ÓBUDAI EGYETEM

SZÁMÍTÓGÉP ARCHITEKTÚRÁK ALAPJAI

3-4. előadás

Durczy Levente

2025/2026/1

Az aritmetikai egység felépítése és működésének alapelvei:

A processzor szintű fizikai architektúra részei:

- műveletvégző
- vezérlő
- I/O rendszer
- megszakítási rendszer

A CPU két fő funkciója a műveletvégzés és vezérlés

CPU típusok:

- Szinkron CPU: Órajel generátorra működik
  H: késleltetés: a következő órajelet mindig meg kell várni, így holtidő keletkezhet
  E: egyszerű, gyors
- Aszinkron CPU:
  Egy utasítás befejezése után szinte közvetlenül indul a következő utasítás
  H: speciális áramkör kell az utasítás befejezésének érzékelésére, ami drága, és érzékelése plusz idővel jár
  E: nincs holt idő

Műveletvégző (ALU) részei:

- regiszterek
- adatutak
- kapcsolópontok
- szűkebb értelemben vett ALU

Regiszterek:

- látható regiszterek
  - univerzális: bármilyen értéket beletehet a programozó (méret korlát!)
  - dedikált, pl.: stack
- rejtett regiszterek:

  adatfeldolgozáshoz szükséges puffer regiszterek. Hivatkozni nem lehet rájuk, de alacsony szintű programozásnál számításba kell venni őket.

Adatutak:

FONTOS, hogy ez NEM BUSZ! Adatbuszon értelmezett a címzés, míg adatutak esetében nem. A processzor belső részeit kötik össze. Az adatút egy vezetékrendszerként fogható fel, mely összeköti a regisztereket, puffer regisztereket és ALU-t. Adatúton egyszerre csak egy adat lehet!

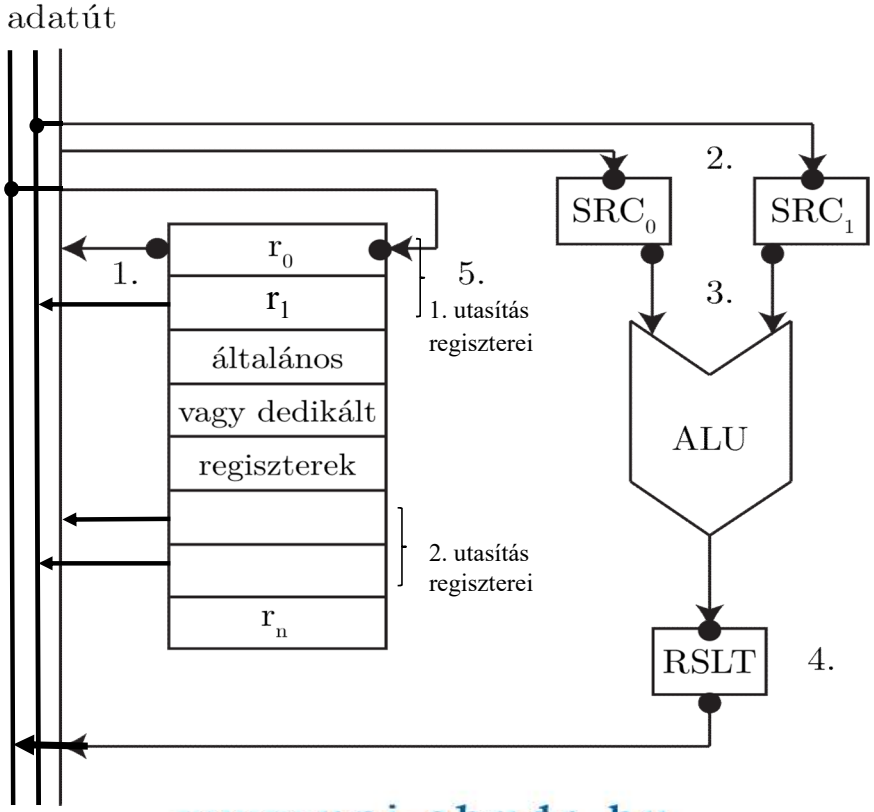
1.2.3. Kapcsolópontok

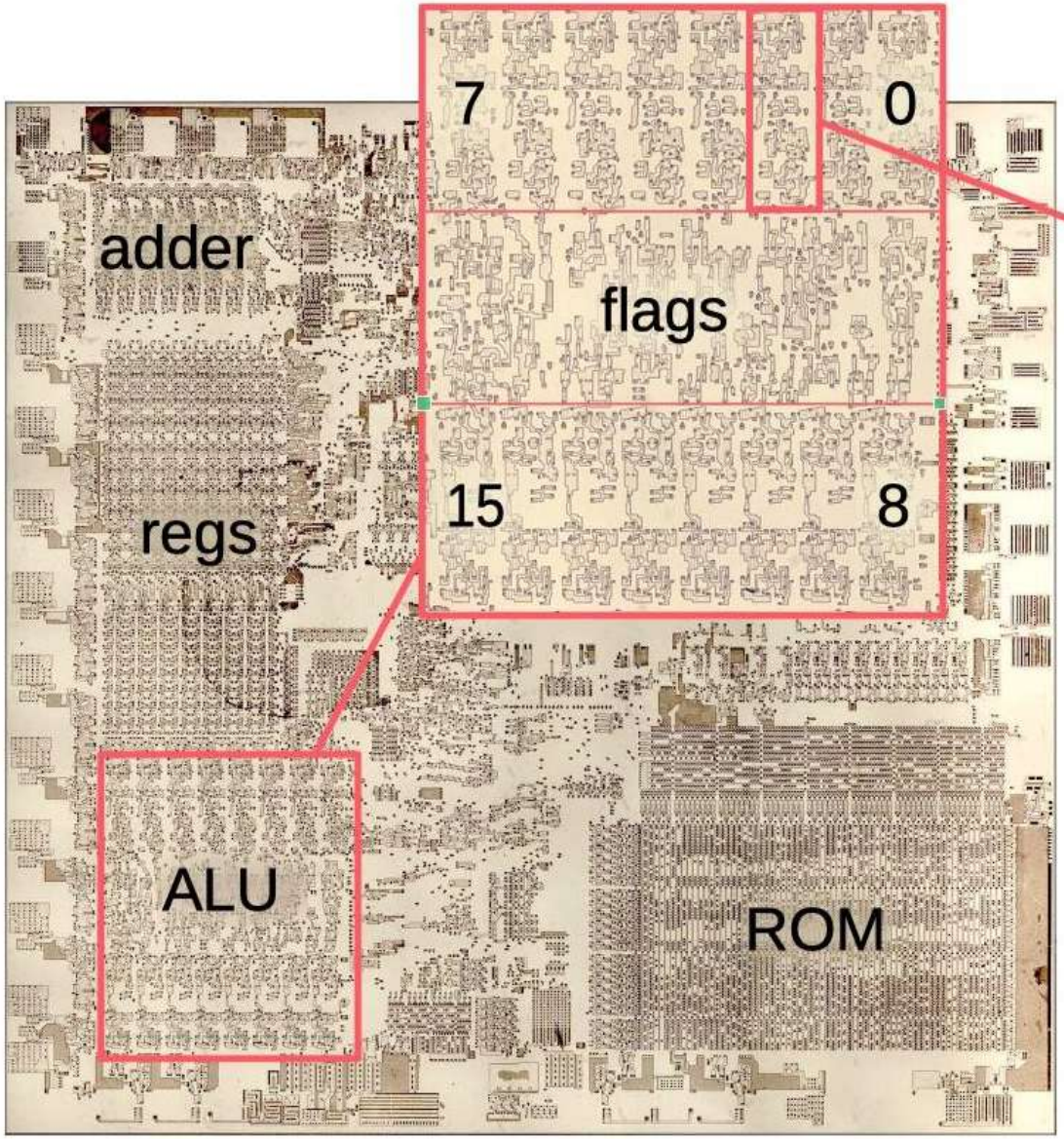
A regiszterek bemenetén és kimenetén lévő **tranzisztorok**.

