Logikai áramkörök, Boole algebra

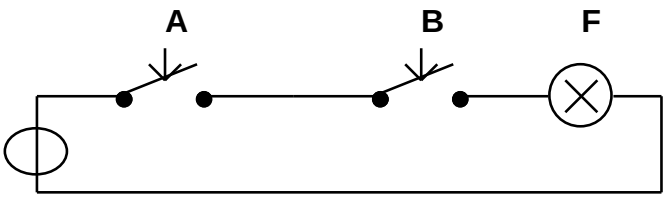
A digitális technikán alapuló eszközök már régóta a környezetünk részei. A mérnökök mindennapi munkaeszközei a számítógépek és a digitális műszerek. Ezért működésük alapelveit azoknak sem árt ismerni, akik csak használják, de nem tervezik ezeket.

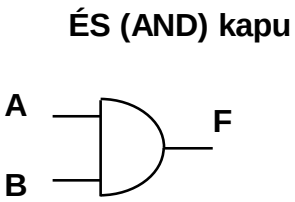
Logikai érték, műveletek a Boole algebrában

A algebrákban egy halmazon értelmeznek műveleteket. A Boole algebrában a halmaz a logikai konstansok (0, 1) és a logikai változók (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
Az igaz értéket 1-el, a hamisat 0-val jelöljük.





Az ***F*** lámpa akkor világít, ha ***A ÉS B*** kapcsoló le van nyomva.

- Ez a ***logikai ÉS (AND) művelet***.
- Leírása a ***Boole algebrában***: **F = A*B**
- Az alapműveleteket egy digitális áramkörben (többek között) ***logikai kapukkal*** lehet megvalósítani.
- A valóságos logikai kapu be és kimenetein az igaz (1) értéknek egy magasabb feszültség, a hamisnak (0) egy alacsonyabb feszültséget feleltetnek meg.
- Ez tulajdonképpen 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 meg lehet adni. 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*….= 1, ha A,B,C.. = 1

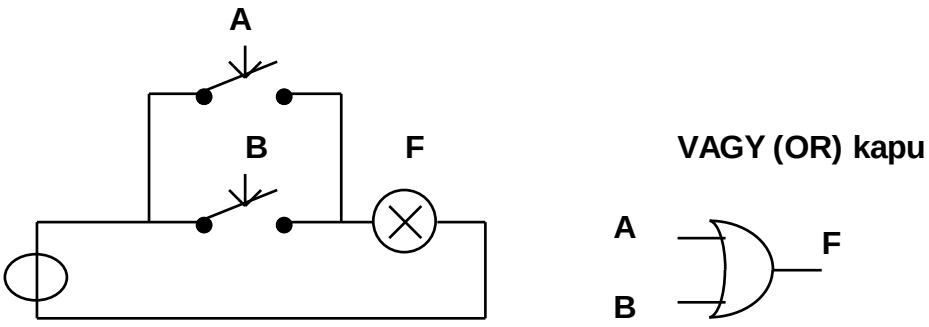
- ***A szorzás műveleti jelét többnyire nem szokták kiírni***: F = A*B*C = ABC

(Tetszőleges sok sorbakapcsolt kapcsoló és lámpa esetén a lámpa csak akkor világít, ha az összes kapcsoló be van kapcsolva.)

További tulajdonságok: **A*0 = 0 A*1=A**

A VAGY (OR) művelet

Az alábbi ábrán párhuzamosan kötött kapcsolók szerepelnek.



Az ***F*** lámpa akkor világít, ha ***A VAGY B*** kapcsoló le van nyomva.

- Ez a ***logikai VAGY (OR) 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**

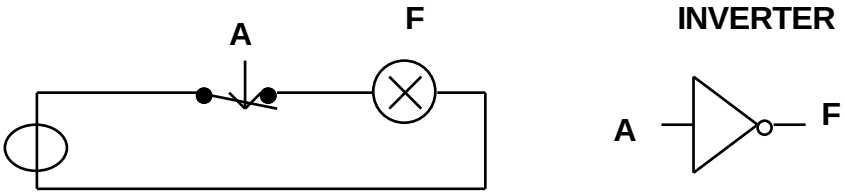
Tetszőleges számú változó OR kapcsolata akkor 1, ha bármelyik 1. (Akkor 0, ha mind 0.)
F = A+B+C+ ….= 0, ha A,B,C.. = 0

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

További tulajdonságok:
A + 0 = A (Ha a kapcsolóval egy szakadást kötünk párhuzamosan, az nem befolyásolja a kapcsoló működését.)
A + 1 = 1 (Ha a kapcsolóval rövidzárat kötünk párhuzamosan, a lámpa mindig világít.)

Az invertálás

Az alábbi kapcsoló kikapcsol, ha lenyomják.



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

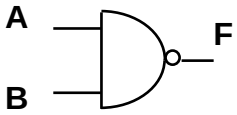
- Ez az invertálás (negálás) művelet. (Mert a kapcsoló lenyomott állapotához rendeltük az ***A*** változó igaz értékét.)
- Leírása a Boole algebrában: $F = \overline{A}$
  Ezt billentyűzeten egyszerűbb így írni:
  F = /A
- A negálás igazságtáblája:

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

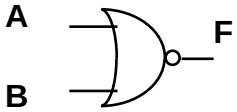
Az előbbi kapuknak a negált változata is létezik, ezek igazságtáblájában a kimenet az eredeti negáltja. A negálást itt egy kis kör jelöli (ezt máshol is szokás így jelölni):

NAND (NEM ÉS) ***kapu***



F = /(A*B)

NOR (NEM VAGY) ***kapu***

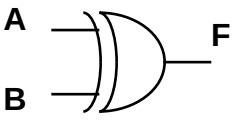


F = /(A+B)

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

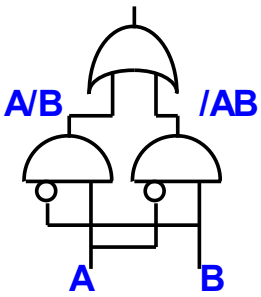
A	B	**F**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Rajzjele:



Felépítése kapukkal:

F = A XOR B = A/B+/AB



Néhány további Boole algebrai azonosság:

Kommutativitás (felcserélhetőség):

A + B = B + A, A*B = B*A

Asszociativitás (csoportosíthatóság):

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

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

Disztributivitás (felosztható ság):

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

A + (B*C) = (A + B)(A + C) szokatlan!

