2018-as digsor 2H (A)

1. RISC processzor jellemzői:

- * Memória elérés csak load és store műveletek segítségével
- * Egyszerűsített címzési módok
- * Minden utasítás ugyanolyan hosszúságú
- * Az utasításokat lehetőleg egy órajel ciklus alatt hajtsa végre
- * Nagyszámú általános célú regiszter

RISC (Reduced Instruction Set Computer)
↳ Csökkentett utasításkészletű számítógép

pl.: * Motorola 88000 RISC rendszer
* Berkeley Egyetem RISC I. rendszere

jellemző (másik forrás):

* Csak a kívánt alkalmazásra jellemző utasítástípusokat tartalmazza. Ezáltal nő a sebesség.
* Minimális utasításkészletet és címzési módot, gyors hardver- és szoftverelemeket használ.
* A komplex függvények elvégzéséhez szubrutinokra és hosszabb utasítássorozatokra van szükség.
* Azonos hosszúságú utasításformátummal a végrehajtandó utasítás dekódolása minimális idejű lesz.
* Hullámlapolt utasításdekódolás
* Egyszerű ciklusvégrehajtás
* LOAD/STORE memóriaszervezés: 2 művelet - tölt és tárol (regiszter ↔ memória)

2. 15 bites 2-es komplemens fix pontos rendszer. $p = 7$

differencia -> Δr

N -> bites szám

$\Delta r = 2^{-p}$

↳ fix pontos rendszer ezzel tér a finom-ság

Megoldás:

előjel bit

↑ 0 -> +

1 -> −

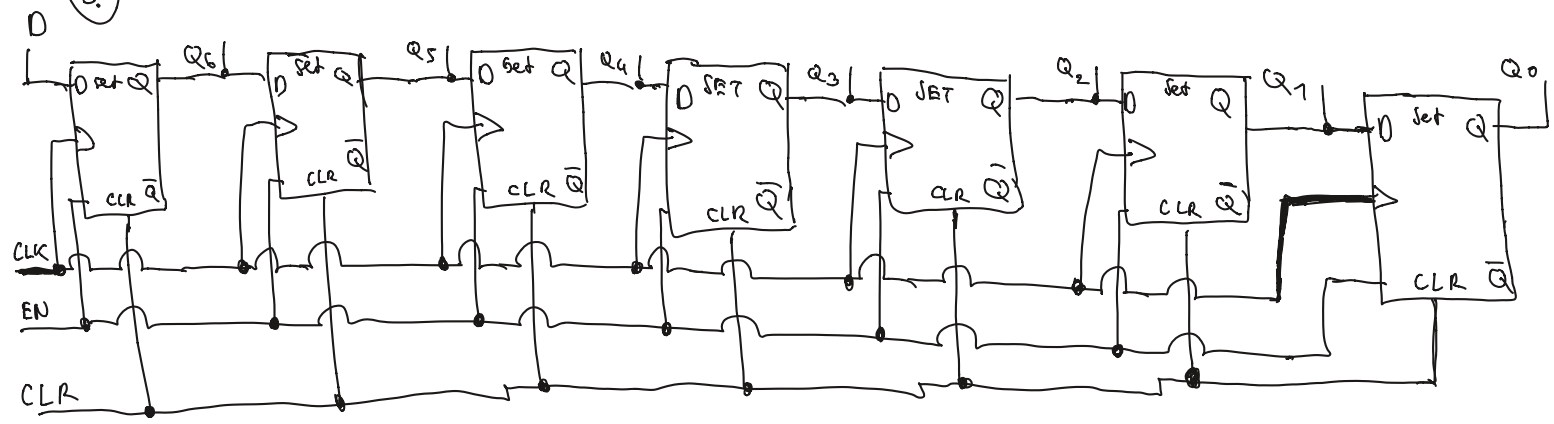
V(smallest absolute) = $0\,0000000.0000001 = 2^{-7}$

