

Informatika alapjai

Logikai áramkörök

Logikai áramkörök, Boole algebra

A digitális technikán alapuló eszközök már régóta a környezetünk részei. A mérnökök mindennapi munkaeszközei a számítógépek és a digitális műszerek. Ezért működésük alapelveit azoknak sem árt ismerni, akik csak használják, de nem tervezik ezeket.

Logikai érték, műveletek a Boole algebraiban

A algebraikban egy halmazon értelmeznek műveleteket. A Boole algebraiban a halmaz a logikai konstansok (0: hamis, 1: igaz) és a logikai változók (A, B, C,...) melyek a konstansok értékét vehetik fel). A műveleteket az alábbiakban vezetjük be.

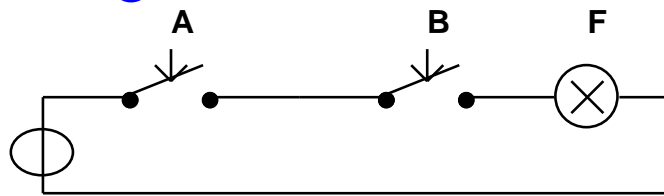
Az ÉS (AND) művelet

Az alábbi ábrán *sorba kötött* kapcsolók szerepelnek.

A “kapcsoló le van nyomva” állítás 2 értéket vehet fel: igaz, hamis. A hozzá rendelt változó értéke ennek megfelelően 0 vagy 1.

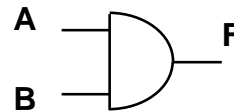
Ugyanez vonatkozik a “lámpa világít” állításra is.

Az igaz értéket 1-el, a hamisat 0-val jelöljük.



Magyarozó ábra

ÉS (AND) kapu



kapcsolási rajz szimbólum

Az F lámpa akkor világít, ha A **ÉS** B kapcsoló le van nyomva ($A = 1$ **ÉS** $B = 1$).

- Ez a **logikai ÉS (AND, szorzás) művelet**.
- Leírása a **Boole algebrában**: $F = A * B$
- Az alpműveleteket egy digitális áramkörben (többek között) **logikai kapukkal** lehet megvalósítani. (Ezek más elven működnek, mint a kapcsolós mintapéldák áramkörei. Utóbbiak azonban a könnyebb érthetőség miatt alkalmasabbak a Boole algebra magyarázatára.)
- A valóságos logikai kapu be és kimenetein az igaz (1) értéknek egy magasabb feszültség, a hamisnak (0) egy alacsonyabb feszültséget feleltetnek meg.
- Az logikai ÉS az egyik legegyszerűbb **logikai függvény**.

- A logikai függvényeket logikai változókon végzett logikai műveletekkel állíthatjuk elő.
- A *logikai változók* és a belőlük előállított függvény a (2 értékű logikában) *csak a 0 és 1 értéket vehetik fel.*
- Az *AND* függvény csak akkor 1, ha minden változója 1. (Itt a lámpa csak akkor világít, ha mindkét kapcsoló le van nyomva.)
- *A logikai függvényeket táblázat segítségével teljesen specifikálhatók.* Minden lehetséges bemenethez meg tudjuk adni a kimenetet. (Ez nem igaz pl. a valós függvényekre.)
- A logikai függvények táblázatos megadását *igazságtáblának* nevezzük.
- A 2 változós *AND* művelet igazságtáblája:

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

Tetszőleges számú változó AND kapcsolata csak akkor 1, ha mindegyik 1.

$$F = A * B * C * \dots = 1, \text{ ha } A, B, C, \dots = 1$$

- *A logikai szorzás (ÉS, AND) műveleti jelét többnyire nem szokták kiírni: $F = A * B * C = ABC$*

Tetszőleges sok sorba kapcsolt kapcsoló esetén a lámpa csak akkor világít, ha az összes kapcsoló be van kapcsolva. (Nem világít, ha bármelyik ki van kapcsolva.)

Az ÉS művelet további tulajdonságai:

$A * 0 = 0$ (Ha az egyik sorba kötött mindig nyitva, akkor nem világít a lámpa.)

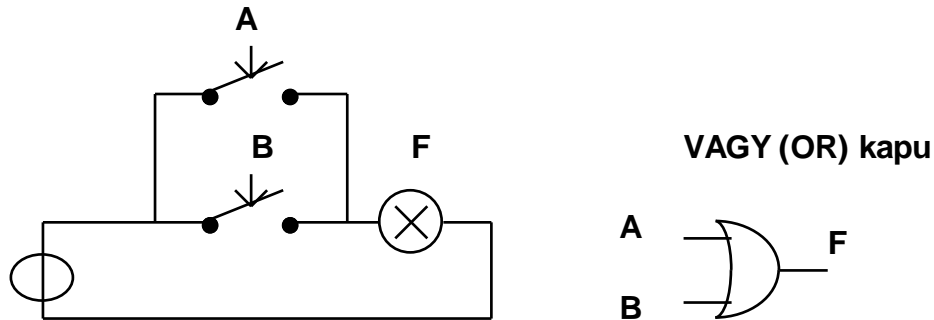
$A * 1 = A$ (Ha az egyik sorba kötött mindig zárva, akkor az olyan, mintha ott sem lenne.)

$A * A = A$ (Egyszerre működtetett sorba kötött kapcsolók úgy működnek mint egyetlen egy.)

$A * B = B * A$ kommutativitás, felcserélhetőség (A sorba kötés sorrendje lényegtelen.)

A VAGY (OR) művelet

Az alábbi ábrán párhuzamosan kötött kapcsolók szerepelnek.



Az *F* lámpa akkor világít, ha *A VAGY B* kapcsoló le van nyomva. Egyik, másik *vagy mindkettő*. (A magyar nyelvben a „vagy” nem egészen ezt jelenti...)

- Ez a *logikai VAGY (OR, összegzés) művelet*.
- Leírása a Boole algebrában: $F = A + B$
- Értéke akkor 1, ha bármely változója 1.

A 2 változós *OR* művelet igazságtáblája:

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

Tetszőleges számú változó OR kapcsolata *akkor 1, ha bármelyik 1, csak akkor 0, ha mind 0.*

$$\mathbf{F = A+B+C+ \dots = 0, ha A,B,C.. = 0}$$

Tetszőleges számú párhuzamosan kapcsolt kapcsolóval sorba kötött lámpa világít, ha bármely kapcsoló be van kapcsolva. (Nem világít, ha egyik sincs bekapcsolva.)

További tulajdonságok:

$\mathbf{A + 0 = A}$ (Ha a kapcsolóval egy szakadást kötünk párhuzamosan, az nem befolyásolja a kapcsoló működését.)

$\mathbf{A + 1 = 1}$ (Ha a kapcsolóval rövidzárat kötünk párhuzamosan, a lámpa mindig világít.)

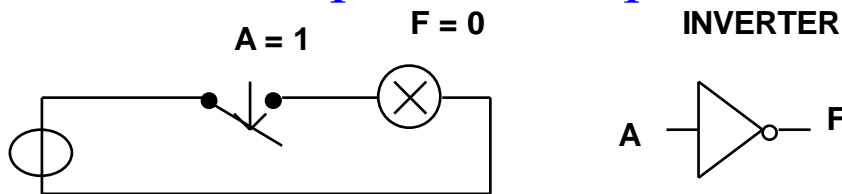
$\mathbf{A+A = A}$ (Tetszőleges számú egyszerre vezérlés párhuzamosan kötött kapcsoló ugyanúgy viselkedik, mint egyetlen egy)

$\mathbf{A+B = B+A}$ kommutativitás, felcserélhetőség (A párhuzamosan kötés sorrendje lényegtelen.)

