakozóba lehet dugaszolni a különféle periféria-modulokat, többet is egymás tetejére: ezen keresztül jutnak el hozzájuk a Basic Stamp II. I/O lábairól a jelek és a tápfeszültség. Ha mindez nem elég, a jobb oldalon látható panelen tetszőleges TTL áramkört is megépíthetünk.

7.3.1 Perifériák

A legalapvetőbb perifériák a szervo motorok: ezeket impulzusok sorozatával lehet forgásra bírni, sebességük az impulzusok szélességének arányában változik. A mikroszámítógép BASIC nyelvében erre külön utasítást találunk, így a robot mozgatása nem okoz problémát.

Az alaplap és a Basic Stamp II. kialakítása lehetővé teszi bármely olyan intelligens periféria csatlakoztatását, amelyik bedugaszolható a tábla közepén kialakított konnektorba, és ismeri a szabványos soros vagy párhuzamos kommunikációs protokollok egyikét. Ez szinte korlátlan lehetőségeket nyújt robotunk továbbfejlesztésére. Kaphatunk hozzá például nyomkövető, iránytű, LCD kijelző, adó-vevő, ultrahangos távolságérzékelő modulokat, de irányíthatjuk számítógépes egérrel is a megfelelő egység beszerzése után. A robothoz kifejlesztett intelligens kamera képes meghatározott színű tárgyak helyzetét követni, vagy nagyságukat meghatározni.

Látnunk kell, hogy ehhez a sokszínű felhasználhatósághoz és bővíthetőséghez csak a bemutatott fejlesztési lépések során juthattunk el. Bár a felhasznált mikrokontoller önmagában is el tudná látni a robot irányításának a feladatát, a perifériákat pedig vele együtt közvetlenül beforraszthattuk volna egy nyomtatott áramköri lapkába, a robot megtervezése és felépítése valószínűleg felőrölte volna az energiáinkat. A mikroprocesszoros (és bármely egyéb mérnöki) rendszerek különböző absztrakciós szintjeinek kidolgozása lehetővé teszi, hogy a megvalósítani kívánt célra koncentrálhassunk, és ne kelljen mindent újra az alapoktól felépítenünk.

8. Irodalomjegyzék

- Dr. Arató Péter: Logikai rendszerek tervezése
  Tankönyvkiadó, Budapest, 1990
- Dr. Gál Tibor: Digitális rendszerek I-II.
  Tankönyvkiadó, Budapest, 1991
- Dr. Szittya Ottó, Hunwald György: Logikai elemek adatgyűjteménye
  Tankönyvkiadó, Budapest, 1990
- U. Tietze – Ch. Schenk: Analóg és digitális áramkörök
  Műszaki Könyvkiadó, Budapest, 1990
- Zsom Gyula: Digitális technika I-II.
  Műszaki Könyvkiadó, Budapest, 2000
- www.microchip.com
- www.parallaxinc.com

9. Tartalomjegyzék

1. BEVEZETÉS 1

2. ALAPFOGALMAK 2

3. KOMBINÁCIÓS HÁLÓZATOK 5

3.1 A kombinációs hálózatok működésének igazságtáblás felírása 5

3.2 A kombinációs hálózatok működésének definiálása logikai függvényekkel 6
3.2.1 Példa: a „folyosó-kapcsolás" logikai függvényének meghatározása 7
3.2.2 Példa: logikai függvény felírása az igazságtáblából 8

3.3 Logikai függvények algebrai egyszerűsítése 8
3.3.1 Példa algebrai egyszerűsítésre 9

3.4 Logikai kapcsolási vázlat 9
3.4.1 Példa: logikai kapcsolási vázlat felrajzolása a logikai függvényből 12
3.4.2 Példa: logikai függvény felírása a logikai kapcsolási vázlatból 12
3.4.3 Példa: kombinációs hálózat megvalósítása kizárólag NOR illetve NAND kapuk felhasználásával 13

3.5 Logikai függvények kanonikus (normál) alakjai 14
3.5.1 Diszjunktív kanonikus alak 14
3.5.2 Konjunktív kanonikus alak 15