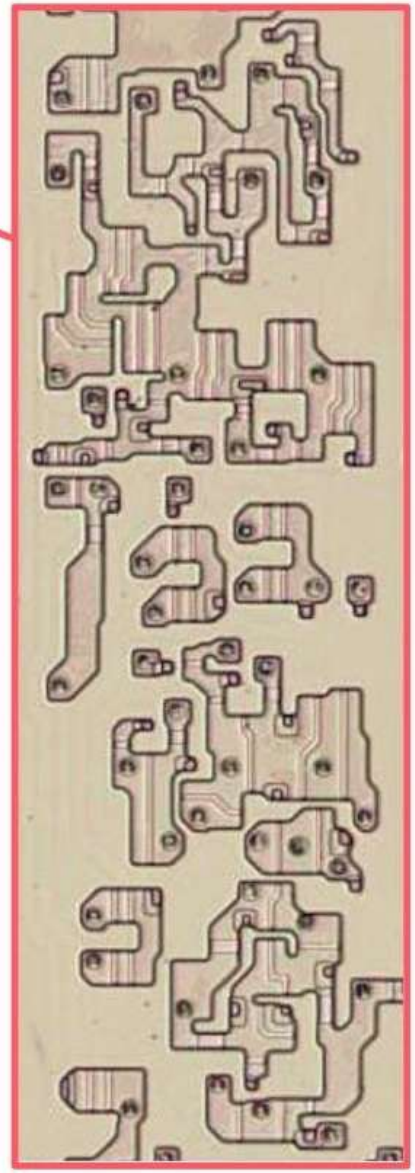
A vezérlő feladata a kapcsolók állapotának megváltoztatása.

Kimeneti kapcsoló: 3 állapot: 1, 0 vagy zárt

Bemeneti kapcsoló: 2 állapot zárt vagy nyitott







Intel 8086 CPU

ALU felépítése

ÓBUDAI EGYETEM

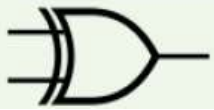
www.uni-obuda.hu

Az aritmetikai egység megvalósítása:

Legegyszerűbb logikai áramkörök:

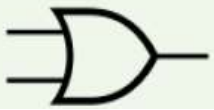
- XOR (Kizáró VAGY) kapu:

Bemenet		Kimenet
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



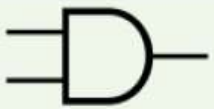
- OR (VAGY) kapu:

Bemenet		Kimenet
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1



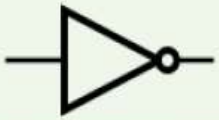
- AND (ÉS) kapu:

Bemenet		Kimenet
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1



- NOT (Inverter) kapu:

Bemenet	Kimenet
A	NOT A
0	1
1	0



Az egybites félösszeadó:

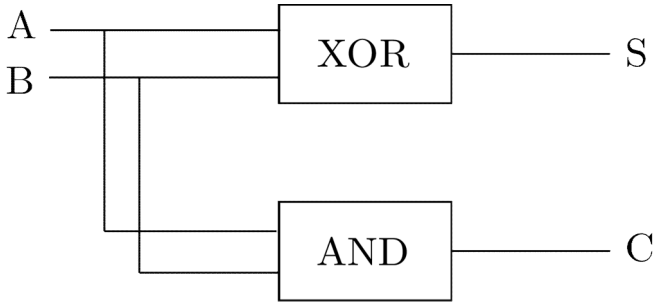
A és B bemeneti operandus esetén összeadás eredménye S, ahol C az átvitel a következő igazságtábla alapján:

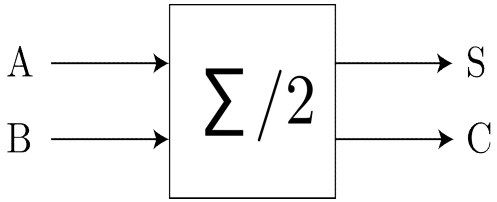
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Megvalósítása:

$S = \overline{A}B + A\overline{B}$ = A XOR B

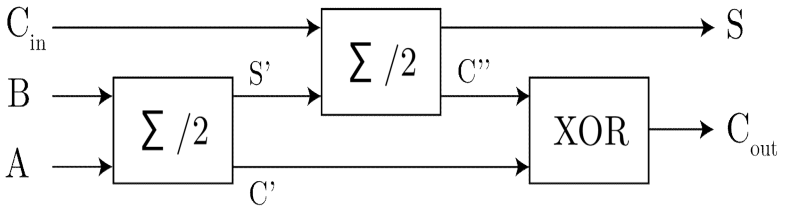
$C = AB$





Az egybites teljes összeadó:

Megvalósítás félösszeadókkal:

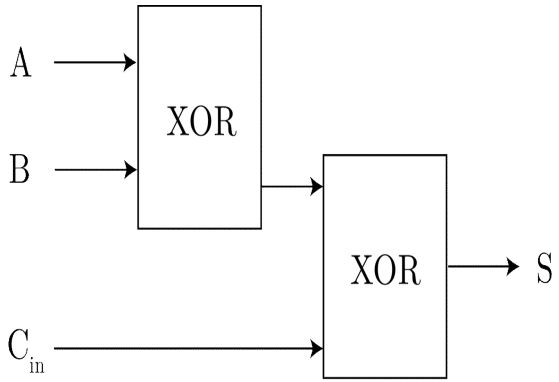


Logikai függvény (igazságtábla alapján):

$$S = \overline{A}BC_{in} + A\overline{B}C_{in} + \overline{A}\overline{B}\overline{C}_{in} + ABC_{in} = \underbrace{(\overline{A}B + A\overline{B})}_{A \text{ XOR } B}C_{in} + \underbrace{(A\overline{B}+\overline{A}B)}_{A \text{ XOR } B}\overline{C}_{in}$$

Legyen: $X = A \text{ XOR } B \longrightarrow \overline{X}\,C_{in} + X\,\overline{C}_{in}$

$$S = \overline{X}C_{in} + \overline{X}C_{in} = X \text{ XOR } C_{in} = A \text{ XOR } B \text{ XOR } C_{in}$$



A	B	C_{in}	S	C_{out}
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

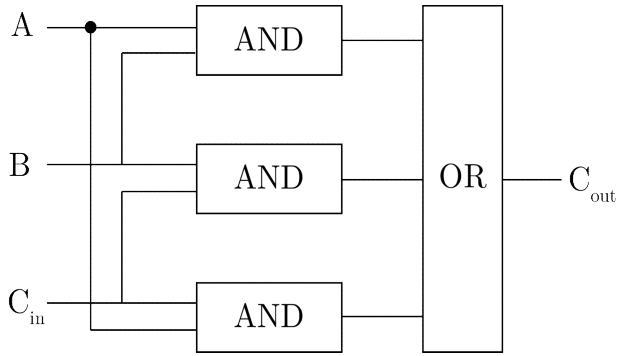
Logikai kapuk: AND, NAND, OR, NOR, XOR, NXOR, NOT (inverter)

$C_{out} = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC + ABC + ABC$

Azonosságok: ($A+A=A$, $AB+AB=AB$, $A+\bar{A}=1$!!!)