A logiai szorzás prioritása nagyobb mint a logikai összeadásé: $A*B + A*C$ (a * műveletet nem kell zárójelbe tenni.)

Az invertálás

Az alábbi kapcsoló kikapcsol, ha lenyomják.



Az F lámpa akkor világít, ha A kapcsoló nincs lenyomva.

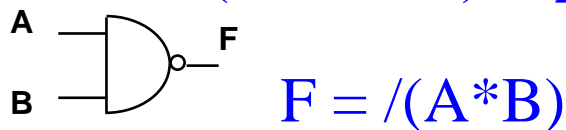
- Ez az invertálás (negálás) művelet. (A kapcsoló *lenyomott állapotához* rendeltük az *A* változó *igaz értékét*, de ekkor a *lámpa nem világít.*)
- Leírása a Boole algebrában: $F = \bar{A}$
Ezt billentyűzetten egyszerűbb így írni:
 $F = /A$
- A negálás igazságtáblája:

A	F
0	1
1	0

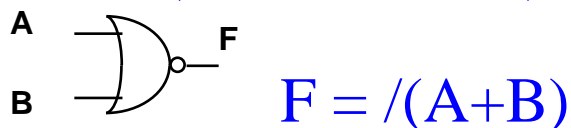
További tulajdonságok: $/1=0$, $/0=1$, $A*/A=0$,
 $A+/A = 1$, $//A = A$ páros számú negálás
önmagát adja, $///A = /A$ páratlan számú
negálás a negáltat adja

Az előbbi kapuknak a negált változata is létezik, ezek igazságtáblájában a kimenet az eredeti negáltja. A negálást itt egy kis kör jelöli (ezt máshol is szokás így jelölni):

NAND (NEM ÉS) *kapu*



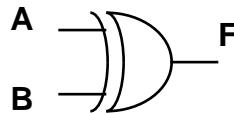
NOR (NEM VAGY) *kapu*



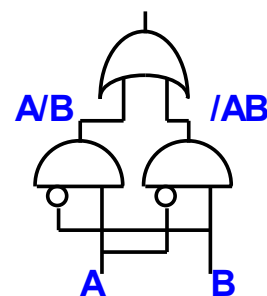
Különleges kapu a **KIZÁRÓ VAGY** kapu (eXclusive OR) amit **XOR** kapunak neveznek. Ennek kimenete *csak akkor 1, ha a két bemenetének logikai értéke ellentétes.*

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

Rajzjele: Felépítése kapukkal:



$$F = A \text{ XOR } B = A/B + /A/B$$



Néhány további Boole algebrai azonosság:

Kommutativitás (felcserélhetőség):

$$A + B = B + A, \quad A * B = B * A$$

Asszociativitás (csoportosíthatóság):

$$A + (B + C) = (A + B) + C$$

$$A * (B * C) = (A * B) * C$$

Disztributivitás (feloszthatóság):

$$A * (B + C) = A * B + A * C$$

$$A + (B * C) = (A + B)(A + C) \text{ szokatlan!}$$

Elnyelés:

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

De' Morgan azonosságok:

$$\neg A + \neg B = \neg(A * B), \quad \neg A * \neg B = \neg(A + B)$$

A fentiek beláthatók a jobb és baloldal igazságtábláinak felírásával.

Dualitás elve:

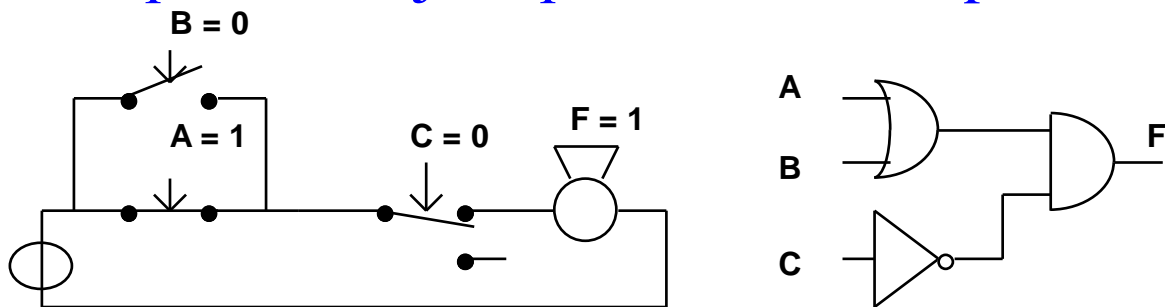
$A + \leftrightarrow *$, és $0 \leftrightarrow 1$ felcserélésével is igazak maradnak a Boole algebrai azonosságok.

Pl. Lásd De' Morgan, és alább

$$A * 1 = A, \quad A + 0 = A$$

Tervezzünk az összes műveletet felhasználó áramkört! A riasztó akkor jelezzon, ha az ajtó érzékelő (A) vagy az ablak érzékelő (B) jelez és a riasztó tiltó kapcsoló (C) nincs tiltás állapotban.

A kapcsolási rajz kapcsolókkal és kapukkal:



A logikai függvény megadása a kapcsolás alapján, általános Boole algebrai alakban:

$$F = (A + B) * \bar{C}$$

Az F függvény igazságtáblája:

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

A bemeneti változók egy konkrét értékét *bemeneti kombinációnak* nevezzük. (Pl. ABC: 110)

A logikai függvény felírható az igazságtábla alapján. Itt az ún. **SOP** (Sum Of Product, *szorzatok összege*) kanonikus (egyszerűsítetlen) alakú felírást mutatjuk meg. Az *igazságtábla 1-eseit logikai szorzatokkal fejezzük ki*. Az egyes szorzatokban *az 1 értékű logikai változók ponáltan (nem negáltan), a 0 értékűek negáltan szerepelnek*, mivel a *szorzat pontosan ilyen változó értékeknél ad 1-et*.

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

/AB/C
A/B/C
AB/C

Pl. $\neg ABC=1$, ha $A=0, B=1, C=1$

Ezek OR kapcsolata akkor 1, ha bármely szorzat tag 1, így a teljes függvényt megadja.

$$F = \neg AB/C + A/B/C + AB/C$$

A közvetlenül az igazságtábla alapján felírható a függvény többnyire nem a legegyszerűbb alakú. Itt a legegyszerűbb alak: $F = (A + B)/C$

- A logikai függvényeket megvalósító logikai hálózatot *kombinációs hálózatnak* nevezzük.
- Tervezés során általában az igazságtáblából kiindulva egyszerűsítő (minimalizáló) eljárások alkalmazásával jutunk el a legegyszerűbben megvalósítható kombinációs hálózathoz. Itt nem foglalkozunk az egyszerűsítéssel.

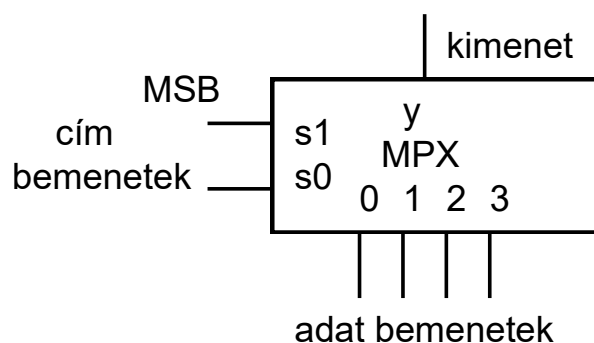
Kombinációs funkcionális elemek

A tervezés során bizonyos logikai funkciókra gyakran szükség van. Ezért megalkották az ún. funkcionális elemeket. A kombinációs funkcionális elemeket kombinációs hálózat valósítja meg. Itt a teljesség igénye nélkül csak néhányal foglalkozunk.

