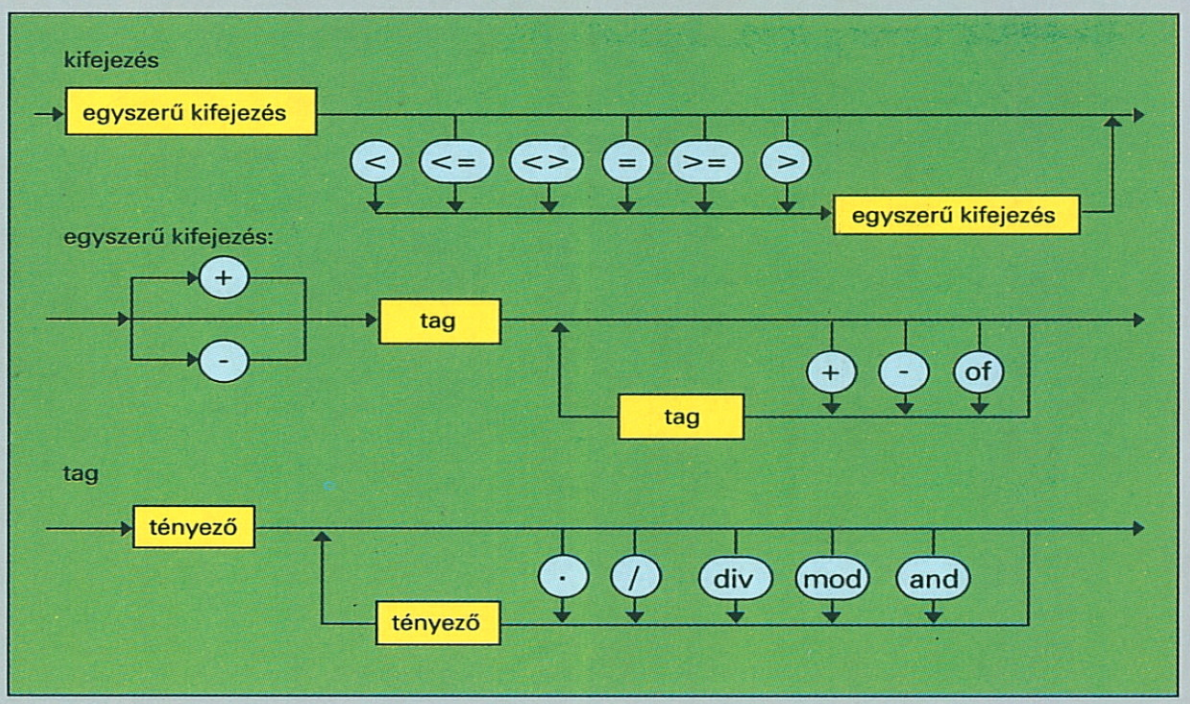
SH atlasz

Informatika

106 színes oldal, 1258 tárgyszó



Springer Hungarica

Az SH atlaszok kötetei:

Biológia 2., javított kiadás
Világtörténelem 3., javított kiadás
Csillagászat 2. kiadás
Építőművészet
Fizika
Filozófia
Matematika 2., javított kiadás
Ökológia 2., javított kiadás
Pszichológia
Élettan
Zene
Atomfizika
Kémia
Informatika
Anatómia I–III.
Űrtan

Hans Breuer

SH atlasz

Informatika

106 színes oldal, 1258 tárgyszó

Rajzolta:

Rosemarie Breuer

Springer-Verlag Budapest Berlin
Heidelberg New York Barcelona Hong Kong London Milan
Paris Santa Clara Singapore Tokyo 1995

Az eredeti mű:
dtv-Atlas zur Informatik

Fordította:
Ungvárai János
Ungvárainé dr. Nagy Zsuzsanna

A fordítást lektorálta:
Dr. Horváth Elek

BREUER, H.
Informatika / Hans Breuer ; rajzolta Rosemarie Breuer. – Budapest ; Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Santa Clara ; Singapore ; Tokyo ; Springer-Verlag, 1995. (SH atlasz)
ISBN 963 8455 70 5
ISBN 963 8455 63 2 össz.

Springer Hungarica Kiadó Kft.
Felelős kiadó: Prof. Dr. Árky István

Felelős szerkesztő: Füleki Lászlóné
Műszaki igazgató: Müllner János
Műszaki szerkesztő: Kellermann József
ITJ 66-12-2

Előszó

Az információs társadalom korában élünk, azonban az információáradat leküzdésére rendkívül nagy erőfeszítésekre van szükség. Az ezredfordulóra az informatika és alkalmazása válhat a legfontosabb tudománnyá – de ez a fordulat talán már be is következett.

Az SH atlasz jelen kötete az informatika alapjait tárgyalja: az információelméletet, a számrendszereket és a számábrázolást, a számítógépeket és programozásukat, beleértve a számítógépes grafikát is. Az informatika történetének és a számítógépek úttörőinek rövid áttekintésén kívül az olvasó a mesterséges intelligencia és a számítógép jövőjének kérdéseivel is megismerkedhet.

A kötet végén részletes tárgymutató található, ez megkönnyíti az anyagban való eligazodást. Az informatikai fogalmakra a legkorszerűbb szakkifejezéseket használjuk, de ezen a téren még sok minden kialakulatlan. A technika is szédítő iramban fejlődik, ami különösen a személyi számítógépek piacán szembetűnő: 18 hónaponként egy-egy új számítógép-generáció kerül le a futószalagokról. Világszerte már 140 millió PC üzemel.

Az SH atlasz Informatika azok számára készült, akik többet szeretnének tudni az emberiség információs korszakának tudományos és technikai alapjairól. Az anyag felépítése nemcsak arra alkalmas, hogy a könyvet pontról pontra átolvassuk, hanem lehetővé teszi az egyes fejezeteknek a többi résztől függetlenül történő önálló feldolgozását is.

Megköszönöm kedves Rosemarie-m türelmét és segítségét, amelyet nemcsak a rajzok elkészítése során tanúsított.

Köszönettel tartozom Klaus fiamnak is, aki informatikaszakos egyetemi hallgató.

Minden észrevételt, kritikát és megjegyzést szívesen fogadok, és minden ilyen jellegű írásra válaszolok.

Hans Breuer

Tartalom

Rövidítések

i. e.	időszámításunk előtt
ill.	illetve
i. sz.	időszámításunk szerint
kb.	körülbelül
o.	oldal
PC	személyi számítógép (personal computer)
pl.	például
stb.	és a többi
szül.	született
ún.	úgynevezett

Átszámítások

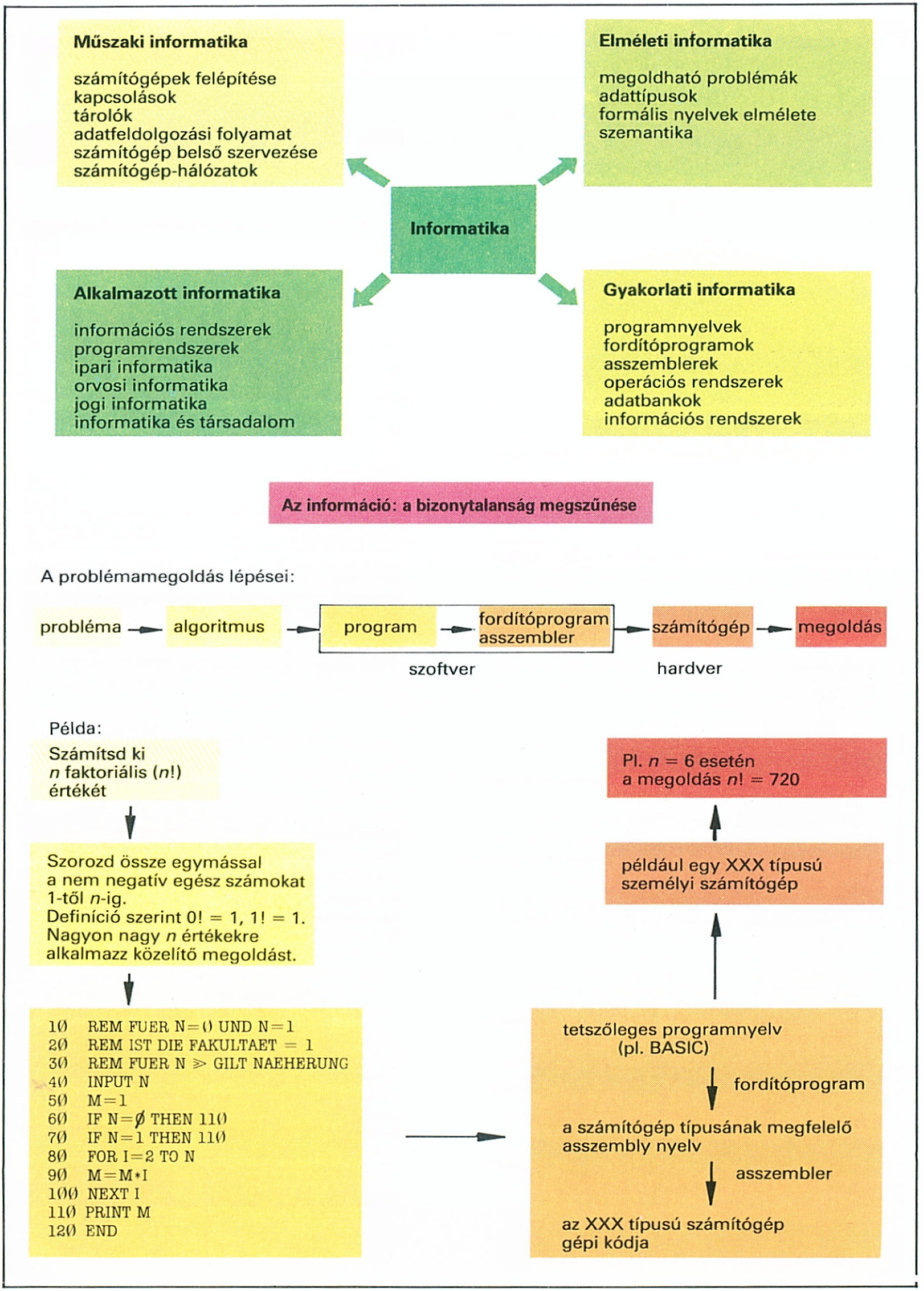
1000 bit = 1 kbit (kilobit)
1024 bit = 1 Kbit (K)
2^{20} bit = 1 Mbit (megabit)
2^{30} bit = 1 Gbit (gigabit) ill. 128 Mbyte
2^{40} bit = 1 Tbit (terabit) ill. 128 Gbyte

1000 byte = 1 kbyte
1024 byte = 1 Kbyte (KB)
2^{20} byte = 1 Mbyte (MB)
2^{30} byte = 1 Gbyte (GB)
2^{40} byte = 1 Tbyte (TB)

Tehát:
16 Kbyte = 16 384 byte = 2^{14} bit = kb. 16 kbyte
256 Kbyte = 262 144 bit = 2^{18} bit = kb. 262 kbyte

Rövidítések		Magyarázat
A/D	Analog/Digital	analóg/digitális
AI	Artificial Intelligence	mesterséges intelligencia
ALU	Arithmetic Logical Unit	a CPU aritmetikai egysége
ANSI	American National Standards Institute	Amerikai Nemzeti Szabványügyi Hivatal
APR	Arbeitsplatzrechner	munkahelyi számítógép
ASCII	American Standard Code for Information Interchange	7 bites kód (ASCII)
BCD	Binary Coded Decimal	binárisan kódolt decimális
BDOS	Basic Disc Operating System	alapoperációs rendszer
BDSG	Bundesdatenschutzgesetz	a német adatvédelmi törvény
BFD	Bundesbeauftragter für Datenschutz	a szövetségi adatvédelmi felügyelő
BIOS	Basic Input/Output System	alap be-/kimeneti rendszer
BPS, bps	Bits Per Second	bit per másodperc
BTX	Bildschirmtext	interaktív videotex
CAD	Computer-Aided Design	számítógéppel támogatott tervezés
CD-ROM	Compact Disc Read Only Memory	csak olvasható adattárolás CD lemezen
CGA	Color Graphics Adaptor	színes grafikus vezérlőkártya
CP/M	Control Program for Microcomputers	mikroszámítógépekre kifejlesztett operációs rendszer
CPS	Characters Per Second	karakter per másodperc
CPU	Central Processing Unit	központi feldolgozóegység
CR	Carriage Return	„kocsi vissza" (nyomtatókon)
CRAM	Card Random Access Memory	mágneskártyás adattároló
D/A	Digital/Analog	digitális/analóg
DATEL	Data Telecommunication	németországi speciális adathálózat
DB	Data Base	adatbázis
DBASE		egy adatbázis-kezelő rendszer
DD	Double Density	kétszeres írássűrűség
DFV	Datenfernverarbeitung	táv-adatfeldolgozás
DIN	Deutsches Institut für Normung	Német Szabványügyi Hivatal
DOS	Disc Operating System	PC operációs rendszer
DPI, dpi	Dots Per Inch	a nyomtatási sűrűség mértéke
DV	Datenverarbeitung	adatfeldolgozás
DVA	Datenverarbeitungsanlage	adatfeldolgozó berendezés
E/A	Eingabe/Ausgabe	(adat)bevitel/kivitel
EAPROM	Electrically Alterable PROM	elektromosan írható PROM
EBCDIC	Extended Binary Coded Decimal Interchange Code	kiterjesztett BCD kód
EDV	Elektronische Datenverarbeitung	elektronikus adatfeldolgozás
EGA	Enhanced Graphics Adapter	színes grafikus vezérlőkártya
EMS	Electronic Mail System	elektronikus levelező rendszer
EPROM	Erasable Programmable Read Only Memory	törölhető és újraírható PROM
EVA	Eingabe, Verarbeitung, Ausgabe	adatbevitel, adatfeldolgozás, adatkivitel
FIFO	First In First Out	adattárolási struktúra (az elsőnek beírt adatot lehet először kiolvasni)
FLOPS	Floating Point Operations Per Second	mp-ként végrehajtott lebegőpontos művelet
HD	High Density	nagy írássűrűség (mágneslemezen)

IMS	Information Management System	adatbázis rendszer
I/O	Input/Output	bemenet/kimenet
ISDN	Integrated Services Digital Network	digitálisan működő adathálózat
KI	Künstliche Intelligence	mesterséges intelligencia
KIPS, kips	Kilo Instructions Per Second	ezer utasítás per másodperc
LCD	Liquid Crystal Display	folyadékkristályos kijelző
LED	Light Emitting Diode	világító dióda
LIFO	Last In First Out	spec. adattárolási strukt. (verem szerk.)
LPS, lps	Lines Per Second	másodpercenként kinyomtatott sorok sz.
MIMD	Multiple Instruction Stream, Multiple Data Stream	többszörös utasításfolyam többszörös adatfolyam
MIPS, mips	Mega Instructions Per Second	millió utasítás per másodperc
MISD	Multiple Instruction Stream, Single Data Stream	többszörös utasításfolyam egyszeres adatfolyam
MOS	Metal Oxide Semiconductor	integrált áramkör típus
MS-DOS	Microsoft Disk Operating System	PC operációs rendszer
MTBF	Mean Time Between Failures	a sz. gép megbízhatóságának mértéke
NC	Numerical Control	számjegyvezérlés
NOS	Network Operating System	számítógéphálózat-operációs rendszer
NUA	Network User Address	felhasználó azonosító száma (Datex-P)
NUI	Network User Identity	Datex-P hálózaton a felhasználó jelszava
OCR	Optical Character Recognition	optikai karakterfelismerés
OROM	Optical Read Only Memory	optikai ROM
OS	Operating System	operációs rendszer
PC	Personal Computer	PC, személyi számítógép
PCB	Printed Circuit Board	nyomtatott áromkört tartalmazó kártya, (NYÁK-lap)
PIS	Personal Information System	személyi információs rendszer
PIXEL	Picture element	képelem
PROM	Programmable Rad Only Memory	programozható ROM
RAM	Random Access Memory	tetszőleges hozzáférésű memória
ROM	Read Only Memory	csak olvasható memória
RTL	Resistor-Transistor Logic	RTL logikai áramkör
SAM	Sequential Access Method	szekvenciális hozzáférési módszer
SD	Single Density	egyszeres írássűrűség
SIMD	Single Instruction Stream, Multiple Data Stream	egyszeres utasításfolyam többszörös adatfolyam
SISD	Single Instruction Stream, Single Data Stream	egyszeres utasításfolyam egyszeres adatfolyam
SOB	Start Of Data Block	adatblokk kezdete
TP	Teleprocessing	táv-adatfeldolgozás
TTL	Transistor-Transistor Logic	TTL logikai áramkör
V.24		szabványosított illesztőegység
VGA	Video Graphics Adapter	grafikus vezérlőkártya
VLSI	Very Large Scale Integration	nagy bonyolultságú integrálás (mikrochip)
ZE	Zentraleinheit	központi egység



Informatika, problémamegoldás

Az **informatika** *(computer science)* az információ rendszeres és automatikus – elsősorban számítógépek segítségével történő – feldolgozásával és továbbításával foglalkozó tudomány.

Az informatika más tudományágak (matematika, logika, elektrotechnika, elektronika) részterületeiből fejlődött ki. A hatvanas évek óta önálló szakterületnek számít, amelyet időközben az alaptudományokhoz sorolnak. Az egyetemeken és főiskolákon külön szakot létesítettek az „informatikus" hallgatók három- és ötéves képzésére.

Az **elméleti informatika** a matematikából ismert módszereket és modelleket használja fel algoritmusok készítéséhez és számítógépek tervezéséhez. A formális nyelvek, az automaták, az adattípusok elméletével, a szemantikával stb. foglalkozik.

A **gyakorlati informatika** módszereket dolgoz ki az algoritmusoknak a számítógépek számára megfelelő formában történő felírásához. Elsősorban a programnyelvek, a fordítóprogramok és az operációs rendszerek kifejlesztésével foglalkozik. Ezenkívül ide tartozik azoknak a módszereknek a kidolgozása is, amelyek lehetővé teszik a nagyterjedelmű adattömegek adatbankokban történő összegyűjtését és az ezekhez való hozzáférést.

A **műszaki informatika** a számítógépek és számítógéprendszerek funkcionális felépítésével, elektronikus kapcsolásaival és szervezésével, elsősorban a hardver kérdéseivel foglalkozik.

Az **alkalmazott informatika** mindenekelőtt a számítógéppel megoldható feladatok témakörét vizsgája, valamint azt, hogy ezeket milyen egyéb szakterületeken lehet felhasználni. Egyszerű programnyelveket fejleszt ki pl. az adatbankok kezelésének megkönnyítésére.

Az alkalmazott informatika az informatikának olyan részterülete, amely az információs adattömegeknek a társadalomra és az egyes egyénre gyakorolt hatását vizsgálja; ide tartoznak például az adatvédelem kérdései.

Alapfogalmak

Az **algoritmusok** valamilyen formában leírható és lépésről lépésre végrehajtható eljárások az adott problémák megoldására. Általános megoldási módszereket jelentenek. Ugyanaz a probléma különböző algoritmusokkal is megoldható.

A programnyelvek (és ezek fordításai) olyan algoritmusokat fogalmaznak meg, amelyeket a számítógép közvetlenül tud végrehajtani.

A **program** egyértelmű szabályok segítségével felírt algoritmus, amely ebben az alakban egy meghatározott számítógépbe táplálható és azon lefuttatható, végrehajtható. A program általában azt is rögzíti, hogy a programhoz esetleg hozzá tartozó adatokat milyen alakban kell előállítani. A különböző programnyelvek, pl. ALGOL, FORTRAN, PASCAL vagy BASIC megkönnyítik a programok írását.

Az **információ** ismeretnyereséget, az ismeretanyag növekedését, ill. a bizonytalanság csökkenését jelenti. Szigorúan véve azonban az információ független egy üzenet tartalmának jelentésétől. Az információ az energia és az anyag mellett a tudomány és a technika három alapvető fogalmának egyike. Egy adott üzenet által tartalmazott **információ mennyiségének** az üzenet **információtartalmának**) egysége a **bit** (információs bit, angolul: **b***asic indissouble* **i***nformation uni***t**).

Példa: A német nyelvben egy betű információtartalma kb. 1,6 bit. Az információtartalom készülékkel nem mérhető, csak számítható.

Az információtartalom csak számítással határozható meg, műszerrel nem mérhető.

A kettes számrendszerben ábrázolt adatok alapegységét szintén **bit**nek nevezzük (jelbit, angolul: **bi***nary digi***t**). A kettes számrendszer egy-egy számjegye (0 vagy 1) 1 bitet jelent. A bit és a byte (=8 bit) elsősorban a tárolók kapacitásának jellemzésére szolgál.

Az információtartalom egysége és a kettes számrendszer alapegysége között összefüggés állapítható meg: Egy n bit formális információtartalmú üzenet számítógépben történő ábrázolásához legalább n bit (jelbit) szükséges.

Az **információátvitel** (vagy **információáramlás**) **sebessége** *(information rate)* az időegység alatt átvitt információtartalmat jelenti; egysége bit/s.

Példa: Olvasáskor az információátvitel átlagos sebessége kb. 50 bit/s.

Az **üzenet** diszkrét jelek vagy állapotok olyan véges sorozata, amely információ átvitelére szolgál. Tartalmazhat információt, de ez nem szükségszerű. Az üzenet fogalmát gyakran az információval azonos értelemben használják.

A **számítógép** *(computer)* az előzetesen algoritmusok alakjában megfogalmazott általános problémák önálló megoldására alkalmas készülék. A számítógép a hardver és szoftver kombinációjának is tekinthető.

Hardvernek *(hardware)* nevezzük a számítógép valamennyi műszaki berendezését és alkotóelemét.

Példák: Mikroprocesszor, tárolóchipek, be- és kimeneti egységek, vezetékek.

A szoftverrel ellentétben a hardver (viszonylag) állandó, változatlan alkotórészeket jelent.

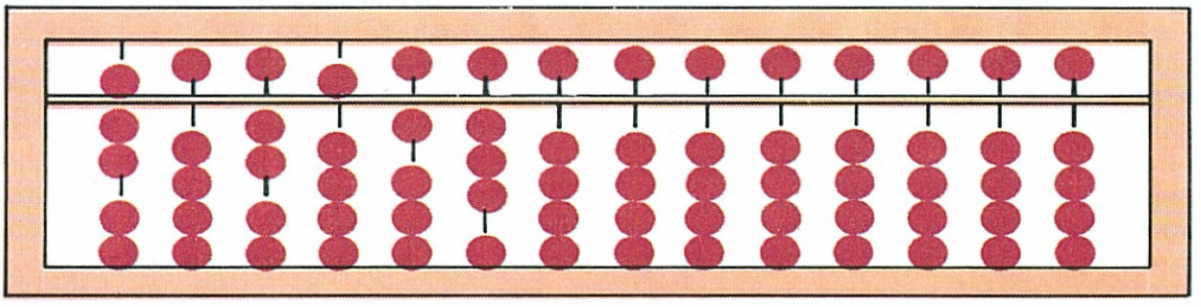
Szoftvernek *(software)* nevezzük a számítógép működéséhez rendelkezésre álló valamennyi programot. Ide tartoznak a számítógép felhasználójának saját programjain kívül azok a programok is, amelyek magát a számítógépet működtetik és egy-egy meghatározott programot a számítógép számára végrehajthatóvá tesznek. A szoftvert könnyű megváltoztatni és másolni.

A **rendszerszoftver** (alapszoftver) a számítógép egészének működéséhez szükséges összes programot jelenti, ilyenek pl. az operációs rendszerek, a fordítóprogramok, és a hibakereső programok.

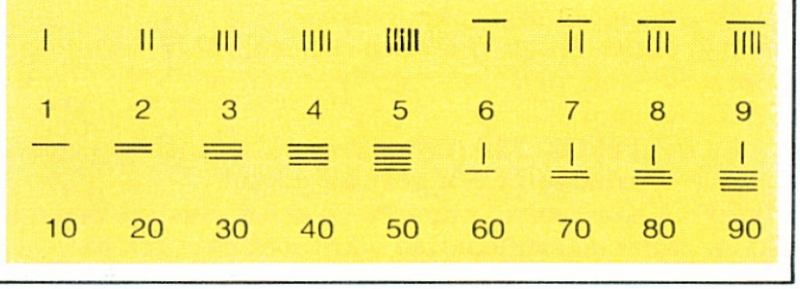
A **felhasználói szoftver** azokat a programokat jelenti, amelyeket a felhasználók egy-egy adott probléma megoldására alkalmaznak, ide tartoznak pl. a szövegszerkesztő programok.

számrendszer	jelkészlet	alap	példa	helyiértékrendszer
decimális	0 1 2 3 4 5 6 7 8 9	10	1993 = 1000 + 900 + 90 + 3 = 1×10^3 + 9×10^2 + 9×10^1 + 3×10^0 = 1 9 9 3	
bináris	0 1	2	350 = 1×2^8 + 0×2^7 + 1×2^6 + 0×2^5 + 1×2^4 = 1 0 1 0 1 = 1×2^3 + 1×2^2 + 1×2^1 + 0×2^0 = 1 1 1 0	
hexadecimális	0 1 2 3 4 5 6 7 8 9 A B C D E F	16	1993 = 7×16^2 + 12×16^1 + 9×16^0 = 7 C 9	
babilóniai	▼ ❮	60	16468 = 14400 + 2040 + 28 = 4×60^2 + 34×60^1 + 28×60^0 =	
római	I V X L C D M		1992 = 1000 + (−100 + 1000) + (−10 + 100) + 2 M C M X C II	

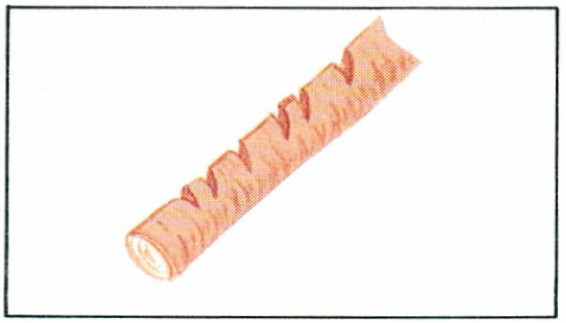
Különböző számrendszerek



A 702 513 szám ábrázolása a 4+1 felépítésű abakuszon



Kínai pálcikaszámjegyek



Rovásos pálca

7 x 8 = 56

	1	2	3	4	5	6	7	8	9	10
	2	4	6	8	10	12	14	16	18	20
	3	6	9	12	15	18	21	24	27	30
	4	8	12	16	20	24	28	32	36	40
	5	10	15	20	25	30	35	40	45	50
	6	12	18	24	30	36	42	48	54	60
	7	14	21	28	35	42	49	56	63	70
	8	16	24	32	40	48	56	64	72	80
	9	18	27	36	45	54	63	72	81	90

A Püthagorasz-féle számolódeszka

A számlálás az ember legkorábbi elvont, intellektuális képességeinek egyike. Ehhez az első segédeszközt a kéz ujjai jelentették, később kövek, fadarabok, pálcákba vésett rovások, zsinegre kötött csomók szolgáltak a számok tárolására, pl. az adósság rögzítésére.

Ezek az eszközök az összeszámlálás ill. kisebb számok kivonása során elegendő segítséget nyújtottak. Gyakorlati szempontból könnyen érthető azonban, hogy gyorsan ki kellett fejlődnie olyan számrendszereknek, amelyek nagyobb számok egyszerű ábrázolását is lehetővé tették. A szorzás és az osztás csak így vált lehetségessé.

Számrendszerek

A számrendszerek az egyes számjegyeket úgy rendezik el, hogy viszonylag kevés számjegy nagy számokat is ábrázolni tudjon. Különböző számrendszerek fejlődtek ki, annak ellenére, hogy az emberi kéz ujjainak száma miatt a tízes számrendszer lett volna legkézenfekvőbb.

A helyiértékes számrendszerekben az egyes számjegyek értéke attól függ, hogy a számon belül melyik helyen találhatók. A ma használatos helyiértékes számrendszerekben a számjegyek értéke jobbról balra haladva nő.

Másfajta számrendszerre példa a római számok rendszere: XVIII = 18, XIX = 19, XX = 20, XXI = 21.

A **tízes alapú (tízes) számrendszer** tíz számjegyet alkalmaz. A számjegykészletnél nagyobb számokat tíz többszöröseiként helyiértékes rendszerben ábrázolja. Minden tízes számrendszerbeli szám tíz hatványainak összegeként állítható elő.

Példa: Az 1992-es szám a tízes számrendszerben $1 \times 10^3 + 9 \times 10^2 + 9 \times 10^1 + 2 \times 10^0 = 1992$ alakban írható fel.

A **kettes számrendszer** kettő alapú helyiértékek szerinti számrendszer, melynek két számjegye a 0 és az 1 (ezeket néha Ø és L szimbólumokkal is jelölik). Minden kettes számrendszerbeli szám *(bináris szám)* kettő alapú hatványok összege.

Példa: A tízes számrendszerben felírt 350 kettes számrendszerbeli alakja: 101011110.

A kettes számrendszert Gottfried Wilhelm Leibniz fejlesztette ki a 17. század vége felé. Ez a digitális adatfeldolgozás alapja, mivel a kettes számrendszer két számjegye megfeleltethető a kapcsoló- és tárolóelemek lehetséges *be/ki* illetve *igen/nem* állapotainak.

A **hexadecimális számrendszer (tizenhatos számrendszer)** 16 alapú helyiértékek szerinti számrendszer, amely 16 számot illetve jelet tartalmaz: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E és F.

Példa: A tízes számrendszerbeli 1992-es szám hexadecimális számrendszerbeli alakja: 7C8.

A kevesebb jegyből álló tömörebb ábrázolási lehetőség miatt a regiszterek és a tárolóelemek tartalmát legtöbbször tizenhatos számrendszerben ábrázolják.

Számrendszerek további leírása még az 57. o.-n.

A **tucat** (12 kifejezés egy régen használatos tizenkettes számrendszer maradványa. A szög hagyományos mértékegységei ($1° = 60' = 3600''$), valamint az idő mértékegységei (1 h = 60 min, 1 min = 60 s) egy régi babilóniai hatvanas számrendszerből erednek.

Számolási segédeszközök

Az ember első számolási segédeszközei az ujjai voltak és ez a funkciójuk a számolást tanulók körében mind a mai napig megmaradt. Emellett egyéb szemléletes segédeszközök is kifejlődtek.

A **számolólécek** megkönnyítik a tíznél nagyobb számok összeadását és kivonását. A legrégebbi kínai számjelek még jól felismerhetően erre a segédeszközre utalnak.

Régen a **számolópad (számolódeszka)** sorokra és oszlopokra beosztott felületén „állította ki a számlát" az eladó a vevőnek. A számolódeszka minden sora a *számolópfenniggel* megjelölt szám oszlopának egy-egy sokszorosát jelölte. Útjaikra a kereskedők számolódeszka helyett ugyanígy működő *számolókendőt* vittek magukkal.

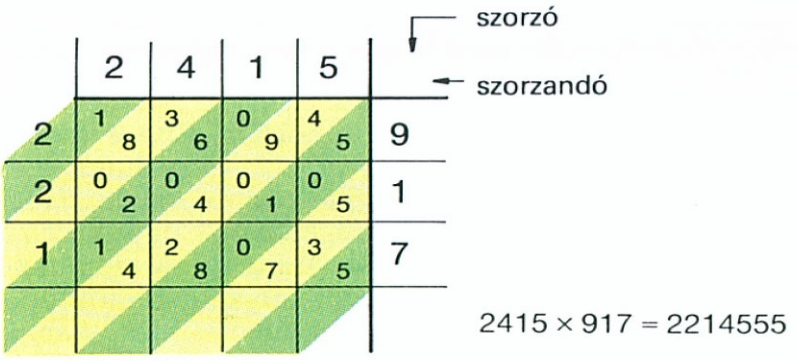
Az **abakusz** vagy **szorobán** ázsiai eredetű egyszerű számológép, amely rudakon, drótokon vagy hornyokon ide-oda mozgatható számológolyókat tartalmaz. A golyók helyzete egy-egy rúdon számjegyeket, minden rúd pedig egy-egy helyiértéket jelent. Az abakusz már a régi keleti kultúrák idején ismert volt. Ázsiában még századunk hetvenes éveiben is használták.

Az abakusz legegyszerűbb változatában minden rúdon tíz golyó található. A golyóknak az az elrendezése, amelyben az egy rúdon levő valamennyi golyó szorosan a jobb oldalon helyezkedik el, az illető rúd által jelentett helyiértéken 0 számjegyet jelöl. Ha valamelyik rúdon n golyót bal oldali ütközésig elhúzunk, akkor ez az elrendezés a rúd által jelentett helyiértéken n számjegyet jelöl. Egy hatsoros (hat rudat tartalmazó) abakuszon tehát a legnagyobb ábrázolható szám 999999.

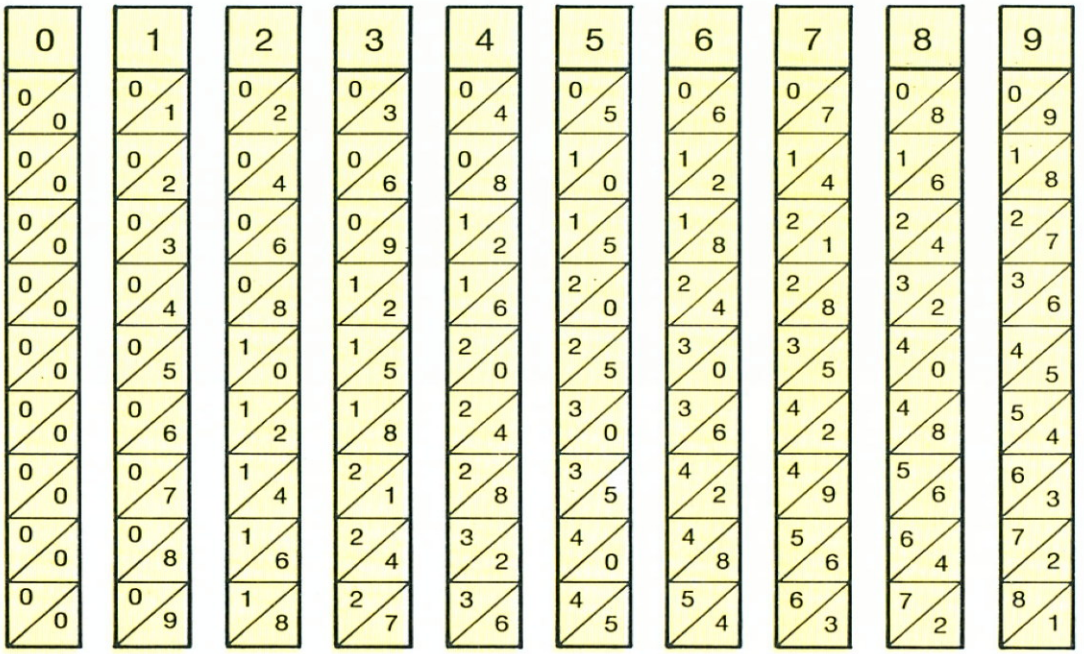
Ha az abakusz elválasztólécet tartalmaz, akkor minden rúd elválasztóléc alatti részén öt (egyeseket jelölő), a felső részén pedig kettő (ötösöket jelölő) golyó elegendő a számok ábrázolásához. Ennél is kevesebb golyót tartalmaz a (4 + 1)-es abakusz, amely azt használja ki, hogy minden számjegy ábrázolásához csak 1 ötösöket és 4 egyeseket jelölő golyóra van szükség, a kerethez ütköző golyó pedig mindig nullahelyzetet jelent. Az összeadás és kivonás az abakusszal egyszerűen és gyorsan elvégezhető. A szorzás és osztás több gyakorlatot és időt igényel.

Számtankönyveiben Adam Ries (1492–1559) még leírta az abakusz („a vonalon számolás") híveinek és az algoritmus („a tollal számolás") híveinek egymással szemben álló és versengő számolási módszereit.

A **Püthagorasz-féle számolódeszka** az ókori Görögországban kifejlesztett számolási segédeszköz. A gyakran szükséges számításokat egy-egy táblázatban foglalta össze, amelyből az eredményeket egyszerűen leolvasták. A Püthagorasz-féle számolódeszka a matematikai táblázatok ősének tekinthető.

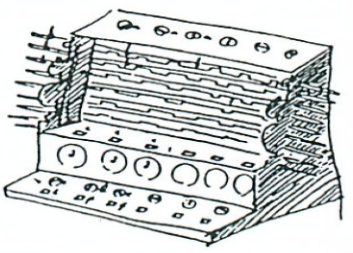


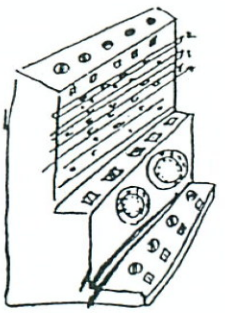
Gelosia-féle szorzás

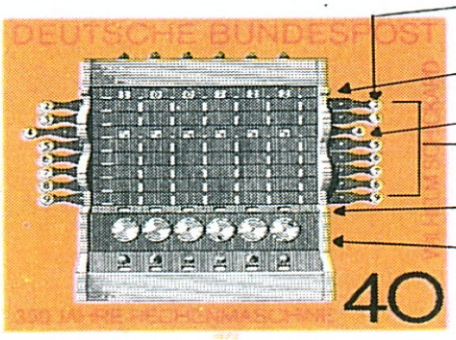


Neper-féle pálcakészlet

Schickardnak Keplerhez küldött vázlatai:







Schickard számológépe

A szorzás és az osztás lényegesen nehezebben kivitelezhető műveletek, mint az összeadás és a kivonás. Elsősorban a részeredmények átvitele során léphetnek fel hibák. Ezért a szorzás és az osztás elvégzéséhez különösen szívesen alkalmaztak számítási segédeszközöket.

A **gelosia-módszer** (rácsos módszer) a középkor kezdete óta széles körben elterjedt. Először Indiában, Perzsiában, Kínában és az arab kultúrkörben jelent meg. Európában a 14. század elején vált ismertté.

A név a korai olasz építészet geometrikus, osztott rácsos ablakkereteinek nevéből származik.

Függőleges és vízszintes vonalak egy mátrixot képeznek. Minden mezőt egy átló két részre oszt. A szorzáshoz a szorzandó számjegyeit oszlopról oszlopra haladva a legfölső sorba, a szorzó számjegyeit pedig a legkülső oszlop egymás alatti soraiba írjuk. A felhasználó a mátrix minden egyes mezejét a hozzátartozó oszlop és sor szorzatát jelentő két számmal tölti ki. Az egyeseket az alsó, a tízeseket a felső háromszögbe írja. A teljes szorzatot úgy kapja meg, hogy a jobb alsó saroktól kezdve a bal felső sarok felé haladva összeadja a felrajzolt, egymással szomszédos átlókon elhelyezkedő számokat. Ha valamelyik átlós összeg kétjegyű szám, annak első jegyét a felette álló átlós összeghez adja. Ilyen módon jobbról balra haladva számjegyenként megkapja a végeredményt. Az eljárás nemcsak szorzásra, hanem osztásra is alkalmazható, de nagyon sok írást igényel.

John Napier, latinosan **Neper** (1550–1617) skót tudós és matematikus számolópálcákkal egyszerűsítette a rácsos módszert.

A pálcákat Neper-pálcáknak vagy Neper-csontoknak is nevezik, mivel a tartósabb darabok készítéséhez Neper-csontot használt.

Minden pálca a rácsos (gelosia-) mátrix egy lehetséges oszlopát, tehát egy számjegy egész számú többszöröseit ábrázolja. Ha pl. 987 és 123 szorzatát akarjuk kiszámítani, akkor a felső sorban 9-et, 8-at és 7-et tartalmazó pálcát egymás mellé kell helyezni, és ezután a rácsos módszert kell alkalmazni.

A számolópálcákat a misszionáriusok még Napier életében egészen Kínáig elterjesztették. Napier kortársa, a jezsuita **Gaspard Schott** henger alakú számolópálcákat esztergált, amelyek felületére oszlopokban egy teljes sorozat Neper-pálca összes számjegyét bevéste. Több ilyen hengeres pálcát egy keretben forgatható módon egymás mellé helyezett, a keretet nyílással ellátott fedővel zárta le, hogy azon keresztül az egyes pálcahelyzetek láthatók legyenek. Ez a készülék azonban nem terjedt el a gyakorlatban.

1885-ben **Henri Genaille** a Neper-pálcák különleges alakításával és elhelyezésével megoldotta a szorzásnál fellépő kétjegyű összeadandók átvitelének problémáját is. A széles körű elterjedéshez azonban ez már túl késő volt.

Schott forgatható Neper-pálcái az abakuszt mint számítási segédeszközt alig múlták felül.

Az első számológépet kevéssel később készítette el egy tübingeni professzor és sokoldalú zseni:

Wilhelm Schickard (1592–1635) matematikus, csillagász, műszerész, festő, rézmetsző, orientalista és pap 1623-ban Neper-pálcák segítségével a négy alapművelet elvégzésére képes számológépet épített. Külön mechanizmus szolgált a szorzás és az osztás során fellépő részeredmények tárolására és hozzáadására.

A készülék összeállítására valószínűleg a híres csillagásszal, Johannes Keplerrel folytatott beszélgetései ösztönözték. A számológépről Keplernek pontos rajzot küldött. Számológépének másodpéldánya, amelyet Schickard Keplernek ígért, még a műhelyben elégett.

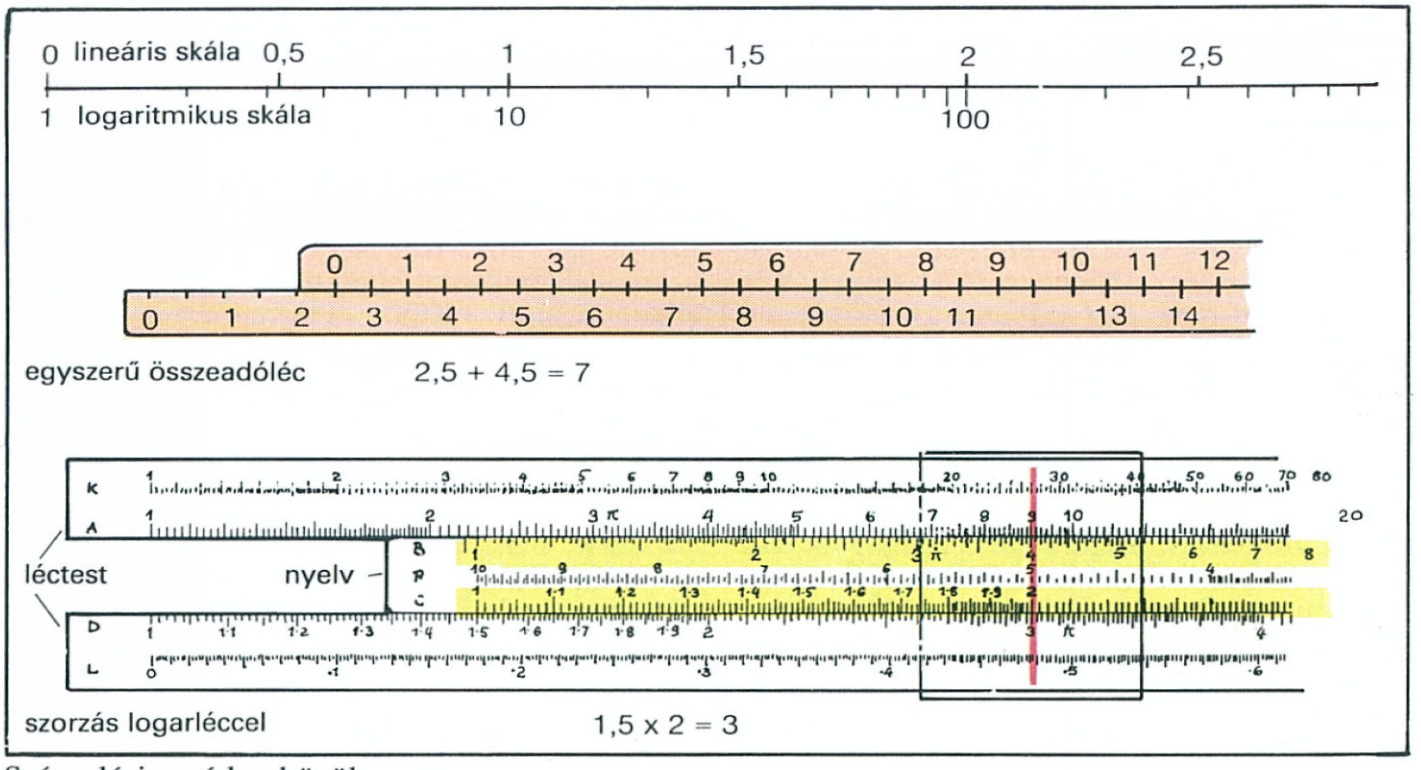
Maga az eredeti számológép a harmincéves háború zűrzavarában eltűnt, viszont Kepler iratai között találtak róla egy vázlatot. Ennek alapján 1960-ban sikerült egy jól működő rekonstrukciót készíteni:

A számológép felső része függőlegesen elrendezett, hengeres Neper-pálcákat tartalmaz. Forgatható gombokkal ezeken (legfeljebb) hatjegyű számokat lehet beállítani. Alatta fogaskerekekből készített számlálómű található. A felhasználó a Neper-pálcákról leolvasott részeredményeket kézzel viszi át a számlálóműbe. A végeredmény a készülék alján levő kis nyílásokban jelenik meg. A gép alapzatában elhelyezett számlapok hatjegyű számok tárolását teszik lehetővé, valószínűleg azért, hogy a felhasználónak a részeredményeket ne kelljen külön leírnia.

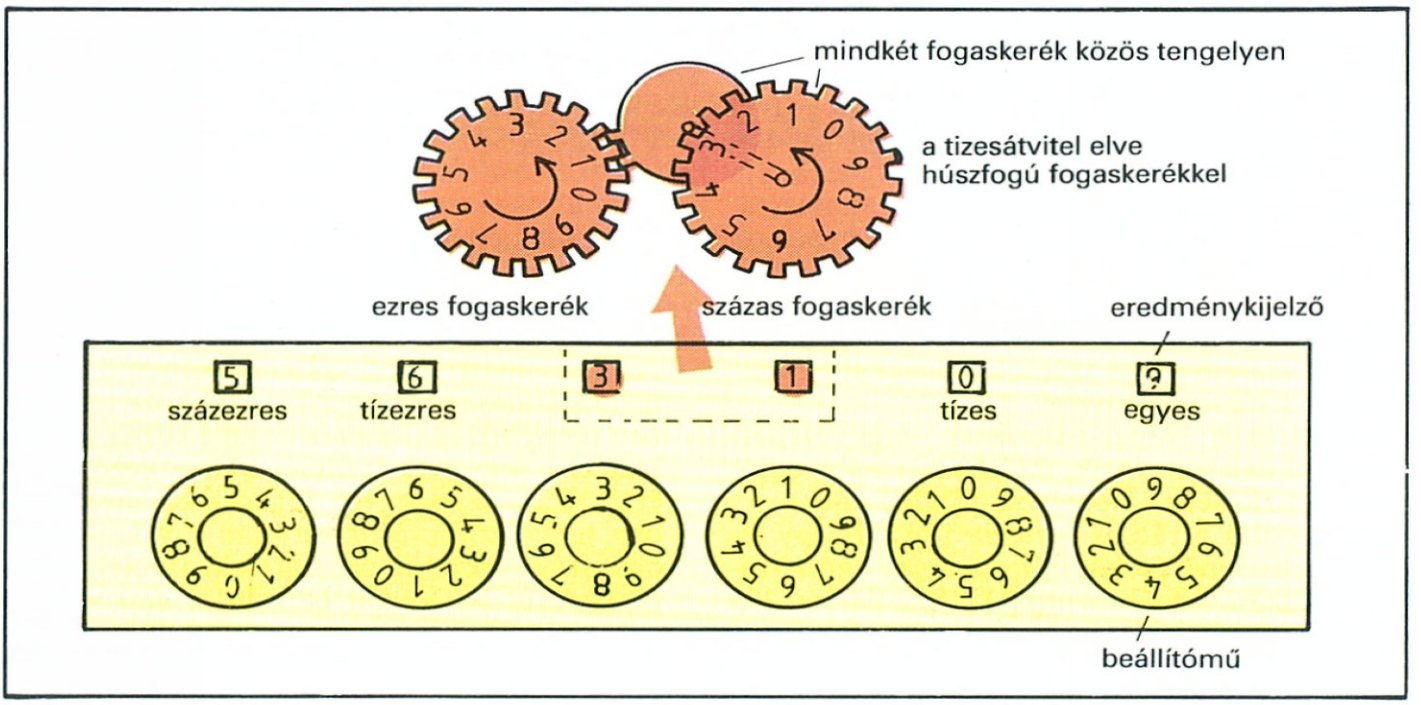
Az összeadást végző számlálóműben a szomszédos számjegyek fogaskerekei közé elhelyezett járulékos fogaskerék végzi a kétjegyű összeg első jegyének átvitelét a következő helyiértékre. A számjegyek fogaskerekeinek minden teljes körbefordítása után egy külön beépített, egyedi fog a megfelelő járulékos fogaskereket 36 fokkal elfordítja, ami viszont a következő számjegy fogaskerekét az eggyel magasabb számértéknek megfelelő helyzetbe fordítja tovább.

Schickard bölcs belátással hat jegyre korlátozta számológépét. A határesetben, pl. 999999 + 1 összeadásakor ugyanis az egyes számjegyhez kapcsolódó egyetlen fogaskerékfognak kellene a teljes számolóművet mechanikus úton átfordítania. Schickard ezen a ponton elérte a korabeli mechanika lehetőségeinek határát. Kepler számításaihoz viszont hat jegynél többre volt szükség. Ezért a tübingeni mester gyűrűket készített, amelyekről a csillagásznak minden alkalommal egyet másik helyre kellett volna húznia, valahányszor a készülékbe beépített csengő jelzi, hogy a gép a 7. helyiértékre próbál lépni.

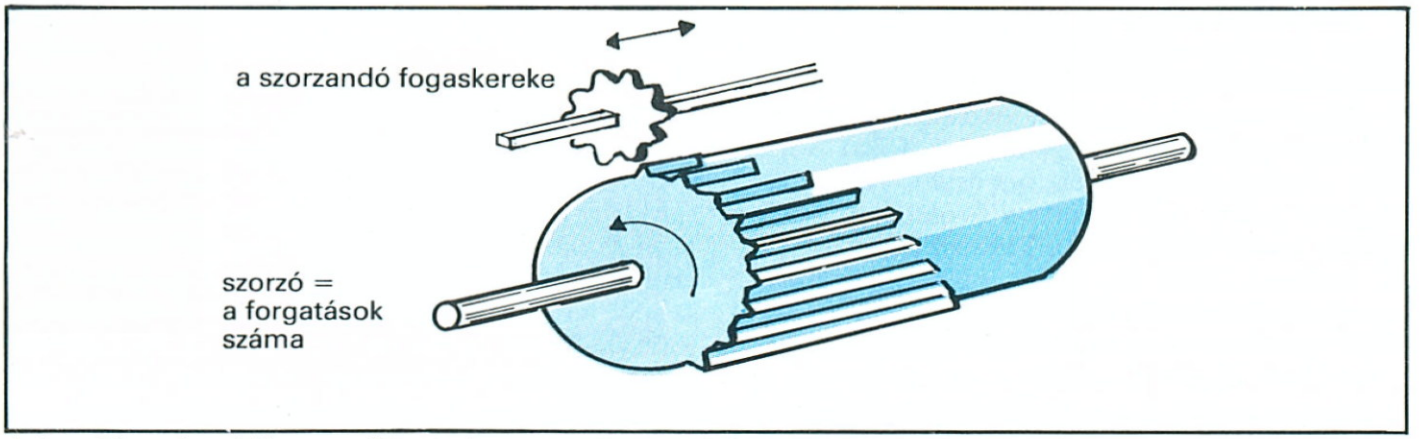
Schickard gépe volt az első, amely a számolási tartomány túllépését (a túlcsordulást, angolul *overflow*) is jelezte a felhasználónak.



Számolási segédeszközök



A „Pascaline" felépítése



A bordás számolóhenger közvetlen szorzást tesz lehetővé

Számolóléc, logarléc

A 16. század vége felé az egyik legjelentősebb tudományos teljesítmény a **logaritmus** feltalálása volt. JOST BÜRGI (1552–1632) 1588-ban, JOHN NAPIER pedig 1594-ben egymástól függetlenül összeállította az első logaritmustáblázatot. HENRY BRIGGS (1561–1630) pedig 1615-ben bevezette a tízes alapú logaritmust.

Ettől kezdve a szorzás és az osztás korábban olyan fáradságos műveletét az összeadás és a kivonás váltotta fel, mivel:

$\lg(a \cdot b) = \lg a + \lg b$ és

$\lg(a/b) = \lg a - \lg b$.

A számok logaritmusát az egyre pontosabbá váló logaritmustáblázatokból keresték ki.

Ekkor mutatkoztak meg igazán a számegyenesen való számolás előnyei. Két szám összeadása ugyanis két megfelelő hosszúságú szakasz egymás mellé illesztésével modellezhető. A két szám összegét ekkor a két szakasz hosszának összegeként lehet kiszámítani. A kivonás úgy történik, hogy a két szakaszt a számegyenesen kivonjuk egymásból. Egymáshoz képest eltolható, azonos skálabeosztású vonalzókat már jóval BÜRGI és NAPIER előtt is használtak összeadásra és kivonásra. Elsőként 1622-ben WILLIAM OUGHTRED (1574–1660) alkalmazott logaritmikus beosztást a vonalzókon és a logaritmikus beosztású vonalzókat csúsztatta el egymáson. A két szakasz összeadása így a két szakasz beosztásával jelképezett szám szorzásának felelt meg. Hasonlóan egyszerűvé vált az osztás is. 1650 körül alakult ki a logarléc mai formája. Egy skálabeosztással és számokkal ellátott „nyelv" csúszik az ugyancsak skálával és számokkal ellátott „léctestben".

Egyéb skálabeosztásokat is készítettek pl. a hatványozás, a gyökvonás, a szögfüggvények és a reciprok értékek leolvasására. Több skálabeosztás egyidejű használatát tette lehetővé a „logarléctesten" elcsúsztatható ablak, amelyet 1851-ben vezettek be. Számos, különleges célokra alkalmas logarléc is készült.

A logarléc – az első modern analóg számítógép – használata a számjegyekkel történő számolással szemben két lényeges hátránnyal jár:

1. A gyakorlatban a skálahosszúság határozza meg a számítás pontosságát.

   A szokásos, 25 cm-es logarlécekkel legfeljebb 0,1% pontosság érhető el. A hengeres, csavarvonal mentén elhelyezett skálabeosztású és csavar alakú nyelvvel ellátott logarlécek a 12 m hosszúságot is elérik, és pontosságuk ezért két nagyságrenddel nagyobb. Ennél nagyobb pontosságot logarléccel nem lehet elérni.

2. A tizedesvessző helyét az eredményben nagyságrendszámítással kell meghatározni.

A logarlécet az ezerkilencszázhetvenes években felváltották a gyorsabb, pontosabb és a nagyságrendet is helyesen ábrázoló **elektronikus zsebszámológépek.**

Egyszerű mechanikus számológépek

Blaise Pascal (1623–1662) 1641-ben SCHICKARDtól függetlenül hatjegyű számok összeadására fából készült mechanikus számológépet épített (elsősorban azért, mert unalmasnak találta apja adóbevételeinek összeadását és átszámítását). Számológépe megmaradt az utókornak.

Gépe számjegyekkel számol. Az egyes számjegyek 0-tól 9-ig fogaskerekek meghatározott beállításának felelnek meg (a későbbi feltalálók fogaskerekek helyett fogasléceket, létrafokkerekeket, bordáshengereket stb. is alkalmaztak). A számok ábrázolásához annyi, azonos tengelyen elhelyezett fogaskerékre van szükség, ahány számjegyből áll a szám. Az egy fogaskerékfoggal történő elfordítás a forgási iránytól függően 1 hozzáadását vagy levonását jelenti. A 9-ről a rákövetkező 0-ra való átmenet során a gép a legközelebbi helyiértéken álló számot automatikusan eggyel megnöveli, ez az eljárás a *tízesátvitel*. Az összeadásnál, pl. a 19-ről 20-ra történő váltáshoz erre elengedhetetlenül szükség van. Ha több kilences számjegy is elfordul, akkor egyidejűleg több tízesátvitelre is sor kerül.

PASCAL a számítás műveletét egymásba illeszkedő mechanikus alkotórészek mozgására vezette vissza.

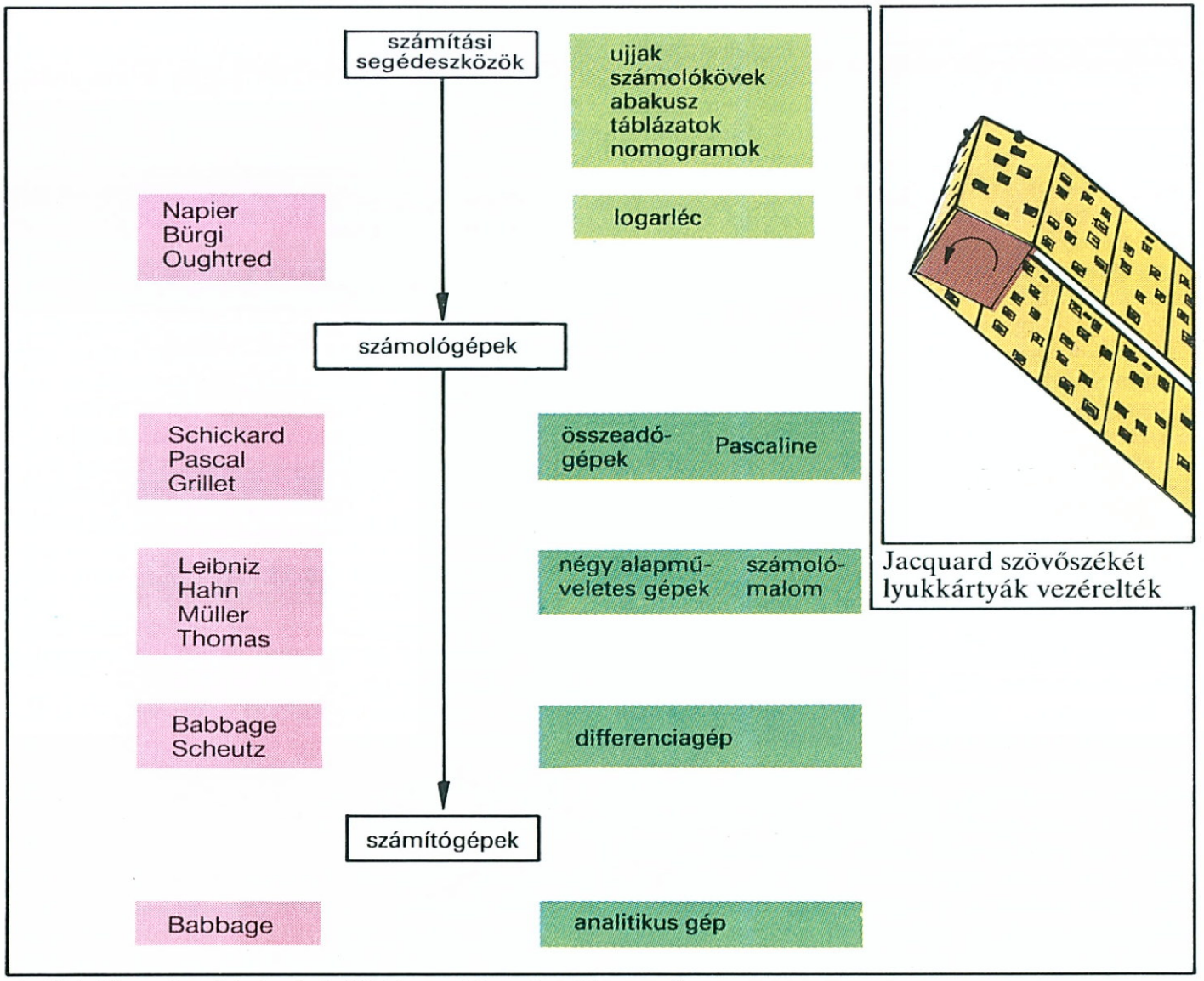
A szorzást ismételt összeadással, az osztást pedig ismételt kivonással végezte el.

Gottfried Wilhelm Leibniz (1646–1716) 1670 után *bordáshenger* alkalmazásával tökéletesítette PASCAL gépét. Így elsőként sikerült két szám szorzását illetve osztását közvetlenül, egyetlen körülforgatással megoldania. Négy alapműveletes számológépe nyolcjegyű számokkal végzett műveletekhez készült, azonban LEIBNIZ idejében a tízesátvitel során felmerülő mechanikus problémák miatt sohasem működött kielégítően.

A gép eredeti példánya kétszáz évre eltűnt, és csak 1879-ben került elő egy göttingeni ház padlásáról.

A 19. század végéig a szorzás elvégzésére a bordás számolóhenger jelentette az egyetlen, gyakorlatilag kivitelezhető mechanikus megoldást, és századunk közepéig alkotórésze maradt a mechanikus asztali számológépeknek.

Philipp Matthäus Hahn (1739–1790) kornwestheimi pap 1770-ben a bordáshengereket egy központi tengely körül kör alakban helyezte el. Ez a „számolómalom" megbízhatóan működött és hamarosan széles körben elterjedt. A négy alapművelet elvégzésére alkalmas számológépeket egyre tovább tökéletesítették. Hamarosan már csak egyetlen bordáshengerrel működtek; ezt követte a billentyűzet és a villamos meghajtás alkalmazása. 1905-ben készült az első teljesen automatikus, gombnyomásra működő készülék.



Jacquard szövőszékét lyukkártyák vezérelték

A digitális számológépek fejlődése a 19. század közepéig

szám	négyzete	1. diff.	2. diff.
1	1		
		3	
2	4		2
		5	
3	9		2
		7	
4	16		2
		9	
5	25		2
		11	
6	36		2
		13	
7	49		2

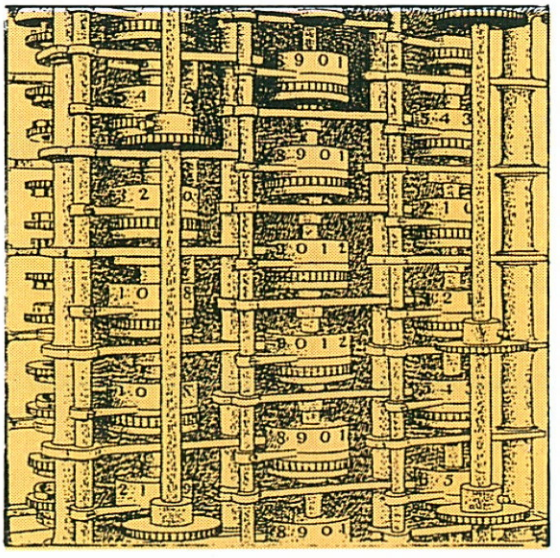
négyzet + 1. differencia + 2. differencia

= a következő négyzetszám

9 + 5 + 2 = 16

a négyzetszámok differenciái

A differenciagép működésének matematikai alapja



Babbage differenciagépének részlete

Leibniz a szorzás műveletét közvetlenül végző számológépen kívül két további, elméleti meggondolással is hozzájárult az informatika fejlődéséhez: 1666-ban bebizonyította, hogy egy számolási művelet egymás után elvégezhető több elemi lépés sorozatára bontható, 1679-ben pedig ismertette a kettes számrendszert, amely az összes modern számítógép működésének az alapját képezi.

A 17. és a 18. században épített számítógépek drágák és megbízhatatlanok voltak, inkább kuriózumnak, mint számítási segédeszköznek számítottak. A helyzet csak 1820-ban, **Charles-Xavier Thomas de Colmar** (1785–1870) „Arithromètre" nevű gépével változott meg, amely egy Leibniz-féle bordáshengerrel működött. Az első 50 évben 1500 darab került forgalomba. A négy alapműveletes számológépek virágkora a 20. század hatvanas éveire tehető (Friden Monroe).

Charles Babbage (1792–1871) 1821 után két új típusú számológépet épített: a differenciagépet és az analitikus gépet.

A **differenciagépek** az egyszerű függvényértékek kiszámítását, pl. a négyzetszámok és a harmadik hatványok vagy a logaritmusok sorozatának kiszámítását különbségek összeadására vezetik vissza.

Példa: A négyzetszámok sorozatának első tagjai 1, 4, 9, 16, 25, 36 stb. Az egymás után következő értékek különbsége (differenciája) 3, 5, 7, 9, 11 stb. A négyzetszámok különbségének különbsége (a differenciák differenciája, a 2. differencia) mindig 2. Egy a sorban következő négyzetszámot tehát úgy lehet megkapni, hogy az előző négyzetszámhoz hozzáadjuk a megfelelő első és második differenciát. Azaz: 4^2 = előző négyzetszám ($3^2 = 9$) + 1. differencia ($3^2 - 2^2 = 5$) + 2. differencia (= 2) = 16.

A harmadik hatványok és a logaritmusok kiszámításához több differenciát kell összeadni, de az elv mindig azonos: ha a sorozat előző értéke ismert, a következő érték a differenciákból kiszámítható. Babbage tervezte az első differenciagépet, amely a számításokhoz egészen a 6. rendig használta a differenciákat. A gépnek logaritmusokat kellett volna kiszámítania és a táblázatokat kinyomtatnia. A működéséhez szükséges 6 egymáshoz kapcsolódó számlálómű tervei ugyan – mai ismereteink szerint – hibátlanok, de Babbage anyagi okokból csak a számítógép egyes részeit tudta elkészíteni.

A differenciagép költségeit 1834-ben 17 470 font sterlingre becsülték. Összehasonlításképpen: egy mozdony csak 1000 fontba került.

A londoni Science Museum 1991-ben az eredeti tervek alapján (és a korabeli mechanikai lehetőségek figyelembevételével) rekonstruálta a differenciagép egyik egyszerűsített változatát, amelyet maga Babbage tervezett. Méretei: 3,4 m × 0,5 m × 2,1 m, súlya 3 tonna és 4000 egyedi részből áll. A gép néhány nappal Babbage 200. születésnapja előtt hibátlanul kiszámította a 7. hatványok táblázatának első száz értékét, ami késői diadalt jelent Charles Babbage számára.

Pehr Scheutz (1785–1873) és fia **Edvard** (1821–1881) egyszerűsítették Babbage készülékét és 1853-ban elkészítették az első működő differenciagépet.

Christel Hamann (1870–1948) tökéletesítette a berendezést és segítségével 1910-ben nagy sikerű kilencjegyű logaritmustáblázatot jelentetett meg. A differenciagépeket egészen az 1940-es évekig matematikai táblázatok készítésére használták.

Babbage második gépét, amelyen 1834-től kezdve dolgozott, **analitikus gépnek** nevezte. Univerzális gépet tervezett, amely adatbeviteli és eredménykimeneti egységből, számológépből és részeredmény-tárolóból állt volna. Az adatbevitelt Jackard-féle lyukkártyákkal akarta megoldani. Lyukkártyáknak kellett volna vezérelni a gép tulajdonképpeni számítási folyamatait is. A tárolómű 200 részeredmény tárolására lett volna alkalmas; erre a célra 1000 oszlop szerepelt a tervekben, és minden egyes oszlopban 50 fogaskerék helyezkedett volna el.

A gép építése már kezdetben megakadt. A 19. század finommechanikai lehetőségeivel ezt a „programvezérlésű" számológépet nem lehetett megvalósítani. Az analitikus gép egyes részei 1940 körül kerültek elő. Az analitikus gép működési elve miatt azonban feltalálóját így is a **számítógép atyjának** tekintjük.

A 19. század kutatásai elsősorban a folyamatok vezérlésének új lehetőségeire irányultak.

Folyamatszabályozás. A folyamatok lefutásának vezérlésére különböző módszerek léteznek. A zenegépekben pl. hengerek és láncok végzik ezt a feladatot. A gépek belsejében lejátszódó folyamatok kívülről történő gyors megváltoztatásához könnyen cserélhető eszközökre van szükség. A mintás szövőszékekhez pl. többféle vezérlési módszert találtak fel.

Brösel 1690 körül vászonszalagokra faelemeket ragasztott, ezek határozták meg a szőtt anyag mintáját. A vászonszalagok cseréjével lehetett a mintát változtatni. A lyoni selyemszövőgépekben kb. 1725 óta lyukasztott papírcsíkok látták el ugyanezt a feladatot. A fejlődésben **Joseph-Marie Jacquard** (1752–1834) találmánya jelentette a csúcspontot, amikor 1810-ben lyukkártyákon kódolta a textilmintákat. A lyukkártyákat láncra fűzte, és ezzel lehetővé tette az automatikus szövőszékek vezérlésének gyors és könnyű változtatását.

Hermann Hollerith (1860–1929) tárolt elsőként (1880-ban) számértékeket szabványosított lyukkártyákon. Villamos érintkezők tapogatták le a lyukkártyákat, az adatok további feldolgozását pedig egy számlálómű végezte.

Hollerith céget is alapított, ebből alakult 1924-ben az International Business Machine Corporation (IBM).

A különleges elektromechanikus számológépek felhasználása könyvelésre, biztosítási számításokra és táblázatok készítésére századunk hatvanas éveiben érte el tetőpontját. CHRISTEL HAMANN a vezető konsruktőrök közé tartozott.

A harmincas években a számológépek fejlődésében teljesen új korszak kezdődött, amelyet a kettes számrendszer alkalmazása és a nagymértékben rugalmas programozás jellemzett. Ennek alapjait **Gottfried Wilhelm Leibniz** teremtette meg a kettes számrendszerről 1679-ben írt közleményével.

George Boole (1815–64) és **Augustus de Morgan** (1806–71) alkotta meg 1847-től kezdve a formális logikát. A logikai kijelentések ábrázolásához és kiszámításához használható mechanikai kapcsolásokat először **William Jevons** (1835–82) alkalmazta. Az órák, automaták és záróberendezések készítéséhez már régóta bináris kapcsolásokat használtak. A mechanikus reléket – elsősorban a nagy mennyiségben előállított telefonreléket – csakhamar az elektromechanikus relék váltották fel. Az (1904-ben feltalált) elektroncsövek gyorsabb kapcsolóköröket és adattárolást tettek lehetővé. 1955 után az elektroncsöveket tranzisztorok váltották föl, ezeket pedig 1960 óta fokozatosan kiszorították a félvezető egykristályból készült integrált áramkörök (chipek).

Példák: Az 1945-ben készült relék másodpercenként legfeljebb 250-szer tudtak kapcsolni. Az 1993-ban készült nagy integráltságú félvezető áramkörök másodpercenként már több, mint 2 milliárd állapotváltozásra képesek.

1911 óta ún. **totalizátorok** számítják ki (valós idejű üzemmódban) a kutya- és lóversenyek fogadásesélyeit. Már az első készülékek fix programozású, számjegykijelzős elektromechanikus gépek voltak. Az egyes berendezések mérete a szoba nagyságot is elérte.

Leonardo Torres y Quevedo (1852–1936) 1914-ben vezette be a számítógép-építésben a lebegőpontos számábrázolást. 1910 és 1920 között egyedi számítási célokra (pl. két komplex szám szorzatának kiszámítására) olyan programvezérlésű mechanikus számológépeket épített, amelyek kimeneti egysége írógép volt. Tőle származnak a programnyelvek első kezdeményezései is.

Konrad Zuse (szül. 1910-ben) 1932-ben építette az első mechanikus tárolót tetszőleges adatok, elsősorban lebegőpontos ábrázolású (16 bit mantissza, 7 bit kitevő, 1 bit előjel, tehát 24 bit szóhosszúságú) számok tárolására, és 1934-től kezdve kifejlesztette a programvezérlésű számítógépek alapelveit.

Leslie Comrie (1893–1950) alapította 1937-ben Londonban az első kereskedelmi jelleggel működő számítóközpontot. A nagyobb feladatok megoldására több számító- és Hollerith-gépet kapcsolt össze.

Alan Turing (1912–54) 1936-ban leírta egy olyan számítógép matematikai modelljét, amely mint legegyszerűbb lehetséges univerzális számítógép-automata véges matematikai és logikai problémákat tud megoldani.