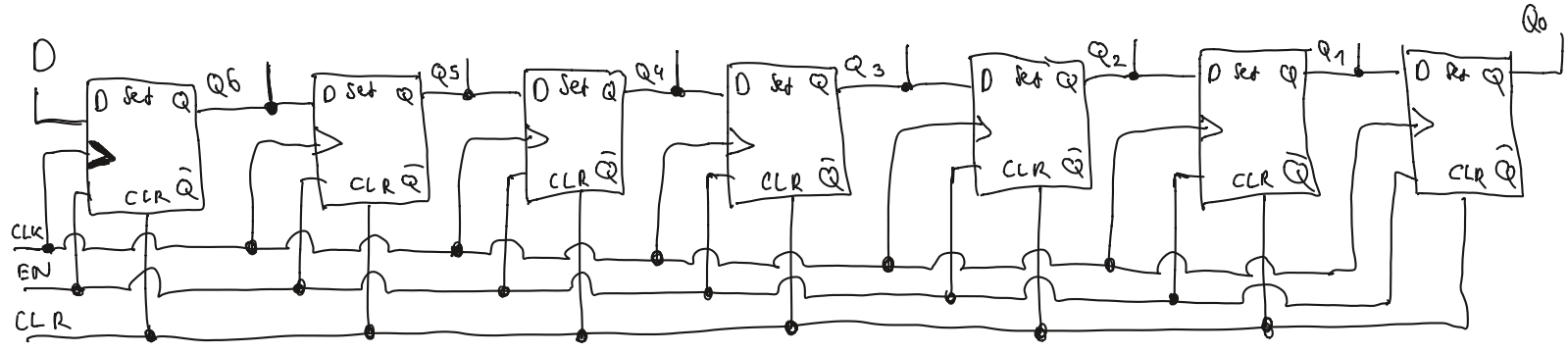
V(largest absolute) = $0\,1111111.1111111 = \frac{16383}{128} = 127,9921875 \approx 128$

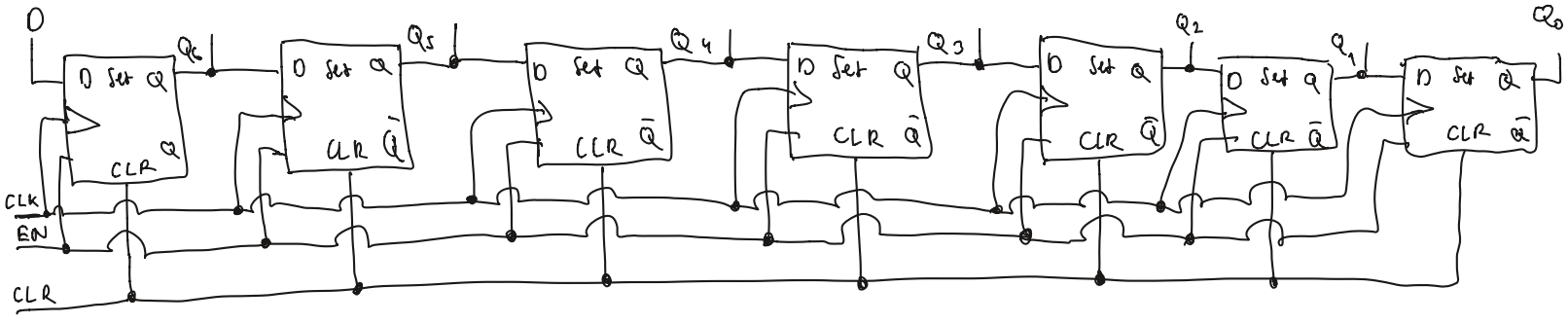
V(largest negative) = $1\,0000000.0000000 \approx -128 = -2^{N-p-1} = -2^{7}$