Multiplexer (adat választó)

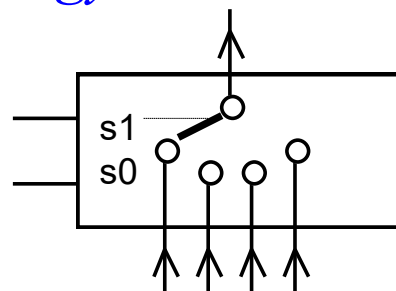
A multiplexer funkciója ***adat választás***.

rajzjele:



a)

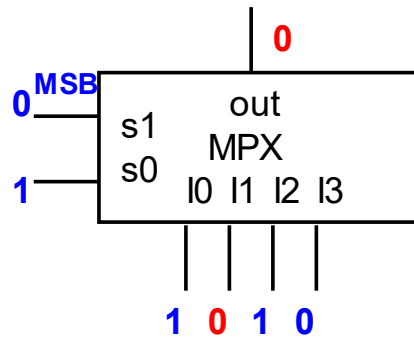
***viselkedését
magyarázó ábra:***



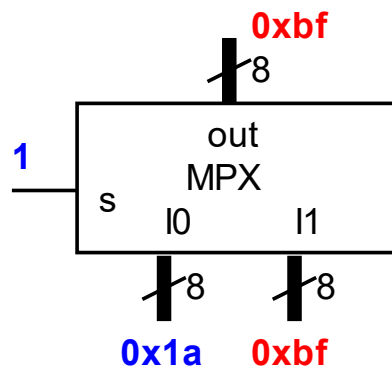
b)

A fenti „a” ábrán egy 4 db 1 bites ***adat bemenet közül választani képes*** ún. 4/1-es multiplexer rajzjele látható. A jobb oldali „b” ábra a működés megértését segíti. A bemenet kiválasztása az *s1s0 select* jelekkel történik. ***A multiplexer kimenetén a kiválasztott bemeneten levő logikai érték jelenik meg.***

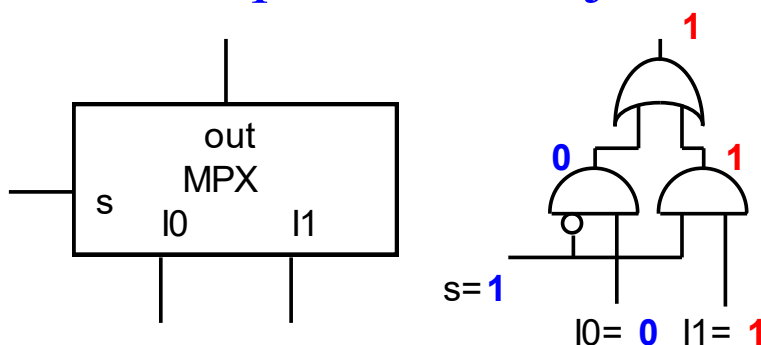
Pl. ha $s_1s_0 = 01$, akkor az I1 bemeneten levő érték.



A bemenetek és a kimenet több bites is lehet. Pl. 2 db 8 bites bemenetből választani képes multiplexer, ha $s = 1$, az I1 bemeneti adatot választja ki:

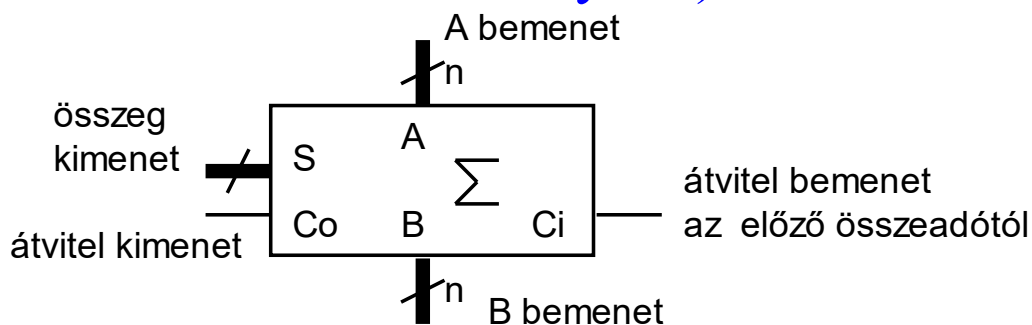


A legegyszerűbb a 2/1-es multiplexer amelynek 1 bitesek a bemenetei. Ennek a belső felépítését is lerajzoltuk.



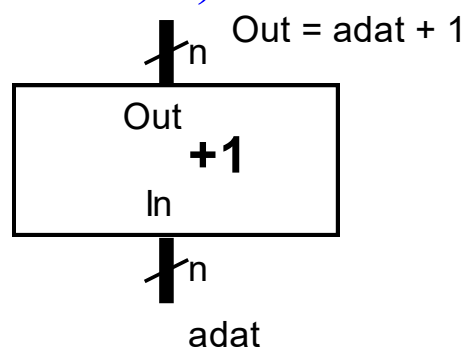
Összeadó

Az összeadó az A és B bemenetére adott n bites számok összegét adja az S (summma) n bites kimenetén és Co (carry, átvitel) kimenetén jelzi, ha túlcsordulás (átvitel) van. Ci egy bites bemenetén az előző egységről jövő túlcsordulást veszi figyelembe (Ci = 1 esetén 1-et hozzáad az eredményhez).

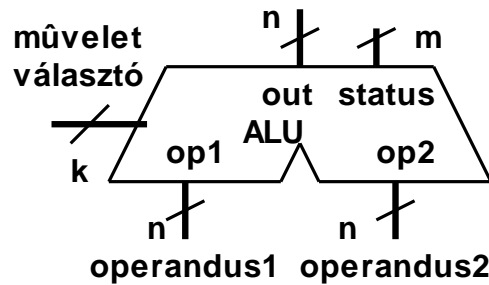


Inkrementer

1-et ad hozzá az n bites bemenetére érkező számhoz. (Összeadóból is könnyen készíthető, de annál egyszerűbb.)



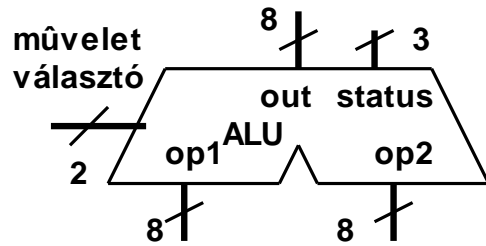
ALU (aritmetikai logikai egység)



Egy viszonylag bonyolult kombinációs funkcionális elem az ALU. Ez *aritmetikai és logikai műveleteket* tud végezni a bementére adott adatokon. A művelet eredménye az *adat kiementén* (out) jelenik meg. A művelet eredménye alapján bizonyos többlet információkat is ad, ezt nevezik *státus kiemenetnek* (pl. hogy az összeadás során volt-e átvitel, azt a carry státus bit jelzi). *Az elvégzendő műveletet a művelet választó bemenetre adott kombinációval lehet kijelölni.*

Aritmetikai műveletek: pl. összeadás, kivonás
Logikai műveletek: pl. bitenkénti AND, OR, XOR, invertálás stb.