A **Turing-gép** három részből áll: egy mindkét oldalon végtelen hosszúságú munkatárszalagból, egy vezérlőegységből és egy olvasó-/írófejből. A szalag olyan mezőkre oszlik, amelyek mindegyike egy-egy jelet képes befogadni. Csak a fej alatt elhelyezkedő mező olvasható illetve írható. A Turing-gép a következőképpen működik: Kezdetben a gép meghatározott kiindulási állapotban van. A beolvasott jelektől függően végrehajt valamilyen eljárást, ezután a gép új állapotba jut. A fej a következő jelre kerül, beolvassa azt, és így tovább, egész addig, amíg az olvasófej alatt „stop" utasítás jelenik meg. Ilyen módon elvileg minden algoritmus kivitelezhető.

Konrad Zuse 1938-ban (otthon, szülei lakásának nappali szobájában) építette Z1 néven az első szabadon programozható számítógépet, amely kettes számrendszerben, lebegőpontos ábrázolású számokkal működött. Az adatbevitelre billentyűzet szolgált, az adatkivitel pedig ugyancsak kettes számrendszerben egy világító tábla (fénymátrix) segítségével valósult meg. A számolómű és a tároló telefonrelékből készült. A következő modell, a Z2 már lyukfilmes adatbeviteli egységet tartalmazott.

John Atanasoff (szül. 1903-ban) és **Clifford Berry** (1918–63) 1939-ben 25 bit, később pedig 50 bit szóhosszúságú adatokkal dolgozó bináris összeadógépet építettek. Az ötvenszavas tárolóegységet egy kondenzátorokkal felszerelt forgó dob alkotta. Az adatbevitel lyukkártyákkal, az adatkivitel pedig beégetett jelöléseket tartalmazó kártyákkal történt. A számítógép lineáris egyenletrendszereket oldott meg. A gép további fejlesztésének 1942-ben a háború vetett véget.

George Stibitz (szül. 1903-ban) ZUSEtől függetlenül építette *complex number calculator* nevű gépét, amely bináris aritmetikával és relé-tárolóegységgel dolgozott. Az adatbevitel távírógéppel történt. A gép egy javított (fix programozású) változatát 1943-ban ballisztikai számításokra használták.

Konrad Zuse 1941-ben fejezte be az első teljesen működőképes, szabadon programozható, programvezérlésű számítógépet (Z3). A teljesítményét jellemző műszaki adatok a következők: lebegőpontos számábrázolás, 22 bit szóhosszúság, 64 lebegőpontos adat tárolására képes tárolóegység, 1600 mechanikus relé a tárolóban, 400 relé a számolóműben (az eletroncsövek még túl drágák és megbízhatatlanok voltak), (3 s alatt) automatikus szorzás, osztás és gyökvonás.

ZUSE 1945 elején mutatta be Z4 nevű számítógépét (32 bit szóhosszúság, 64 fixpontos adat tárolása, 5500 relé), amelyet kísérletképpen a repülőgéptervezésben használtak fel. 1950-től a Z4 a Zürichi Műszaki Egyetemen (ETH) működött mint Európa egyetlen számítógépe. (1960 óta Münchenben a Deutsches Museumban található.)

COLOSSUS. Nagyon gyors, bináris rendszerben dolgozó automata számítógép, amelynek építését ALAN TURING kezdeményezte. Kizárólag kódok megfejtésére használták (ezzel fejtették meg a németek ENIGMA nevű rejtjelét is). Angliában készült 1943-ban. 1500 elektroncsövet tartalmazott, kvarcvezérléssel, 5 kHz-es órajellel dolgozott. Összesen 10 darab ilyen gép készült.

Harvard Mark I, más néven **ASCC** (*Automatic* Sequence *Controlled Calculator*). HOWARD AIKEN (1900–1973) fejlesztette ki és az IBM építette meg az Egyesült Államokban ezt a lyukkártyás, relé alapú számítógépet. 750 000 alkatrészből állt. Fixpontos ábrázolású számokkal (10 jegy a tizedespont előtt, 13 jegy a tizedespont után) dolgozott. Az adatbevitel lyukkártyákkal történt. A programot lyukszalagok vezérelték. Az összeadáshoz ⅓ s, a szorzáshoz 6 s időre volt szükség. A számítógép 1944-ben készült el, lassan, de megbízhatóan üzemelt 1959-ig.

Továbbfejlesztett változatai: a **Mark II** 1948-ban (lebegőpontos számábrázolással kiegészítve), a **Mark III** pedig 1950-ben elektroncsövek felhasználásával készült.

ENIAC (*Electronic Numerical Integrator And Calculator*). Az első teljesen elektronikus számítógép. Az Egyesült Államokban készült, tervezői: JOHN MAUCHLY és JOHN ECKERT. Ballisztikai és szélcsatornaszámításokra használták. 18 000 elektroncsövet tartalmazott, fogyasztása 800 kW volt, és 220 m^2 alapterületet foglalt el. Három nagyságrenddel gyorsabb volt, mint a relés számítógépek (az összeadást 0,2 ms, a szorzást 3 ms idő alatt végezte). Villamos csatlakozások útján feltételesen programozható volt. 1946-ban készült el és 1955-ig működött.

Továbbfejlesztett változata az első, kereskedelemben is kapható számítógép, az **UNIVAC I** (**UNIV***ersal* **A***utomatic* **C***alculator*).

EDVAC (*Electronic* **D***iscrete* **V***ariable* **A***utomatic* **C***alculator*). Elektroncsöves technikával készült 1944-től 1948-ig MAUCHLY és ECKERT vezetésével. Az első olyan számítógép volt, amely az adatokat és a programokat is maga tárolta. Az EDVAC olyan univerzális számítógép volt, amely a további fejlődést is jelentősen befolyásolta.

SSEC (*Selective Sequence Electronic Calculator*). WALLACE ECKERT (1902–71) építette 1947-ben az IBM megbízásából hidrodinamikai problémák megoldására. 12 500 elektroncsövet és 21 400 relét tartalmazott, tízes számrendszerben dolgozott, a programokat lyukkártyák tárolták.

A Turing-gép elvén alapuló digitális számítógép az ötvenes évek közepétől vette át a vezető szerepet. Ehhez a műszaki fejlődésen (elektroncsövek, mágneses tárak, tranzisztorok, integrált áramkörök megjelenésén) kívül a feladatorientált programnyelvekkel megvalósuló, egyre egyszerűbbé váló programozhatóság is hozzájárult.

A fejlődés állomásai címszavakban:

1944 ZUSE „ált. struktúrájú számítási tervet" készít. Ebből keletkezik 1946-ban az első magas szintű programnyelv, a „plankalkül".

1945 NEUMANN, MAUCHLY és ECKERT javasolják: a programokat a számítógépben tárolják.

1946 GEORGE R. STIBITZ (Bell Telephone Laboratory) megépíti első „univerzális számítógépét" (**Model IV.,** 9000 relé, a szorzás időtartama 1 s, a gyökvonásé 4,5 s).
NEUMANN és csoportja a princetoni **I**nstitut for **A**dvanced **S**tudies intézetben megépíti az **IAS** számítógépet. Az **IAS** építésében alkalmazott elvek (pl. gyors tárak párhuzamos összeadás) tartósan befolyásolták a számítógépek további fejlődését.

1948 **Manchester Mark I** (F. C. WILLIAMS és T. KILBURN) az első olyan számítógép, amely saját programját tárolta.

1949 **EDSAC** (*Electronic* **D***elay* **S***torage* **A***utomatic* **C***alculator*) Cambridge/Anglia. MAURICE WILKES építette.
Az IBM kereskedelmi számítások céljára sorozatban kezdi gyártani a **CPC** (*Card* **P***rogrammed* **C***alculator*) számítógépet.

1951 Megkezdődik az univerzális számítógépek sorozatgyártása USA: **UNIVAC I** (5600 elektroncső, 18 000 dióda, 19 tonna súly, 1 millió $); Anglia; **Ferranti Mark I.**
Az első japán számítógép (**ETL,** Fujitsu Company), amely relékkel és bináris lebegőpontos aritmikával működött.

1952 SZERGEJ LEBEGYEV (1902–74) Moszkvában befejezi az **MESM** és a **BESM** építését, ezeket követi 1953-ban a **STRELA** (tervezője J. BAZILEVSZKIJ).
HEINZ RUTISHAUER (1918–70) nyilvánosságra hozza az „automatikus számítási terv készítés" programozási nyelvet.

1954 Az IBM bevezeti a FORTRAN programozási nyelvet.
Megjelenik az első nagy sorozatban gyártott számítógép (**IBM 650,** 2200 darab).

1955 Az elektroncsöves számítógépek elérik legmagasabb fejlettségi fokozatukat (**IBM NORC,** a szorzás időtartama 31 μs).

1956 A ZUSE KG sorozatban gyártja a **Z11** számítógépet. A Darmstadti Műszaki Főiskolán számítógéptől független programnyelveket (ALGOL58, ALGOL60) fejlesztenek ki. Az MIT/USA megépíti az első teljesen tranzisztorizált számítógépet **(TX–0).**

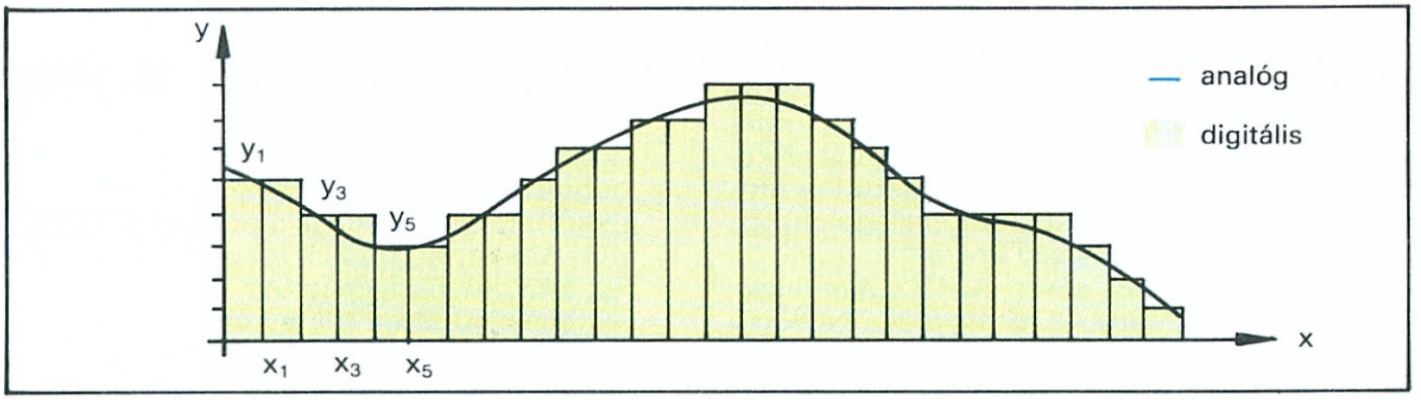
1964 Kereskedelmi forgalomba kerülnek az integrált áramköröket tartalmazó, részben már byte-szervezésű számítógépek.

1972 **CRAY–1** szuperszámítógép (részben párhuzamos feldolgozás, nagy sűrűségű felépítés, félvezetőtárak, az alapműveletek időtartama 12,5 ns).

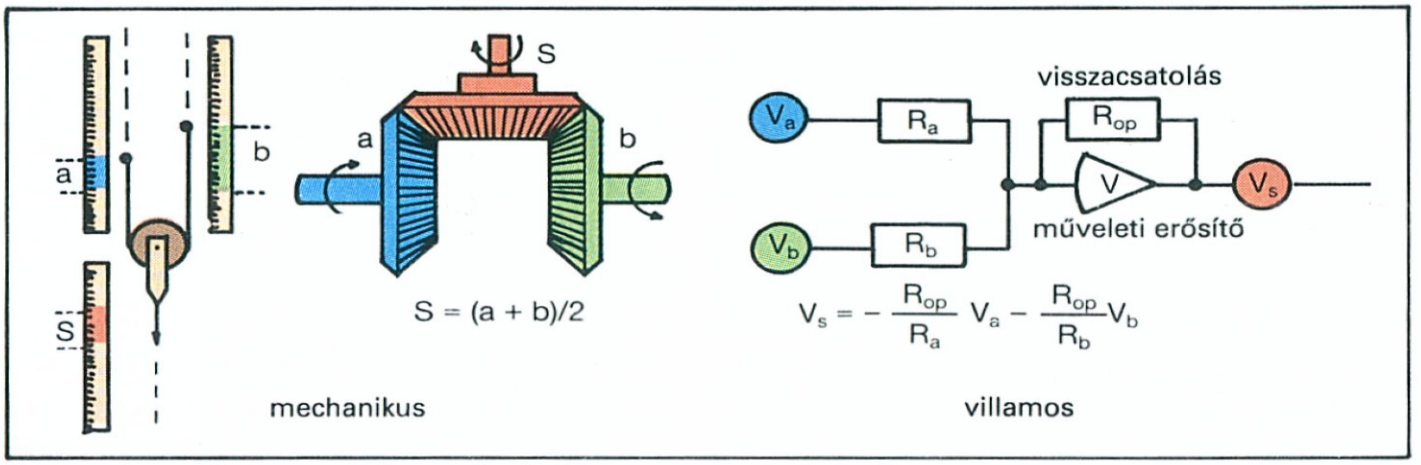
1974 Forgalomba kerülnek a programozható zsebszámológépek **(HP 65)**

1984 Az IBM aug. 12-én bemutatja a személyi számítógépet (PC, *Personal Computer*).

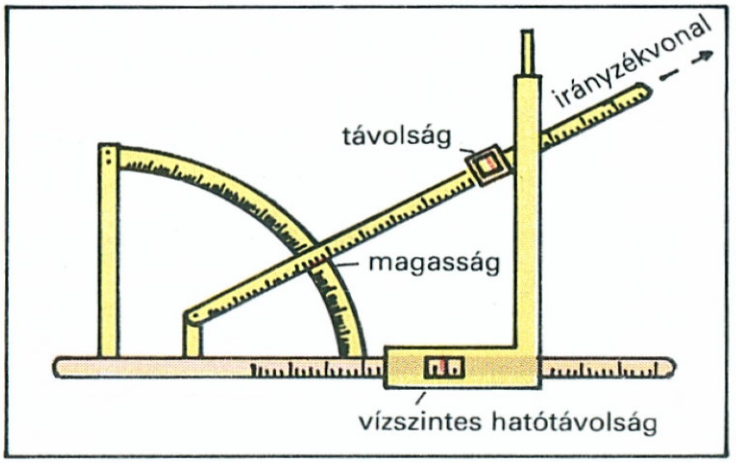
1994 Többségében párhuzamos adatfeldolgozással működő számítógépek megjelenése. Világszerte kb. 120 millió „IBM-kompatibilis" személyi számítógépet használnak.



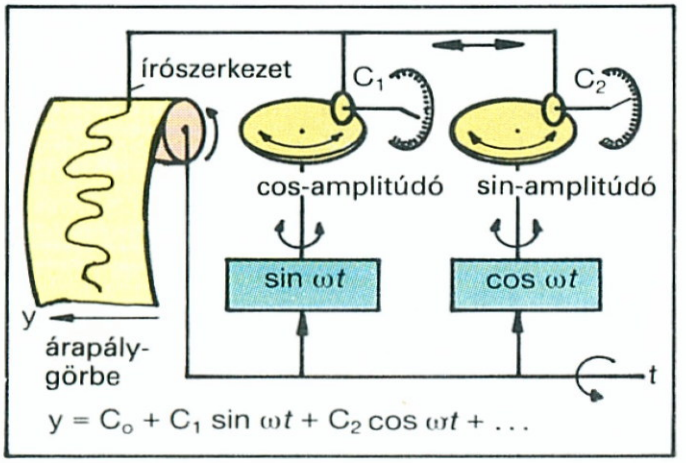
Analóg és digitális értékfelosztás



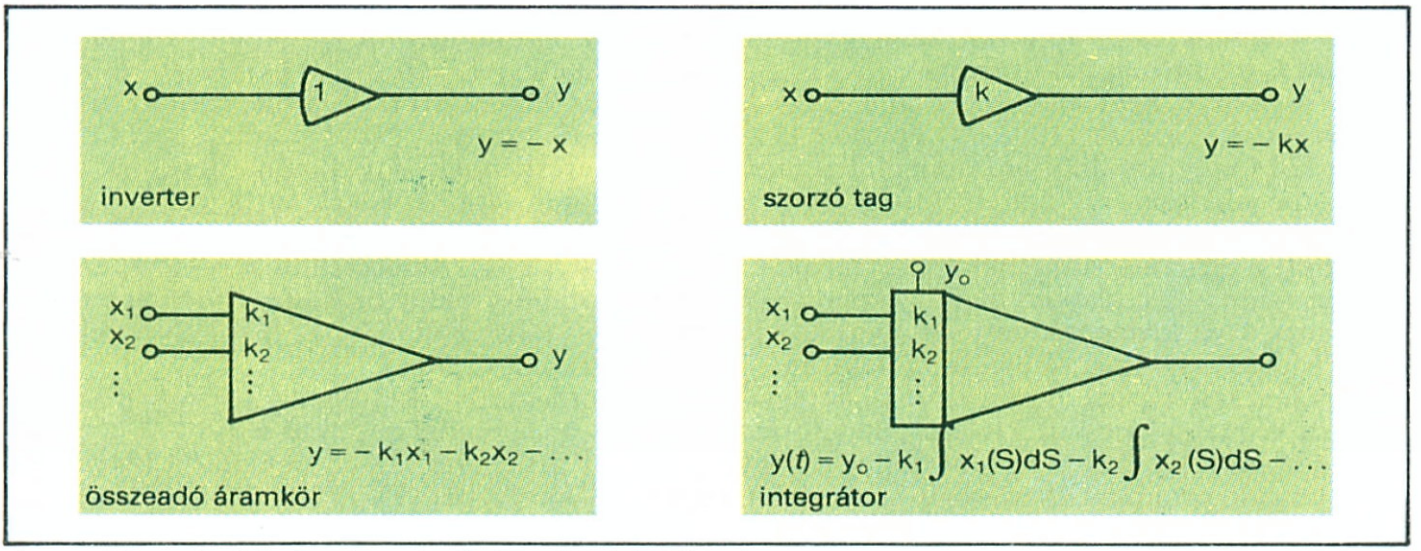
Egyszerű analóg összeadó berendezések



Légvédelmi analóg számítókészülék



Lord Kelvin „harmonikus analizátora”



Az analóg számítógép elemeinek áramköri jelölései

Az **analóg számítógépek** a mennyiségeket arányos hosszúságok, elfordulások vagy villamos feszültségek segítségével ábrázolják. A tisztán mechanikus számítások léceken történő hosszúságeltolással, kerekek elfordításával, fogaskerekek vagy fogaslécek, kötéltekercsek, csigák, orsók, csuklók és hengerek segítségével valósíthatók meg. A tisztán villamos analóg számítógépekben feszültségforrásokat, ellenállásokat, potenciométereket, kondenzátorokat, műveleti erősítőket stb. alkalmaznak. Az analóg számítógépeket napjainkban szinte kizárólag csak matematikai függvények és egyenletek ábrázolására és megoldására, valamint különböző folyamatok modellezésére használják.

Az analóg mennyiségek elvileg tetszőleges közbenső értéket felvehetnek. Ezzel szemben a digitalizált mennyiségek esetén az egymás után következő, szomszédos értékek között mindig van egy minimális érték, ill. egy minimális eltérés.

Példa: az analóg és a digitális óra.

A **digitális számítógépek** egyedi, jól elkülönült, stabil állapotokat használnak fel a mennyiségek valamilyen számrendszerben történő ábrázolásához. A számítások maguktól a számértékektől független absztrakt szabályok segítségével történnek. A digitális számítógépek univerzális számítóberendezések, elvileg bármely megoldható probléma kiszámítására alkalmasak.

A **hibrid számítógépek** analóg és digitális számítógépek kombinációi. Minden egyes alkotórészük a működési módjának leginkább megfelelő feladatot végzi.

Példa: A differenciálegyenletek megoldása során az analóg rész közelítő megoldást határoz meg, amelyet ezt követően a digitális rész pontosít.

Zaj

A mechanikus, villamos vagy elektronikus úton előállított mennyiségeket az ábrázoló közeg szükségképpen pontatlanul képezi le.

Példák: A fogaskerék helyzete az elérhető mechanikus tűrőképességtől függ. A villamos feszültség nagysága a villamos kapcsolásokban egy középérték körül ingadozik. A digitális számábrázolás mindig véges számú helyiértékkel dolgozik.

Az informatikában ezt az elkerülhetetlenül fellépő pontatlanságot **zajnak** nevezzük.

Az **analóg számítógépekben** illetve számítóeszközökben teljesen nyilvánvalóan a zaj korlátozza az eredmények pontosságát, különösen akkor, ha sok részeredményt kell feldolgozni. Napjainkban 0,1% (1:1000) pontosság könnyen elérhető. A 0,01%-os pontosság elérése már nagy ráfordítást igényel; 0,001% pedig valószínűleg a pontosság gyakorlati, tehát véges ráfordítással elérhető határértékét jelenti.

Példa: A 25 cm hosszú logarléc pontossága kb. 0,1%. A 0,01% pontosság eléréséhez már kb. 12 m hosszú skálát kellene alkalmazni.

A **digitális számítógépekben** az elérhető pontosság csak az (anyagi) ráfordítástól függ, mivel a mennyiségek elvileg tetszőleges számú helyiérték felhasználásával ábrázolhatók és feldolgozhatók. A legegyszerűbb digitális zsebszámológépek is elérik a 0,01% pontosságot.

A gyakorlatban a zaj által meghatározott pontosság csak korlátozott szerepet játszik. A bemeneti mennyiségek ritkán pontosabbak, mint 0,01%.

Az analóg számítógépek előnyei és hátrányai

A legtöbb analóg számítógép és számítóeszköz, mint pl. a logarlécek és a planiméterek felépítése egyszerű és áttekinthető. Az egyes függvényeket közvetlenül ábrázolják és ezek ennek megfelelően gyorsan feldolgozhatók.

Ezzel ellentétben a digitális számítógépek a függvényeket sorfejtéssel álítják elő. Megfelelő pontosság eléréséhez a számítógépnek nagyszámú tagból álló sorfejtést kell elvégeznie.

A szükséges számítási idő nagymértékben független a számítás bonyolultságától. Az eredmény gyakorlatilag rögtön rendelkezésre áll, mivel az analóg számítóberendezések – működési módjukat tekintve – párhuzamos üzemmódban dolgoznak.

Hátrányaik: Az analóg számítógépek különleges egyedi feladatok megoldására szolgálnak, a működésük során felhasznált algoritmusok kevéssé változtathatóak. Ha a megoldandó feladat változik, a számítógép működési egységeit is ki kell cserélni, ha ez egyáltalán lehetséges.

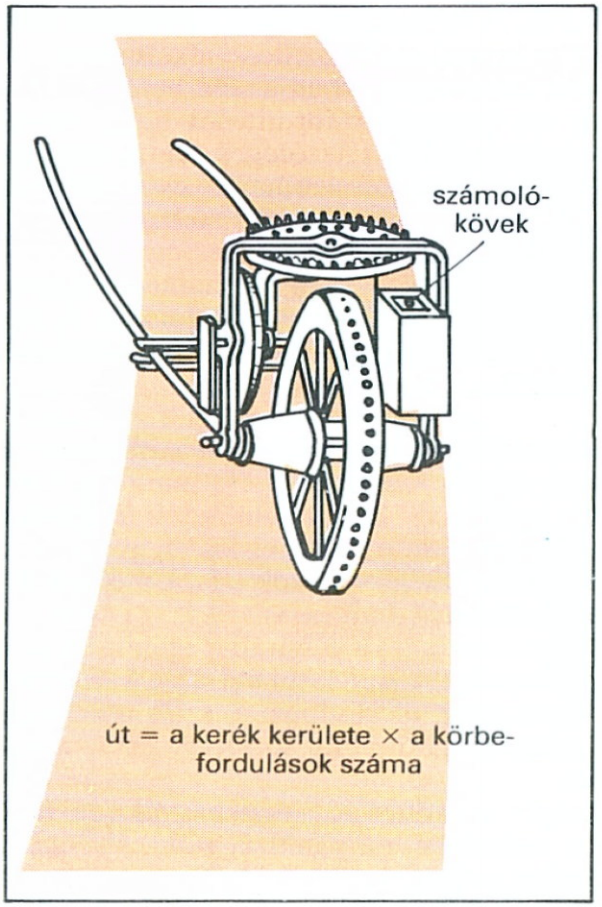
Ezzel szemben a digitális számítógépek tetszőlegesen programozható berendezések

A nagy pontosság elérése az analóg számítógépek esetében komoly műszaki ráfordítást igényel, pontosságuk gyakorlatilag nem haladhatja meg a 0,01%-ot.

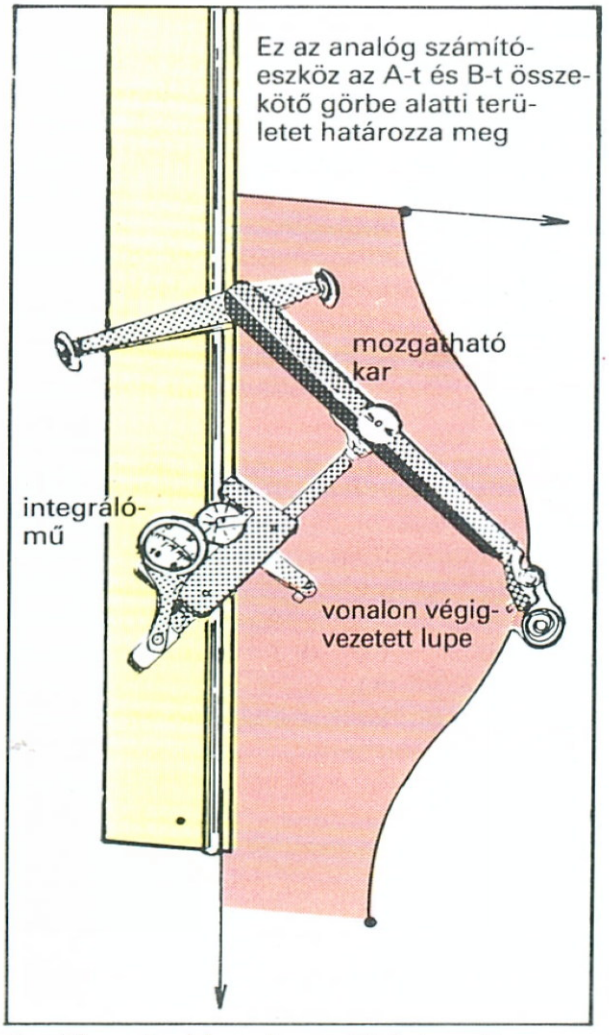
A digitális számítógépek napjainkban már szinte teljesen kiszorították az analóg számítógépeket. Az analóg számítógépeket már csak egészen egyedi feladatok megoldására, pl. villamos hálózatok és többparaméteres folyamatok modellezésére, magasabb rendű differenciálegyenletek megoldására használják.

Németországban kb. tízszer nagyobb analóg számítógép van használatban.

Összehasonlításképpen: a digitális számítógépek száma több millió.



Ókori úthosszmérő



Az 50-es években használt integrátor

i. e. 60.	Antikythera-számoló, 1901-ben találták, 39 részből áll, differenciálművet is tartalmaz
1. sz.	*Alexandriai Héron* útmérője, az első analóg-digitális átalakító
13. sz.	Órák, harangjátékok, zeneautomaták
1364	*Giovanni de Dondi* csillagászati órája
1614	*John Napier* közli az első logaritmustáblázatot *Edmund Gunter* logaritmikus beosztású számolólécet használ
kb. 1620	Egymáson elcsúsztatható skálákból álló logarléc *(William Oughtred)*
1712	*Earl of Orray* óraművei bemutatják a bolygók Nap körüli mozgását
1814	*Hermann* megalkotja a planimétert
1848	*Wetli* dörzshajtású planimétert épít, amely a 20. sz. közepéig használatban marad
1849	planiméterek sorozatgyártása
1850	*Amedee Mannheim* bevezeti a léctestben csúszó logarlécnyelvet
1856	*Jacob Amsler*-féle polárplaniméter még mindig használatban van
1876	*Lord Kelvin* árapályszámító berendezése. Az analóg számítóberendezések első csúcspontja. Fuorier-analízisre használt tárcsa-golyóhenger szerkezetű „harmonikus analizátor"
1881	A területet körüljáró planiméterek területnövekedést mérnek, azaz pontról pontra haladva integrálnak egy függvényt
1910	Ötismeretlenes lineáris egyenletrendszert megoldó számítógép *(Josef Nowak)*
1914	Az első menetrendkészítő diagráf *(Udo Knorr)*, amely a hetvenes évekig használatban marad
1920-tól	Villamos alkatrészek felhasználása analóg számítóeszközökben
1923	Planetárium-berendezés *(Zeiss)*
1927	Vickers-lőelemképző
1928	Légvédelmi számítóberendezések sorozatgyártása *(Zeiss, Siemens)*
1930	*Vannevar Bush* megépíti az első univerzális analóg számítógépet. Az analóg számítóeszközök második csúcspontja
1936	Az első, részben villamos analóg számítógép
1938	A műveleti erősítők feltalálása
1942	Nagypontosságú „differenciálanalizátor" *(Wannevar Bush)* Az első programvezérlésű számítógép *(Konrad Zuse)*. Analóg folyamatok digitális irányítása
1950-től	A teljes villamos analóg számítógépek uralják a piacot
1951	A Schoppe & Faeser cég megépíti az utolsó, teljes egészében mechanikus integrálókészüléket. Az analóg számítókészülékek 3. csúcspontja
kb. 1970	A mechanikus lőelemképzőket digitális készülékek váltják fel. A digitális számítógépek széles körben kiszorítják az analóg számítógépeket

Az analóg számítógépek fejlődése

A legrégibb ránk maradt analóg számítóeszköz már viszonylag bonyolult szerkezet, amelynek hajtóműve 39 részből áll. Egy görög hajó rakományához tartozott, amely i. e. az 1. században Kréta közelében Antikythera szigete mellett elsüllyedt **(Antikythera-számoló).** Az adatbevitel a holdszaknak a beállítógyűrűn való beállításával történt (a gyűrű 29,53 napos periódust vett figyelembe). Ezután egy mutatószerkezet különböző körgyűrűkön feltüntette a Nap és a Hold csillagképekhez viszonyított állását, valamint a legközelebbi hold- és napfogyatkozásokat. A ládában elhelyezett fogaskerekes szerkezet már abban az időben differenciálmeghajtással működött. A láda tetején a restaurátorok egy használati utasítást is felfedeztek.

A szerkezet pontos hasonmását elkészítették, így bebizonyosodott, hogy ez az antik analóg számítóeszköz helyesen működött.

Sokkal egyszerűbb Alexandriai Heron **útmérő berendezése,** amely az i. sz. 1. századból való.

Egy szabvány kereket végiggurítottak a mérendő útszakaszon. A kerék meghatározott számú körülfordulása után a szerkezet egy számolókövet dobott egy kosárba. A kövek száma arányos volt a megtett úttal.

Ez a készülék tekinthető az első analóg-digitális átalakítónak.

Az ókor óta ismert számos *asztrolábium* nem tekinthető számítóeszköznek. Ezek csupán az égitestek egymáshoz viszonyított helyzetének gömbökön vagy lemezeken történő ábrázolását teszik lehetővé.

A 11. században már ismerték a kötélcsiga- és a kerékhajtást; ezek segítségével működtek az esztergapadok.

A 14. században jelent meg a legelterjedtebb analóg számítóeszköz: mechanikus óraművek hajtották a toronyórákat, a csillagászati órákat, a harangjátékokat és a zeneautomatákat. A logaritmus felfedezése vezetett a 17. század elején a **logarléc** elkészítéséhez, amely a szó legszorosabb értelmében analóg számítóeszköz.

A 18. század eleje óta dörzskerékhajtású szerkezeteket használtak a görbe vonalak hosszúságának meghatározására. Ezekből alakult ki kb. egy évszázaddal később a **planiméter,** amely szintén igazi analóg számítóeszköz: egy síkidom kerületét körbejárva összegezte az általa bezárt területet, tehát már egyszerű adattárolóval is rendelkezett. Az eredménykijelzés kezdetben analóg módszerrel, mutatók segítségével, később pedig digitális módon, számtárcsákkal történt. Ezeket az integrátorokat egyes műhelyek a 19. század közepén már kisebb sorozatokban is gyártották.

Az analóg számítóeszközök fejlődésének egyik fénypontját William Thompson (a későbbi Lord Kelvin) **árapályszámító berendezése** (angolul: *tide predictor*) jelentette. Az 1876-ban épített modell előre kiszámította a Temze torkolatában a vízállás időbeli alakulását. A szerkezet magvát tizenegy, kötélcsigahajtással szinkronizált integrátor képezte. Az analóg számítóeszköz egyes alkotórészei az árapály alakulására gyakorolt hatás szempontjából a következő tényezőket vette figyelembe: a Hold és a Nap helyzete, a Föld forgása, a Föld lapultsága és a Föld tengelyének hajlásszöge. Az idő függvényében felrajzolt folytonos görbékről a vízállást leolvasták, az adatokat táblázatokba gyűjtötték és kinyomtatták.

Az árapályszámító berendezés építése rendkívül sokba került, mivel azonban a Temze medrét nem mélyítették ki, ezért a rendkívüli módon növekvő hajóforgalom megkövetelte a vízállás pontos ismeretét.

Az 1870-es évek közepén építette Lord Kelvin a **harmonikus analizátort** (angolul: *harmonic analyser*), amely az árapálygörbék Fourier-analízisét végző analóg számítóeszköz volt. Tetszőleges rendű differenciálegyenletek megoldására egy univerzális analóg számítóeszközt is tervezett, azonban ennek elkészítése a mechanikai kivitelezhetőség korlátaiba ütközött.

A századforduló idején a hajók kilövőberendezéseihez, valamint mozgó célok követésére bonyolult mechanikai felépítésű **lőelemképzőket** szerkesztettek. A ballisztikai egyenletek megoldása analóg módon, mechanikus hajtómű segítségével történt, mivel csak ez a módszer szolgáltatta azonnal a számítás eredményét.

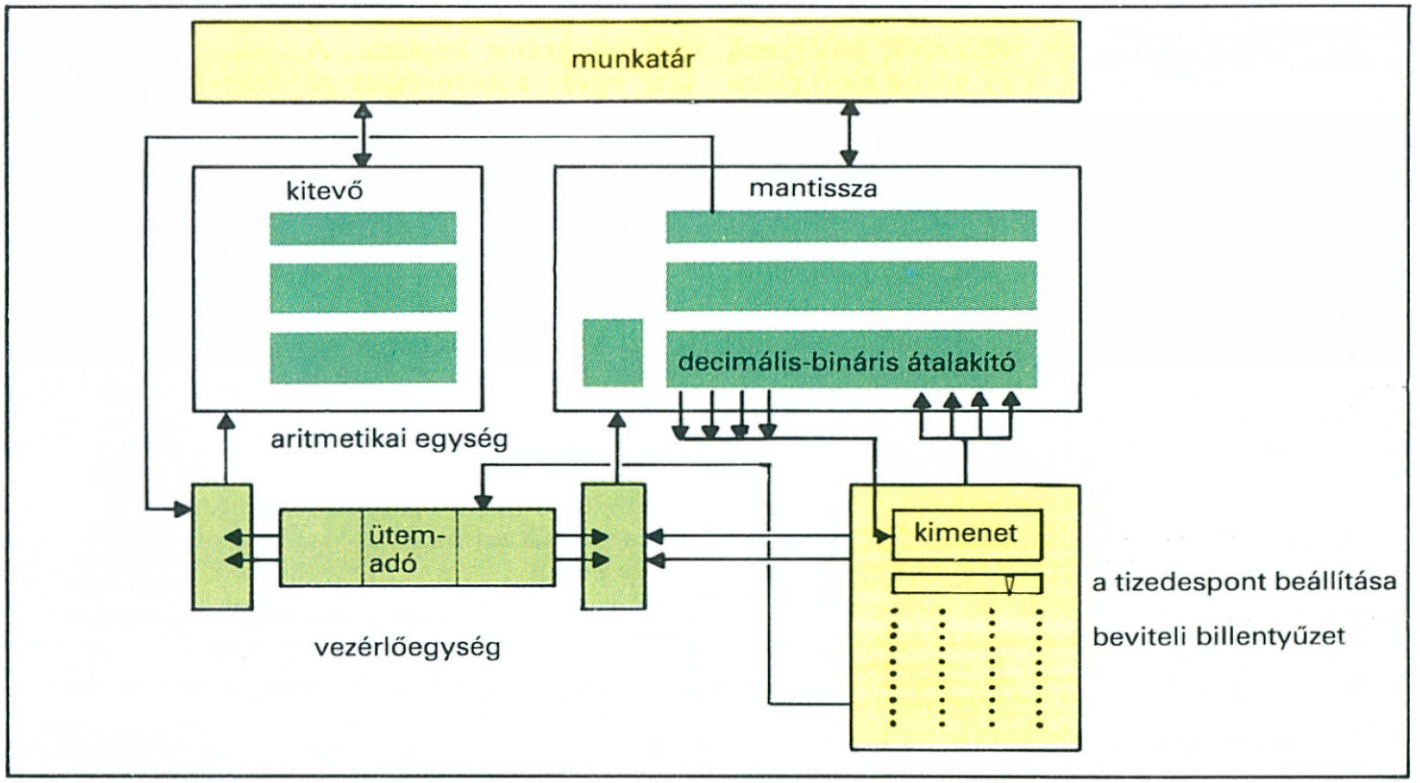
A ballisztikai célokra szolgáló analóg számítógépeket egészen a 20. század közepéig használták, ezt követően (elsősorban anyagi megfontolásból) a digitális számítógépek használatára tértek át.

1914-ben Udo Knorr a vasút számára **menetrendkészítő diagráfot** épített. A készülék cserélhető vonallapok segítségével egy-egy vasútvonal mentén kiszámította a menetsebességet és a menetidőt a vasútvonal profiljának, a mozdony típusának és a szállított teher súlyának függvényében.

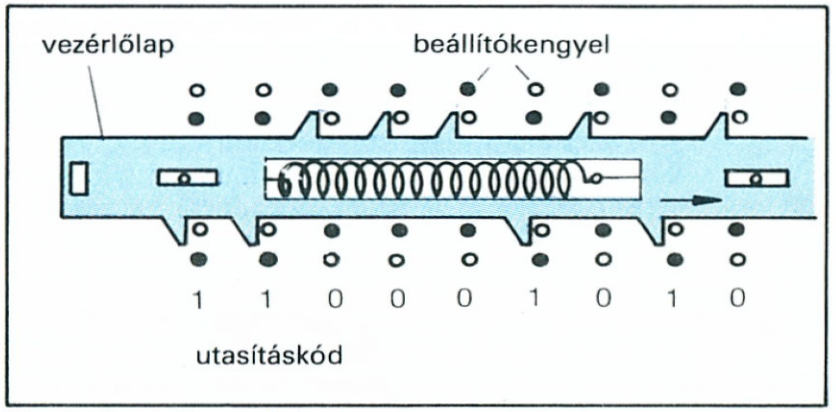
Ennek az analóg számítóeszköznek a javított változatát Németországban az 1970-es évek végéig használták.

1920 és 1930 között az analóg számítóeszközök fejlődése újabb csúcspontot ért el. Vannevar Bush integrátorai, amelyeket két függvény szorzására használtak, valamint a Fourier-analízist végző differenciálanalizátor (angolul: *differential analyser*) 0,1% pontosságot értek el és évtizedekig konkurencia nélkül uralták a piacot. Sok mechanikus alkotórészüket már villamos áramköri elemek, pl. potenciométerek, relék, szervomotorok helyettesítették. A harmincas évek közepén a (nagyon nagy erősítésű és nagy visszacsatolású) *műveleti erősítők* feltalálása tette lehetővé az első teljes egészében villamos alkotórészekből álló analóg számítógép megépítését, amelyben az egyes mennyiségeknek villamos feszültségek feleltek meg.

A nagyobb analóg számítógépek drágák, ezért csak kis darabszámban készülnek.



A Zuse-féle Z1 típusú számológép elvi felépítése



Vezérlőegység vázlata

Jellemzői:
kettes számrendszer
lebegőpontos számábrázolás
22 bit szóhosszúság
64 szavas tárolóegység
elektromágneses relétechnika
(2000 relé)
adatbevitel: billentyűzetről
kivitel: fénymátrix

Z3 – a világ első univerzálisan programozható számítógépe

név	gyártási év	szó-hossz-szúság (bit)	tárrekeszek száma	egy szorzás ideje (ms)	számolómű	tároló
Z1	1938	24	16		relék	mechanikus
Z2	1939	16	16		relék	mechanikus
Z3	1941	22	64	3000	relék	mechanikus
Z4	1944	32	64 (500)	3500	relék	mechanikus
DERA	–	51	3000	12	elektroncsövek	ferritmagos
Z11	1951				relék	mechanikus
G1	1952	32	312	500	elektroncsövek	mágnesdobos
G2	1954	50	2048	10	elektroncsövek	mágnesdobos
PERM	1956	51	8192	0,1	elektroncsövek	ferritmagos
D1	1956					mágnesdobos
Z22	1957	38	8192	10	elektroncsövek	mágnesdobos
Siemens 2002	1959	49	12000	1,3	tranzisz-torok	mágnesdobos ferritmagos
ER56	1959	16	22000	0,6	tranzisz-torok	ferritmagos
G3	1960	42	4096	2	elektroncsövek	ferritmagos
TR4	1961	52	10000	0,03	tranzisztorok	ferritmagos

Régebbi német számítógépek adatai

Digitális számítógépek

Németországban **Konrad Zuse** határozta meg a 30-as évektől kezdve a számítógépek fejlődését. Készülékeit nem a magas fejlettségi fokot elért mechanikus számítóberendezések és lyukkártyás gépek elveinek alapján építette, amelyek egészen az ötvenes évekig uralták a számítógéppiacot. Shannontól függetlenül, saját „feltételkombinatorikájából" kiindulva kapcsolásalgebrát fejlesztett ki, és 1936-tól kezdve készülékei ennek felhasználásával működtek.

Zuse elgondolásai és készülékei kb. a második világháború végéig előbbre tartottak, mint a világ többi országában történt fejlesztések. Ennek ellenére Németo.-n kívül nagyon kevéssé befolyásolták a számítógépek fejlődését, mivel Zuse eleinte otthon építette készülékeit, később pedig titoktartásra kényszerült.

Zuse kezdettől fogva bináris számábrázolást alkalmazott, ő építette az első decimális-bináris (illetve bináris-decimális) átalakítókat az adatok be- és kiviteli egységéhez. Készülékei (a Z2 kivételével) kezdettől fogva lebegőpontos számábrázolással dolgoztak. Kapcsolóelemként megbízható szabványos telefonreléket alkalmazott.

Zuse ugyan korán felismerte a vákuumcsövek előnyeit, de árukat túl magasnak, élettartamukat pedig túl alacsonynak találta.

1936-ban kezdte el az első digitális számítóeszköz építését (amelyet később Z1-nek nevezett), és 1938-ban fejezte be a munkát. Ezt 1941-ig két további gép (a Z2 és a Z3) követte. A Z3 volt a világ **első működőképes, teljesen automatikus, programvezérlésű digitális számítógépe.** 1942-ben fejezte be az első (S1 elnevezésű) folyamatirányító számítógépének építését. Ezt repülőgépek szárnyfelületének meghatározására használták és 1944-ig működött. Az eredeti Z1, Z2 és Z3 számítógépek megsemmisültek.

A Z1 hasonmását utólag megépítették, és Berlinben a Közlekedési és Műszaki Múzeumban látható. A Z3 hasonmás változata Münchenben, a Deutsches Museumban található.

1945 elején Zuse bemutatta a berlini Aerodinamikai Kutatóintézet számára épített Z4 számítógépet. Ezt Allgäuban biztonságba helyezte és 1946-tól a helyi tejüzem számára ezzel számította ki a tej zsírtartalmát. A Z4 közben kissé módosított változatát 1950-től a zürichi Műszaki Főiskola bérelte. 1955-től 60-ig a Z4 egy aerodinamikai intézetben működött. Ma a Deutsches Museumban található Münchenben.

Helmut Schreyer (1912–1984) Zuse munkatársa volt a Z1, Z2 és Z3 építése idején. Logikai alapáramkörök kifejlesztésével foglalkozott, amelyeket elsőként valósított meg triódák, tetródák és glimmlámpák kombinációjával. Schreyer 1944-ben tíz bináris jeggyel dolgozó elektronikus aritmetikai egységet épített.

Gerhard Dirks (1910–90) 1944-ben feltalálta az ütemsávos mágnesdobos tárat.

A 60-as évek végéig tartó fejlődés

1946 A Darmstadti Műszaki Főiskolán **Alwin Walther** (1898–1967) vezetésével kifejlesztették a **DERA** (**D**armstädter **E**lektrischer **Re**chenautomat) nevű számítógépet, amelynek építését 1950-ben abba kellett hagyni.

1949 A Müncheni Műszaki Főiskolán **Hans Piloty** (1894–1969) és **Robert Sauer** (1898–1970) vezetésével megépítik a **PERM** (**P**rogrammgesteuerter **E**lektrischer **R**echenautomat **M**ünchen) számítógépet (1956–1972-ig üzemelt).

1950 A göttingeni Egyetemen **Ludwig Biermann** (1907–86) vezetésével elkezdték építeni az (1952-ben üzembe helyezett) **G1** nevű számítógépet, amelyet a (1960-ban elkészült) **G2** és a **G3** követett. A Drezdai Egyetemen **Nikolaus Lehmann** (szül. 1921-ben) vezetésével elkezdték a **D1** építését, amelyet 1956-ban helyeztek üzembe. A Zuse KG megkezdte a Z11 nevű programvezérlésű relés számítógépek gyártását (kb. 50 db készült).

1954 A Siemens & Halske cég elkészítette a **Siemens 2002** univerzális számítógép terveit. Gyártása 1959-ben indult el.

1955 Feloldják az elektronikus gépek gyártásának Németországra vonatkozó tilalmát.

Az IBM sindelfingeni üzemében elkezdik az **IBM 604** lyukasztógép, majd 1956-tól az **IBM 650** számítógép gyártását.

1956 A Telefunken (AEG) cég kifejlesztette az **ER56** spec. számítógépet (1959-től árusították).

1957 A Zuse KG megkezdte az elektroncsövekkel és ALGOL 60 (illetve ALCOR) fordítóprogrammal működő **Z 22** számítógép gyártását, amely az **IBM 650** mellett ekkor az egyetlen olyan számítógép, amelyet Németországban ipari körülmények között állítottak elő.

1961 A SEL cég megkezdi a **TR4** univerzális számítógép gyártását, azonban ezt követően különleges számítógépekre szakosodik.

1966 A német egyetemeken bevezetik az informatika szakot.

1967 A Siemens cég átveszi a Zuse KG-t.

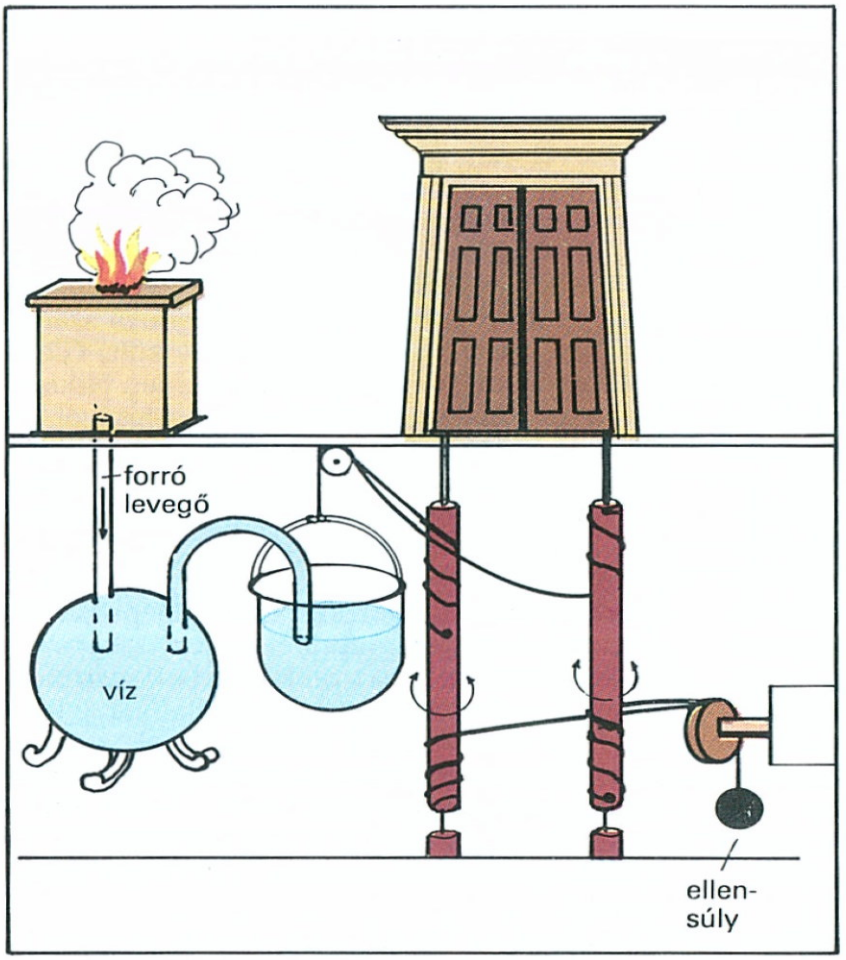
Analóg számítógépek

Az Udo Knorr cég kezdte el 1914-ben a mechanikus elven működő speciális analóg számítóeszközök gyártását, amelyeket vasúti menetrendek készítésére használtak.

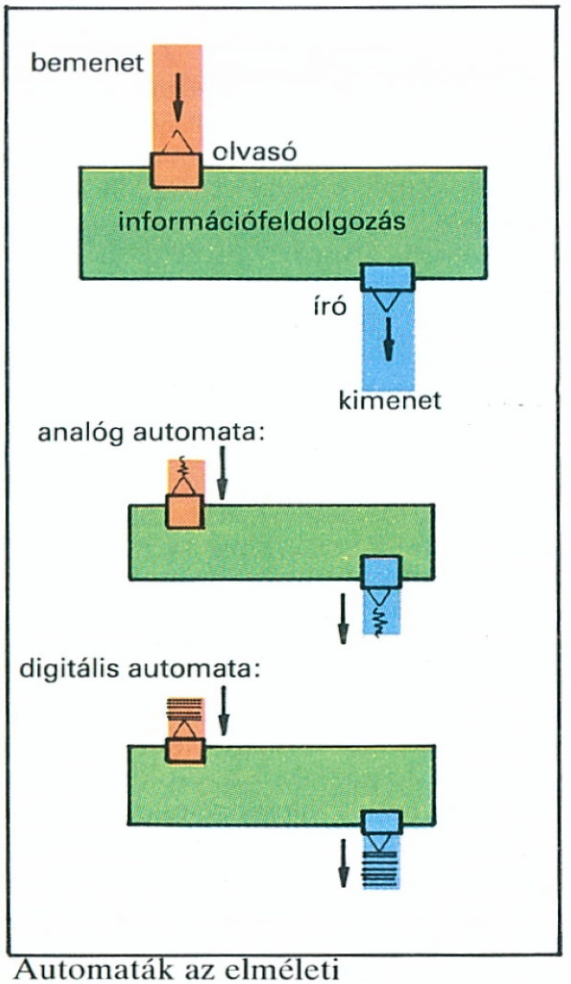
Vannevar Bush mechanikus elven működő univerzális analóg számítógépei közül 1945 előtt egyetlen darab sem került Németo.-ba.

Az Askania és Ott gépek 1940-től kezdve mechanikus integrátorokat állítottak elő. A Siemens és a Zeiss az analóg lőelemképzők építésére specializálódtak. Konrad Zuse 1942-ben bemutatta az első (S1 elnevezésű, szárnyfelületek meghatározására használt) folyamatirányító számítógépét. 1944-ben Alwin Walther és Robert Sauer nemlineáris differenciálegyenletek megoldására használható analóg integrálóberendezés kifejlesztésével kezdtek foglalkozni. A készülék 1948-ban készült el és 1956-ig működött.

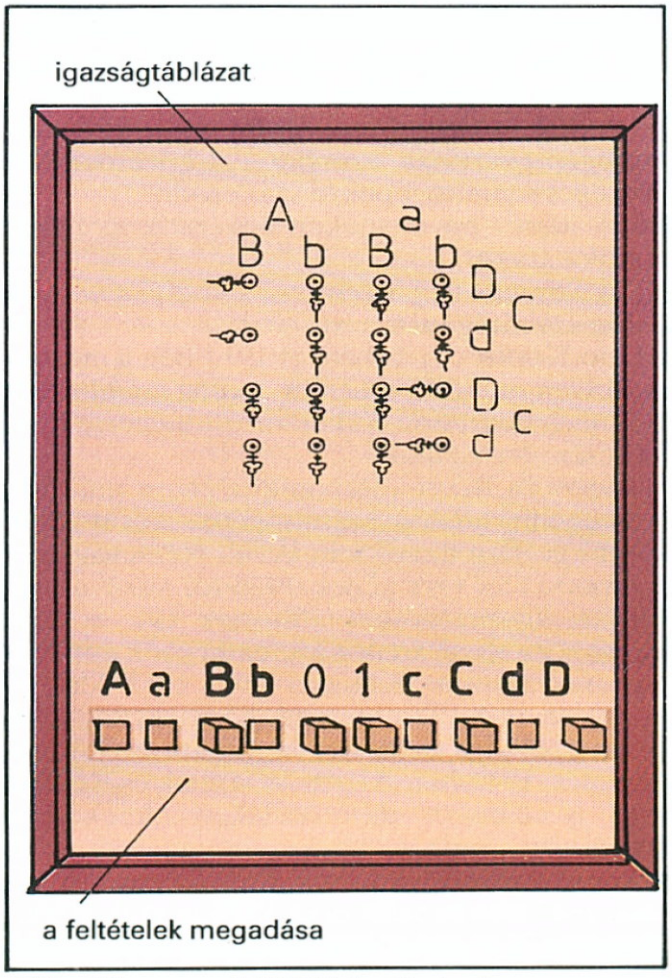
A Schoppe & Faeser cég 1948 után három analóg számítógépet épített, amelyek ebben a kategóriában a legnagyobb teljesítményű készülékek közé tartoztak. A teljes egészében elektronikus felépítésű analóg integrátorok (Telefunken) 1957-ben kerültek piacra.



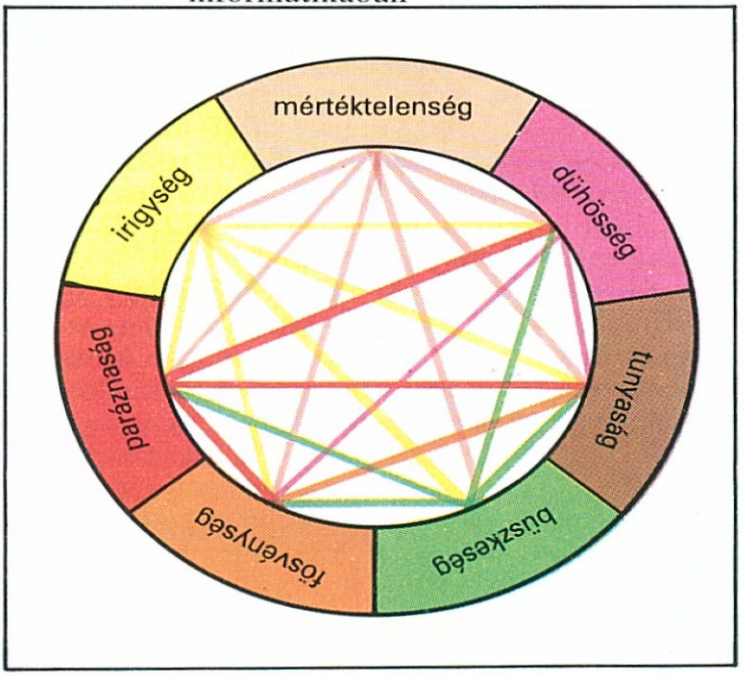
Az Alexandriai Heron automatikus ajtónyitója



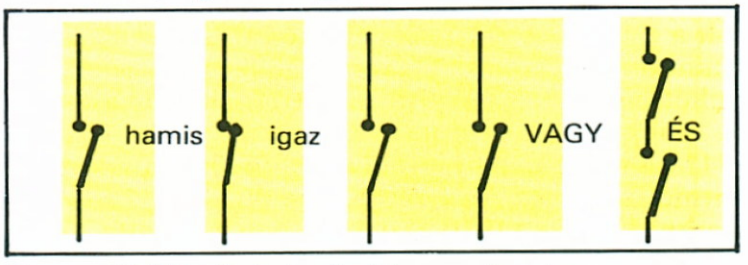
Automaták az elméleti informatikában



Allen Marquand (1881) logikai gépe



A 7 főbűn, logikai diagramon feltüntetve



Kapcsolóállások és logikai kapcsolatok

A számítógépekkel nem csupán nagyon sok aritmetikai műveletet lehet nagyon rövid idő alatt elvégezni, hanem ennél sokkal többre is képesek. A modern számítógépek úgy programozhatók, hogy velük formális logikai következtetések is levonhatók.

Automaták

Már a hellenisztikus korszakban készítettek olyan automatákat, amelyek emberi cselekvéseket szimuláltak.

Az **automaták** olyan gépek, amelyek valamilyen bemeneti mennyiség hatására a kimenetükön meghatározott eredményt hoznak létre.

A **véges automaták** tárolókapacitása korlátozott, és a műveleteket késleltetés nélkül végrehajtják.

Példa: az automatikus telefonközpont.

Az „automata" és a „véges automata" kifejezéseket gyakran szinonímnek tekintik.

Az informatikában automatákon az információfeldolgozást szolgáló (segéd)eszközök, berendezések matematikai modelljeit értjük. Például a különböző programnyelvek fordítóprogramjait automatáknak tekintjük.

A **számítógépek** olyan automaták, amelyek egy program segítségével a kiinduló adatok halmazából – közbenső számításokon keresztül – a kimenő adatok halmazát előállítják és a kimenetükön megjelenítik. Az adatbevitel és a végeredmény megjelenítése között késleltetés lép fel.

Az Alexandriai Heron az i. sz. 1. században automatikus ajtónyitó szerkezetet épített, és színpadát mechanikus táncosokkal népesítette be. Már a 14. században léteztek szoborcsoportot mozgató toronyórák. A 16. századtól az európai kastélyok látogatóit hidraulikusan működtetett automaták szórakoztatták. A 18. században a mechanikusan furulyázó, táncoló, éneklő, író bábuk és állatfigurák elsősorban Angliában készültek (Anglia volt a felhúzható gépek időszakának szellemi központja).

1914-ben Torres y Quevedo sakkozó automatát szerkesztett a végjáték lejátszására: bástya és király állt az ellenfél királyával szemben.

A logika mechanizálása

A logika a gondolkodás művészete, a helyes és rendezett gondolkodás, érvelés és következtetés tana. Ez a formális meghatározás már korán felvetette azt a kérdést, hogyan lehetne a végkövetkeztetésekhez lehetőleg automatikusan eljutni.

Raimundus Lullus (1232?–1316) posztulátuma szerint a logika segítségével véges számú elv felhasználásával a teljes igazság levezethető. Megmutatta, hogy diagramokkal és forgatható korongokkal tisztán mechanikusan logikus következtetésekre lehet jutni.

Példa: Két koncentrikusan elhelyezett állítható gyűrűre 14 olyan tulajdonságot írt fel, amelyet isteninek tulajdonítottak (örökkévaló, jóságos, nagy stb.). Forgatással összesen $(14 \times 14 =) 196$ igazsághoz jutott, mint pl. az Isten örökkévaló és jóságos.

Gottfried Wilhelm Leibniz ezekből a kezdeti próbálkozásokból kifejlesztett egy logikai nyelvezetet, a *kalkulust*. Formális-deduktív logikája formális szabályok segítségével – tehát alapvetően automatikus módon – logikai következtetések levonását teszi lehetővé.

Augustus de Morgan és George Boole a 19. sz. közepén algebrai szabályok segítségével felépítette és matematizálta a logikát.

A logikai gépek

Ezek a gépek demonstrálják a logikai eljárások mechanikus kivitelezhetőségét. Elsősorban didaktikai célokat és matematikai bizonyításokat szolgálnak.

Charles Stanhope (1753–1816) szerkesztette a **demonstrator** elnevezésű eszközt. Ez egy olyan tábla volt, amelynek szélére számokat írt fel, és mozgatható vonalzók segítségével logikus következtetéseket tudott levonni.

William Jevons (1835–82) 1896-ban megkísérelte, hogy a kijelentéslogikát egy **logikai zongora (logical piano)** segítségével mechanizálja. Készüléke max. négy állítást tudott feldolgozni. A készülék belsejében emelők rudakat mozgattak el, amelyek az igazságtáblázatból eltávolították azokat a logikai értékeket, amelyek a billentyűzeten beadott feltételekkel összeegyeztethetetlenek voltak. Jevons készüléke további gépek mintájául szolgált; 1947-ben az emelőket, mutatókat és rudakat villamos elemek váltották fel.

John Venn (1834–1923) mechanikusan kimutatta a halmazelmélet és a Boole-algebra kapcsolatát. Így egyidejűleg négy Venn-diagram kiértékelése vált lehetővé.

Napjainkban a logikai gépek működését digitális számítógépek szimulálják, és ezeket elsősorban újabb logikai következtetések kutatására vagy tételek bizonyítására alkalmazzák.

A logikai gépeket tulajdonképpen csak időmegtakarítás miatt használják. Egy matematikus bármikor helyettesítheti ezeket.

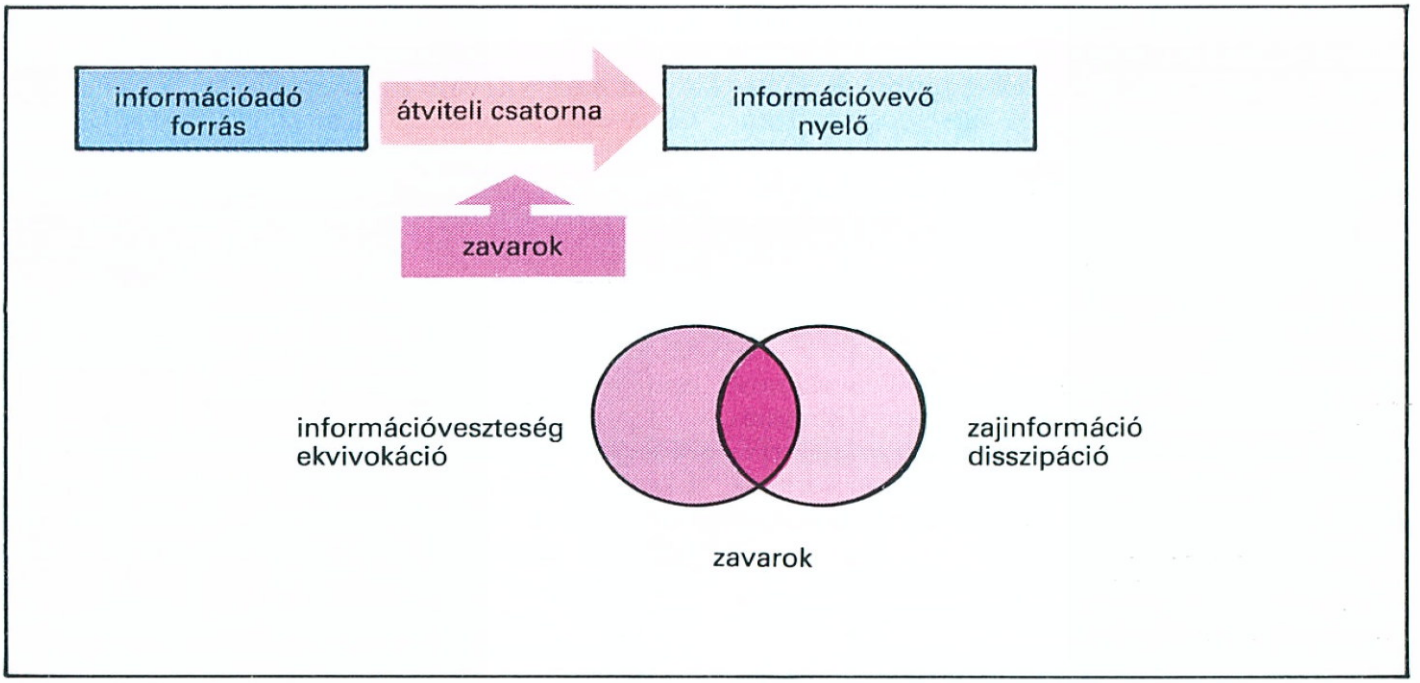
Példa: 1960-ban egy megfelelően programozott számítógép 220 matematikai tételt bizonyított be 3 perc alatt. Egy matematikusnak ehhez több napra lett volna szüksége.

A logika és a számítógépek

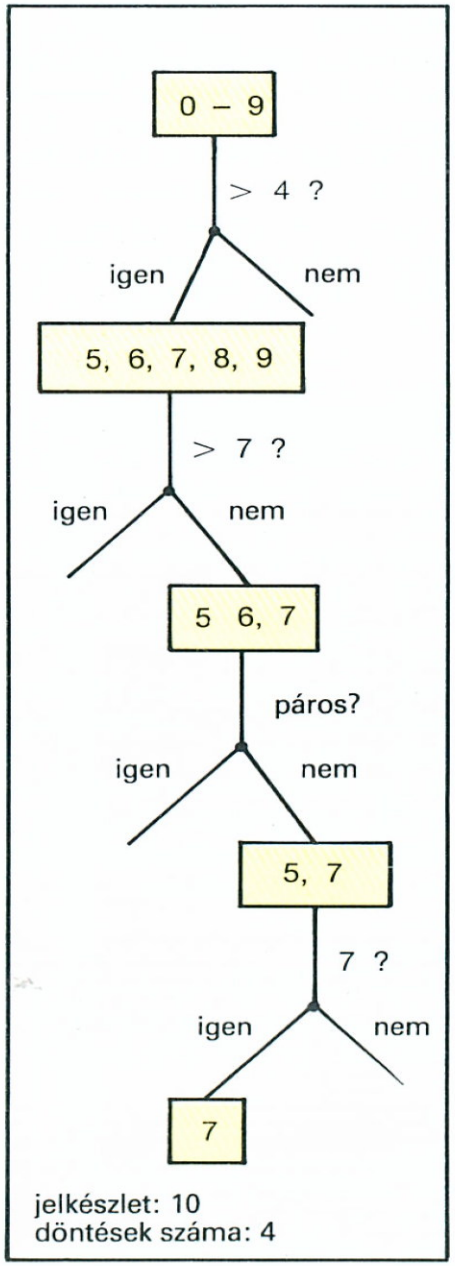
Claude Shannon (szül. 1916-ban), a matematikai információelmélet egyik megalapítója, 1943-ban logikai állításokhoz egyszerű kapcsolóállásokat rendelt (47. o.):

igaz	→	zárt
hamis	→	nyitott
És	→	soros kapcsolás
VAGY	→	párhuzamos kapcsolás

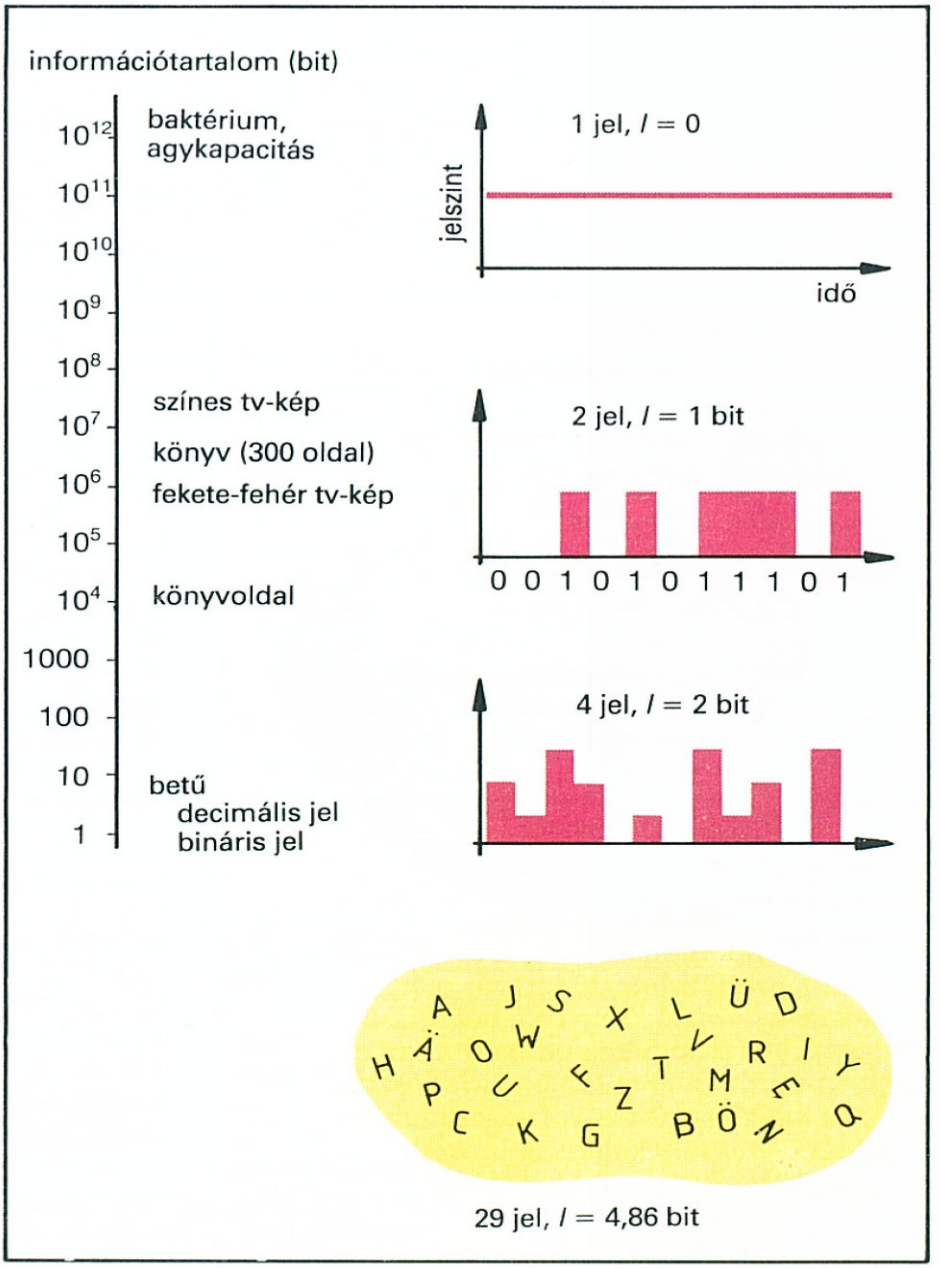
Ezzel az áramkörök elméletének alapjait sikerült megalkotnia, amely lényeges segítséget jelent a digitális számítógépek áramköreinek tervezése és egyszerűsítése során.



Információátvitel



Bináris döntések száma



Az *I* információtartalom

Az információcsere minden élő szervezet alapvető jellemzője: az információ az anyag és az energia mellett az anyagi világ egyik alapvető mennyisége.

Az **információelmélet** az információ megjelenítésével, tárolásával és az információcserével (az információ átvitelével) foglalkozik. Azonban az információt csak formálisan ábrázolható szempontból vizsgálja. Az információ jelentéstartalma tárgykörén kívül esik.

Az **információforrás** (adó) információt közvetít az **információvevőhöz** (nyelőhöz). Az átvitel az **átviteli csatornán** belül jelsorozatok segítségével történik.

Az információelmélet az üzenetek és átvitelük vizsgálatához elsősorban a valószínűségszámítás és a matematikai statisztika módszereit alkalmazza.

Az információ olyan **kódjelek** és **jelsorozatok** segítségével ábrázolható, amelyek egy véges, rendezett jelkészlet (ún. **ábécé, kód-ábécé**) elemei.

Példák: A latin ábécé, morzejelek, ASCII-kód, zászlójel-ábécé.

Az **üzenet** a (kód)jelek és állapotok előzőleg rögzített szabályok szerint összeállított véges sorozata, amely információt közvetít. A híren belül minden (kód)jel meghatározott valószínűséggel fordul elő.

A **fizikai jelek** olyan (fizikai) mennyiségek, amelyek segítségével egy kódjelkészlet (kódábécé) elemei fizikailag ábrázolhatók, létrehozhatók, és amelyek segítségével az adó és a vevő közti átvitel megvalósítható, pl. hang, villamos és optikai impulzusok.