Differencia = $\Delta r = 2^{-7} = 0,0078125$

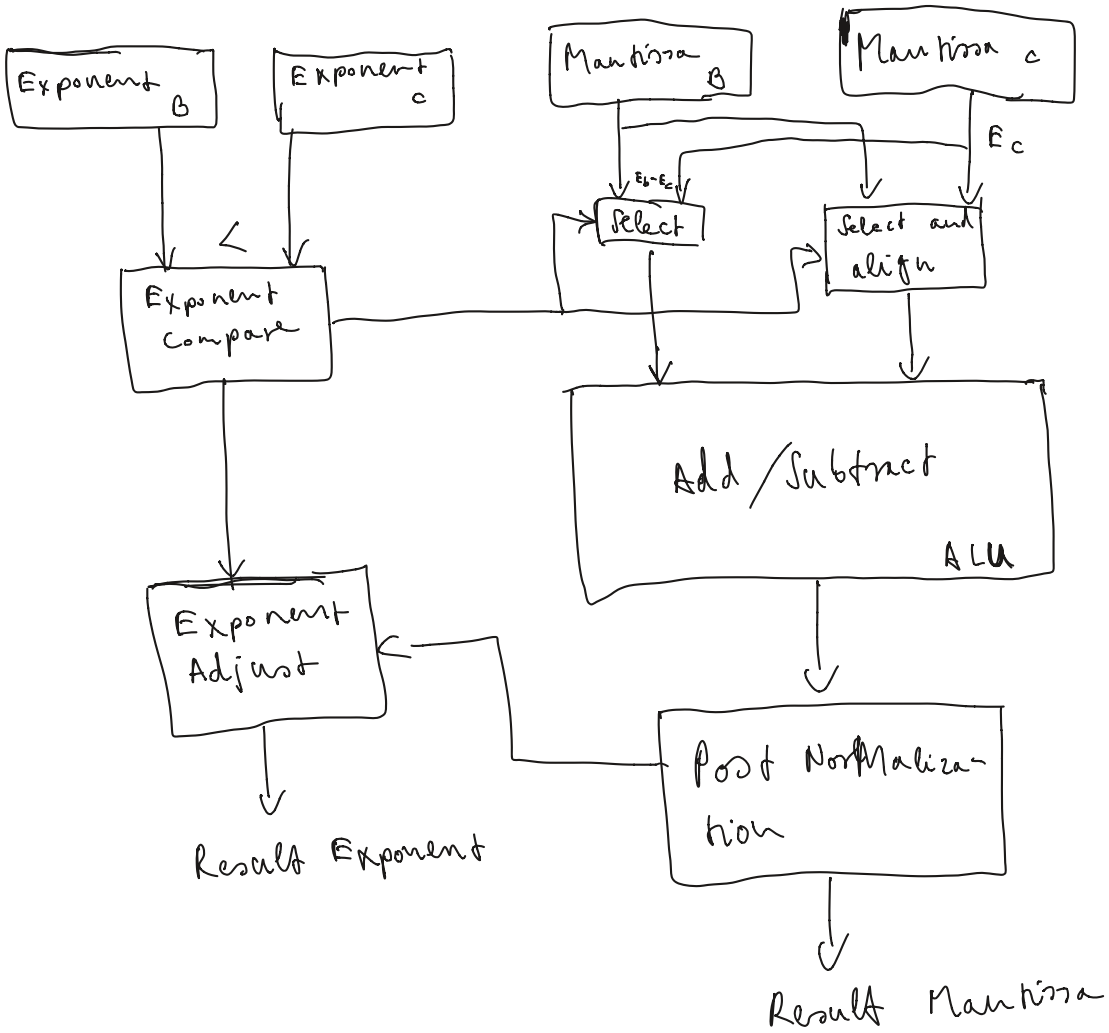
3. 7 bites, sorosan, siftregiszter, D tárolós







(4.) Adja meg a lebegőpontos kivonó blokkdiagramját, az egyes blokkok funkciójával együtt!