Egyszerű példa ALU-ra



- Adat bemenetek mérete 8 bit
- műveletek: összeadás, kivonás: (előjel nélküli és 2-es komplementes számokra is működnek), AND, OR (művelet kiválasztó bemenetek: s1,s0)
- státus információ:
 - Z:** a művelet eredménye 0
 - N:** az eredmény (2-es komplementes) előjel bitje 1 értékű vagyis *negatív* (logikai műveletek nem állítják elő, vagyis ilyenkor 0)
 - C:** *túlcsordulás* volt a művelet során (logikai műveletek nem állítják elő, vagyis ilyenkor 0)

Példaként megmutatjuk, milyen kimenetet ad, az alább megadott műveleti kódok esetén, ha az operandusok:

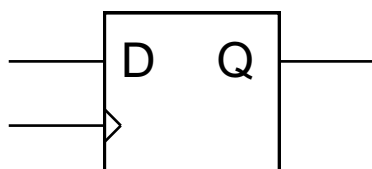
$$\text{op1} = 11001011$$

$$\text{op2} = 00110100$$

s1s0	művelet	eredmény	Státus:Z,C,N
00	ADD	11111111	0,0,1
01	XOR	11111111	0,0,0
10	AND	00000000	1,0,0

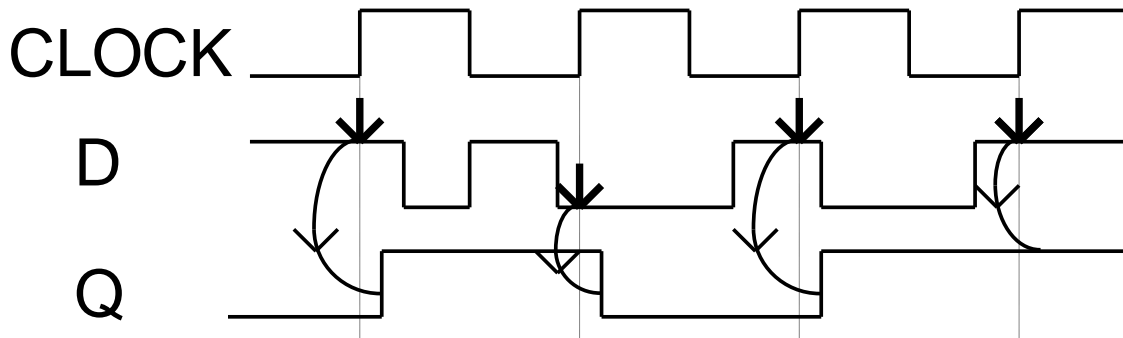
Szinkron sorrendi hálózatok (állapotgépek, synchronous Final State Machines)

A kombinációs hálózatok kimenete mindig csak az aktuális bemenettől függ. Ezzel szemben *a sorrendi hálózatok emlékeznek* valamire *az előző bemeneti sorozatból*. Így *ugyanarra a bemeneti kombinációra később más kimeneti értéket adhatnak*. Itt csak *szinkron sorrendi hálózatokról* lesz szó. *Működésüket* egy periodikus négyszögjel, az *órajel aktív éle ütemezi*. Tipikusan *minden változás* csak az órajel (clock) *felfutó élre* történik. (Az órajel 0-1 átmenete az aktív él.) Az emlékezetet memória jellegű logikai áramkörök biztosítják. Ezek a *flip-flop*-ok. Több típusa van itt csak a leggyakrabban használt szinkron *D flip-flop*-ról beszélünk.

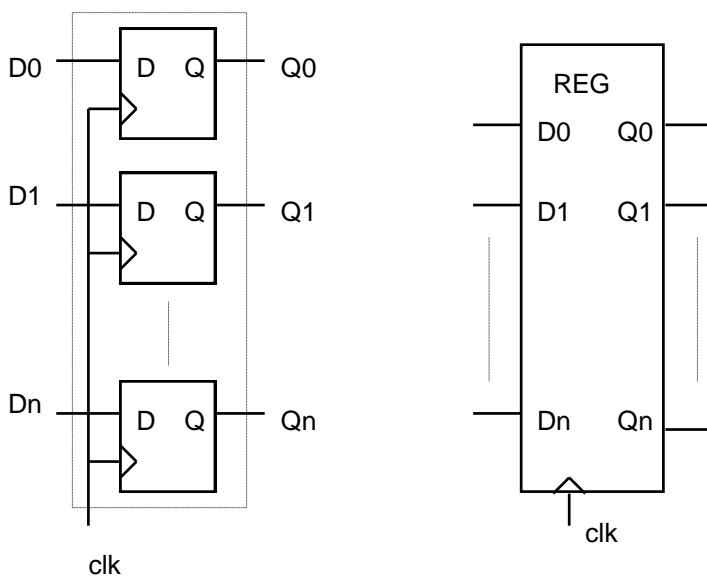


felfutó él érzékeny
D flip-flop

A D bemenet értékét megjegyzi az órajel felfutó élénél és a kimenetén ezt az értéket tartja a következő órajel felfutó élig.

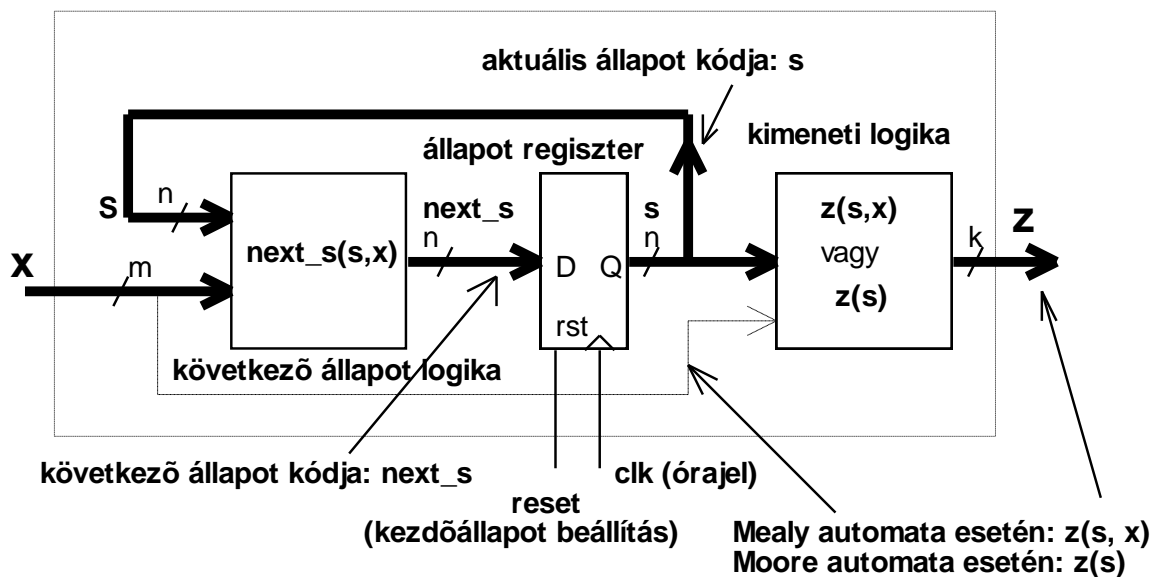


A több közös órajelű D flip-flopból készített tárolót *regiszternek* nevezzük. Ez tulajdonképpen egy olyan *sorrendi funkcionális elem*, mely minden órajel élre eltárolja a bemenetén levő értéket (a következő órajel élig).