Az **analóg** jelek időben és térben folytonosak. Az alsó és felső határ között a jel nagysága tetszőleges közbenső értéket felvehet. Analóg jelek esetén az információt a jel szintje és időtartama hordozza.

Példa: Az emberi kommunikáció hanghullámokkal.

A **digitális** jelek időtartama rövid. A jelek nagysága csak korlátozott számú – gyakran csak két – értéket vehet fel.

Példa: Egy ideg áramimpulzusai, fényimpulzusok egy üvegszálban.

A digitális jelek esetén az információt az impulzusok száma, távolsága és (esetleg) időtartama hordozza.

Példa: Morzejelek.

Az analóg jeleket digitális jelekké, a digitálisakat pedig analógokká lehet alakítani. A kisebb zavarérzékenység és egyéb technikai előnyök miatt információátvitelre a digitális jelek alkalmasabbnak bizonyulnak.

Az **információ** (latinul: *informatio,* képzés, felvilágosítás, kioktatás) a mindennapi életben tudásnyereséget jelent. Az információelméletben az információ (rövidítése: *I*) tisztán egy technikai mérték, amelyet az üzenethordozó jelsorozathoz rendelünk. Az „információ" szigorú, általános definíciója azonban még nem alakult ki.

Információtartalom

Egy jel információtartalma csak a jel előfordulásának valószínűségétől függ. Egy jelsorozat, egy ábécé vagy egy üzenet információtartalma az egyes jelek információtartalmának az összege.

Definíció:

$$I_z = \mathrm{lb}(1/p) = -\mathrm{lb}p,$$

I_z: egy jel információtartalma.

lb: 2 alapú logaritmus.

p: az a valószínűség, amellyel ez a jel előfordul.

Érvényes az alábbi összefüggés:

$$0 \leqslant p \leqslant 1.$$

Az információtartalom fordítottan arányos a valószínűség logaritmusával. Ez azokból a feltételekből következik, amelyek I definíciójához vezetnek. A 2 alapú logaritmus választásának kizárólag távközléstechnikai és számítástechnikai okai vannak.

Mivel p csak a $p = 0$ (teljes bizonytalanság, hogy a jel előfordul-e) és a $p = 1$ (bizonyosság) közötti értékeket veheti fel, az I információtartalom mindig pozitív.

Az **információtartalom egysége** a bit. Ezt nem szabad összetéveszteni a jelbittel, amely a bináris számábrázolás alapegysége. Az információs bittel ellentétben a jelbit csak egész számú lehet. A kettő közötti összefüggés: n bit formális információtartalom számítógépben történő ábrázolásához legalább n bit (jelbit) szükséges.

Egy jelkészlet egyetlen jelének (egyetlen karakterének) információtartalma azoknak az optimálisan megválasztott bináris döntéseknek a számával egyenlő, amelyekkel ez a jel (karakter) kiválasztható.

Példa: Ahhoz, hogy az A,B,C,D,E,F,G,H jelkészletből egy bizonyosat megtaláljunk, három döntésre van szükség: 1.: < D? (igen, a maradék A,B,C). 2.: > A? (igen, a maradék B,C). 3.: B? (nem, a maradék C).

Digitális jelek információtartalma. Ha a vevő mindig ugyanazt a jelet veszi, akkor $p = 1$ és $I = 0$ bit.

Információátvitel nem történik, mert biztos, hogy a jelsorozatban (az üzenetben) melyik jel fog következni. Az üzenet *üres.*

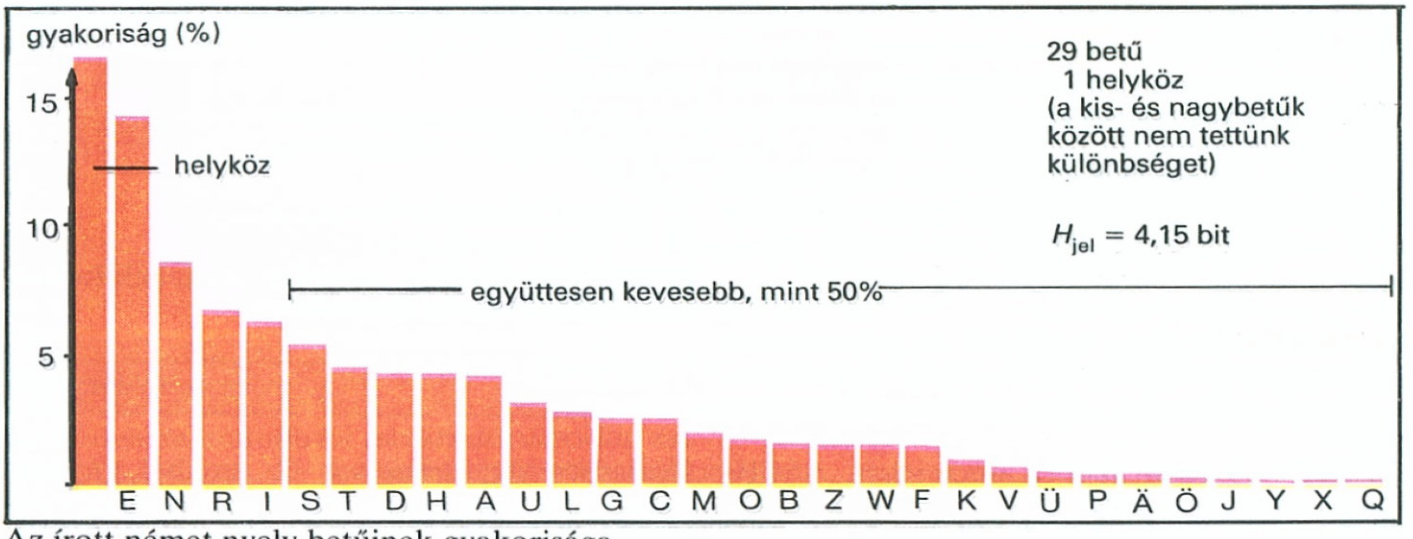
Ha az információforrás a bináris számábrázolás mindkét jelét egyforma valószínűséggel adja, tehát $p = ½$, akkor minden egyes jel esetén $I = 1$ bit. Ha a forrás n különböző jelet ad egyenlő valószínűséggel ($p = 1/n$), akkor az információtartalom minden jel esetén $I = \mathrm{lb}n$.

Csökkent redundancia esetén az olvasás sokkal nehézkesebb

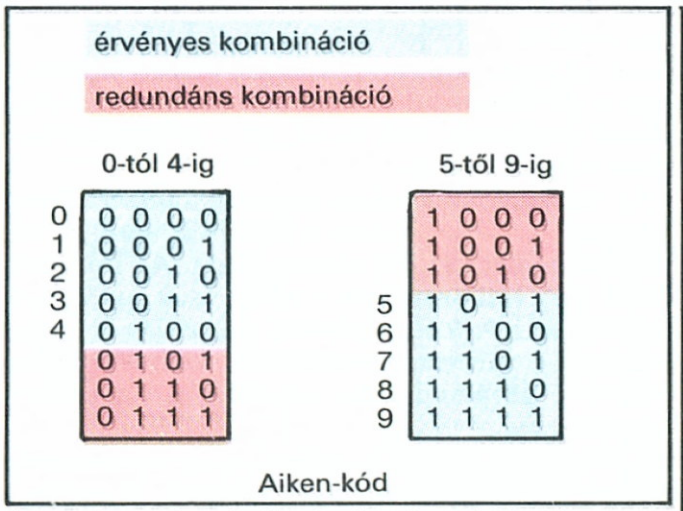
CSÖKKENT REDUNDANCIA ESETÉN AZ OLVASÁS SOKKAL NEHÉZKESEBB

CSÖKKENTREDUNDANCIAESETÉNAZOLVASÁSSOKKALNEHÉZKESEBB

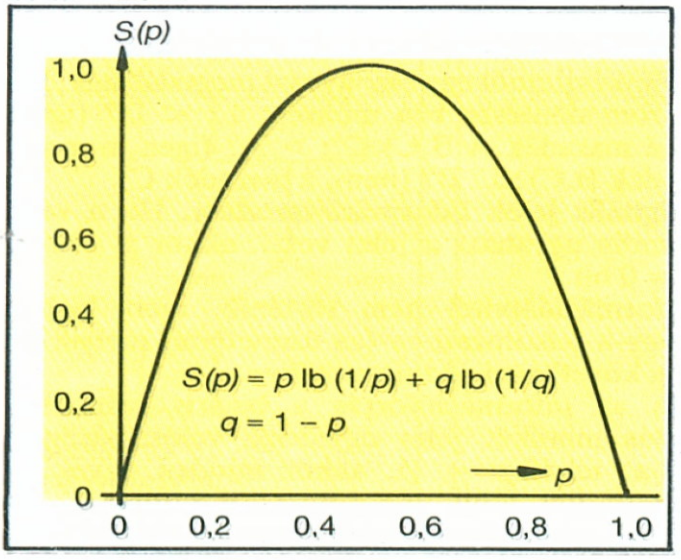
CS KK NT RE UN AN IA ES TÉ AZ OL AS SS KK LN HÉ KE EB



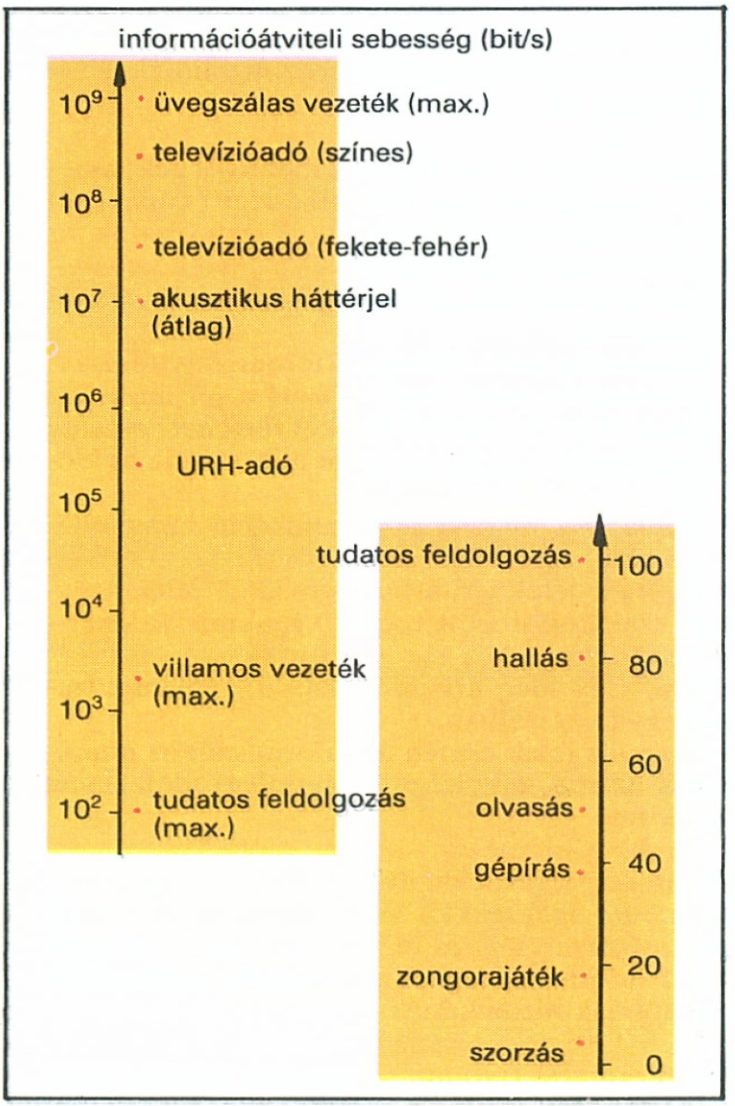
Az írott német nyelv betűinek gyakorisága



Az egész számok ábrázolásakor fellépő redundancia



Shannon-függvény



Információátviteli sebességek

A tíz decimális számjegy mindegyikének valószínűsége $p = 1/10$, így $I = 3{,}322$ bit. Tehát egy decimális számjegy információtartalma 3,322 bit. Ha egy decimális számjegyet binárisan akarunk kódolni, akkor ehhez legalább 4 bináris jegyre (vagyis 4 jelbitre) van szükség.

Példa: Az írott német nyelv információtartalma. Tételezzük fel, hogy mind a 30 jel (a betűk és a szóköz) előfordulásának valószínűsége azonos. Ekkor egy jel információtartalma $I = \text{lb}30 = 4{,}9$ bit, így bináris ábrázolásához legalább 5 (jel)bitre van szükség. Ez azonban csak hozzávetőleg igaz, mert a betűk előfordulásának valószínűsége nem azonos.

Az **analóg jelek *I* információtartalma** csak a jelek digitális átalakítása után számítható ki. Ehhez a jelet m azonos nagyságú és diszkrét értékre kell felosztani. Az analóg jel I információtartalma ekkor $\text{lb}m$.

Példa: Digitalizált telefonbeszélgetés. A hanghullámok amplitúdóját pl. max. 128 részre osztjuk fel, tehát minden amplitúdóérték 7 bináris jellel ábrázolható. Ez másodpercenként 6000-szer ismétlődik. Így tehát az 1 s időtartamú telefonbeszélgetés információtartalma legfeljebb $I = 42\,000$ információs bit.

Az **információtartalom középértéke:**

$$H = p_1 I_1 + p_2 I_2 + P_3 I_3 + \ldots + p_n I_n =$$

$$= - \sum_{i=1}^{n} p_i \, \text{lb} p_i \,,$$

ahol

H: a jelkészlethez tartozó jelek I információtartalmának átlaga bit/jel egységben,

p_i: az i-edik jel előfordulásának valószínűsége,

I_i: az i-edik jel információtartalma.

A H mennyiséget a statisztikus termodinamikában **entrópiának** nevezzük, és bit egységben adjuk meg. Az entrópia egy zárt termodinamikai rendszerben a rendezettség mértéke. A termodinamikai analógia alapján egy üzenet információtartalmát entrópiának nevezzük.

Példa: Ha egy adó bináris jeleket bocsát ki, akkor a 0 bináris számjegy p, az 1 bináris számjegy pedig $p - 1$ valószínűséggel fordul elő. A H entrópia p függvénye:

$$H(p) = p\text{lb}(1/p) + (p-1)\text{lb}(1/(p-1)).$$

A $H(p)$ függvényt **Shannon-függvénynek** nevezzük. Lefutásából látható, hogy $H(p)$-nek maximuma van a $p = ½$ értéknél, amikor a 0 és 1 bináris számjegyek előfordulásának gyakorisága azonos. Általánosan is érvényes, hogy a $H(p)$ függvény akkor éri el maximumát, ha a jelkészlet valamennyi eleme azonos valószínűséggel fordul elő.

Az írott német nyelv jeleinek átlagos információtartalma: a betűk különböző valószínűséggel fordulnak elő, pl. az „E" előfordulási valószínűsége 14,7%, a szóközé még ennél is nagyobb, 15,2%. Ha figyelembe vesszük a betűk valószínűségeloszlását, akkor $H = 4{,}15$ bit/jel. Ha azt is figyelembe vesszük, hogy egyes betűcsoportok (pl. CH, ER, EN, UNG) gyakrabban fordulnak elő, akkor $H = 1{,}6$ bit/jel (a kis- és nagybetűs írásmódot nem különböztettük meg).

Redundancia

Az adatfeldolgozásban a redundancia az üzenet információt nem tartalmazó részarányának mértéke; a redundancia egysége is bit. Bináris kód esetén a redundancia közvetlenül azt jelenti, hogy a kódolt üzenet hány bittel haladja meg a szükséges jelek minimális számát.

Példa: A német nyelvben valamennyi betű (és a szóköz) átlagos információtartalma 4,9 bit. A betűk és egyes betűcsoportok valószínűségeloszlását figyelembe véve azonban H csupán 1,6 bit. Az írott német nyelv redundanciája tehát (4,9–1,6) bit = 3,3 bit. A viszonylag magas redundancia azt jelenti, hogy egy szöveg még akkor is olvasható, ha minden második betű hiányzik.

A **Fano-módszerrel** egy adott jelsorozat (pl. betűk a rájuk vonatkozó gyakoriságeloszlás figyelembevételével) optimálisan bináris jelekké ill. jelsorozattá alakítható.

A redundancia teszi lehetővé azt, hogy egy üzenet még akkor is értelmezhető, ha az átvitel során zavarok (pl. zaj) lépnek fel – és ezek a zavarok a gyakorlatban mindig elkerülhetetlenek.

Egy kód redundanciáját rendszerint szándékosan növelik, hogy a zavarokkal szembeni érzékenységét csökkentsék.

Az információátvitel sebessége, információáramlás:

Az időegység alatt átvitt információmennyiség

$$c = I/t,$$

ahol

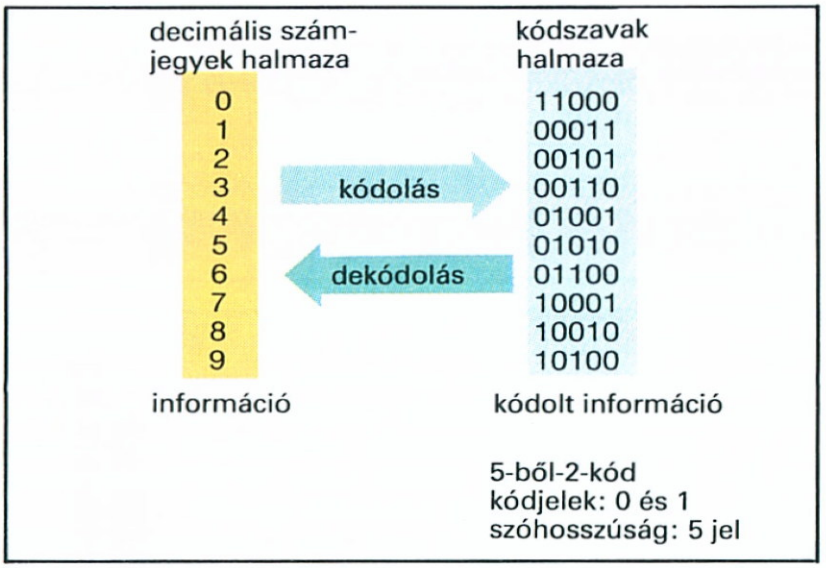
c: az információátviteli sebesség bit/s egységben,

t: az átvitel időtartama.

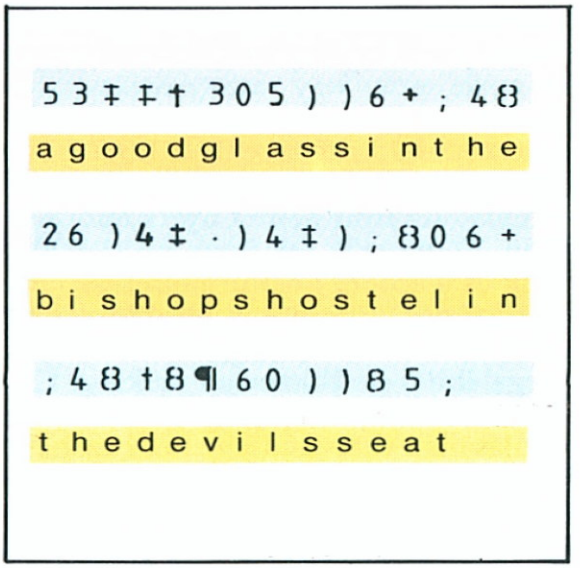
A c mennyiséget nem szabad összetéveszteni az adatátvitel sebességével, amelynek mértékegysége (jel)bit/s = baud. Néha az információátviteli sebesség egységére is a baud megjelölést használják.

Példák: fekete-fehér televízió. Minden kép 625 sorból, minden sor 800 képpontból áll, és Európában másodpercenként 25 képet sugároznak. 15 szürkeárnyalatot lehet megkülönböztetni. $c = 5 \times 10^7$ bit/s. Olvasás esetén $c = 50$ bit/s.

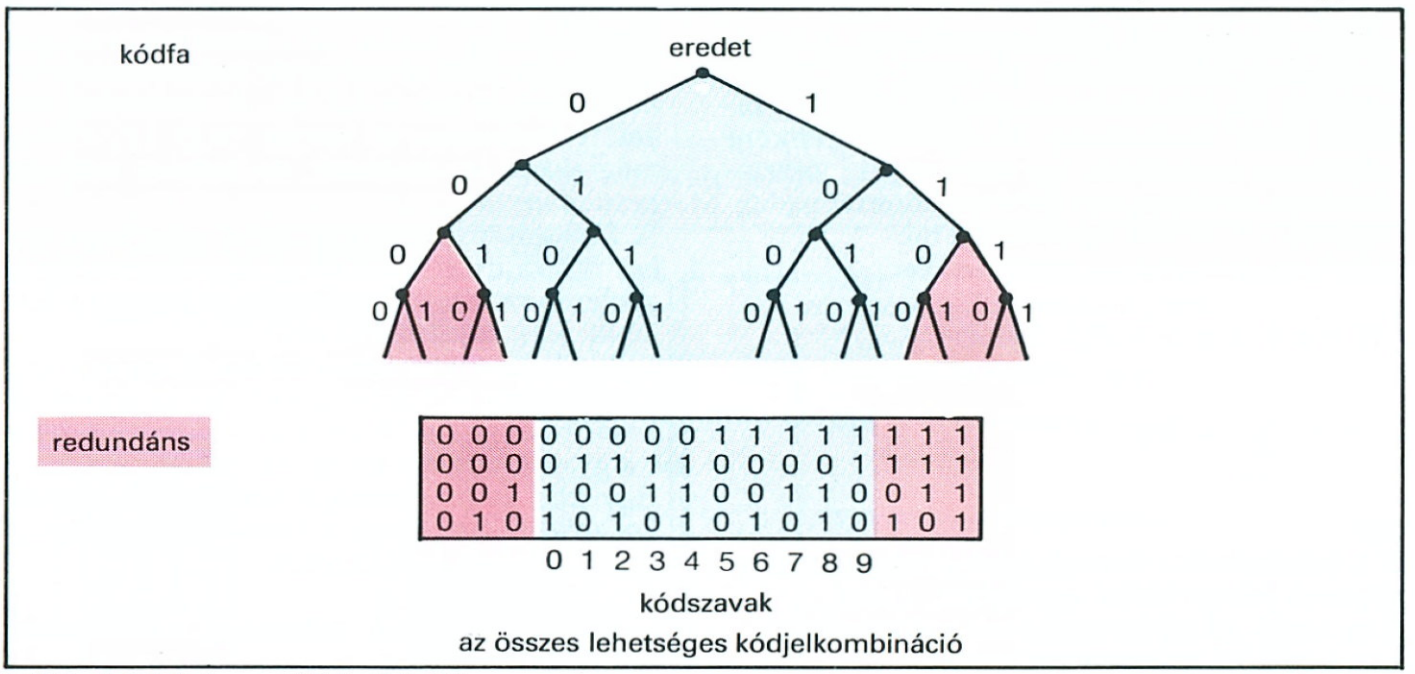
Az adatátvitel jellemzése során c nagyon fontos mennyiség. Ha például az adó nagyobb információátviteli sebességgel ad, mint amekkorát az átviteli csatorna átvinni vagy a vevő venni képes, akkor információveszteség lép fel.



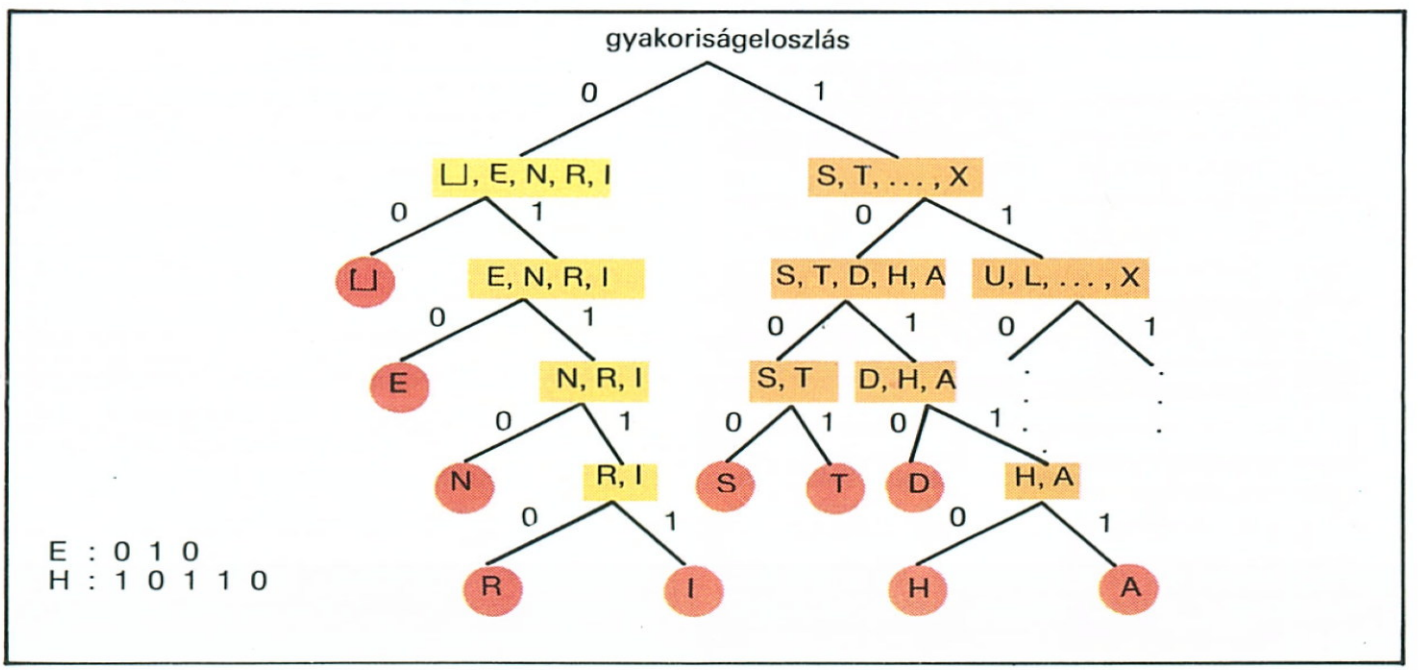
Táblázattal megadott kódolási előírás



E. A. Poe kódolása (The Goldbug)



Stibitz-kód (háromtöbbletű kód)



Az írott német nyelv egy lehetséges Fano-kódja (részlet)

A számítógéppel megoldani kívánt feladatok szöveges és decimális számokkal felírt formában vannak megadva. Ezeket az információkat azonban nem lehet a számítógéppel közvetlenül feldolgozni. A feladat megoldására szóló utasításokat programozni, majd kódolni kell, vagyis a számítógép számára érthető formában kell közölni. Ezeket a kódolási műveleteket vagy a perifériás egységek, vagy maga a számítógép végzi el.
A kódolásnak nemcsak az a feladata, hogy az elektronikus adatfeldolgozást lehetővé tegye, hanem az is, hogy az információt egyszerűen és tömören ábrázolja, egy adott információcsatornának megfelelő formára hozza, egy üzenet zavarérzékenységét csökkentse vagy az információt titkosítsa.
A kódolási problémák az információelmélet egyik fontos részterületét képezik.

A **kód** olyan utasítás, amely egy *A* halmaz jeleit egy *B* halmaz jeleihez *(kódjelekhez)* vagy jelsorozataihoz *(kódszavakhoz)* rendeli. Ennek a megfeleltetésnek egyértelműnek és megfordíthatónak kell lennie.

Példa: az 5-ből-2-kód a 4 decimális számot kölcsönösen egyértelmű módon a 01001 bináris jelsorozatra képezi le. A jelsorozat be/ki feszültségimpulzusokkal villamos kábelen át továbbítható. Mivel az egyesek összegének mindig páros számot kell adnia, a zavarok könnyen felismerhetők.

A **numerikus kódok,** mint pl. az 5-ből-2-kód, kizárólag számjegyeket képeznek le; az **alfanumerikus kódok** számjegyeket, betűket és más jeleket.
Az egyes jelek kódolása során létrejött jelsorozatot **kódszónak (szónak)** nevezzük.

Példa: Az „S" betűhöz az ASCII-kód az 10100111 jelsorozatot rendeli.

A **szóhosszúság** egy kódszóban lévő jelek száma. A kód típusától függően valamennyi kódolt jel szóhosszúsága azonos vagy változó lehet (ez utóbbi ritkább).

Példák: Az ASCII-*kód* szóhosszúsága 8 és állandó. A *Morse-kód* szóhosszúsága változó, ezért minden szó után elválasztójelnek kell következnie, ami ebben az esetben egy helyköz (szünet).

Érvényes a következő összefüggés

$$N = n^m,$$

ahol
N: a kódjelek kombinációinak maximális száma,
n: a különböző kódjelek száma,
m: a szóhosszúság.
Például az ASCII-kód esetén $n = 2$, a szóhosszúság $m = 8$, ez összesen $2^8 = 256$ különböző jelkombinációt tesz lehetővé. Ha a kód által használt kombinációk száma kisebb, mint *N,* akkor **redundáns kódról** beszélünk.

A **kódolási utasításokat kódtáblázatban** vagy fa szerkezetű **kódgráfban (kódfában)** lehet megadni. A fa szerkezetben a kódolni kívánt jelek mint levelek vannak jelen. A kódolás egészen kívül kezdődik, és a gyökér felé halad. Minden elágazásnál fel kell jegyezni a megfelelő kódjelet. A jelek megfordított sorrendje adja a kódszót. A szimmetriából adódó lehetséges, de levél nélküli ágak a redundáns kombinációkat jelentik.

Példa: Ha a Stibitz-kódfában a 7 decimális számjegy esetén követjük az elágazásokat, akkor az 1101 kódszóhoz jutunk. A kódszó tehát 1101.

Bináris kód esetén a jelkészlet mindössze két különböző jelből áll, pl. a 0 és az 1 bináris számjegyből. A kódszavak ebben az esetben ebből a két kódjelből álló sorozatok.

Példa: Az 5-ből-2-kód és a távírókód állandó szóhosszúságú bináris kódok ($m = 5$). Ez maximálisan mindössze $2^5 = 32$ kódszót tesz lehetővé, ezért egy külön jelkombináció mutatja, hogy a következő kódszavak betűket vagy számokat jelentenek-e (betű- vagy számátkapcsoló).

A számítógépekben különböző bináris kódokat alkalmaznak.

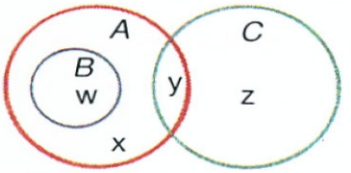
Az optimális kódok az információ kódolását a lehetséges legkevesebb jellel oldják meg, tehát redundanciájuk kicsi.

A **Fano-kód** (Robert Fano, szül. 1917.) olyan változó szóhosszúságú bináris kód, amely minden jelet az előfordulásának valószínűsége szerint rangsorol. A jelekből két csoportot állít fel, úgy, hogy a csoportba sorolt jelek valószínűségének összege (lehetőleg) megegyezzen. A jelek előbbi rangsora változatlan marad. Az egyik csoport a bináris 1 jelet kapja, a másik a bináris 0-t. Ezután minden csoportot hasonlóan két alcsoportra oszt, amelyekhez szintén a bináris 1-et és a bináris 0-t rendeli. A kódolás mindaddig folytatódik, amíg a legvégső csoportok már csak egyetlen jelből állnak. Ezzel a kódfa elkészült.

A **dekódolás** csak akkor lehet egyértelmű, ha a kód a **Fano-feltételt** kielégíti: Egyetlen kódszó sem egyezhet meg (ugyanazon a kódon belül) egy másik kódszó kezdetével. Például a bináris Morse-kód (· és –) nem teljesíti ezt a feltételt. Az egyértelmű dekódolhatóság érdekében Samuel Morse (1791–1872) harmadik jelként a szünetjelet is alkalmazta.

Egy kód **teljesítőképessége** többek között a kódolás és információfeldolgozás hatékonyságától, a zavarérzékenységtől, megtanulhatóságtól és a redundanciától függ.

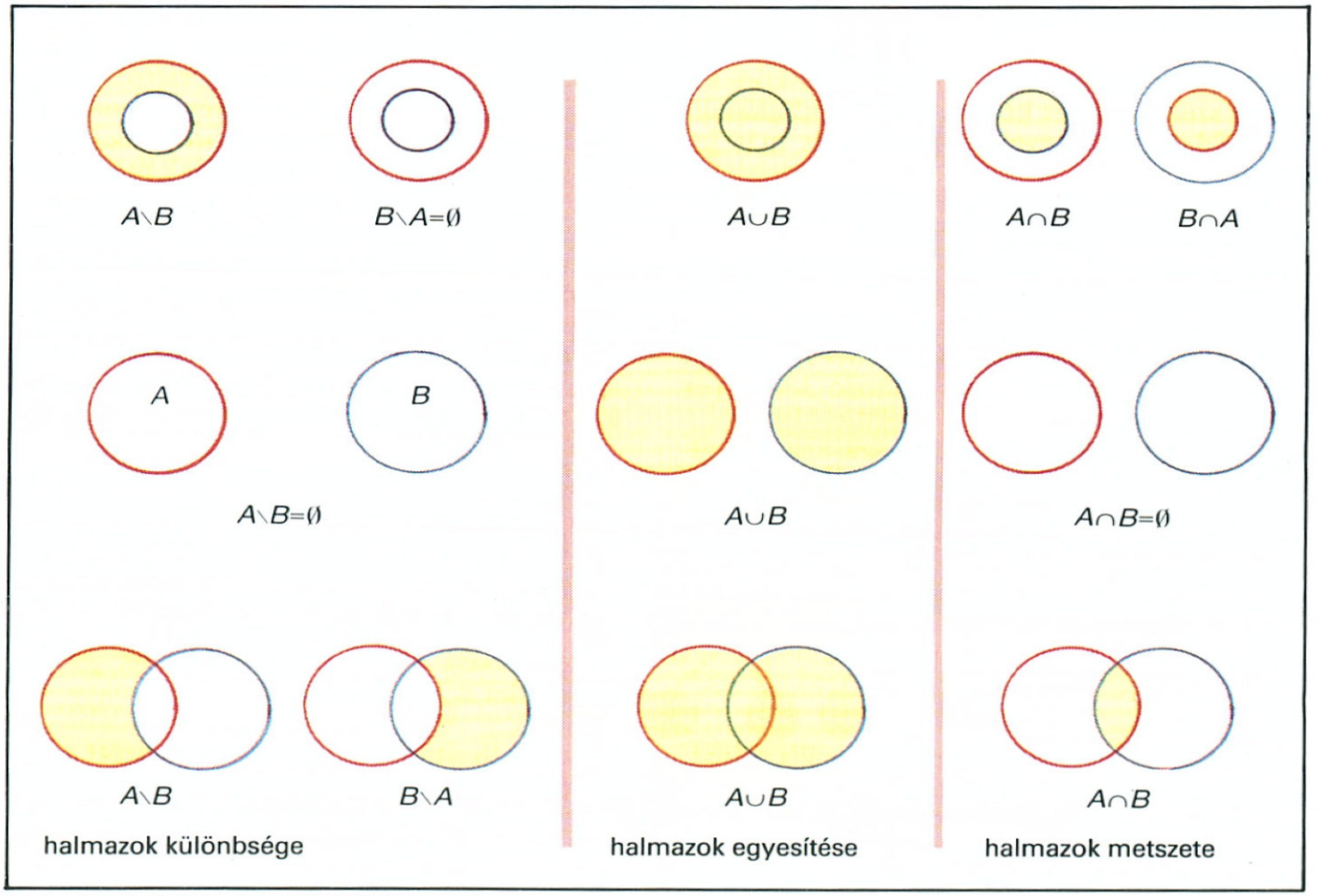
jelölés	jelentés	példa
$\in$	eleme	$x \in A$
$\notin$	nem eleme	$x \notin B$
$\{\ldots\}$	elemek halmaza	$\{w,x,y,z\}$
$\subseteq$	részhalmaza	$B \subseteq A$
$\subset$	valódi részhalmaza	$B \subset A$
$\supseteq$	tartalmazza	$A \supseteq B$
$\supset$	valóban tartalmazza	$A \supset B$
$\setminus$	különbség, maradék	$A \setminus C = \{x \mid x \in A \text{ és } x \notin C\}$
$\cup$	egyesítés	$A \cup C = \{x \mid x \in A \text{ vagy } x \in C\}$
$\cap$	metszet	$A \cap C = \{y \mid y \in A \text{ és } y \in C\}$
$\emptyset$	üres halmaz	



rövidítés:
$\{x \mid p(x)\}$
x annak a halmaznak az eleme, amelyre p(x) fennáll

A halmazelmélet matematikai szimbólumai

A halmazok jelölése



Műveletek két halmaz között

A halmazelmélet megalkotása az elmúlt évszázadban GEORG CANTOR (1845–1918) nevéhez fűződik. A szakmai körök a halmazelméletet először elutasították, de napjainkban a matematika alapvető résztudományai közé tartozik. A halmazelmélet fogalmai és következtetései magán az elméleten kívül is számos tudományterületre hatást gyakoroltak.

Alapfogalmak

Definíció: a halmaz – amelyet dőlt nagybetűkkel jelölünk – meghatározott, jól megkülönböztethető, szemléletes vagy gondolati dolgok egésszé foglalt összessége. A halmazhoz való tartozást szabályok rögzítik. A halmazba sorolt egyes dolgokat a halmaz **elemeinek** nevezzük. Az elemeket latin kisbetűkkel jelöljük, a halmazokat pedig – ha véges számú elemet tartalmaznak – elemeik kapcsos zárójelben történő felsorolásával is ábrázolhatjuk: $M = \{a_1, a_2, \ldots, a_n\}$.

Példák véges halmazokra: $A = \{1, 3, 5, 7, 9\}$ vagy $M = \{a, b, c, d, e, f, g\}$ vagy $S = \{\text{Köln, Aachen, Hamburg, London, Fokváros, Wismar}\}$.

Néhány különösen fontos halmaz nemzetközi egyezményes jele:

$\emptyset$: üres halmaz

$\mathbb{Q}$: a racionális számok halmaza

$\mathbb{R}$: a valós számok halmaza

$\mathbb{C}$: a komplex számok halmaza.

Ha x az A halmaz eleme, akkor ennek jelölése: $x \in A$. Ha x *nem* eleme az A halmaznak, akkor ennek rövidítése: $x \notin A$.

Egy véges halmaz elemeinek számát **kardinális számnak** vagy a halmaz **számosságának** nevezzük.

Egy halmaz **elemei** nem szükségszerűen egyedi dolgok; párok, hármasok stb. is alkothatnak halmazt.

Rendezett párok: Egy halmaz x_1 és x_2 elemét akkor nevezzük rendezett párnak, ha az elemek sorrendje lényeges. Ezeket az elemeket kerek zárójelbe írjuk, és érvényes rájuk a következő összefüggés: $(x_1, x_2) \neq (x_2, x_1)$.

A halmazhoz való tartozás szabályát az A halmaz x elemének esetében a következő módon jelöljük:

$$\{x \in A \mid p(x)\} \text{ vagy}$$

$$A = \{x \mid p(x)\},$$

ahol

$p(x)$: a halmazhoz való tartozás szabálya, amely az A halmaz valamennyi x elemére érvényes.

Példák: $\{x \in A \mid x$ természetes szám, amely $\leq 15\}$, tehát az A halmaz elemei a pozitív egész számok 1-től 15-ig, a 15-öt is beleértve. $\{x \in M \mid \mathbb{R}\}$, tehát az M halmaz eleme valamennyi való szám.

A halmazok felosztott szakaszokkal, VENN-diagramokkal vagy felületrészekkel szemléletesen ábrázolhatók.

Az EULER- vagy VENN-diagramok. Már a középkor logikával foglalkozó gondolkodói is diagramokat használtak a logikai összefüggések ábrázolására. Ezek rendszerbe foglalt gyűjteményét LEONHARD EULER (1707–83) jelentette meg 1768-ban. JOHN VENN 1894-ben az EULER-diagramokat módosította és a ma is érvényes alakra hozta. Ezek a VENN-diagramok olyan felületek, amelyek körbezárják a halmaz elemeit. Elsősorban olyan halmazok szemléltetésére alkalmasak, amelyek halmazok közti kapcsolatok eredményeképpen jönnek létre.

Halmazok

A **véges halmazok** meghatározott véges számú elemet, a **végtelen halmazok** pedig végtelen sok elemet tartalmaznak.

Példák: Véges halmazok: $\{0, 1, 2, 4, 6, 8, 10\}$, $\{a, b, c, d, \ldots, z\}$, $B :=$ az összes városi jogú német település. A természetes számok vagy a prímszámok végtelen halmazt képeznek.

Az **üres halmaz** $\emptyset = \{\}$ nem tartalmaz elemet.

Az A és B akkor **ekvivalens halmazok**, ha A minden egyes elemének B egy és csakis egy eleme felel meg és fordítva.

Példa: A páros természetes számok halmaza és a páratlan természetes számok halmaza ekvivalens halmazok.

Az egymással ekvivalens halmazok egy **osztályt** képeznek.

Példa: A páros számok halmaza 2-től 40-ig, tehát $\{2, 4, 6, \ldots, 40\}$ és a páratlan számok halmaza 1-től 39-ig, tehát $\{1, 3, 5, \ldots, 39\}$ egy 20 számosságú osztályt képeznek.

Két halmaz akkor **egyenlő,** ha a halmazok mindegyike ugyanazokat az elemeket tartalmazza. Az elemek sorrendje ebben az esetben nem játszik szerepet.

Példa: $A = B$, ha $A = \{1, 3, 5, 9, 7\}$ és $B = \{1, 3, 7, 5, 9\}$.

Részhalmaz. Az A halmaz akkor részhalmaza a B halmaznak, azaz rövidítve $A \subseteq B$, ha A minden eleme B-nek is eleme. Fordítva megfogalmazva: B tartalmazza A-t, azaz $B \supseteq A$.

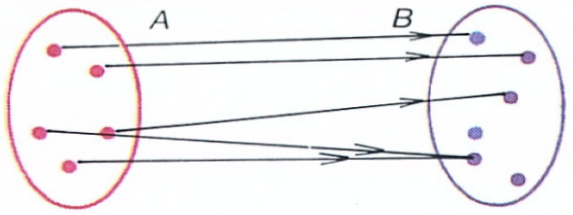
Valódi részhalmaz: $A \subset B$. A a B halmaz valódi részhalmaza, ha B-nek vannak olyan elemei, amelyek A-nak nem elemei.

Példa: $B =$ a természetes számok 12-ig. Ennek lehetséges (valódi) részhalmazai a következők: $A_1 = \{1, 3, 5, 7, 9, 11\}$, $A_2 = \{2, 4, 6, 8, 10, 12\}$, $A_3 = \{1, 2\}$ és $A_4 = \{\}$.

Hatványhalmaz. Egy A halmaz összes részhalmazainak halmazát hatványhalmaznak nevezzük: $\mathbb{P}(A)$.

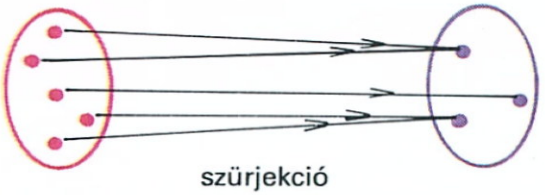
Példa: Az $A = \{1, 2, 3\}$ halmazra $\mathbb{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$.

általános leképezés

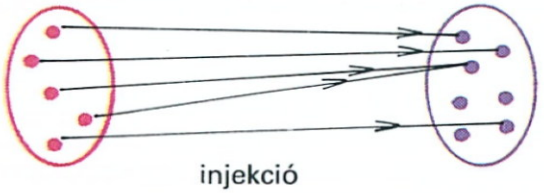


értelmezési tartomány értékkészlet

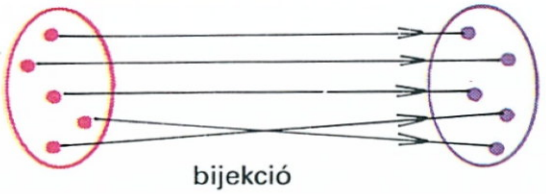
leképezési szabály: $F: A \to B$



szürjekció



injekció



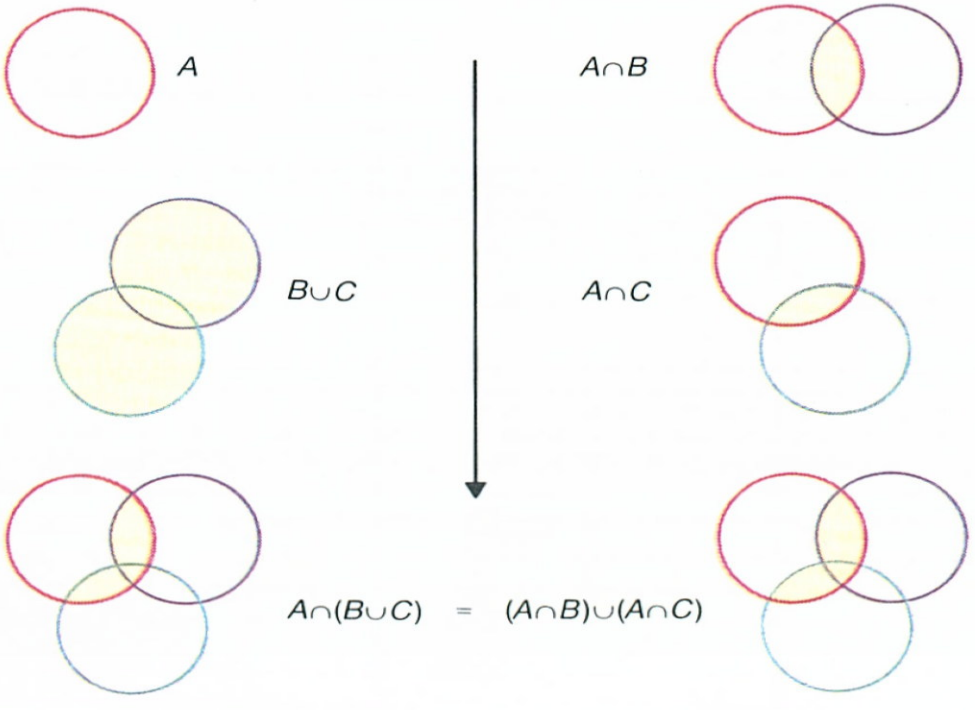
bijekció

Leképezések

$A \cup \emptyset = A$ $A \cap \emptyset = A$	
$A \cup A = A$ $A \cap A = A$	idempotencia
$A \cup B = B \cup A$ $A \cap B = B \cap A$	kommutativitás
$A \cup (A \cap B) = A$ $A \cap (A \cup B) = A$	andjunktivitás
$(A \cup B) \cup C = A \cup (B \cup C)$ $(A \cap B) \cap C = A \cap (B \cap C)$	asszociativitás
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	disztributivitás

kontinuitás:

disztributivitás (lépésről lépésre, grafikusan):



Halmazalgebra (Venn-diagramok és kapcsolatok)

Halmazalgebra

Műveletek halmazokkal

A halmazokkal végzett különböző műveletek új halmazokat eredményeznek. A műveletek halmazoperátorok segítségével végezhetők: \ (különbség), $\cup$ (egyesítés), és $\cap$ (metszet). A következőkben csak a nem-triviális eseteket vizsgáljuk, tehát amikor a műveletekben részt vevő halmazok legalább részben átfedik egymást.

Az A és B halmazok különbsége (maradékhalmaza, angolul: *difference*) azoknak az elemeknek a halmaza, amelyek A-nak elemei, B-nek azonban nem. Jelölése:

$D = A \setminus B$,

kimondva: A mínusz B.

A D különbséghalmaz minden elemére érvényes a következő összefüggés:

$\{x \mid x \in A, x \notin B\}$.

Példa: $A :=$ az összes német város és $B :=$ az összes tengerparti német város esetén $D :=$ az összes német város, amely nem a tengerparton fekszik.

A különbséghalmaz képzése nem kommunatív művelet, azaz D függ a sorrendtől:

$A \setminus B \neq B \setminus A$.

Az A halmaz **komplementer halmaza** (angolul: *complement of A*). Ha A a B halmaz valódi részhalmaza, tehát $A \subset B$, akkor a $B \setminus A$ különbséghalmazt A komplementer halmazának nevezzük rövidítve C A.

$B \setminus A = \mathrm{C}\,A$

(Az A komplementerére használt $\bar{A}$ jelölés elavult.)

Az A és a B halmazok **egyesítése** (**összege,** angolul: *union*) azoknak az elemeknek az összessége, amelyek az A vagy a B halmaznak elemei. Jelölése:

$V = A \cup B = \{x \mid x \in A \text{ vagy } x \in B\}$

kimondva: A unió B.

Példa: $V :=$ az összes német város $\cup$ az összes tengerparti város (a nem németek is) = az összes német város és az összes tengerparti város együttes halmaza.

A halmazok egyesítése kommunatív művelet:

$A \cup B = B \cup A$.

Az A és *B* halmazok **metszete** (**közös része,** angolul: *intersection*) azoknak az elemeknek az összessége, amelyek egyidejűleg mind az A, mind a B halmazhoz tartoznak. Jelölése:

$S = A \cap B = \{x \mid x \in A \text{ és } x \in B\}$

kimondva: A metszet B.

Példa: $S :=$ az összes német város $\cap$ az összes tengerparti város = az összes német tengerparti város.

A halmazok metszetének képzése kommutatív művelet:

$A \cap B = B \cap A$

Relációk

A relációk egy, illetve két vagy több halmaz elemei között fennálló összefüggést fejeznek ki. Az összes reláció a halmazok Descartes-féle szorzatán alapszik. A következőkben legfeljebb két halmaz között fennálló összefüggéseket vizsgálunk.

Descartes-féle szorzat. Az A és a B halmaz Descartes-féle szorzatának jelölése:

$A \times B$

A szorzás során keletkezett új halmaz elemei az

$A \times B = \{(a, b) \mid a \in A, b \in B\}$

rendezett elempárok. A Descartes-féle szorzat tehát olyan új halmazt eredményez, amely rendezett elempárokból áll.

A Descartes-féle szorzat képzése nem kommutatív művelet, azaz

$A \times B \neq B \times A$.

Példa: Az $A = \{a_1, a_2\}$ és a $B = \{b_1, b_2\}$ halmazokra $A \times B = \{(a_1, b_1), (a_1, b_2), (a_2, b_1), (a_2, b_2)\}$. Az új halmaz elemei rendezett párok, tehát pl. $(a_1, b_2) \neq (b_2, a_1)$.

A legegyszerűbb Descartes-féle szorzat a halmaz önmagával képezett szorzata:

$A \times A = \{(a_1, a_1), (a_1, a_2), \ldots, (a_1, a_n), (a_2, a_1), (a_2, a_2), \ldots, (a_n, a_1), (a_n, a_2), \ldots, (a_n, a_n)\}$.

A Descartes-féle szorzathalmaz minden részhalmaza egy **reláció.** Két halmaz R relációja pl.:

$R \subseteq A \times B$

Fontos reláció az I **ekvivalencia-reláció:**

$I \subseteq A \times A$,

$I = \{(a_1, a_1), (a_2, a_2), \ldots, (a_n, a_n)\}$.

Leképezés (angolul: *map*). Az A és B közti leképezések, relációk, azaz A és B halmazok Descartes-féle szorzatának részhalmazai, amelyekben minden $a \in A$ elemnek egy és csak egy $b \in B$ elem felel meg úgy, hogy $(a, b) \in f$.

Érvényes a következő összefüggés:

$f \subseteq A \times B$,

ahol

f: A és B közti leképezés

A: a leképezés értelmezési tartomány (angolul: *domain*)

B: a leképezés értékkészlete (angolul: *codomain, range*)

A leképezést három adat határozza meg:

1. a leképezési szabály: $(a, b) \in f$,
2. az értelmezési tartomány: A,
3. az értékkészlet: B.

A **leképezéseknek** különböző **típusai** ismertek, pl. a *szürjektív leképezés* (A-nak B-re történő leképzése), az *injektív leképezés* (A-nak B-be történő beágyzása) és a *bijektív leképezés* (A-nak B-re történő leképezése és B-be való beágyazása).

Példák leképezésekre: a számítógépek programozása során a kódjelsorozatok a tárolócímek injektív vagy bijektív leképezései.

$a \vee 0 = a$
$a \vee 1 = 1$
$a \vee a = a$
$a \vee \neg a = 1$
$a \vee b = b \vee a$
$a \vee (a \wedge b) = a$
$a \vee (\neg a \wedge b) = a \vee b$
$a \vee (b \vee c) = (a \vee b) \vee c$
$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

$a \wedge 0 = 0$
$a \wedge 1 = a$
$a \wedge a = a$
$a \wedge \neg a = 0$
$a \wedge b = b \wedge a$
$a \wedge (a \vee b) = a$
$a \wedge (\neg a \vee b) = a \wedge b$
$a \wedge (b \wedge c) = (a \wedge b) \wedge c$
$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

$\neg (a \vee b) = \neg a \wedge \neg b$
$\neg (a \wedge b) = \neg a \vee \neg b$

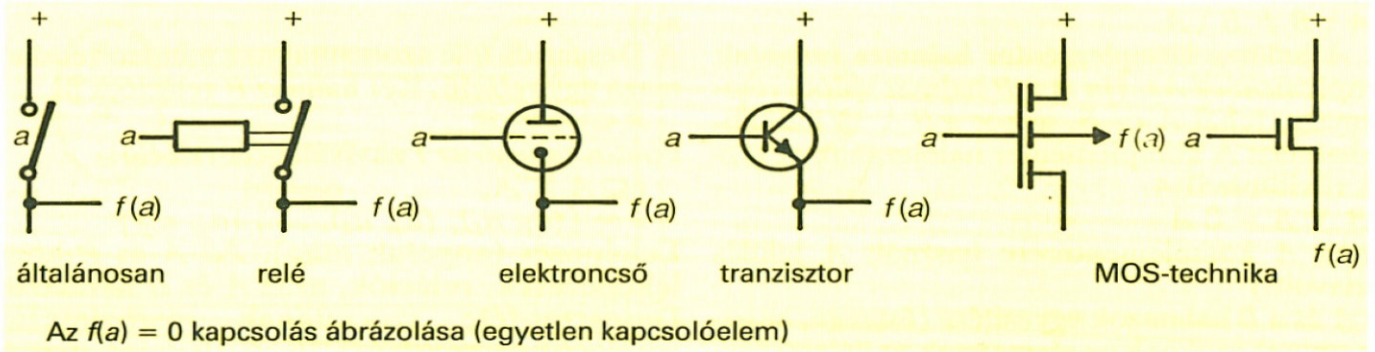
$a \bar{\vee} b = \neg (a \vee b)$
$a \bar{\wedge} b = \neg (a \wedge b)$
$a \bar{\vee} (b \bar{\wedge} c) = \neg a \wedge b \wedge c$

$a \equiv b = (a \wedge b) \vee \neg (a \vee b)$
$a \oplus b = \neg (a \equiv b)$
$a \oplus b = (\neg a \wedge b) \vee (a \wedge \neg b)$
$a \oplus b = (a \vee b) \wedge \neg (a \wedge b)$

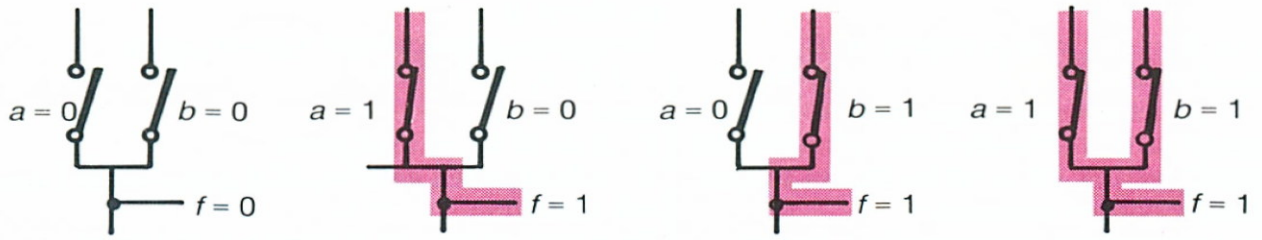
$a \rightarrow b = \neg a \vee b$

$\vee$: diszjunkció
$\wedge$: konjunkció
$\neg$: negáció
$\bar{\vee}$: NOR-funkció
$\bar{\wedge}$: NAND-funkció
$\equiv$: ekvivalencia
$\oplus$: kizáró VAGY-funkció
$\rightarrow$: implikáció

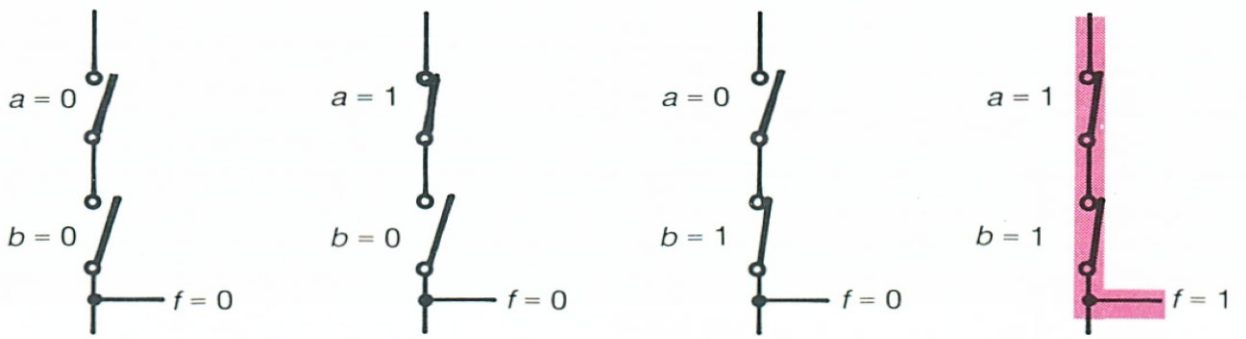
Boole-algebra összefüggései



Az $f(a) = 0$ kapcsolás ábrázolása (egyetlen kapcsolóelem)



Az $f = a \vee b$ VAGY-funkció ábrázolása (két párhuzamosan kötött kapcsoló)



Az $f = a \wedge b$ ÉS-funkció (két sorba kötött kapcsoló)

Logikai műveletek megvalósítása kapcsolókkal

A programozott számítógép különleges, felépítésének megfelelő matematikai formalizmust használ, amely az algebra egyik részterülete, az ún. **Boole-algebra**. GEORGE BOOLE (1815–64) és AUGUSTE DE MORGAN (1806–71) egymással egy időben fejlesztette ki a magasabb matematikának ezt az ágát, amelyet a programozó végső soron minden feladat megfogalmazásához felhasznál.

LEIBNIZ már 1666-ban megfogalmazta az algebra és a logika között fennálló összefüggéseket, azonban erre vonatkozó kéziratai csak a 20. század elején kerültek elő.

A modern programnyelvek részben elfedik ezt a hátteret, azonban ha a számítógép belsejében végbemenő elemi folyamatokat tekintjük, akkor a Boole-algebra szerepe világossá válik.

A Boole-algebrát Mezopotámia, az ókori Görögország és Arábia matematikusainak meggondolásai alapozták meg, de csak sokkal később keletkeztek azok a formalizmusok és összefüggések, amelyek ismerete nélkül egyetlen modern számítógép sem létezhetne. Fordítva ez természetesen nem igaz, mert a Boole-algebra számítógépek nélkül is a matematikai tudományok egyik fontos ágát képezi.

A programozáshoz a Boole-algebra elemeinek ismerete is elegendő.

Boole-algebráról akkor beszélünk, ha egy M halmaznak a matematikai logika három műveletével–a $(\wedge)$ konjunkcióval, a $(\vee)$ diszjunkcióval (adjunkcióval) és a $(\neg)$ negációval – összekapcsolt $a, b, c, \ldots$ elemei között a következő axiómák teljesülnek:

Asszociativitás

$$a \vee (b \vee c) = (a \vee b) \vee c$$
$$a \wedge (b \wedge c) = (a \wedge b) \wedge c$$

A művelet eredménye nem függ a zárójelzéstől, a műveletek sorrendje tetszőleges.

Kommutativitás

$$a \vee b = b \vee a$$
$$a \wedge b = b \wedge a$$

A műveletek eredménye független az operandusok sorrendjétől.

Abszorpciós (elnyelési) törvények

$$a \wedge (a \vee b) = a$$
$$a \vee (a \wedge b) = a$$

Disztributivitás törvénye

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$
$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

A közös művelettel kapcsolódó elem (a közös tényező) a zárójelen kívülre kiemelhető.

Ezen kívül létezik az M halmazban két kitüntetett elem: a 0 **nullelem,** és az 1 **egységelem.** A 0 és az 1 elemek a matematikai logika operátorainak vonatkozásában semleges elemek, amelyekre a következő összefüggések érvényesek:

$$a \triangle 0 = 0 \quad \text{és} \quad a \triangledown 0 = a$$
$$a \triangle 1 = 0 \quad \text{és} \quad a \triangledown 1 = 1$$

Az M halmaz minden elemének létezik a **komplementere** is, amely a következő tulajdonsággal rendelkezik:

$$a \triangle \neg a = 0 \quad \text{és} \quad a \triangledown \neg a = 1$$

A matematikai logika oprerátorainak rangsora nem egyértelműen definiált. Általában a következő sorrendet szokták alkalmazni: $\neg$, $\wedge$, $\vee$. Tehát ha a zárójelek nincsenek feltüntetve, akkor először a negáció, majd a konjunkció, végül pedig a diszjunkció műveletét végezzük el. Az egyértelműség hiánya miatt ajánlatos mindig zárójeleket alkalmazni.

Példák: Az aritmetikai algebrában az $a + b \cdot c$ kifejezés kiszámításakor mindig a szorzást kell először elvégezni. Tehát a $(b \cdot c) + a$ sorrend érvényes, mert az aritmetikai operátorok rangsorában a szorzás megelőzi az összeadást. A Boole-algebrában az $a \wedge b \vee c$ kifejezés nem egyértelmű, mert

$$(a \wedge b) \vee c \neq a \wedge (b \vee c).$$

A logikai operátorok megállapodás szerinti rangsora alapján általában az $(a \wedge b) \vee c$ sorrend érvényes, azonban teljes egyértelműséget csak a zárójelek használata biztosít.

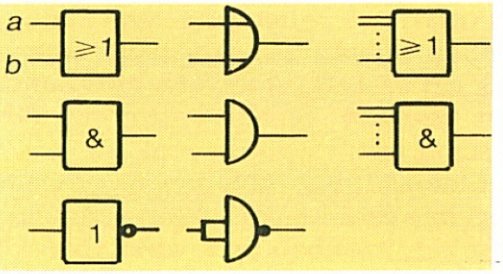
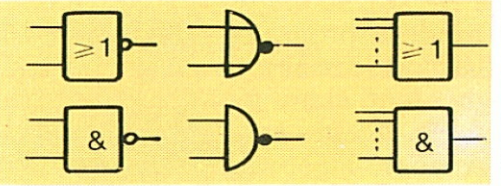
Egy M halmaz elemei között fennálló logikai kapcsolatokat az informatikában igazságtáblázatokkal ábrázoljuk. (Az igazságtáblázat a függvények táblázatos megadási módjának a logikai függvények esetében alkalmazott változata.)

Kapcsolásalgebra

A **kapcsolásalgebra** a két különböző elemet tartalmazó halmazra alkalmazott speciális Boole-algebra. A halmaz két eleme egy elektronikus hálózat két lehetséges („ki" és „be") állapotát reprezentálja. Ennek a kettes számrendszerben a 0 és 1 bináris értékek felelnek meg.

Az elektronikus áramkörök tervezésekor a megoldani kívánt probléma Boole-algebrai megfogalmazása nagyon hasznos, sőt nélkülözhetetlen, ha bonyolult kapcsolásokat kell egyszerűsíteni.

Mihelyt az optikai számítógép fejlődése kezdeti szakaszán túljut, a programozásnak nem kell többé a kapcsolásalgebra szabályait követni, mert pl. az optikai kapcsolóelemek stabil állapota nem csak kettőre korlátozódik.

kapcsolat	jelölés DIN		angolul	a megfelelő kapuáramkör kapcsolási jele (elavult / kettőnél több bemenet esetén)
diszjunkció (adjunkció)	$a \vee b$	*a* VAGY *b* $a + b$	*a* OR *b*	
konjunkció	$a \wedge b$	*a* ÉS *b* $a \cdot b$ ab	*a* AND *b*	
negáció	$\neg a$	NEM *a* $\bar{a}$	NOT *a*	
NOR művelet	$a \bar{\vee} b$	*a* NOR *b* $a \downarrow b$	*a* NOR *b*	
NAND művelet (exklúzió)	$a \bar{\wedge} b$	*a* NAND *b* $a \mid b$	*a* NAND *b*	
ekvivalencia	$a \equiv b$	$a \Leftrightarrow b$	$a \equiv b$	
KIZÁRÓ VAGY művelet (antivalencia)	$a \oplus b$	$a \not\equiv b$	$a \not\equiv b$	
implikáció	$a \rightarrow b$	$a \Rightarrow b$	$a \supset b$	

Az *a* és a *b* elem közötti Boole-függvények (logikai kapcsolatok)

VAGY függvény

a	*b*	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

ÉS függvény

a	*b*	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

NEM függvény

a	$\neg a$
0	1
1	0

NOR függvény

a	*b*	$a \bar{\vee} b$
0	0	1
0	1	0
1	0	0
1	1	0

NAND függvény

a	*b*	$a \bar{\wedge} b$
0	0	1
0	1	1
1	0	1
1	1	0

implikáció

a	*b*	$a \rightarrow b$
0	0	1
0	1	1
1	0	0
1	1	1

ekvivalencia

a	*b*	$a \equiv b$
0	0	1
0	1	0
1	0	0
1	1	1

KIZÁRÓ VAGY függvény

a	*b*	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Boole-függvények igazságtáblázatai

A Boole-algebrából levezetett kapcsolásalgebrában a 0 és az 1 kapcsolási állapotok között lehetséges logikai műveletekre a következő összefüggések érvényesek:

VAGY ($\vee$)

0 VAGY 0 = 0 ill. $0 \vee 0 = 0$
0 VAGY 1 = 1 ill. $0 \vee 1 = 1$
1 VAGY 0 = 1 ill. $1 \vee 0 = 1$
1 VAGY 1 = 1 ill. $1 \vee 1 = 1$

A fenti összefüggéseknek az alábbi számítási szabályok felelnek meg:

1. Ha a két kapcsolási elem egyikét (x, amely tehát 0 vagy 1 lehet) a VAGY operátor az 1 elemmel kapcsolja össze, akkor az eredmény mindig 1, azaz
   x VAGY 1 = 1 és 1 VAGY x = 1
2. Ha az x kapcsolási elemet VAGY művelet kapcsolja össze a 0 kapcsolási elemmel, akkor az eredmény ismét x, azaz
   x VAGY 0 = x és 0 VAGY x = x
3. Ha az x kapcsolási elemet VAGY művelet kapcsolja össze komplementerével ($\neg x$), akkor az eredmény mindig 1, azaz
   x VAGY $\neg x$ = 1
4. Ha bármelyik x kapcsolási elemet VAGY művelet kapcsolja össze önmagával, akkor az eredmény ismét x, azaz
   x VAGY x = x

ÉS ($\wedge$)

0 ÉS 0 = 0 ill. $0 \wedge 0 = 0$
0 ÉS 1 = 0 ill. $0 \wedge 1 = 0$
1 ÉS 0 = 0 ill. $1 \wedge 0 = 0$
1 ÉS 1 = 1 ill. $1 \wedge 1 = 1$

Ezeknek az összefüggéseknek az alábbi számítási szabályok felelnek meg:

x ÉS 1 = x és 1 ÉS x = x
x ÉS 0 = 0 és 0 ÉS x = 0
x ÉS x = x

NEM ($\neg$)

NEM 0 = 1 ill. $\neg 0 = 1$
NEM 1 = 0 ill. $\neg 1 = 0$

Ezeknek az összefüggéseknek az alábbi számítási szabályok felelnek meg:

x ÉS NEM 1 = 0, valamint
NEM 1 ÉS x = 0
x ÉS NEM x = 0
x VAGY NEM x = 1

Az a elem negációját a komplementerének is nevezik.

A kapcsolásalgebra minden kifejezése leírható a NEM, a VAGY, valamint az ÉS operátorokkal vagy ezek valamilyen kombinációjával.

Két operátor **dualitása** fontos tulajdonság a kapcsolóhálózatok gyakorlati tervezése során: egy duális kapcsolási elem az ellenpárjából a polaritások felcserélésével keletkezik.

Duális operátorok: az ÉS-VAGY, a NAND-NOR és az EKVIVALENCIA-ANTIVALENCIA párok.

Példa: Ha egy ÉS kapu be- és kimenő jeleinek komplementerét vesszük, akkor az áramkör a VAGY kapunak megfelelően viselkedik.

Igazságtáblázatok. A műveletek eredményei egyszerű szorzótáblázatokhoz hasonlóan ábrázolhatók, pl. az a VAGY b esetre:

VAGY	0	1
0	0	1
1	1	1

A logikai műveleteket általában olyan igazságtáblázatokban ábrázolják, amely az összes lehetséges kombinációkat feltünteti a megfelelő eredményekkel együtt. A táblázat pl. a VAGY b esetre a következő:

a	b	a VAGY b
0	0	0
0	1	1
1	0	1
1	1	1

A kapcsolásalgebra kifejezéseinek átalakítása mindig óvatosságot igényel, ezért akár a „felesleges" zárójeleket is érdemes feltüntetni.

A feltételes utasítások kiértékelése ugyancsak a Boole-algebra szabályai szerint történik. A műveleti táblázatok ennek során közvetlenül felhasználhatók annak figyelembevételével, hogy a bináris 1 elem az „igaz" Boole-állandónak, a 0 elem pedig a „hamis" Boole-állandónak felel meg.

Tételek

A következő két tétel a hosszabb logikai kifejezések egyszerűsítésére szolgál.

De Morgan tétele

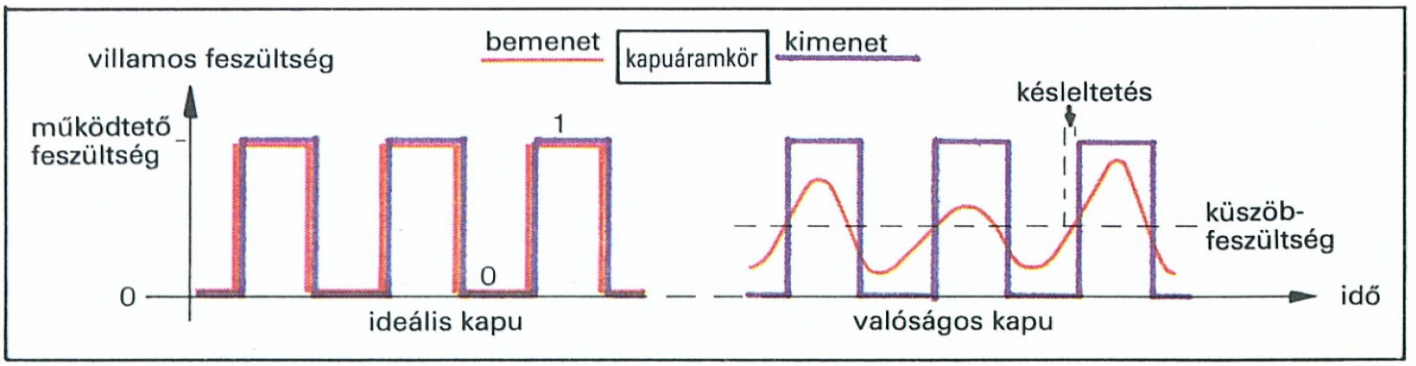
Egy egyszerű (csak ÉS illetve csak VAGY operátort tartalmazó) logikai művelet komplementere megkapható, ha az ÉS/VAGY utasításokat VAGY/ÉS utasítással, a műveletben szereplő egyes elemeket komplementerükkel cseréljük fel:

NEM (a ÉS b ÉS c ÉS ...)
= NEM a VAGY NEM b VAGY
NEM c VAGY NEM ...
NEM (a VAGY b VAGY c VAGY ...)
= NEM a ÉS NEM b ÉS NEM c
ÉS NEM ...