$C_{out} = (A+\bar{A})BC_{in} + (B+\bar{B})AC_{in} + (C_{in}+\bar{C}_{in})AB = BC_{in} + AC_{in} + AB$

$C_{out} = AB + (A+B)C_{in}$



N bites soros összeadó:

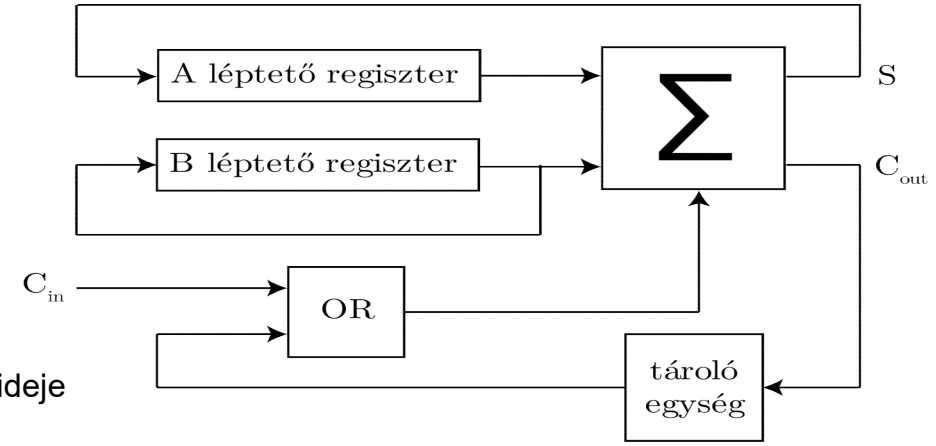
E: egyszerű, olcsó

H: lassú

Végrehajtási idő: T = n * t

„n” : bitek száma

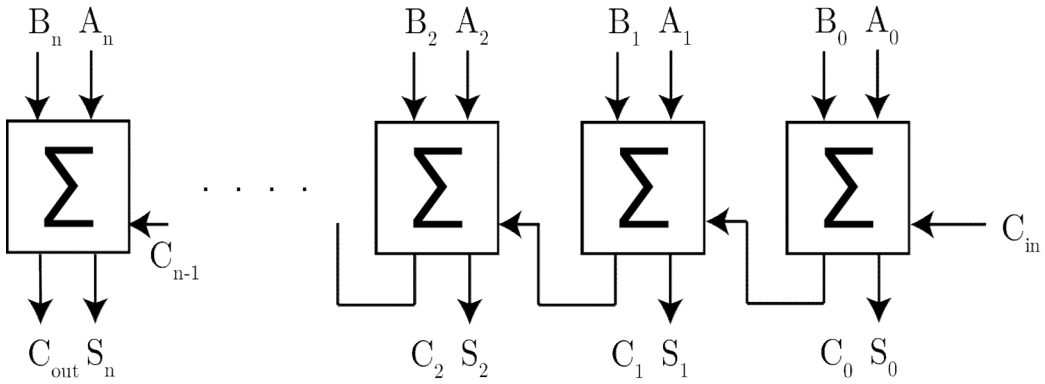
„t” : egybites összeadó végrehajtási ideje



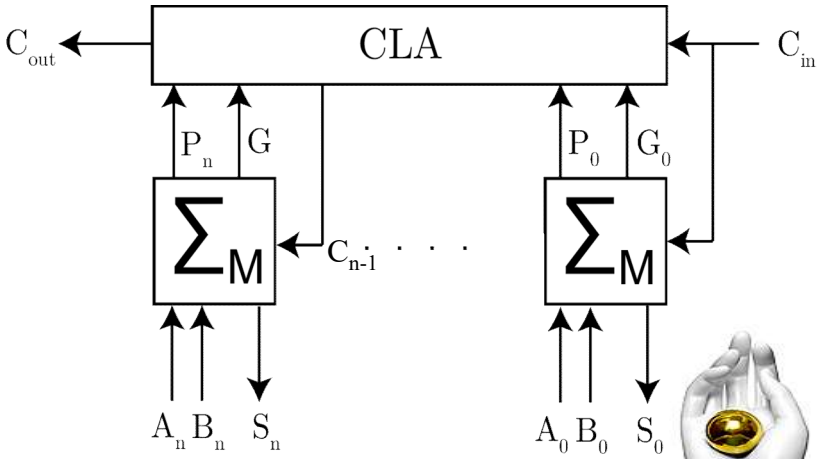
(1bites késleltető)

N bites párhuzamos összeadó:
(„ripple carry adder”)

E: párhuzamos összeadás
H: hullámzó végrehajtási idő!
(sok carry esetén lassabb)



Megoldás: Előre jelzett átvitellel felépített n-bites összeadó:
(CLA: carry look-ahead, vagy rekurzív módszer)

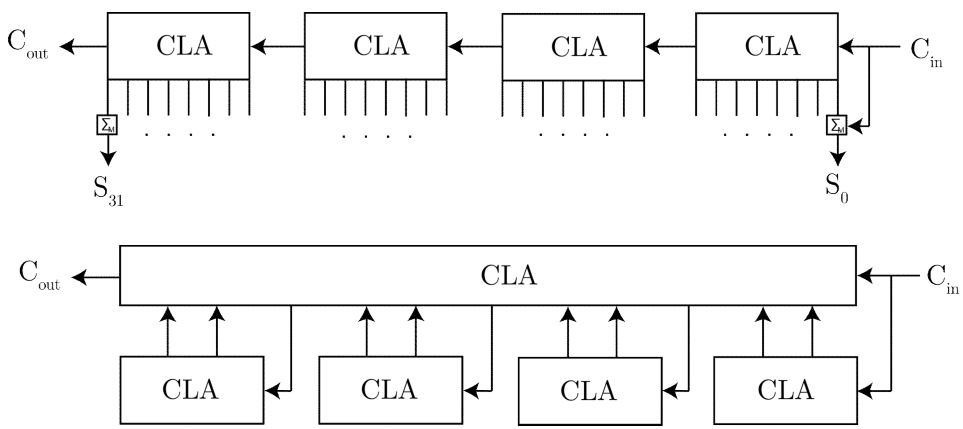


$C_{out} = AB + (A+B)C_{in}$
Legyen AB = **G** (generálja az átvitelt)
A+B = **P** (propagálja az átvitelt)
$C_{out} = G + Pc_{in}$
$C_0 = G_0 + P_0C_{in}$
$C_1 = G_1 + P_1C_0 = G_1 + P_1G_0 + P_1P_0C_{in}$
$C_2 = G_2 + P_2C_1 = G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_{in}) = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_{in}$
$C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{in}$

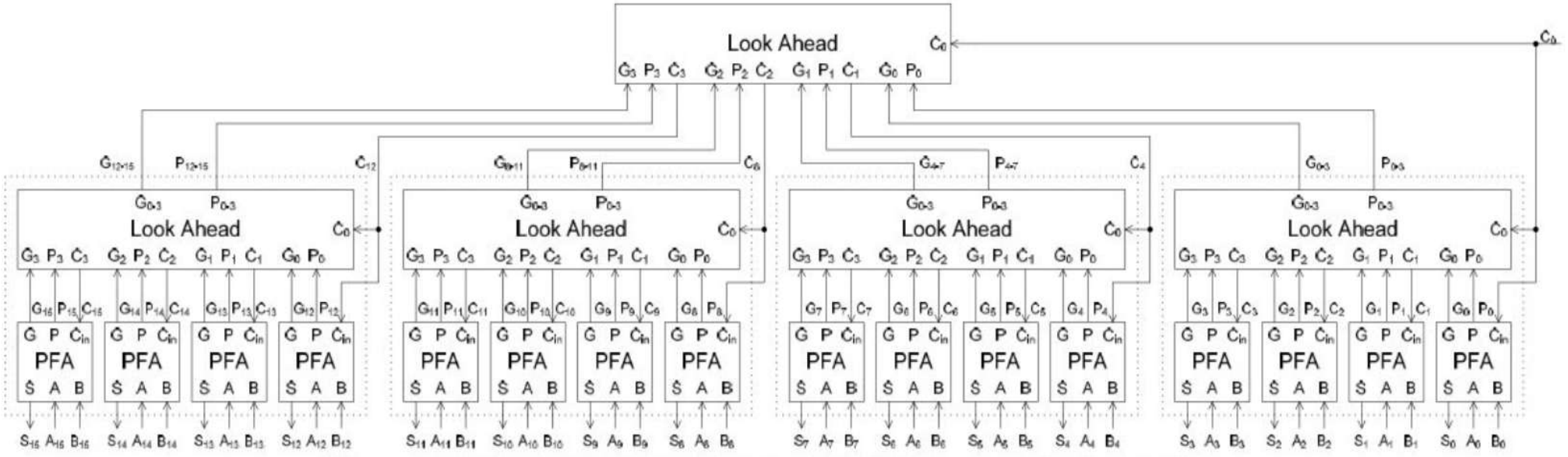
Egy „OR” kapunak általában max. 8 bemenete lehet!
16 bit esetén már 4db CLA áramkör kell (lassul!)

$T_n = 2t + t$ (P és G előállítása)

Megoldás:



Pl: 16 bites CLA összeadó ábrája



Fixpontos szorzás:

Komplex feladat: összeadás, invertálás, léptetés, … időigényes!