Elnyelés:

A + A*B = A A*(A + B) = A

De' Morgan azonosságok:

/A + /B = /(A*B), /A*/B = /(A + B)

A fentiek beláthatók a jobb és baloldal igazságtábláinak felírásával.

Dualitás elve:

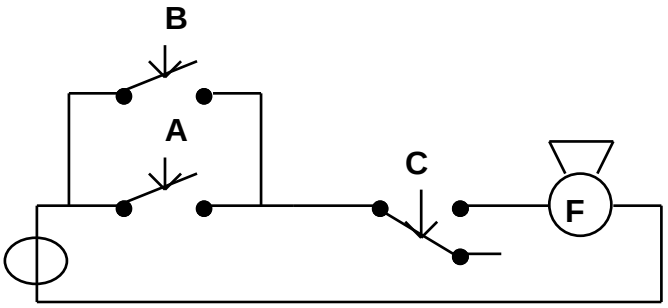
A + ↔ *, és 0 ↔ 1 felcserélésével is igazak maradnak a Boole algebrai azonosságok.

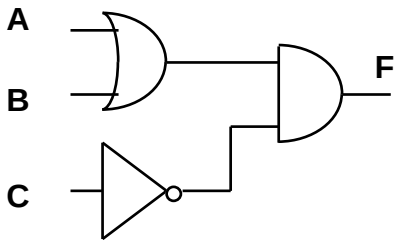
Pl. Lásd De' Morgan, és alább

A*1 =A, A+ 0 = A

Tervezzünk az összes műveletet felhasználó áramkört! A riasztó akkor jelezzen, 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 kapukal:





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

F = (A + B)*/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 logikai függvény felírható az igazságtábla alapján. Itt az ún. ***SOP*** (Sum Of Product, szorzatok összege) kanonikus alakú (egyszerűsítetlen) alakú felírást mutatjuk meg. Az igazságtábla 1-eseit logikai szorzatokkal fejezzük ki. Az egyes szorzatokban az 1 értékű logikai változók ponáltan, a 0 értékűek negáltan szerepelnek, mivel így 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

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

F = /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.

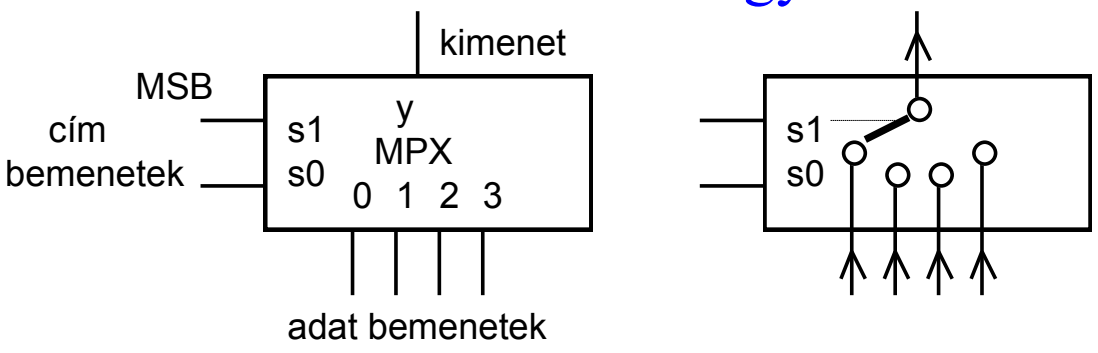
Kombinációs funkcionális elemek

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

Multiplexer

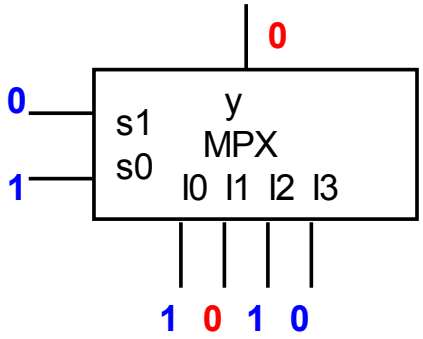
A multiplexer funkciója adat választás.

rajzjele: viselkedését magyarázó ábra:

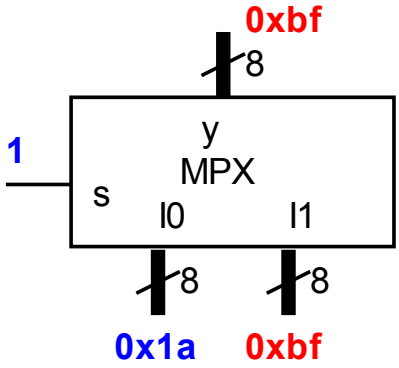


A fenti „a” ábrán egy 4 db 1 bites adat bemenet küzül választani képes ún. 4/1-es multiplexer rajzjele látható. A jobb oldali „b” ábra a működés megértését segíti. A bemenet kiválasztása az s1s0 select jelekkel történik. Az y kimeneten a kiválasztott bemeneten levő