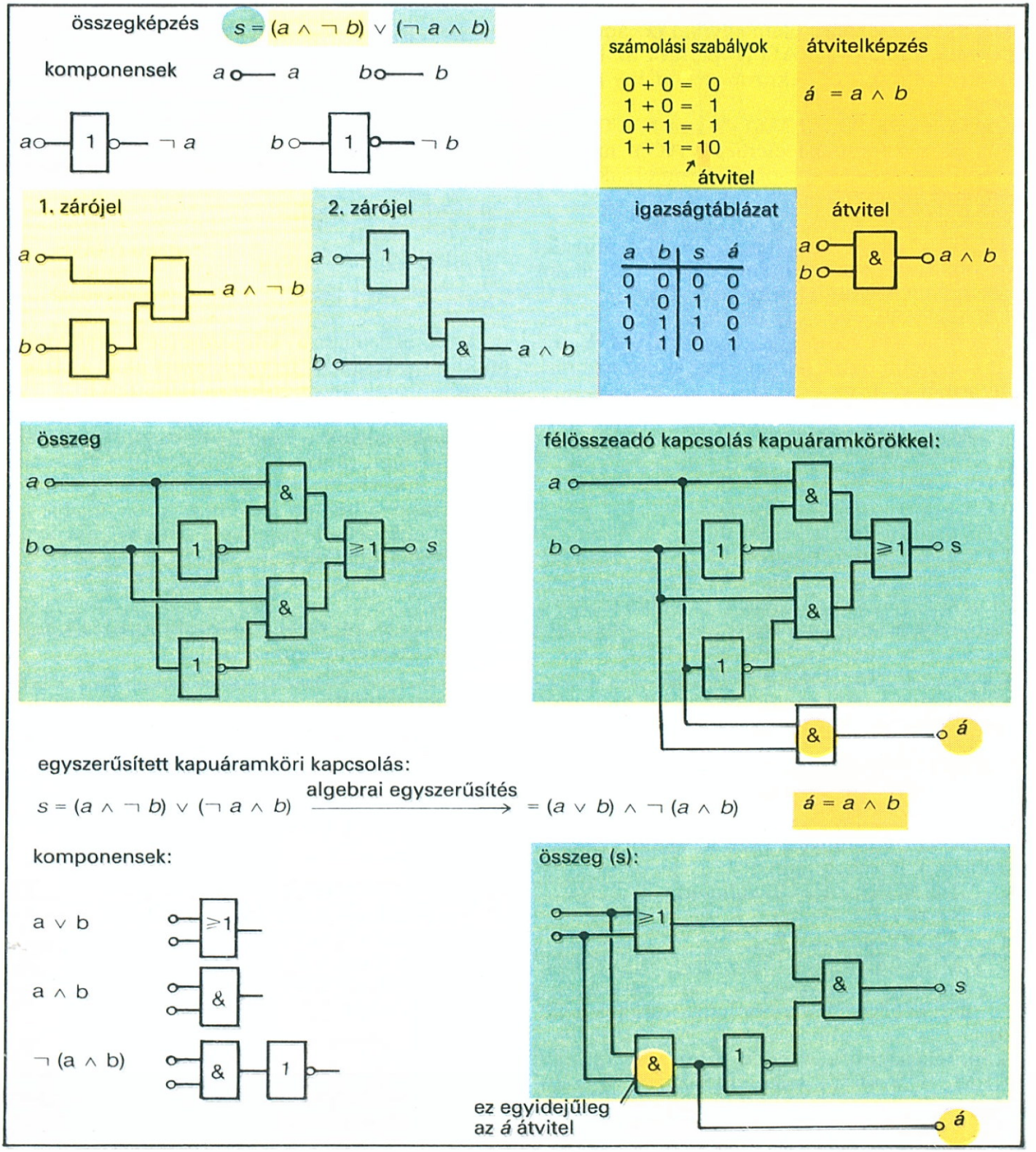
Shannon-tétel

Egy vegyes (ÉS illetve VAGY operátort is tartalmazó) logikai kifejezés komplementere a zárójelezés megtartásával De Morgan tétele segítségével kapható meg, tehát pl.

NEM ((a VAGY b) ÉS c)
= (NEM a ÉS NEM b) VAGY NEM c



Kapuáramkörök jelviszonyai



Félösszeadó

Példa: összeadó-áramkör egyszerűsítése.
Az *a*, *b* és *c* bináris jegyek összeadása során keletkező átvitel a (NEM *a* ÉS *b* ÉS *c*) VAGY (*a* ÉS NEM *b* ÉS *c*) VAGY (*a* ÉS *b* ÉS NEM *c*) VAGY (*a* ÉS *b* ÉS *c*) függvénnyel számítható ki. A kapcsolóhálózatokban ennek megvalósításához 4 elektronikus alkatrészre van szükség.
Az ÉS-kapcsolat miatt a következő összefüggés érvényes:

a ÉS *b* ÉS *c* = (*a* ÉS *b* ÉS *c*) VAGY (*a* ÉS *b* ÉS *c*) VAGY (*a* ÉS *b* ÉS *c*).

A kifejezés utolsó tagját a kapcsolási függvénynek megfelelően behelyettesítjük, majd valamennyi (*a* ÉS *b* ÉS *c*) tagot összevonjuk. Ekkor pl. az első tagból a következőt kapjuk:

(NEM *a* ÉS *b* ÉS *c*) VAGY (*a* ÉS *b* ÉS *c*),

kiemelés után

(*b* ÉS *c*) VAGY (NEM *a* VAGY *a*).

Mivel az első zárójeltől jobbra eső kifejezés értéke 1, az eredmény

(*b* ÉS *c*).

Ha ezt a másik két taggal is elvégezzük, a kapcsolási függvényre a következő kifejezés adódik:

(*a* ÉS *b*) VAGY (*b* ÉS *c*) VAGY (*a* ÉS *c*).

Az eredmény: az összeadás átvitelét megadó kifejezésben már csak három tag szerepel. Minden tagnak egy-egy elektronikus alkatrész felel meg. Tehát a szükséges kapcsolóelemek számát a kapcsolásalgebra segítségével sikerült 4-ről 3-ra csökkenteni.

A gyakorlatban további szempontok is szerepet játszanak: pl. a térigény és a jelek futási ideje olyan járulékos tényezők, amelyeket egyidejűleg optimalizálni kell.

Kapcsolásalgebra és kapcsolási elemek

Kapcsolók

A 0 és 1 bináris értékek technikailag egy egyszerű bistabil kapcsolóelemmel, pl. egy flipfloppal megvalósíthatók. A nyitott kapcsoló a 0, a zárt kapcsoló pedig az 1 állapotot jelenti.
A *kapcsoló* kifejezést itt a szó legtágabb értelmében használjuk. A legegyszerűbb esetben ez egy kézi kapcsoló, amely egy áramkört szakít meg (0) vagy zár (1).

Zuse első számítógépéhez villamosan működtetett telefonreléket használt. Később az elektroncsövek és tranzisztorok az egyes állapotok között lényegesen gyorsabb váltást tettek lehetővé. A modern számítógépekben **MOS**-technikával (angolul: ***M**etal **O**xide **S**emiconductor*) készült félvezetőket alkalmaznak. Az integrált áramkörök egy kb. 5 mm^2 felületű chipen több, mint 10^6 kapcsolóelemet tartalmaznak.

A modern kapcsolások a 0 és 1 állapotok megvalósítására két különböző nagyságú villamos feszültségszintet alkalmaznak jelként. A kapcsolóelem villamos feszültsége tehát akkor sem egyenlő nullával, ha az elem a 0 állapotban van. A gyakorlatban a jelek nagyon rövid ideig tartó feszültségimpulzusok.

Kapuáramkörök

Kapunak nevezzük a számítógépben a bináris jelek feldolgozásához használt legkisebb bistabil kapcsolóelemeket. Ezek a digitális áramköri technika alapvető építőelemei. Az **ideális kapu** az áramkörben a 0 bináris értéket a kapu kimenetén 0 volt feszültséggel jeleníti meg, az 1 bináris jelnek pedig a hálózat üzemi feszültsége felel meg. Az ideális kapu időbeli késleltetés nélkül kapcsol át.

Ezzel ellentétben a **valóságos kapu** bemenetén nullánál nagyobb küszöbfeszültségnek, az ún. *átkapcsolási feszültségnek* kell jelen lennie ahhoz, hogy a kimeneten az 1 jel (vagyis a teljes üzemi feszültség) megjelenjen. A 0 jelnek ekkor a küszöbfeszültségnél kisebb villamos feszültség felel meg. A valódi kapuáramkör kapcsolási ideje – nagyon kicsi – véges érték, amelyet **késleltetési időnek** nevezünk. Pl. a MOS-technikával készült kapuáramkörök késleltetési ideje kisebb, mint 10^{-9} s.

Minden kapuáramkör kapcsolókból (flipflopokból) tevődik össze, azonban mégsem osztható fel kapcsolókra. Az egyes kapuáramkörök összekapcsolásával megvalósíthatók a kapcsolásalgebra által kínált lehetőségek.

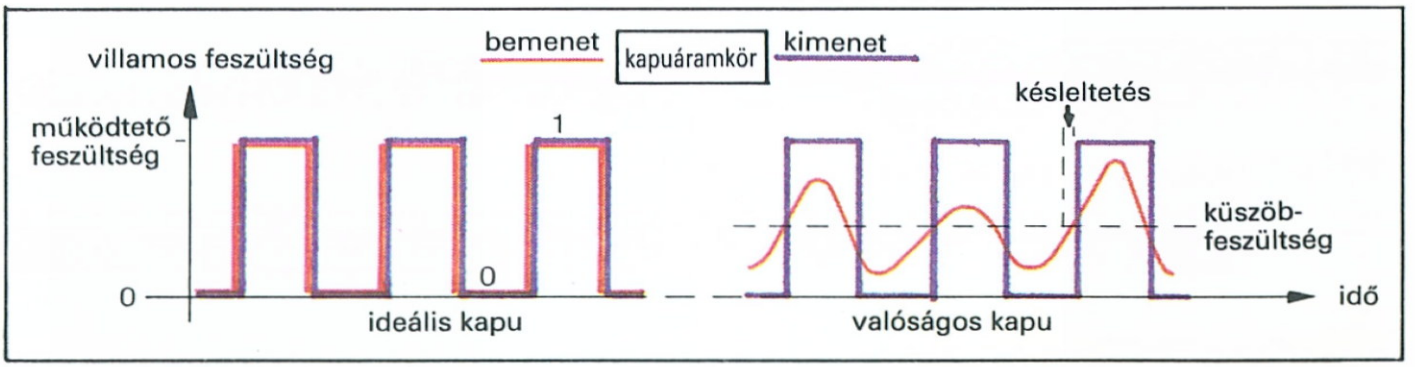
A **VAGY-kapuk** csak egyetlen kimenettel, de legalább két bemenettel rendelkeznek. A kimenőjel akkor és csak akkor 1, ha legalább az egyik bemenőjel 1, különben a kimenőjel 0. A VAGY-kapu a legegyszerűbb esetben a következő logikai kapcsolatot (Boole-függvényt) valósítja meg:

$f = a$ VAGY b ill. $f = a \vee b$

f: kimenőjel.
a, b: bemenőjel az a ill. a b bemeneten.
A megfelelő igazságtáblázat az a és b bemenőjelek lehetséges kombinációihoz tartozó f kimenőjel értékeit tartalmazza.

Példák: $f = 1$, ha $a = 1$ és $b = 0$. $f = 0$, ha $a = b = 0$.

A VAGY-kapu legegyszerűbben két párhuzamosan kötött kapcsolóval (flipfloppal) valósítható meg. Ha a két áramköri elem bármelyike zárva van, az áramkörben folyhat áram.



Kapuáramkörök jelviszonyai

összegképzés $s = (a \wedge \neg b) \vee (\neg a \wedge b)$

komponensek a — a b — b

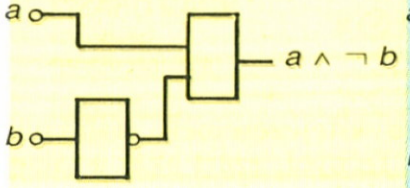
számolási szabályok

$0 + 0 = 0$
$1 + 0 = 1$
$0 + 1 = 1$
$1 + 1 = 10$
↗ átvitel

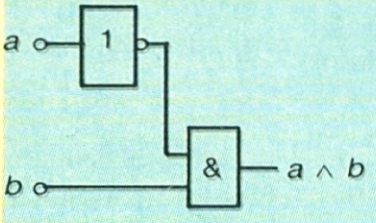
átvitelképzés

$\acute{a} = a \wedge b$

1. zárójel



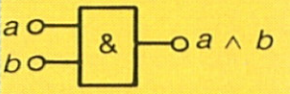
2. zárójel



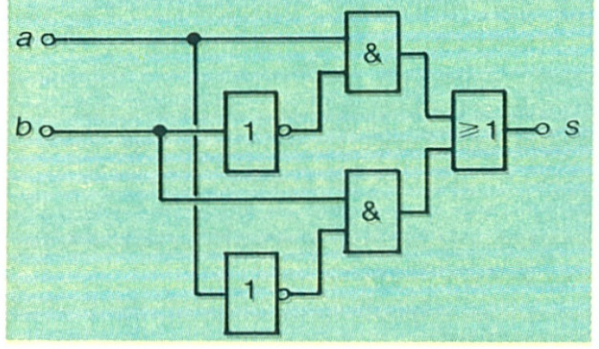
igazságtáblázat

a	b	s	$\acute{a}$
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

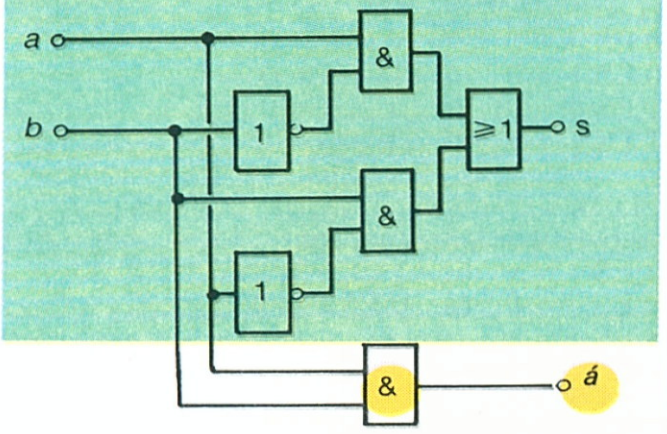
átvitel



összeg



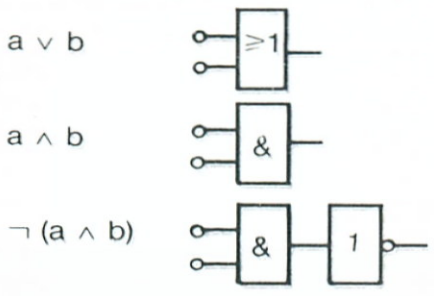
félösszeadó kapcsolás kapuáramkörökkel:



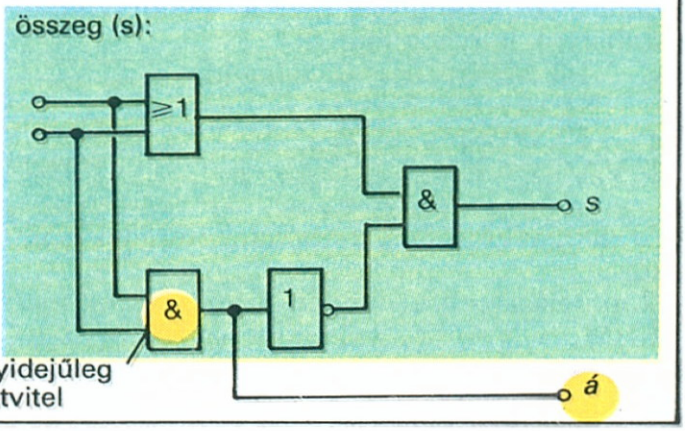
egyszerűsített kapuáramköri kapcsolás:

$s = (a \wedge \neg b) \vee (\neg a \wedge b)$ —algebrai egyszerűsítés→ $= (a \vee b) \wedge \neg (a \wedge b)$ $\quad \acute{a} = a \wedge b$

komponensek:



összeg (s):



Félösszeadó

Példa: összeadó-áramkör egyszerűsítése.
Az *a*, *b* és *c* bináris jegyek összeadása során keletkező átvitel a (NEM *a* ÉS *b* ÉS *c*) VAGY (*a* ÉS NEM *b* ÉS *c*) VAGY (*a* ÉS *b* ÉS NEM *c*) VAGY (*a* ÉS *b* ÉS *c*) függvénnyel számítható ki. A kapcsolóhálózatokban ennek megvalósításához 4 elektronikus alkatrészre van szükség.
Az ÉS-kapcsolat miatt a következő összefüggés érvényes:

a ÉS *b* ÉS *c* = (*a* ÉS *b* ÉS *c*) VAGY (*a* ÉS *b* ÉS *c*) VAGY (*a* ÉS *b* ÉS *c*).

A kifejezés utolsó tagját a kapcsolási függvénynek megfelelően behelyettesítjük, majd valamennyi (*a* ÉS *b* ÉS *c*) tagot összevonjuk. Ekkor pl. az első tagból a következőt kapjuk:

(NEM *a* ÉS *b* ÉS *c*) VAGY (*a* ÉS *b* ÉS *c*),

kiemelés után

(*b* ÉS *c*) VAGY (NEM *a* VAGY *a*).

Mivel az első zárójeltől jobbra eső kifejezés értéke 1, az eredmény

(*b* ÉS *c*).

Ha ezt a másik két taggal is elvégezzük, a kapcsolási függvényre a következő kifejezés adódik:

(*a* ÉS *b*) VAGY (*b* ÉS *c*) VAGY (*a* ÉS *c*).

Az eredmény: az összeadás átvitelét megadó kifejezésben már csak három tag szerepel. Minden tagnak egy-egy elektronikus alkatrész felel meg. Tehát a szükséges kapcsolóelemek számát a kapcsolásalgebra segítségével sikerült 4-ről 3-ra csökkenteni.

A gyakorlatban további szempontok is szerepet játszanak: pl. a térigény és a jelek futási ideje olyan járulékos tényezők, amelyeket egyidejűleg optimalizálni kell.

Kapcsolásalgebra és kapcsolási elemek

Kapcsolók

A 0 és 1 bináris értékek technikailag egy egyszerű bistabil kapcsolóelemmel, pl. egy flipfloppal megvalósíthatók. A nyitott kapcsoló a 0, a zárt kapcsoló pedig az 1 állapotot jelenti.
A *kapcsoló* kifejezést itt a szó legtágabb értelmében használjuk. A legegyszerűbb esetben ez egy kézi kapcsoló, amely egy áramkört szakít meg (0) vagy zár (1).

Zuse első számítógépéhez villamosan működtetett telefonreléket használt. Később az elektroncsövek és tranzisztorok az egyes állapotok között lényegesen gyorsabb váltást tettek lehetővé. A modern számítógépekben **MOS**-technikával (angolul: **M***etal* **O***xide* **S***emiconductor*) készült félvezetőket alkalmaznak. Az integrált áramkörök egy kb. 5 mm^2 felületű chipen több, mint 10^6 kapcsolóelemet tartalmaznak.

A modern kapcsolások a 0 és 1 állapotok megvalósítására két különböző nagyságú villamos feszültségszintet alkalmaznak jelként. A kapcsolóelem villamos feszültsége tehát akkor sem egyenlő nullával, ha az elem a 0 állapotban van. A gyakorlatban a jelek nagyon rövid ideig tartó feszültségimpulzusok.

Kapuáramkörök

Kapunak nevezzük a számítógépben a bináris jelek feldolgozásához használt legkisebb bistabil kapcsolóelemeket. Ezek a digitális áramköri technika alapvető építőelemei. Az **ideális kapu** az áramkörben a 0 bináris értéket a kapu kimenetén 0 volt feszültséggel jeleníti meg, az 1 bináris jelnek pedig a hálózat üzemi feszültsége felel meg. Az ideális kapu időbeli késleltetés nélkül kapcsol át.

Ezzel ellentétben a **valóságos kapu** bemenetén nullánál nagyobb küszöbfeszültségnek, az ún. *átkapcsolási feszültségnek* kell jelen lennie ahhoz, hogy a kimeneten az 1 jel (vagyis a teljes üzemi feszültség) megjelenjen. A 0 jelnek ekkor a küszöbfeszültségnél kisebb villamos feszültség felel meg. A valódi kapuáramkör kapcsolási ideje – nagyon kicsi – véges érték, amelyet **késleltetési időnek** nevezünk. Pl. a MOS-technikával készült kapuáramkörök késleltetési ideje kisebb, mint 10^{-9} s.

Minden kapuáramkör kapcsolókból (flipflopokból) tevődik össze, azonban mégsem osztható fel kapcsolókra. Az egyes kapuáramkörök összekapcsolásával megvalósíthatók a kapcsolásalgebra által kínált lehetőségek.

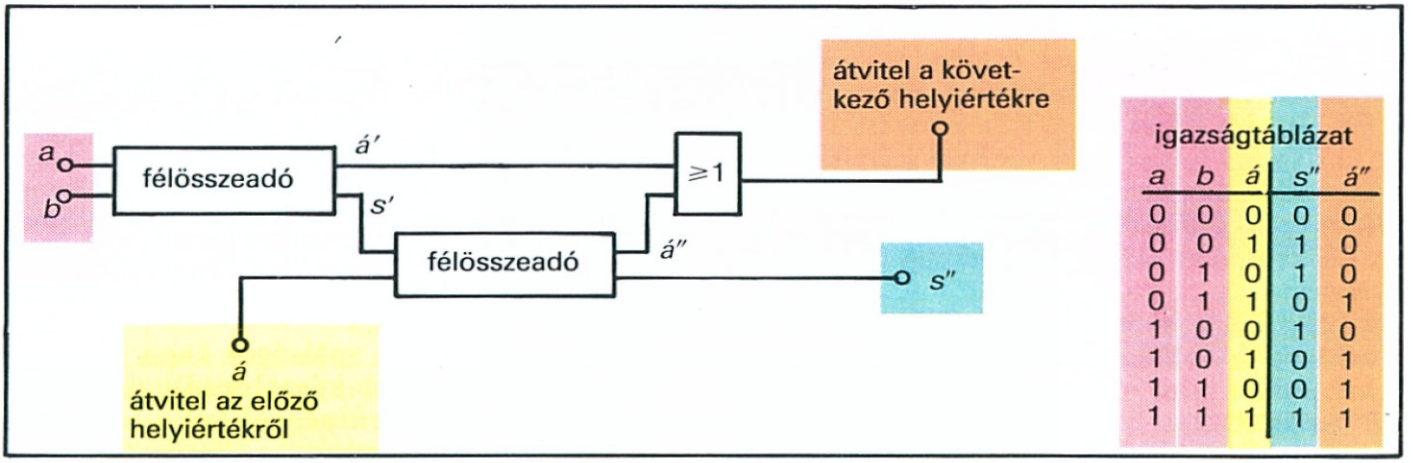
A **VAGY-kapuk** csak egyetlen kimenettel, de legalább két bemenettel rendelkeznek. A kimenőjel akkor és csak akkor 1, ha legalább az egyik bemenőjel 1, különben a kimenőjel 0. A VAGY-kapu a legegyszerűbb esetben a következő logikai kapcsolatot (Boole-függvényt) valósítja meg:

$f = a$ VAGY b ill. $f = a \vee b$

f: kimenőjel.
a, *b*: bemenőjel az *a* ill. a *b* bemeneten.
A megfelelő igazságtáblázat az *a* és *b* bemenőjelek lehetséges kombinációihoz tartozó *f* kimenőjel értékeit tartalmazza.

Példák: $f = 1$, ha $a = 1$ és $b = 0$. $f = 0$, ha $a = b = 0$.

A VAGY-kapu legegyszerűbben két párhuzamosan kötött kapcsolóval (flipfloppal) valósítható meg. Ha a két áramköri elem bármelyike zárva van, az áramkörben folyhat áram.



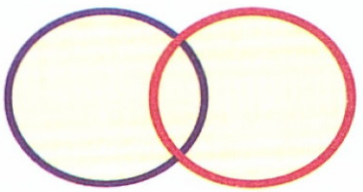
igazságtáblázat

a	b	á	s''	á''
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

Teljes összeadó megvalósítása

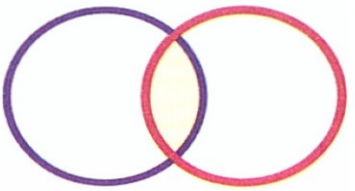
$\neg a$	$= a \barwedge a$	$= a \barvee a$
$a \vee b$	$= (a \barwedge a) \barwedge (b \barwedge b)$	$= (a \barvee b) \barvee (a \barvee b)$
$a \wedge b$	$= (a \barwedge b) \barwedge (a \barwedge b)$	$= (a \barvee a) \barvee (b \barvee b)$

A NAND és NOR helyettesítheti a NEM, a VAGY valamint az ÉS műveleteket



egyesítés $A \cup B$

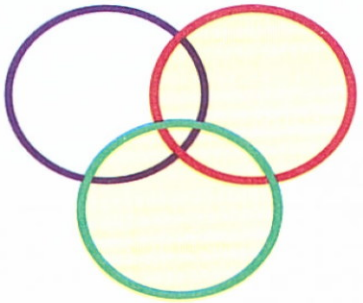
diszjunkció $a \vee b$

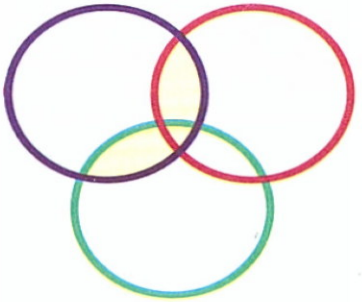


metszet $A \cap B$

konjunkció $a \wedge b$

Venn-diagram





disztributivitás törvénye:

$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

A halmazelmélet és a Boole-algebra között fennálló megfelelések

A VAGY-kapu a kapcsolásalgebra egyik (közvetlen) eleme. A VAGY Boole-függvényt, tehát a diszjunkciót valósítja meg.

Az **ÉS-kapunak** egy kimenete, és legalább két bemenete van. A kimenőjel akkor és csakis akkor 1, ha valamennyi bemenőjel (egyidejűleg) 1, különben a kimenőjel 0. Az ÉS-kapu a legegyszerűbb esetben a következő logikai kapcsolatot (Boole-függvényt) valósítja meg:

$f = a$ ÉS b ill. $f = a \wedge b$

f: kimenőjel.
a, b: bemenőjel az a ill. a b bemeneten.
A megfelelő igazságtáblázat az a és b bemenőjelek lehetséges kombinációihoz tartozó f kimenőjel értékeit tartalmazza.

Példák: $f = 1$, ha $a = b = 1$. $f = 0$, ha $a = b = 0$ vagy $a \neq$ b.

Az ÉS-kapu legegyszerűbben két sorbakötött (egymás után kötött) kapcsolóval (flipfloppal) valósítható meg. Az áramkörben csak akkor folyhat áram, ha mindkét áramköri elem zárt állapotban van.

Az ÉS-kapu a kapcsolásalgebra (oszthatatlan) alapeleme. Az ÉS Boole-függvényt, tehát a konjunkciót valósítja meg. Az ÉS-kaput egy speciális szimbólummal jelöljük.

A **NEM-kapu** csak egyetlen kimenettel és csak egyetlen bemenettel rendelkezik. A kimenőjel a bemenőjel negáltja (komplementere). A kimeneten akkor jelenik meg az 1 jel, ha a bemenőjel 0, és fordítva. A NEM-kapu a következő logikai kapcsolatot (Boole-függvényt) valósítja meg

$f =$ NEM a ill. $f = \neg$ a

f: kimenőjel.
a: bemenőjel.
Az igazságtáblázat ebben az esetben különösen egyszerű: $f = 1$, ha $a = 0$. $f = 0$, ha $a = 1$.

A NEM-kaput legegyszerűbben egy félvezető inverterkapcsolással, pl. egy pnp-tranzisztorral lehet megvalósítani.

A NEM-kapu a kapcsolásalgebra (oszthatatlan) alapeleme. A NEM Boole-függvényt, tehát a negációt (komplementerképzést) valósítja meg. A NEM-kaput speciális szimbólummal jelöljük.

A VAGY-, az ÉS- ill. a NEM-kapukkal a problémák megoldásakor felmerülő valamennyi logikai összefüggés megvalósítható. A kapuáramkörök ekkor a kapcsolásalgebra elemeit reprezentálják, és felhasználásukkal a kapcsolásalgebra szabályai szerint kapcsolóhálózatok építhetők fel.

Példa: **Félösszeadó.** Kiszámítja két bináris számjegy összegét. Félösszeadónak nevezik, mert – a teljes összeadókkal ellentétben – a lehetséges előző helyiértékről történő átvitelt nem veszi figyelembe. A félösszeadókat a *Neumann-összeadóműben* alkalmazzák, amely két szám összegét gyorsabban számolja ki, mint a soros, de lassabban, mint a párhuzamos összeadók.

A számolási szabályoknak megfelelően a félösszeadásnál átvitel léphet fel, ezt az összeg számjegyeitől elkülönítve kell végrehajtani. Az $s = a + b$ összegre és az $á$ átvitelre vonatkozó igazságtáblázat a következő:

a	b	s	$á$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Ebből a következő két kapcsolási függvény határozható meg:

$s = (a \triangle \neg b) \triangledown (\neg a \triangle b)$
$á = a \triangle b$

Két NEM-kapuval, három ÉS-kapuval és egy VAGY-kapuval megfelelően kialakított kapcsolóhálózatban mindkét egyenlet ábrázolható.

A **NOR-kapu** (angolul: **N***ot* **OR**) a kapcsolásalgebra egy további építőeleme. A VAGY függvény negáltjaként viselkedik. A következő összefüggés érvényes:

a NOR $b =$ NEM (a VAGY b)

Kizárólag NOR-kapuk felhasználásával tetszőleges logikai kapcsolóhálózat építhető fel.

A **NAND-kapu** (angolul: **N***ot* **AND**) ugyancsak a kapcsolásalgebra építőeleme. Az ÉS függvény negáltjaként működik. A következő összefüggés érvényes:

a NAND $b =$ NEM (a ÉS b)

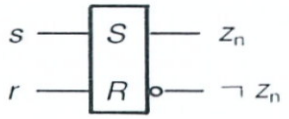
Kizárólag NAND-kapuk felhasználásával tetszőleges logikai kapcsolóhálózat felépíthető.

Az **XOR-kaput** (angolul: *e***X***clusive* **OR**) gyakran a számítógépes grafikában, valamint az információ kódolása során alkalmazzák.
Az alábbi igazságtáblázat szerint működik:

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

kapcsolási jelölések:

bemenet flipflop kimenet



$z_{n+1} = s \vee (\neg r \wedge z_n)$

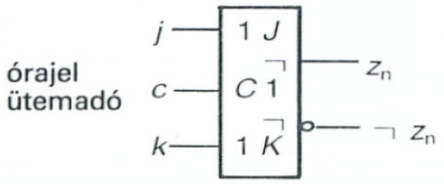
mellékfeltétel:
$r \wedge s = 0$

RS-flipflop

igazságtáblázat:

bemenet | tárolt állapot | következő állapot

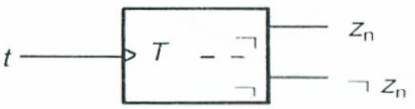
r	s	z_n	z_{n+1}	$\neg z_{n+1}$	
0	0	0	0	1	tárolás
0	0	1	1	0	
0	1	0	1	0	beírás
0	1	1	1	0	
1	0	0	0	1	törlés
1	0	1	0	1	
1	1	0	?	?	(határozatlan)
1	1	1	?	?	



$z_{n+1} = (j \wedge \neg k) \vee (\neg j \wedge k)$

JK-flipflop

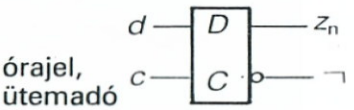
j	k	z_n	z_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



$z_{n+1} = (t \wedge \neg z_n) \vee (\neg t \wedge z_n)$

T-flipflop

t	z_n	z_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

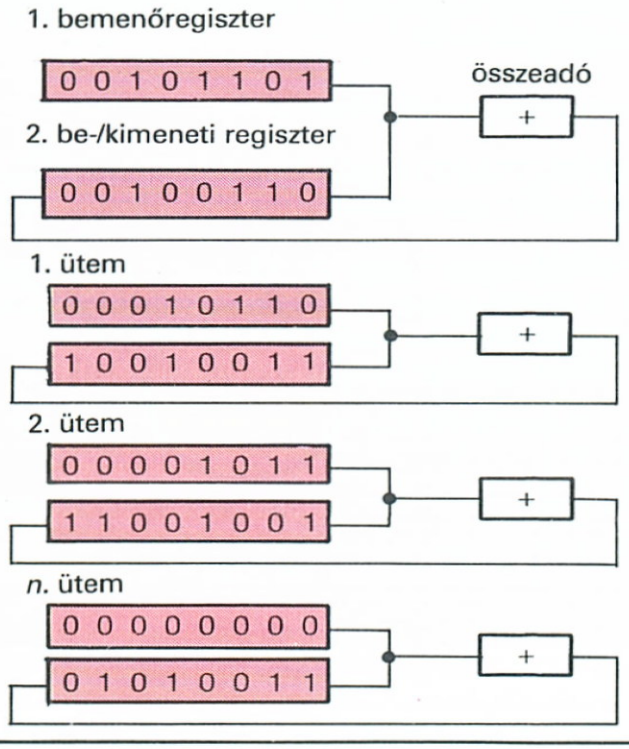


$z_{n+1} = d$

D-flipflop (késleltetés)

d	z_n	z_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Bistabil áramkörök (flipflopok)



	léptető regiszter bináris	decimális
a	0 0 0 0 0 1 1 0	$a = 6$
$2a$	0 0 0 0 1 1 0 0	$2^1\ a = 12$
$4a$	0 0 0 1 1 0 0 0	$2^2\ a = 24$
$a/2$	0 0 0 0 0 0 1 1	$a/2^1 = 3$

szorzás/osztás

Léptetőregiszterekkel megvalósított összeadás, szorzás és osztás

A kapcsolásalgebra formalizmusa időtől független formát tükröz, vagyis a jelek feldolgozására látszólag egy időben kerül sor. Figyelmen kívül marad, hogy az egyes kapcsolási elemeknek nagyon rövid, de végleges működési idejük van, és a jelek terjedése is bizonyos időt igényel. Tehát késleltetési idők lépnek fel, és az egymást szorosan követő műveleteket nem lehet egyidejűleg végrehajtani.

Példa: Az átvitelnek megfelelő jel a félösszeadót csak azután hagyja el, miután a megfelelő helyiértékek összeadása befejeződött. A következő helyiértékek összeadása késik, mert az átvitelnek megfelelő jel beérkezésére várni kell.

A **sorrendi** vagy **szekvenciális hálózatok** (angolul: *sequential circuits* vagy *networks*) kapuáramkörök és tárolóelemekből (flipflopokból) állnak.

A **szinkron szekvenciális hálózatok** a bennük lévő valamennyi kapcsolóelemet egységes órajellel látják el és így küszöbölik ki a késleltetéssel kapcsolatos problémákat. Minden elemi kapcsolási folyamat két órajel között játszódik le. Az órajel alatt minden kapcsolóelem állapota konstans marad. Ha pl. két helyiérték összeadása az n-edik és az $(n+1)$-edik órajel között történik, akkor az átvitel az $(n+1)$-edik és az $(n+2)$-edik órajel között éri el a következő helyiértéket.

Órajelnek (időzítőjelnek) nevezzük azt a jelet, amely a számítógépben végbemenő folyamatokat szinkronizálja. Az órajel „triggereli" a kapcsolási folyamatokat. Az órajelet egy óragenerátor állítja elő. A mikroprocesszorok órajelének frekvenciája 4,77 MHZ-től több, mint 100 MHz-ig terjed. Ez a frekvencia a mikroprocesszor működési sebességének egyik jellemző adata.

Abból a célból, hogy az elvégzett operációk eredménye a következő órajel után rendelkezésre álljon, tároló kapcsolóelemekre van szükség, Egy bistabil kapcsolás a 0 vagy 1 állapotot mindaddig megőrzi, amíg ez az állapot egy újabb utasítás hatására meg nem változik.

A **flipflop** bistabil kapcsolási elem, amelynek két stabil állapota van, és a számítógépekben gyors 1 bites tárolóként alkalmazzák.

Minden flipflop alapja az **RS-flipflop** (angolul: **R***eset*-**S***et flip-flop*). Ez két kölcsönösen visszacsatolt NOR-kapuból áll, amely két (r és s) bemenettel és két kimenettel rendelkezik. Az RS-flipflop pillanatnyi tárolóállapota a legutóbbi órajel alatti állapottól függ. Az s-bemenet az új (0 vagy 1) állapot beállítására szolgál, amely az első kimeneten megjelenik. Az r-bemenet törli a RS-flipflopot. A második kimenet az első kimenet negáltja. Hátránya: ha mindkét bemenőjel = 1, akkor a kimenet definiálatlan.

A **JK-flipflopnak** három bemenete van. A j-bemenet beállítja, a k-bemenet törli a flipflopot, a c-bemenetre pedig az órajel kapcsolódik. A JK-flipflopnak két kimenete van, amelyek egymás negáltjai. Ha $j = k = 1$, akkor a JK-flipflop állapota megváltozik. A szinkronizált szekvenciális hálózatok esetén az órajelet fogadó bemeneteket és vezetékeket a kapcsolási rajzokon legtöbbször nem jelölik. A JK-flipflopot elsősorban számlálókban és léptetőregiszterekben alkalmazzák. A gyakorlatban a JK-flipflop két (*master* és *slave*) részből áll: az elsőt az órajel felfutó, a másodikat a lefutó éle vezérli. Ez határozza meg a lépések sorrendjét és így szinkronizálatlanság nem lép fel. Az ilyen elrendezést **master-slave-flipflopnak** nevezzük.

A **T-flipflopnak** csak egy bemenete, de két kimenete van. Úgy működik, mint egy összekötött bemenetű JK-flipflop. A T-flipflop bemenetét minden bemenőjel megváltoztatja.

Egy T-flipflop és egy RS-flipflop öszekapcsolása egy JK-flipflopot eredményez.

A **regiszterek** nagyon gyors – és viszonylag drága – tárolóelemek, amelyek rendszerint flipflopokból állnak. Egy regiszter flipflopjainak p számát a regiszter **szóhosszúságának** nevezzük, amely 4 bittől 64 bitig terjed. Néhány mikroprocesszor regiszterei kétszeres szóhosszúság elérése érdekében összekapcsolhatók. Egy p flipflopot tartalmazó regiszter 2^p bináris számjegyet tárol. A mikroprocesszorokban a regiszterek elsősorban részeredmények és a gyakran használt számértékek tárolására szolgálnak. A 80386-os mikroprocesszornak pl. hét szabadon hozzáférhető, 32 bit szóhosszúságú regisztere van.

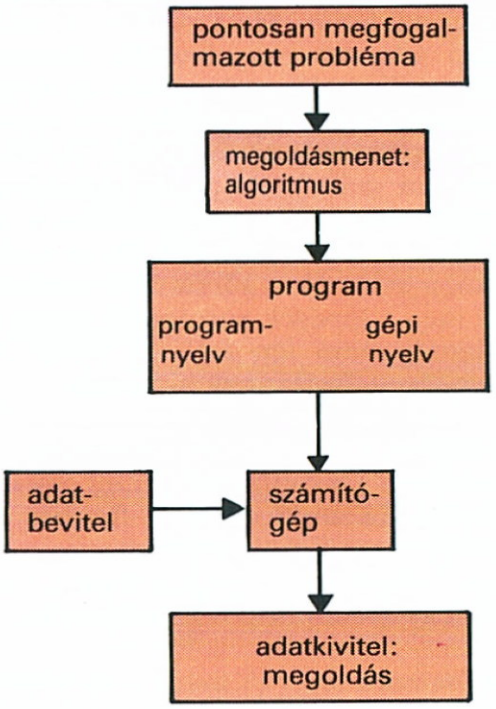
A **léptetőregiszterek** (angolul: *shiftregister*) olyan gyors tárolóelemek, amelyek tartalma az órajel hatására egy helyiértékkel balra vagy jobbra tolódik. A regiszter kimenetét az eltolt bináris jegyek a további feldolgozásra rendelkezésre állnak.

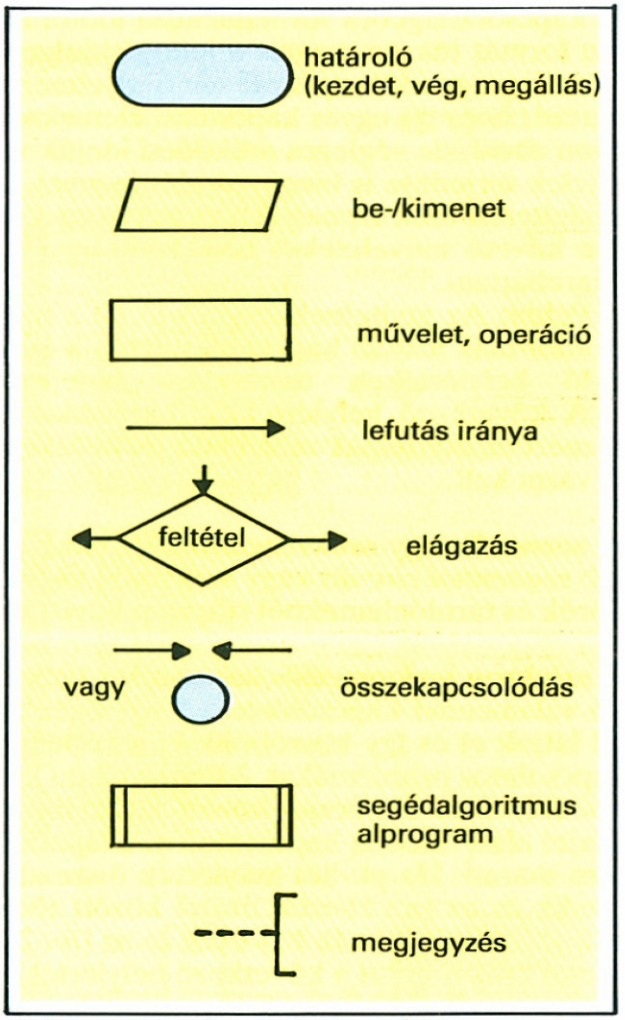
A **gyűrűs számláló** olyan léptetőregiszter, amelyben a tárolt tartalom eltolódáskor pl. a legszélső bal regiszter-flipflop tartalma a következő órajel hatására a jobb oldali legszélső flipflopba íródik. A regiszter tartalma körben forog.

Összeadás. Ha két, p szóhosszúságú, párhuzamosan kapcsolt regiszter tartalma az órajel ütemében egy összeadóműbe tolódik, akkor egy (harmadik) kimenőregiszterben ütemről ütemre az összeg jelenik meg.

Egy a bináris szám **szorzása/osztása** valamely 2 hatványaként kifejezhető számmal különösen egyszerű: a és 2^n szorzata n-szeres balra léptetésnek, a és 2^n hányadosa pedig n-szeres jobbra léptetésnek felel meg.

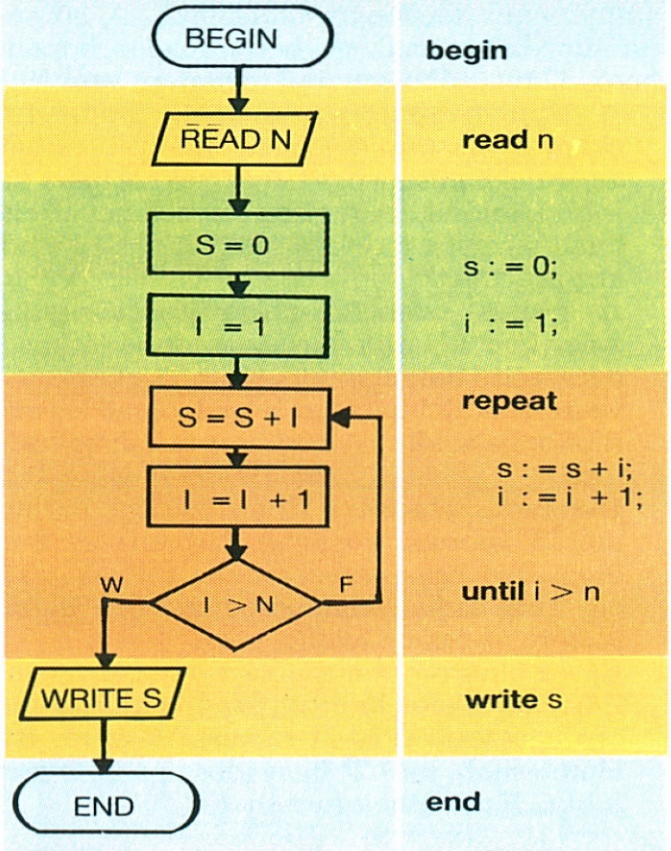
Problémamegoldás:





Folyamatábrák szimbólumai

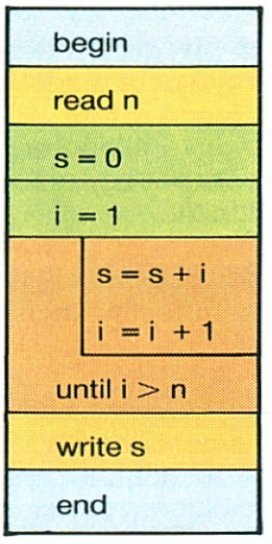
folyamatábra, blokkdiagram:



lineáris leírás:

```
begin
read n
s : = 0;
i : = 1;
repeat
    s : = s + i;
    i : = i + 1;
until i > n
write s
end
```

struktogram:



számítási táblázat: (az n=5 esetre)

```
kezdet
n = 5
s = 0
i = 1
s = 1
i = 2
i > 5 ?   F
s = 3
i = 3
i > 5 ?   F
s = 6
i = 4
i > 5 ?   F
s = 10
i = 5
i > 5 ?   F
s = 15
i = 6
i > 5 ?   W
s = 15
vég
```

Példa: a természetes számok összege n-ig

Algoritmusok ill. programok ábrázolási lehetőségei

Probléma

Mielőtt egy probléma megoldásához hozzákezdünk, a problémát egyértelműen meg kell fogalmazni. A megfogalmazás viszonylag egyszerű, ha matematikai problémával állunk szemben.

Általában minden megoldható problémának tetszőleges számú megoldási algoritmusa létezik, ezek közül az optimálisat kell megkeresni.

Algoritmus

Az algoritmus egy teljesen, egyértelműen és pontosan megfogalmazott feldolgozási előírás, amelyet egy mechanikus vagy elektronikus készülék el tud végezni. Minden olyan feladat, amelynek a megoldási eljárását valamilyen algoritmussal le lehet írni, elvileg számítógép segítségével is megoldható.

Majdnem minden algoritmust soros, tehát egymást követő lépésekben történő feldolgozásra készítenek. A párhuzamos (paralel) számítógépek számára szükséges párhuzamos feldolgozású algoritmusok és az ehhez megfelelő programok kifejlesztése még gyermekcipőben topog.

A következőkben összefoglaljuk azokat a tulajdonságokat, amelyekkel minden algoritmusnak rendelkeznie kell.

Végesség. A megoldáshoz az algoritmust elemi műveletek véges számának kell leírnia, az algoritmus szövege tehát csak véges hosszúságú lehet. A hosszúság nem előre meghatározott hosszúságot jelent, az algoritmus készítése közben részalgoritmusokkal bővülhet, de ezeknek is véges hosszúságúnak kell maradniuk.

A **közelítések (approximációk)** segítségével a végtelen algoritmusok végessé alakíthatók. A közelítő és az egzakt megoldás közötti eltérés csak a számítástechnikai ráfordítástól függ.

Determináltság, egyértelműség, beleértve a **lokális egyértelműséget** is. Az algoritmusnak minden bemenő adatra egyértelmű eredményt kell szolgáltatnia. Ha adott bemenő adatok feldolgozását az algoritmussal megismételjük, a számításnak mindig ugyanahhoz az eredményhez kell vezetnie. Ez azt jelenti, hogy a bemenő adatok tulajdonságait (pl. természetes számok, valós számok) és ábrázolásukat is (fixpontos, lebegőpontos) pontosan meg kell határozni.

Teljesség. Ide tartozik az, hogy az algoritmus elkerüli az olyan zsákutcákat, amelyekből külső beavatkozás nélkül nem tudna kikerülni. Pl. egy feltételes utasításban az „igaz" a feltételes utasítás elé vezethetne vissza és a „hamis" feltétel sosem fordulhatna elő.

Az **univerzalitást** szintén elvárjuk, ami azt jelenti, hogy az algoritmus egy meghatározott értéktartományon belül nagyszámú különböző bemenő adatra használható legyen.

Az **érthetőség** (áttekinthetőség) különösen akkor fontos, ha az algoritmus különböző részeit különböző személyek készítették.

A **tesztelhetőség, ellenőrizhetőség** nem triviális követelmény az algoritmussal szemben, mivel a terjedelmes algoritmusok helyességét csak olyan adatok segítségével lehet ellenőrizni, amelyekre az eredmény ismert.

Az algoritmusok leírása

Minden algoritmus bizonyos számú elemi műveletre és a műveletek sorrendjét meghatározó előírásra vezethető vissza. Az algoritmusoknak alapvetően két leírási módja létezik: a folyamatábra (blokkdiagram) ill. a lineáris ábrázolás (pszeudokód).

Folyamatábra. Az algoritmus lefutása folyamatábra segítségével grafikusan ábrázolható. Az ábrázolás szabványosított grafikus elemek segítségével történik, amelyeket nyilak kötnek össze egymással. A folyamatábrák a rövid algoritmusokat áttekinthetően és jól érthetően ábrázolják, de már közepes hosszúság esetén is alkalmatlannak bizonyulnak.

A **struktogram** lényegesen tömörebb grafikus ábrázolást tesz lehetővé. Ez a módszer hosszabb algoritmusok leírásakor is még viszonylag jól áttekinthető.

A **lineáris ábrázolás** (pszeudokód). Ez a leírási mód a megoldást sorról sorra elemi műveletek segítségével írja le. Felépítése hasonlít egy problémaorientált programnyelven írt számítógépprogramhoz.

Általános érthetősége miatt ennél a leírási módnál a PASCAL programnyelvet részesítik előnyben.

Az **optimális algoritmusok** a probléma megoldását vagy közelítő megoldását a rendelkezésre álló erőforrások lehető legkisebb mértékű igénybevételével és a lehető legrövidebb időn belül hajtják végre. Eközben kompromisszumokra van szükség pl. a program kódolásának időtartama. Az adatbeviteli követelmények és a számolási (futási) idő tekintetében, de az algoritmus érthetősége és áttekinthetősége is szerepet játszik. A gyakorlatban az alábbi szempontok különösen lényegesek:

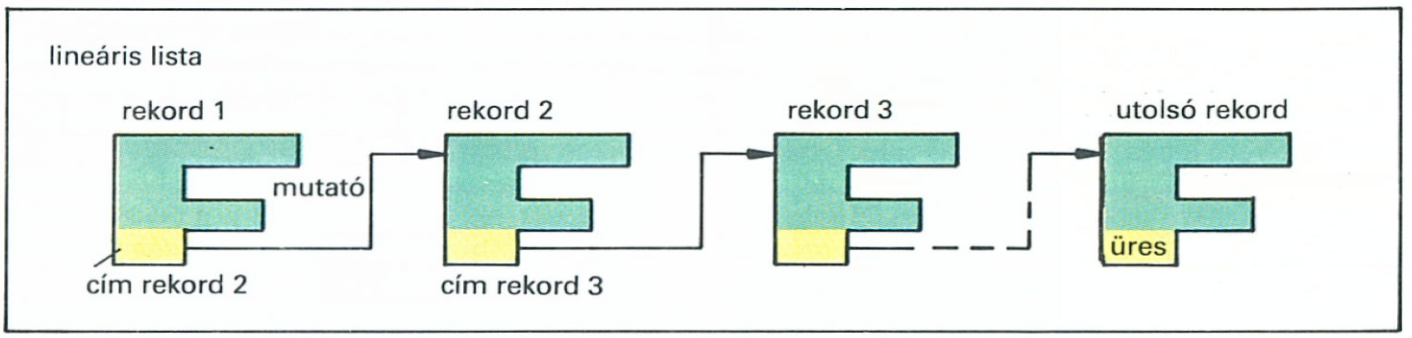
A **probléma egyszerűsítése** azt jelenti, hogy a problémát egy egyszerűen ábrázolható modellel közelítjük. Ez csak akkor alkalmazható, ha a modellel ábrázolt, valamint a kérdéses probléma megoldása közötti eltérés kvantitatívan megbecsülhető.

A **problémaredukció** során a feladatot visszavezetjük egy egyszerűbb, esetleg egy már korábban megoldott problémára. Ebben a programkönyvtárak fontos szerepet játszanak.

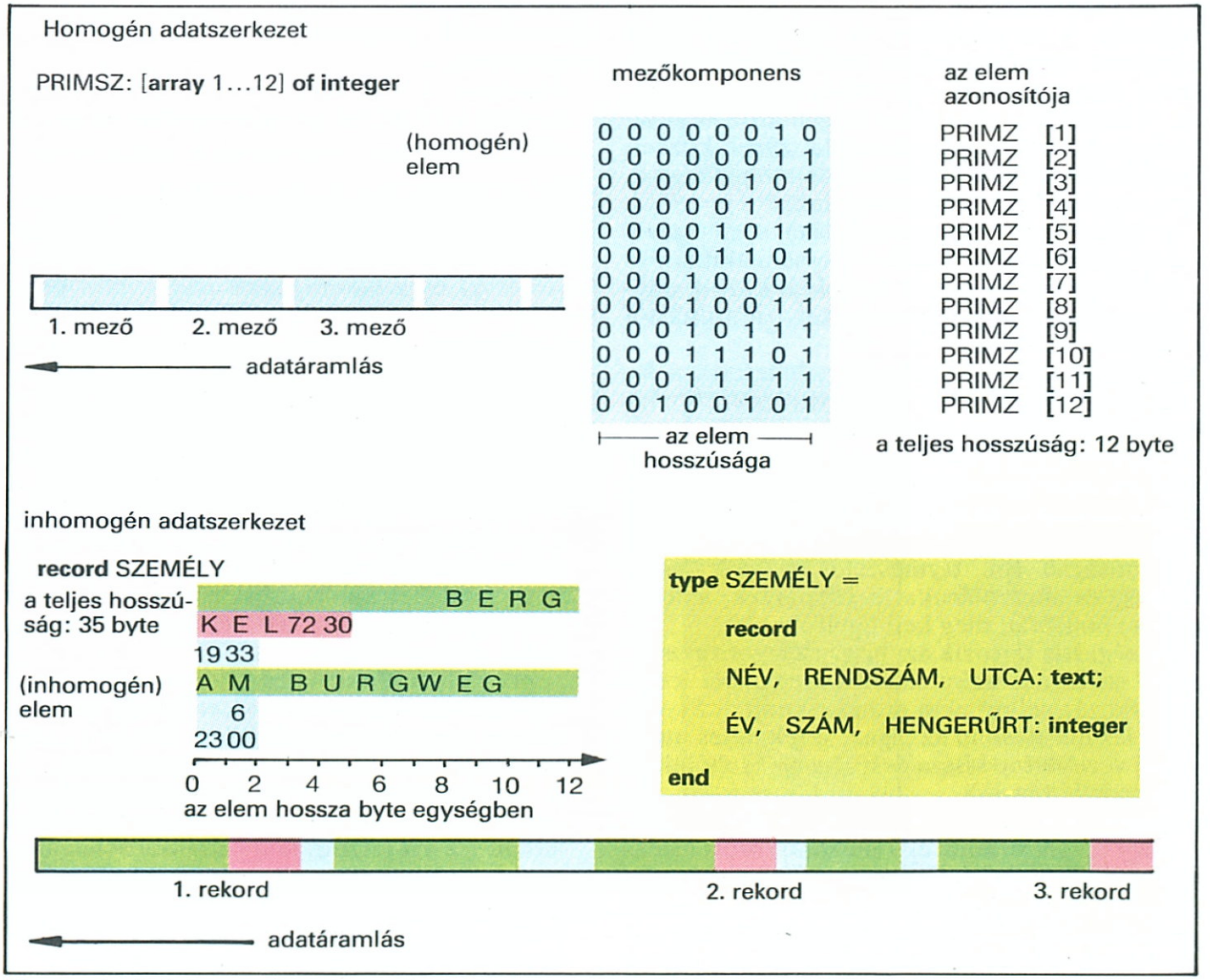
A **részproblémákra való felbontás** a szerteágazó problémák megoldása során elkerülhetetlen. Ez azt jelenti, hogy a részproblémákat külön oldjuk meg, majd a megoldásokat egymással ismét kombináljuk. A modern programnyelvek a moduláris programfelépítés lehetőségével elősegítik ezt a módszert.

bitsorozat	lehetséges értelmezés	eredmény
0 1 0 1 0 0 1 1	természetes szám bináris kódja	83
0 1 0 1 0 0 1 1	egy szám 1-es komponensű ábrázolása	+44
0 1 0 1 0 0 1 1	egy szám 2-es komponensű ábrázolása	+45
0 1 0 1 0 0 1 1	egy betű ASCII-kódja	S
0 1 0 1 0 0 1 1	egy jel EBCDI-kódja	(nincs jelentése)
0 1 0 1 0 0 1 1	tömörített számábrázolás, 1 byte	53 (az előjel határozatlan)
0 1 0 1 0 0 1 1	jelsorozat	(a sorozat 83, egyenként 1 byte hosszú jelet tartalmaz)

Azonos bitsorozat különböző értelmezési lehetőségei



Dinamikus adatstruktúra



Homogén és inhomogén adatstruktúra

Az adatok és az algoritmusok együttesen alkotják a programot. Az adattípusoknak és az adatstruktúráknak a választott algoritmushoz illeszkednie kell.

Adatok és adatstruktúrák

Az **adatok** az információ, különösen a számítógépben feldolgozásra kerülő információ ábrázolására használt jelsorozatok. A feldolgozásra vonatkozó utasítások szintén adatoknak minősülnek. A számítógépben az adatokat rendszerint bináris ábécé reprezentálja.

Az **elemi adatstruktúrák** azok a legkisebb adatobjektumok, amelyeknek még önálló jelentés tulajdonítható. A legtöbb programnyelvben az egyszerű konstansok és változók természetes, egész és valós számok, valamint jelek, jelsorozatok és logikai értékek lehetnek. Mivel a számítógép csak bináris jelsorozatokat fogad el, a várható adattípust explicit vagy implicit módon deklarálni kell.

Példa: Az 1010010 bitsorozat a 82 természetes számot ábrázolhatja. Ha azonban a számítógép ezt a bitsorozatot egész *(integer)* számként értelmezi, akkor jelentése kettes komplemens ábrázolásban –44. A bitsorozat jelentése a 7-bites-ASCII-kód szerint az *R* betű. Néhány számítógép esetében azonban az 1010010 bitsorozat egy utasítás.

Az adattípus ismerete a memóriában történő valósidejű elegendő helyfoglalás miatt is fontos. A legtöbb számítógépben pl. egy betű ábrázolásához 1 byte, egy lebegőpontos szám ábrázolásához pedig 8 byte-ig terjedő tárolóhely szükséges.

Az összetett **adatstruktúrák** elemi adatstruktúrákból tevődnek össze. Ehhez tartoznak például:

Az **adattömb (tömb,** angolul: *array*) olyan indexelt elemeket tartalmaz, amelyek azonos adattípushoz tartoznak.

A **rekord** olyan zárt adategység, amely több, nem azonos típusú elemet foglal magába. Minden elemnek saját külön azonosítója van.

Példa: Kartotékrendszer, amelynek elemei: név, születési dátum, a születés helye, lakóhely, postai irányítószám, bankszámlaszám stb.

A **halmaz** (angolul: *set*) legtöbbször bináris jelek olyan sorozata, amely egy (jelsorozathoz tartozó) elem jelenlétét vagy hiányát jelöli.

Példa: a prímszámok sorozatát a természetes számokon belül halmazként jelölhetjük.

A **dinamikus adatstruktúrák** láncolt adattípusok. Minden elem egy **mutatót** tartalmaz, amely a következő elemre utal. Ha az utolsó elem az elsőre mutat, akkor gyűrűs adatszerkezet keletkezik. Az elemek egynél több mutatót is tartalmazhatnak, ez pl. a döntési fák esetén fordul elő.

A **mutatók** (angolul: *pointer*) a tárolócímekre utalnak. Ezek pl. egy utasítás címét tartalmazhatják. A mutatókkal elsősorban a program hatékonyságát lehet fokozni.

Algoritmusok

A **numerikus algoritmusok** pl. szorzást és osztást hajtanak végre, vagy egy függvény integrálját képezik adott integrációs határok között.

A **nem-numerikus algoritmusok** pl. adatokat rendeznek, vagy valamilyen szövegből egy adott szót keresnek ki.

Helyesség. A felhasznált algoritmus helyességéről analitikus módszerrel kell meggyőződni. Nem elegendő, ha az algoritmust egyedi értékekkel teszteljük. A program utasításainak helyes sorrendje a tesztelés lényeges része. Ha a sorrend korrekt, akkor rendszerint az algoritmus is a helyes eredményre vezet.

Hatékonyság. Egy algoritmus ill. program teljesítőképessége azon mérhető le, milyen gyorsan jut el az adott probléma megoldásához, és mekkora az ehhez felhasznált számítástechnikai erőforrás (pl. a tárolóigény). Ez magától az algoritmustól és a feldolgozásra kerülő adatok típusától függ. A hatékonyságról – ugyanúgy, ahogy a helyességéről – nem lehet egyedi példák vizsgálatával megbizonyosodni. A tesztelés gyakran statisztikus módszerekkel végezhető el.

Példa: Egy n elemből álló számsorozat legnagyobb elemének *(max)* kiválasztásához szükséges t gépidő nyilvánvalóan n-től és a *max:* = ... értékadó utasítások m számától függ.

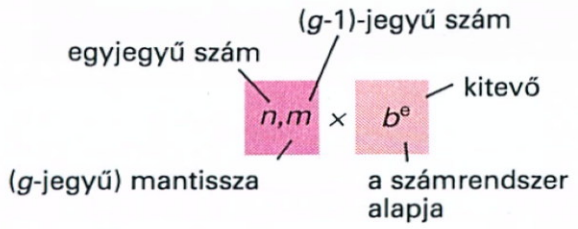
A *legkedvezőtlenebb esetben* minden szám nagyobb az előtte álló számnál. Ekkor $m = n$, és t maximális.

A *legkedvezőbb esetben* az első szám a legnagyobb, ekkor $m = 1$, t pedig minimális.

A *legvalószínűbb esetben* a számsorozat rendezetlen. Annak a valószínűsége, hogy a második szám az elsőnél nagyobb, 1/2. Egy értékadó utasítás valószínűsége 1/2. Annak a valószínűsége, hogy a harmadik elem nagyobb, mint az első kettő, 1/3. Ezzel az értékadó utasítás valószínűsége $1/2 + 1/3 + \ldots + 1/n$. Ennek a sorozatnak az összege $\lg n + 0{,}577\ldots$, tehát $t \sim \lg n$.

A **komplexitásvizsgálat** (bonyolultságvizsgálat) során mind az időigényt, mind a tárolóigényt megvizsgáljuk. Egy gyorsabb algoritmus esetleg aránytalanul nagy tárolóhelyet vesz igénybe.

Az **adatstruktúrát és a komplexitást** egymással összhangba kell hozni. Egy adott adatstruktúra az algoritmustól függően rövid végrehajtási idővel és csekély tárolóigénnyel párosulhat, ezért az adatállományok gyakran dinamikus adatstruktúrával rendelkeznek.



Példa:

$8{,}7654 \times 10^5$

alap: 10
kitevő: 5
mantissza: 8,7654
a jegyek száma: 5

Egy g-jegyű szám normál alakja

bináris	0 1
oktális	0 1 2 3 4 5 6 7
decimális	0 1 2 3 4 5 6 7 8 9
hexadecimális	0 1 2 3 4 5 6 7 8 9 A B C D E F

Különböző számrendszerek számjegykészlete

számrendszer	alap	szám	jegyek száma
bináris	2	$11\ 000\ 000\ 111\ 001_2$	14
oktális	8	$361\ 100_8$	6
decimális	10	$123\ 456_{10}$	6
hexadecimális	16	$1E\ 240_{16}$	5

Minél nagyobb a számrendszer alapja, annál tömörebb a számírás

$$\sum_{i=0}^{g-1} z_i\, b^i = z_0\, b^0 + z_1\, b^1 + z_2\, b^2 + \ldots + z_{g-1}\, b^{g-1}$$

Példák: (számozás jobbról balra)

$$987654_{10} = \sum_{i=0}^{5} z_i\, 10^i = 4 \times 10^0 + 5 \times 10^1 + \ldots + 9 \times 10^5$$

$$10111010_2 = \sum_{i=0}^{7} z_i\, 2^i = 0 \times 2^0 + 1 \times 2^1 + \ldots + 1 \times 2^7$$

Egy g-jegyű egész szám értéke

2^n	n
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1 024	10

8^n	n
1	0
8	1
64	2
512	3
4 096	4
32 768	5
262 144	6
2 097 152	7
16 777 216	8
134 217 728	9
1 073 741 808	10

10^n	n
1	0
10	1
100	2
1 000	3
10 000	4
100 000	5
1 000 000	6
10 000 000	7
100 000 000	8
1 000 000 000	9
10 000 000 000	10

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 717 476 736	9
1 099 479 627 776	10

2, 8, 10, és 16 hatványai

A számokat a különböző számrendszerekben számjegyekkel ábrázoljuk. Valamennyi napjainkban használatos számrendszer **helyiértékes rendszerű,** azaz egy számjegy értéke a számon belül elfoglalt helyétől függ.

A rendelkezésre álló számjegyek száma az alkalmazott számrendszer **alapjával (bázisával)** egyenlő. A tízes (decimális) számrendszerben pl. a számok ábrázolására legfeljebb tíz különböző számjegyet használunk, az oktális számrendszerben nyolcat, a bináris számrendszerben pedig csak kettőt. Minél nagyobb a számrendszer bázisa, annál kevesebb számjegy szükséges egy szám ábrázolásához.

Példa: A Naprendszerben a bolygók számának (9) megadásához a tízes számrendszerben egyetlen számjegy elegendő, bináris számrendszerben viszont ugyanehhez 4 számjegy (1001) szükséges.

A helyiértékes számrendszerekben egy szám **normálalakja:**

$z = m \times b^n$,

ahol

z: a szám,

m: a mantissza,

b: az alap,

n: a kitevő.

A nem egész számok ábrázolásakor a mantissza számjegyeinek sorát egy elválasztójel (vessző vagy pont) két részre osztja.

A helyiértékes számrendszer megnevezése, **alapja** alapján történik.

A **tízes számrendszert** (amelynek alapja 10) egy fejlődéstörténeti véletlennek köszönhetjük – annak, hogy két kezünk ujjainak száma tíz.

A **bináris számrendszer** (alapja kettő) eredete a matematikai logikához nyúlik vissza. Nagy gyakorlati jelentőségre tett szert a számítógépek megjelenésével, mert mind a kapcsolóelemeknek mind a tárolóelemeknek csak két lehetséges állapota van, amelyek a 0 és az 1 számjegyekkel optimálisan ábrázolhatók.

A **hexadecimális számrendszert** (alapja 16) az adatfeldolgozásban gyakran használják, mivel a tárolóelemekben tömör számábrázolást tesz lehetővé.

Különböző számrendszerekben ábrázolt számok egyidejű használata esetén a számrendszer bázisát alsó indexben tüntetjük fel.

Példák: Az 101_8 oktális, a $2BA_{16}$ pedig hexadecimális szám. Az alsó index nélkül felírt számok mindig decimális számot jelentenek, tehát $2345 = 2345_{10}$

Tízes alapú (decimális) számrendszer

A decimális számrendszer alapja 10.

$z = m \times 10^n$

A decimális számokat rövid vagy normálalakban szokás felírni.

A **rövid alakban** az egyes számjegyek helyiértékét a tizedesvessző határozza meg.

Példa: $z = 1992{,}35$

Minden helyiérték tíz különböző hatványának felel meg. Balról jobbra haladva: ezresek, százasok, tízesek, egyesek, tizedek, századok. Tehát:

$z = 1 \times 10^3 + 9 \times 10^2 + 9 \times 10^1 + 2 \times 10^0 + 3 \times 10^{-1} + 5 \times 10^{-2}$

A z szám 6 számjegye közül 4 a tizedesvessző előtt, 2 pedig az után helyezkedik el.

A **normálalakban** a tizedesvessző és tíz hatványa határozza meg az egyes számjegyek helyiértékét.

Példa: $z = 1{,}99235 \times 10^3$

A z szám összesen 6 jegye közül 1 áll a tizedesvessző előtt.

A normálalak az aritmetika számítási szabályait követve más tízes hatványokká alakítható.

Példa: $z = 19{,}9235 \times 10^2 = 199{,}235 \times 10^1 = 0{,}199235 \times 10^4$.

Két azonos kitevőjű decimális szám **összeadása/kivonása** a mantisszák összeadása/kivonása útján történik:

$z_1 \pm z_2 = (m_1 \pm m_2) \times 10^n$.

Ha a kitevők különbözőek, akkor a művelet előtt a számokat olyan alakra kell hozni, amelyben a kitevők megegyeznek.

Két decimális szám **szorzása/osztása** a mantisszák szorzása/osztása és a kitevők összeadása/kivonása útján történik:

$z_1 \times z_2 = (m_1 \times m_2) \times 10^{(n1 + n2)}$

és

$z_1/z_2 = (m_1/m_2) \times 10^{(n1 - n2)}$

Bináris (kettes alapú) számrendszer

A bináris számrendszer alapja 2.

$z = m \times 2^n$.

A bináris számrendszerben két különböző számjegy fordul elő (0 és 1). Elválasztójelként pontot használunk.

Példa: $z = 110001.01_2$.

Minden helyiérték 2 különböző hatványának felel meg.

Tehát:

$z = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$.

A z szám 8 bináris jegye közül kettő az elválasztójel után áll.

Két bináris szám **összeadása** a következő táblázat segítségével végezhető el:

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 10$

Példa:

$$\begin{array}{r} 10011011.11 \\ +\ 11001010.01 \\ \hline =101100110.00 \end{array}$$

0	0	
1	0	1

$1_2 \times 1_2 = 1_2$

bináris egyszeregy

0	0						
1	0	1					
2	0	2	4				
3	0	3	6	11			
4	0	4	10	14	20		
5	0	5	12	17	24	31	
6	0	6	14	22	30	36	44
7	0	7	16	25	34	43	52	61
	0	1	2	3	4	5	6	7

$6_8 \times 4_8 = 30_8$

oktális egyszeregy

0	0								
1	0	1							
2	0	2	4						
3	0	3	6	9					
4	0	4	8	12	16				
5	0	5	10	15	20	25			
6	0	6	12	18	24	30	36		
7	0	7	14	21	28	35	42	49	
8	0	8	16	24	32	40	48	56	64
9	0	9	18	27	36	45	54	63	72	81
	0	1	2	3	4	5	6	7	8	9

$7_{10} \times 3_{10} = 21_{10}$

decimális egyszeregy

0	0														
1	0	1													
2	0	2	4												
3	0	3	6	9											
4	0	4	8	C	10										
5	0	5	A	F	14	19									
6	0	6	C	12	18	1E	24								
7	0	7	E	15	1C	23	2A	31							
8	0	8	10	18	20	28	30	38	40						
9	0	9	12	1B	24	2D	36	3F	48	51					
A	0	A	14	1E	28	32	3C	46	50	5A	54				
B	0	B	16	21	2C	37	42	4D	58	63	6E	79			
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90		
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

$E_{16} \times C_{16} = A8_{16}$

hexadecimális egyszeregy

Bináris, oktális, decimális és hexadecimális szorzótábla

Módszer, amelyet a $0{,}3_{10}$ szám példáján mutatunk be

$0{,}3 \times 2 = 0{,}6$
$0{,}6 \times 2 = 1{,}2$
$0{,}2 \times 2 = 0{,}4$
$0{,}4 \times 2 = 0{,}8$
$0{,}8 \times 2 = 1{,}6$
$0{,}6 \times 2 = 1{,}2$
$0{,}2 \times 2 = 0{,}4$
$0{,}4 \times 2 = 0{,}8$
$0{,}8 \times 2 = 1{,}6$

$0{,}3_{10} \simeq 0.010\ 011\ 001$

Átalakítás a negatív bináris kitevőket tartalmazó táblázat segítségével

tizedestört	bináris kitevő	bináris szám
0,5	2^{-1}	0.1
0,25	2^{-2}	0.01
0,125	2^{-3}	0.001
0,062 5	2^{-4}	0.000 1
0,031 25	2^{-5}	0.000 01
0,015 625	2^{-6}	0.000 001
0,007 812 5	2^{-7}	0.000 000 1
⋮	⋮	⋮

$0{,}3 = 0{,}25 + 0{,}031\ 25 + 0{,}015\ 625 + 0{,}001\ 953\ 125$

$= 0.01 + 0.000\ 01 + 0.000\ 001 + 0.000\ 000\ 001$

Decimális törtszám átalakítása bináris törtszámmá

Két **bináris** szám kivonása a tízes számrendszerben alkalmazott összeadáshoz hasonlóan történik. Egyszerűbb megoldást jelent azonban, ha a kivonandó kettes komplemens kódját képezzük (103. o.), és ezt a kisebbítendőhöz hozzáadjuk (a bal oldalon egy esetleg fellépő átvitelt nem veszünk figyelembe).