Pl.: Intel 8086 CPU: ADD ⟶ ~ 3 ciklus

MUL ⟶ ~ 118-133 ciklus! (DIV ~ 144-162)

Két „n” bites szám szorzásának eredménye „2*n” bit lesz!

Gyorsítási lehetőségek:

1. Bitcsoporttal történő szorzás:

Pl.: 10-es számrendszerben: 7*9=63

2-es számrendszerben: 0111*1001

Ahelyett, hogy bitenként hajtjuk végre a szorzást, történhet ez bitcsoportonként:

$$
\begin{array}{rl}
0111*10|01 & \\
\hline
0000 & \textit{gyűjtő} \\
01\overset{\wedge}{1}1 & \textit{egyszerese és léptetés kétszer} \\
\hline
01\overset{\wedge}{1}1 & \\
1\overset{\wedge}{1}1000 & \textit{kétszerese és léptetés kétszer} \\
\hline
1\overset{\wedge}{1}1111 &
\end{array}
$$

Szabály:

00: léptetünk kettőt
01: hozzáadjuk az egyszeresét és léptetünk kettőt
10: hozzáadjuk a kétszeresét, majd léptetünk kettőt
11: négyszeresét adjuk hozzá és kivonjuk belőle az egyszeresét

Booth algoritmus:

Ha a szorzóban sok 1-es van, akkor lassú a szorzás, mert sok összeadást kell végezni.

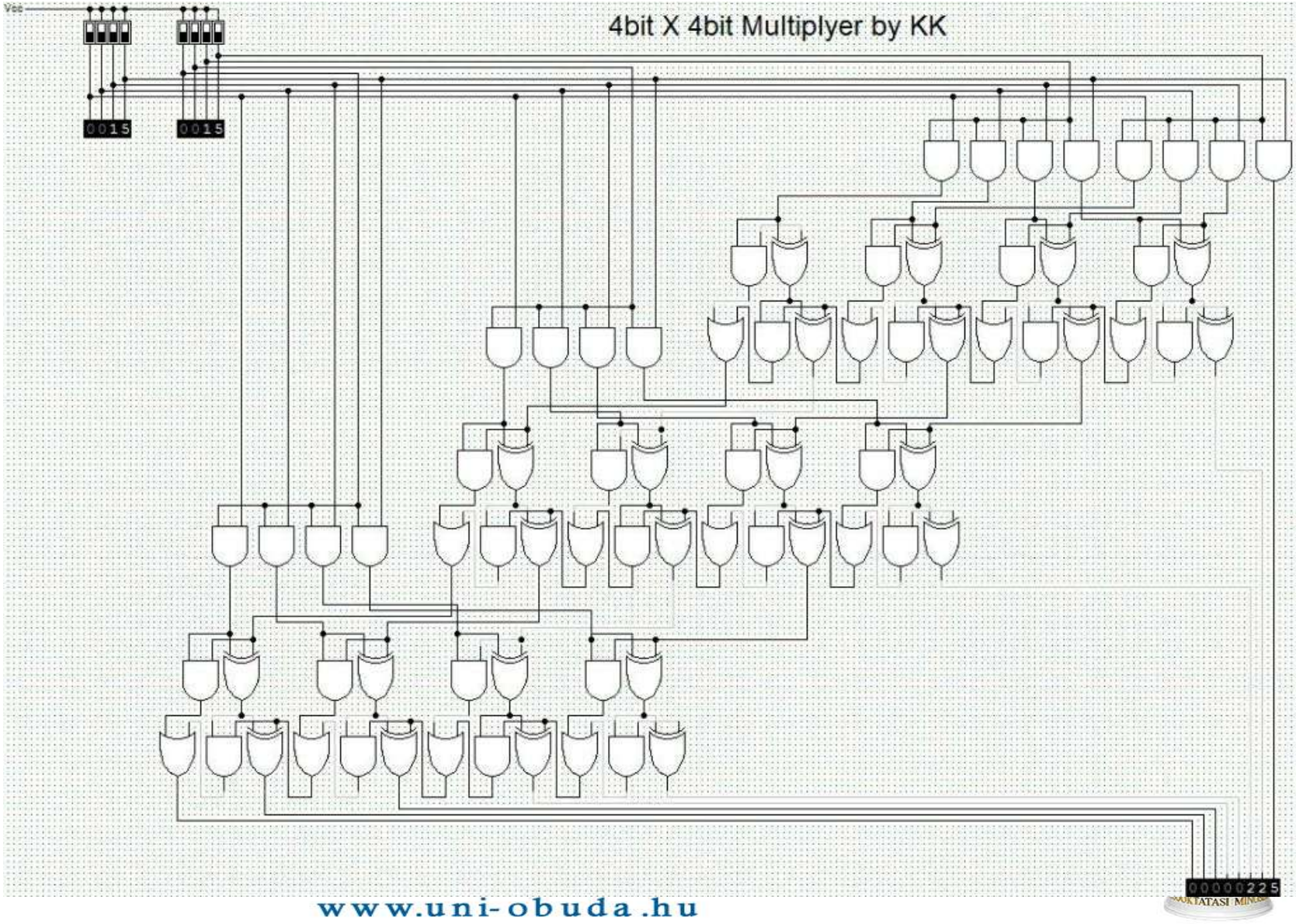
Pl.: X*62 (111110) szorzást több lépcsőben kellene elvégezni. Helyette:

Keressük meg a szorzóhoz legközelebbi „kerek” számot és állítsuk elő az eredményt úgy, hogy:

X*(64-2) = X*64 – X*2

- 64 (1000000b) → sok léptetés + 1 összeadás
- 2 (10b) → 1 összeadás
- kivonás → 1 összeadás (negált-tal)

4 bites szorzó áramkör példa:



Ma 64 bites szorzók vannak,

Bitenként ~ 16db logikai kapu.

ÓBUDAI EGYETEM

Lebegőpontos számok:

Kezdet: Konrad Zuse 1941: „Z3” , 1 tonna, FP számolás (rejtett bit!), összeadás ~0.7 sec
Howard: Mark I. 1944: fél focipálya, csak FX, összeadás ~ 3-5 sec

A FX számok értelmezési tartománya viszonylag kicsi, pontosság sem túl jó!
Például 16 bit esetén -32768 ⟶ +32767

Szabvány (1985): IEEE754 (a legjobb megoldásokat gyűjtötték egybe!)
FP = M * r^k **M**: mantissza, **r**: radix (számrendszer alapja), **k**: karakterisztika
Elvárás, hogy a radix egyezzen meg a mantisszánál használt számrendszer alapjával!
Példa 2-es számrendszerben**:** **$0{,}10011*10^{101}$**
Pontosság: pl.: 0,3 ⟶ 0.01001100110011001100 (végtelen törtszám)

Ábrázolása: Normalizált formátumban! $0{,}001*2^k \rightarrow 0{,}1*2^{k-2}$

Mantissza értéke:
10-es számrendszernél: $0{,}1 <= M < 1$
2-es számrendszer: $\frac{1}{2} <= M < 1$
általános számrendszer: $1/r <= M < 1$

ÓBUDAI EGYETEM

Értelmezési tartomány:

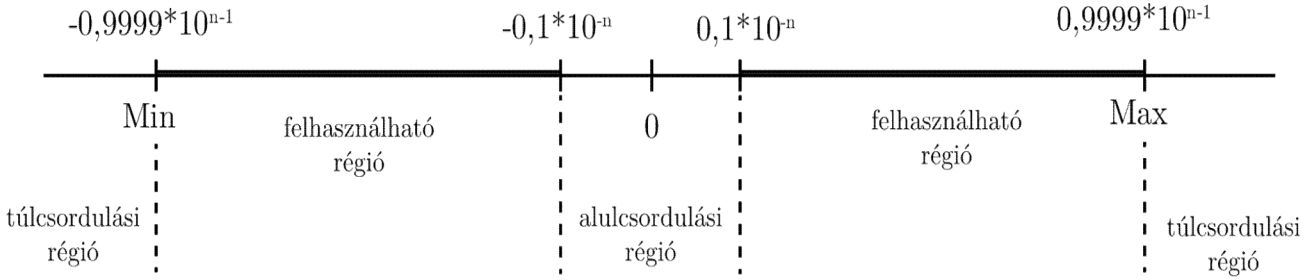
Függ: - karakterisztika számára rendelkezésre álló bitek számától

- radixtól

Karakterisztika bitek száma	Legnagyobb érték$_{10}$	Értelmezési tartomány$_{10}$	Legnagyobb érték$_2$	Értelmezési tartomány$_2$
1	±9	$10^{\pm 9}$	1=1	$2^{\pm 1}$
2	±99	$10^{\pm 99}$	11=3	$2^{\pm 3}$
3	±999	$10^{\pm 999}$	111=7	$2^{\pm 7}$
4	±9999	$10^{\pm 9999}$	1111=15	$2^{\pm 15}$ (=FX16 bit: 32768)

Pontosság: Mantisszai bitjeinek számától függ!