3.6 Logikai függvények grafikus minimalizálása 17
3.6.1 4 változós Karnaugh-tábla 23
3.6.2 Nem teljesen definiált logikai függvény grafikus minimalizálása 26
3.6.3 Konjunktív alak felírása Karnaugh-táblával 27
3.6.4 Nem teljesen definiált függvény konjunktív alakja 29
3.6.5 Több kimenetű kombinációs hálózatok grafikus egyszerűsítése 29
3.6.6 Példa grafikus egyszerűsítésre: hét szegmenses dekódoló tervezése 32

3.7 Logikai áramkörök jellemzői 38
3.7.1 Integrált áramkörök feszültségszintjei, DC zajtávolság 39
3.7.2 AC (váltakozó áramú) zajtávolság 41
3.7.3 Logikai áramkörök dinamikus jellemzői 42
3.7.3.1 Felfutási és lefutási idő 42
3.7.3.2 Késleltetési vagy terjedési idő 42

3.7.3.3 Terhelhetőség, Fan-out 43
3.7.3.4 Egység-terhelés, Fan-in 44
3.7.3.5 Egyéb jellemzők 44
3.7.4 A TTL és CMOS elemcsaládok összehasonlítása 45
3.7.5 Speciális kimeneti kapcsolatok TTL áramköröknél 46
3.7.5.1 Nyitott kollektoros kimenetek 46
3.7.5.2 Háromállapotú (tristate) kimenetek 47

3.8 A jelterjedési idők hatása a kombinációs hálózatok működésére 48
3.8.1 A statikus hazárd 48
3.8.1.1 Példa statikus hazárdmentesítésre 52
3.8.1.2 Statikus hazárdok konjunktív hálózatokban 53
3.8.2 A dinamikus hazárd 54
3.8.3 A funkcionális hazárd 54

3.9 Néhány gyakrabban használt kombinációs hálózat 57
3.9.1 Kódátalakító egységek 57
3.9.2 Kódoló 58
3.9.3 Dekódoló 59
3.9.4 Adatút-választó eszközök 61
3.9.4.1 Multiplexer 61
3.9.4.2 Demultiplexer 62
3.9.5 Bináris aritmetikát végző áramkörök 63
3.9.5.1 1 bites teljes összeadó 63

4. SZEKVENCIÁLIS (SORRENDI) HÁLÓZATOK 67

4.1 Állapotgráfos leírás 69

4.2 Állapottáblás felírás 69

4.3 Aszinkron és szinkron sorrendi hálózatok 70

4.4 Elemi sorrendi hálózatok (flip-flopok) 71
4.4.1 D tároló 71
4.4.2 S-R tároló 72
4.4.2.1 Az S-R tároló megvalósítása NOR illetve NAND kapukkal 75
4.4.3 J-K tároló 76
4.4.4 T tároló 78
4.4.5 Master-Slave flip-flopok 80
4.4.6 A tárolók jellegzetes alkalmazásai 81
4.4.6.1 Pergésmentesítés 81

4.4.6.2 Frekvenciaosztás 83
4.4.6.3 Aszinkron számlálók 83
4.4.6.4 Szinkron számlálók 86
4.4.6.5 Még néhány szó a számlálókról 88
4.4.6.6 Regiszterek 88

5. MEMÓRIÁK 91

5.1 Kombinációs hálózatok megvalósítása programozható logikai elemek felhasználásával 94

6. A MIKROPROCESSZOROS RENDSZEREK ALAPJAI 98

6.1 A mikroprocesszoros rendszerek főbb egységeinek feladatai 108
6.1.1 Be- és kimeneti egység 108
6.1.2 Memória 109
6.1.3 CPU 109

6.2 A gépi kódú programozásról 114

6.3 Váratlan események kezelése 115

7. ESETTANULMÁNY: MIKROVEZÉRLŐVEL IRÁNYÍTOTT AUTONÓM MŰKÖDÉSŰ ROBOT 121

7.1 Mikrovezérlő-szint 122

7.2 Mikroszámítógép-szint 123

7.3 Rendszer-szint 124
7.3.1 Perifériák 125

8. IRODALOMJEGYZÉK 127

9. TARTALOMJEGYZÉK 128