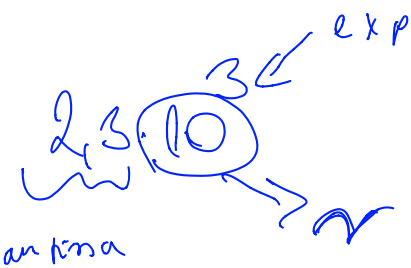
Művelet:

$$A = B - C = M_B \times r^{E_B} - M_C \times r^{E_C} = \left(M_B \times r^{E_B - E_C} - M_C\right) \times r^{E_C}$$

M_B és M_C -> mantissza

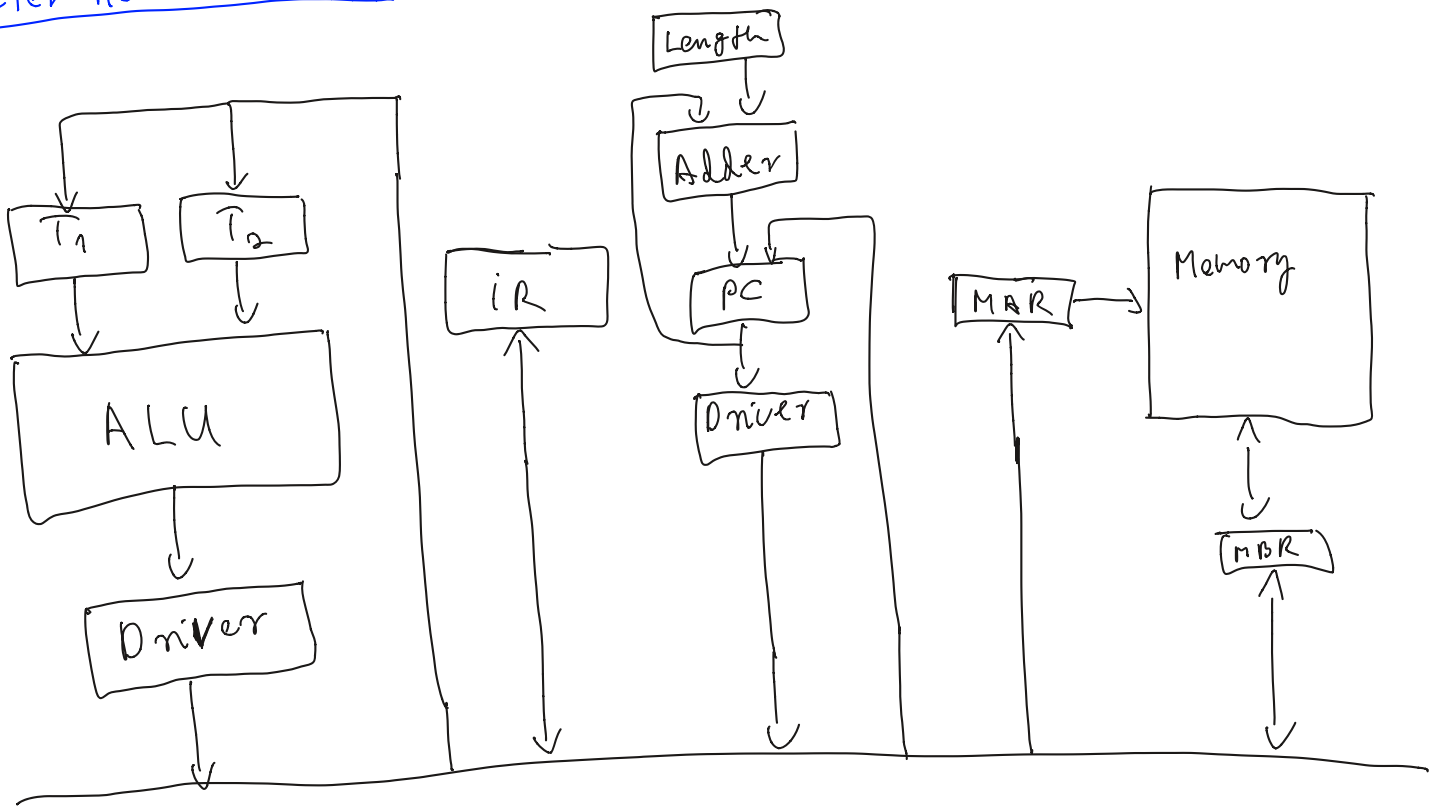
r a számrendszer alapja

E_B és E_C -> exponens



(5.) Rajzolja fel a 2 című számítógép blokkdiagramját! Def. az egyes blokkok funkcióját is!