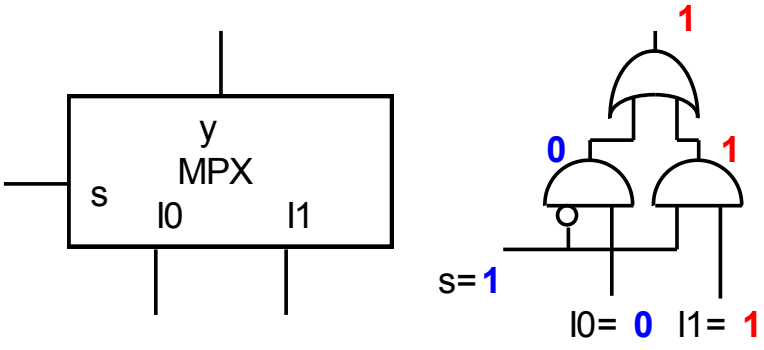
logikai érték jelenik meg. Pl. ha s1s0 = 01, akkor az I1 bemeneten levő érték.



A bemenetek több bitesek is lehetnek.
Pl. 2 db 8 bites bemenetből választani képes multiplexer, ha s = 1, az I1 bemeneti adatot választja ki:

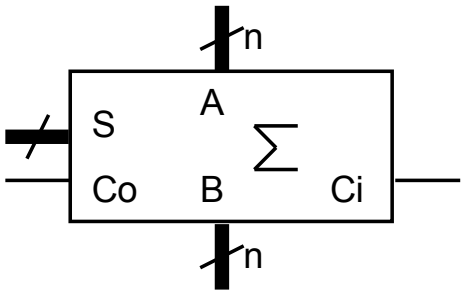


A legegyszerűbb a 2/1-es multiplexer amelynek 1 bitesek a bemenetei. Ennek a belső felépítését is lerajzoltuk.



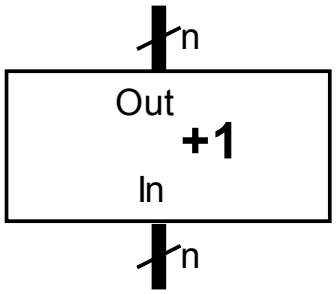
Összeadó

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

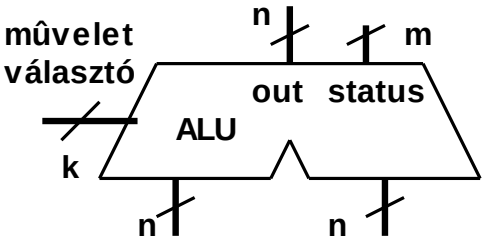


Inkrementer

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

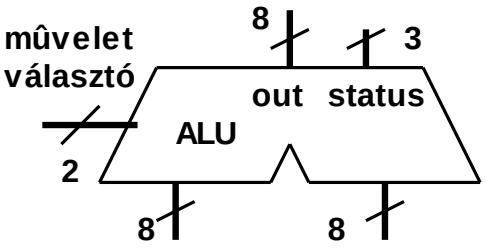


ALU (aritmetikai logikai egység)



Egy viszonylag bonyolult kombinációs funkcionális elem az ALU. Ez ***aritmetikai és logikai műveleteket*** tud végezni a bementére adott adatokon. A művelet eredménye az ***adat kimentén*** (out) jelenik meg. A művelet eredménye alapján bizonyos többlet információkat is ad, ezt nevezik ***státus kimenetnek*** (pl. hogy az összeadás során volt-e átvitel, azt a carry státus bit jelzi). Magát 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: 2-es komplemens összeadás, kivonás, AND, OR (művelet kiválasztó bemenetek: s1,s0)
- státus információ:
  Z (a művelet eredménye 0),
  N (az előjel bit 1 értékű, vagyis negatív),
  C (túlcsordulás volt a művelet során)

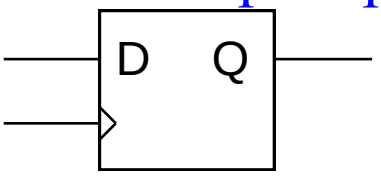
Példaként megmutatjuk, milyen kimenetet ad, az alább megadott műveleti kódok esetén, ha az operandusok:

op1 = 11001011
op2 = 01010101

s1s0	**művelet**	**eredmény**	***Státus:Z,C,N***
00	ADD	00100000	0,1,0
01	XOR	10011110	1,0,1
10	OR	11011111	0,0,1

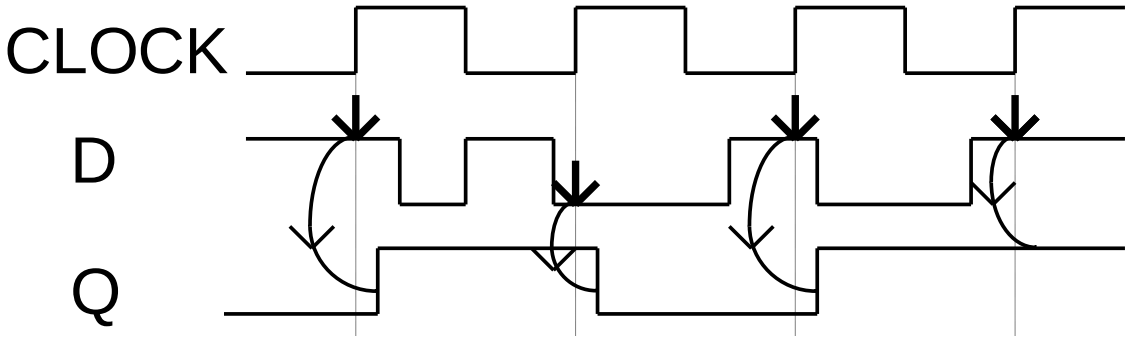
Szinkron sorrendi hálózatok (állapotgépek, synchronous Final State Machines)

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. Ezek működését egy periodikus négyszögjel, az órajel éle ütemezi. Tipikusan minden változás csak az órajel (clock) felfutó élre történhet. Az emlékezetet memória jellegű logikai áramkörök biztosítják. Ezek a flip-flop-ok. Több típusa van itt csak a legyakrabban használt szinkron D flip-flopról beszélünk.