Példa: 11001010.01
−01011011.11

tehát

11001010.01
+ 10100100.00
=101101110.01

A **szorzás** és az **osztás.** A szorzás és az osztás során használt bináris „egyszeregy" különösen egyszerű:

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

A **hexadecimális** számrendszer

A hexadecimális számrendszer alapja 16.

$z = m \times 16^n$.

A hexadecimális számrendszerben 16 különböző számjegy fordul elő (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E és F).

A hexadecimális számrendszert azért alkalmazzák, mert egy számjegye fél byte (= 4 bit) segítségével ábrázolható: a számítógépben 4 tárolóhellyel 16-féle számjegyet lehet tárolni (előjel *nélkül*).

Példák: $z_1 = 1A3B_{16}$.

Minden helyiérték 16 valamelyik hatványának felel meg, tehát

$z_1 = 1 \times 16^2 + 3 \times 16^1 + 11 \times 16^0$.

Ez a szám a tízes számrendszerben kifejezve:
4096 + 2560 + 48 + 11 = 6715.

$z_2 = 1A.3B_{16} =$
$= 1 \times 16^1 + 10 \times 16^0 + 3 \times 16^{-1} + 11 \times 16^{-2}$.

Ez a szám a tízes számrendszerben kifejezve
16 + 10 + 3/16 + 11/256 = 26,2305.

A hexadecimális számok aritmetikája megfelel a decimális számok aritmetikájának.

Oktális számrendszer

Az oktális számrendszer alapja 8.

$z = m \times 8^n$.

Az oktális számrendszerben 8 különböző számjegy fordul elő (0. 1, 2, 3, 4, 5, 6 és 7).

Az oktális számjegyek a számítógépben (előjelükkel együtt) fél byte (= 4 bit) tárolóhelyet vesznek igénybe.

Példa:

$z = 1234_8 =$
$= 1 \times 8^3 + 2 \times 8^2 + 3 \times 8^1 + 4 \times 8^0$.

Decimális számokkal kifejezve:
512 + 128 + 24 + 4 = 668.

Az oktális számok aritmetikája megfelel a decimális számok aritmetikájának.

A számrendszerek közötti átváltás

A z_{10} tízes számrendszerben felírt egész számnak egy b alapú formára történő **átváltása** (átalakítása) úgy történik, hogy a z_{10} számot b-vel ismételten elosztjuk, mindaddig, amíg a hányados 1-nél kisebbé nem válik. Minden egyes osztás maradéka az új számrendszerben egy-egy számjegyet jelent. Figyelem: az új számjegyeket jobbról balfelé haladva kell olvasni (lásd az alábbi példákat):

Decimális → bináris átváltás

Példa: az 1993 decimális szám átváltása.

1993:2 = 996 + 1/2;	996:2 = 498;	498:2 = 249;	249:2 = 124 + 1/2;
maradék 1	0	0	1
124:2 = 62;	62:2 = 31;	31 :2 = 15 + 1/2	15:2 = 7 + 1/2
maradék 0	0	1	1
7:2 = 3 + 1/2;	3:2 = 1 + 1/2;	1:2 = 1/2	
maradék 1	1	1	

Az eredmény: $1993_{10} \rightarrow 11111001001_2$.

Decimális → hexadecimális átváltás

1993:16 = 124 + 9/16;	124:16 = 7 + 12/16;	7:16 = 0 + 7/16
maradék 9	12	7

Az eredmény: $1993_{10} \rightarrow 7C9_{16}$,

Decimális → oktális átváltás

1993:8 = 249 + 1/8:	249:8 = 31 + 1/8;	31:8 = 3 + 7/8;	3:8 = 3/8
maradék 1	1	7	3

Az eredmény: $1993_{10} \rightarrow 3711_8$.

Egy z szám b bázisú számrendszerben felírt alakját úgy váltjuk át tízes számrendszerbeli formára, hogy z egyes számjegyeit a helyiértéknek megfelelő b hatvánnyal megszorozzuk és az így nyert számokat összeadjuk.

Bináris → decimális átváltás

Példa: $z = 110010_2$

$110010_2 \rightarrow 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^1 = 32 + 16 + 2 = 50_{10}$.

Hexadecimális → decimális átváltás

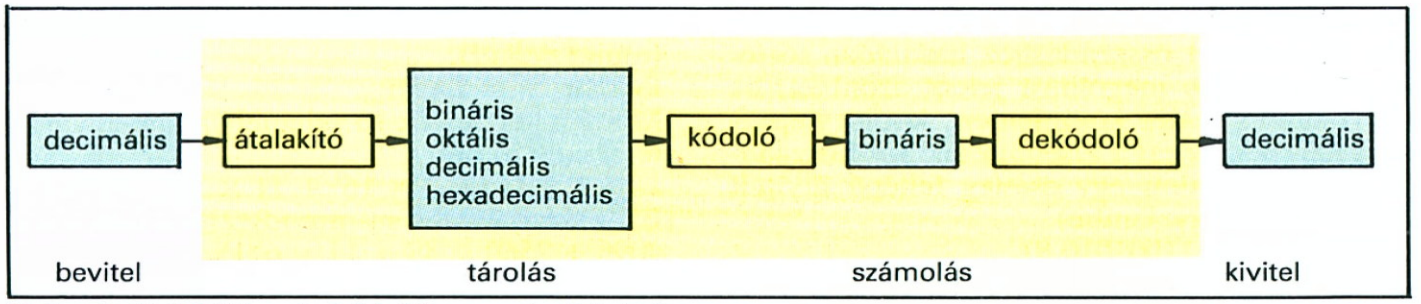
Példa: $z = 3D2_{16}$

$3D2_{16} \rightarrow 3 \times 16^2 + 13 \times 16^1 + 2 \times 16^0 = 768 + 208 + 2 = 978_{10}$.

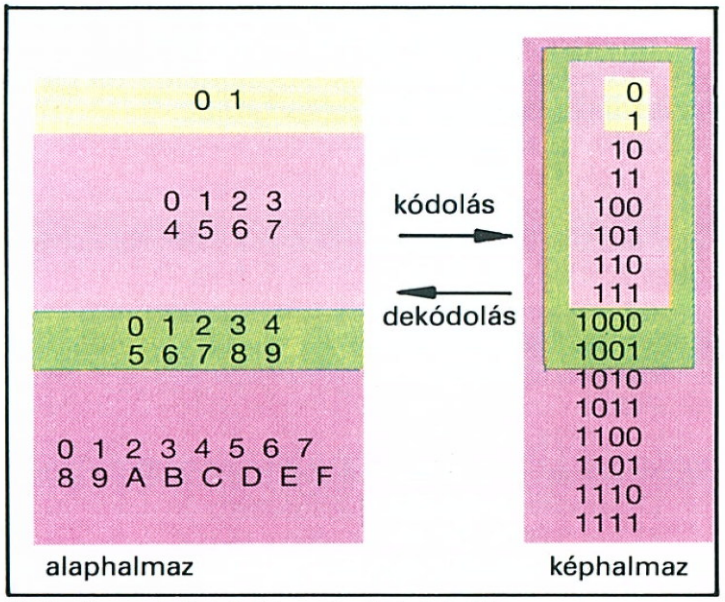
Oktális → decimális átváltás

Példa: $z = 2317_8$

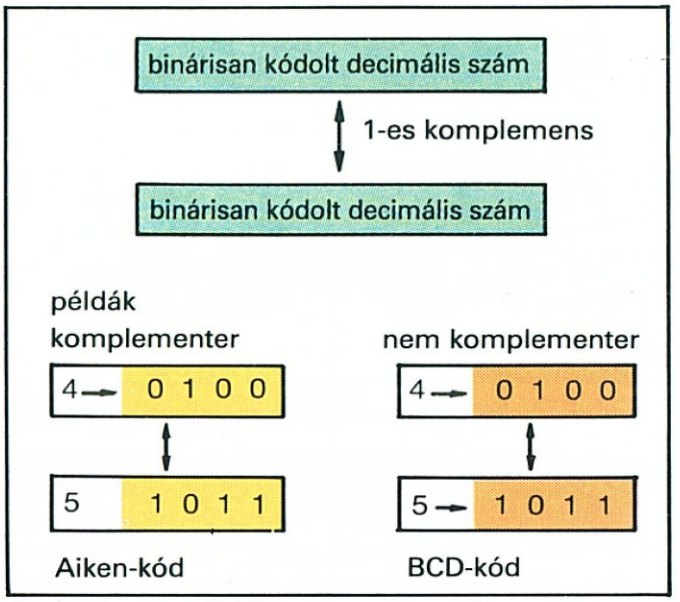
$2317_8 \rightarrow 2 \times 8^3 + 3 \times 8^2 + 1 \times 8^1 + 7 \times 8^0 = 1024 + 192 + 8 + 7 = 1231_{10}$.



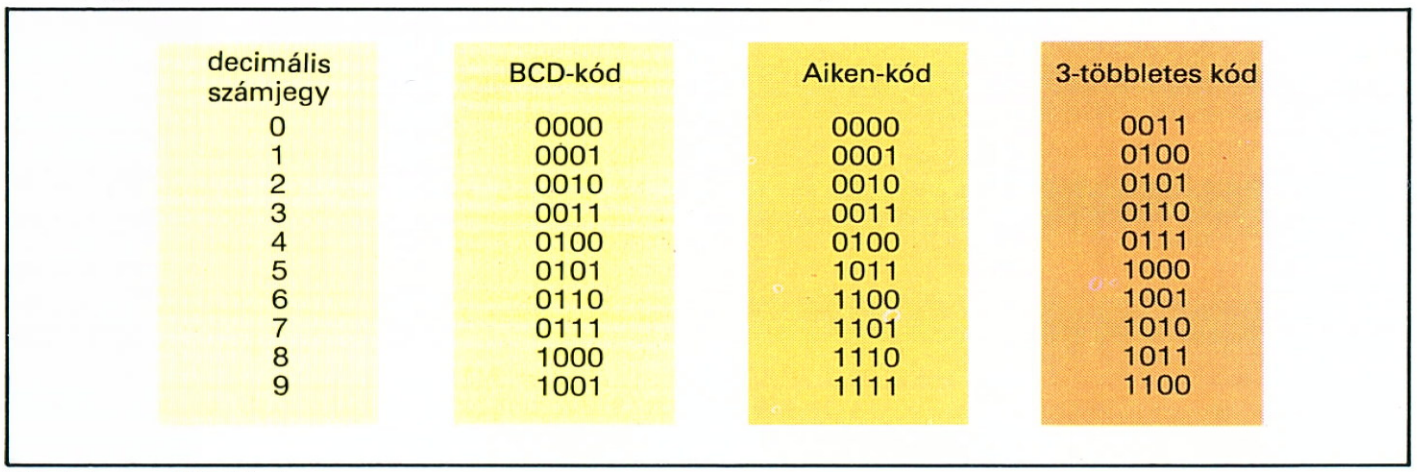
Számok átalakítása és kódolása a számítógépben



Számjegyek kódolása és dekódolása

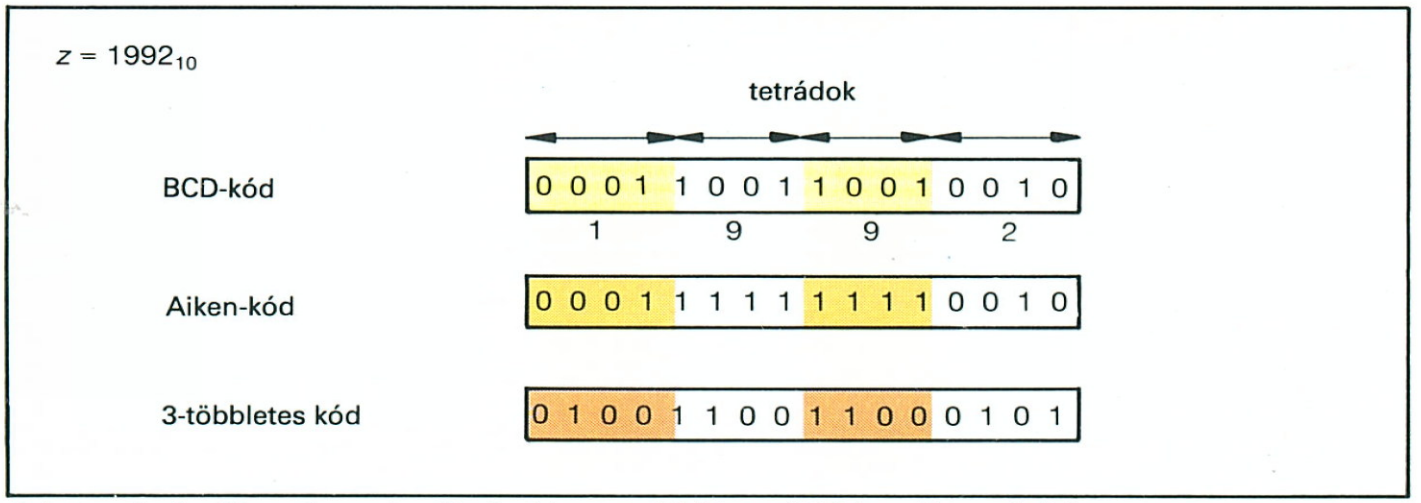


Komplemens kód



decimális számjegy	BCD-kód	Aiken-kód	3-többletes kód
0	0000	0000	0011
1	0001	0001	0100
2	0010	0010	0101
3	0011	0011	0110
4	0100	0100	0111
5	0101	1011	1000
6	0110	1100	1001
7	0111	1101	1010
8	1000	1110	1011
9	1001	1111	1100

Bináris tetrádkódok



Adott z decimális egész szám ábrázolása tetrádokban (16 bit szóhosszúság)

A számrendszer kiválasztása a körülményektől függ. A kézzel történő számításokat egészen biztosan mindig a meghitt tízes számrendszerben végezzük. Számítógép használata esetén egyéb szempontok fontosabbá válhatnak. Pl. a hexadecimális számábrázolás viszonylag kevés tárolóterületet vesz igénybe.

Ha adatfeldolgozó rendszert használunk, akkor az alkalmazott számrendszert a számítógép számára érthetővé kell tenni. A számoknak a számítógépes használatra történő ábrázolása a kódolás speciális esetének tekinthető.

Különbséget kell tenni az **átalakítás** (egy bizonyos számrendszerben ábrázolt szám másik számrendszerbe történő átváltása) és az **ábrázolás** (a számoknak a számítógép számára érthető alakban történő kódolása) között.

A gyakorlatban a számok gépi ábrázolására egyelőre csak a bináris kódolás jöhet szóba. Ennek oka az alkalmazott elektronikus építőelemek bináris természete, amelynek lényege az, hogy két lehetséges stabil állapot közül csak az egyiket vehetik fel.

A jövendő optikai számítógépeinél ez nem lesz szükségszerűen így. Kapcsolóelemeiknek kettőnél több stabil állapota létezhet, tehát esetleg a decimális számábrázolás is lehetségessé válik.

A jelenlegi körülmények között kettes számrendszer alkalmazása tűnik legcélszerűbbnek. A kettes számrendszerben ábrázolt számok bináris kódolása rendkívül egyszerűen megvalósítható. Másrészről viszont a bináris számok tárolóigénye viszonylag nagy.

Leibniz már 1676-ban szerkesztett egy bináris kódolást alkalmazó golyós számológépet. Az első, bináris kódolással működő automatikus számítógépet Zuse építette 1938-ban.

Kódok

A **kód** leképezési előírás, amely az *alaphalmaz* minden jeléhez egyértelműen hozzárendeli a *képhalmaz* egy jelét vagy jelsorozatát. Pl. a bináris kód bináris jelek sorozatát rendeli egyértelműen hozzá minden decimális számot ábrázoló jelhez.

A legtöbb kód kölcsönösen egyértelmű, vagyis az alaphalmaz különböző jelsorozatai a képhalmaz különböző jelsorozatainak felelnek meg és fordítva. Az alaphalmaznak a képhalmazba történő leképezését **kódolásnak** (rejtjelezésnek), az ellentétes folyamatot pedig **dekódolásnak** (megfejtésnek) nevezzük.

A gyakorlatban a kódoknak a kölcsönös egyértelműség követelményen kívül a következő további feltételeket kell kielégíteniük:

- A képelemeknek lehetőleg rövidnek kell lennie, hogy csekély átviteli időt és tárolóhelyet igényeljenek.
- A gépi kódolást lehetővé kell tenniük.
- A kódnak a kódolási és átviteli hibákkal szemben nagymértékben érzéketlennek kell lennie.
- Ehhez járul még az, hogy a legtöbb számítógép számára a jeleket állandó szóhosszúságú képhalmazzá kell leképezni.

Bináris kódok

A bináris kód jelkészlete a 0 és 1 bináris jelekből áll. A decimális – vagy bármely más számrendszerben ábrázolt – számokat különböző módon lehet binárissá alakítani: vagy tetrádokban, bináris számokká történő közvetlen kódolással (egész számok és egész számokból képzett valódi törtek esetén), vagy fixpontos ill. lebegőpontos ábrázolással (valós számok, hatvány alakban ábrázolt számok esetén). A digitális számítógépekben a bemenő jeleket bináris kóddá alakítják, a bináris jeleket pedig ismét dekódolják.

A **tetrádkódok** a számokat számjegyként, helyiértéküknek megfelelően *tetrád* sorozatok segítségével, azaz egyenként négybites csoportokban képezik le. Ha egy ilyen módon kódolt decimális számjegy 1-es komplemense (103. o.) ismét decimális számjegyet reprezentál, akkor **komplementerkóddal** állunk szemben. Ez a kódolás előnyökkel jár a tízes alapú aritmetika műszaki megvalósítása során.

Egy decimális számjegynek egy tetrádba történő kódolása az adott tárolóhely 37%-át elpazarolja, mivel egy tetrád tíz különböző számjegy (0, 1, 2, 3, 4, 5, 6, 7, 8 vagy 9) egyikét ábrázolja, pedig maximálisan $2^4 = 16$ különböző számjegyet kódolhatna. Így tehát hat olyan tetrád, ún. *pszeudotetrád* van, amelyhez nem rendelünk decimális számjegyet.

Történeti és gyakorlati okokból többféle tetrádkód létezik:

A **BCD-kódot** (angolul: **B***inary* **C***oded* **D***ecimal*) 8–4–2–1-kódnak is nevezik. Minden egyes decimális számjegyet közvetlenül és helyiértékének megfelelően egy tetráddá alakítja. A kódolási előírást egy táblázatból lehet kikeresni.

Példa: A 10 369 decimális szám öt tetrádban kódolható: 0001 0000 0011 0110 1001.

Az **Aiken-kód,** amelyet 2–4–2–1-kódnak is neveznek, tetrád- és komplementerkód.

Példa: A 10 369 decimális szám öt tetrádban kódolható: 0001 0000 0011 1100 1111.

A **3-többletes kód,** amelyet **Stibitz-kódnak** is neveznek, tetrád- és komplementerkód.

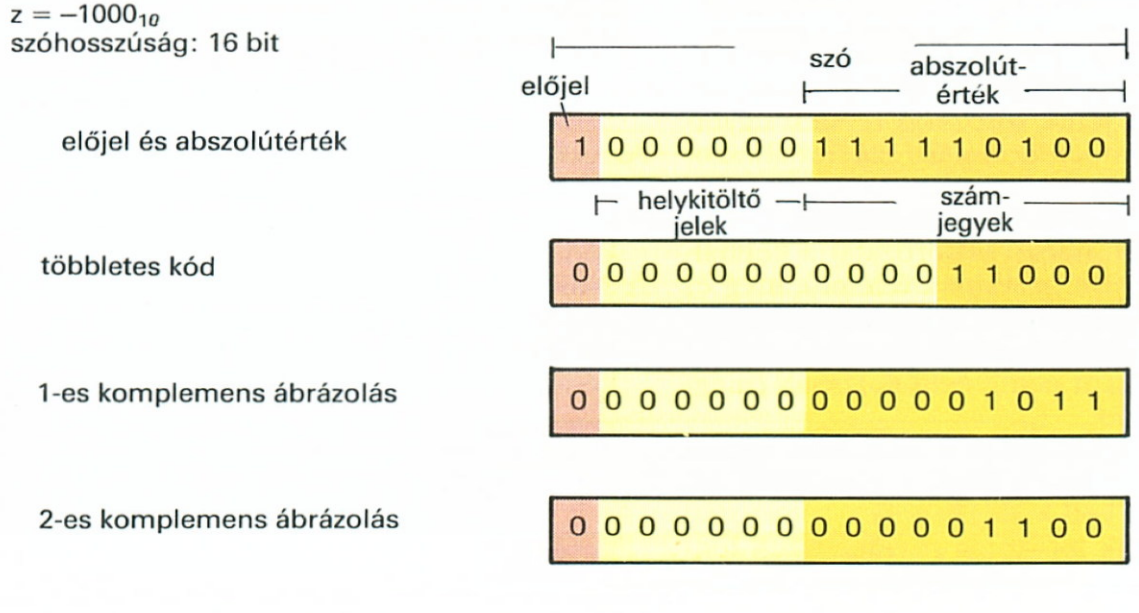
Példa: A 10 369 decimális szám öt tetrádban kódolható: 0100 0011 0110 1001 1100.

ábrázolási mód	ábrázolási tartomány / bit szóhosszúság esetén	előjelbit negatív	előjelbit pozitív
előjel és abszolútérték	$+2^{l-1}-1$-től $-2^{l-1}+1$	1	0
többletes kód	$+2^{l-1}-1$-től -2^{l-1}	0	1
1-es komplemens	$+2^{l-1}-1$-től $-2^{l-1}+1$	1	0
2-es komplemens	$+2^{l-1}-1$-től -2^{l-1}	1	0

z decimális szám	előjel és abszolútérték	többletes kód	1-es komplemens ábrázolás	2-es komplemens ábrázolás
+ 128	túl nagy	túl nagy	túl nagy	túl nagy
+ 127	0111 1111	1111 1111	0111 1111	0111 1111
+ 126	0111 1110	1111 1110	0111 1110	0111 1110
.	.	.	.	
.	.	.	.	
.	.	.	.	
+ 3	0000 0011	1000 0011	0000 0011	0000 0011
+ 2	0000 0010	1000 0010	0000 0010	0000 0010
+ 1	0000 0001	1000 0001	0000 0001	0000 0001
+ 0	0000 0000	1000 0000	0000 0000	0000 0000
− 0	1000 0000	0111 1111	1111 1110	1111 1111
− 1	1000 0001	0111 1110	1111 1101	1111 1110
− 2	1000 0010	0111 1101	1111 1100	1111 1101
− 3	1000 0011	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	0000 0010	1000 0001	1000 0010
− 126	1111 1110	0000 0001	1000 0000	1000 0001
− 127	1111 1111	0000 0000	túl kicsi	1000 0000
− 128	túl kicsi			

A *z* szám kódolási lehetőségei
l = 8 bit szóhosszúság esetén

Előjeles egész számok ábrázolási lehetőségei



Példa adott *z* egész szám ábrázolására

Néhány algoritmus közvetlenül tetrádokkal működik. Bonyodalmat jelent, ha a tetrádok között maradék jelenik meg, a számítógépben a műveletek eredményeit tároló táblázatokkal ez a probléma kezelhető.

Tetrádábrázolásban az összeadáshoz pl. két számítási táblázat szükséges. Az egyik az összeadás eredményéhez, a másik pedig az átvitelhez.

A tetrádokat általában csak a számok tárolásához használják, a tulajdonképpeni számítás – újabb kódolás után – bináris alakban történik.

Az **EBCDI-kódban** (73. o.) 8 bit = 1 byte áll rendelkezésre a decimális számok, a betűk és a különleges jelek kódolásához.

Az **ASCII-kódban** (71. o.) 8 bit = 1 byte áll rendelkezésre, de a kód csak 7 bitet használ fel a decimális számok, a betűk és különleges jelek kódolásához. A 8. bit legtöbbször ellenőrző bitként szolgál.

Egész számok ábrázolása

Az egész számok ábrázolása a kódolás különleges esetét jelenti.

A számítógép a betáplált egész számokat először a gép által alkalmazott számrendszerbe alakítja át. Ezt követi a bináris jellé történő kódolás. A legtöbb számítógép műszaki okokból meghatározott számú bitet (állandó *szóhosszúságot*) használ a számok ábrázolására. Ha a szám az adott szóhosszúságot nem tölti ki, akkor a bináris jelsorozatot balról nullákkal egészítik ki az adott szóhosszúságra.

Példa: $l = 16$ bit szóhosszúság esetén a decimális 1993 → 11111001001 számot 0000 0111 1100 1001 alakban kódolják (a helyköz csak a jelsorozat olvasásának megkönnyítésére szolgál, a számítógép tárolójában a jelek helyköz nélkül követik egymást).

Előjel

A számok előjelét általában a kódolt szám első bitje reprezentálja. Erre a kódolásra – sajnos – különböző lehetőségeket ismerünk.

Előjel és abszolútérték-módszer. A kódszóban (balról) az első hely jelöli az előjelet. A 0 pozitív, az 1 pedig negatív előjelet jelent. A kódszó többi része az abszolútértéket kódolja jobbra igazodva (tehát ha az abszolútérték esetleg nem tölti ki a rendelkezésre álló helyet, akkor a bal oldalon üresen maradó részre nullák kerülnek). Ilyen módon l bit szóhosszúság esetén $l-1$ bit jut az abszolútérték kódolására.

Példák: A +10 decimális számot 8 bites szó alakjában a 0000 1010, a –10-et pedig az 1000 1010 jelsorozat kódolja.

Ezzel a módszerrel a z egész számok a

$$-2^{l-1} + 1 \leq z \leq +2^{l} - 1$$

tartományban kódolhatók.

Hátrányai: a 0 decimális szám mint +0 vagy –0 kódolható. Az operandusok előjelét az összeadás vagy kivonás előtt össze kell hasonlítani. Általában először meg kell állapítani, melyik a legnagyobb abszolútértékű operandus.

Többletes számábrázolás. Minden számhoz hozzáadunk 2^{l-1}-t (ún. *többletet*). Így az eredetileg negatív számokból pozitív számok keletkeznek a 0-tól a $2^{l-1}-1$-ig terjedő tartományban; az eredetileg is pozitív számok a 2^{l-1}-től $2^{l}-1$-ig terjedő tartományba kerülnek. A számok csak ezután az összeadás után állnak operandusként rendelkezésre. Pl. $l = 8$ bit szóhosszúság esetén a kódolás során $2^7 = 128$-at kell a számhoz hozzáadni, a dekódolás során pedig ugyanennyit (tehát 128-at) levonni.

A z szám előjelét az előjelbit ábrázolja. A negatív előjelet 0, a pozitív előjelet pedig 1 jeleníti meg.

Példák: A $z_1 = +10$ decimális számot egy 8 bites szóban a $z_1 + 2^7 = 0000\ 1010 + 1000\ 0000 = 1000\ 1010$ jelsorozat kódolja.

A $z_2 = -10$ decimális számot a $z_2 + 2^7 = -1000\ 1010 + 1000\ 0000 = 0111\ 0110$ jelsorozat kódolja.

Ezzel a módszerrel a z egész számok a

$$-2^{l-1} \leq z \leq +2^{l-1} - 1$$

tartományban kódolhatók.

Az összeadás/kivonás művelete egyszerűen, ezzel szemben a szorzás/osztás viszonylag nehezen végezhető el.

A **2-es komplemensű ábrázolás** a pozitív számokat előjel és abszolútérték együtteseként kódolja (lásd az előzőkben). A negatív számok ($-z$) kódolása a z 2-es komplemenseként történik (103. o.).

A z szám előjelét az előjelbit mutatja. A 0 pozitív, az 1 pedig negatív előjelnek felel meg.

Példák: A $z_1 = +10$ decimális számot egy 8 bites szóban a 0000 1010 jelsorozat kódolja.

A $z_2 = -10$ decimális szám kódolt alakja 1111 0110.

Ezzel a módszerrel a z egész számok a

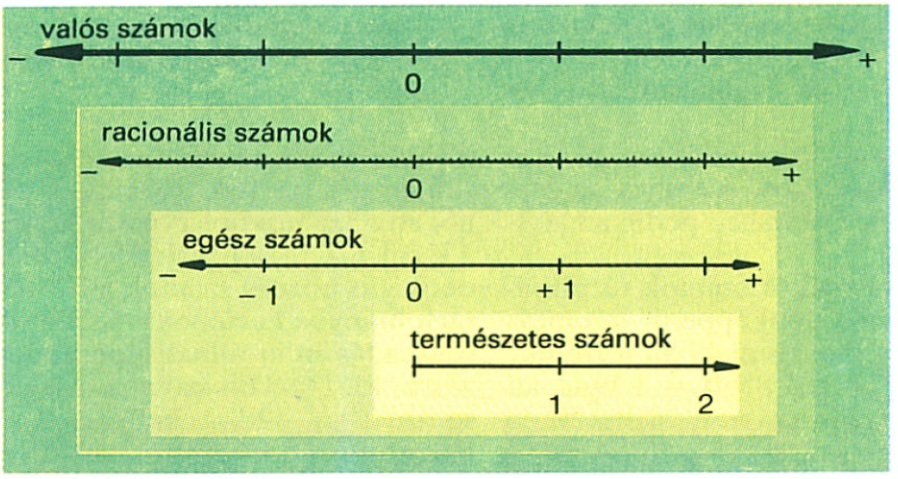
$$-2^{l-1} \leq z \leq +2^{l-1} - 1$$

tartományban kódolhatók.

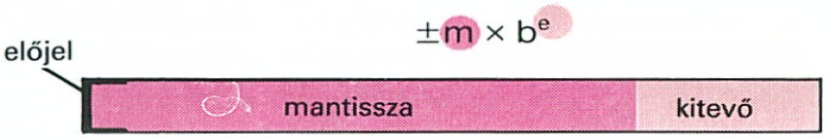
Előnyei: A matematikai alapműveletek ennek a kódolásnak a felhasználásával különösen egyszerűen kivitelezhetők.

Az összeadás független az operandusok előjelétől. A módszer az előjelet kódoló helyen esetleg megjelenő átvitelt nem veszi figyelembe.

A kivonás ugyancsak az összeadási algoritmus segítségével történik. Ha a két szám előjele megegyezik, akkor az előjelekre vonatkozó átvitel ugyancsak figyelmen kívül marad.



Különböző számhalmazok



Lebegőpontos számábrázolás

lebegőpontos szám: $-1{,}6022 \times 10^{-19}$

normalizált lebegőpontos számábrázolás $-0{,}16022 \times 10^{-18}$ kitevő

normalizált mantissza

kódolás

a mantissza mint abszolútérték és előjel

kitevő

a mantissza mint kettes-komplemens

A $-1{,}6022\ldots\times10^{-19}$ valós szám ábrázolása egy 32 bites szóban

szám:	+ 9,75	– 9,75
átalakítás bináris számmá	1001.11	– 1001.11
lebegőpontos számmá	1001.11×2^0	-1001.11×2^0
normalizált lebegőpontos számmá	0.100111×2^4	-0.100111×2^4
az exponens 6 bites többletes kódja	100100	100100
normalizált lebegőpontos számábrázolás (12 bites szóhosszúság)	0100111100100 előjel – abszolútérték kód	011001100100 2-es komplemens kód
24 bites szóhosszúságú kód	000000000000010011110100	000000000000011001100100

Adott szám átalakítása normalizált lebegőpontos gépi kóddá

Az adatfeldolgozás során legtöbbször valós számok, végtelen tizedes törtek fordulnak elő. Mivel a számítógépek csak véges hosszúságú számjegysorozatokat tudnak feldolgozni, a valós számokat egy közelítő értékkel helyettesítve racionális számmá kell alakítani.

Lebegőpontos számábrázolás

Minden valós illetve racionális szám leképezhető a lebegőpontos számok halmazába. Ez a leképezés a *kerekítés* (69. o.).

A **lebegőpontos szám** (angolul: *floating point number*) felírható a következő alakban:

$$z = m \times b^n,$$

ahol
z: lebegőpontos szám,
m: mantissza,
b: az alkalmazott számrendszer alapja,
n: a kitevő.

Mind a mantissza, mind a kitevő negatív szám is lehet. Az alkalmazott számrendszerek a kettes, a tízes és a tizenhatos. *n* határozza meg a szám *ábrázolási tartományát* (lásd az alábbiakban), *m* pedig a *pontosságát* (67. o.).

A számoknak ez a leképezése általában közelítő értéket jelent, mivel a mantissza a programban előre rögzített, meghatározott számú számjegyet tartalmaz.

Példa: A 12,3400 decimális számot az 1010.01010111... bináris szám kódolja. A mantissza 8 jegy hosszúságúra történő rögzítése esetén csak az 1010.0101 jelsorozatot vesszük figyelembe, amely a 12,3125 decimális számnak felel meg.

A digitális számítógépekre a következő megállapodás érvényes:

A **normalizált lebegőpontos szám** alakja

$$z = m \times b^n,$$

ahol $1/b \leq m < 1$ és $m \neq 0$.

A normalizált lebegőpontos szám tehát egészen balról nullával kezdődik, amelyet elválasztójel (vessző vagy pont) követ. Az elválasztójel után áll az első nullától különböző számjegy.

Példák: a 12,34 decimális szám normalizált alakja $0{,}1234 \times 10^2$, az 1010.0101 bináris számé pedig 0.10100101×2^4.

Minden számítógép meghatározott számú tárolóhelyet biztosít a mantissza és a kitevő számára. Mivel minden számítás azonos bázisú számrendszerben ábrázolt számokkal történik, a bázist nem tárolják. A normalizált lebegőpontos számok *m* mantisszáját kettes komplemens vagy előjel és abszolútérték alakjában kódolják (63. o.).

A normalizált lebegőpontos számok kitevőjét legtöbbször többletes kód segítségével (63. o.) kódolják, ebben az esetben *n*-et a lebegőpontos szám **karakterisztikájának** nevezik.

Ha egy szám tárolására 32 bit tárolóhelyet szánnak, akkor a (bináris) mantissza tárolására pl. 26 bit, a kitevő tárolására pedig 6 bit biztosítható.

Példa: Ilyen feltételekkel az 1010.0101 szám tárolása
10100101000000000000000000000100
alakban történik.

Ábrázolási tartomány

A kitevő jegyeinek száma határozza meg az ábrázolt szám ábrázolási tartományát. A következő összefüggés érvényes:

$$2\exp(-2^{r-1} - 1) < z < 2\exp(2^{r-1} - 1),$$

ahol
r: a kitevő bitben kifejezett hosszúsága.

Példa: Ha a kitevő hatjegyű, azaz $r = 6$, akkor $2^{-31}{,}082\ z < 2^{31}$.

Ennek megfelelően az ábrázolási tartomány alsó határa kb. 10^{-10}, a felső határ pedig kb. 10^{10}.

Ha a mantisszát egész számnak tekintjük, akkor a mantisszával ábrázolható számok tartománya

$$0 \leq m \leq 2^p$$

ahol
p: a mantissza bitben kifejezett hosszúsága.

Példa: Ha a (bináris) mantissza 25 jegyű, azaz $p = 25$, akkor a legnagyobb ábrázolható mantissza 2^{25}. Normalizált alakban tízes hatványként felírva ez egy hétjegyű decimális mantisszának felel meg.

A $|z|$ ábrázolási tartomány túllépése esetén a számítógép **túlcsordulást** (angolul: *overflow*), ill. **alulcsordulást** (angolul: *underflow*) jelez. Ilyenkor a számítás menete általában megszakad.

Lebegőpontos aritmetika

Legyen a két operandus $z_1 = m_1 \times b^{n_1}$ és $z_2 = m_2 \times b^{n_2}$.

Összeadás. A z_1 és a z_2 számot először azonos kitevőjű alakra hozzuk, mégpedig úgy, hogy a kisebb kitevőjű számot alakítjuk nagyobb kitevőjűvé. Ha $n_1 > n_2$, akkor

$$z_1 + z_2 = (m_1 + m_2/(b^{n_1 - n_2})) \times b^{n_1}.$$

Kivonás. A kivonandó mantisszájából kettes komplemens alakot képezünk és ezután hozzáadjuk a kisebbítendőhöz.

Szorzás. A mantisszákat összeszorozzuk, a kitevőket pedig összeadjuk.

$$z_1 \times z_2 = m_1 \times m_2 \times b^{n_1 + n_2}.$$

Osztás. A mantisszákat elosztjuk, a kitevőket pedig kivonjuk egymásból.

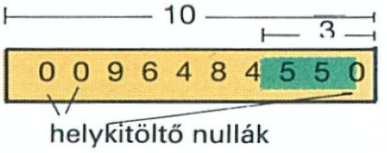
$$z_1/z_2 = (m_1/m_2) \times b^{n_1 - n_2}.$$

Az eredményt a tárolás előtt általában normalizálják.

A részeredményeknek ez a normalizálása időigényes és tulajdonképpen nincs szükség rá. Néhány számítógép a felhasználóra bízza a választást. A végeredmény átlagos hibája mindkét esetben azonos.

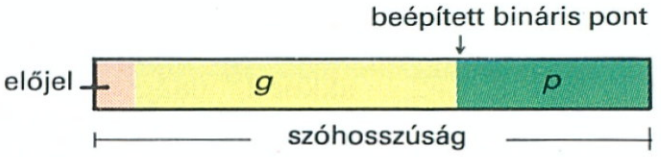
decimális szám: $96484{,}55_{10}$

fixpontos számábrázolás (deklaráció (10.3)):



Decimális számok fixpontos ábrázolása

bináris fixpontos ábrázolás: ± g . p



decimális fixpontos szám:	+ 9,75	− 9,75
szóhosszúság: 16 bit		
p: 6 bit		
bináris fixpontos szám:	1001.11	− 1001.11

kódolás

a bináris pont előtt álló jegyek — a bináris pont után álló jegyek

A 9,75 racionális szám ábrázolása egy 16 bites szóban

példa:

$$z = 1/7 = 0{,}1428571\ldots \times 10^{0} \quad (p,\ b)$$

kerekítési hiba:

$$\frac{|z - \bar{z}|}{|z|} \leq 0{,}5 \times b^{1-p}$$ — gépi pontosság

$p = 4$ jegyre történő csonkítás:	$= 0{,}1428$
4 jegyre történő kerekítés:	$\bar{z} = 0{,}1429$
kerekítési hiba:	$= 0{,}0003 \leq 0{,}0005$

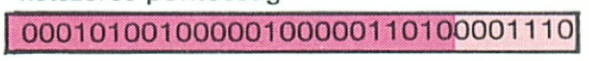
Kerekítés és csonkítás

mantisszahosszúság		
5	00010	$= -0{,}12500_{10}$
6	000101	$= -0{,}15625_{10}$
7	0001010	$= -0{,}15625_{10}$
8	00010100	$= -0{,}15625_{10}$
9	000101001	$= -0{,}16016_{10}$
10	0001010010	$= -0{,}16016_{10}$
.	.	
.	.	
.	.	
16	0001010010000010	$= -0{,}16019_{10}$

A számábrázolás pontossága a mantissza hosszúságával növekszik (az ábra a − 0,16022 szám példáját mutatja)

egyszeres pontosság

kétszeres pontosság



A $-1{,}6022 \times 10^{-19}$ lebegőpontos ábrázolása egyszeres (16 bit) és kétszeres (32 bit) pontossággal

Lebegőpontos számábrázolás

Fixpontos számábrázolás

A **fixpontos szám** (angolul: *fixed point number)* helyiértékes (bináris, oktális, decimális vagy hexadecimális) számrendszerben ábrázolt számjegysorozat. Nem egész számok, tehát pl. a 12,34 és a 0,001234 decimális számok, vagy az 1001.01 és a 0.11101 bináris számok ábrázolására szolgál. A vessző ill. a pont helyzetét a programdeklaráció rögzíti, magából a számjegysorozatból ezt nem lehet felismerni.

Példa: A PL/I programnyelv DECLARE X REAL FIXED DECIMAL (8,2) deklarációja az x decimális számra összesen 8 bit szóhosszúságot rögzít és a tizedespontot jobbról a második helyre helyezi.

A számítógép belső tára a fixpontos számokat elválasztójel nélkül, egyszerű számjegysorozatként ábrázolja. A tizedespont helyzete a deklarációból derül ki.

Példák: a szóhosszúság előjel nélkül: 8):
1234,56: belső ábrázolása → 00123456
1001.011: belső ábrázolása → 01001011

A számítógép a fixpontos számokat egyszerű számjegysorozatként kezeli, az eredményben a helyiértéket csak a számítás végén állapítja meg.

A fixpontos ábrázolás előnye a lebegőpontos műveletekhez képest nagyobb számítási sebesség.

Hátrányai: A számábrázolási tartomány viszonylag kicsi. Ha a művelet különböző módon deklarált fixpontos számokkal történik, akkor ez az eredményt meghamisíthatja.

Fixpontos aritmetika

Az összeadás/kivonás ugyanúgy történik, mint egész számokkal. A szorzás/osztás esetén egy állandó skálatényezőt (2^4, 8^4, 10^4 vagy 16^4) kell figyelembe venni.

A fixpontos számok átalakítása

Egy adott számrendszerben ábrázolt fixpontos számokat egy másik számrendszerbeli fixpontos számokká a következő módszerrel lehet átalakítani.

Az átalakítási módszer a lebegőpontos számok mantisszáira is alkalmazható.

1. Az elválasztójeltől balra álló számjegysorozatot az új számrendszer bázisával (maradékosan) addig osztjuk, amíg az eredmény 0 lesz. Az osztás során kapott maradékok alkotják – fordított sorrendben – az új számsorozat elválasztójeltől balra álló egész részét.
2. A fixpontos szám törtrészét az új bázissal megszorozzuk. A kapott szám egész részét leválasztjuk, ez lesz az új számrendszerben ábrázolt szám elválasztójeltől jobbra álló első számjegye. Az egész rész leválasztása után visszamaradt törtszámmal ugyanezt a műveletet ismételjük. Ilyen módon számjegyenként megkapható az új szám elválasztópont ill. -vessző után álló számjegysorozata.

Példa: A $z = 123{,}45$ decimális szám átalakítása a megfelelő bináris számmá.

Bal oldali rész: 123:2 = 61 (maradék **1**); 61:2 = 30 (maradék **1**); 30:2 = 15 (maradék **0**); 15:2 = 7 (maradék **1**); 7:2 = 3 (maradék **1**); 3:2 = 1 (maradék **1**); 1:2 = 0 (maradék **1**), a sorrend felcserélésével az eredmény → **1111011.**

Jobb oldali számjegysorozat: 0,45 × 2 = 0,9 (egész rész: **0**); 0,9 × 2 = 1,8 (egész rész: **1**); 0,8 × 2 = 1,6 (egész rész: **1**); 0,6 × 2 = 1,2 (egész rész: **1**); 0,2 × 2 = 0,4 (egész rész: **0**); 0,4 × 2 = 0,8 (egész rész: **0**); 0,8 × 2 = 1,6 (egész rész: **1**); 0,6 × 2 = 1,2 (egész rész: **1**); .... → **01110011...**

Eredmény: A 123,45 fixpontos decimális szám fixpontos bináris megfelelője 1111011.01110011.

A valós számok ábrázolásának pontossága

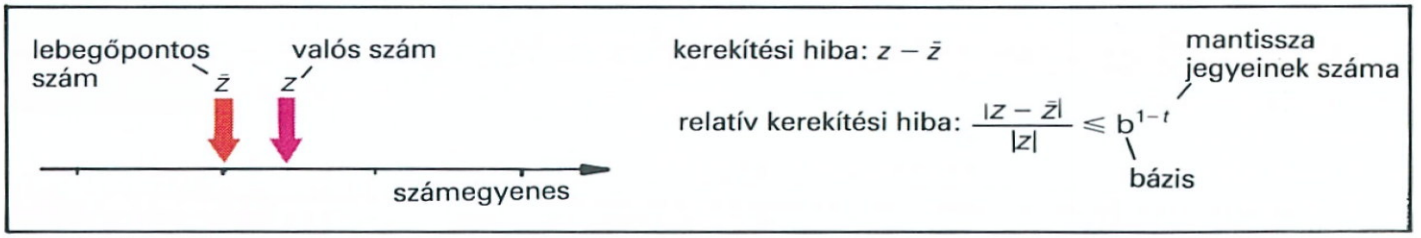
A valós számok ábrázolása során a pontosságot lebegőpontos számok esetén a p mantisszahosszúság, fixpontos számok esetén pedig a számjegyek p száma határozza meg.

p bit kb. $p/(\text{lb } 10)$ decimális helyiértéket jelent, tehát az ábrázolás kb. 0,3 p tizedeshely pontosságú. A nagy pontosság elérése érdekében a számítógépek nagy mantisszahosszúsággal dolgoznak.

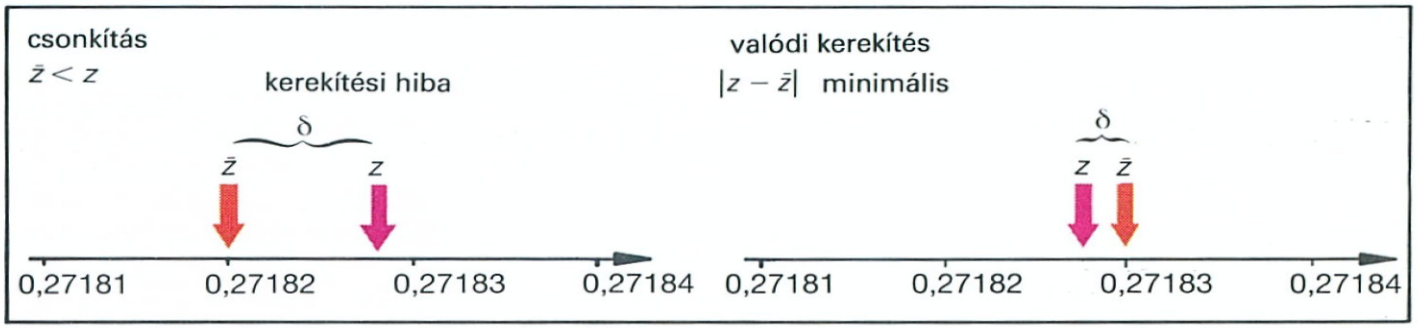
Példa: Ha egy bináris szám mantisszája 8 jegyből áll, akkor a megfelelő decimális szám mantisszahosszúsága csak $3p/10$ jegy, amely 2 és 3 között van. Ez legtöbbször nem elegendő. A 24 bit bináris mantisszahosszúság 7 decimális helynek felel meg, amely a legtöbb esetben kielégítő pontosságot jelent.

Az értékes decimális jegyek száma **kétszeres pontosság** alkalmazása esetén jelentős mértékben növekszik. Ilyenkor két egymás után sorolt szó ábrázol egy számot.

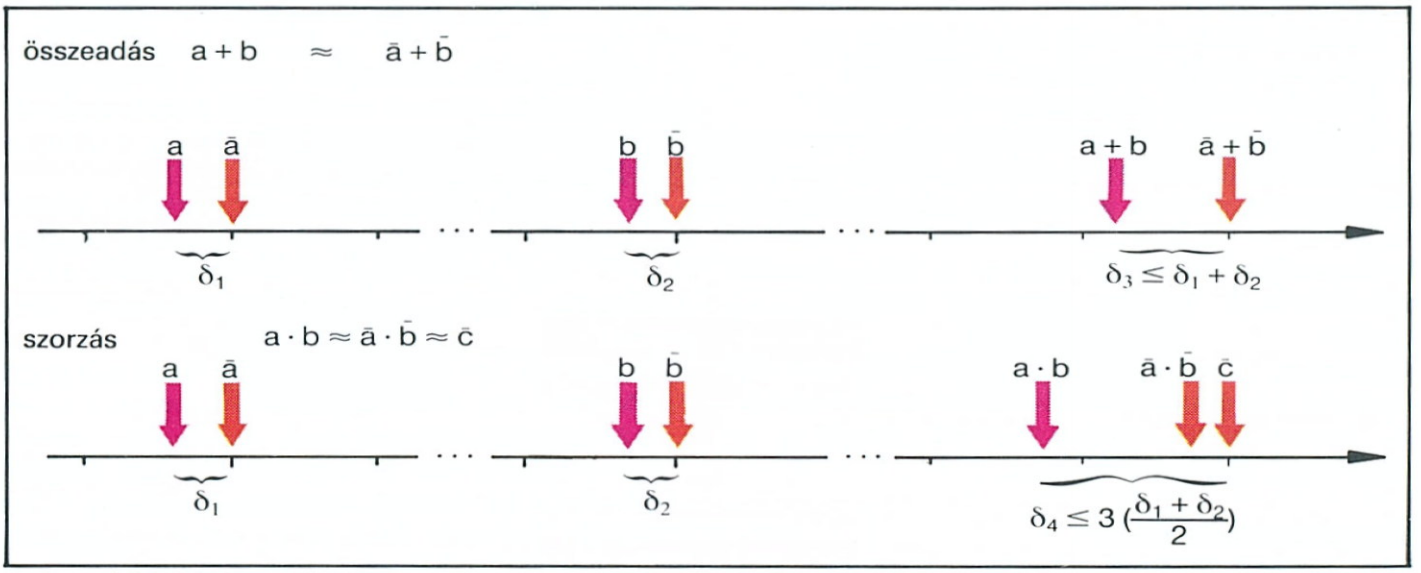
Példa: 32 bit szóhosszúság mellett (amelyből 1 bit az előjel, 7 bit a kitevő, 24 bit pedig a mantissza) a kétszeres pontosság (64–1–7) bit = 56 bit mantisszahosszúságot eredményez. Az értékes decimális jegyek száma tehát kb. 7-ről mintegy 17-re nő.



Kerekítés



Az Euler-féle szám mantisszájának kerekítése



A kerekítési hiba terjedése

valós számok $a, b \xrightarrow[\text{kerekítési hiba}]{\text{kerekítés}} \bar{a}, \bar{b}$ lebegőpontos számok

$\bar{a} = a (1 + \delta_1) \quad \bar{b} = b (1 + \delta_2)$

összeadás

$\bar{a} + \bar{b} = (a + b) (1 + \delta_3)$

$= (a (1 + \delta_1) + b (1 + \delta_2)) (1 + \delta_3)$

$|\delta_i| \; (i = 1, 2, 3, 4) \leq \varepsilon$ esetén

$\leq |a + b| (1 + \varepsilon)^2$

$\leq |a + b| (1 + 2\varepsilon + \varepsilon^2)$

$\varepsilon \ll 1$ esetén: $\bar{a} + \bar{b} \approx |a + b| (1 + 2\varepsilon)$

szorzás

$\bar{a} \cdot \bar{b} = (a \cdot b) (1 + \delta_4)$

$= (a \cdot b) (1 + \delta_1) (1 + \delta_2) (1 + \delta_4)$

$\leq |a \cdot b| (1 + \varepsilon)^3$

$\leq |a \cdot b| (1 + 3\varepsilon + 3\varepsilon^2 + \varepsilon^3)$

$\bar{a} \cdot \bar{b} \approx |a \cdot b| (1 + 3\varepsilon)$

Egyszerű aritmetikai műveletek kerekítési hibája

A valós számok lebegőpontos számokká történő leképezését **kerekítésnek** nevezzük. Ennek során elkerülhetetlenül kerekítési hibák lépnek fel.
A $\bar{z}$ *normalizált* lebegőpontos szám ábrázolása

$$\bar{z} = m \times b^n$$

alakban történik.
Az m mantissza véges sorozat segítségével állítható elő:

$$m = \sum_{k=1}^{t} a_k \, b^{-t},$$

ahol
t: a sorozat elemeinek száma,
a: az alkalmazott számrendszer jegyei,
b: a számrendszer alapja,
n: kitevő, amelynek ábrázolási tartománya $-I_1, \ldots, n, \ldots, I_2$.

Kerekítési eljárások
Minden adott lebegőpontos számrendszerben többféle kerekítési lehetőség létezik, ami azt jelenti, hogy egy z valós számot többféleképpen lehet a $\bar{z}$ lebegőpontos számba leképezni.

Csonkítás. A legegyszerűbb esetben $\bar{z}$ a z számmal szomszédos lebegőpontos szám, amelyre a $\bar{z} < z$ feltétel teljesül.
Ez a kerekítési eljárás tehát úgy végzi a z-ből $\bar{z}$-be történő leképezést, hogy z jegyeit egy bizonyos helytől kezdve levágja.

Valódi kerekítés. $\bar{z}$ megválasztása úgy történik, hogy a $|\bar{z} - z|$ különbség abszolútértéke minimális legyen. A **kerekítési hiba** ebben az esetben z eltérése a legközelebbi lebegőpontos számtól, tehát $= \bar{z} - z$.
Érvényes a következő összefüggés:

$$\bar{z} = z(1 \pm \delta),$$

ahol $\delta \leq b^{1-t}/2$: a **relatív kerekítési hiba.**
δ-t *gépi pontosságnak* is nevezik.
Példa: $b = 10$ és $t = 3$ esetén a relatív kerekítési hiba $<0{,}005$ és ezzel $\bar{z} = z(1 \pm 0{,}005)$.
A kétféle kerekítési eljárás **összehasonlítása** az $e = 2{,}71828182\ldots$ *Euler-féle szám* példáján bemutatva $b = 10$ és $t = 5$ esetén:
csonkításkor: $\bar{e} = 0{,}27182 \times 10^{-1}$,
valódi kerekítéskor: $\bar{e} = 0{,}27183 \times 10^{-1}$ (itt a kerekítési hiba kisebb).

Kerekítési hiba a lebegőpontos számokkal történő számításokban
A lebegőpontos számok használata az aritmetikai műveletek során általában nem lebegőpontos számot ad eredményül. Tehát a műveletet ismét kerekítés követi, ami azt jelenti, hogy minden műveletben ismételt kerekítési hiba lép fel. Ez teljesen hamis eredményekhez vezethet, különösen akkor, ha egymástól csak kismértékben eltérő számok különbségét kell képezni.

Sok programnyelv lehetővé teszi a hosszabb mantisszával való számolást. Erre a FORTRAN nyelvben pl. a DOUBLE PRECISION, a PASCAL nyelvben pedig a LONGINT utasítás szolgál. Ez a módszer azonban nem eredményez szükségképpen arányosan kisebb hibát.

A kerekítési hibák halmozódása
A kerekítési hibák összeadódnak *(halmozódnak)*. Ha az egyes hibák előjele megegyezik, akkor a kerekítési hiba maximális.
A következőkben a négy alapművelet halmozott maximális relatív hibáját **(teljes relatív hibáját)** tárgyaljuk.
Előfeltételek: a kerekített operandusok relatív hibája kicsi ($\delta_i < 5\%$), előjelük azonos.
Az operandusok:

$$\bar{a} = a(1 + \delta_1) \text{ és } \bar{b} = b(1 + \delta_2),$$

ahol
a, b: valós számok,
a, b: kerekített számok.

Összeadás. $\bar{a} + \bar{b} \approx |a + b|(1 + 2\varepsilon)$,

ahol $|\delta_i| < \varepsilon$, ha $i = 1, 2$.
A teljes relatív hiba legfeljebb az egyes relatív kerekítési hibák *kétszerese* lehet.

Kivonás. A teljes hiba az operandusok nagyságától függ.
A legkedvezőtlenebb esetben, amikor $\bar{a} \approx \bar{b}$, a következő összefüggés érvényes:

$$\bar{a} - \bar{b} \approx |a - b|(1 + 2|a|).$$

A teljes relatív hiba tehát az egyik operandus értékének kétszeresét is elérheti!

Szorzás. $\bar{a} \cdot \bar{b} \approx |a \cdot b|(1 + 3\varepsilon)$.

Osztás. $\bar{a} / \bar{b} \approx |a / b|(1 + 3\varepsilon)$.

A két utóbbi művelet teljes relatív hibája legfeljebb az egyes relatív kerekítési hibák *háromszorosát* érheti el.

A halmozott hiba becslése
A hosszabb programok eredményeire a hibabecslés a hiba halmozódásának programlépésről programlépésre való követésével lehetséges. Ennek során statisztikus módszerekkel azt is figyelembe kell venni, hogy az egyes hibák előjele különbözhet.
A vizsgálatok azt mutatják, hogy a pl. több összeadás során a halmozott hiba az operandusok sorrendjétől függ. A teljes hiba tehát többek között a program **kondicionálásával,** vagyis a műveletek sorrendjének optimalizálásával csökkenthető.
A halmozott hiba pontos meghatározása csak akkor lehetséges, ha a programban minden valós számot *mindkét* szomszédos lebegőpontos számmal ábrázolunk; ez azonban nagyon munka- és időigényes módszer.

7 bites ASCII-kód

szűkített jelkészlet

							0	0	0	0	1	1	1	1	sorszám
							0	0	1	1	0	0	1	1	
							0	1	0	1	0	1	0	1	
			0	0	0	0	NUL	DLE	SP	0	@	P	'	p	0
			0	0	0	1	SOH	DC 1	!	1	A	Q	a	q	1
			0	0	1	0	STX	DC 2	"	2	B	R	b	r	2
			0	0	1	1	ETX	DC 3	#	3	C	S	c	s	3
			0	1	0	0	EOT	DC 4	$	4	D	T	d	t	4
			0	1	0	1	ENQ	NAK	%	5	E	U	e	u	5
			0	1	1	0	ACK	SYN	&	6	F	V	f	v	6
			0	1	1	1	BEL	ETB	'	7	G	W	g	w	7
			1	0	0	0	BS	CAN	(	8	H	X	h	x	8
			1	0	0	1	HT	EM	)	9	I	Y	i	y	9
			1	0	1	0	LF	SUB	*	:	J	Z	j	z	10
			1	0	1	1	VT	ESC	+	;	K	[	k	{	11
			1	1	0	0	FF	FS	,	<	L	\	l	|	12
			1	1	0	1	CR	GS	–	=	M	]	m	}	13
			1	1	1	0	SO	RS	·	>	N	^	n	~	14
			1	1	1	1	SI	US	/	?	O	_	o	DEL	15
oszlopszám							0	1	2	3	4	5	6	7	

példák:

6: 0 1 1 0 1 1 0

CR: 0 0 0 1 1 0 1

néhány vezérlő utasítás jelentése

NUL	nulla (helykitöltő karakter)
EOT	az átvitel vége
ACK	pozitív visszajelzés
BEL	csengő
LF	soremelés
CR	kocsi vissza
NAK	negatív visszajelzés
CAN	érvénytelen
DEL	törlés
SP	szóköz

A 7 bites ASCII-kód kódolási előírása

Az adatfeldolgozás során nemcsak számjegyek, hanem betűk, különleges karakterek és vezérlőkarakterek is előfordulnak. A számítógépben minden jelet bitsorozatok ábrázolnak.

A számjegyeket, betűket és különleges karaktereket együttesen **alfanumerikus jeleknek** nevezzük. Megállapodás szerint a *vezérlőkaraktereket,* mint pl. az **FF** (angolul: **F***orm* **F***eed*), lapemelést, vagy az **SPA** (angolul: **SPA***ce*), helyközt, nem számítjuk az alfanumerikus jelek közé.

Jelek

A **jelek** (**karakterek,** angolul: *characters*) egy jelkészlet elemei.

Jelkészlet. A jelkészlet terjedelme, azaz a különböző jelek száma, a jelek leképezésére használt bitsorozat maximális hosszától függ. Ha a bitsorozathoz egy újabb bitet csatolunk, akkor a jelkészlet megkétszereződik.

Ha pl. jelenként 6 bit áll rendelkezésre, akkor ennek segítségével $2^6 = 64$ különböző jel ábrázolható. 7 bit $2^7 = 128$ jel ábrázolását teszi lehetővé. Ez elegendő a szokásos számjegyek, betűk és a legegyszerűbb matematikai szimbólumok ábrázolására. Technikai okokból a jeleket a 8 bit = 1 byte hosszúságú jelsorozattal szokás ábrázolni.

A *jelkészlet* egyben a *karakterkészletet,* vagyis a képernyőn vagy a nyomtatón megjeleníthető jelek halmazát is jelenti.

A jeleket úgy képezik le bitsorozattá, hogy lehetőleg egyszerű és optimális felhasználást tegyenek lehetővé.

Példa: Az ASCII-kódábécében az 1111111 bitsorozat a **DEL** (angolul: **DEL***ete*), törlés utasítást jelenti. Ezzel az utasítással pl. a lyukkártyás kódolás során a helytelenül lyukasztott jelet egyszerűen átlyukasztással eliminálni lehet.

Egy jelkészlet jeleiből összefűzött és egymás mellé sorolt véges számú jelek együttesét **karaktersorozatnak** (angolul: *string*) vagy **karakterláncnak** nevezzük. A karakterlánc pl. egy azonosítót jelölhet. A karaktersorozatot gyakran egy különleges karakter zárja le. A számítógép belső számábrázolásában a kódolt karaktersorozat hossza rendszerint 1 byte egész számú többszöröse. A számítógép a karaktersorozatokat mint egységet kezeli.

Egy kódábécé egymáshoz fűzött jeleinek véges sorozatát *szónak* is nevezik. (Ezt a fogalmat nem szabad a tárolóval kapcsolatban alkalmazott *tárolószó* fogalmával összetéveszteni, mert az utóbbi több tárolórekesz együttesét jelenti, 107. o.)

Ábécé

A rendezett jelkészleteket **ábécéknek** (kódábécéknek) nevezzük. Ez általában véges jelsorozatot jelent.

Példák: A latin és görög ábécé, PASCAL-ábécé, (a, b, c, ..., <, >, ,], ¬), az ASCII- és az EBCDI-kódok.

Egy ábécére vonatkozó **rendezési relációk** pl. $A < B < \ldots$ és $(<) < {}^* < + < \ldots$. A legtöbb programnyelv lehetőséget nyújt arra, hogy az összehasonlító műveletekben rendezési relációkat alkalmazzunk.

Ha jelek között rendezési relációk nincsenek értelmezve, akkor a jelek csupán *karakterkészletet* alkotnak. A szakirodalomban alkalmazott terminológia azonban nem teljesen egyértelmű.

ASCII-ábécé

Az **ASCII** (angolul: **A***merican* **S***tandard* **C***ode for* **I***nformation* **I***nterchange*) az alfanumerikus jelek és a vezérlőkarakterek számítógépes ábrázolására alkalmazott széles körben elterjedt ábécé. Az ASCII és részhalmazai majdnem minden számítógépen megtalálhatók.

Az ASCII rendszerint 7 bites kód. Az ábécé ekkor 128 jelet tartalmaz, amelyből 32 a vezérlőkarakterek számára van fenntartva. Az 1 byte szóhosszúság alkalmazásakor fennmaradó nyolcadik bit paritásellenőrzésre szolgál.

A *szűkített* ASCII-ábécé csupán 64 jelből áll, hiányoznak belőle többek között a kisbetűk.

A 8 bitre kiterjesztett ASCII-ábécé (extended ASCII) 256 különböző jelet tartalmaz, így a 7 bites ASCII-ábécéhez képest 128 újabb jel, pl. a német nyelv különleges betűi vagy grafikus jelek ábrázolására is lehetőség nyílik.

Az ASCII kódolási előírás. A kódtáblázat 8 oszlopra (0–7) és 16 sorra (0–15) tagolódik. A sorok és az oszlopok sorszámai minden jelet egyértelműen megjelölnek. Az oszlopok fölött elhelyezkedő 3 bit a bal oldali 3 bitet, a sorok mellett balra elhelyezkedő 4 bit pedig az utolsó 4 bitet jelenti.

Példák: A 6 számjegy a 3. oszlop 6. sorában helyezkedik el. A táblázatból kiolvasható az ehhez tartozó 011 és 0110 bitsorozat. A 6 számjegyet tehát a 0110110 bitsorozat ábrázolja.

Paritásbit alkalmazása esetén a belső gépi ábrázolás: 01101101. A 0. oszlopban és a 13. sorban található **CR** vezérlőkarakter kódja a 000 és az 1101 bitsorozatból tevődik össze. Tehát a **CR** vezérlőkarakter belső gépi ábrázolása a 0001101 bitsorozat.

Paritásbit alkalmazása esetén a belső gépi ábrázolás: 00011010.

A **számokat** az ASCII-ábécé számjegyenként ábrázolja. Minden decimális helyiérték 1 byte hosszú bitsorozatot vesz igénybe.

Példák: A 25 decimális szám a 0110010 és a 01101101, a 128 decimális szám pedig a 0110001, a 0110010 és a 0111000 bitsorozatokkal ábrázolható.

Paritásbit alkalmazás esetén a megfelelő bitsorozatok a következők: 01100100 és 01101011, ill. 01100010, 01100100 és 01110000.

EBCDI

1. tetrádok

2. tetrádok	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	sorszám
	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	
	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	
0 0 0 0	NUL				SP	&	–									0	0
0 0 0 1							/		a	j			A	J		1	1
0 0 1 0									b	k	s		B	K	S	2	2
0 0 1 1									c	l	t		C	L	T	3	3
0 1 0 0	PF	RES	BYP	PN					d	m	u		D	M	U	4	4
0 1 0 1	HT	NL	LF	RS					e	n	v		E	N	V	5	5
0 1 1 0	LC	BS	EOB	UC					f	o	w		F	O	W	6	6
0 1 1 1	DEL	IL	PRE	EOT					g	p	x		G	P	X	7	7
1 0 0 0									h	q	y		H	Q	Y	8	8
1 0 0 1									i	r	z		I	R	Z	9	9
1 0 1 0			SM		¢	!	^	:									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
oszlopszám	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

példák:

6: 1 1 1 1 0 1 1 0

NL: 0 0 0 1 0 1 0 1

néhány vezérlő utasítás jelentése

NUL	nulla (helykitöltő karakter)
LC	kisbetűk
DEL	törlés
NL	soremelés és kocsi vissza
LF	soremelés
EOB	blokk vége
RS	rekordelválasztó karakter
UC	nagybetűk
EOT	átvitel vége
SP	szóköz

Az EBCDI-kód kódolási előírása

Az **előjeleket** az ASCII-kód külön ábrázolja. (Ez nem minden kódnál magától értetődő, lásd az alábbiakban az EBCDI-kódot.)

Példa: a +25 jelsorozat belső gépi ábrázolása a 0101011, a 0110010 és a 0110101 bitsorozat; a –25 jelsorozaté pedig a 0101101, a 0110010 és a 0110101.

EBCDI-ábécé

Az **EBCDI** (angolul: ***E**xtended **B**inary **C**oded **D**ecimal **I**nterchange*) az alfanumerikus jelek és a vezérlőkarakterek számítógépes ábrázolására alkalmazott 8 bites kód. Az ábécé tehát maximálisan $2^8 = 256$ különböző jelet tartalmaz. Elsősorban nagyobb számítógépekben használják.

Az EBCDI-kód a BCD-kódnak (61. o.) a számjegyeken kívül egyéb jelek ábrázolására történt továbbfejlesztése.

Minden egyes jel két, egyenként négybites tetrádból tevődik össze. Az első tetrád a **zónarész,** az ezt követő tetrád a **számjegyrész.**

Az **EBCDI-kódolási előírás** az ASCII-kód előírásához hasonlít (71. o.). A kódtáblázat 16 oszlopra (0–15) és 16 sorra (0–15) tagolódik. Az oszlopok fölött elhelyezkedő 4 bit a zónarészt, a sorok mellett balra elhelyezkedő 4 bit pedig a számjegyrészt jelenti.

Példák: A 6 számjegy a 15. oszlop 6. sorában helyezkedik el. Az ehhez tartozó tetrádok az 1111 és a 0110 bitsorozat. A 6 számjegyet tehát az 11110110 bitsorozat ábrázolja.

Az 1. oszlopban és az 5. sorban található **NL** (kocsi vissza, soremelés) vezérlőkarakter kódja a 0001 és a 0101 bitsorozatból tevődik össze. Tehát a 00010101 bitsorozat az **NL** vezérlőkarakter leképezése.

A **számokat** az EBCDI-ábécé számjegyenként ábrázolja. A zónarész minden számra 1111, a számjegyrész az egyes számjegyeket közvetlenül bináris szám alakjában tartalmazza.

Példák: A 25 decimális szám az 11110010 és az 11110101, a 128 decimális szám pedig az 11110001, az 11110010 és az 11111000 bitsorozatokból tevődik össze.

Az EBCDI-kód magas redundanciájú *zónázott számábrázolás,* tárolóigénye azonban nagy.

Az előjeleket az utolsó számjegy zónarészében tároljuk: az 1100 bitsorozat a pozitív, az 1101 pedig a negatív előjelet tárolja.

Példák: a +25 decimális számot a számítógépben az 11110010 és az 11000101 bitsorozat; a –25 számot pedig az 11110010 és az 11010101 bitsorozat ábrázolja.

Tömörített ábrázolás. A számábrázolásnak ez a módja a tárolóterületet hatékonyabban használja ki, mint a tömörítetlen ábrázolás, amelyben minden számjegyet 1 byte kódol, az előjelet pedig az utolsó számjegy zónarésze tartalmazza. A tömörített ábrázolás során az összes számjegynél elhagyjuk a zónarészt, tehát minden byte két kódolt számjegyet tartalmaz. A számok előjelét mindig a legutolsó tetrádban kódoljuk (lásd az alábbi példát).

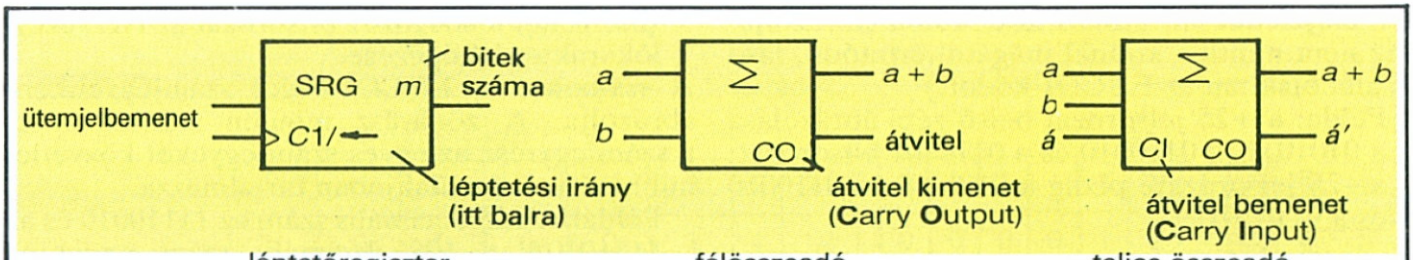
A külső adattárolók, mint pl. a 9 sávos mágnesszalagok esetében legtöbbször az EBCDI-ábécét alkalmazzák. A 9. sáv egy paritásbitet tartalmaz. Ezeket a mágnesszalagokat azonban már szinte alig használják.