Példa: 10-es számrendszer, 4 mantissza bit esetén:



A karakterisztika maximális értéke (minden bitje 1-es) a +- ∞ kijelzésére van fenntartva!

Az architektúrának biztosítania kell a túlcsordulás és alulcsordulás felfedezését, jelzését és kezelését.

Túlcsordulás esetén:

- kijelzi és beállítja a legnagyobb megengedett értéket, vagy
- előjeles végtelent jelez ki

Alulcsordulás esetén:

- kijelzi és 0-ra konvertál, vagy
- a denormalizált számot jelzi ki

Elvárás, hogy ha a mantissza 0, akkor a karakterisztika is 0 legyen!

Pontosság növelése:

Rejtett bit használata:

Mivel normalizált formátumban az első értékes bit van a tizedespont után és az minden esetben 1-es, így **tárolásnál** annak nincs információ tartalma.

0,101

0,1110

0,10001 Ezért csak a második bittől történik a tárolás

Ha a mantissza 23 bit, akkor 24 bitet tudunk eltárolni ⟶ nő a pontosság

Őrző bitek:

Elvárás a lebegőpontos számokra, hogy a relatív hiba kisebb legyen, mint a normalizált eredmény legkisebb számjegye!

Megoldás: a CPU-n belül a regiszterek több biten tudják tárolni a mantisszát (általában plusz 3-15 bit)

Felhasználásuk:

- a rejtett bit balra léptetésekor értékes bitet tudunk beléptetni (helyreállítás)
- tárolási formátum kérésekor kerekített értéket tárolhatunk
- normalizáláskor értékes biteket tudunk felhasználni

FP számok kódolása:

- Mantissza kódolása: 2-es komplemens
- Karakterisztika: többletes kódolás ⟶ kialakítása gyorsabb, de csak alapműveletekre alkalmas

Szabvány: **IEEE 754:**

1985-ben jelent meg az eddigi legjobb megoldások alapján. Célja megkönnyíteni a különböző CPU-k esetén az adatszintű kompatibilitást, portabilitást.

Rendszerszintű megoldás, vagyis a hardvernek és a szoftvernek együtt kell biztosítania a szabványnak való megfelelést.

Szabvány: **IEEE 754:**

Fejezetei:

1. adattípus 2. formátumok 3. műveletek 4. kerekítések 5. kivételek

Adattípus: egyszeres (32 bit), kétszeres (64 bit), kiterjesztett (80 bit), négyszeres (128 bit) pontosságú

Formátumok:

1. Szabványos: háttértáron való tároláshoz, kötött
   a) egyszeres pontosságú (32 bit: kisebb, gyorsabb, pontatlanabb)

1 előjel bit	8 bit karakterisztika	23 bit mantissza

   b) kétszeres pontosságú (64 bit: nagyobb, lassabb, jóval pontosabb)

1 előjel bit	11 bit karakterisztika	52 bit mantissza

(értelmezési tart.: ~$10^{\pm 308}$)

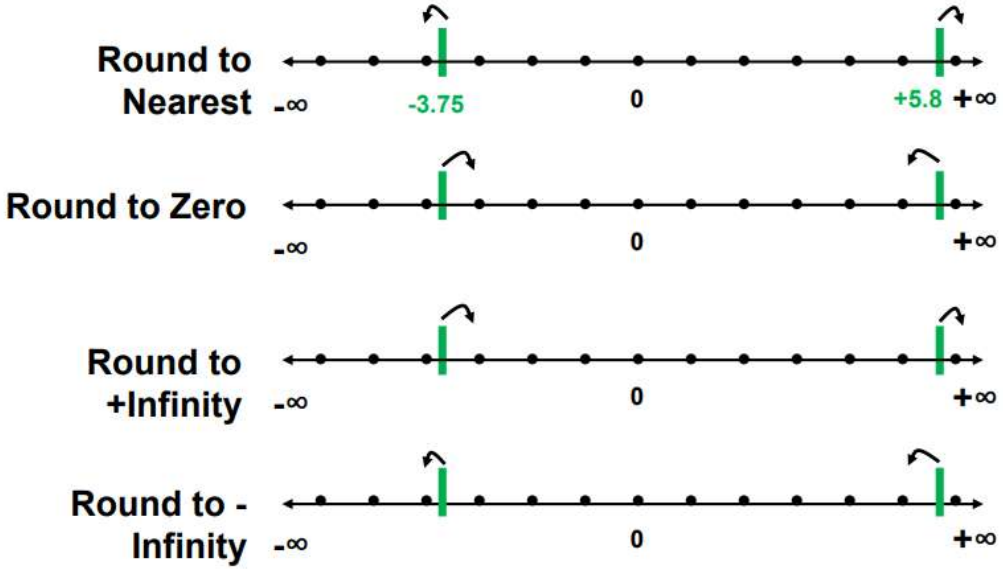
2. Bővített: CPU-n belül, nagyobb szabadság
   a) egyszeres pontosságú (min. 43 bit)
   b) kétszeres pontosságú (min. 79 bit)

Értelmezett műveletek:

4 aritmetikai alapművelet, maradékképzés, gyökvonás, bináris - decimális konverzió, végtelennel való műveletvégzés, kivételek kezelése, …
pl.: összeadás esetén először a karakterisztikákat azonos értékre kell hozni!

Kerekítések:

- legközelebbire való kerekítés
- 0-ra kerekítés (őrző bitek levágását jelenti)
- kerekítés pozitív végtelen felé
- kerekítés negatív végtelen felé
  (első kiberbűncselekmények!)



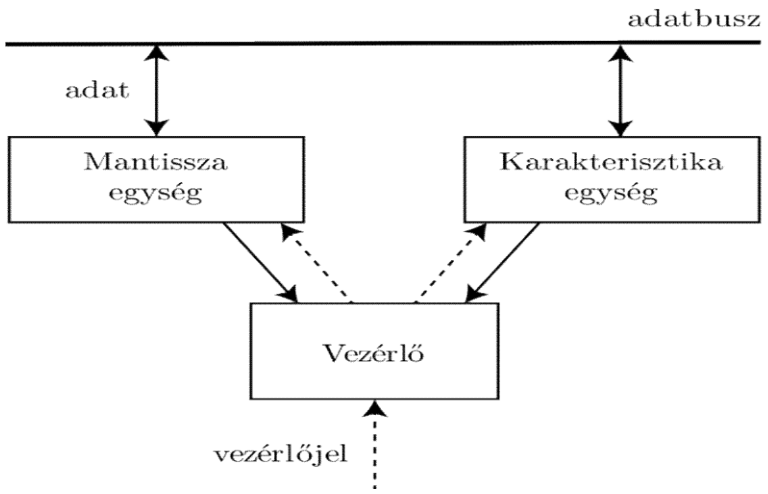
Kivételkezelés:

Felbukkanásuk általában megszakítást eredményez.
Pl.: túlcsordulás, alulcsordulás, 0-val való osztás, gyökvonás negatív számból, …

Fizikai megvalósítás:

kezdetben többszörös FX műveletek

- dedikált FP műveletvégző:
  (párhuzamos műveletvégzés!)



Példa: Intel 8087 coprocessor (1980):

Ebből lett a szabvány alapja!

3 µm, max. 10 MHz,

~ 45.000 tranzisztor

(8086-os CPU: ~ 29.000!)

Teljesítmény növekedés:

20%-500%!