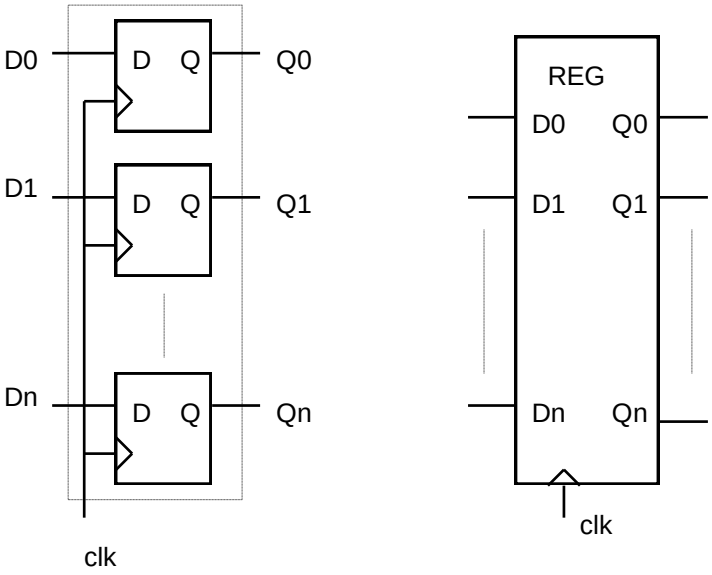


felfutó él érzékeny
D flip-flop

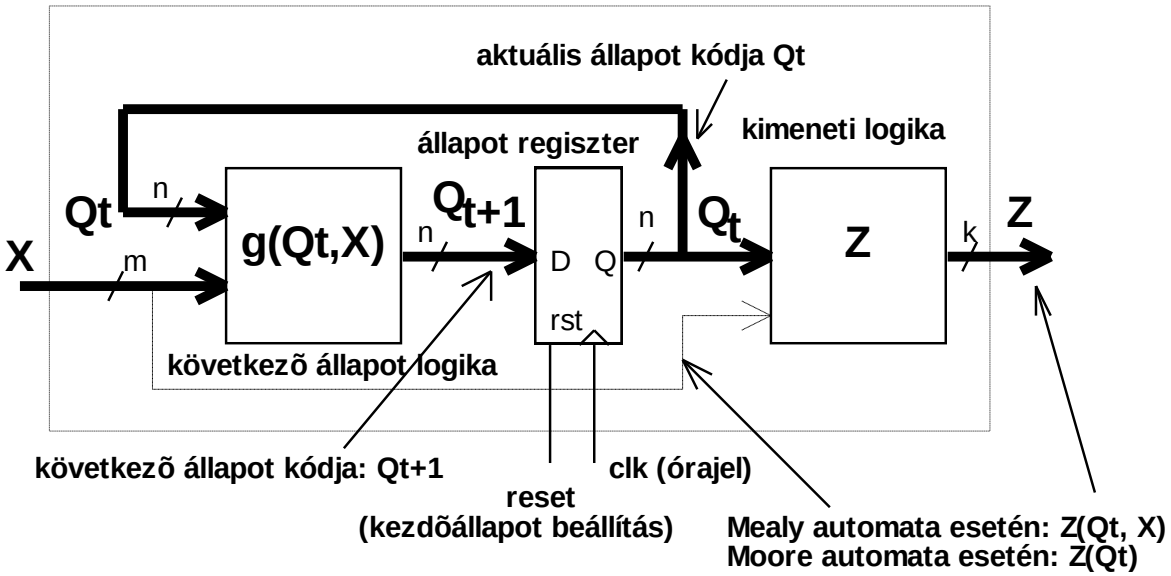
A D bemenet értékét megjegyzi az órajel felfutó élénél és a kimenetén ezt az értéket tartja a következő órajel felfutó élig.



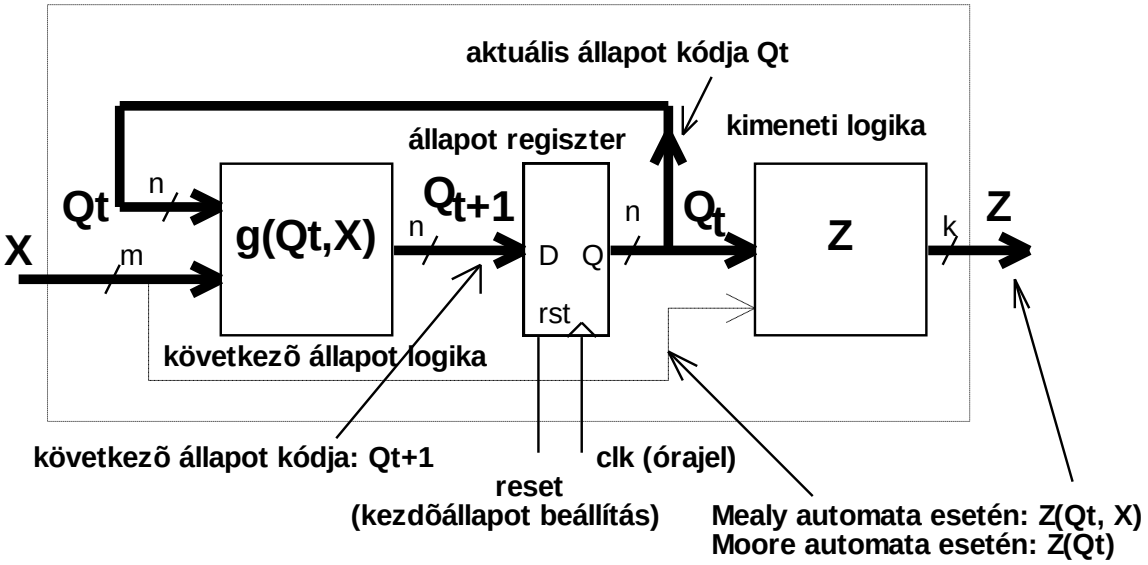
A több közös órajelű D flip-flopból készített tárolót regiszternek nevezzük. Ez tulajdonképpen egy ***sorrendi funkcionális elem***.