Példa: A +123 szám ábrázolása

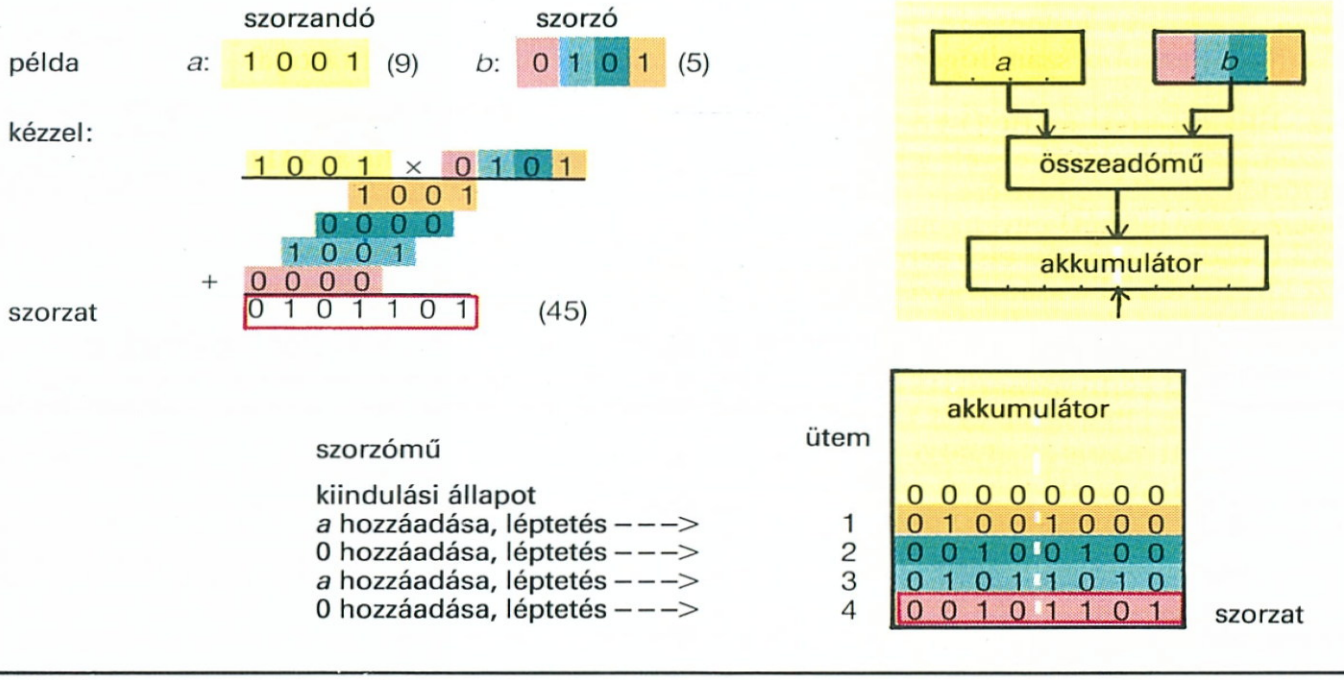
Tömörítetlenül:	11110001	11110010	11000011
	1	2	előjel 3
Tömörítve:	00010010	00111100	
	1 2	3 előjel	

A –123 szám ábrázolása ehhez hasonló:

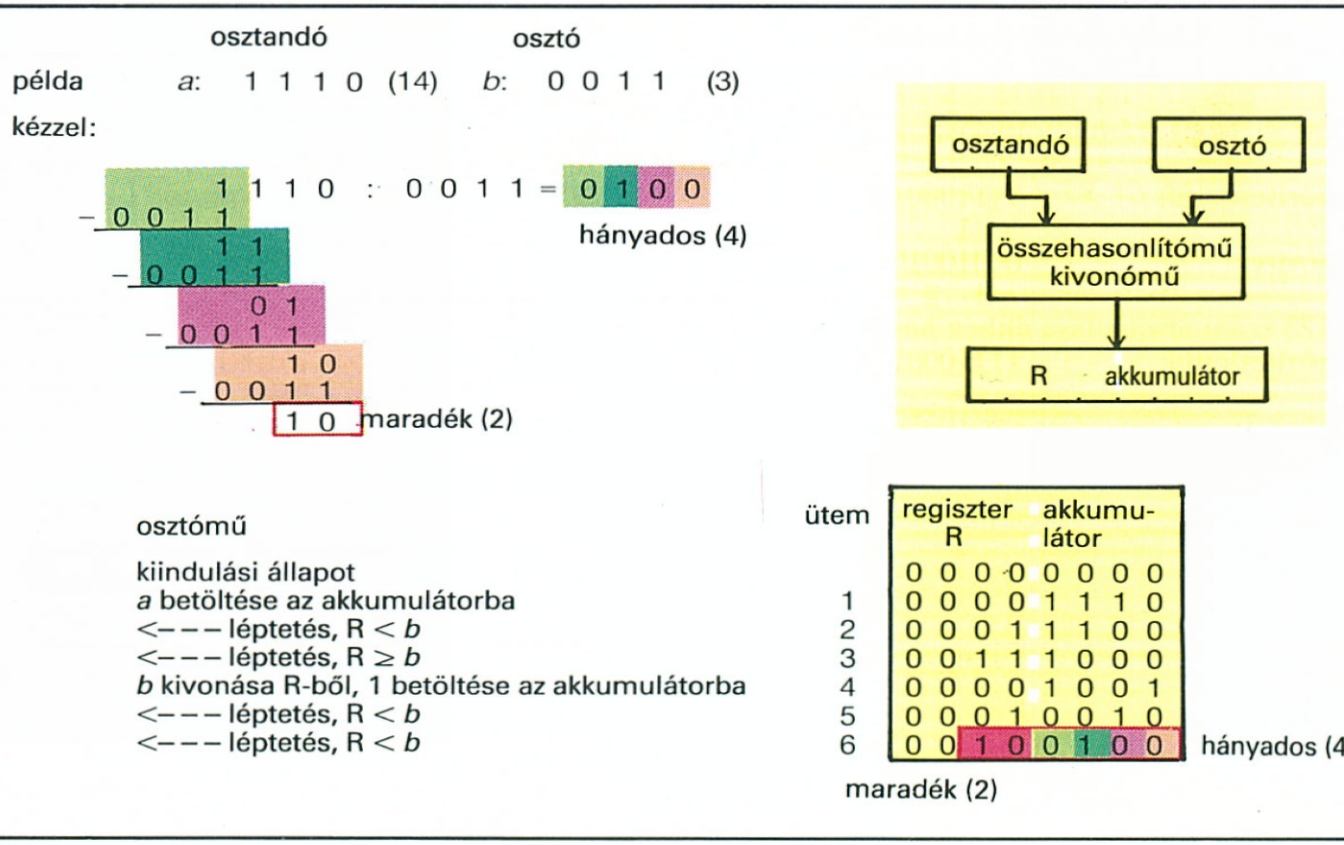
Tömörítetlenül:	11110001	11110010	11010011
			előjel
Tömörítve:	00010010	00111101	
		előjel	



Léptetőregiszter és összeadó kapcsolási jelölése



Két bináris egész szám szorzása



Két bináris egész szám osztása

A digitális számítógépek a számítási alapműveleteket ismételt összeadásra vezetik vissza.

Összeadás

Az összes elektronikus összeadómű alapja a teljes összeadó, amely általában két félösszeadóból és egy VAGY-kapuból áll.

Az **összeadóművek** többjegyű bináris számokat adnak össze. Két alaptípusuk van, amelyek technikai megvalósításában és számítási sebességben különböznek egymástól.

A **soros összeadóművek** két n-jegyű bináris számot jegyről jegyre haladva összegeznek. Az összeadandók számára két bemeneti léptetőregisztert valamint egy teljes összeadót és (az egy ütemmel való késleltetés céljából) egy D-flipflopot tartalmaznak. Kiviteli léptetőregiszterre nincs szükség, ha az egyik bemeneti léptetőregiszter egyidejűleg akkumulátorként szolgál, tehát az eredmény is ebben képződik. Minden órajel hatására mindkét regiszter tartalmából egy helyiérték a teljes összeadóba jut. Az utóbbi a két bináris számjegyet összeadja és a tárolóba továbbítja. Az esetleg fellépő átvitel a D-flipflopba kerül, amely azt a következő órajelig tárolja és a teljes összeadó harmadik bemenetére juttatja. Az összeadás n ütem után befejeződik. A soros összeadóművek technikai felépítése egyszerű, azonban viszonylag lassúak.

A **párhuzamos összeadóművek** két bináris szám n jegyét egyszerre összegzik. Ezért minden számjegypár összeadásához egy teljes összeadó szükséges. Elektronikai megoldásokkal több átvitel egyidejű továbbítása lehetséges, így az összeadás ideje a soros összeadóval összehasonlítva rövidebb. A bonyolultabb technikai felépítés ellenére a modern számítógépek túlnyomórészt párhuzamos összeadóműveket tartalmaznak.

Kivonás

Két bináris szám kivonása ($a - b$) az összeadóműben a-nak és b kettes komplemensének összeadása útján történik. Ha a és b előjele megegyezik, akkor az utolsó átvitelt el kell hagyni.

b kettes komplemense egy NEM-kapu segítségével és ezt követően 1 hozzáadásával történik.

Szorzás

Két bináris szám szorzata többszörös összeadással és az ennek megfelelő helyiérték-eltolásokkal képezhető. A szorzóművet ezért összeadóművek és léptetőregiszterek alkotják. Ha mind a szorzandó, mind a szorzó n-jegyű, akkor az **akkumulátornak** (a szorzat regiszterének) $2n$ bit (dupla szó-) hosszúságúnak kell lennie. Általában a bináris számoknak csak az abszolútértékeit szorozzák össze, az előjelet külön állapítják meg.

A **soros szorzás** technikai megvalósítása egyszerű, mert mindössze egy összeadóművet és néhány léptetőregisztert igényel.

Bár a **párhuzamos szorzás** nagyon gyors, azonban műszaki megvalósítása bonyolultabb. Két négyjegyű bináris szám szorzásához pl. négy teljes összeadó, négy félösszeadó valamint 16 ÉS-kapu szükséges.

Lebegőpontos számok szorzása. Először a két mantissza szorzatát, majd a kivetők összegét képezzük, végül a helyiértékeket megfelelően eltoljuk.

Először mindkét mantisszát egész számmá kell alakítani. Egy n helyiértékű mantissza esetén ez úgy történik, hogy a két bináris számot 2^{n-1}-nel meg kell szorozni.

Az n-jegyű **egész számokat** jegyenként, és csak egy számjeggyel szorozzuk meg. Mivel ez a számjegy vagy 0, vagy 1 lehet, minden ilyen szorzás eredménye vagy 0, vagy a szorzandóval egyezik meg. A végeredmény kiszámításához a részeredmények összegét kell képezni. A szorzás tehát n-szer ismételt összeadásra vezethető vissza.

Példa: Az $a \times b = 1001 \times 0101$ szorzás lépései.

Mivel két négyjegyű bináris számról van szó, az akkumulátornak legalább 8 jegyet kell tárolnia.

Az akkumulátor törlése (>00000000<). a és b betöltése a két bemeneti regiszterbe.

b első (jobb szélső) jegyének leolvasása. Ez 1, tehát $1 \times a = a$ értéket kell hozzáadni az akkumulátor bal oldalához (>10010000<). Az akkumulátor léptetése egy jeggyel jobbra (>01001000<).

b második jegyének leolvasása. Ez 0, tehát $0 \times a = 0$ érték hozzáadása következik (>01001000<). Léptetés (>00100100<).

b harmadik jegyének leolvasása. Ez 1, tehát $1 \times a = a$ érték hozzáadása következik (>10110100<). Léptetés (>01011010<).

b negyedik jegyének leolvasása. Ez 0, tehát $0 \times a = 0$ érték hozzáadása következik (>01011010<). Léptetés (>00101101<).

Ezzel a szorzás befejeződött, az eredmény (>00101101<), azaz decimális alakban 45) az akkumulátorban található.

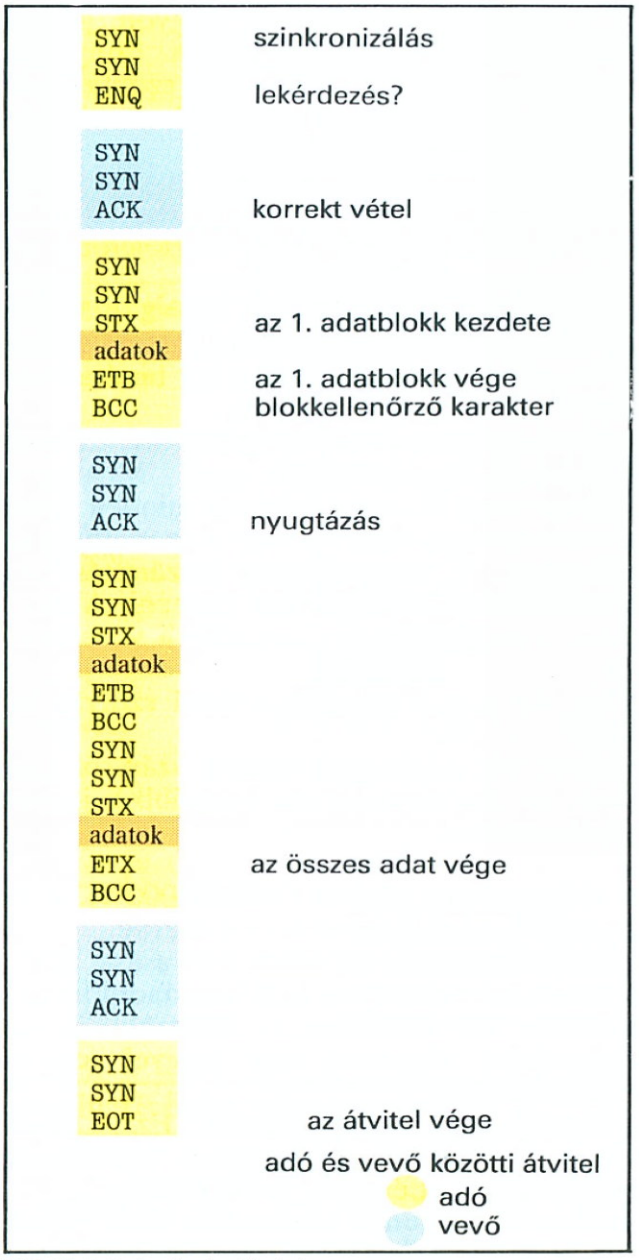
Különleges esetekben a szorzás sebessége növelhető.

Példa: különleges kapcsolásokkal közvetlenül képezhető a számok négyzete. Ennek segítségével két szám szorzását összeadásra lehet visszavezetni, mivel

$a \times b = 1/4((a + b)^2 - (a - b)^2)$.

Osztás

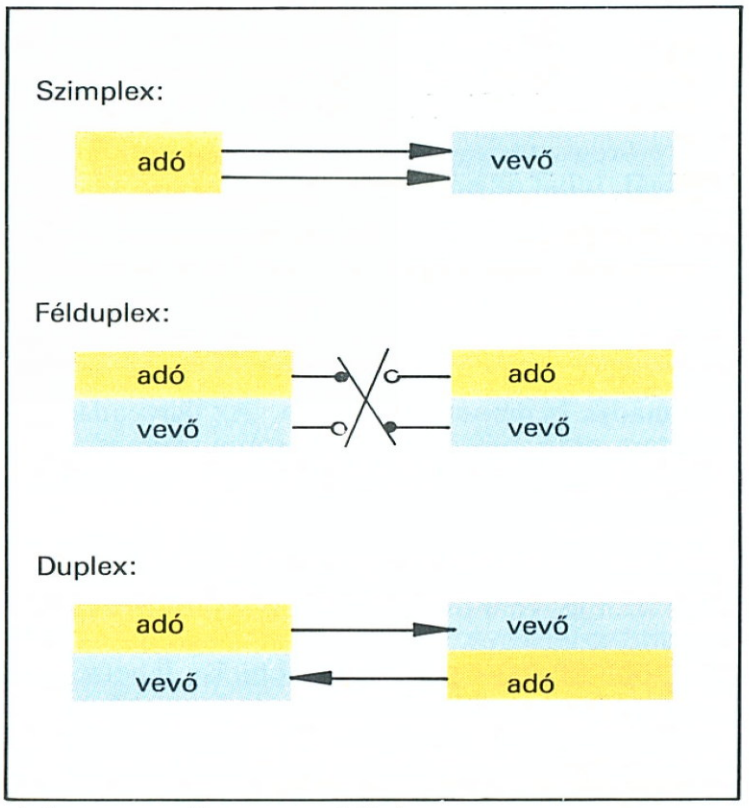
Két szám osztása az osztóműben – a kézzel végzett osztáshoz hasonló módon – történik. Az osztandó és az osztó jegyenkénti összehasonlításának eredménye csak 0 vagy 1 lehet. Két n-jegyű bináris egész szám hányadosa legfeljebb n bit helyet foglal el, az osztás maradékát külön regiszter tárolja.



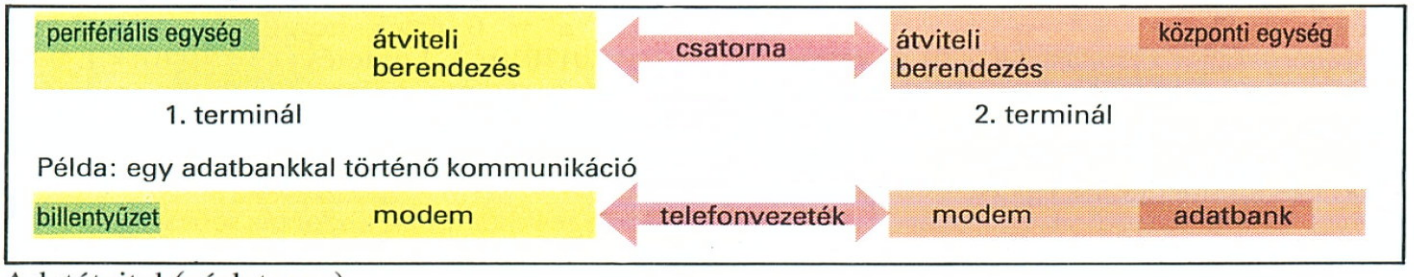
BDC jelek átviteli folyamata (szinkron, félduplex, hibamentes)

adatcsatorna	csatornakapacitás (baud)
telex	50
idegszál	800
vonalválasztásos telefon	2400
állandó kiépítésű telefon	9600
datexvezeték	64 000
száloptika	$> 2 \times 10^6$
tv-csatorna	$1{,}3 \times 10^8$
optikai kábel	$> 1 \times 10^9$

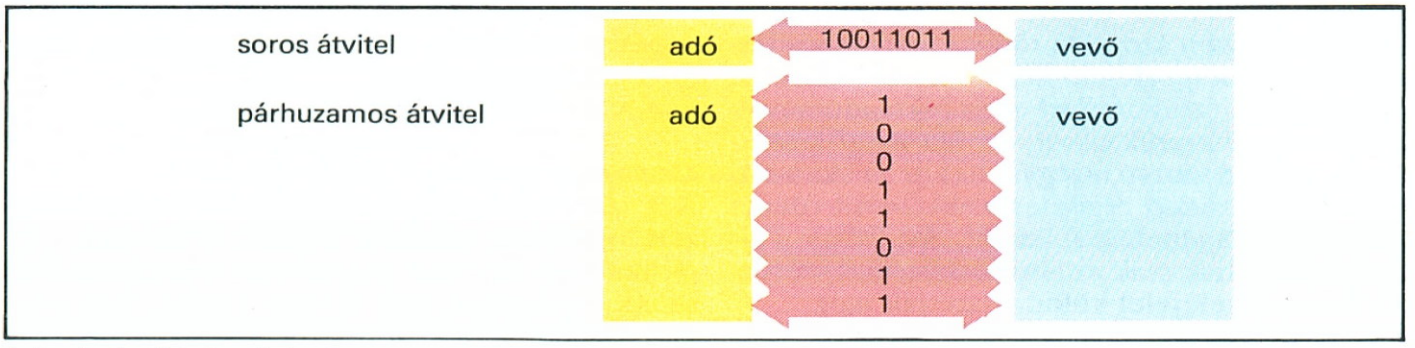
Csatornakapacitások



Adatcsatornák üzemmódjai



Adatátvitel (vázlatosan)



Adatátviteli lehetőségek

A perifériás egységek (billentyűzet, olvasófej, külső tár, nyomtató, rajzgép, meghajtók stb.) és a számítógép központi egysége között adatcsere (számok, jelek, utasítások cseréje) játszódik le. Az adatátvitel egyes adatvonalakból (adatcsatornákból, 103. o.) álló vezetékeken keresztül történik. Az **adatátviteli protokoll** előre rögzíti az átvitel technikai részleteit és az átvitelhez felhasznált algoritmusokat: az alkalmazott kódokat, a maximális átviteli sebességet, a vétel nyugtázását, az átviteli hiba esetén foganatosítandó intézkedéseket stb.
Az adatátvitel során a következő fontos szempontokat kell szem előtt tartani:
1. Lehetőleg független legyen a perifériás készülékek típusától.
2. A perifériás készülékeknek az adatokat ugyanakkora sebességgel kell felvenni, amekkora sebességgel a központi egység leadja őket; ezt általában puffertár (közbenső tár) segítségével oldják meg.
3. Az egyes perifériális készülékek egymástól függetlenül működjenek.

Elnevezések
A következő elnevezések nemcsak a periféria és a központi egység közötti adatcserére, hanem az egyes állomások közötti adatátvitelre általánosan is érvényesek.

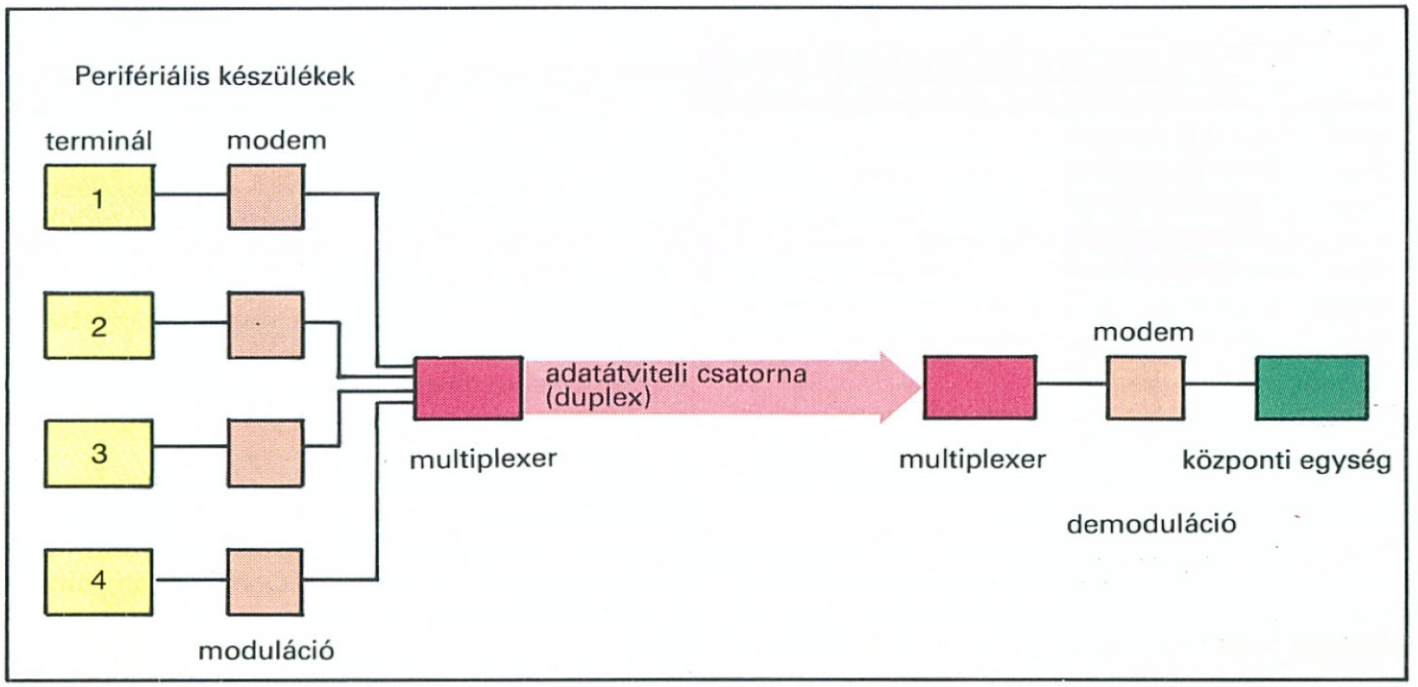
Online. A kapcsolat közvetlen és késleltetésmentes.
Példák: A párbeszédes üzemmód billentyűzet segítségével és a telefonbeszélgetés is *online* adatcserét jelent.
Offline. Adathordozón keresztül megvalósuló közvetett kapcsolat. Az adatátvitel időben késleltetett.
Példák: Egy program floppydiszkre másolása és későbbi beolvasása valamint a távirat is *offline* adatátvitelt jelent.
Az **adatátvitel sebessége** annak a sebességnek a mértéke, amellyel az adatok átvitele az egyes állomások között lejátszódik. Az adatátvitel sebességét alapvetően a vezeték sávszélessége határozza meg.
A **baud** – amelyet Jean Maurice Baudot-ról (1845–1903) neveztek el – az adatátvitel sebességének egysége. 1 baud = 1 bit/s.
Az angol nyelvterületen a következő jelölés is előfordul:
bps (angolul: ***b**its **p**er **s**econd*).
1 bps = 1 baud.
A **vonalkapacitás** egy adatvezeték maximális adatátviteli sebességét adja meg. Ennek túllépése esetén az információ egy része elveszik.

Adatvezeték, adatátviteli út
Az információ szállítása adatvezetékeken történik. Villamos vezetékek, optikai kábelek és rádióhullámok szolgálhatnak erre a célra. Ha a vezeték hossza meghaladja az 1 kilométert, akkor megállapodás szerint **távadatátvitelről** beszélünk.
A **soros vezetékek** az adatokat bitenként egymás után továbbítják. Ehhez egyetlen adatvezeték elegendő.
Példa: A billentyűzeten és a villamos vezetékeken keresztül az adatok soros üzemmódban áramlanak a központi egységhez. A központi egység nem képes egyidejűleg ugyanezen a vezetéken az eredményt visszafelé eljuttatni a perifériához.
A **párhuzamos vezetékek** az adatokat egyidejűleg és egymástól függetlenül továbbítják.

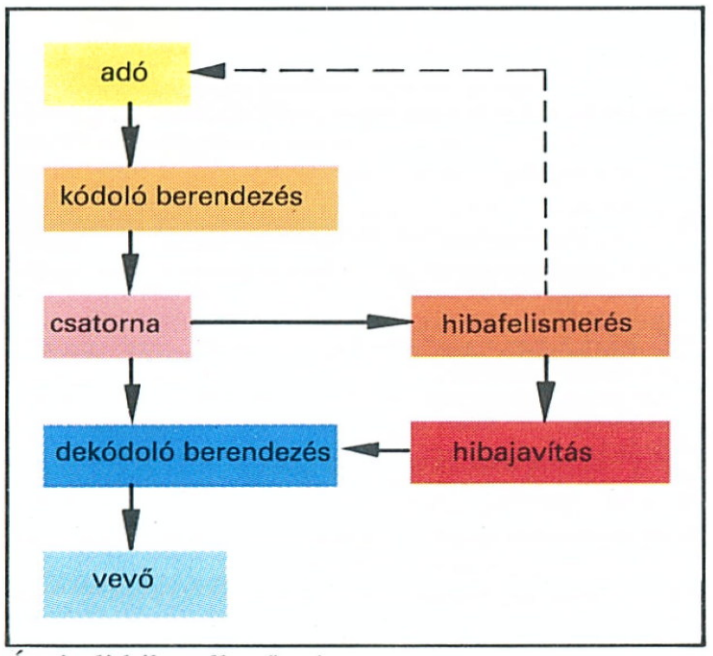
Az egyes adatvezetékek különböző üzemmódban működnek:
Szimplex üzemmód. Az adatátvitel csak egy irányban folyik. A vezeték egyik végén mindig az információadó, a másikon a vevő található.
Példák: rádió, televízió.
Félduplex üzemmód. Váltakozva, mindkét irányban lehetséges az adatátvitel.
Példa: a telefaxgép adásra vagy vételre is kapcsolható.
Duplex üzemmód. Az információadás és -vétel egyidejűleg mindkét irányban megvalósítható.
Példa: telefon.
Multiplex üzemmód. Egy adatvezeték több perifériális egységet köt össze a központi egységgel, vagy több központi egységet a periféria egyetlen készülékével. Az egyes komponensek között a kapcsolat megfelelő időzítését a **multiplexer** biztosítja.
Példa: Egy gyors számítógép a multiplexer segítségével ugyanazon az adatvezetéken keresztül több lassú nyomtatóhoz is továbbíthat adatokat.
Időosztásos multiplex módszer. Az adás és a vétel (vagy a különböző készülékek) számára megegyezés szerint periodikusan ismétlődő – legtöbbször nagyon rövid – időintervallumok állnak rendelkezésre.
Frekvencia-multiplex módszer. A párhuzamos adatátvitelt ugyanazon a vezetéken különböző vivőfrekvenciák teszik lehetővé.

Az adatátvitel aszinkron vagy szinkron módon valósulhat meg:

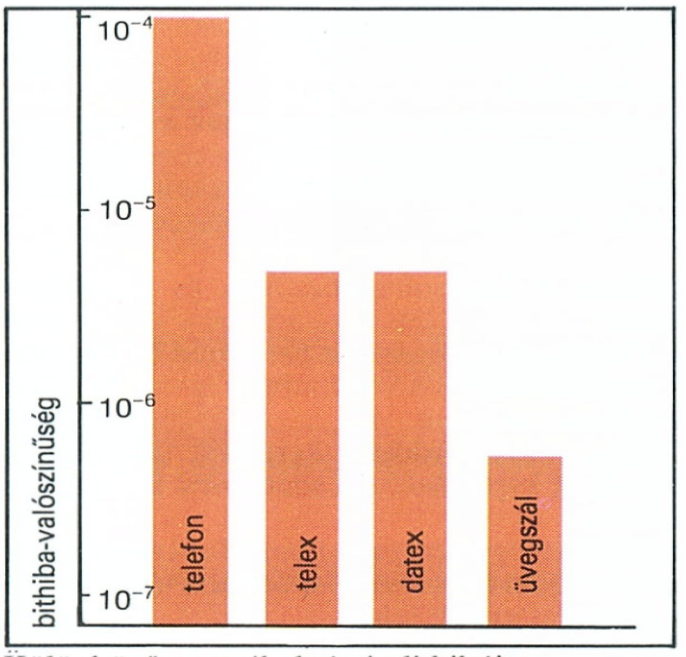
Aszinkron átvitel. Minden jel (karakter) egymástól elválasztva, tehát start- és stopbit közé ágyazva kerül átvitelre. Az egymást követő jelek időbeli távolsága tetszőleges lehet.
Szinkron átvitel. Egy egész adatblokk egyszerre kerül átvitelre. Az adatblokk előtt leadott (ún, ütemadó) jel szinkronizálja az adót és a vevőt, egy további jel pedig az adatblokk végét jelöli. Az átviteli sebesség magas, azonban az adatblokkot az átvitel előtt általában egy közbenső tárolóban kell elhelyezni.



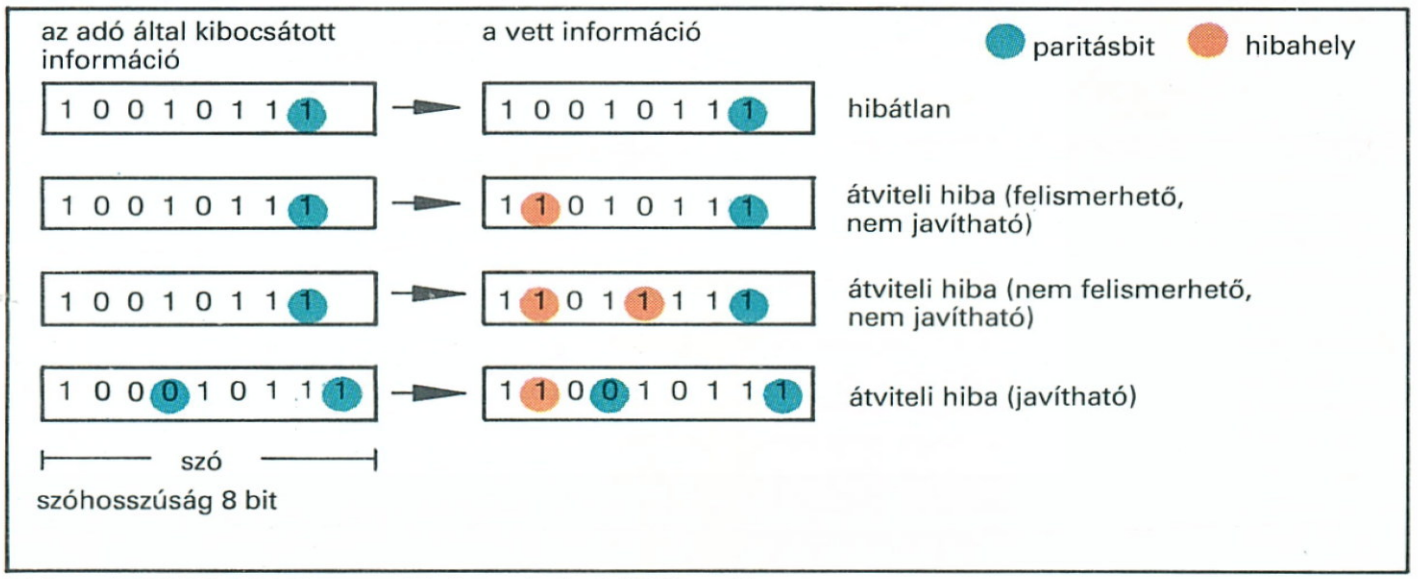
Táv-adatátvitel



Átviteli hiba ellenőrzése



Különböző vezetékek átviteli hibája



8 bites ASCII-kóddal megvalósított átvitel ($DD = 2$)

Az adatvezetékeket meghatározott jelek, pl. villamos áramimpulzusok, fényjelek, szinusz alakú feszültségingadozások átvitelére telepítik. Ezen kívül a központi egység is csak meghatározott jeltípust, pl. bináris áramimpulzust képes feldolgozni. A jeleknek az átvitel ill. a feldolgozás előtti illesztését *modulációnak* vagy *demodulációnak* nevezzük.

A **modem** (**mo***dulator*/**dem***odulator*) a szó szűkebb értelmében olyan készülék, amely az analóg/digitális villamos jeleket digitális/analóg jelekké alakítja át. Az átvitel legtöbbször frekvencia- vagy fázismodulált váltakozó feszültség alakjában történik. Az átvitelt végző modemek szinkronban vannak és egymással közvetlen villamos kapcsolatban állnak. A modem elsősorban telefonvezetékeken keresztül történő adatátvitelre, pl. interaktív videotex vagy elektronikus levelezés (angolul: *E-mail*) továbbítására szolgál.

Az **akusztikus csatolók** a digitális villamos jeleket meghatározott frekvenciájú hangrezgésekké, és fordított irányban a hangrezgéseket villamos jelekké alakítják át. A csatoló és az adatvezeték között nincs közvetlen villamos érintkezés.

Példa: Az adatbankokkal PC segítségével történő kapcsolatteremtés során egy akusztikus csatoló a számítógép kimenetén megjelenő digitális áramimpulzusokat különböző (a 0-t 1180 Hz, az 1-et pedig 980 Hz) frekvenciájú hanggá alakítja át.

Az **optikai csatolók** olyan készülékek, amelyek a villamos jeleket optikai jelekké alakítják át és fordítva. Az optikai csatolók terminológiája még nem teljesen kiforrott.

Kódok

A kódoló berendezések az adatokat az adatvezetéknek optimálisan megfelelő jelsorozattá alakítják (rejtjelezik). A dekódoló berendezések a kódolt adatokat dekódolják. A kódok tehát egy – a szó legtágabb értelmében vett – ábécét egy másik ábécébe képeznek le. Kódolás után minden egyes jelnek általában egy jelsorozat, ún. **szó** felel meg. A szóban lévő jelek számát **szóhosszúságnak** nevezzük.

Különböző kódok léteznek (61. o.):

A **Morse-kód** 3 jelet (pontot, vonást és szünetet) használ, a szóhosszúság pedig különböző (1–4 jel).

Az **EBCDI-kód** a 0 és az 1 bináris jeleket használja a betűk, számok és a különleges karakterek kódolására. A szóhosszúság 8 jel.

Az **ASCII-kód** a 0 és 1 bináris jeleket használja, a szóhosszúság 7. Ha a szóhosszúság 8, akkor a járulékos jel a *paritásbit* (lásd az alábbiakban).

Átviteli hibák

Az adatátvitel során hibák léphetnek fel, pl. egy szóban az átvitel után a 0 jel helyett az 1 jel jelenhet meg, vagy fordítva.

A **bithiba-valószínűség** egy hibásan átvitt bit előfordulásának valószínűsége.

Példák: A telefonvezetékben 1 hibás bit átvitelének valószínűsége 1×10^{-4}, vagyis átlagosan 10 000 helyesen átvitt bitben 1 hibásan átvitt bit fordul elő. Datexvezeték: a hiba valószínűsége 2×10^{-6}, vagyis átlagosan 500 000 helyesen átvitt bitre 1 hibásan átvitt bit jut. Az optikai vezetékekben a villamos vezetékekhez képest az előforduló hiba valószínűsége lényegesen kisebb.

Ha egy hibásan átvitt bit valószínűsége 1×10^{-9}-nél kisebb, akkor az átvitelt definíciószerűen „hibátlannak" tekintjük.

Az egy szón belüli **hibafelismerés** legegyszerűbben a **paritásbit-ellenőrzés** segítségével valósítható meg: a szó végén elhelyezett **paritásbit** értéke 1, ha a byte-ban az egyesek száma páros, és 0, ha az egyesek száma páratlan. A dekódoló berendezés a paritásbit-ellenőrzést szavanként végzi.

Ha az átvitel hibátlan, a vevő pl. **...ACK...** (angolul: **ACK***nowledgement*) jelet küld. Ha hibát észlel, akkor a visszajelzés pl. **...NAK...** (angolul: **N***egative* **AcK***nowledgement*); az adó ilyenkor megismétli a megfelelő adatblokkot (szócsoportot).

Abban az esetben, ha egy szón belül egyszerre két jel hibás, a hibaellenőrzésnek ez az egyszerű módja nem működik.

A nagyobb számítógépek a Hamming-távolság segítségével **automatikus hibajavítást** végeznek.

Két azonos hosszúságú szó D **Hamming-távolsága** a két szó eltérő bitjeinek számát jelenti. Pl. a 0100111 és a 0101001 szavak esetén $D = 3$. Egy kód DD Hamming-távolsága a kód azonos hosszúságú szavaira vonatkozó legkisebb Hamming-távolság. A 7 bites ASCII-kód esetében $DD = 1$.

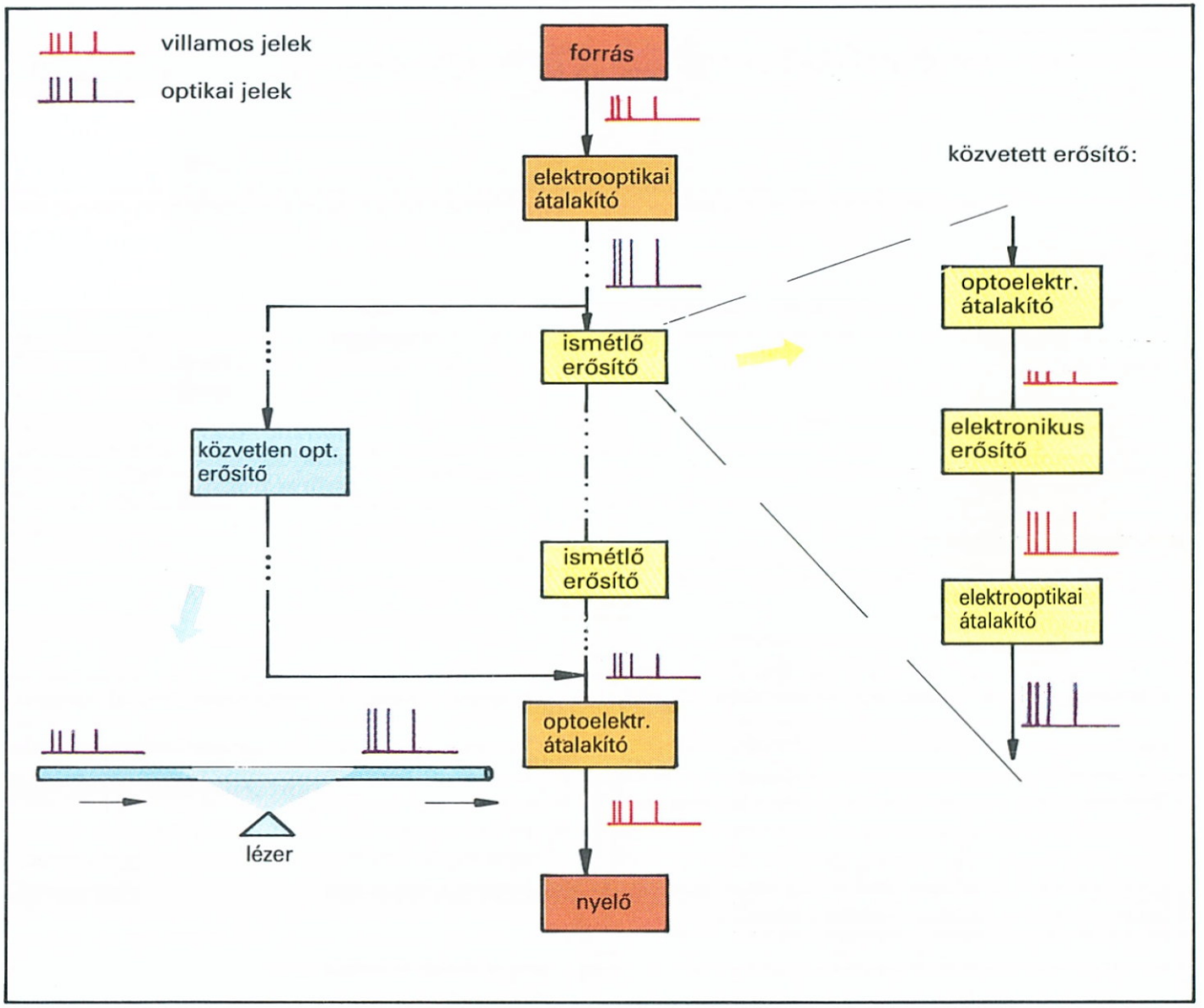
1. *szabály*: Egy szón belül x hibás pozíció akkor ismerhető fel, ha $x < DD$.

Példa: A 7 bites ASCII-kódban a hiba nem ismerhető fel, mert $DD = 1$. Ha azonban minden szó (járulékosan) egy paritásbitet is tartalmaz (mint a 8 bites ASCII-kód esetén), akkor $DD = 2$ és $x = 1$, tehát a szóban 1 hiba megállapítható. A hibás bit helyzete azonban ismeretlen marad.

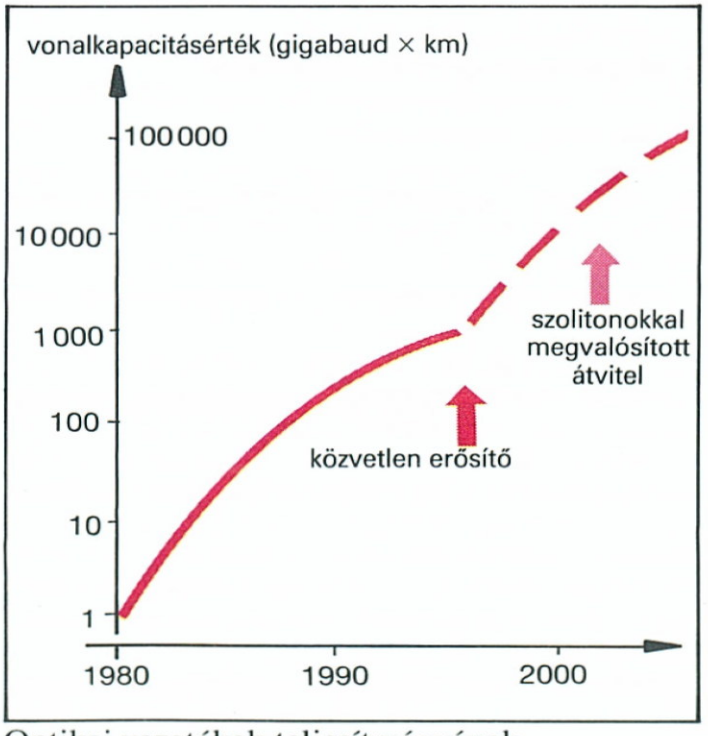
2. *szabály*: Ha egy szóban $DD/2$-nél kevesebb hiba fordul elő, akkor arra is lehetőség van, hogy a szón belül a hiba helyét is megállapítsuk.

Példa: Ha a 8 bites ASCII-kódot egy további paritásbittel egészítjük ki (ez a 9 bites ASCII-kód, amelyben $DD = 3$), akkor a szón belül a hiba helye az ellenőrző egyenletek segítségével kiszámítható. A megfelelő jel automatikusan megváltozik.

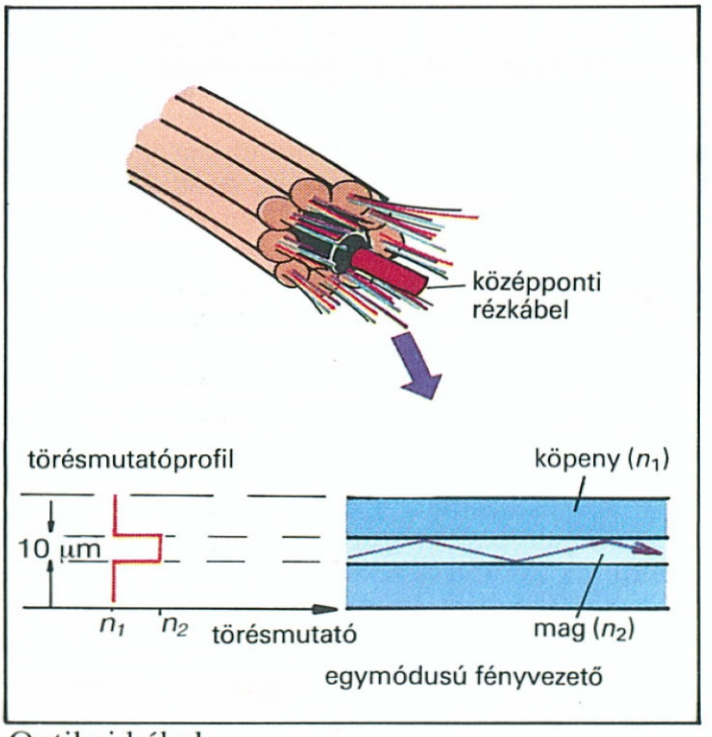
Ha egyazon szón belül két helyen van hiba, az automatikus hibajavítás csak olyan kódok esetén lehetséges, amelyekre $DD > 4$.



Közvetett és közvetlen erősítésű digitális optikai adatátvitel



Optikai vezetékek teljesítményének folyamatos növekedése



Optikai kábel

A villamos összeköttetésekkel max. megabaud nagyságrendű adatátviteli sebesség valósítható meg, az üvegszálas vezetékekkel viszont nagyon kis hibavalószínűségek mellett terabaud tartományba eső átviteli sebességek is elérhetők.

Az **optikai vezetékek** nagyon vékony ($\varnothing < 0{,}1$ mm) hajlékony kvarcüvegszálakból készülnek. Ezeket gyakran műanyagbevonattal látják el, és műanyagcsövekben optikai kábelekké kötegelik. Megfelelő törésmutatóprofil biztosítja, hogy az optikai vezeték magjából a palástra ne lépjen ki fény. A fényforrások infravörös tartományban működő lézerek, amelyek hullámhossza 800 nm és 1600 nm közé esik; ebben a hullámhossztartományban a kvarc fényabszorbciójának minimuma van.

A digitális információt rendkívül rövid fényimpulzusokból és szünetekből álló jelsorozat formájában kódolják.

Ha a fényvezető kábelt analóg jelek továbbítására használják (pl. telefonvezetékekben), akkor a kódolás a vivőfénysugár amplitúdó-, frekvencia- vagy fázismodulációja útján történik.

Információátvitel. Az elektrooptikai átalakító az információforrás (pl. a számítógépkimenet) villamos jeleit optikai jelekké alakítja. A modulált fénysugár, vagyis a fényimpulzusok sorozata az optikai vezetékben tovaterjed, és az átviteli szakasz végén egy optoelektronikai átalakító az információt ismét villamos jelekké alakítja.

Az optikai jel gyengülése

Csillapodás. Az optikai vezetékben a fény amplitúdója a kvarcban történő elnyelés (abszorpció), az inhomogenitásokon fellépő szóródás és a csatolási helyeken történő veszteségek miatt csökken. A veszteség mértékét decibel/kilométer (dB/km) egységekben adják meg.

Példa: Egy optikai vezetékben a jelcsillapodás 1500 nm-es sugárzás esetén 0,1 dB/km, vagyis a jel amplitúdója 1 km úthosszúság megtétele után az eredeti amplitúdóérték 1%-ára csökken. A közönséges ablaküveg csillapítása ennek kb. tízezerszerese.

A csillapodás miatt fellépő veszteség az optikai jel erősítésével küszöbölhető ki.

Diszperzió. A fény terjedési sebessége az optikai kábelben függ a fény hullámhosszától. Ha a fény összetett, akkor a jel az adótól mért távolság növekedésével időben »elkenődik« *(futásidő-torzítás)*. A diszperzió lézer fényforrással kiküszöbölhető.

Mivel az optikai vezeték átmérője és a fény hullámhossza azonos nagyságrendű, a különböző terjedési módusok ugyancsak torzítják a jel időbeli lefutását. Ennek kiküszöbölésére egymódusú optikai vezetéket alkalmaznak.

A **szolitonok** olyan fényimpulzusok, amelyek több hullámhosszt tartalmaznak. Az egyes hullámhosszakhoz tartozó amplitúdók nagysága olyan, hogy a »fordított Kerr-effektus« a diszperziót kompenzálja.

Jelerősítés

A nagy távolságú adatátvitel esetén, de az egyre kiterjedtebb (pl. kábeltelevíziós) hálózatokban is, az optikai kábelek hasznos fényamplitúdóit ismételten erősíteni kell.

A **közvetett erősítők** (angolul: *repeater*) a fényjelet először villamos jelekké alakítják, majd hagyományos elektronikus erősítés után ismét fényjelekké alakítva visszajuttatják az optikai kábelbe. A különböző illesztőegységekben és az elektronikus alkatrészekben átviteli hibák lépnek fel. Egy hibás bit előfordulásának valószínűsége átlagosan kb. 1×10^{-4}. A fényamplitúdó **közvetlen erősítése** során a villamos jelekké történő közbenső átalakításra nem kerül sor.

Példa: *Erbiummal szennyezett száloptika szakasz.* Az erbiumatomokat egy lézerdióda által kibocsátott fény gerjeszti. A gerjesztő energia a fényjeleket indukált emisszió segítségével felerősíti. Ez az optikai erősítő a fénysugarat nem szakítja meg, tehát nem »repeater«. Egy hibás bit előfordulásának valószínűsége 10^{-9} érték alá csökken.

A vonalkapacitás és az információátvitel távolsága

Egy információt továbbító rendszert aszerint lehet minősíteni, hogy rajta maximálisan hány baud továbbítható, és a jelek regenerálása mekkora távolság után válik szükségessé. Az erre jellemző mennyiséget **vonalkapacitás-hossznak** nevezzük, mértékegysége: baud × km. A vonalkapacitás-hossz elsősorban a fény hullámhosszától, spektrális szélességétől, a terjedés módusától, a továbbító közegtől és az erősítés módjától függ.

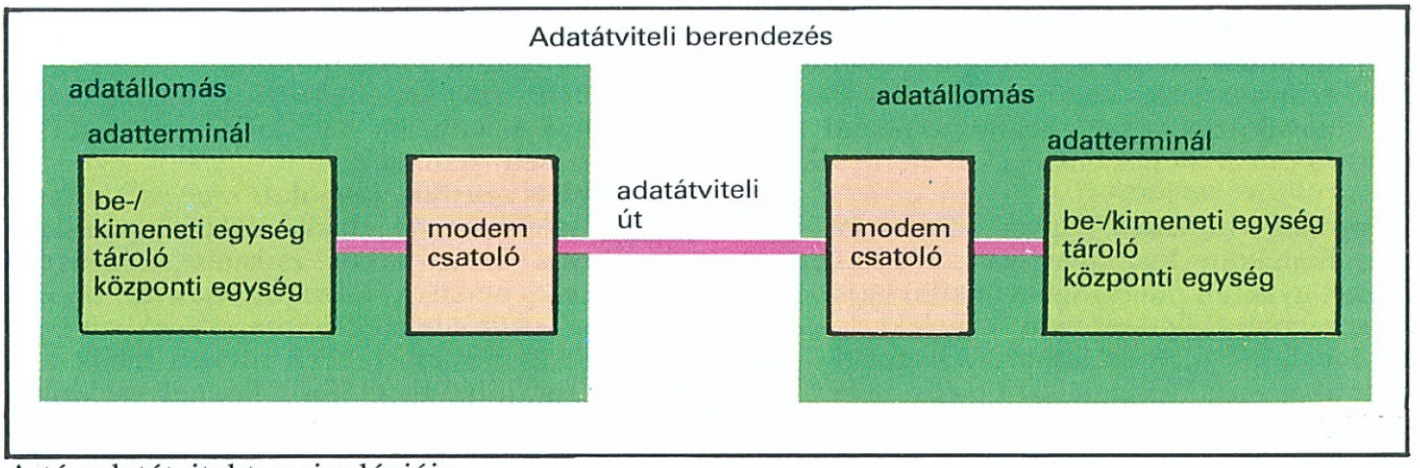
A 70-es évek végén alkalmazott többmódusú optikai szálakban az információtovábbítást 870 nm hullámhosszúságú lézerfénnyel valósították meg, amellyel néhány gigabaud × km vonalkapacitás-hosszúság érhető el. Ez azt jelenti, hogy az optikai kábelt kb. 10 kilométerenként egy-egy repeater szakította meg. A rendkívül tiszta, egymódusú optikai szálak alkalmazása (diszperzió nem lép fel) és a fény hullámhosszának 1550 nm-re történő konvertálása (ezen a hullámhosszúságon van a kvarc abszorpciójának minimuma) néhány száz gigabaud × km vonalkapacitás-hossz elérését teszi lehetővé.

A közvetlen erősítés – a csekély bit hibavalószínűség miatt – a vonalkapacitás-hosszt néhány nagyságrenddel tovább növeli.

A 80-as évek vége óta a fénysugár amplitúdó- vagy fázismodulációjával olyan eljárás valósult meg, amellyel az analóg jelek átvitelének távolságát jelentősen meg lehet növelni.

Példa: Az óceánon keresztül vezető optikai kábelek néhány tucat erősítő alkalmazásával egyidejűleg 500 000 telefonbeszélgetést képesek szinte torzításmentesen továbbítani.

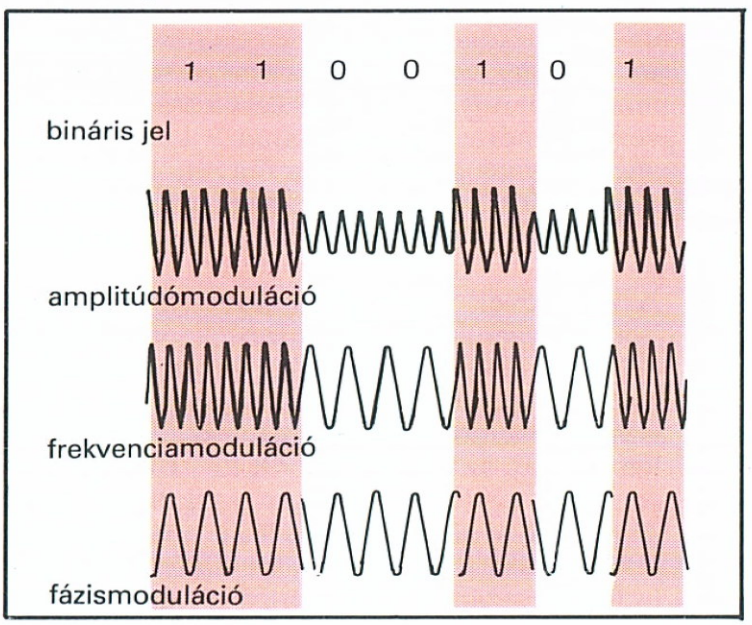
A szolitonokkal történő információátvitel 100 terabaudnál nagyobb átviteli sebesség elérését teszi lehetővé.



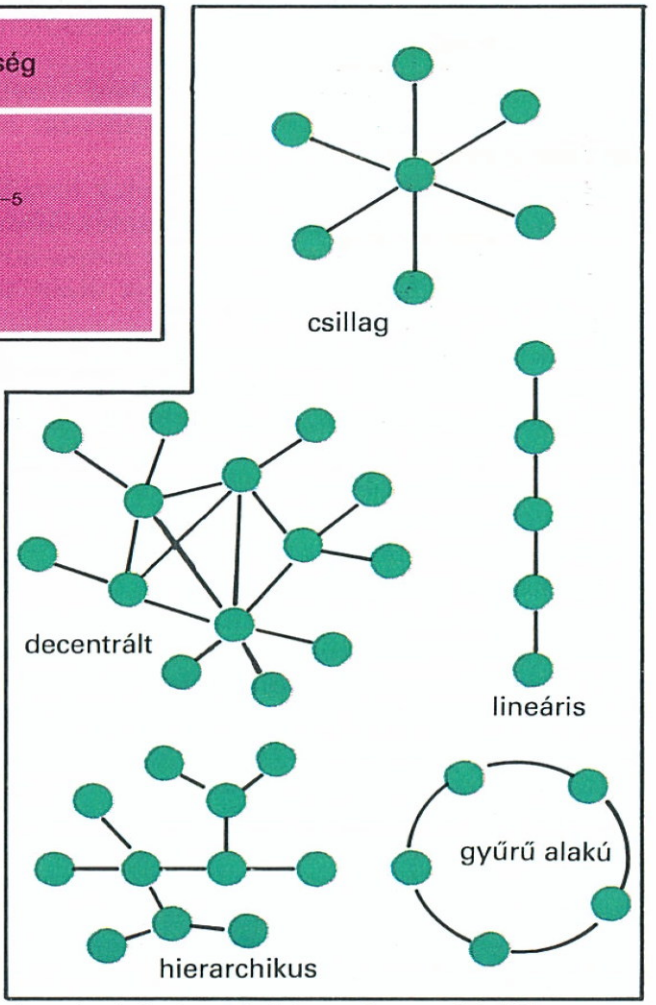
A távadatátvitel terminológiája

szolgáltatás	adatátviteli sebesség (baud)	bithiba-valószínűség
telefonösszeköttetés		
állandó kiépítésű tel.	9600	5×10^{-5}
vonalválasztásos tel.	300 – 4800	2×10^{-4}
telex	50	$10^{-6} - 10^{-5}$
datex		
L	9600 – 48000	kb. 10^{-6}
P	1200	kb. 10^{-5}

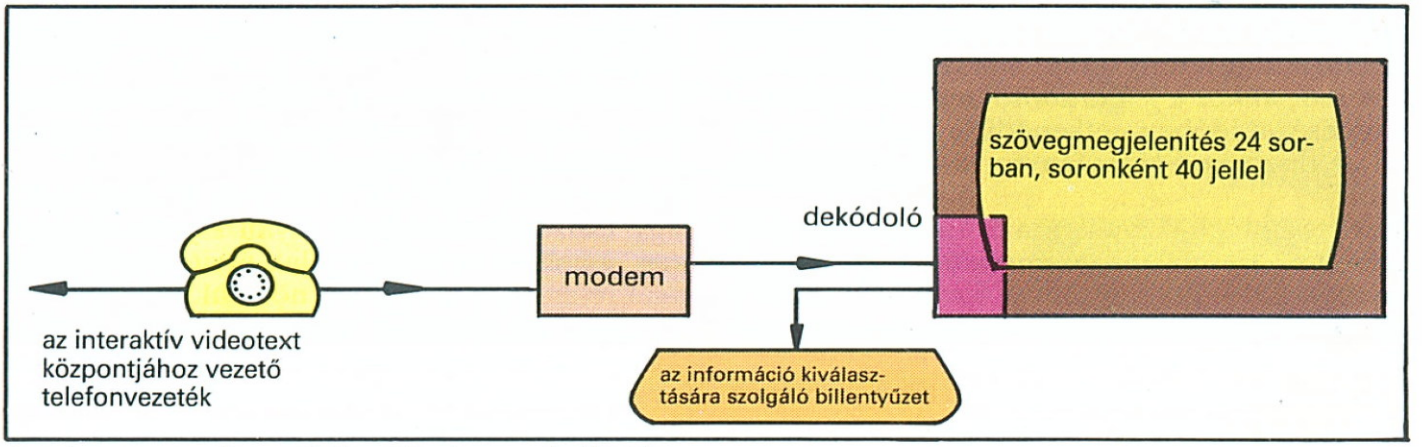
A német posta Datel-szolgáltatása



Modulációs lehetőségek



A számítógépes hálózatok topológiája



A német posta interaktív videotex szolgáltatása

Táv-adatfeldolgozásról megállapodás szerint akkor beszélünk, ha az egymással összekapcsolt számítógépek illetve perifériális egységek közötti távolság meghaladja az 1 kilométert.

Távadatátvitel
A távadatátvitel az adatterminál (be-/kimeneti egység, adattároló, központi egység) kimeneténél kezdődik, az adatátviteli csatorna (modem, akusztikus csatoló, közbenső tároló, villamos- vagy optikai kábel, ritkábban rádió-összeköttetés stb.) igénybevételével megy végbe és egy másik adatterminál bemeneténél végződik.
Az adatterminál és az adatátviteli berendezés közötti átmenetet **illesztőegységnek** (angolul: *interface*) nevezzük. Ezt legtöbbször egy csatlakozóval megvalósított kapcsolat, ill. adatátviteli szabályok rendszere (átviteli protokoll) vagy a kettő kombinációja alkotja. Az illesztőegységek biztosítják azt, hogy a táv-adatfeldolgozás egyes összetevői optimálisan ki legyenek használva és egymás zavarása nélkül működjenek.

Példa: Az X.25 az egyéni felhasználók és a nyilvános számítóközpontok közötti adatátvitel hierarchikusan felépített protokollja.

Németországban kizárólag a TELEKOM jogosult arra, hogy nyilvános adatátviteli lehetőségeket a felhasználók rendelkezésére bocsásson (postai monopólium).
Az **adatátvitel** a különböző vezetékekben általában modulált váltakozó áram formájában történik. A moduláció lehet amplitúdó-, frekvencia- vagy fázismoduláció. *Adatcsomag-átvitel* során az adatáramlás szakaszosan, viszonylag kevés jelből álló **adatcsomagokban** megy végbe. Minden csomag saját címmel rendelkezik, és – közbenső tárolás után – az éppen szabad adatvezetéken kerül továbbításra. Ehhez a módszerhez a folyamatos átvitellel összehasonlítva kevesebb adatvezetékre van szükség. Átviteli zavar esetén nem az egész üzenetet, hanem csak a hibás adatcsomagot kell ismételten elküldeni.

A nagy vonalkapacitású (pl. műholdas) adatátvitel esetén az adatokat az átvitel előtt időben tömörítik (komprimálják), az átvitel után pedig ismét kibontják.

Adatátviteli lehetőségek
A távadatátvitel kezdetben (anyagi és jogi megfontolások miatt) a meglévő összeköttetéseken, pl. telefon- és telexvezetékeken zajlott. A 60-as évek óta léteznek speciális adatvezetékek is.
A német posta három nyilvános adatátviteli lehetőséget kínál, ennek a **Datel**-szolgáltatások (angolul: **Da***ta* **tel***ecommunications*) összefoglaló elnevezést adták. Mindhárom lehetőség kapcsolóközpontokon keresztül megvalósított, eltérő vonalkapacitással és bithiba-valószínűséggel rendelkező összeköttetés.

A **távbeszélő-** és a **telexhálózatot** tulajdonképpen nem a számítógépes adatátvitelre tervezték, azonban bizonyos megszorításokkal erre a célra is használható.

A **datex-hálózat** 1967 óta létezik, és speciálisan a számítógépes adatátvitelre tervezték. A DATEX-L egy központokon keresztül megvalósuló kapcsolatot hoz létre, a DATEX-P pedig a posta adatcsomagküldő szolgáltatását jelenti. Ennek során az adó nem áll közvetlen kapcsolatban a vevővel, hanem a megcímzett üzenetet a hálózatnak adja át, és ezután a hálózat már önállóan gondoskodik az adatcsomag eljuttatásáról. (Az adatcsomag 128 karakterből + vezérlő- és kontrollkarakterekből áll; az adatátvitel sebessége: 1200 baud).

Az **ISDN-hálózat** (angolul: ***I**ntegrated **S**ervices **D**igital **N**etwork*) olyan unierzális, digitális kommunikációs hálózat, amelyen a felhasználók az információtovábbítás összes formáját (beszéd, adatok, képek stb.) igénybe vehetik. Az optikai kábelekre történő átállás után a vonalkapacitás több, mint 2×10^6 baud, a bithiba-valószínűség pedig igen alacsony értékű lesz.

Számítógépes hálózatok
A számítógépeket hálózatokba lehet szervezni, ekkor a gépek a hálózat egyes csomópontjait képezik. A hálózatok alapvetően interaktívak, az adatok minden létrejött kapcsolat során mindkét irányban áramlanak. A *homogén* hálózatokban valamennyi azonos jellegű komponens egymással kompatibilis.

Az **adathálózatokban** az adatokat és a programokat a hálózat számítógépei közösen használják.

Példa: A könyvtári számítógépekkel az egy körzetben fellelhető valamennyi könyvcím hozzáférhető.

Az **erőforrást megosztó hálózatokban** a számítógépek a szoftvert és a különösen nagy teljesítményű komponenseket közösen használják. Ezek heterogén hálózatok.

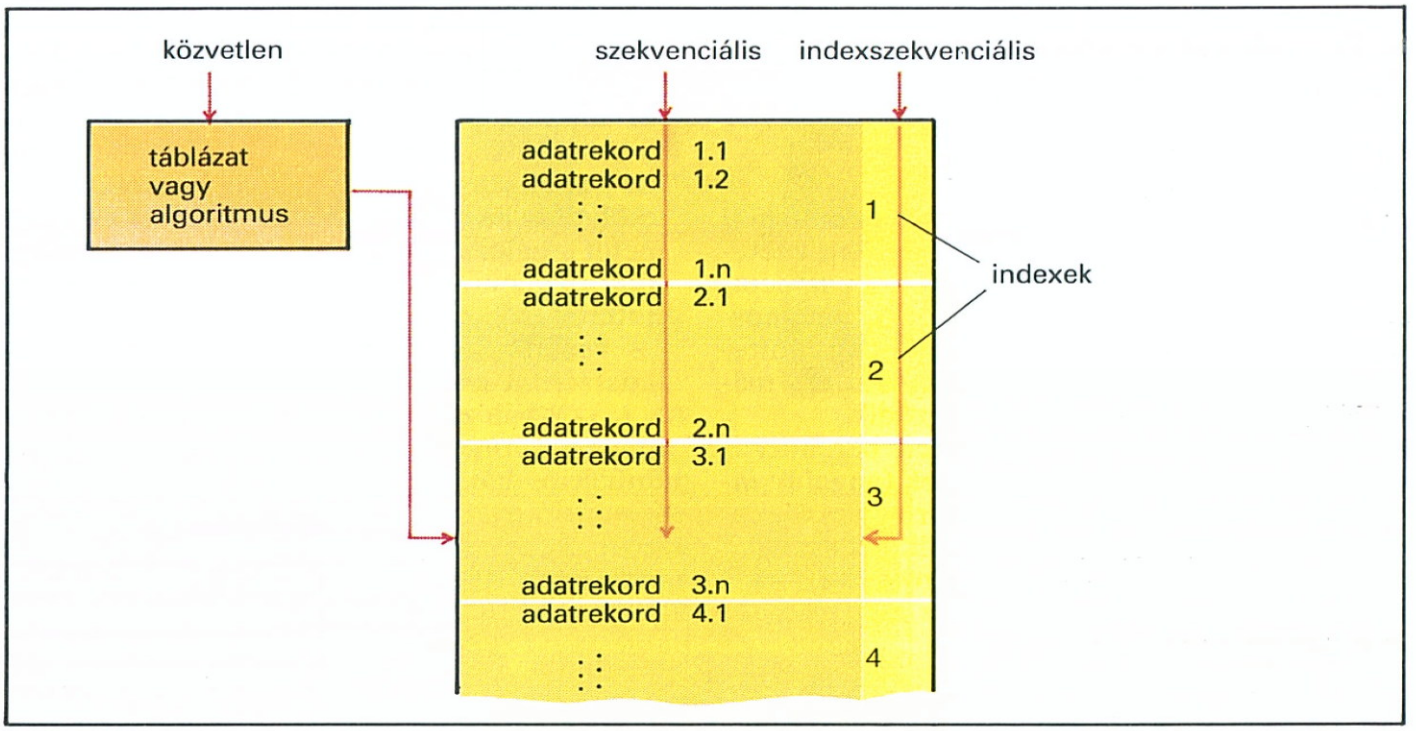
A **munkafolyamatot (a terhelést) megosztó hálózatokban** a mindenkori feladatot a pillanatnyilag szabad számítógépek végzik el.

Egy számítógép-hálózat **topológiája** a feladatoktól, és a rendelkezésre álló számítógépek típusától függ. A legegyszerűbb hálózatszerkezet a csillag- vagy fastruktúra, amelynél a partnerek közötti kommunikáció egy központi (közvetítő) csomóponton keresztül valósul meg. Egyéb topológiai alapszerkezetek: a lineáris, a gyűrű- és a hálószerkezet.

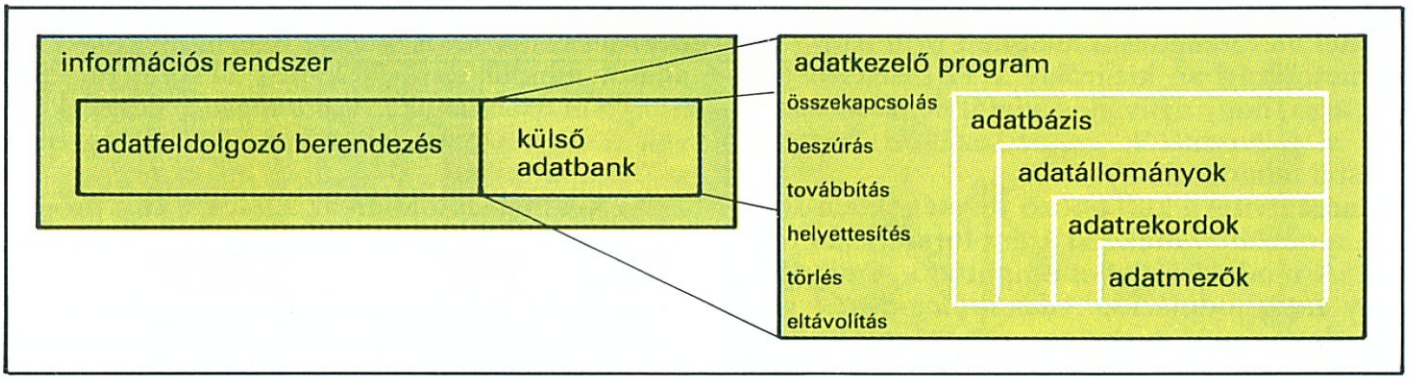
Néhány kiterjedt számítógépes hálózatban a terminálok közötti kommunikáció a legjelentősebb, és a tulajdonképpeni számítási folyamatok alárendelt szerepet játszanak.

Példa: INTERNET (kb. 400000 terminál, és 45 MBund sávszélesség).