Regiszter nélküli eset:



T_1, T_2 : regisztrek, adattárolók.

ALU: Arithmetic Logic Unit. olyan adatkezelő egység, amely matematikai/logikai műveletek végrehajtását végzi
Aritmetikai Logikai Egység

Driver: Driver. Egy adott eszköz elérését, funkcióinak használatát lehetővé tevő szoftver. Meghajtó program.

IR: Instruction register (Utasítás Regiszter). Tárolja az éppen végrehajtás alatt álló utasítást. Engedélyezi a gép vezérlő részeinek, hogy a regisztrek, memóriák, aritmetikai egységek vezérlő vonalait a végrehajtáshoz szükséges működési módba állítsák.

PC: Program Counter (Programszámláló). A soron következő (végrehajtandó) utasítás helyét azonosítja.

MAR: Memory Address Register (Memória-cím Regiszter).

A memóriából érkező vagy a memóriában lévő információ helyét azonosítja adott memóriacím alapján.

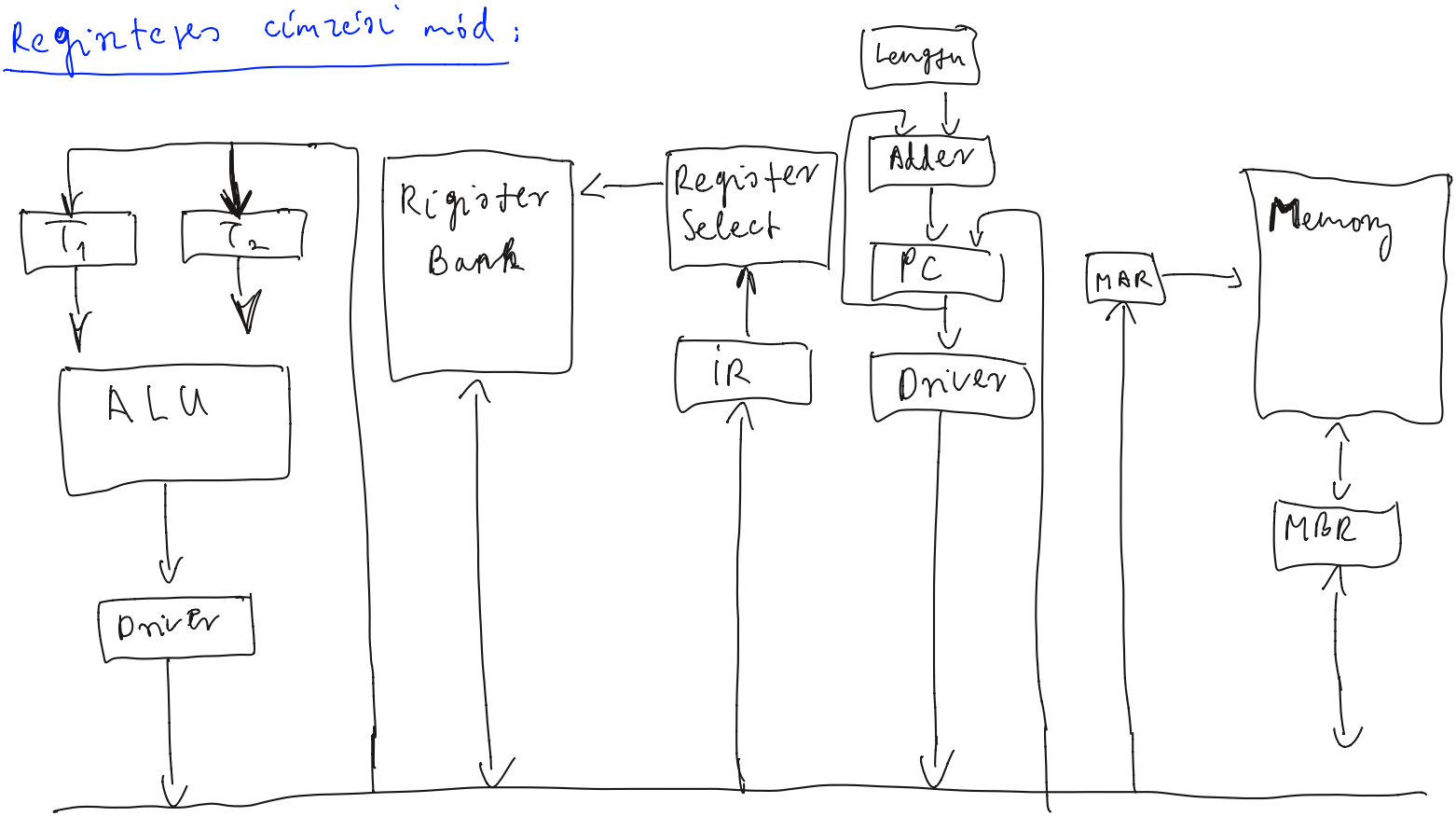
MBR: Memory Buffer Register (Memória Puffer Regiszter).
Tárolja a memóriába beírt ill. az onnan érkező információt.