A szinkron sorrendi hálózat felépítése



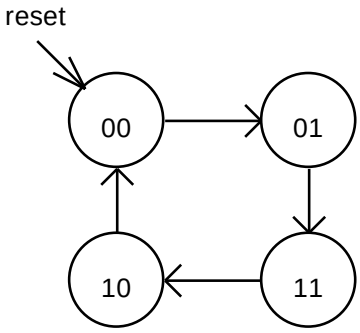
A regiszter tárolja az „emlékezni valót”, ez egy n-bites kód, ezt nevezzük az aktuális állapot kódjának.

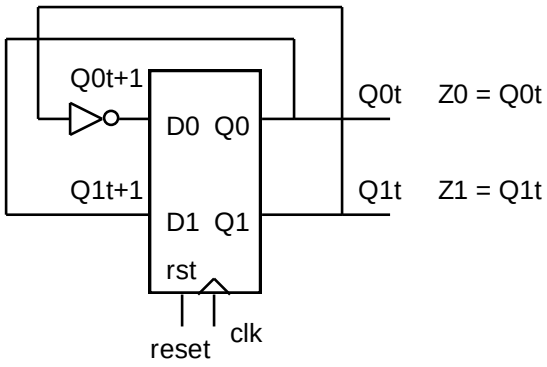


- A g(X,Qt) n kimenetű logikai függvény (n db logikai függvény, melyeknek ugyanazok a változói) az aktuális állapotkód (Qt) és bemenet (X) alapján előállítja a következő állapot kódját (Qt+1).
- Ez a következő órajel felfutó élére eltárolódik az állapotregiszterben. (Ez lesz az új aktuális állapot.)
- 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 állapotkód és esetleg az aktuális bemenet alapján. Mivel a kimenet nem csak az aktuális bemenettől (X), hanem az állapottól (az állapot kódjától) is függ, ugyanazon bemenetre a későbbiekben más lehet a Z kimenet.

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

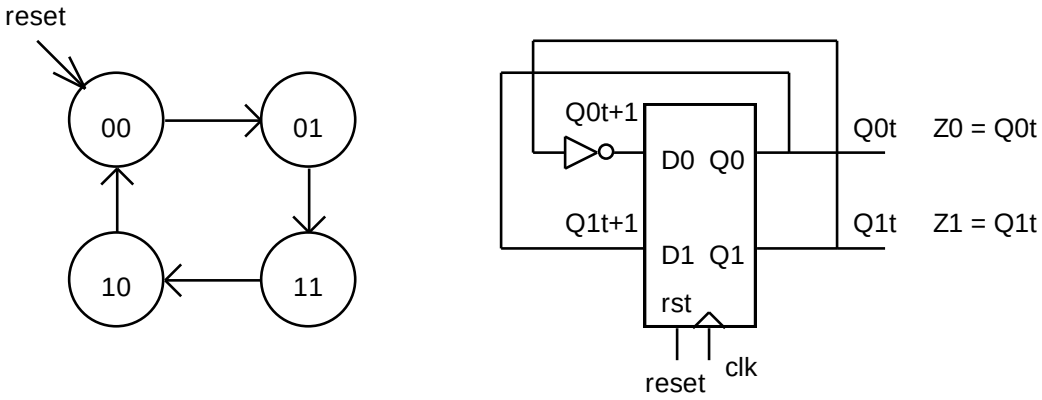
Pl. Egy 4 állapotú olyan hálózat állapot gráfja, melynek nincs bemenete (az órajelet nem tekintjük bemenetnek) és kimenete közvetlenül az állapotregiszter:





A következő állapotot meghatározó függvény:
D0 = Q0t+1= /Q1t, D1 = Q1t+1 = Q0t

A kimeneti függvény:
Z0 = Q0t, Z1 = Q1t

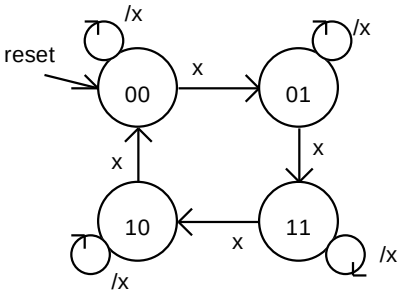


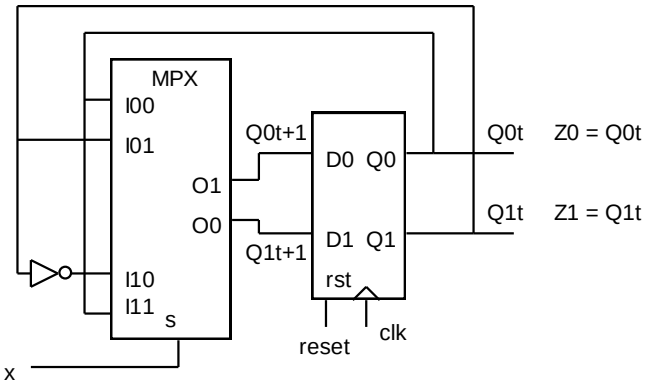
Az állapotváltozások Q1tQ0t = 00 kezdő állapotból indulva:
Aktuális állapot következő állapot