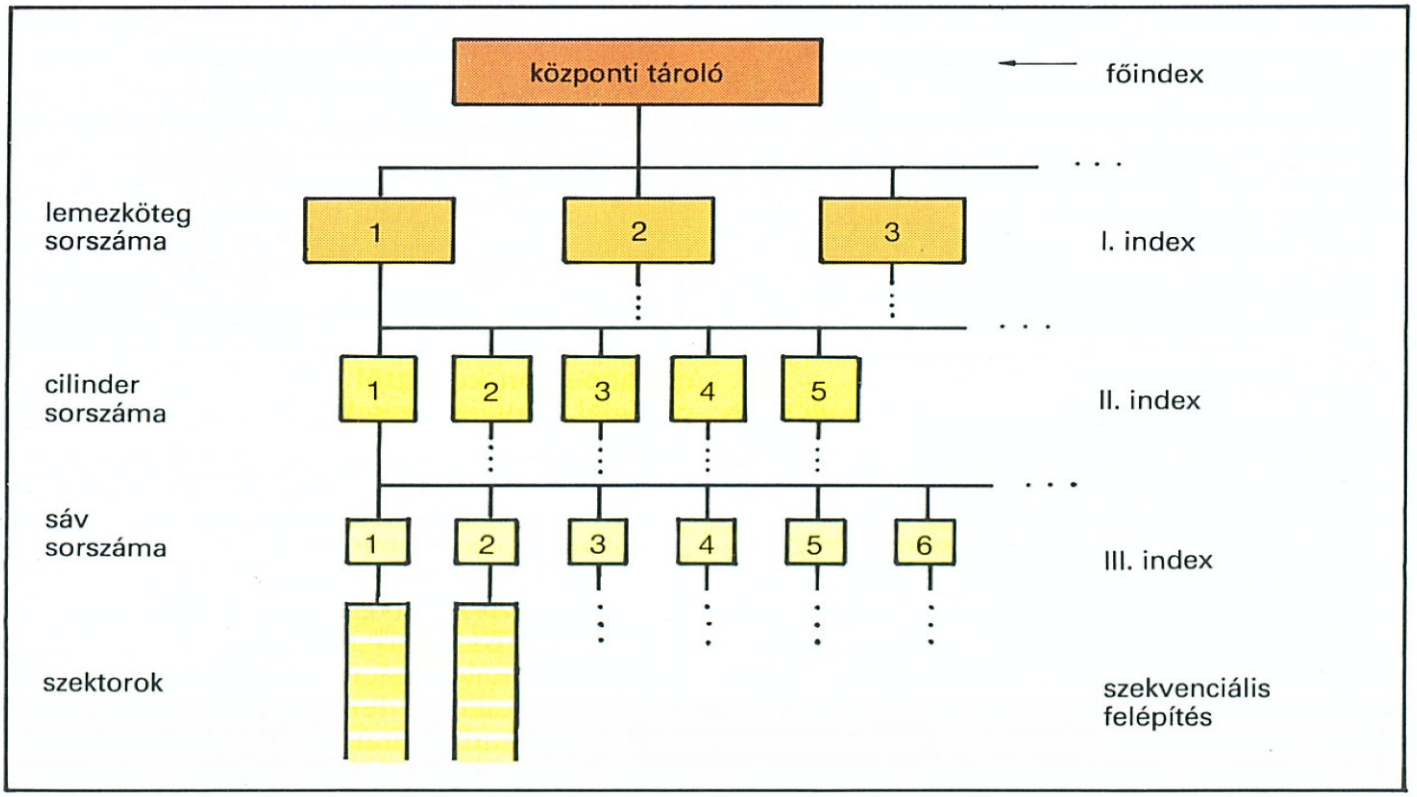
A **nyilvános hálózatokat** Németországban a TELEKOM bocsátja a felhasználók rendelkezésére. A központi számítógép a csillag-topológia szerint telefonvezetékeken keresztül áll kapcsolatban az egyes terminálokkal. A TELEKOM bizonyos felhasználócsoportoknak **magánhálózatokat** is bérbe ad.



Az adathozzáférés lehetőségei



Adott információs rendszer felépítése



Mágneslemezes adatbázis hierarchikus szervezése

Adatgyűjtés már régóta létezik, de csak azóta beszélünk **adatbankról** (angolul: *data base*), amióta ezeket az adatokat számítógép kezeli. Az adatbankok egyrészt az adatok rendkívül gyors és rendezett tárolását valamint elérését teszik lehetővé, másrészt a tárolt információsűrűség is igen nagy.

Minden számítógép saját belső és külső adatbankokkal rendelkezik. A következő megfontolások azonban csak a szorosabb értelemben vett adatbankokra (vagyis a nagyméretű, külső adattárolókra) vonatkoznak.

Az adatbankok gyakran egy átfogó információs rendszer részei, amelynek alrendszerei nagymértékben specializáltak: adónyilvántartás, törvénygyűjtemények, dokumentációk, irodalmi idézetek, raktári nyilvántartások, pótalkatrész azonosító számok stb.

A modern adatbankok **feladata** a nagy adatmennyiségek megbízható tárolása. Ezenkívül arra is szükség van, hogy az adatokat minden jogosult felhasználó gyorsan elérhesse, függetlenül attól, hogy az így nyert adatokat a továbbiakban hogyan hasznosítja. Az adatbanknak egyidejűleg és egymástól függetlenül több felhasználót kell kiszolgálnia. Szükség van arra is, hogy az adatokat bármikor új adatokkal lehessen kiegészíteni, és/vagy a régi adatokat törölni ill. aktualizálni lehessen.

Az adatbankok hálózatszerű összekapcsolását **adathálózatnak nevezzük.** A felhasználó az adathálózattal látszólag ugyanolyan kapcsolatban van, mintha csak egyetlen adatbank szolgáltatásait venné igénybe. Az adatátvitel vezetékek, vagy műholdak segítségével történik. Európai adathálózat: az Euronet-DIANA.

Az adatbankok felépítése

Az adatbank (és az adatbankok rendszere) alapvetően az *adatbázisból* és az *adatbázis-kezelő* (angolul: *database manager*) programból áll, ez utóbbi az adatokat tárolja, keresi stb.

Az **adatbázis** több **adatállományból** (angolul: *file*) áll. Az adatállományok a tartalmi szempontból összetartozó adatokat összefoglalják és valamilyen szempont szerint (pl. ábécésorrendben) rendezik. Minden állomány több, hasonló ismertetőjegyekkel rendelkező adatrekordból (angolul: *record*) tevődik össze, ezek mérete azonos és különböző is lehet.

Példa: egy ember személyi adatai egy rekordot képeznek.

Az adatrekord **adatmezőkből** (angolul: *fields*) áll, ilyenek az adatrekordon belül pl. egy személy neve, születési ideje és címe.

Adatkezelő program. Az adatkezelés a következő fontos feladatokat látja el:

- az adatállományok adatbázissá történő *összekapcsolását,*
- új adatok *beszúrását* az egyes állományokba,
- az adatok meghatározott szempontok szerint történő keresését, másolását és a felhasználóhoz való *továbbítását,*
- meglévő adatok új adatokkal történő *helyettesítését,*
- egyes adatok *törlését,*
- egész adatállományok *eltávolítását.*

Más szempontból az adatbankok az alábbi négy fő részből állnak:

- A *hardverhez* tartoznak az adathordozók, pl. a mágnesszalagok és a mágneslemezek, a be- és kimeneti egységek, az adatcsatornák.
- A *szoftver* foglalja össze az adatbázis kezeléséhez és a felhasználóval történő kommunikációhoz szükséges programokat.
- Az *adatok* a számítógép által olvasható formában, adathordozókon elhelyezett rendezett adathalmazt jelentenek, amelyek az adathálózatokban térben elkülönítve lehetnek jelen.
- A kikeresett adatokat a *felhasználó* vagy a programján belül használja fel, vagy közvetlenül *on line* egy adatkimeneti egységen jeleníti meg. A felhasználónak az adatbázissal való kommunikációhoz egyszerű programnyelvek állnak rendelkezésre. Az adatbank rendszergazdája egyúttal állandó felhasználó is.

  Példa: **IMS** (angolul: **I***nformation* **M***anagement* **S***ystem*). Az IBM 1968-ban egy széles körben elterjedt adatbankrendszert fejlesztett ki. Az adatbázis indexszekvenciális szerkezetű, az adatkezelés pedig a **DL/I** (angolul **D***ata* **L***anguage/***I**) feladatorientált programnyelv segítségével történik.

Adathozzáférés

Mivel az adatbankok általában fontos személyi és gazdasági adatokat tartalmaznak, lényeges, hogy az adatokhoz való illetéktelen hozzáférés kizárható legyen. Minden jogosult felhasználó jelszóval rendelkezik, amely egy vagy több adatállomány elérését teszi lehetővé.

Az adatokhoz való elektronikus hozzáférés szekvenciálisan, közvetlenül vagy indexszekvenciális formában történik.

A **szekvenciális** szerkezetű adatállományokban az adathordozón valamennyi adatot ill. az adatok címét egymást követve (szekvenciálisan) végig kell keresni, amíg a kívánt adatot megtaláljuk. Hosszú elérési idő jellemzi.

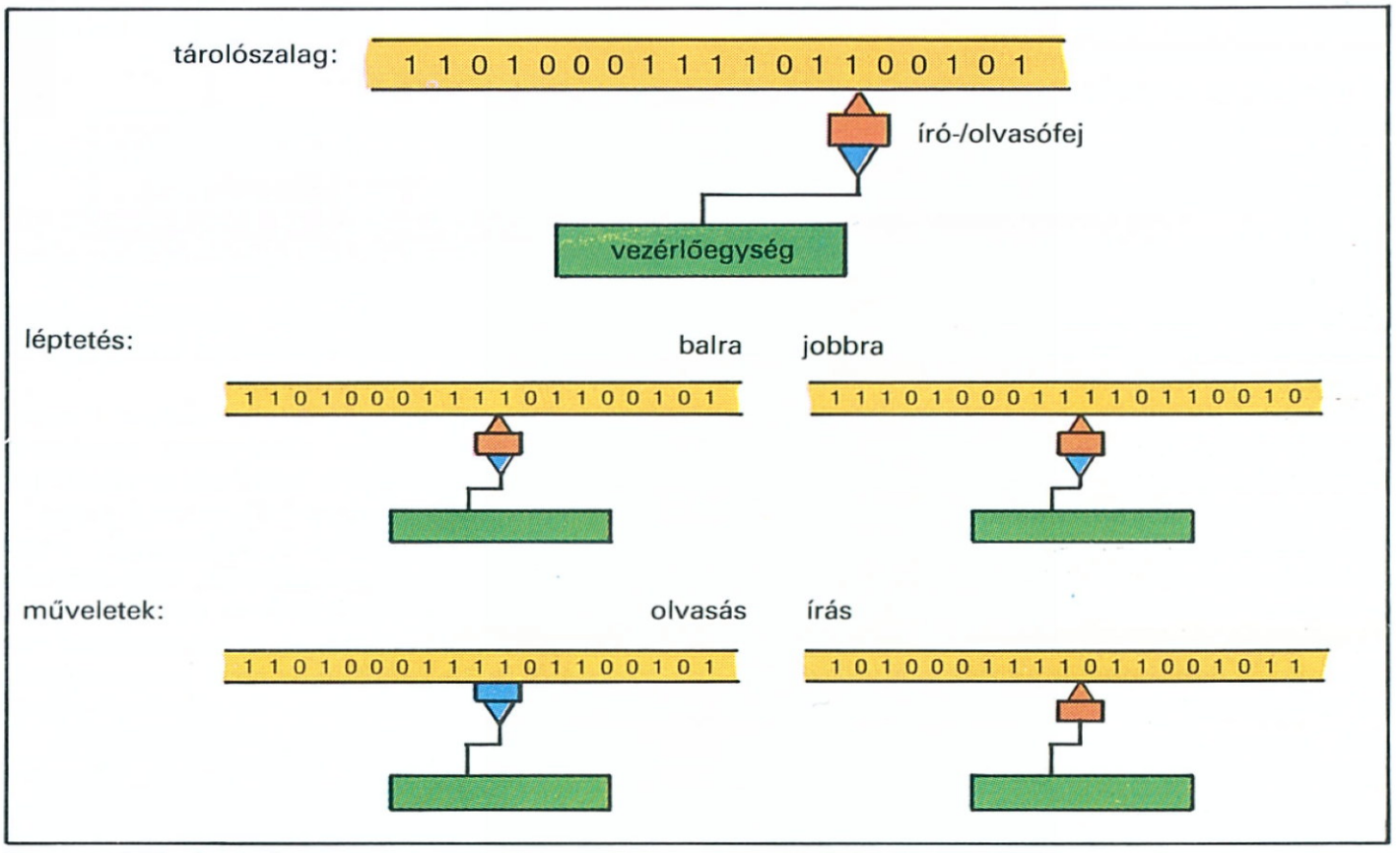
Példa: mágnesszalagos tárolók.

A **közvetlen** hozzáférésű adatállományokban egy olyan algoritmus – vagy olyan táblázat – található, amely a keresett adatrekord címét meghatározza, emiatt az adatok elérési ideje rövid.

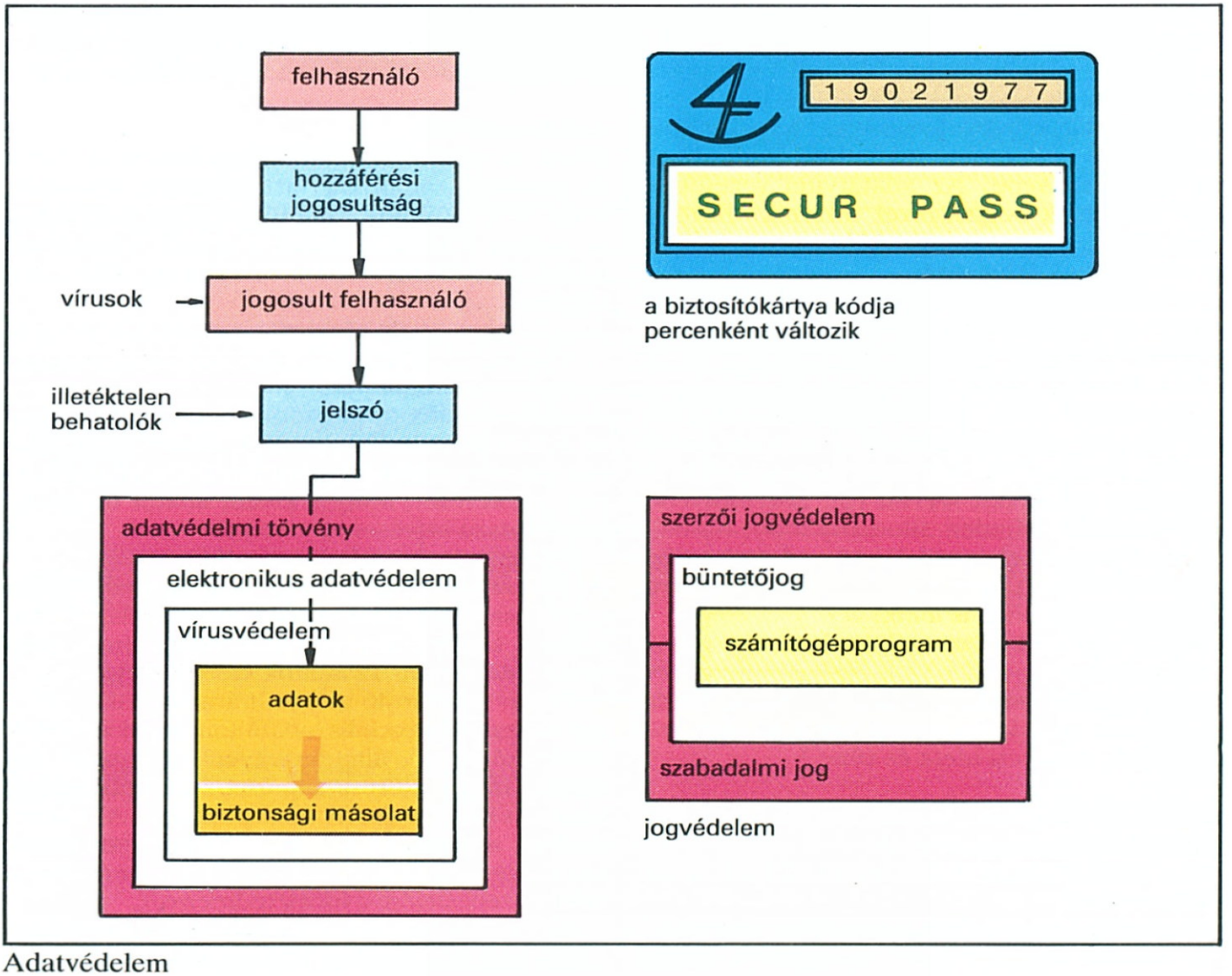
Példa: adattároló integrált áramkörök.

Az **indexszekvenciális** adatállományokat közös ismertetőjegyeik alapján egyenlő nagyságú, **indexszel** megjelölt részekre osztják. Több indexet hierarchikus rendszerbe szerveznek. A hierarchiában a legalsó szint indexein belül az adatokat szekvenciálisan rendezik. Ez a kevert elérési mód közepes elérési időt eredményez.

Példa: mágneslemezes tároló. Minden cilinder egy indexet, minden sáv egy alindexet képez, a sávok minden szektora pedig szekvenciális szervezésű.



Egyszerű Turing-gép



Adatvédelem

Az **adatvédelem** szűkebb értelemben a személyes jellegű (pl. orvosi) adatokat védi attól, hogy illetéktelen személyek megismerjék, kiértékeljék vagy megváltoztassák azokat.

1970-ben Hessen német tartományban lépett életbe a világ első adatvédelmi törvénye. Németországban az 1977-ben életbe léptetett szövetségi adatvédelmi törvény szabályozza a személyes jellegű adatok kezelését. Megengedhetetlennek nyilvánítja a személyi adatok bármilyen feldolgozását, ha azt a törvény nem írja elő vagy ha az érintett személy nem egyezett bele. Munkahelyi és állami **adatvédelmi megbízottak** ellenőrzik az adatvédelmi törvény betartását. A legfelső ellenőrző szerv a **szövetségi adatvédelmi megbízott,** akit a kormány javaslatára a szövetségi elnök nevez ki.

Személyi adatait tekintve Németország minden polgára igényt tarthat:
a személyét érintő, tárolt adatokról történő **felvilágosításra,** az adatok **helyesbítésére,** ha azok a valóságnak nem felelnek meg, a nem egyértelműen helyes adatok **zárolására,**
az illetéktelenül tárolt adatok **törlésére.**

Az **adatbiztonság** magában foglalja az adatok és programok műszaki eredetű, hanyagságból eredő vagy célzatos megváltoztatása ill. törlése ellen foganatosított összes védőintézkedést. Ide tartozik az illetéktelen hozzáféréssel szembeni biztonság is. A biztonság fokozásának módszerei elsősorban technikai jellegűek:

A **biztonsági tartalékmásolatot** (angolul: *backup copy*) mágneslemezen vagy mágnesszalagon tárolják. Ez olyan elkülönített archív adattároló, amely a felhasználó számára adatainak véletlenül (pl. áramszünet miatt) bekövetkező megváltozása esetén legalább azt a lehetőséget biztosítja, hogy az adatállapot legutolsó változatát visszaállítsa.

Visszaállítási tartalékállomány (angolul: *backup file*). Ha egy olyan adatállomány kerül átdolgozásra, amely a mágneslemezen már rögzítve van, akkor a rendszer az új változat mellett a régi változatot is tárolja. Ennek során a megváltoztatott állomány megjelölése az érvényes elnevezés marad, míg a régi verzió elnevezésében a név mellett egy megkülönböztető (pl. SIK vagy BAK) kiterjesztés szerepel.

A jogosult felhasználók **jelszó** (angolul: *password*) megadása után juthatnak az előre meghatározott adatokhoz vagy programokhoz.

A szoftvergyártók az adatok és programok illetéktelen másolása ellen technikai trükkök (pl. a részletek speciális kódolása, másolás esetén önmegsemmisítés, spirális sávok alkalmazása, lézerrel előállított lyukak stb.) segítségével igyekszenek – általában sikertelenül – védekezni. A legtöbb cég azonban megszüntette a védekező intézkedéseket, amióta az USA hadserege nem vásárol másolásvédelemmel ellátott szoftvert.

Sok felhasználó eredeti szoftvert használ, mert ezek (szinte teljesen) hibátlanok, nincsenek számítógépvírusokkal megfertőzve, és bizonyos garanciára is jogosítanak. A shareware valamint a public-domainfreeware szoftverek kivételével a szoftvermásolás tilos.

Hibafelismerő kódok (79. o.).

A **számítógépvírusok** olyan programrészek, amelyek egy számítógéprendszer vagy egy számítógép-hálózat szoftverébe észrevétlenül beágyazódnak és szétterjednek. Rendszerint az ún. gazdaprogram részei és annak lefutása közben fejtik ki (többnyire káros) hatásukat.

Példa: A 80-as évek végén az INTERNET hálózatában egy vírus több, mint 6000 számítógépet megbénított.

A **programférgek** (angolul: *worms*) eredetileg a számítógépben elhelyezett, helyzetüket szabadon változtató, és a számítógéprendszert vagy a számítógép-hálózatot tesztelő programok voltak. Arra szolgáltak, hogy pl. az egyes alkotórészek működését periodikusan ellenőrizzék. Időközben a programférgek gyakran „rosszindulatúak", és a gazdaprogram egyes részeit letörlik. A vírusokkal ellentétben a programférgek nem szaporodnak.

A „trójai faló" típusú programok olyan ismeretlen programrészek, amelyek elsősorban hosszú és áttekinthetetlen szoftverbe vannak beépítve. Külső hívásra („ébresztésre"), vagy a számítógép bizonyos belső folyamatainak hatására aktivizálódnak.

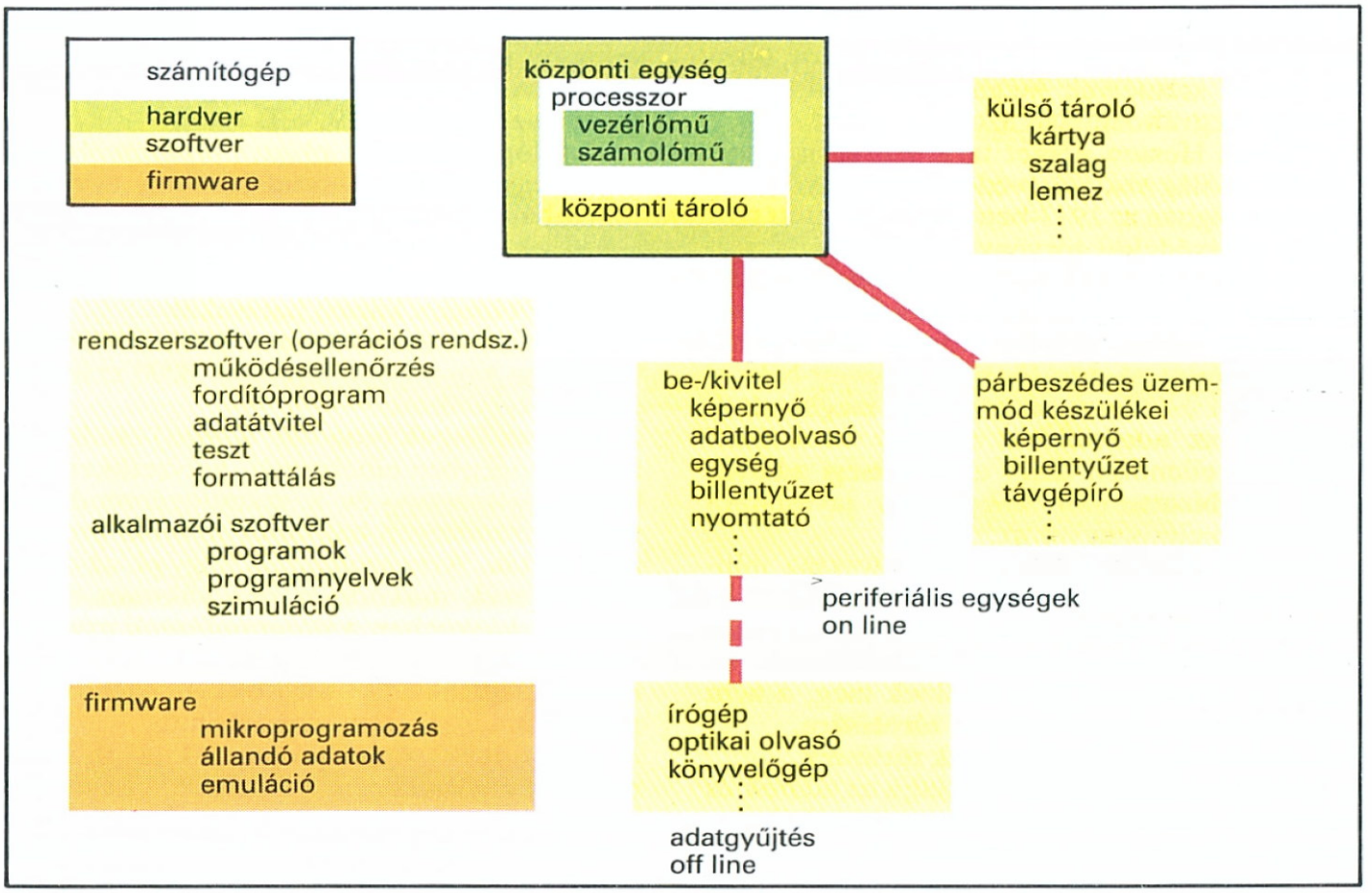
A számítógépprogramok jogi oltalma

A megfelelő rendelkezéseket a különböző törvények tartalmazzák.

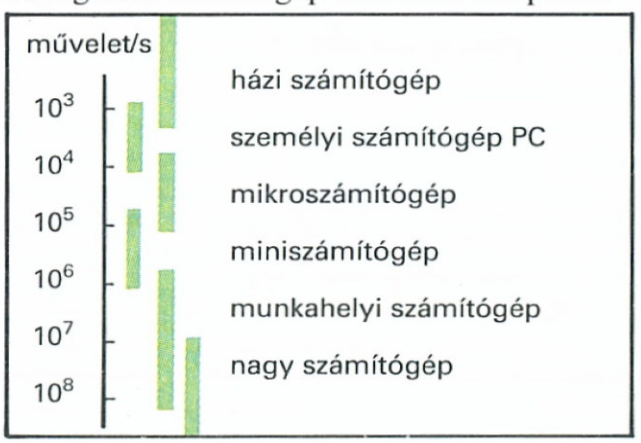
Németországban a **szerzői jogi törvény** (2. §, 1. bekezdés, 1. pont) oltalma az adatfeldolgozás céljait szolgáló számítógépprogramokra is kiterjed. Minden olyan műnek, amely a szerzői jog oltalma alatt áll, ki kell elégítenie azt a feltételt, hogy önálló szellemi terméket képvisel. Németországban a szerzői jog megváltoztatására hozott második törvény 1993. június 24-én lépett életbe, ezzel a régi terminológiát és a sokszorosítás addig érvényben levő szabályait az új nyolcadik bekezdés (UrhG 69a–69g §§: „A számítógépprogramokra vonatkozó különleges rendelkezések") váltotta fel. A programok oltalmazhatóságával szemben támasztott követelmények lényegesen csökkentek, és a fenti időponttól kezdve azonosak a szerzői jogvédelemben részesülő egyéb művekre vonatkozó előírásokkal.

A szoftverkalózkodásra, az adatkémkedésre, a számítógéprendszerekbe történő illetéktelen behatolásra, a szabotázsprogramok és számítógépvírusok előállítóira a **büntetőjog** előírásait kell alkalmazni. Az ide vonatkozó jogi előírásokat Németországban a Büntető Törvénykönyv (StGB) 303. paragrafusa tartalmazza.

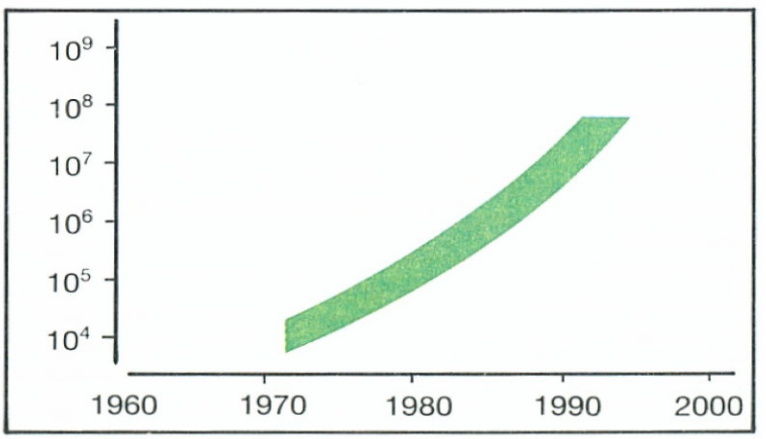
A **szabadalmi jog** (Németországban: 1. §, 2. bekezdés, 3. pont) a szoftver szabadalmazását kifejezetten kizárja. Ha azonban a program egy szabadalmi jogvédelem alá tartozó technikai berendezés része, akkor a megfelelő program is jogvédelemben részesül, összhangban az *Európai Szabadalmi Megállapodás* 52, 2c cikkelyével.



A digitális számítógép általános felépítése



A számítógépek osztályozása



Az előállított PC-k száma

	kapcsolási elemek	két tízjegyű szám összeadásának ideje	
Hőskor	relé	200 ms	
1. generáció	elektroncsövek	1 ms	50-es évek
2. generáció	tranzisztorok diódák	0,1 ms	60-as évek
3. generáció	részben integrált áramkörök	2 μs	60-as évek közepe
4. generáció	nagyintegráltságú áramkörök	100 ns	70-es évek
5. generáció	legnagyobb integr. áramkörök,	5 ns	80-as évek
	megabites chipek	< 1 ns	
	párhuzamos feldolgozás		90-es évek
	neuronális szerkezetű	< 1 ps	
(6. generáció	kapcsolódások		21. század)

A számítógép-generációk

A **számítógépek** (angolul: *computers*) adatok és jelsorozatok feldolgozására szolgáló elektronikus műszaki berendezések.

A számítógépek minőségileg és mennyiségileg is különböznek a teljesen mechanikus működésű számítóeszközöktől.

A számítógépek olyan feladatokat dolgoznak fel, amelyeknek kézzel történő megoldása elképzelhetetlenül hosszú ideig tartana, ill. amelyeket számítógép nélkül egyáltalán nem lehetne megoldani. A számítógépes feladatmegoldás részletes menetét a **számítógépprogramok** írják le.

A számítógépek sora a háztartási gépekben alkalmazott mikroprocesszoroktól a programozható zsebszámítógépeken keresztül egészen az időjárás előrejelzéséhez használt nagy számítógépekig és a globális hadviselés során bevethető szuperszámítógépekig terjed. A számítógép általában *digitális számítógépet* jelent. Az *analóg számítógépek* (25. o.) száma a digitális számítógépekhez viszonyítva napjainkban már nagyon csekély.

- **Hardvernek** nevezzük a számítógép mechanikus és elektronikus építőelemeit, valamint az ezekhez tartozó perifériális készülékeket.
- A **szoftver** a számítógép hibátlan belső működéséhez szükséges valamennyi program (rendszerszoftver) és a feladatok megoldásához felhasznált programok (felhasználói szoftver) összességét jelenti; ezen kívül ide tartoznak a kézikönyvek és a használati utasítások is.
- A **firmware** a gyártó által az egyes alkotórészek mikroprogramozására a számítógépbe beépített szoftvert jelenti, amelyet a felhasználó legtöbbször nem tud megváltoztatni. Ide tartoznak pl. a csak olvasható elektronikus tárolók (ROM) is.

A **számítógépek jelentősége.** A fejlett országokban a számítógép mint számítási és kommunikációs segédeszköz a társadalom szerves részévé vált. Nemcsak a tudomány, a technika és a gazdasági élet területeire hatolt be, hanem a személyi számítógép formájában mindennapi életünk elválaszthatatlan tartozéka lett. Alaptalan az a félelem, amely szerint a számítógép az ember elől elveszi a munkahelyet. Éppen ellenkezőleg, a számítógép megjelenésének hatására rendkívül sok új munkahely jött létre.

A **számítógép-generációkon** nagy vonalakban nyomon követhetők a számítógépek egyes fejlődési szakaszai. A jelenlegi legmodernebb számítógépek az 5. generációhoz tartoznak.

A számítógépek számolási sebessége és tárolókapacitása az elmúlt ötven évben több nagyságrenddel megnőtt. A negyvenes években két tízjegyű szám összeadása még kb. 1/1000 másodpercig tartott, az 5. generációs számítógépek viszont ennyi idő alatt kb. egymillió összeadást képesek elvégezni. A tárolókapacitásnak érdekes módon nincs felső határa, de a számolási sebesség nem lehet tetszőlegesen nagy.

A **számolási sebesség** elméletileg elérhető felső határát a Heisenberg-féle határozatlansági reláció (a kapcsolási időkben fellépő bizonytalanság) és a Schwarzschild-sugár (a foton és a gravitáció rövid hatótávolságú kölcsönhatása) korlátozza. A napjainkban megvalósítható mintegy 1×10^{-11} s kapcsolási idő még messze elmarad a kb. 2×10^{-32} másodperces elméleti alsó határtól.

Turing-gép

Minden fix programozású számítógépet **Turing-gépként** lehet leírni és elemezni. A számítógépek így nem mások, mint univerzális Turing-gépek, amelyek minden algoritmust megfelelő programmal valósítanak meg.

Alan Turing 1936-ban a logikai folyamatok leírására és vizsgálatára megtervezte egy univerzális automata modelljét. Ezt a – Turing-gépnek nevezett – absztrakt modellt később a számítógépekre is alkalmazni lehetett.

A Turing-gép három alkotórészből áll (86. o.):

1. A végtelen hosszú **munkatárszalag** egyes mezőkre tagolódik, amelyek mindegyike egy jelet tartalmaz (vagy üres). A legegyszerűbb esetben bináris jelek foglalják el az egyes mezőket. A szalag lépésenként egy mezővel mozdulhat el előre vagy hátra. Mindig csak egyetlen mező olvasása vagy írása történik.
2. A szalag mezőről mezőre elhalad az **író-/olvasófej** előtt. Csak a fej alatt elhelyezkedő mező olvasható vagy írható.
3. A szalag mozgását a **vezérlőegység** szabályozza. A szabályozás a vezérlőegység állapotának és a vezérlőegységben tárolt állapotváltozás-táblázatnak megfelelően történik.

A vezérlőegységet és az író-/olvasófejet gyakran egy egységnek tekintik, ez az egység a **processzor.**

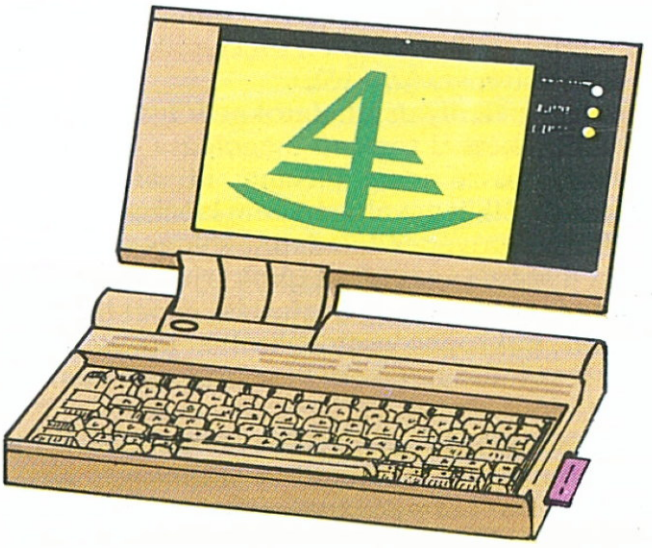
A Turing-gép egy meghatározott állapotban van, amely a beolvasott jelnek illetve a vezérlőegység állapotának megfelelően változhat vagy változatlan marad. Miután a gép végrehajtotta a beolvasott jelnek megfelelő műveletet, új állapotba kerül. A folyamatot a stop utasítás beolvasásakor fejeződik be. A Turing-gép által kiszámított eredmény az összes állapotváltozás következményeként jön létre és a szalag megírt részén jelenik meg.

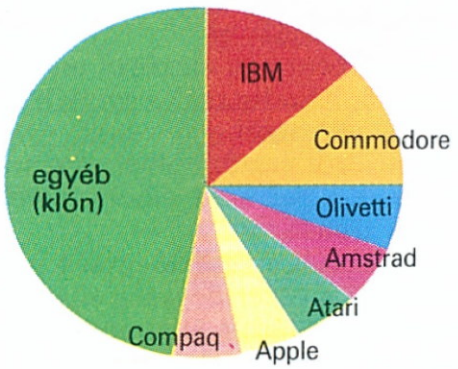
A Turing-gép segítségével lehet tanulmányozni pl. az algoritmusok kiszámíthatóságát vagy a komplexitáselmélet kérdéseit.

1936-ból származik a **Church-féle tétel:** Minden intuitív értelemben kiszámíthatónak tekinthető függvényt ki lehet számítani a Turing-géppel. (Az *intuitív* szó itt azt jelenti, hogy az algoritmust sokan kiszámíthatónak tekintik.)

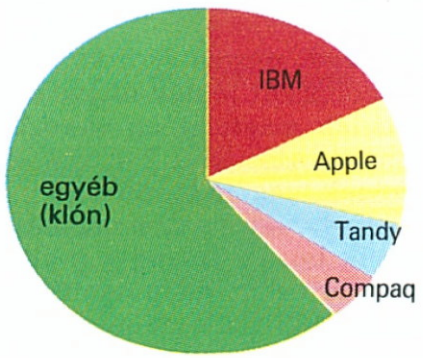
1974	Mark-8	*Radio Electronics*
1975	Altair 8800	*Popular Electronics*
1976	Apple I	*Apple Computers*
1977	Apple II	
	TRS-80	*Tandy*
	Pet	*Commodore*
1981	IBM-PC	*IBM*

Az első személyi számítógépek





Európa



USA

PC-piac megoszlása 1992-ben

PC (laptop) műszaki adatai

mikroprocesszor: Intel 80386 SX/20

80387 társprocesszor számára fenntartott foglalat

órajel 20 MHz

munkatár 2 MB

mágneslemez 3,5 coll

merevlemezes tároló 40 MB

(hátulról megvilágított) LCD képernyő

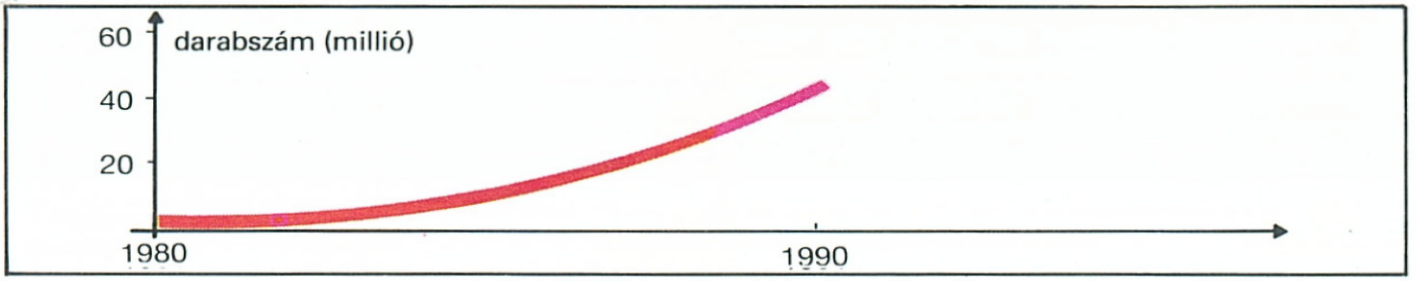
VGA-felbontás: 640×480 pont

beépített illesztő-egységek: nyomtató, 2. lemezmeghajtó, külső VGA-monitor, külső billentyűzet

akkumulátor és hálózati tápegység

szoftver: MS-DOS 5.0, LOTOS MAGELLAN 2.0, WAYS 1.0, további opciók

Laptop-PC



A világon eladott (PC-nél nagyobb teljesítményű) számítógépek darabszámának alakulása

A **személyi számítógép** (**PC,** angolul: ***p**ersonal **c**omputer*) egy kompakt, sőt akár hordozható, nagy teljesítménnyel rendelkező számítógép. Az ára és a teljesítménye alapján az egyszerű házi számítógépek és a *munkaállomások* (angolul: *workstations*) közötti helyet foglalja el. A PC-k egyre több olyan feladatot látnak el, amely korábban csak nagy (angolul: *main frame*) számítógépekkel volt elvégezhető.

A **személyi számítógépek története** az USA-ban kezdődött 1974-ben a Radio Electronics cég Mark–8 elnevezésű építőkészletével. Ez egy Intel–8080 mikroprocesszort és 256 byte-os munkatárolót tartalmazott. A legújabb PC-k Intel 586-os *(Pentium)* mikroprocesszorral készülnek, és max. 4 Gbyte memória közvetlen címzésére képesek.

Az új számítógéptípus az IBM-PC bevezetése (1981 márciusa) után vált népszerűvé. Időközben a világ PC állománya kb. 140 millióra tehető. 1993-ban 32 millió személyi számítógépet adtak el, amelynek 90%-a Intel mikroprocesszort tartalmazott. A PC forradalmasította a számítógépekhez való viszonyunkat. Sokak számára a számítógép először így vált hozzáférhetővé. A PC nemcsak programozhat, hanem azt is lehetővé teszi, hogy a számítási folyamatba közvetlenül beavatkozzunk.

Miközben a személyi számítógépek ára zuhan, teljesítőképességük rendkívüli módon növekszik. A teljes mértékben működőképes PC mérete egyre csökken, már elérte a hordozható írógép nagyságát *(laptop),* sőt egy irattáskában is elférhet *(notebook).*

A személyi számítógépek felépítése

A PC-k moduláris felépítésűek. A központi egység – amely a tulajdonképpeni számítógépet jelenti – a hálózati tápegységből, az elektronikus alkatrészeket tartalmazó alaplapból, a tárolóegységek meghajtóiból és a perifériák illesztőegységeiből áll. A nagyon kis méretű személyi számítógépek (pl. a laptop PC-k) esetén az egész központi egységet a billentyűzet alá építik be.

Az **alaplap** (angolul: *mother board, systems board, main board*) tartalmazza a személyi számítógép működése szempontjából elengedhetetlenül szükséges elektronikus alkotórészeket, pl. a mikroprocesszort, a minimális központi tárat (RAM), az órajel-generátort, a BIOS-ROM tárlót stb. Ezen kívül pl. a grafikus-, a modern- vagy a faxkártyák stb. – összefoglaló néven **bővítőkártyák** (angolul: *extension boards*) – részére csatlakozókat is elhelyeztek rajta.

A PC-k lelke a **mikroprocesszor.** A szóhosszúságuk 8 (már ritka), 16, 32 vagy 64 bit. Ez elsősorban a közvetlenül megcímezhető memóriaterület nagyságát határozza meg.

A **munkatár** (operációs tár) (MOS-chipek formájában) az alaplapon foglal helyet. A tár maximális mérete az alkalmazott processzortól függően jelenleg 4GB.

A hajlékony mágneslemezek és a merevlemezek (a magyar szakzsargonban: floppyk és winchesterek) ún. **külső tárolók.** Napjainkban már > 2GB tárolókapacitású merevlemezes tárolók is vannak forgalomban.

Példa: egy 486-os PC szokásos merevlemezes tárolója 460 MB kapacitású.

A **monitor** (képernyő), az adatmegjelenítés minden PC lényeges tartozéka. A modern gépek színes monitorra vannak felszerelve. Számos feladathoz – pl. a szövegszerkesztéshez – monokróm (fekete-fehér, zöld-fekete, fekete-narancssárga) monitorok is elegendőek. Az elektronikusan vezérelt folyadékkristályos (pl. 2048×1024 képpont felbontású) monitorok egyre inkább felváltják a hagyományos elektronsugaras képernyőket.

A **be-/kimeneti készülékek** párhuzamos vagy soros illesztőegységeken keresztül csatlakoznak a központi egységhez (kivétel: a billentyűzet és a monitor, amelyek közvetlenül az alaplaphoz kapcsolódnak). A személyi számítógépek a hajlékony- és a merevlemez-meghajtók, a nyomtató, az egér és a külső modem csatlakozására további illesztőegységekkel is rendelkeznek. A periferiális készülékek korlátozzák az adatátvitel sebességét. A PC és a nyomtató azonban párhuzanos üzemmódban is használható, ha a nyomtatónak saját adattároló egysége van.

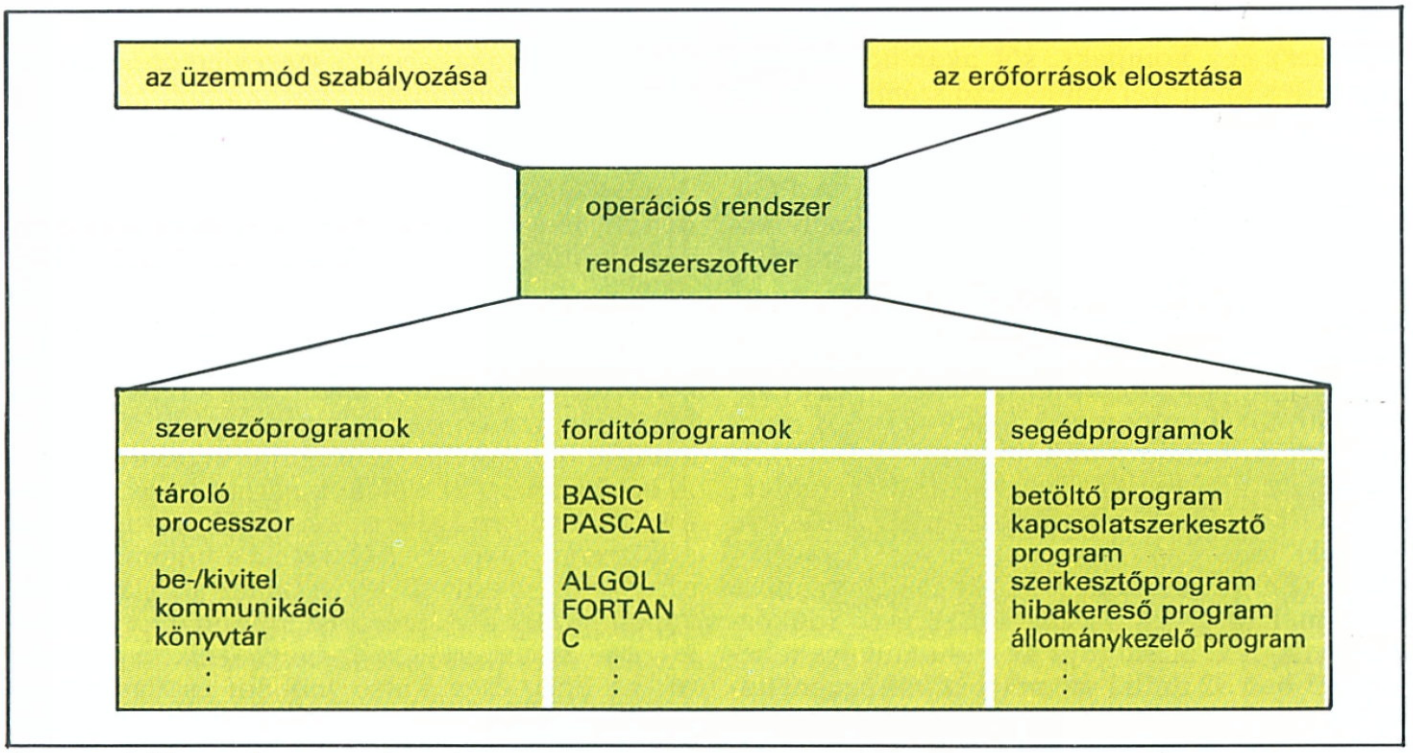
Az **operációs rendszer** bonyolult szoftver (93. o.), amely a PC bekapcsolását követően egy automatikusan elinduló ellenőrző program (teszt) lefutása után a külső tárolóból átkerül a munkatárba. Az operációs rendszer vezérli és ellenőrzi a számítógép működését, közvetít a hadver és a szoftver között, gyakran segédprogramokat és a programnyelvekhez (pl. BASIC és PASCAL) szükséges fordítóprogramokat is tartalmazza. Az alkalmazott operációs rendszer határozza meg, mennyire felhasználóbarát a személyi számítógép.

Az **MS–DOS** (angolul **M***icro***S***oft* **D***isc* **O***perating* **S***ystem*) az IBM-kompatibilis személyi számítógépek legelterjedtebb operációs rendszere. Első változata 1979-ben készült a Seattle Computer Products cégnél. 1981-ben a gyártás joga a Microsoft céghez került, amely az operációs rendszert azóta állandóan aktualizálja. Minden változata a be-/kimeneti utasítások kivételével teljesen független a hardvertől, azaz a programok a különböző típusú számítógépek között (legtöbbször) cserélhetők.

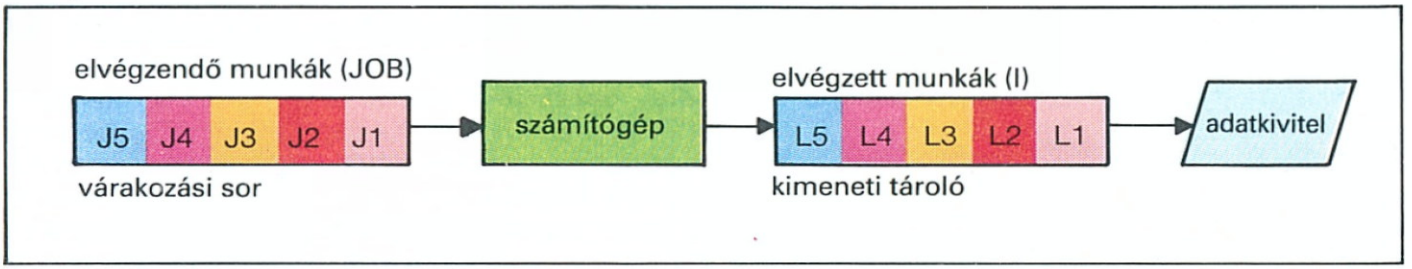
Az MS–DOS operációs rendszer kiegészítése a **Windows** grafikus felhasználói felület (angolul: *graphical user interface*), amelyből időközben egy önálló, többfeladatú párhuzamos végrehajtás lehetőségével rendelkező operációs rendszer fejlődött ki Windows NT néven.

A PC-k felhasználási területei

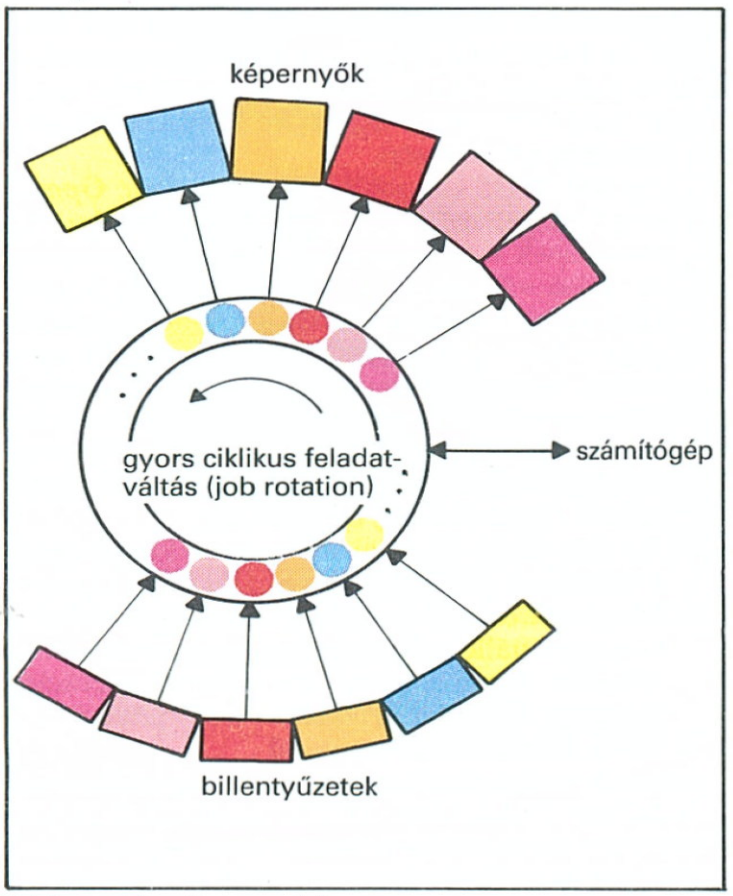
Játékok, szövegszerkesztés, kiadványszerkesztés (*Desktop Publishing,* DTP), adatbázis-kezelés, tervezés, szerkesztés, számítógépes grafika, tudományos és technikai algoritmusok feldolgozása. Hálózattá összekapcsolt személyi számítógépek megfelelő programozással a nagy számítógépek munkáját is átvehetik.



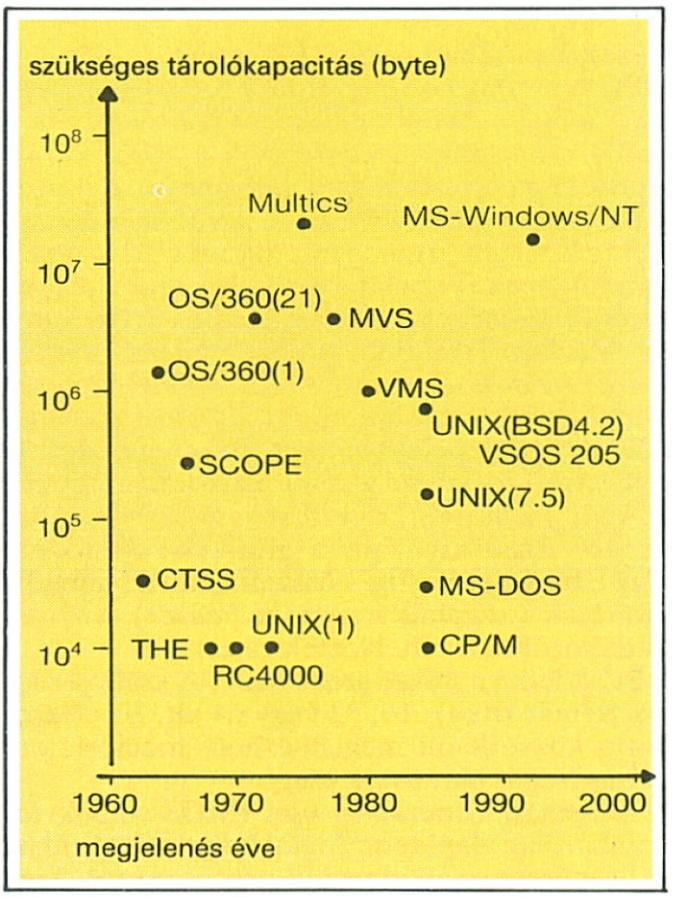
Az operációs rendszer szerkezete



Kötegelt üzemmód



Párbeszédes üzemmód



Különböző operációs rendszerek mérete

Egy számítógép **operációs rendszere** *(operating system)* az üzemelés fenntartásához, a felhasználói programok futtatásához és a számítógép egyes alkotórészei közötti munkamegosztáshoz szükséges összes szoftvert magában foglalja. Az operációs rendszer azoknak az állandóan installált programoknak az összessége, amelyeknek segítségével a számítógép az elvégzendő feladatokat a lehető legkedvezőbb módon oldja meg. A számítógép kikapcsolt állapotában az operációs rendszer az állandó tárban található, bekapcsoláskor onnan töltődik a munkatárba.

Kezdeti betöltőprogramnak nevezzük a ROM munkatárban állandóan jelenlevő, nagyon rövid programot, amely a számítógép bekapcsolása után az operációs rendszert a munkatárba tölti. Ez a folyamat a rendszertöltés (angolul: *bootstrapping* vagy *booting* – az elnevezés Münchhausen báróra utal).

Az operációs rendszer határozza meg azt is, hogy a számítógép milyen sorrendben hajtja végre a felhasználói program utasításait, annak adatait hol és hogyan tárolja, a végrehajtás során melyik tár adataira van szükség, hogyan lehet megakadályozni azt, hogy a számítógéphez illetéktelen felhasználók férjenek hozzá stb. Mindez – valamint a nyilvántartás is – az operációs rendszer feladata.

Az operációs rendszert a számítógépgyártó cégek telepítik, mégpedig állandó aktualizálás után a legújabb változatot. A változtatás legfontosabb szempontjai a felhasználóbarát működés és a számítógép valamennyi alkotórészének optimális kihasználása.

Példák operációs rendszerekre:

Sok cég alkalmazza az *UNIX* különböző változatait amelyek a problémaorientált C nyelven íródtak (170. o.). Ez az első olyan operációs rendszer, amely nem kompatibilis számítógép-rendszerekre is átvihető. A nagy számítógépeknek kb. 25%-a, a személyi számítógépeknek viszont csak elenyésző része használt UNIX operációs rendszert.

Az IBM-kompatibilis PC operációs rendszere az *MS-DOS* és az *OS/2,* amelyet időközben részben a *Windows NT* váltott fel.

A **megszakítás** *(interrupt)* az operációs rendszerek fontos része. Ha valamilyen esemény a program menetét megszakítja, egy ellenőrző program lép működésbe, amely eldönti, hogy a megszakított program ismét folytatódjon, közbenső tárolásra kerüljön vagy megszakadjon-e. A megszakítást az idézheti elő, ha pl. programozási hiba vagy nullával való osztás fordul elő, ha a megengedett számítási idő lejár, ha a perifériális készülék adatokra vár, ha magasabb prioritású feladat érkezik stb.

Üzemmód

A programok kivitelezésének módja szerint a következő üzemmódokat különböztetjük meg:

Kötegelt üzemmód *(batch mode)*. A számítógép az egyes programokat a bemeneti adatokkal együtt várakozási sorba rendezi és ennek megfelelően hajtja végre. A felhasználó a program leadása után az események menetét nem tudja befolyásolni. A 60-as évek közepéig túlnyomó részben ezt az üzemmódot alkalmazták. Manapság elsősorban éjjel használják a nagyon számításigényes programok vagy az állandó jellegű feladatok, pl. a fizetéselszámolások elvégzésére.

Párbeszédes üzemmód *(interactive mode)*. A feldolgozás interaktív, azaz párbeszédek sorozatán keresztül történik. Ez azt jelenti, hogy a felhasználó a megoldandó feladatot vagy annak részeit a számítógépbe juttatja, a számítógép visszajelzéseire – pl. a hibajelzésekre, vagy arra, hogy további adatokra van szükség, vagy lejárt a programfuttatásra szánt idő – azonnal reagálhat is.

Példa: szövegszerkesztés.

Többszörös programozás, multiprogramozás *(multi-programming, multi-tasking)*. A számítógép periodikusan egy-egy rövid időtartamra valamennyi programmal foglalkozik. Ez az egyidejűség benyomását kelti. Mivel a számítási idő több felhasználó között oszlik meg, ezt az üzemmódot időosztásos (angolul: *time sharing*) üzemmódnak is nevezzük. A *multi-tasking* esetén tulajdonképpen prioritás szerinti vezérlés folyik, tehát előre meghatározott elsőbbségi sorrend határozza meg, hogy melyik program mikor és mennyi ideig kerül feldolgozásra.

Példa: Mialatt a számítógép az A program adataira vár, a B programot a háttértárolóból a munkatárba töltheti, a C programot beolvashatja, a D program eredményeit pedig a kiviteli adatcsatornába továbbíthatja. Ha az E program magasabb prioritással rendelkezik, akkor az összes többi program futása leáll, státuszuknak megfelelően egy közbenső tárolóba kerülnek, és a számítógép az E programot hajtja végre.

Ez a működési mód biztosítja a számítógép valamennyi alkotórészének optimális kihasználását.

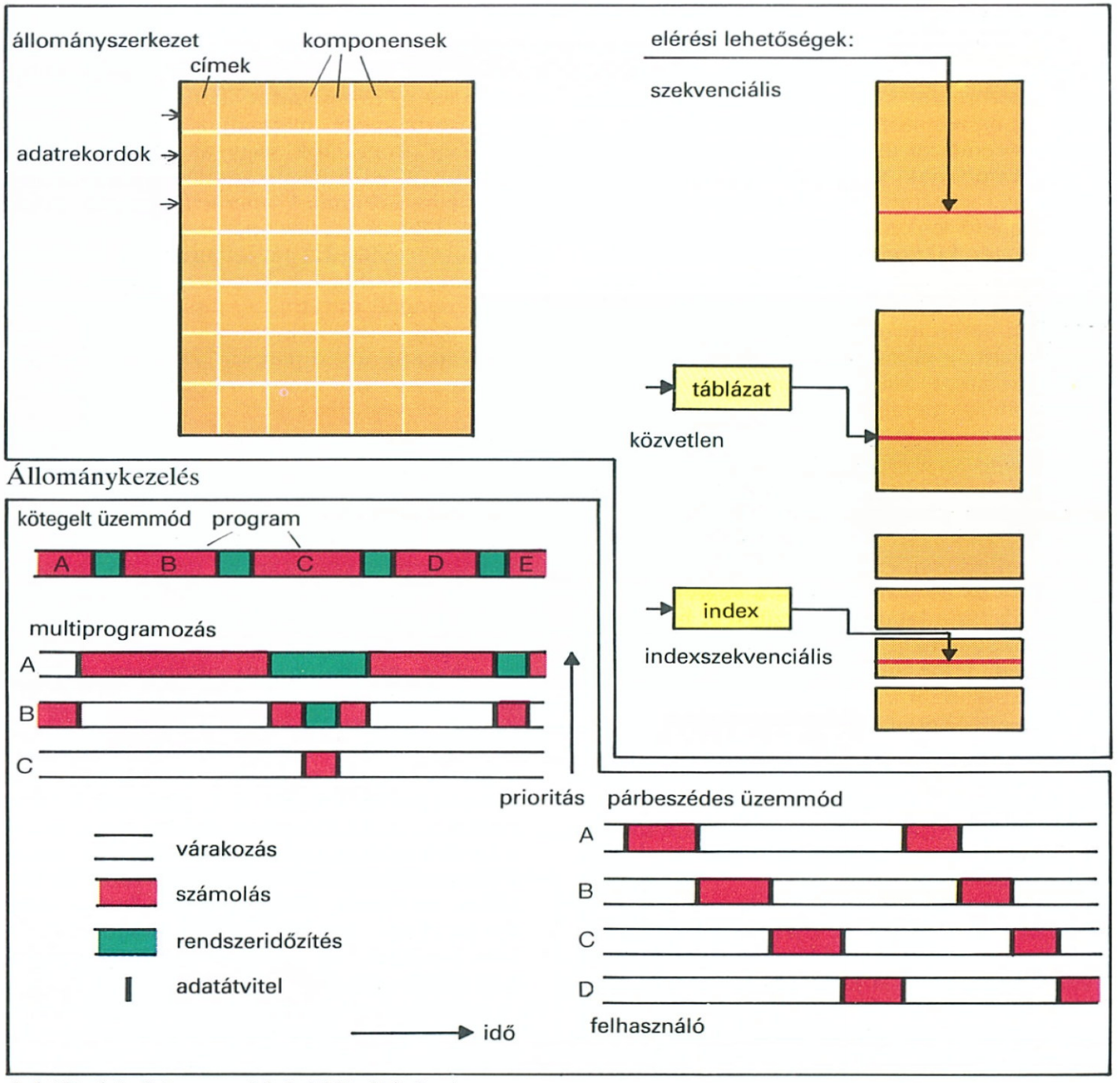
Időosztásos üzemmód. Minden felhasználó egymástól függetlenül használja a számítógépet egyedi programok egyidejű feldolgozására.

On line üzemmód. Több felhasználó hívhatja egyidejűleg ugyanazt a számítógépben állandóan jelenlevő programot.

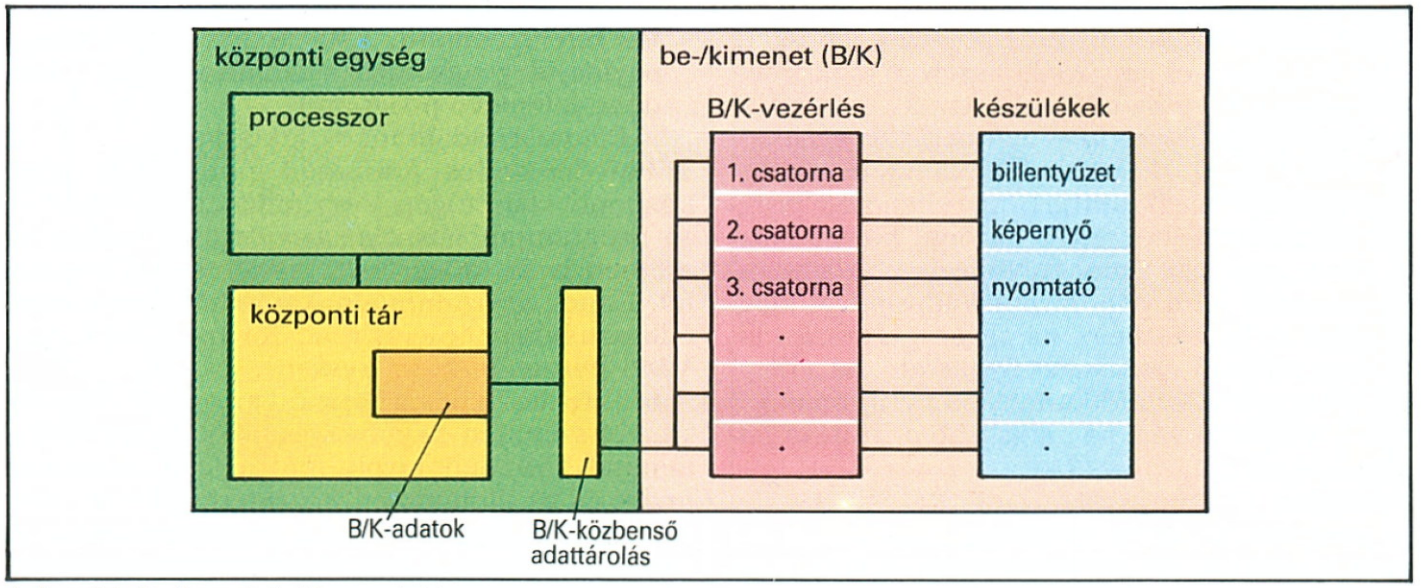
Példa: az adatbankok lekérdezése.

Többfelhasználós üzemmód *(multi-user)*. Egy központi számítógép több felhasználó különböző programjait látszólag egyidejűleg (ebben az esetben is az időosztásos módszerrel) dolgozza fel. Ennek az üzemmódnak alapfeltétele a többfelhasználós operációs rendszer, pl. UNIX vagy VMS *(virtual memory system)*.

A többfelhasználós üzemmód az operációs rendszerrel szemben nagyon nagy követelményeket támaszt. Minden tárolt információt állandóan őrizni és az illetéktelen hozzáféréstől óvni kell. A pillanatnyilag nem aktív programrészeket és adatokat azonnal alacsonyabb prioritású tárolókba kell helyezni. Hiba fellépése esetén az érintett programot közbenső tárolóba kell átvinni és javítás után ismét a munkatárba kell tölteni.



A különböző üzemmódok időbeli lefutása



A be-/kiviteli eszközök vezérlése

Munkavezérlő nyelv

Ezen a szintaktikai szempontból egyszerű nyelven (angolul: *command* vagy *job control language*) történik a felhasználó és az operációs rendszer közötti (pl. a felhasználó nevére, a felhasználni kívánt program megjelölésére, a hibajelzésekre, az adatállomány-nyilvántartás adataira stb. vonatkozó) információcsere.

A munkavezérlő nyelv elemei gyakran különleges karakterek – pl. $ vagy /* – között foglalnak helyet, hogy a valódi programrészektől meg lehessen őket különböztetni.

Példák: a munkavezérlő nyelv utasításaira.

Kötegelt üzemmód esetén:

$ JOB (2210) $	a 2210. számú feladat
$ NOHAWA $	felhasználói azonosító (kódszó)
$ 2 $	a központi feldolgozóegység használatának előrelátható ideje (s)
$ PASCAL $	az alkalmazni kívánt fordítóprogram
$ ITERATION $	a program neve
$ ABX1382 $	a szükséges adatállomány
$ GO $	programindítás

Párbeszédes üzemmód esetén a számítógép egymás után kérdéseket ír ki és várja a választ.

$ JOB NO?
$ NAME?
$ TIME?
$ COMPILER?
$ PROGRAM?
$ DATA?
$ GO?

Ha a felhasználó valamilyen saját adatot nem ad meg, akkor az operációs rendszer egy előre meghatározott standard (angolul: *default*) értéket vesz alapul. Ilyen értékek pl. az előre lefoglalt számítógépidő, be-/kimeneti egységek, könyvelési szám, a külső tároló azonosítója stb.

A be-/kivitel (B/K) vezérlése

Ez a feladat (angolul: *Input/Output, I/0 control*) az operációs rendszer fontos részét alkotja. Mivel a számítógép központi egysége sokkal gyorsabb, mint a be-/kiviteli eszközök, így szinte munkanélkülivé válhatna. A be-/kiviteli szabályozás biztosítja az optimális kihasználtságot, pl. úgy, hogy a bemeneti adatokat egy *puffernek* nevezett közbenső tárolóban átmenetileg tárolja, majd az összegyűlt adatmennyiséget a processzorhoz továbbítja. Ezt a folyamatot segédprogramok vezérlik, és arról is gondoskodnak, hogy az adatcsatornák párhuzamosan és a központi egységtől messzemenően függetlenül dolgozzanak.

A be-/kimenet vezérlése üzenetet küld a processzornak, hogy a használni kívánt készülék üzemkész állapotban van-e (állapotjel, státusjel).

Háttérüzemmódnak (angolul: *spooling: **s**imultaneous **p**eripheral **o**perations **o**n **l**ine*) nevezzük a be-/kiviteli adatok közbenső tárolóban (puffer állományban) történő összegyűjtését, ha különböző, egyidejűleg rendelkezésre álló adatblokkok ugyanazt a periferiális egységet használják.

Adatállományok kezelése

Az **adatállományok** (angolul: *files*) tartalmilag összetartozó adatok együttesei, amelyek közös név alatt kerülnek tárolásra.

Az adatállományok **rekordokból** (angolul: *records*) állnak, amelyek egy logikai be-/kiviteli utasítással az állományból kiolvashatók ill. az állományba írhatók. A rekordok általában több **mezőre** tagolhatók.

Az adatállományok általában külső tárolón helyezkednek el. Az adatállomány-kezelés (angolul: *file management*) feladata:

1. A felhasználó által kívánt állomány megkeresése és a munkatárba történő betöltése.
2. Az új adatok számára szabad tárterület foglalása és fenntartása.
3. A számítógépben található adatállományokról és rekordokról felhasználóbarát katalógus készítése és naprakész állapotban tartása.

A **tartalomjegyzék** vagy **könyvtár** (angolul: *directory*) arra szolgál, hogy az állományok áttekinthetően legyenek elrendezve. Az összetartozó állományok ugyanabban a könyvtárban találhatók. A gyökérkönyvtár (angolul: *root directory*) a legfelsőbb szinten helyezkedik el, és nem törölhető. Az állományok neve mellett az elfoglalt tárolóterületet, valamint az állományok elkészítésének dátumát és időpontját is tartalmazza.

Programkönyvtár

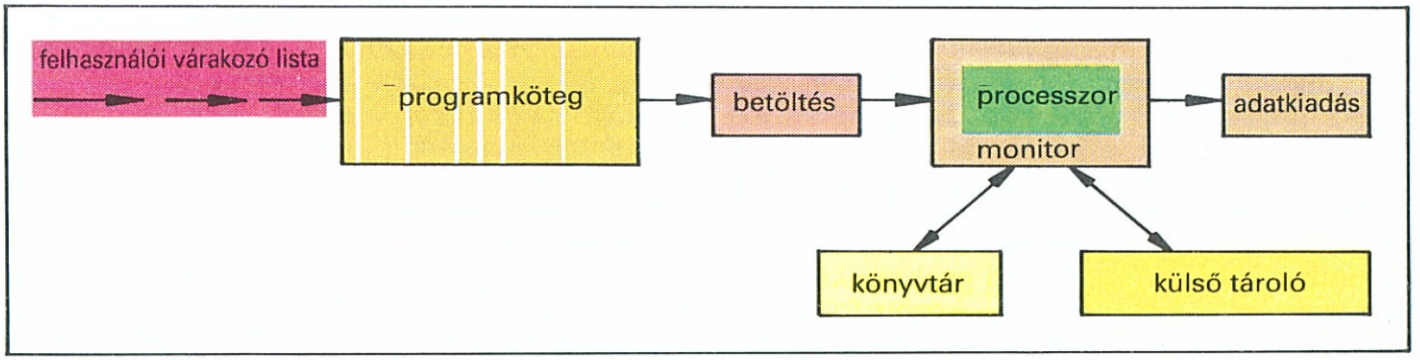
A programkönyvtár a számítógépben (vagy a számítógépes hálózatban) megtalálható valamennyi közvetlen elérésű programot, programrészt vagy eljárást tartalmazza.

A **segédprogramok** (angolul: *utilities*) a rendszerprogramokhoz tartoznak, tehát az operációs rendszer részei, annak működését szolgálják, az operációs rendszer szervezése alatt állnak és onnan hívhatók. Ide tartoznak az állománykezelő és adatrendező programok, a hajlékony lemezek formálására és másolására használatos programok, az adattörlő és adatmásoló programok, a makrodefiníciók előállítására szolgáló programok, valamint a szerkesztési segédprogramok stb.