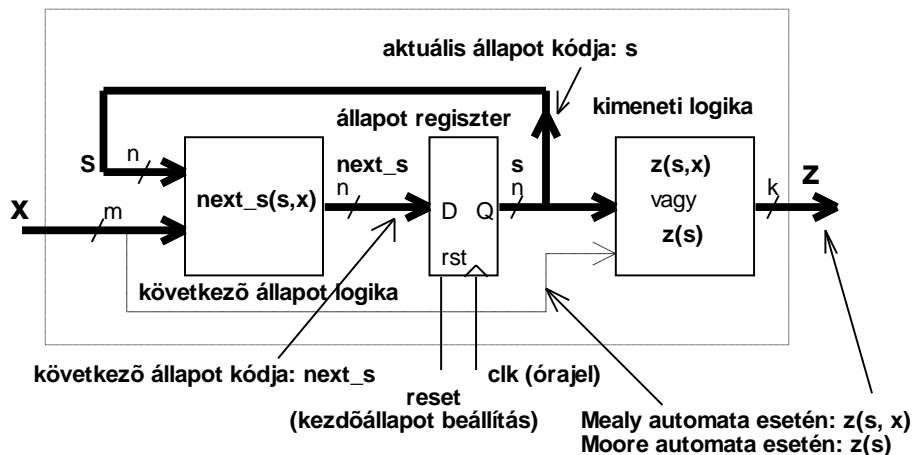
A szinkron sorrendi hálózat felépítése

Az általános állapotgépet *FSM*-nek (Finite State Machine), állapotgépnek is szokás nevezni.



Az *állapotregiszter* tárolja az „emlékezni valót”, ez egy n -bites kód (state: s), ezt nevezzük az aktuális állapot kódjának.

Az X bemenet tetszőleges bismámú lehet. Ezek a változók az FSM bemenetei. (A példákban az egyszerűség kedvéért 1 bites X bemenet szoktunk használni.)

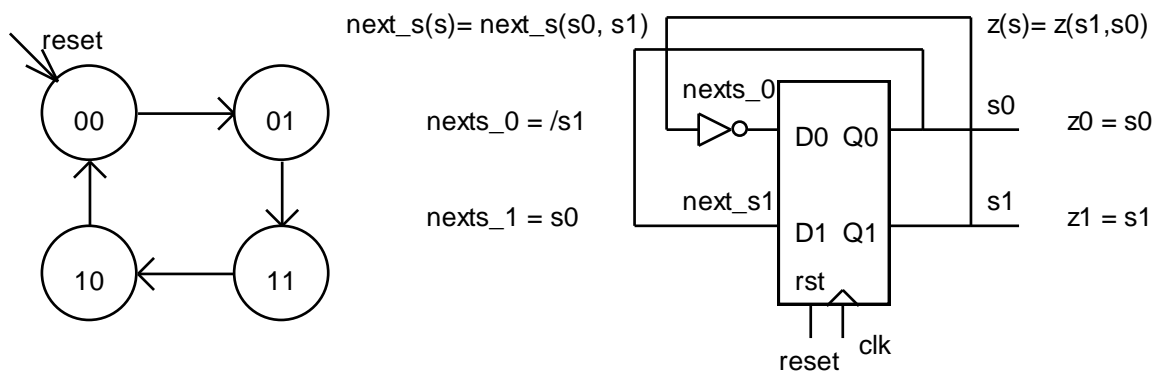


- A **következő állapot logika** (kombinációs hálózat) egy $next_s(X, s)$ n kimenetű logikai függvényt állít elő (n db logikai függvény, melyeknek ugyanazok a bemeneti változói) az **aktuális állapot kód** (s) és **bemenet** (X) alapján előállítja a **következő állapot kódját** ($next_s$, ez is n bites).
- A $next_s$ következő órajel felfutó élére **eltárolódik az állapotregiszterben**. Ez lesz az új aktuális állapot: s .
- Az **állapotregiszter tartalma tehát megváltozik** az órajel felfutó élre, **ha a következő állapot kódja eltér az aktuális állapot kódjától**.
- A kimenetet a **Z kimeneti függvény** állítja elő az **aktuális állapot kódja** és esetleg az aktuális **bemenet** alapján. Mivel a kimenet nem csak az **aktuális bemenettől** (X), hanem az **állapottól** (az aktuális állapot kódjától) is függ, ugyanazon bemenetre a későbbiekben más lehet a Z kimenet.
- Az állapotregisztert bekapcsoláskor egy **kezdőállapotba** kell állítani (reset). Ez azt az információt jelenti, hogy az állpotgép által elvégzendő feladat megoldása még nem kezdődött el.

Állapotgráf

A sorrendi hálózatokat *állapotgráffal* (és *állapottáblával*) írjuk le (specifikáljuk). A *gráfpontokat az állapotoknak* (emlékezet), *az irányított éleket az állapotátmeneteknek* (állapot váltás) *feleltetjük meg*.

Pl. Egy 4 állapotú olyan hálózat állapot gráfja, melynek nincs bemenete (az órajelet nem tekintjük bemenetnek) és kimenete közvetlenül az állapotregiszter (nincs kimeneti kombinációs hálózat, $z(s) = s$):

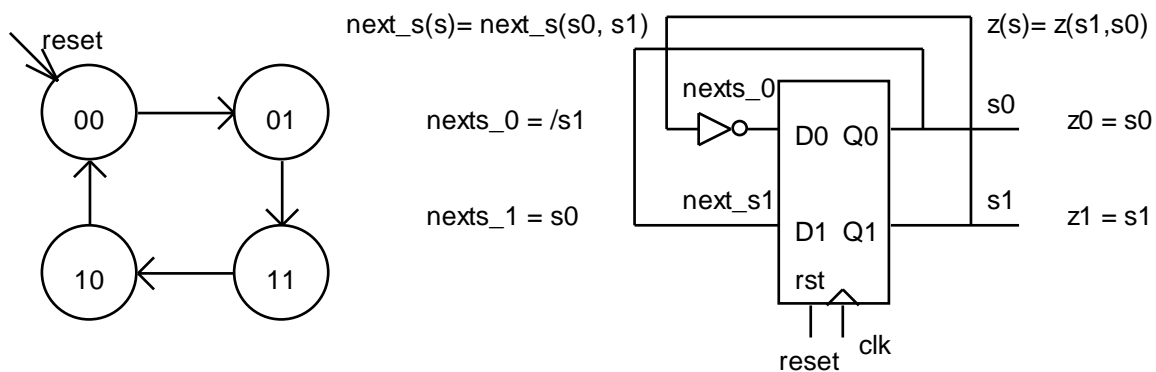


A következő állapotot meghatározó függvény:

$$next_s_0 = /s_1, \quad next_s_1 = s_0$$

A kimeneti függvény:

$$z_0 = s_0, \quad z_1 = s_1$$



Állapottábla

Az állapottábla ugyanazokat az információkat tartalmazza, mint az állapotgráf, csak táblázatos formában.