Q1Q0t	Q1Q0t+1
00	01
01	11
11	10
10	00

Ez tulajdonképpen egy speciális (Gray) kódolású számláló, mely az órajel hatására az alábbi kimeneti sorozatot ismételgeti:
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 a regiszter kimenetét választja ki a bemenetére (az órajelre ugyanaz az érték íródik vissza), x=1 esetén pedig az eredeti függvényeket. A multiplexer belső felépítését nem rajzoltuk ki kapu szinten.

Egészítsük ki a logikát, hogy a 4 db LED közül (L0,L1,L2,L3) az első 3 állapotban mindig csak az állapothoz rendeltet kapcsolja be, az utolsó állapotban pedig mindet!

Az igazságtábla:

Q1Q0t	L0	L1	L2	L3
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	1	1	1	1

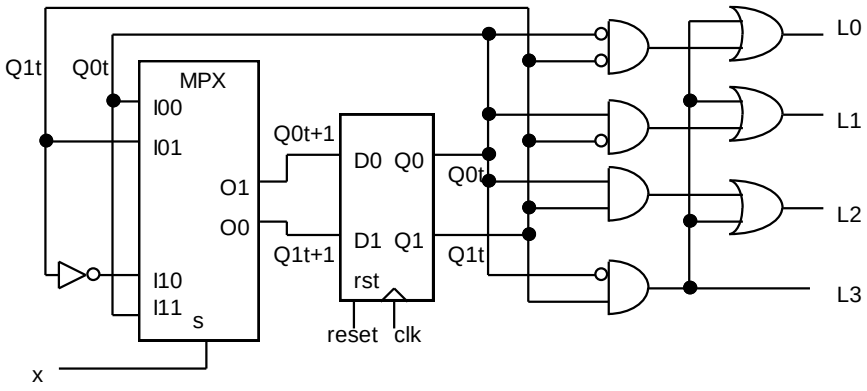
A függvények kiolvashatók az igazságtáblából:

L3 = Q1/Q0

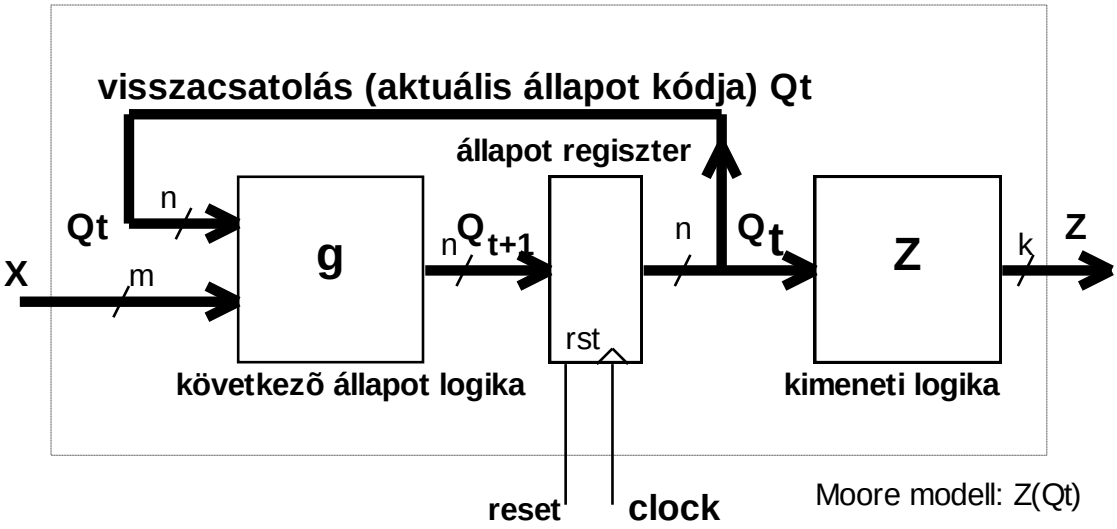
L0 = /Q1/Q0 + L3

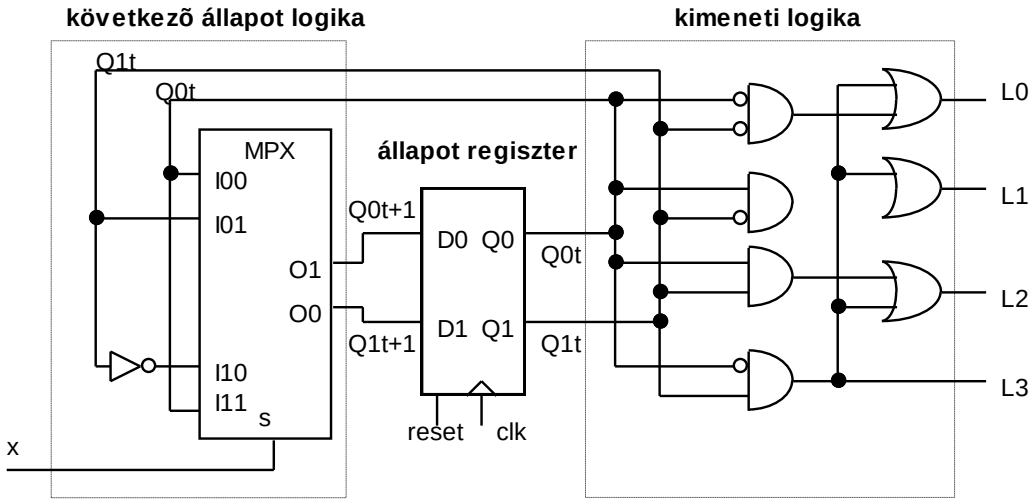
L1 = /Q1Q0 + L3

L2 = Q1Q0 + L3



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





Sorrendi funkcionális elemek

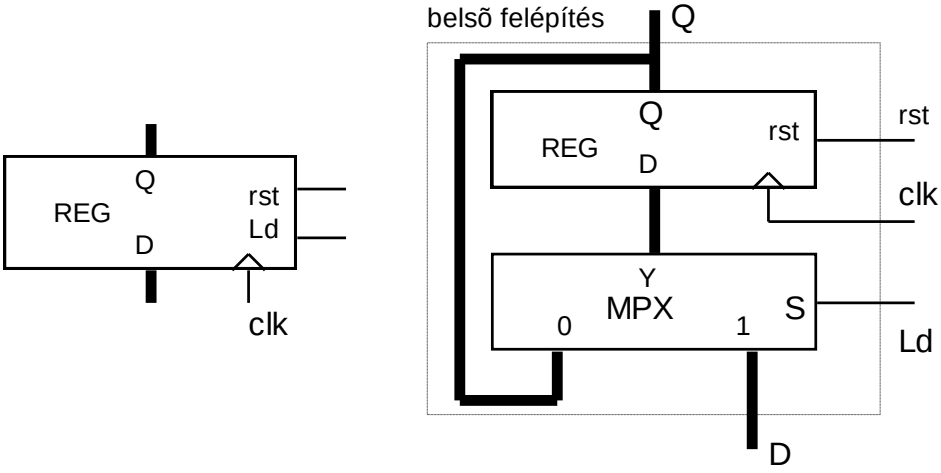
A kombinációshoz hasonlóan sorrendi funkcionális elemek is vannak.

Regiszer

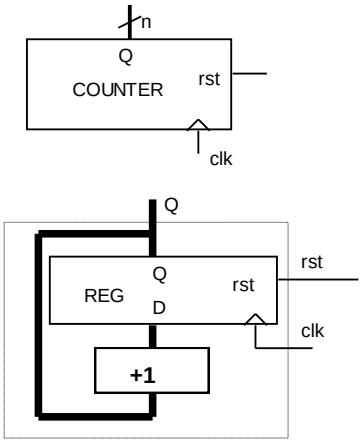
Ezt az előbbiekben már megismertünk. Ennek funkciója, hogy minden aktív órajel élnél, eltárolja a bementén levő adatot.

Regiszter betöltés vezérlő bemenettel (Ld)

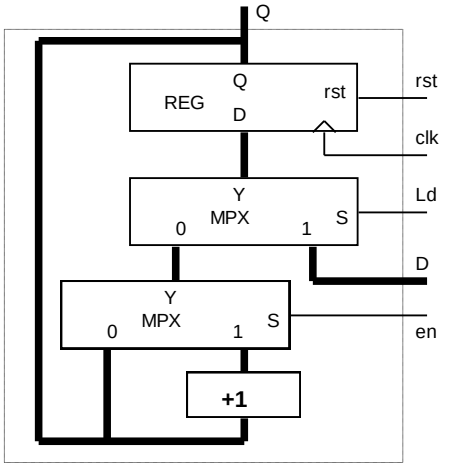
Ebbe a regiszterbe csak akkor íródik be a bementére adott érték, ha az Ld bemenetével ezt engedélyezik. (A rst jel törli a regisztert.)



Számláló (counter)



egyszerû számláló belsõ felépítése
nincs engedélyezõ bemenete
minden órajelre számol

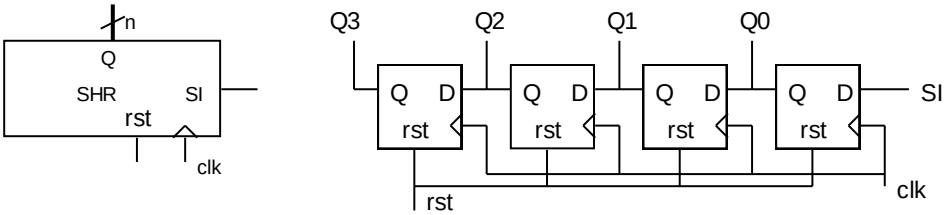


engedélyezhetõ (en), tölthetõ (Ld) számláló belsõ felépítése

A számláló minden órajelre növeli 1-el a kimenete értékét, ha engedélyezett (en=1). Ha elérte a végértéket (bináris számlálónál az 111…1), akkor „átfordul” (000…0) és újrakezdi. 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 adatbiteket valamely irányba.



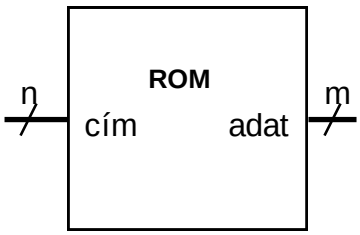
A legegyszerűbb shiftregiszer csak sorba kötött, közös órajelű D flip-flopokat tartalmaz. Pl. 4 bites balra shiftelő shiftregiszter állapotai, ha a kezdőértéke 0000 és a belépő bit (SI) az első órajelnél 1, utána pedig 0:
0001, 0010, 0100, 1000, 0000
Bonyolultabb változatait nem részletezzük.