Memory: Memória

Length:

Adder:

Regiszteres címzési mód:



Register Bank: Regisztereket tárol.

Register Select:

6. Adja meg egy 16·16 bites iteratív szorzóalgoritmus folyamatábráját, és annak áramköri megvalósítását, továbbá az algoritmus végrehajtási idejét!

Pl.: Két 8-bites szám Shift & Add módszerű szorzására. ??? -> Ez vajon kell e

Időszükséglet:

$$T_{Mult} = T_{setup} + N \times T_{Iter} \quad \text{ahol} \quad T_{Iter} = T_{AND} + T_{Sum} + T_{Reg}$$

T_{setup}: Kezdeti ellenőrzések, inicializálás ill. szorzat regiszter törlése

T_{AND}: AND függvények végrehajtása, parciális szorzatok képzése

T_{Sum}: Parciális szorzatok összeadása

T_{Reg}: Betöltésük a regiszterbe.

HIÁNYOS !!!

Folyamatábra ?

Áramköri megvalósítás ?

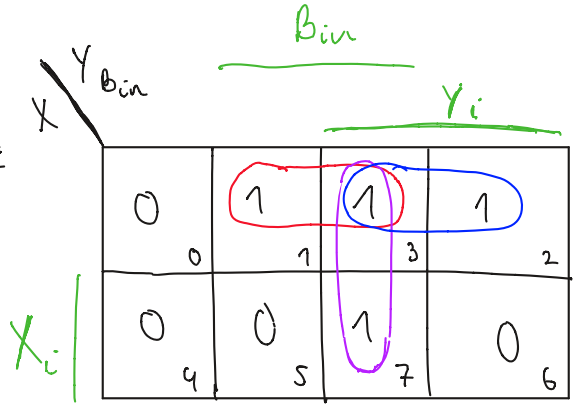
(7.) Adja meg egy 1-bites Full Substractor (FS) igazságtáblázatát, Karnough tábláit, kapuszintű kapcsolási rajzát, továbbá a 6-bites RCA (Ripple Carry Adder) kapcsolási rajzát és számítási időszükségletét!

Igazságtábla:

X_i	Y_i	B_{in}	F_i	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

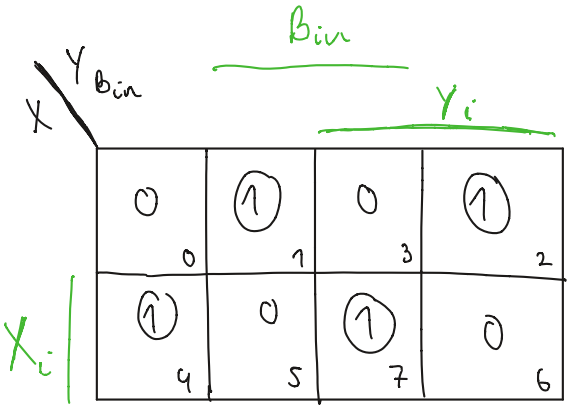
Karnough tábla:

B_{out}:



$B_{out} = \overline{X_i} \cdot B_{in} + B_{in} \cdot Y_i + \overline{X_i} \cdot Y_i$

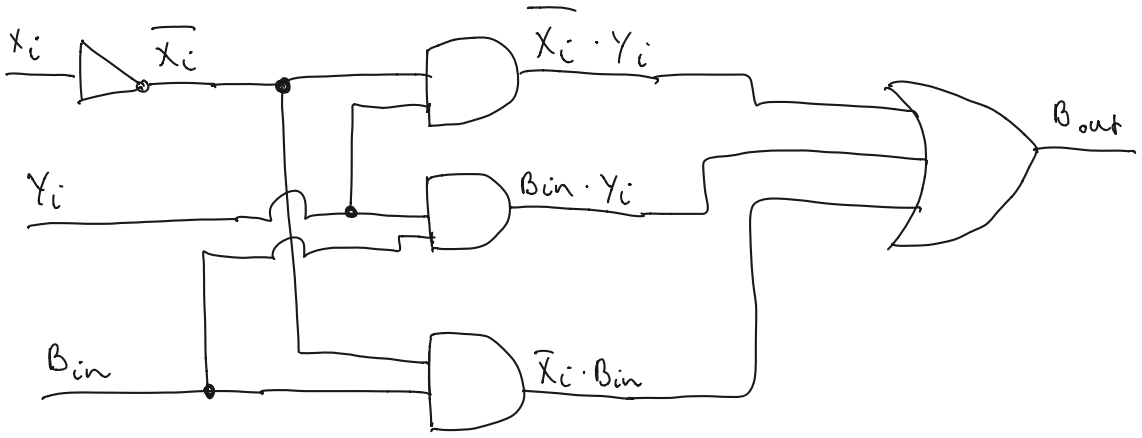
F_i:



$F_i = X_i \cdot \overline{B_{in}} \cdot \overline{Y_i} + \overline{X_i} B_{in} \overline{Y_i} +$

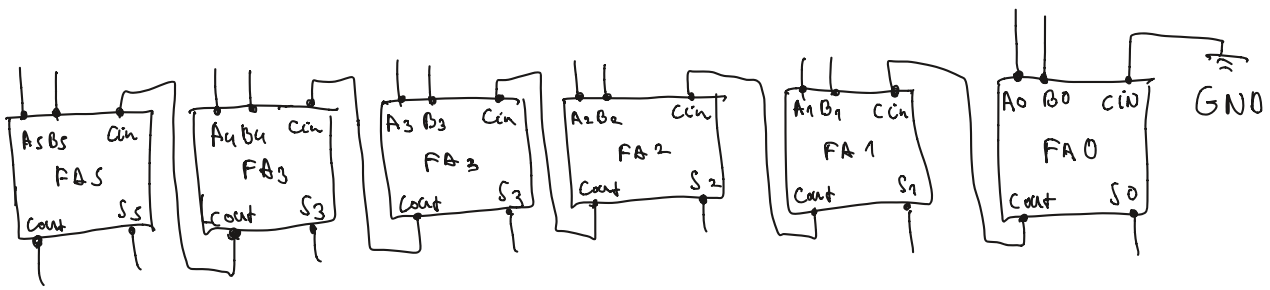
$+ X_i B_{in} Y_i + \overline{X_i} \overline{B_{in}} Y_i$

Kapuszintű kapcsolási rajz



RCA (Ripple Carry Adder) 6-bites

Kapcsolási rajz:



Számítási időszükséglet:

$$T_{(RCA)} = N \cdot T_{(FA)} = N \cdot (2 \cdot G) = 12\,G \quad \text{(6 bites RCA esetén)}$$

N - Bitek száma

T - idő

FA - Full Adder

G - kapu késleltetése

? (8.) Adja meg a magas-szintű szintézis (HLS) elméleti alaptételét!