A fenti állapotgráf állapotváltásai a $s_1s_0 = 00$ kezdő állapotból indulva állapottáblával megadva:

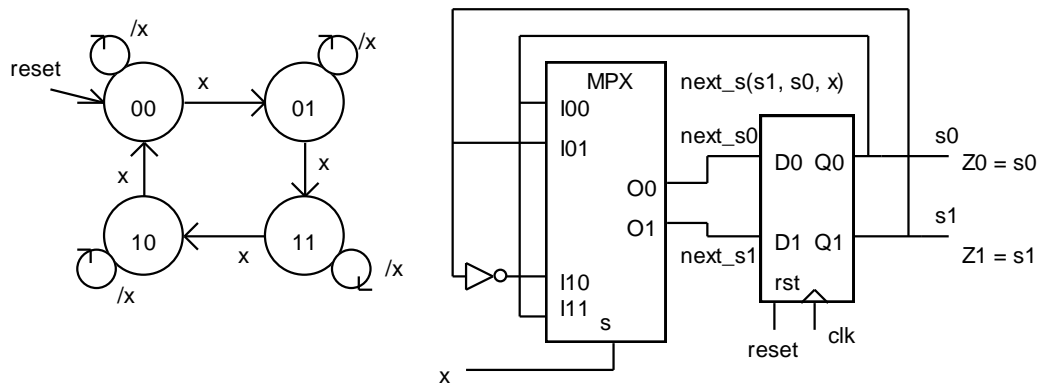
Aktuális állapot	következő állapot
s_1s_0	$next_s_1\ next_s_0$
00	01
01	11
11	10
10	00

Ez tulajdonképpen egy speciális (Gray) kódolású számláló, mely az órajel hatására az alábbi kimeneti sorozatot ismételteti:

00, 01, 11, 10,....

Egészítsük ki egy x engedélyező bemenettel.

Ne váltson állapotot, ha $x = 0$, a fentiek szerint működjön, ha $x = 1$.



A regiszter elé teszünk egy multiplexert, mely $x = 0$ esetén az állapotregiszter kimenetét választja ki a bemenetére (az órajelre ugyanaz az érték íródik vissza, $s = next_s$, nem változik az állapot), $x=1$ esetén pedig az eredeti függvényeket. A multiplexer belső felépítését nem rajzoltuk ki kapu szinten.

Módosítsuk a kimeneti függvényt (amit eddig közvetlenül a regiszter kimenete állított elő), hogy 4 db kimenetre kapcsolódó LED közül (L0, L1, L2, L3) az első 3 állapotban mindig csak az állapothoz rendelt LED-et kapcsolja be, az utolsó állapotban pedig mindet!

Az kimeneti függvény igazságtáblája:

s1s0	L0	L1	L2	L3
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	1	1	1	1

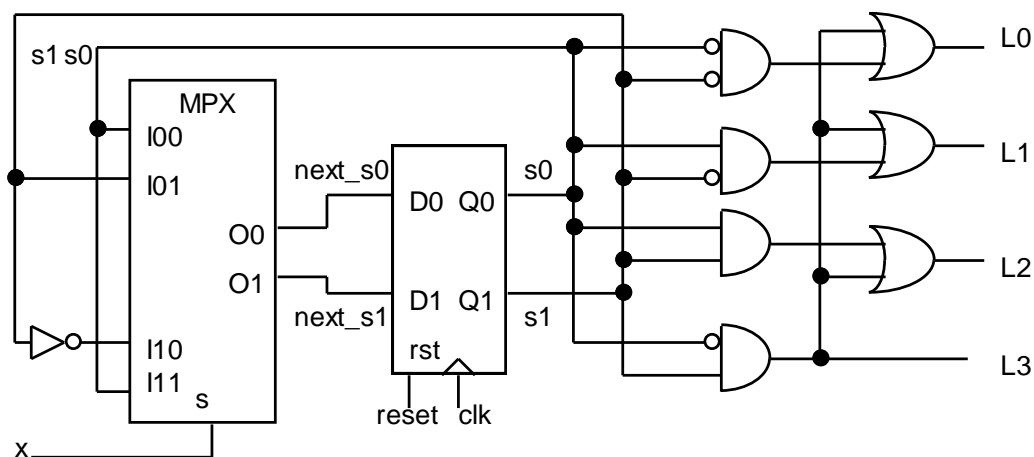
A függvények kiolvashatók az igazságtáblából:

$$L3 = s1/s0$$

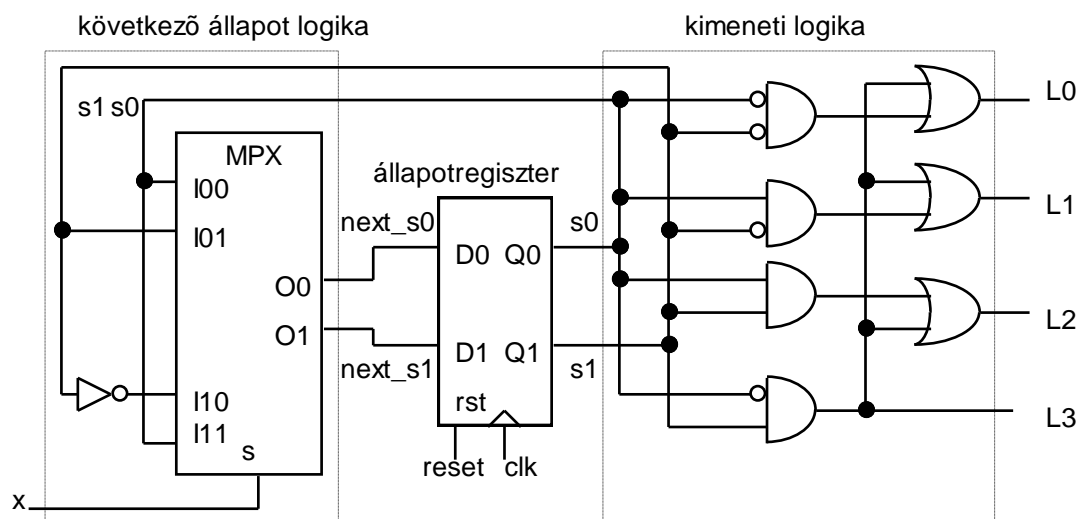
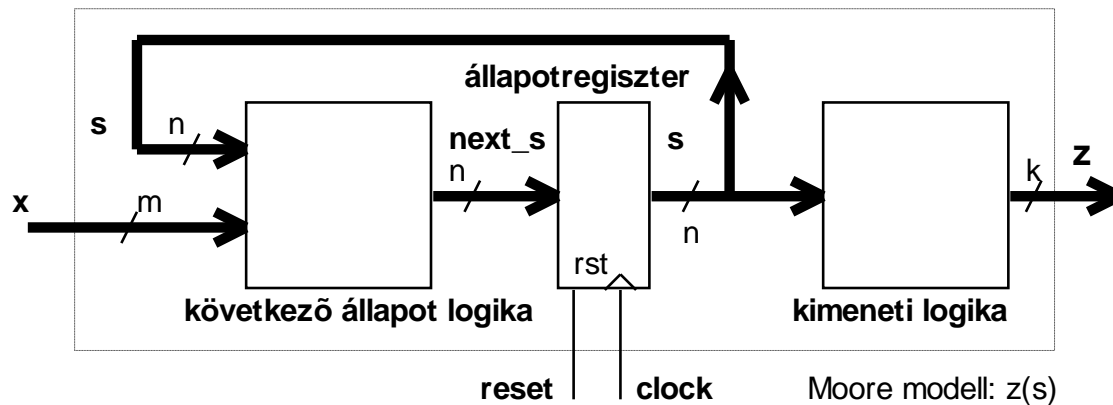
$$L0 = \neg s1/\neg s0 + L3$$

$$L1 = \neg s1s0 + L3$$

$$L2 = s1s0 + L3$$



Az alábbi ábrákon látható hogy mi valósítja meg az általánosított állapotgép egyes részeit (következő állapot logika, állapotregiszter, kimeneti logika)