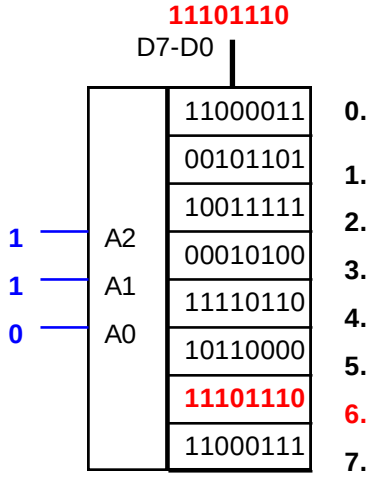
Memóriák

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.



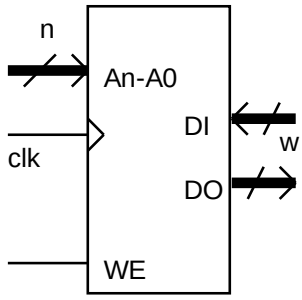


Bizonyos fajta ROM-ok speciális módon írhatók. Pl. a mikrokontrollerek kód memóriája, amelyet FLASH memóriával valósítanak meg.

RAM

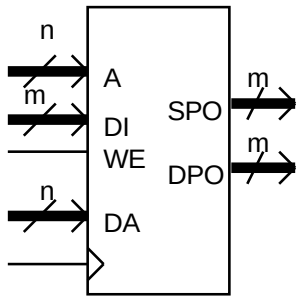
A RAM írható memória, ezért adat bemenete (DI) és írás vezérlő bemenete (WE) is van. Bizonyos típusnál az adat be és kimenet ugyanaz.

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



Az olvasás a ROM-éhoz hasonló. Az n bites cím bemenettel címezhető (A[n-0])
Az írás a szinkron RAM-ok esetén az írás engedélyező jel (***WE***) alatti órajel (***clk***) felfutó élre történik meg.

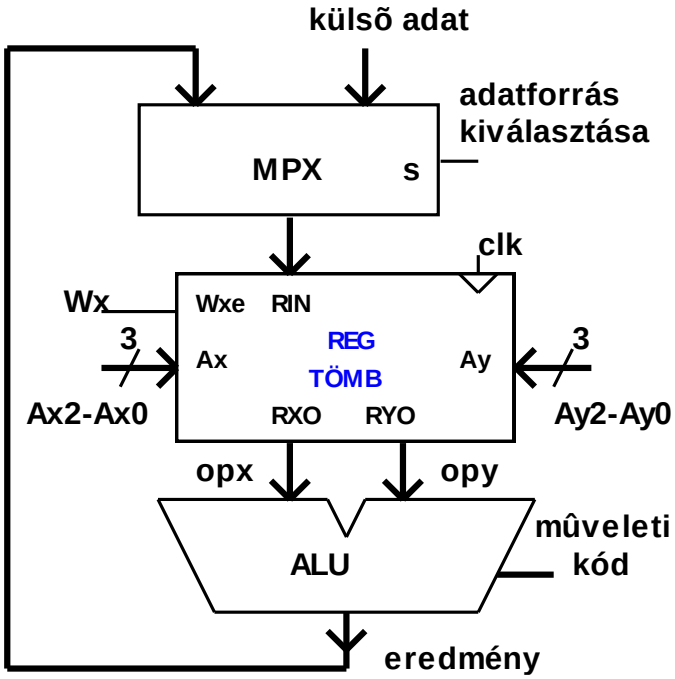
Szinkron dual port RAM (szétválasztott adat be és kimenettel)



A dual port RAM két n bites cím bemenettel (A, DA) egy m bites adat bemenettel (DI) és két m bites adat kimenettel (SPO, DPO) rendelkezik. Olvasáskor egyszerre megcímezhető két rekesze (A és DA n bites cím bemenetekkel). Az íráshoz csak az egyik cím használható (A).

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.

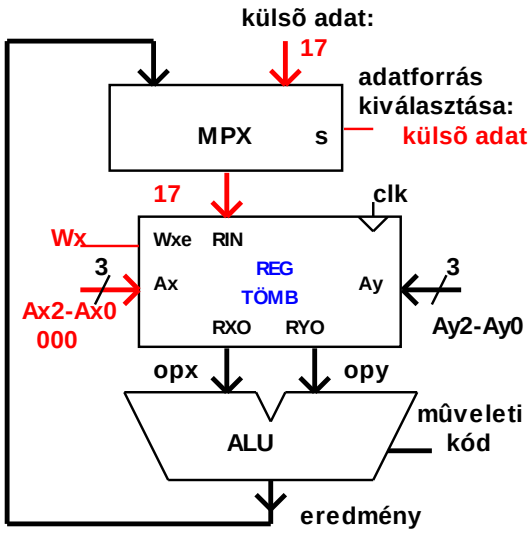


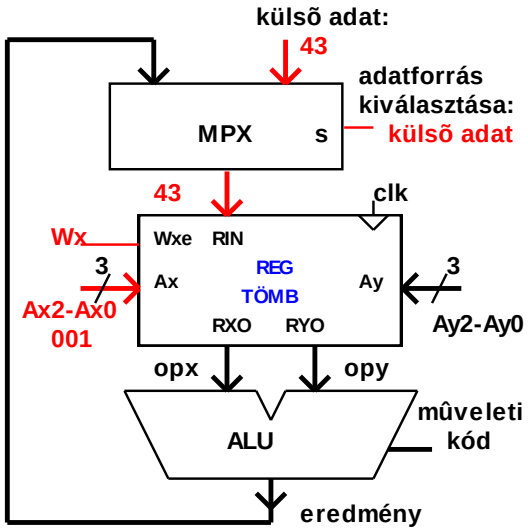
Mi történik az alábbi processzor utasítások végrehajtásakor?

Pl: **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 visszaírását engedélyezni kell (Wx).**

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