50.000 FLOPS

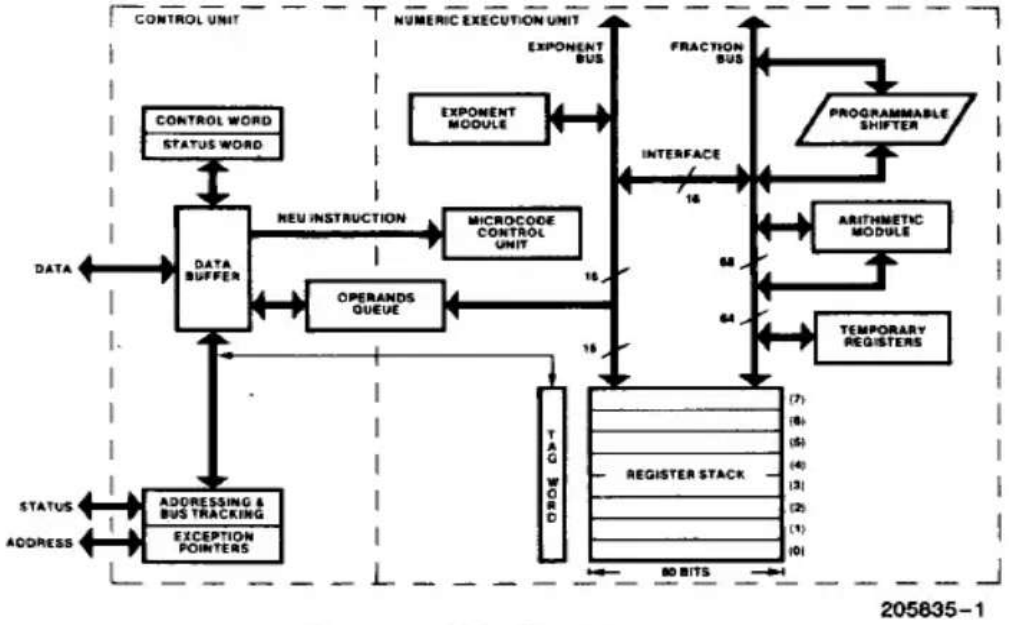


Figure 1. 8087 Block Diagram

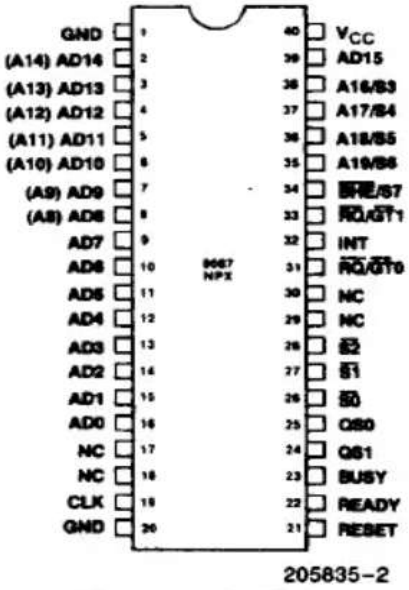
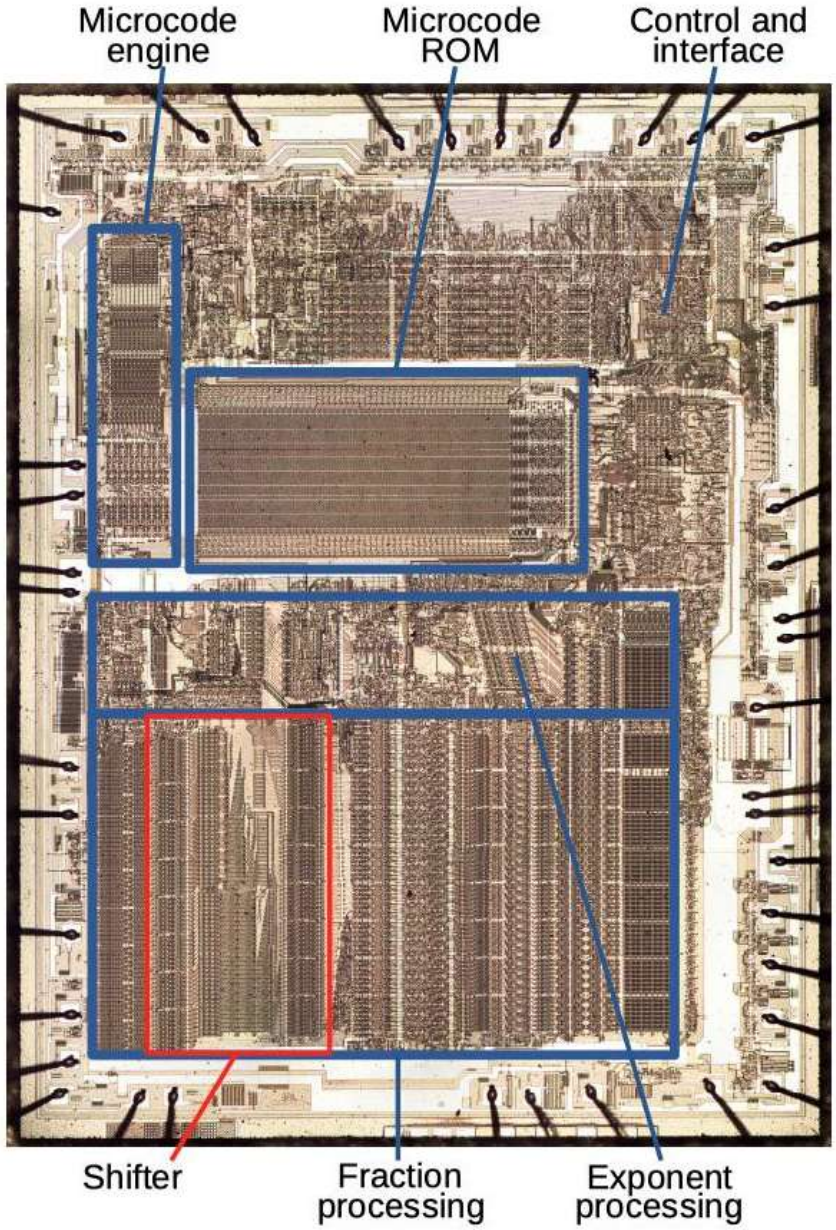


Figure 2. 8087 Pin Configuration

8087 coprocessor áramköri képe:

Teljesítmény: Relatív mutató: 8086 CPU ⟶ 1
Intel 80386 (25 MHz) ⟶ ~ 17
Intel 80486 (60 MHz) ⟶ ~ 1700
Intel Pentium (133 MHz) ⟶ ~ 6000



BCD számábrázolás:

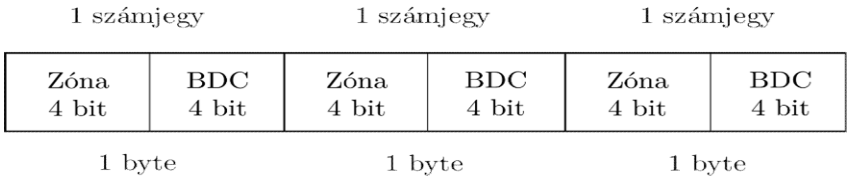
Megjelenésének oka:
FX és FP számábrázolás pontatlansága. Elsősorban adminisztratív alkalmazásoknál használják.
A kódolás pontos megfeleltetés ⟶ nem kell kerekíteni.
(pl.: Basic programnyelv)

Ábrázolás:
4 biten történik 0-9 tartományban, amit tetrádnak hívnak.
A 10-15 tartományba esőket pedig érvénytelen tetrádnak.

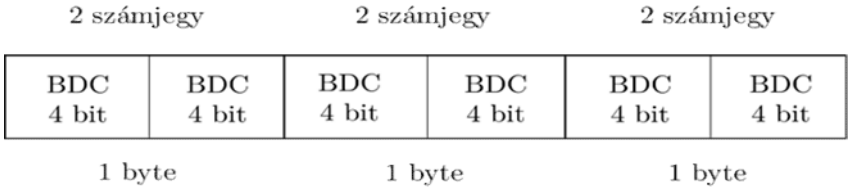
0	0000	
9	1001	
A	1010	Érvénytelen tetrádok
B	1011	
C	1100	
D	1101	
E	1110	
F	1111	

Érvénytelen tetrádot úgy lehet megállapítani, hogyha az első bitje 1-es, akkor vagy a második vagy harmadik bitje is egyes

Formátum: a) zónázott:



b) pakolt:



A hossz lehet fix vagy változó. Változó esetén specifikálni kell!