Sorrendi funkcionális elemek

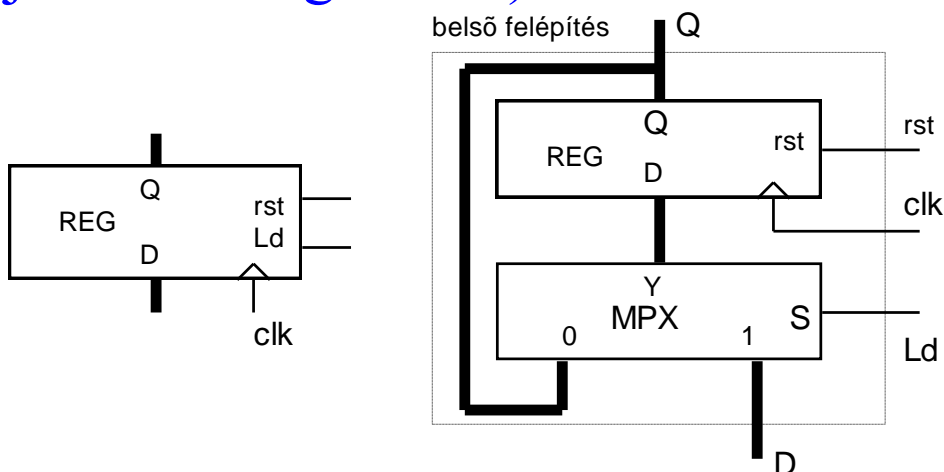
A kombinációshoz hasonlóan sorrendi funkcionális elemek is vannak.

Regiszter(egyszerű)

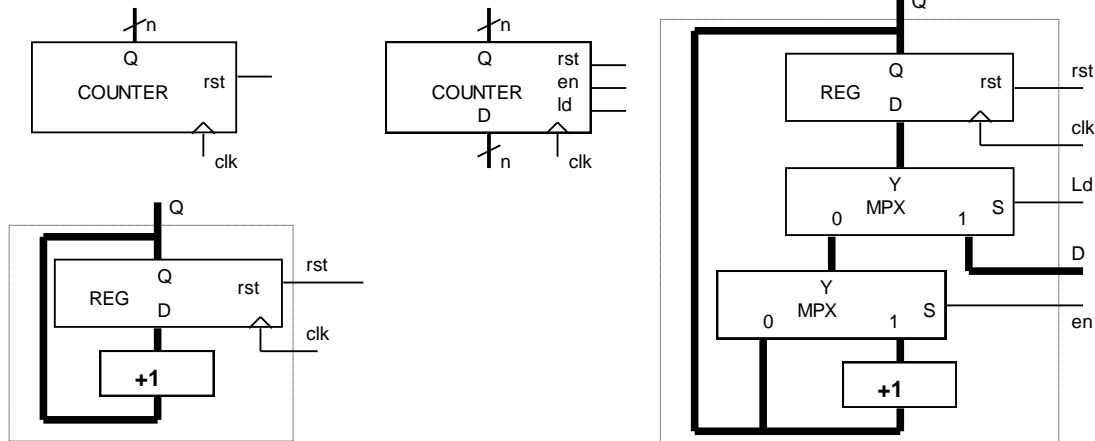
Ezt az előbbieken már megismertünk. Ennek funkciója, hogy minden aktív órajel élnél, eltárolja a bementén levő adatot.

Regiszter beírás engedélyezéssel

Ebbe a regiszterbe csak akkor íródik be a bementére adott érték, ha az Ld betöltés vezérlő bemenetével ezt engedélyezik. (A rst jel törli a regisztert.)



Számláló (counter)



egyszerű számláló belső felépítése
nincs engedélyező bemenete
minden órajelre számol

engedélyezhető (en), tölthető (Ld) számláló belső felépítése

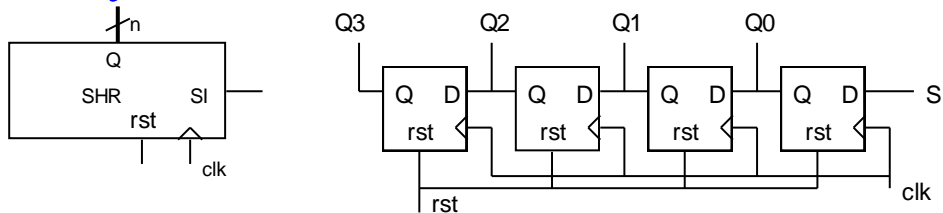
A felfele számláló minden órajelre növeli 1-el a kimenete értékét, ha engedélyezett ($en=1$). Ha elérte a végértéket (bináris számlálónál az $111\dots1$), akkor „átfordul” ($000\dots0$) és újramezdi. Pl. 2 bites bináris számláló egymást követő állapotai: **00**,01,10,11,**00**...

Lehet tölthető. Betölti a D bemenetére adott értéket, ha $ld = 1$ az órajel aktív élére.

Modulusnak nevezik a számláló állapotainak számát. Sokszor bináris számlálókat használnak. Ezek modulusa 2^n , n bites számláló esetén. A számlálónak sok változata van, nem részletezzük.

Shiftregiszter

A shiftregiszter funkciója, hogy elshiftelje (eltolja) a benne lévő adatbitekét valamely irányba.



A legegyszerűbb shiftregiszter csak sorba kötött, közös órajelű D flip-flopokat tartalmaz.

Pl. 4 bites balra shiftelő shiftregiszter állapotai, ha a kezdőértéke 0000 és a belépő bit (SI) az első órajelnél 1, utána pedig 0:

0001, 0010, 0100, 1000, 0000

Bonyolultabb változatait nem részletezzük.

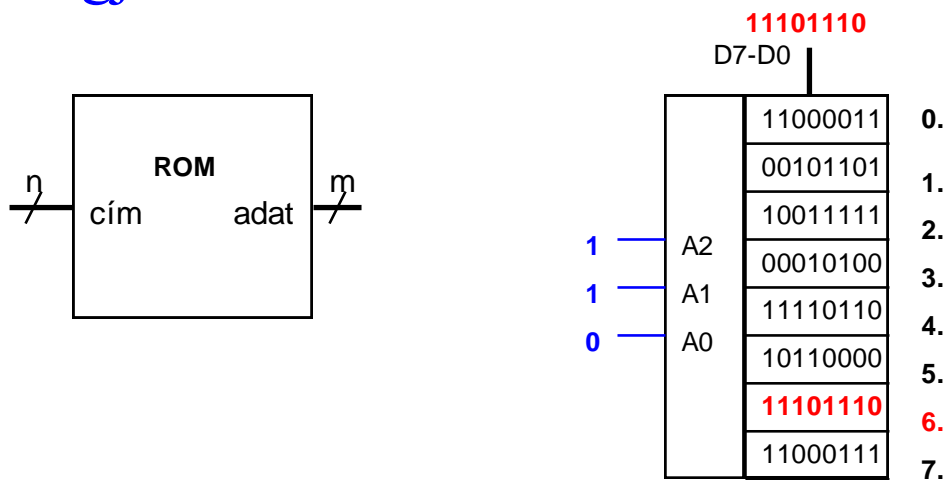
Memóriák

A memóriákat adatok tárolására használjuk. Írhatóság szempontjából két fajtát különböztetünk meg, ROM és RAM.