Turing Church tézis: (emlékeztető)

A szimbolikus számítások 3 fő ekvivalens reprezentációi a következők:

- Turing gép (TM - hardver)
- μ - rekurzív függvény (algoritmusok, program)
- CFG - környezetfüggetlen nyelvtan (OS)

Egy nyelv, amely leírja a teljes tervezési folyamatot, az egyes tervezési lépések jóságát méri: Nyelv <=> HK

9. Adott: memória hozzáférés ideje 20 ns, regiszterből regiszterbe másolás 4 ns, kivonás 6 ns. Adja meg a ADD2 *X, *Y RTL leírását és időszükségletét!

Fetch: (Regiszterek feltöltése utasítással)

PC → MAR 4 ns

M[MAR] → MBR 20 ns

PC + I_len → PC 4 ns

MBR → IR 4 ns ⎫ 32 ns

Decode: (a dekódolást általában 0 idejűnek feltételezzük)

Execute: (végrehajtás)

PC → MAR 4 ns

PC + XA_len → PC 4 ns

M[MAR] → MBR 20 ns

MBR → MAR 4 ns

M[MAR] → MBR 20 ns

MBR → MAR 4 ns

M[MAR] → MBR 20 ns

MBR → T1 4 ns

PC → MAR 4 ns

PC + YA_len → PC 4 ns

M[MAR] → MBR 20 ns

MBR → MAR 4 ns

M[MAR] → MBR 20 ns

MBR → MAR 4 ns

M[MAR] → MBR 20 ns

MBR → T2 4 ns

T1 − T2 → MBR 6 + 4 ns

MBR → M[MAR] 20 ns

$44 + 140 + 6 \rightarrow \Sigma\ 190$ ms

Roli megoldása:

Reg : 4 ns
Memória: 20 ns
Reg : 4 ns
Reg : 4 ns Fetch: 32 ns

Reg : 4 ns
Reg : 4 ns
Mem. : 20 ns
Reg : 4 ns
Mem : 20 ns
Reg : 4 ns
Mem : 20 ns
Reg : 4 ns X értéket T_1-be töltjük : 80 ns

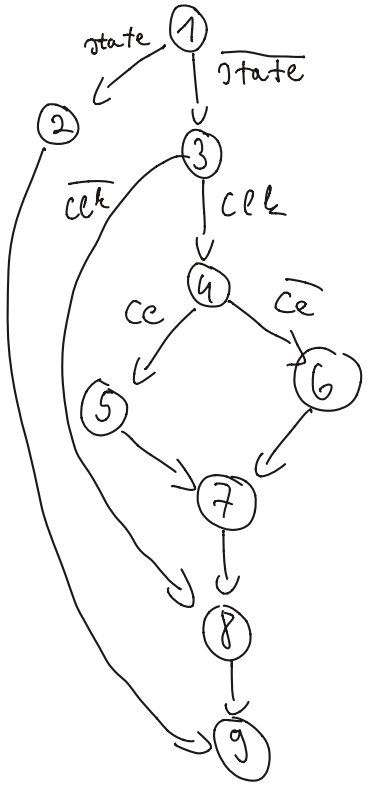
Reg : 4 ns
Reg : 4 ns
Mem : 20 ns
Reg : 4 ns
Mem : 20 ns
Reg : 4 ns
Mem : 20 ns
Reg : 4 ns Y értéket T_2-be töltjük : 80 ns

ALU + Reg : 6 + 4 ns } Σ 30 ns
Mem. : 20 ns

$\Sigma = 222$ ns $\rightarrow$ 3,7 sec

(10.) Adja meg az alábbi VHDL szubrutin CFG (Vezérlésfolyam) gráfját:

```
TemplAddr : process (clk,state)
begin
   if state = Waiting then 1
        TemplAddrReg <= 0; 2
   else
        if clk'event and clk = '1' then 3
             if ce = '1' then 4
                  TemplAddrReg <= TemplAddrRegNext; 5
             else
                  TemplAddrReg <= TemplAddrReg; 6
             end if; 7
        end if; 8
   end if; 9
end process;
```



(11.) Adja meg a szekvenciális hálózatok leírási módjait.

igazságtábla, algebrai kifejezések, elvi logikai vázlat.

állapottábla, állapotgráf,