Műveletvégzés:

Például: 8+7=15

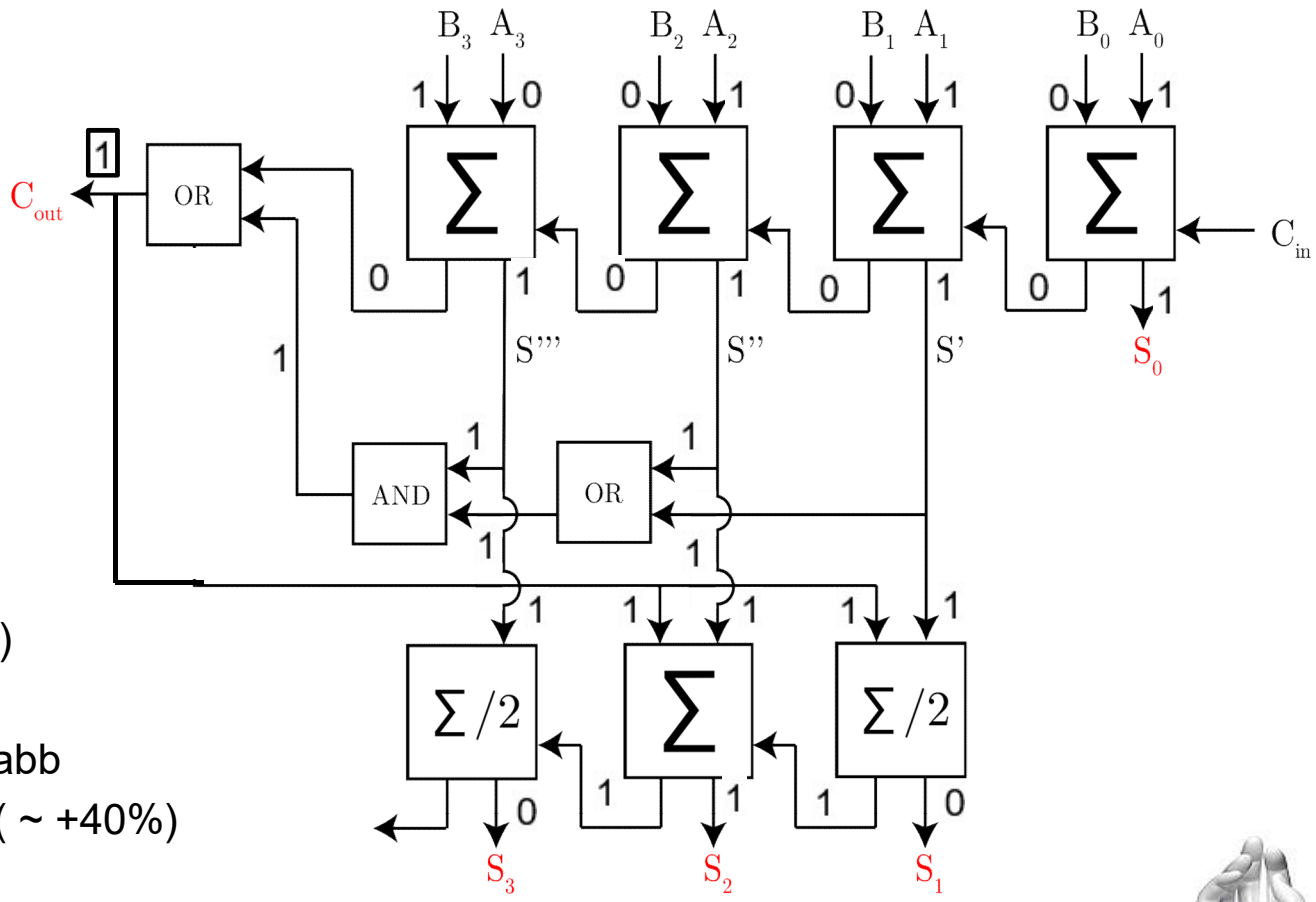
	1000	8
+	0111	7
	1111	15
+	0110	6
	1|0101	1|5

negált: 1010 → 0101 + 0001 = 0110

Megvalósítása:

Példa összeadásra:
8 + 7 = 1000 + 0111

Eredmény: 1 0101
1 5



Előnye: teljesen pontos!
Az FP ábrázolás egy alternatívája
(pl: bank, biztosító, nyugdíj intézetek)

Hátrány: komplexebb, drágább, lassabb
rossz memóriakihasználás (~ +40%)

Összefoglalás (FX, FP, BCD):

Fixpontos számok (FX):

Előny:

- Gyors
- egyszerű a megvalósítása
- kevés helyet igényel
- kezeli a 8, 16, 32, 64 bites formátumot

Hátrány:

- kicsi értelmezési tartomány
- tört számoknál / osztásnál pontatlan lehet

Lebegőpontos számok (FP):

Előny:

- nagy értelmezési tartomány
- általában elég nagy pontosságot biztosít

Hátrány:

- több erőforrást igényel
- legtöbbször kerekíteni kell

ALU egyéb műveletei:

mind a 16 fajta BOOLE művelet (AND, NOR, OR, XOR, …)
Léptetés, invertálás, komparálás (+feltételes ugrás)
LOAD/STORE címszámítás → végrehajtási időben!
karakteres műveletek

Vezérlőegység:

Processzor szintű fizikai architektúra:
- műveletvégző egység
- vezérlőegység
- I/O rendszer
- megszakítási rendszer

A CPU-ban ez az egyik legbonyolultabb áramkör, ez a „lelke” processzornak.

A benne található ütemező felelős azért, hogy úgy állítsa elő a vezérlőjeleket, hogy azok szinkronban legyenek az órajellel.

Fejlődése:

szekvenciális: - centralizált (egy vezérlő van)

lehet huzalozott, vagy mikroprogramozott

párhuzamos: - decentralizált (több vezérlő egy processzoron belül)

szuperskalár, vagy pipeline

(ezek hibrid vezérlők, ami azt jelenti, hogy tartalmaznak huzalozott és mikroprogramozott vezérlőrészeket is)

Vezérlés:

3 részre osztható:

- Front-end: Fetch, Decode, Branch prediction
- Ütemező: a már dekódolt utasításokat megfelelő „útra” tereli
- Back-end: Execute (ALU), Write-back