ROM

A ROM csak olvasható memória. *Az adatot megtartja a kikapcsolás után is.*

A cím bemeneteivel (A[2-0]) lehet kiválasztani a kimenetén (D[7-0]) megjelenítendő adatot tartalmazó rekeszt.

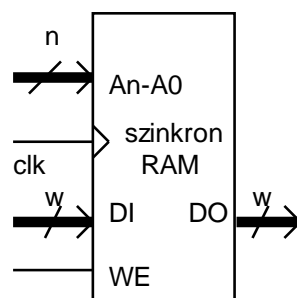


Bizonyos fajta ROM-ok speciális módon írhatók. Pl. a mikrokontrollerek kód memóriája, amelyet FLASH memóriával valósítanak meg.

RAM

A RAM írható/olvasható memória, ezért adat bemenete (DI) és írás vezérlő bemenete (WE) is van. Bizonyos típusnál az adat be és kimenet ugyanaz.

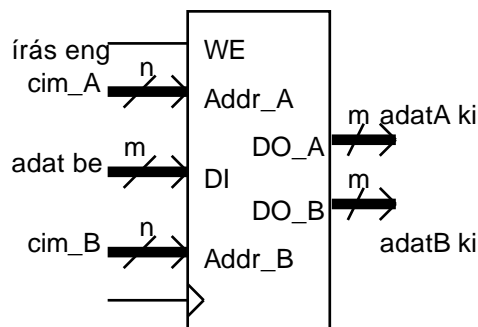
Szinkron RAM szétválasztott adat be (DI) és kimenettel (DO)



Az olvasás a ROM-éhoz hasonló. Az n bites cím bemenettel címezhető (An-A0)

Az írás a szinkron RAM-ok esetén az írás engedélyező jel (***WE***) alatti órajel (***clk***) felfutó élre történik meg.

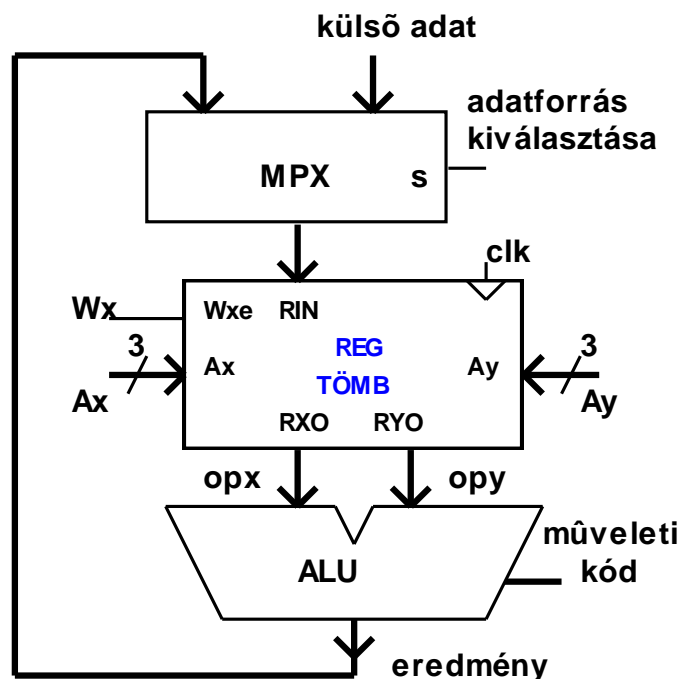
Szinkron dual port RAM



A dual port RAM *két n bites cím bemenettel* (Addr_A, Addr_B) *egy m bites adat bemenettel* (DI) *és két m bites adat kimenettel* (DO_A, DO_B) rendelkezik. Olvasáskor egyszerre megcímezhető két rekesze (Addr_A, Addr_B címekkel). Az íráshoz csak az egyik cím használható (Addr_A). Beírja a DI bemeneten levő értéket, ha WE = 1 és megjön az órajel felfutó éle.

Általános számítási műveletek elvégzésére alkalmas logika

Az eddigi funkcionális elemek ismeretében már megérthetjük a számítógépekben megvalósított *általános számítási műveletek elvégzésére képes logika* funkcionális blokkvázlatát. A regiszter tömböt pl. szinkron dual port RAM valósíthatja meg.



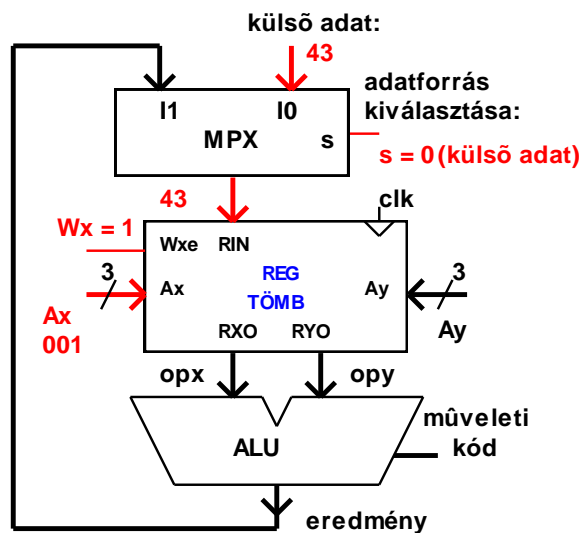
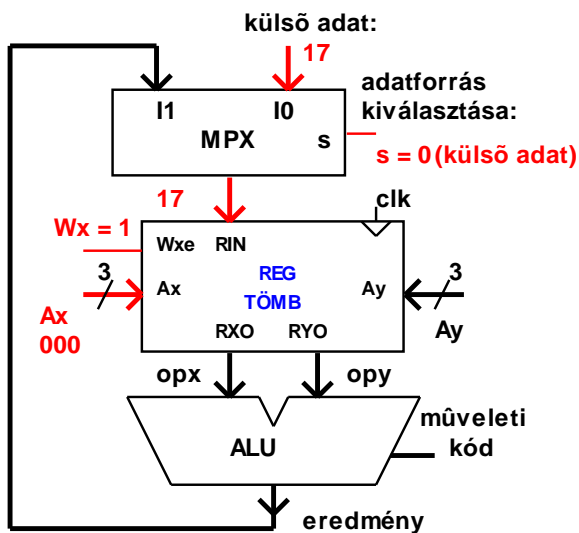
Mi történik az alábbi processzor utasítások végrehajtásakor?

PI: **MOV R0, #17** ;R0 = 17 (0x11)
MOV R1, #43 ;R1 = 43 (0x2B)
ADD R0, R1 ;R0=R1+R0 (R1=60 (0x3C) lesz a művelet után)

- A külső adatokat először adatmozgató utasításokkal be kell írni a regisztertömb megfelelő rekeszeibe (Ax, Ay: regiszter címek Wx: Rx írás engedélyezés), mert az ALU csak regiszterek között képes műveletet végezni. (opx, opy az Ax ill. Ay címen található regiszterek értéke.)

MOV R0, #17 ;R0 = 17

MOV R1, #43 ;R1 = 43



- Az ALU művelethez operandusokat ki kell választani (Ax, Ay), a műveleti kódot be kell állítani és az eredmény Ax címre való visszairását engedélyezni kell (Wx).

ADD R0, R1 ;R0=R1+R0 ;R0=60 (0x3C) lesz a művelet után