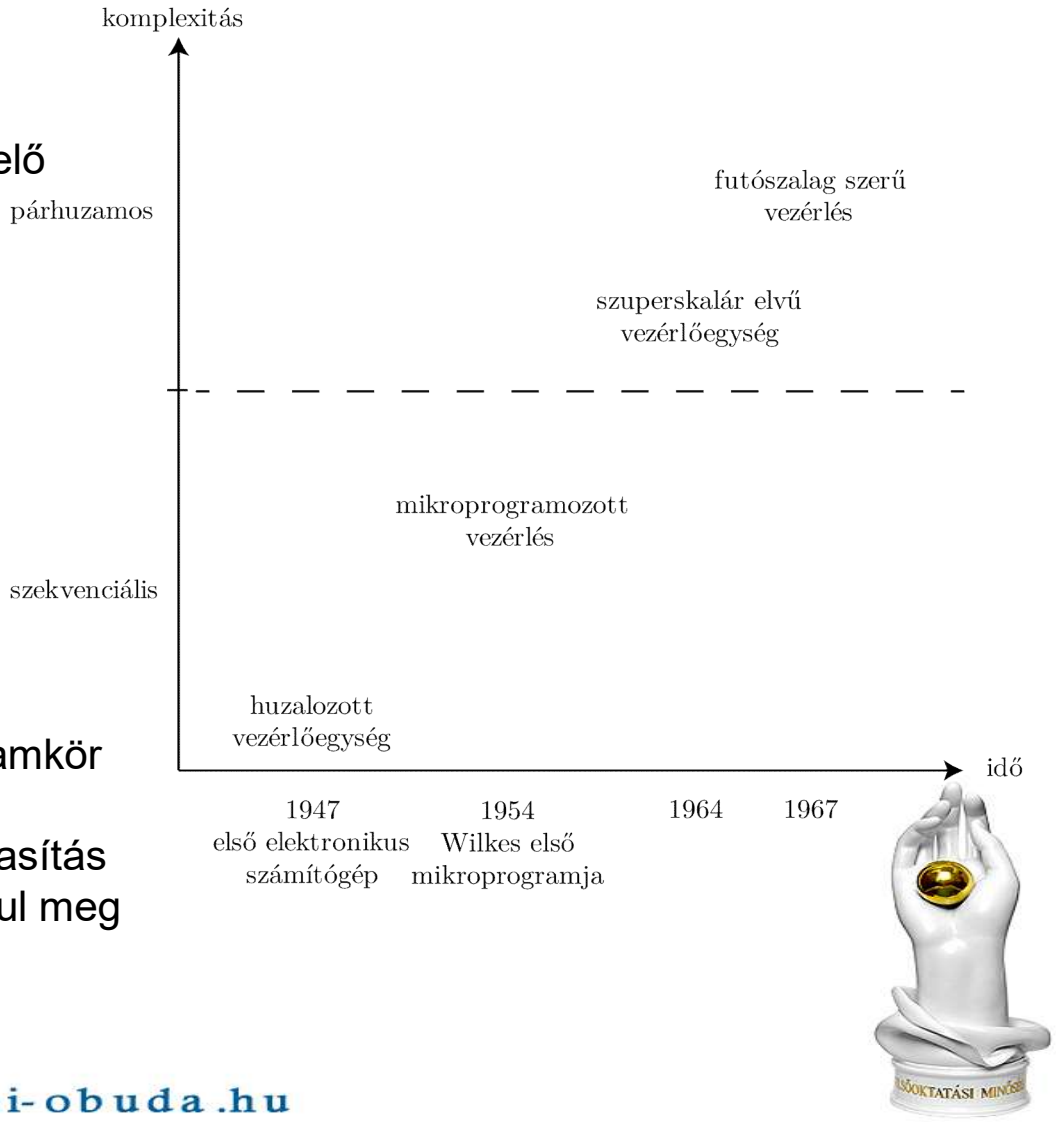
Tisztában kell lenni a késleltetésekkel!

Pl.: Regiszter ~ 1 nsec
L1 cache ~ 4-6 nsec
DRAM ~ 40-130 nsec

Két vezérlési elv:

a) Huzalozott vezérlés: szekvenciális logikai áramkör

b) Mikroprogramozott vezérlés: minden CPU utasítás egy vagy több mikroutasítás sorozattal valósul meg



Huzalozott vezérlő:

Előnye: gyors, mivel csak áramköri elemekből áll

Hátránya: - nehezen átlátható
- merev, nehezen módosítható

Tervezése hasonló módon történik, mint az ALU tervezése (csak sokkal bonyolultabb):

- igazságtábla felírása
- logikai függvények felírása
- azonos átalakítások:
  (áramköri elemek számának minimalizálása céljából → végrehajtási idő csökken)
- megvalósítás

Az ütemező feladata a vezérlőegység többi blokkjának vezérlése, valamint a vezérelt objektumok funkcionális egységeinek vezérlése

Vezérelt objektumok lehetnek:

ALU regiszterei, MAR, MDR, vezérlőbusz vezérlő regiszterei, I/O regiszterek, …

A vezérlés elve:

A forrásregiszter(ek) tartalmának egy módosító áramkörön keresztül a célregiszterbe való vezetése. Vezérlés feladata, hogy ehhez a kapukat nyissa és zárja.

A MAR-ral megcímezzük az operatív tárat,

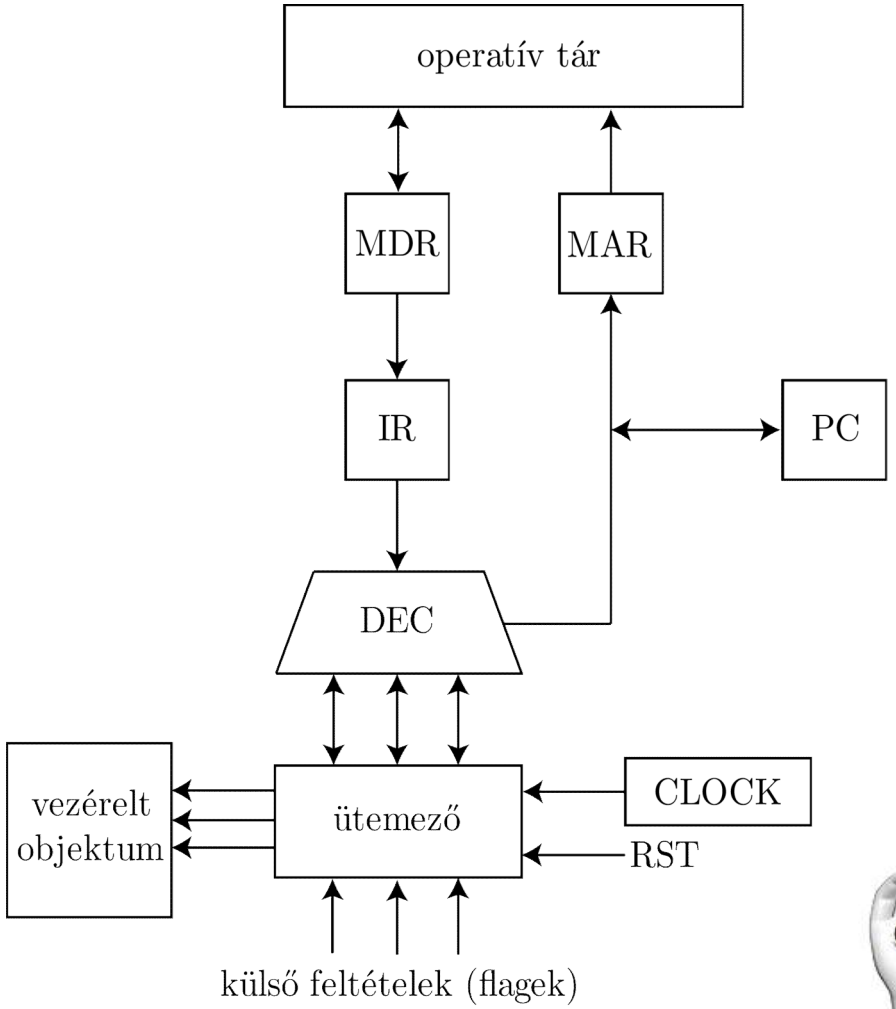
az operatív tárból betöltjük vagy visszaírjuk az adatot a MDR-be.

Onnan betöltjük az IR-be, majd a dekóderbe.

A dekóder a címrészből tudja megcímezni a MAR-t.

A dekóder adja le a jelet az ütemezőnek és az ütemező állítja elő a vezérlőjeleket akár a dekóder, akár más vezérelt egység részére.

A vezérlőjelek egyrészt a dekóder információja alapján, másrészt a külső feltételek alapján órajellel szinkronizálva állítódnak elő.



Mikroprogramozott vezérlés:

Célja: az ember számára áttekinthetővé tenni a vezérlést

Mikroprogramozott vezérlésről akkor beszélünk, amikor egy-egy gépi kódú utasítás végrehajtásának vezérlésére programozott vezérlőegységet használunk fel.

A vezérléshez használt program a mikroprogram, ezek utasítási a mikroutasítások.

A mikrovezérlő a mikroprogram végrehajtásakor a program utasításai alapján vezérlőjeleket ad ki, vagyis mikroutasítás aktivál egy specifikus vezérlő vonalkészletet.

A kimenet lesz a vezérlőjel.

Pl.: a mikroutasítások feladata az adatutak engedélyezése / tiltása

Régen voltak olyan gépek, melyek mikroprogramjai a RAM-ban voltak eltárolva (floppy-ról kellett betölteni)

Mai CPU-kban több ezer vezérlési pont van!

Programozása hasonlít az Assembly-re, csak még több hardver ismeretre van szükség

Előnye: rugalmasabb, áttekinthetőbb, olcsóbb, mikroprogram csere révén módosítható

Hátránya: mindig lassabb