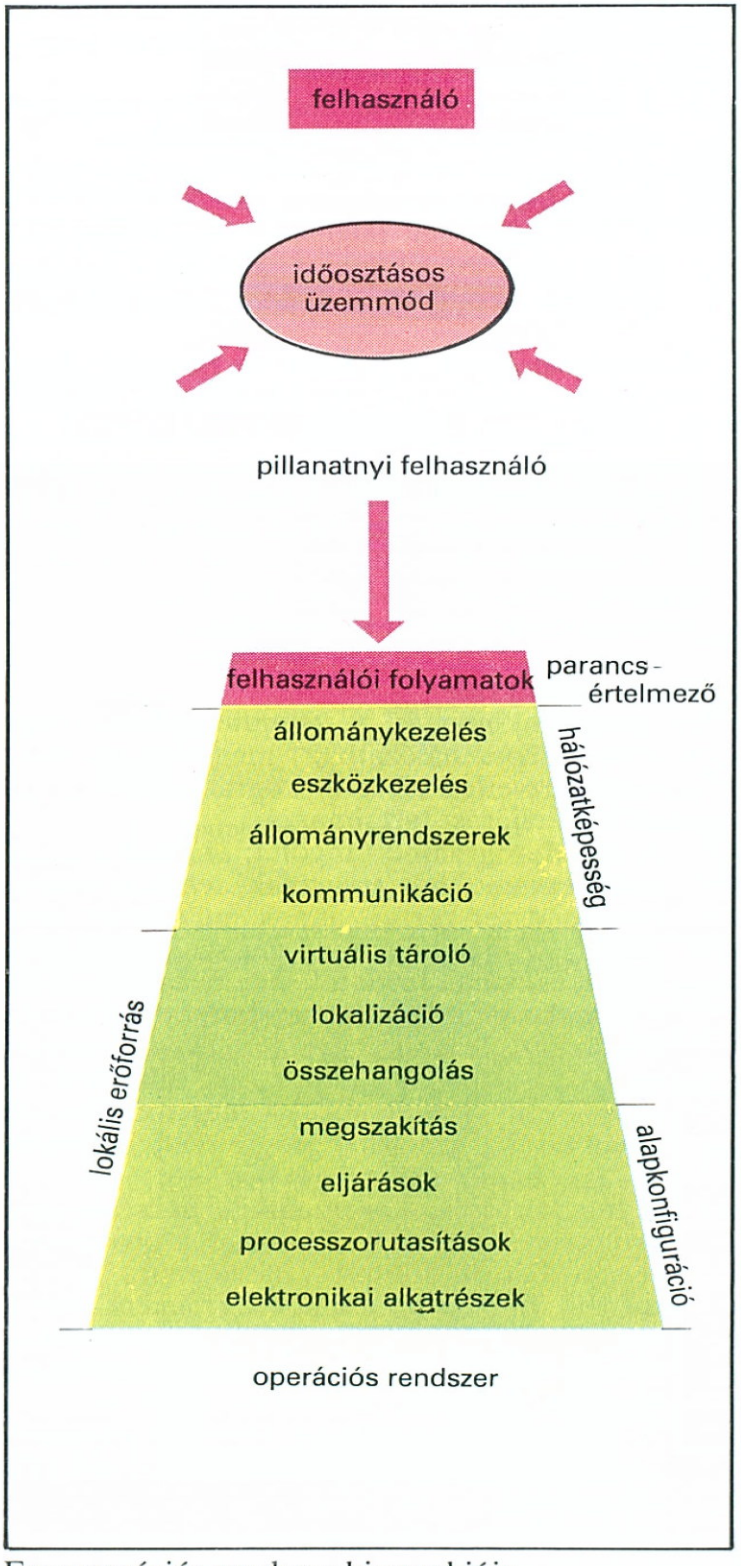
A tuladonképpeni **programkönyvtár** (angolul: *program library*) a felhasználói programok (elsősorban lefordított változatának) gyűjteménye. Ezek a programok egyenként hívhatók és a felhasználói programokba beépíthetők. A könyvtárak a programozásban fontos szerepet játszanak.

A legtöbb számítógépnek kiterjedt programkönyvtára van, ami a felhasználóknak sok időt takarít meg.

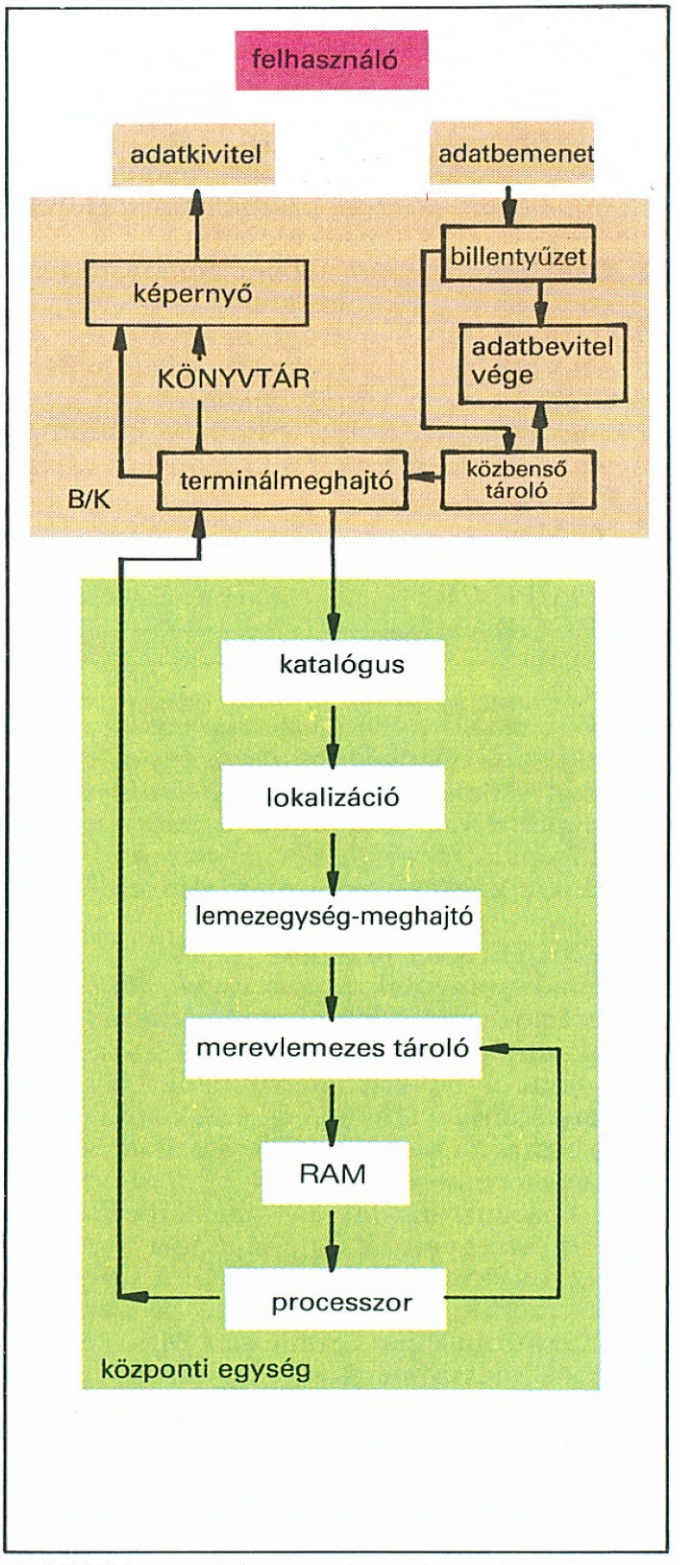
A **könyvelőprogramok** az operációs rendszer részei, és azt rögzítik, hogy melyik program menynyi ideig veszi igénybe a számítógépet és annak komponenseit.



Kötegelt feldolgozás: egy egyszerű operációs rendszer



Egy operációs rendszer hierarchiája



A UNIX-operációs rendszer keresi és megtalálja a gyökérkönyvtárat

Valamennyi operációs rendszer hierarchikus felépítésű. A tapasztalat szerint az ennyire nagymértékben összetett rendszereket csak ilyen módon lehet kezelni. Az operációs rendszerek egymásra épülő, feladataik szerint jól elkülönülő szintekből állnak. Minden egyes szint csak az alsóbb szintek műveleteihez férhet hozzá, de eközben a művelet részletei a felsőbb szint számára rejtve maradnak.

Minden szint a szoftver és a hardver pontosan meghatározott részeivel foglalkozik. A legalsó szint az elektronikus kapcsoló-áramköröket kezeli, csak a legfelső szint fogadja a felhasználó utasításait.

Az ilyen felépítés lehetővé teszi az operációs rendszer szintről szintre alulról felfelé haladó telepítését (vagy aktualizálását).

A fentiekhez járulékosan kapcsolódó szervezési elv a be-/kiviteli eszközök kölcsönös és teljes függetlensége.

Az 50-es évek számítógépeiben a nagyon egyszerű operációs rendszerek gépi kódban írt programjai még csak néhány száz utasítást tartalmaztak. Az ezekhez szükséges tárolóterület megfelelően csekély volt. A nagy számítógépek modern operációs rendszerei – amelyek megírásához legtöbbször a C nyelvet használják – néhány Mbyte tárolóterületet foglalnak el. A legtöbb PC operációs rendszere kevesebb, mint 100 kbyte területet igényel.

Az alábbiakban egy jellegzetes – az 1973-ban kifejlesztett UNIX-hoz hasonló – operációs rendszer 12 szintjét mutatjuk be.

A legalsó szintet az **elektronikai alapkapcsolások,** pl. kapuáramkörök, tárolóelemek, adat- és címbuszok jelentik. A tárolócímek kiolvasása, a regiszterek törlése, a tárolóelemek tartalmának invertálása és komplementálása ezen a szinten történik.

A **processzorutasítások** (a mikroprogramok kiértékelései) regiszterek csoportjaira vonatkoznak, összeadnak és kivonnak, közbenső értékeket tárolnak, előjelet váltanak, és adatmezőkön végeznek műveleteket.

Az **alprogramok** vagy **eljárások** olyan független programrészeket dolgoznak fel, amelyeket a főprogram hív meg. Ez a réteg az eredményeket a hívó rutinnak adja át.

A **folyamatok lefutásának megszakítását** hibajelző rutinok okozzák, pl. a regiszterek túlcsordulása, programhiba, vagy az előírt idő túllépése esetén. Rendellenes befejezés (abortálás) során a megszakított program lényeges paraméterei eltárolódnak, és mihelyt a hiba elhárítása megtörténik, a program újraindításakor ismét betöltésre kerülnek. Közben más programok futása zajlik.

Az **összehangolás (koordinálás)** a különböző feladatok egyszerű folyamatokká történő összefogását jelenti. Lehetővé teszi az ilyen folyamatok megszakítását és újraindítását.

Megszakítás esetén valamennyi regiszter tartalmát egy speciális adatszerkezet, az **állapotszó** veszi át. Újraindításkor egy utasítás az állapotszó tartalmát ismét a processzorba juttatja.

Az összehangolás szintjének lényeges része párhuzamos folyamatok szinkronizálása. Ha egy programrésznek egy másik programrész adataira van szüksége a további számoláshoz, akkor az összehangolás szintjén az első programrész számítási folyamatai mindaddig megállnak, amíg a megfelelő párhuzamos számítás eredménye meg nem születik. Az ennek megfelelő szervezés a szemafor-elvet követi.

Szemafor. Ha a program futása olyan helyre ér, amelyen a programnak egy párhuzamosan futó másik operáció eredményére van szüksége, akkor a szemafor az eredmény elérhetőségétől függően *várakozás* ill. *tovább* utasítást ad. Egy szemafor egyidejűleg több folyamatot is vezérelhet.

A **helymeghatározó tárolóhoz** való hozzáféréssel egy adatblokk olvasását és írását vagy egy író-/olvasófej helyzetét lehet kezelni. Itt részfolyamatokról, pl. egy merevlemezes tár szektorainak és sávjainak kezeléséről van szó.

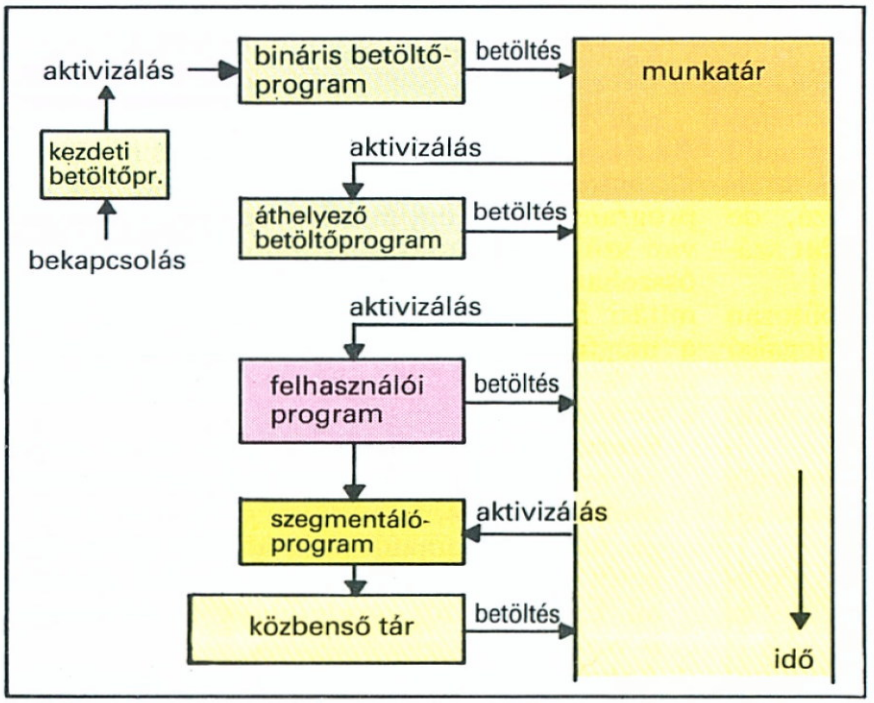
A **tárolókezelés** a számítógép külső és belső tárolóiból *virtuális tárolót* hoz létre. A program virtuális tárolócímeket ad meg. A felhasználónak úgy tűnik, mintha egy szinte tetszőlegesen nagy tároló állna a rendelkezésére. Az operációs rendszer a **számítógép és a perifériák közötti illesztőegység** szintjétől fölfelé tartalmazza a számítógép perifériáinak kezelését és a más számítógépekkel történő kommunikációt is. Az adatforgalom speciális adatvezetékeken (angolul: *pipes*) keresztül zajlik. Az A folyamat adatkivitele ekkor az adatvezetéken keresztül a B folyamat számára adatbemenetet jelent.

Az **állományrendszerek** az állományokat a hardvertől függelenül kezelik. Egy állomány pl. részenként több tárolóhelyen elosztva foglalhat helyet.

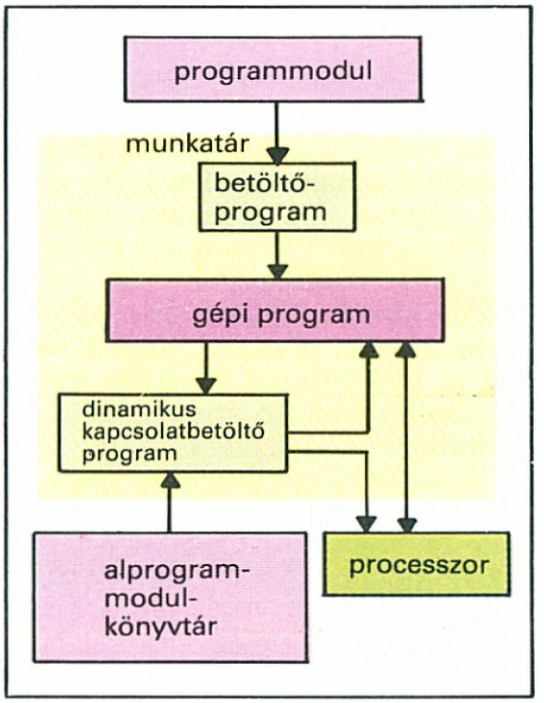
Az **eszközkezelés** feladata az, hogy a számítógép folyamatainak a periferiális eszközök iránti igényét kielégítse. Ezen a szinten a képernyő, a nyomtató, a rajzgép, a billentyűzet és az egér mint külső egységek létrehozása, törlése, megnyitása, lezárása, írása és olvasása történik.

Az **állománykezelő** (angolul: *file manager*) olyan program, amely pl. az állományok törlését, másolását, tervezését stb. végzi. A tartalomjegyzék külső állományneveihez hozzárendeli a belső állományneveket. Az állománykezelő névtáblázata lehetővé teszi az állományokhoz, a készülékekhez és a kommunikációs csatornákhoz való hozzáférést. Ezen a szinten történik a hozzáférés szabályozása is.

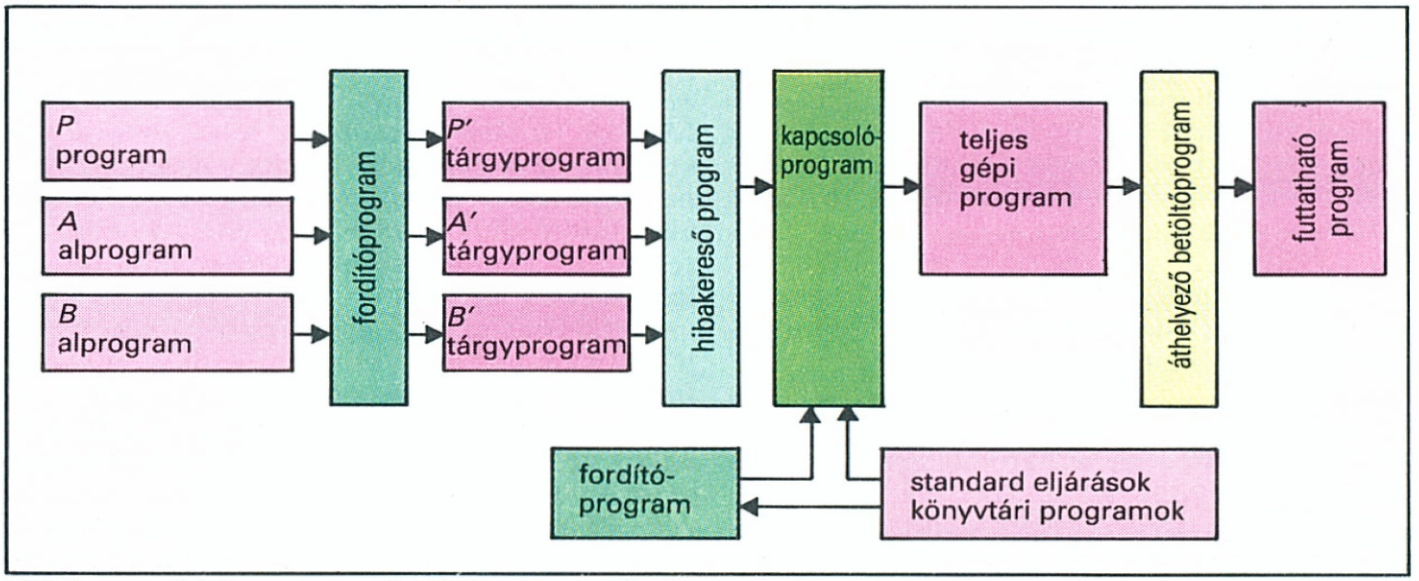
A **felhasználói folyamatok** teljes programok lefutását szabályozzák, és végrehajtják a felhasználó utasításait. Ezt az utolsó szintet a számítógép **burkának** (angolul: *shell*) is nevezik. A felhasználó csak ezen a szinten avatkozhat be a számítógép folyamataiba.



A bekapcsolásnál fellépő betöltési folyamatok



Kapcsolatszerkesztés és betöltés



Néhány segédprogram pozíciója a programfutás során

Hibajelzések a képernyőn:

rövid hibajelzés

```
LINE 205 ERROR 15
```

részletes hibajelzés

```
LINE 205 ERROR 15 SUBSCRIPT MISSING
```

automatikus hibajavítás és alapértelmezett kijelzése

```
LINE 205 ERROR 15 SUBSCRIPT 2 OMITTED TAKEN AS 1
```

A hibakereső program néhány lehetséges üzenete:

```
BAD FILE NAME
DEVICE I/0 ERROR
DISK FULL
DIVISION BY ZERO
FIELD OVERFLOW
FOR WITHOUT NEXT
MISSING OPERAND
OUT OF PAPER
OVERFLOW
RETURN WITHOUT GOSUB
STRING TOO LONG
TYPE MISMATCH
UNDEFINED LINE NUMBER
UNPRINTABLE ERROR
```

A hibakereső program hibát jelez a vizsgált programban

Az operációs rendszer számos segédprogramot tartalmaz, amelyek a számítógép kikapcsolt állapotában külső, pl. mágneslemezes tárakban találhatók, és bekapcsoláskor szükség esetén onnan töltődnek a munkatárba. Ezeknek a programoknak legfontosabb feladatai közé tartozik az operációs rendszer betöltése, a hardver és a szoftver ellenőrzése, valamint a moduloknak a programokba történő beágyazása.

A kisebb számítógépek operációs rendszerét gyakran teljes egészében a csak olvasható ROM tárban helyezik el.

Betöltőprogram

A **betöltőprogram** (angolul: *loader*) olyan segédprogram – tehát szoftver –, amely a bináris alakba átalakított programokat egy háttértárolóból a munkatárba tölti. Ez elsősorban a számítógép bekapcsolása után fontos, mert az operációs rendszer is betöltőprogram segítségével kerül át a munkatárba. Ezen kívül a betöltőprogram végzi a perifériális készülékek megfelelő címzését, ellenőrzi ezek állapotát, valamint megállapítja, hogy a munkatárban elegendő tárolóterület áll-e rendelkezésre.

A **bináris betöltőprogramok** az abszolút címzéseket tartalmazó programokat ismételten ugyanarra a tárolóterületre töltik. Az abszolút címzés azt jelenti, hogy az ilyen programokban található címek megegyeznek a munkatárnak a megfelelő programrészlethez rendelt közvetlen címével. Az abszolút címzéseket tartalmazó programok általában kisebb segédprogramok, pl. az áthelyező és a kapcsolatszerkesztő program, a szegmentálóprogram stb.

Az **áthelyező betöltőprogram** (angolul: *relocating loader*) az egyes programokat mindig szabad, de minden alkalommal más szabad munkatároló területre tölti, és a programban előforduló relatív címeket a betöltésnek megfelelően a munkatár abszolút címeire módosítja. Ennek megkönnyítésére a programok relatív címei minden program elején külön táblázatban rendezve találhatók meg.

A bináris betöltőprogram tölti be az áthelyező betöltőprogramot, amely azután az operációs rendszer többi programját és a felhasználói programokat tölti be.

A **kezdeti betöltőprogram, IPL** (angolul: *Initial Program Loader*) permanens ROM tárolóchipből automatikusan töltődik a munkatárba. A kezdeti betöltőprogram néhány olyan egyszerű utasításból áll, melyek a bináris betöltőprogramot a számítógép bekapcsolása után azonnal a munkatárba töltik (*bootstrapping*, 93. o.).

Szegmentálás. Ha a betöltőprogram egy hosszú programot helyhiány miatt nem tud egyszerre a munkatárba vinni, akkor a programot több részre, *szegmensre* osztja, és a program futtatása során mindig a szükséges szegmenst másolja a munkatárba. A legközelebb szükséges szegmenst az előzőre másolja, így átfedést (angolul: *overlay*) hoz létre, a továbbra is szükséges, mindkét szegmens által használt adatokat viszont megtartja. A fordítóprogramok általában szegmentáltak. Néhány operációs rendszer, pl. az MS–DOS gyakran szegmentálja a programokat.

Kapcsolatszerkesztő program

A **kapcsolatszerkesztő** vagy **kapcsolóprogram** (angolul: *linkage editor* vagy *linker*) a számítógépben gépi nyelven már megtalálható modulokat (önmagukban egységes programrészleteket, pl. standard függvényeket, eljárásokat, könyvtári programokat) a fordítóprogram által megjelölt helyeken beépíti a lefordított programokba. A kapcsolóprogram tehát az egyes gépi nyelvre fordított programrészeket egyetlen futtatható programmá kapcsolja össze.

Az elnevezésekből adódó bonyodalmakat kapcsolóprogram az azonosítók rendszeres átnevezésével oldja meg.

A **kapcsolatbetöltő program** (angolul: *linking loader*) a kapcsoló és a betöltőprogram kombinációja. Hatására a programok az összekapcsolás után nem közbenső tárolóba, hanem rögtön a munkatárba töltődnek és kivitelezésre kerülnek.

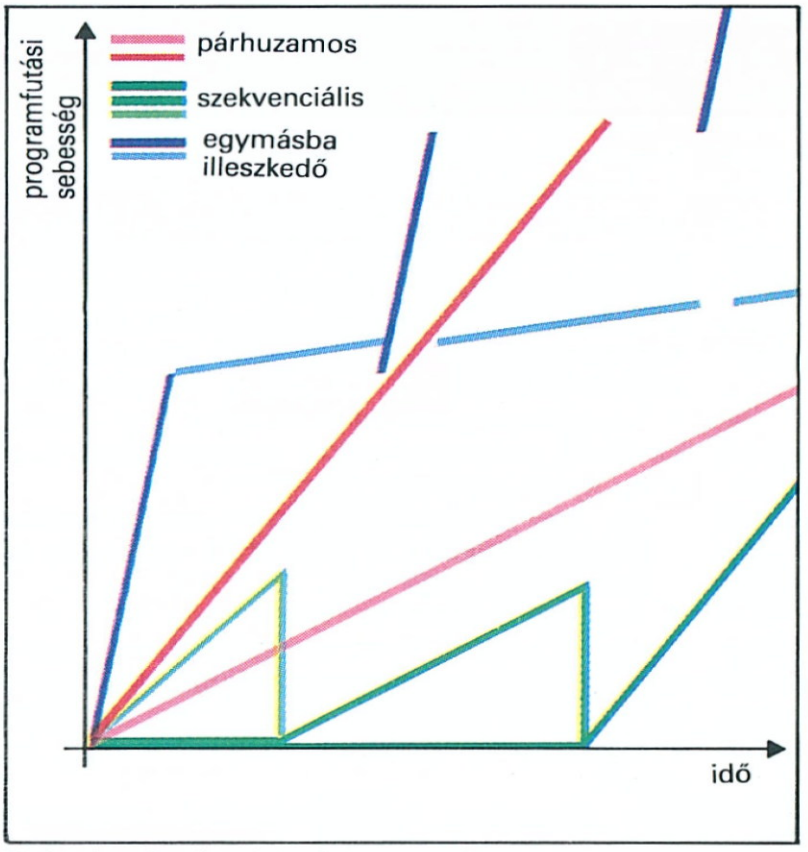
A **dinamikus kapcsolatbetöltő programok** a programot betöltik és elindítják, a modulokat csak akkor keresik meg és kapcsolják össze, amikor a programban sorrakerülnek. Előnyük: mivel a munkatárban csak a pillanatnyilag szükséges programok találhatók, tárolóterületet takarítanak meg. Hátrányuk: a körülményesebb programozás és a valamivel hosszabb kivitelezési idő.

Hibakereső program

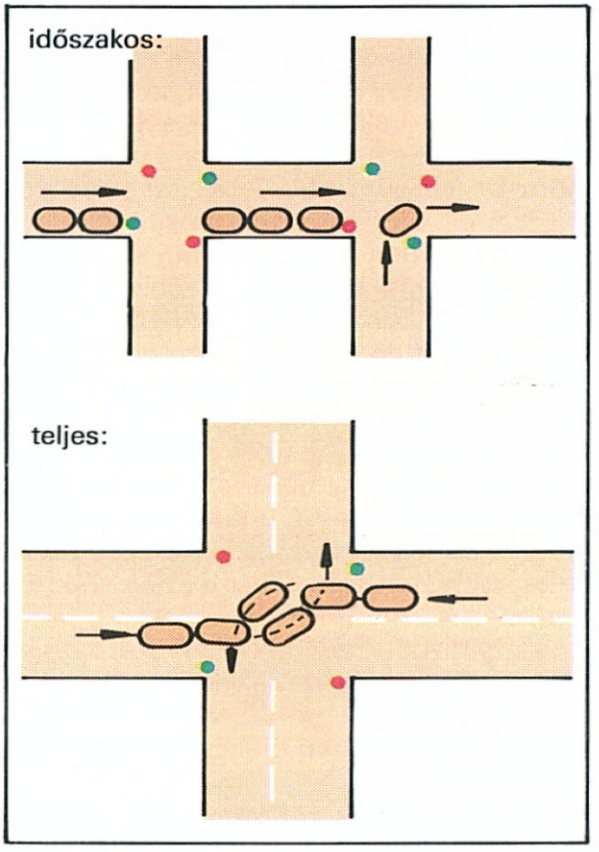
A **hibakereső program** (angolul: *to debug*) minden programnyelven írt programot lépésről lépésre megvizsgál, hogy nem tartalmaz-e formai hibát. Ezeket a felhasználónak jelzi, és helyenként még a helyesbítésre is javaslatot tesz. Ha a lefordított program már fut, akkor a hibakereső program további hibákat keres, pl. hiányzó indexeket, helytelenül formázott adatokat, alulcsordulást, túlságosan kicsi számmal történő osztást stb. A felhasználó a hibakereső program segítségével leállíthatja a program végrehajtását, és tájékozódhat a változók, regiszterek és tárolók állapotáról.

A **hardver-hibakeresők** a számítógép bekapcsolása után azonnal ellenőrzik az elektronikus alkotóelemek hibátlan működését. Nagyobb számítógépek esetén a processzor végzi ezt a feladatot. A személyi számítógépekben külön **szoftver-hibakeresők** ellenőrzik az alkotóelemek működését.

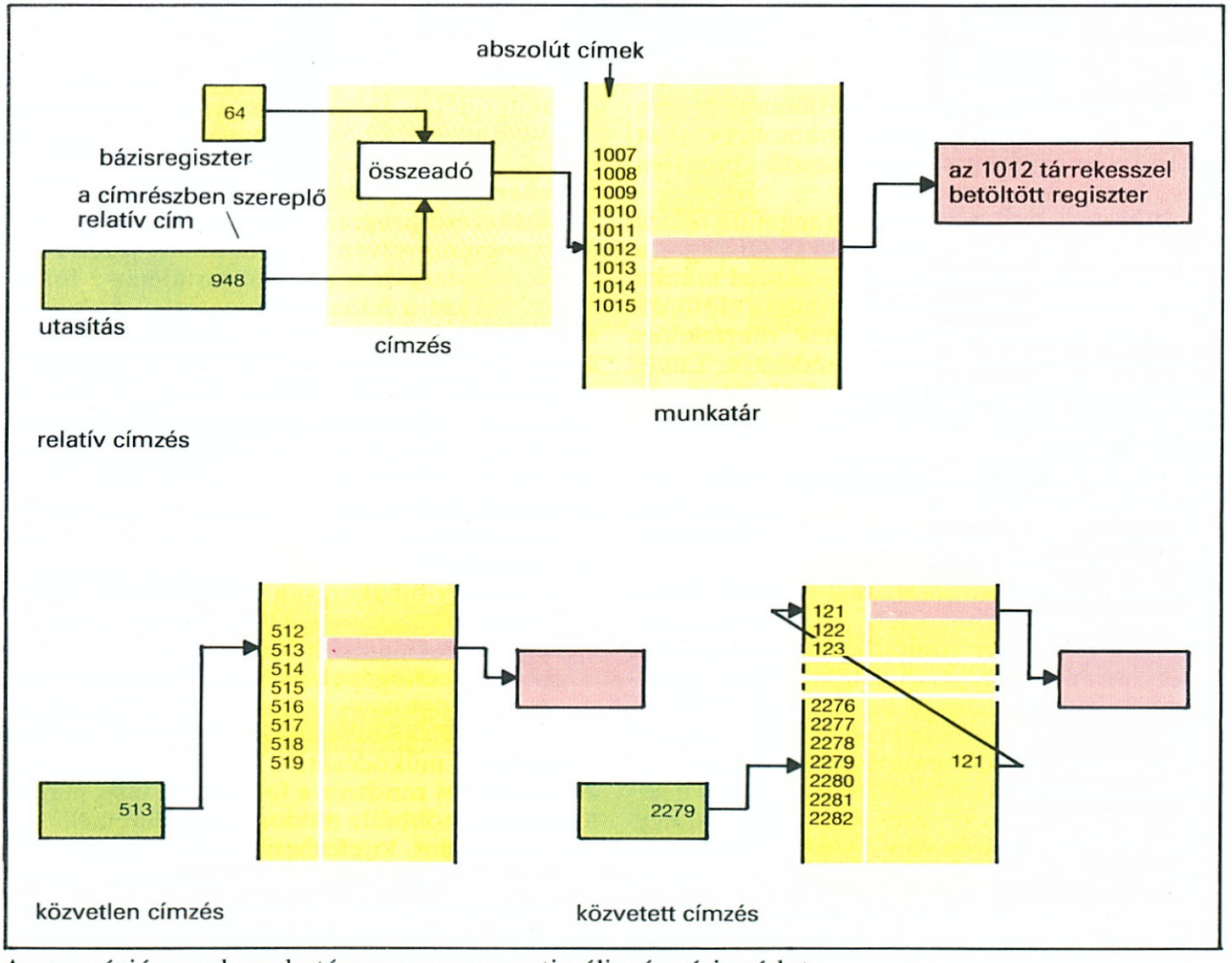
Az operációs rendszer a felfedezett hiba jellegétől függően többféle módon reagálhat: leállíthatja a programot, kijelezheti a hibát, javaslatot tehet az első megtalált hiba kijavítására, a program lefordítása után kijelezheti az összes megtalált hibát (valószínűségszámítási megfontolások alapján), automatikus korrekciót végezhet, ezt a program megszakítás nélkül ki is jelezheti és végül meg is szakíthatja a programot.



Különböző szimultán folyamatok



Elakadás a közúti forgalomban



Az operációs rendszer határozza meg az optimális címzési módot

Az operációs rendszer fontos feladata, hogy a számítógép egyes összetevőinek kihasználtsági fokát optimalizálja.

Szimultán folyamatok optimalizálása

A számítógépben két vagy több folyamatot pl. programot **egyidejűnek** vagy **szimultánnak** (angolul: *concurrent*) nevezünk, ha egymástól függetlenül, szekvenciálisan, párhuzamosan vagy egymásba fonódva lefuttathatók. Azonban mindössze néhány valóban szimultán folyamat létezik, mert a számítógép komponensei ésszerűségi okokból mindig több folyamatot szolgálnak ki. Ha két folyamat egyidejűleg ugyanahhoz a komponenshez próbál hozzáférni, akkor konfliktus lép fel.

Holtpont, végleges kizárás (angolul: *deadlock*) akkor fordul elő, ha két folyamat egymást kölcsönösen akadályozza (blokkolja).

Példa: Az I folyamat éppen hozzáfér a képernyőhöz, és megkísérli, hogy egy azt követő adatkivitelhez a nyomtatót lefoglalja. Azonban a II. folyamat ezalatt éppen adatokat nyomtat a nyomtatón, és a nyomtatót csak akkor tudja szabaddá tenni, ha hozzáfér a képernyőhöz. Ennek eredményeként az I. és II. folyamat kölcsönösen akadályozzák egymást. A megoldási lehetőség: mindkét folyamat már kezdetben megigényeli a várhatóan szükséges perifériákat. Ilyenkor azonban elkerülhetetlenül várakozási idők lépnek fel.

A konkuráló szimultán folyamatok operációs rendszer által történő **szikronizálása** az összes komponens optimális kihasználását eredményezi, ekkor az egy időben igényelt erőforrások konfliktus esetén szekvenciálisan kerülnek felhasználásra.

Minden komponenshez tartozik egy lekérdezhető változó, ennek értéke 0 vagy 1 lehet, amelynek jelentése „szabad" ill. „foglalt". Hozzáférés kérésekor először ez a változó kerül kiértékelésre. Ha értéke „szabad", akkor a folyamat a számítógép megfelelő komponensét lefoglalja, és a változó „foglalt" értéket vesz fel. A használat befejezése után a távozó folyamat a változó értékét ismét „szabadra" állítja.

Minden folyamatban vannak *kritikus szakaszok,* amelyek – más folyamatokkal együtt – közös komponenseket vesznek igénybe. Ha ezek a szakaszok egymástól időben szigorúan nem határolódnak el, akkor átlapolás (angolul: *interleaving*) keletkezik, ami gyakran a rendszer összeomlásához vezet.

A szinkronizálást, és ezzel a folyamatok egyidejű lefutását hardver és szoftver elemek biztosítják.

A tárrekeszeket (vagy egy részüket) a **címek** egyértelműen azonosítják.

A tárrekeszek a logikai vagy a numerikus adatokat, valamint a program pozícióit, pl. a címkéket, és a perifériák vezérlőkaraktereit tartalmazzák.

A címek – a számítógép fajtája szerint – rendszerint 0 és $(2^{16}-1)$ vagy max. $(2^{44}-1)$ közé eső, előjel nélküli számok. Ha a programnak egy bizonyos tárrekeszhez kell hozzáférnie, akkor a megfelelő gépi kódú utasítás címrésze a tárrekesz címét tartalmazza.

Utasításnak (angolul: *instruction*) nevezzük a gépi nyelv legkisebb részét. Ez a munkafolyamat egy lépését adja meg, és a feladatorientált nyelvek *utasításának* felel meg.

A modern számítógépek 1, 2 vagy 3 címet is tartalmazhatnak, ilyen pl. a háromcímű számítógép.

Az operációs rendszer által biztosított **címzési funkciók** az utasításcímeket igény szerint munkatár abszolút címeire számítják át. Ez az átszámítás már program hívása esetén megtörténik.

Címzési módok

A címzés módja határozza meg, hogy az utasítás címrészében található cím ábrázolása abszolút, relatív, virtuális vagy valamilyen közbenső formájú-e.

Az **abszolút (közvetlen) címzés** esetén az utasításcím és a munkatár címe megegyezik.

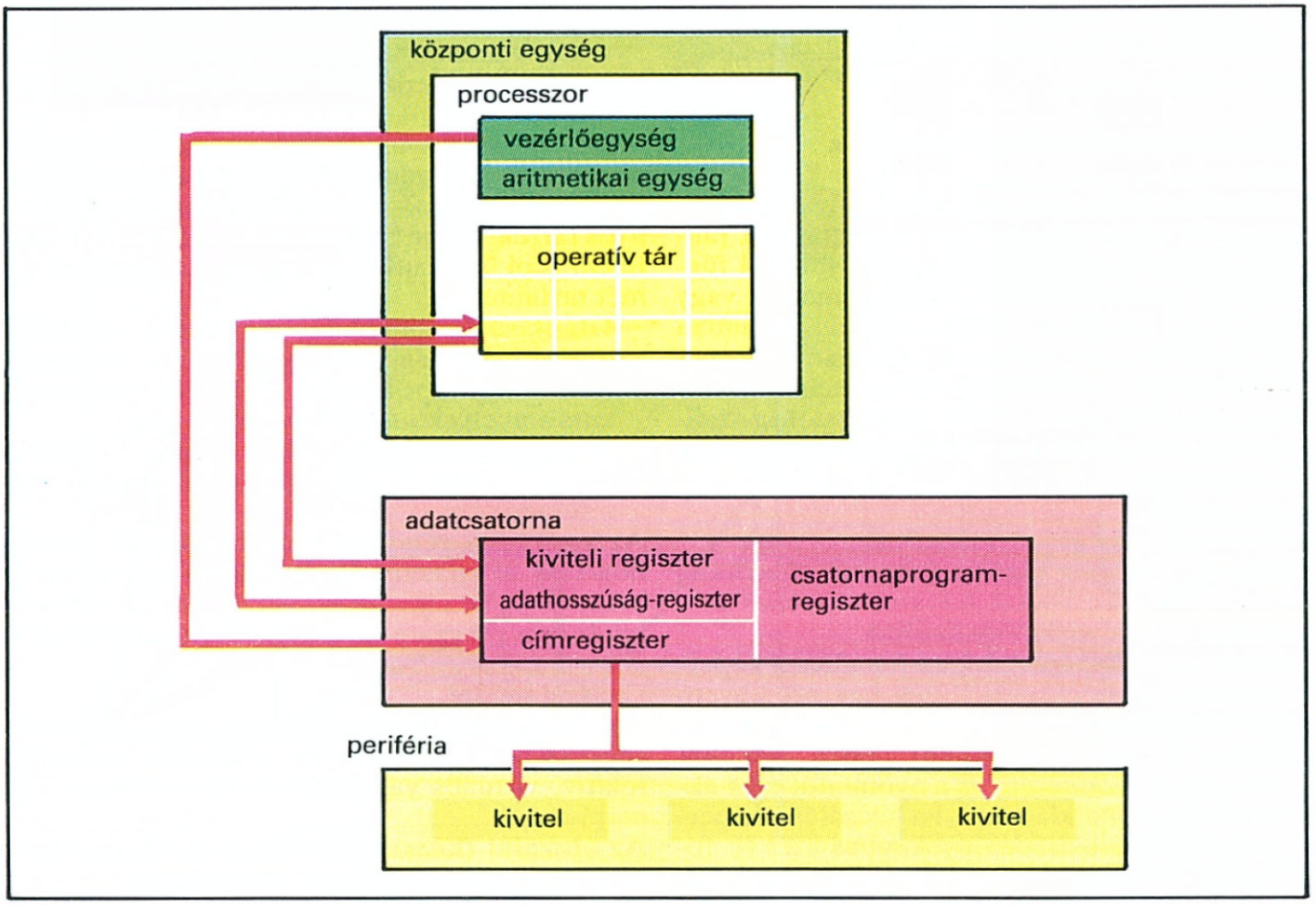
Az **indirekt (közvetett) címzés** esetén a címzett tárrekesz tartalma foglalja magában az utasítás abszolút címét.

A **relatív címzést** egész programok vagy programrészek esetén használjuk. Az abszolút cím ilyenkor a *bázisregiszter* tartalmának és az utasítás címrészében megadott címnek az összege. A program fordítása során báziscím mindig rögzített értéket vesz fel, és valamennyi programutasítás címe erre a báziscímre vonatkozik.

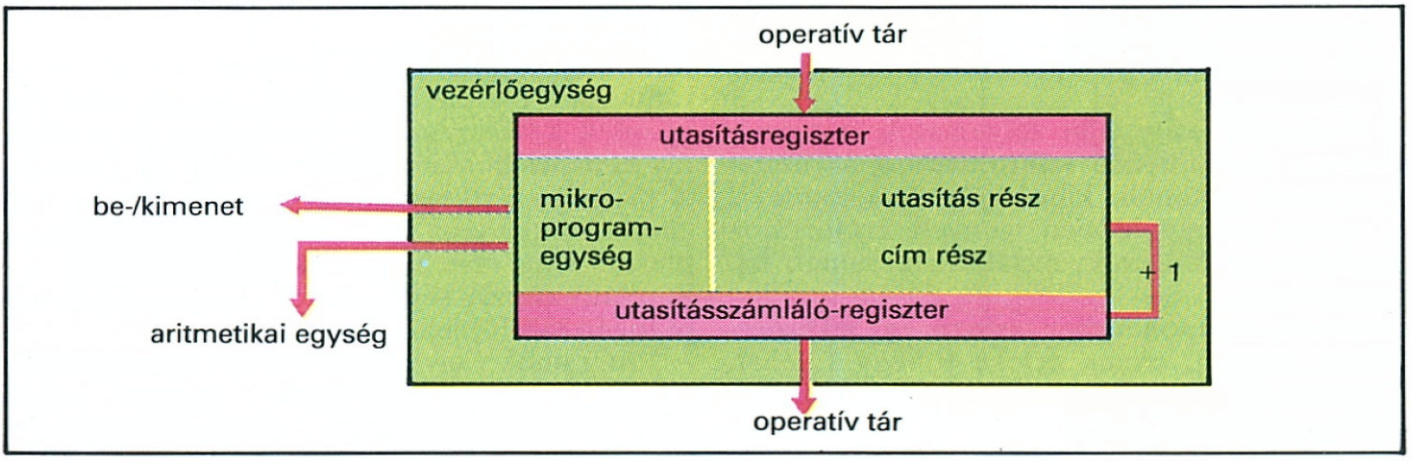
A relatív címzés segítségével a programok a számítógép tárolójában áthelyezhetők.

Programok vagy programrészek **áthelyezhetősége** (angolul: *relocation*) nagyon fontos a multiprogramozású üzemmód esetén: a program ilyenkor úgy tölthető be, hogy a munkatár kellően nagy – talán éppen felszabaduló – címtartományában futtatható. Mivel a program minden címe a báziscímre vonatkozik és a program hossza ismert, az első relatív cím minden további címet egyértelműen meghatároz.

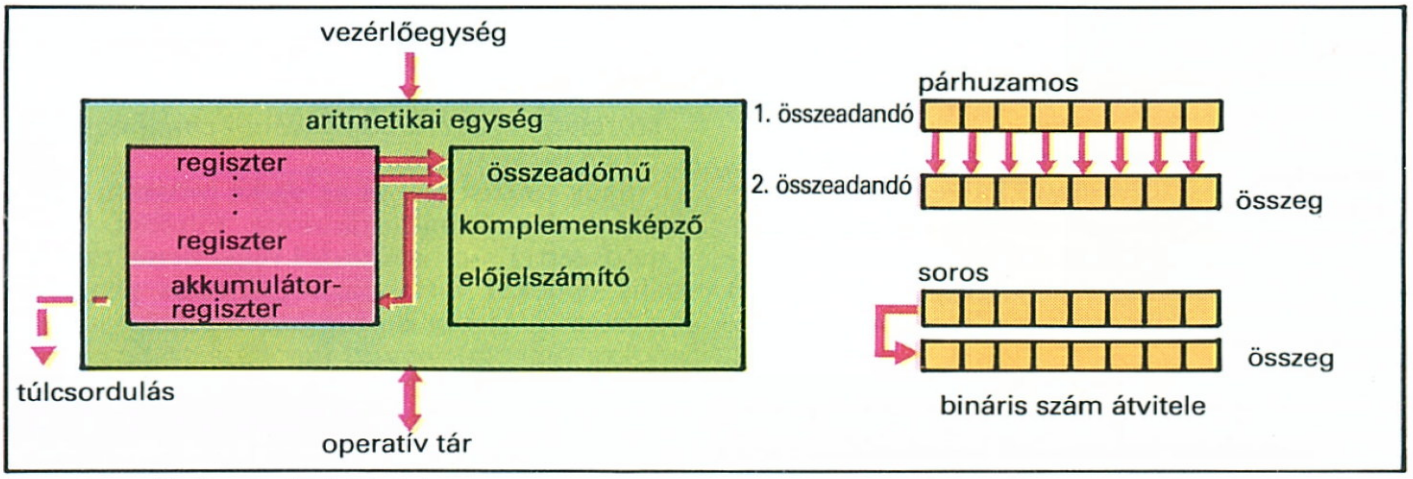
Virtuális címzés esetén az egész program összekapcsolva egy háttértárolóba töltődik. Ezzel a módszerrel a munkatártól függetlenül egy virtuális címeket tartalmazó „virtuális program" jön létre. A virtuális program egyenlő nagyságú lapokra (angolul: *pages*) tagolódik. Nagy számítógépek esetén 8192 (egyenként 2 Kbyte nagyságú) lap is előfordulhat. A programot hívásakor a betöltőprogram laponként továbbítja a munkatárba. Előnye: még nagyon hosszú programok is viszonylag kevés területet foglalnak el a munkatárban.



A központi egység és az adatcsatorna (adatkivitel esetén)



A vezérlőmű felépítése



Az aritmetikai egység felépítése

A **központi egység** (angolul: *Central Processing Unit,* **CPU**) lényegében a számítógép processzorát és főmemóriáját (munkatárát) foglalja magába.

A központi egység a két fő komponensen kívül számos, nagyon rövid elérési idejű, de csak csekély tárolókapacitású **regisztert** is tartalmaz, amelyek elsősorban közbenső eredmények, fontos tárolócímek és gyakran használt állandók értékeit tárolják. Maga a processzor is tartalmaz néhány regisztert.

Példa: A 80386-os mikroprocesszornak 16, egyenként 32 bit tárolókapacitású regisztere van.

A központi egység a számítógép legfontosabb része. A központi egység rendelkezik többek között a munkatárban vagy a csak olvasható ROM-ban tárolt operációs rendszer, valamint az aktuális felhasználói szoftver felett is.

A központi egység állandó összeköttetésben áll a perifériákkal, vagyis a perifériákhoz közvetlen hozzáférése van. Ennek az összeköttetésnek az alábbi feltételeket kell kielégítenie:

1. Függetlennek kell lennie a perifériaegységektől, ami azt jelenti, hogy a perifériák beépített mikroprocesszoruk segítségével átveszik saját vezérlésüket, pl. az adatátvitelt, szinkronizációt stb.
2. A perifériáknak a központi egységet nem szabad lassítaniuk. Mivel az utóbbi sokkal – kb. ezerszer – gyorsabb, a kapcsolatot gyors (gyakran >64 Kbyte kapacitású) közbenső tárak közbeiktatásával valósítják meg. Így a központi egység egyidejűleg gyakorlatilag több perifériát kiszolgálhat.

Processzor

A processzor a vezérlőegységből, aritmetikai egységből és gyors adattárolóból, azaz számos regiszterből áll; az utasítások lefutásának és végrehajtásának részleteit vezérli és ellenőrzi. A processzoradminisztráció határozza meg, hogy a következő lépésben melyik feladat foglalhatja el a processzort. Ezekért a feladatokért az **ütemező** (angolul: *scheduler* vagy *dispatcher*) részprogramok a felelősek.

A **parancs** (angolul: *command*) a processzoron belüli legkisebb munkafolyamat. A feladatorientált programnyelvek munkafolyamatának legkisebb lépését **utasításnak** (angolul: *instruction*) nevezzük.

A **többszörös utasítás** vagy **szekvencia** közvetlenül egymás után végrehajtandó parancsok vagy utasítások sorozata.

A **mikroprocesszorok** egy processzor összes komponensét egyetlen integrált áramköri lapkán a chipen egyesítik (106. o.).

Az **adatcsatornák** perifériaprocesszorok, amelyek az adatátvitelre és a be-/kiviteli egységek vezérlésére szolgálnak. A központi egység utasítására kezdik meg működésüket, ezt követően önállóan a központi egységgel párhuzamosan dolgoznak. Az adatcsatornák az operatív tárhoz *ciklusiopás* (angolul: *cycle stealing*) útján közvetlen hozzáféréssel rendelkeznek.

A **ciklusiopás** a központi egység munkafolyamatát egy tárciklusidővel (ütemmel) késlelteti. Ez alatt az idő alatt a perifériaprocesszor hozzáférhet az operatív tárhoz.

Az adatcsatorna általában a saját programját tárolja. Ez a következő feladatokat látja el: a be-/kimeneti egységek kiválasztása és vezérlése, az adatok és a vezérlésre alkalmazott kódok illesztése, az operatív tárhoz való közvetlen hozzáférés vezérlése.

A **szektor-adatcsatornák** csak egy perifériát látnak el. A **multiplex-adatcsatornák** egyidejűleg (párhuzamosan) több be-/kiviteli egységet is kiszolgálnak.

A processzor központi része a **vezérlőegység (vezérmű),** amely a parancsok feldolgozásának sorrendjét határozza meg. A vezérlőegység az aritmetikai egységgel és az operatív tárral áll kapcsolatban, továbbá a be-/kimeneti egységekkel is, ha azoknak nincs saját processzoruk.

A vezérlőegység a következő feladatokat látja el:

1. Az operatív tárból az utasítások helyes sorrendben történő átvételét és dekódolását.
2. A parancsok értelmezését, a számítógép megfelelő komponenseinek az ehhez szükséges impulzusokkal való vezérlését.

Az **aritmetikai (és logikai) egység** (angolul: *Arithmetic and Logical* **Unit, ALU**) a processzornak mindazon része, amely a vezérlőegységből érkező parancsok (aritmetikai és logikai műveletek) végrehajtására szolgál. Ezzel egyidejűleg a vezérlőegység az operandusokat a megfelelő sorrendben továbbítja. Ellenőrzi, hogy az éppen elvégzett művelet végrehajtása helyesen történt-e. A közbenső eredmények tárolására az aritmetikai egység számos regisztere áll rendelkezésre. Az **akkumulátor** az aritmetikai egység gyors hozzáférésű központi regisztere, amely az operandusokat közvetlenül a műveletek végrehajtása előtt tárolja. Közbenső eredmények felvételére is alkalmas és minden műveletben részt vesz.

Az aritmetikai egység az összes aritmetikai műveletet a regiszterekben tárolt két bináris szám ill. **komplemenseik** összeadására vezeti vissza.

Adott bináris szám **1-es komplemensét** úgy képezzük, hogy minden 0 számjegyet 1-gyel, és minden 1 számjegyet 0-val helyettesítünk.

Példa: Az 1101001 bináris szám 1-es komplemense 0010110.

Adott bináris szám **2-es komplemensét** úgy képezzük, hogy az 1-es komplemenshez +1-et hozzáadunk.

Példa: Az 1101001 bináris szám 2-es komplemense 0010111.

Ha a bináris szám bináris pontot tartalmaz, akkor a jobb szélső helyiértékhez kell 1-et hozzáadni.

Példa: Az 1001.11 bináris szám 1-es komplemense 0110.01.

Összeadás:
1. összeadandó + 2. összeadandó = összeg

1. összeadandó
+
2. összeadandó = összeg

Kivonás:
kisebbítendő – kivonandó = különbség

kisebbítendő
+
kivonandó kettes komplemense = különbség

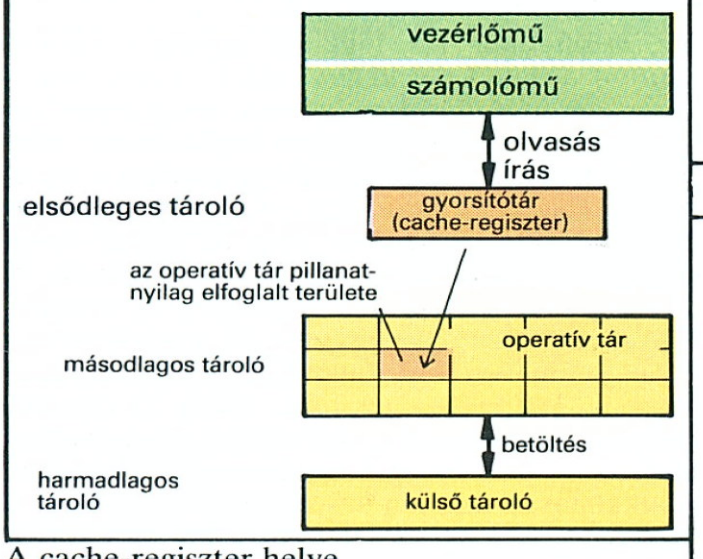
Szorzás:
1. tényező × 2. tényező = szorzat

1. tényező
+
1. tényező = 1. összeg
+
1. tényező = 2. összeg
+ (2. tényező – 3)-szor
= *n.* összeg
+
1. tényező = szorzat

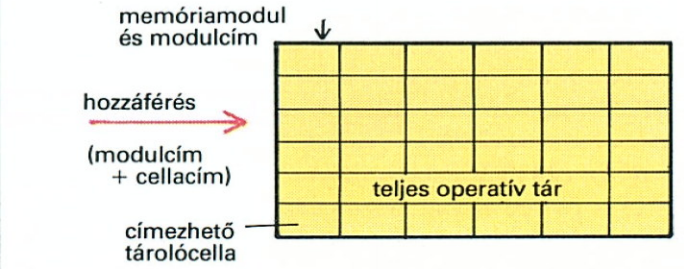
Osztás:
osztandó/osztó = hányados

1. osztandó
–
osztó kettes komplemense = 1. különbség
–
osztó kettes komplemense = 2. különbség
–
amíg az *n.* különbség < *n.* osztandó
helyiérték-eltolás
n. különbség + a következő helyiérték
–
az osztó kettes komplemense = ...

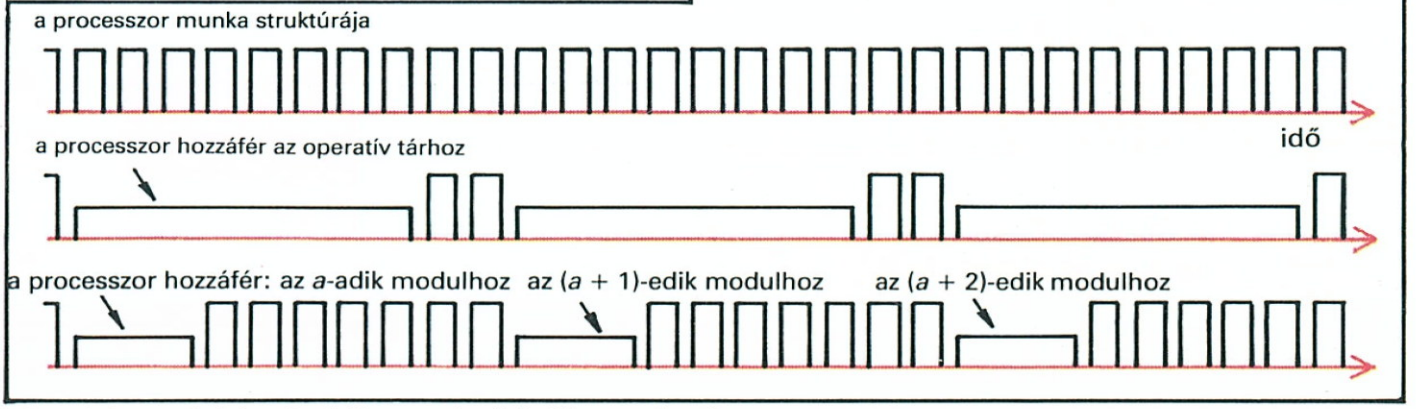
A négy alapművelet végrehajtása az aritmetikai egységben



A cache-regiszter helye
a tárolóhierarchiában



az egymás után következő adatokat
a szomszédos modulok tárolják



Memóriamodul és tárolóbeosztás (ideális esetben)

Az aritmetikai műveletek előjelét legtöbbször külön algoritmus kezeli.

Az **összeadómű** soros üzemmódban összeadja a két, külön regiszterben tárolt bináris számot és az eredményt az akkumulátorban tárolja. Az egymást követő összeadások értelemszerűen az akkumulátorban tárolt részeredmény felhasználásával történnek. A párhuzamos összeadás során valamennyi helyiérték összeadása (szinte) egyidejűleg zajlik. Ez lényegesen gyorsabb, de sokkal nagyobb műszaki ráfordítást igényel. A nagy integráltságú áramköröket tartalmazó számítógépek az összeadást (és a szorzást) párhuzamos üzemmódban végzik.

Túlcsordulás (angolul: *overflow*) vagy **alulcsordulás** (angolul: *underflow*) akkor fordul elő, ha a számítási eredmény több illetve kevesebb jegyből áll, mint az ábrázolható legnagyobb vagy legkisebb bináris szám. A számítógép az ábrázolási tartománynak ezt a túllépését legtöbbször hibaként jelzi.

Az *a* és *b* bináris számok **kivonása** úgy történik, hogy *a*-t hozzáadjuk *b* kettes komplemenséhez.

Két bináris szám **szorzását** ismételt összeadással és a megfelelő helyiértékeltolással hajtjuk végre. Két *n*-jegyű bináris szám szorzata $2n$ helyiértéket foglal el. A szorzás sorosan vagy párhuzamosan történhet.

Az **osztást** ismételt kivonások illetve kettes komplemensek ismételt összeadása és a megfelelő helyiértékeltolások útján lehet megvalósítani.

Operatív tár, munkatár

Ennek a tárnak lehetőleg nagynak, és nagyon gyors hozzáférési idejűnek kell lennie. Azonban a nagy tárkapacitás hosszú címekkel jár együtt, ami meghosszabbítja a hozzáférési időt. A PC-k operatív tárainak jellemző hozzáférési ideje kb. 80ns, a nagy számítógépeké 10 ns. A megfelelő tárolókapacitások 8 Mbyte ill. >512 Mbyte körül vannak.

Összehasonlítások: Ez a könyv betűk, számjegyek és jelek formájában kb. 0,7 Mbyte információt tárol.

Az ember hosszú távú emlékezetének tárolókapacitása mintegy 10^5 Mbyte, az elérési idő pedig kb. 100 ms.

Az operatív tár memóriablokkokra történő felosztása (az önálló blokkcímek miatt) csökkenti a hozzáférési időt. **(Illesztőtár)**

Az operatív tár a külső tárakból adatokat vagy programokat vehet át – ezt a folyamatot **betöltésnek** nevezzük –, azonban ez az átvitel viszonylag lassú. Az operatív tár – a hozzáférési idők szerint – *hierarchikusan* szervezett. A központi egység vezérlőegysége és aritmetikai egysége mindig a hierarchiában legmagasabban elhelyezkedő, azaz a leggyorsabban elérhető tárral áll kapcsolatban. A ritkán használt adatok a tárolóhierarchiában alul, a gyakran használt adatok pedig felül helyezkednek el. Ez a szervezés a tapasztalati **80–20-szabály** alapján csökkenti az átlagos elérési időt: a memóriahozzáférések 80%-a általában az összes adat 20%-ához történik.

A **regiszterek** az operatív tár rendkívül gyors részei (hozzáférési idő < 10 ns).

A **közbenső tár** az operatív tár azon részét veszi át, amelyet a program éppen használ. A kis kapacitás miatt (a címek egyidejűleg lerövidülnek) hozzáférési ideje gyorsabb, így a teljes számítási sebesség nő.

A **cache-regiszterek** nagyon rövid hozzáférési idejű közbenső tárolók, amelyek pl. a 386-os és a 486-os PC-generációkban találhatók meg. Arra szolgálnak, hogy a központi egység különböző sebességű komponenseinek számítási idejét kiegyenlítsék.

Az **utasításregiszter** (angolul: *Instruction Register*, **IR**) olyan párhuzamos regiszter, amely az aritmetikai egység éppen végrehajtás alatt álló utasítását tárolja.

Az **utasításszámláló** (angolul: *Instruction Counter*, **IC**) a soron következő utasítás címét tárolja, és az aritmetikai egységnek jelt ad a végrehajtásra. Ezek a részletek vagy a vezérlőegység fix (huzalozott) részei, vagy programozhatóak **(dinamikus mikroprogramozás).**

A központi egység sebessége

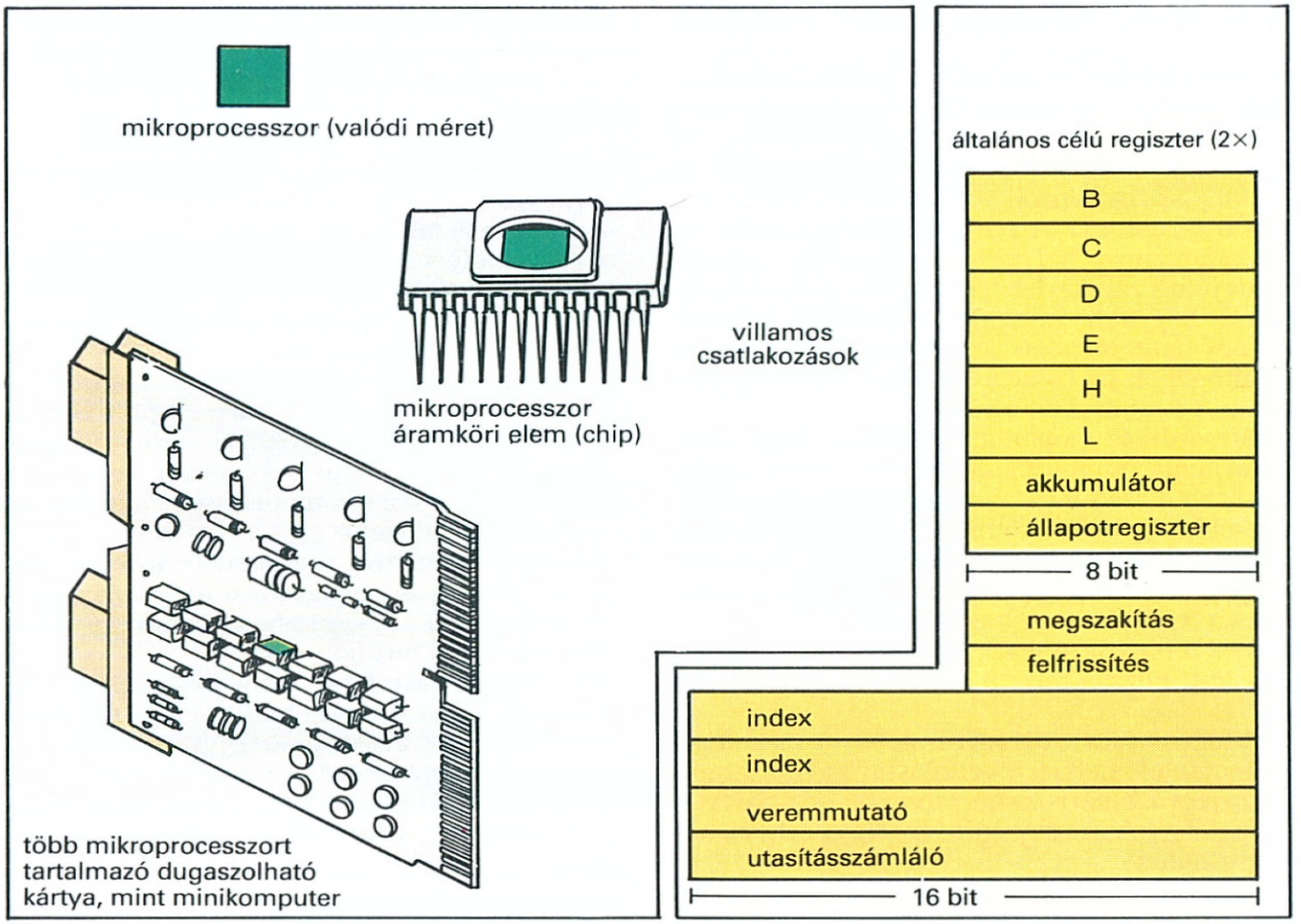
A **CPU-idő** az az időtartam, amelyre a központi egységnek egy meghatározott feladat végrehajtásához szüksége van.

MIPS (angolul: *Mega Instruction per Seconds*) a számítási sebesség súlyozott mértéke (70% összeadás és 30% szorzási utasítás).

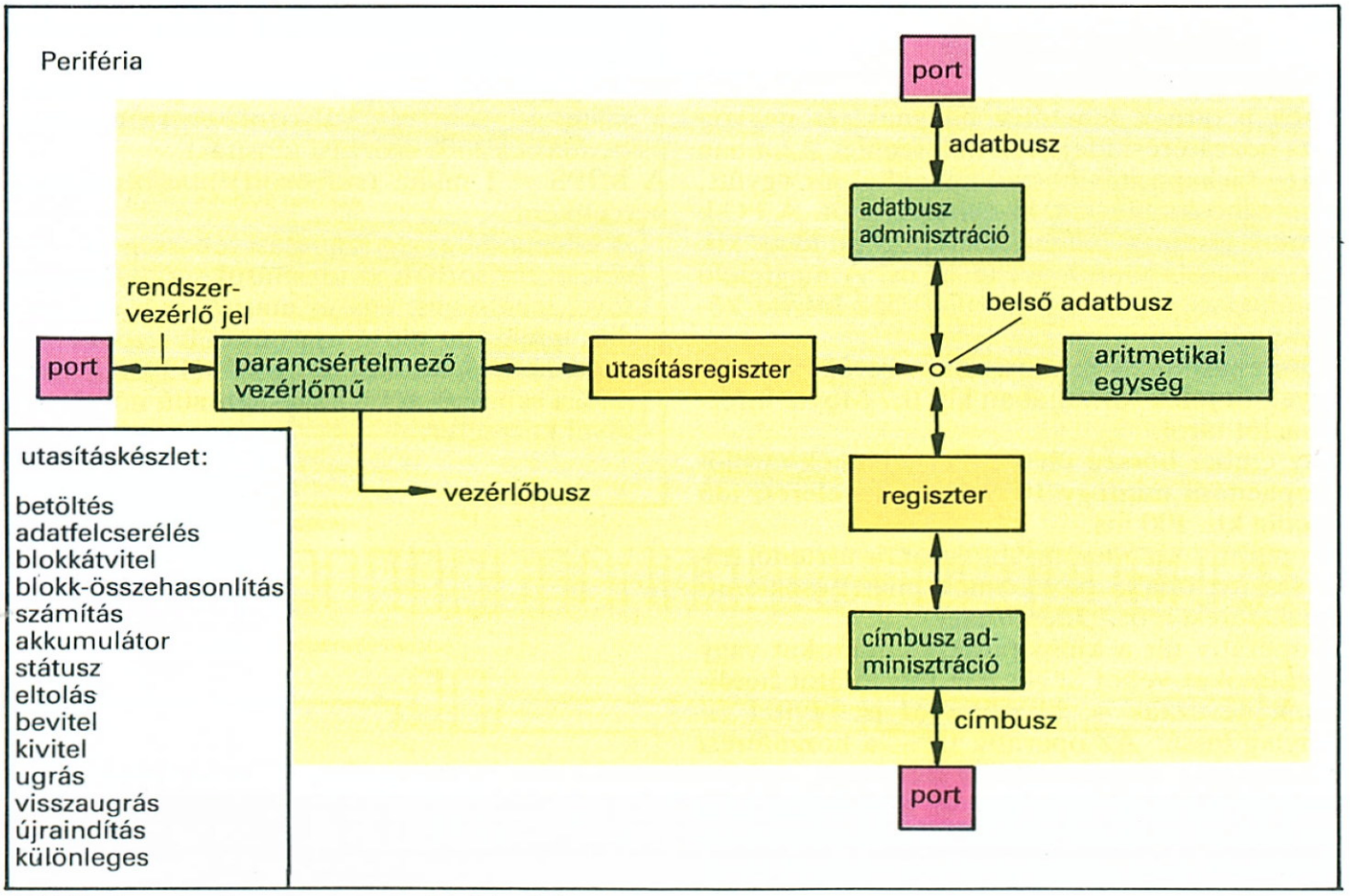
A MIPS = 1 millió (súlyozott) utasítás másodpercenként.

A központi egység számítási sebességének növelése elsősorban az utasítások időbeli átfedésével lehetséges: egy új utasítás már elkezdődik, mielőtt az előző utasítás befejeződne.

A párhuzamos adatfeldolgozás (199. o.) a számítási sebesség több nagyságrendű növekedésével kecsegtet.



Mikroprocesszor Regiszter



A Z80 típusú mikroprocesszor blokkvázlata

Az 1970-es évek óta lehetővé vált, hogy a processzor alkotóelemeit egyetlen nagyintegráltságú áramköri lapon (chipen) állítsák elő. Ezt az elektronikai alkatrészt **mikroprocesszornak** nevezzük. A chipen integrált elektronikai elemek száma rendkívül nagy, pl. a Pentium mikroprocesszor esetén 78 mm^2 felületen 3,1 millió elektronikai alkotóelem helyezkedik el.

A mikroprocesszorok teljesítőképessége a csökkenő árak ellenére növekszik: 1987-ben 1 MIPS még kb. 3000 $-ba került, 1994-ben már csak 200 $-ba.

A mikroprocesszorokat nagyon sokoldalúan lehet felhasználni, pl. háztartási készülékben, zsebszámítógépekben, szabályozó- és mérőkészülékekben.

A mikroprocesszorból, operatív tárból és a perifériákat kezelő vezérlésből álló építőelemek együttesét **mikroszámítógépnek** nevezzük. A mikroszámítógép és mikroprocesszor időközben egymás szinonímája lett, mert napjainkban már a mikroprocesszoron is elhelyeznek tárolót és be-/kimeneti vezérlést.

Az **illesztőegység** (angolul: *port, interface*) a mikroprocesszor és a perifériák közötti csatlakozást valósítja meg.

A mikroprocesszorok **műveleti sebességét** MIPS (angolul: **M***illion* **I***nstruction* **P***er* **S***econd*) egységben vagy BIPS (10^9 utasítás/s) egységben adjuk meg. Ezt a sebességet elsősorban a szinkronizáló órajel frekvenciája, a buszok szélessége és az adatvezetékek hossza határozza meg.

A különböző processzorok sebességének összehasonlítása nem egyszerű, mert a sebesség nemcsak a hardvertől, hanem a mikroprogramozástól is függ. Két szám összeadásának idejét sem tekinthetjük eléggé jellemzőnek. Értelmes becslést csak egy standard program segítségével megállapított számítási idő szolgáltat. A processzor órajelének frekvenciája a műveleti sebességre csupán támpontot nyújthat.

A **szóhosszúság:** a párhuzamosan, tehát egyidejűleg feldolgozható bitek száma, amely elsősorban a közvetlenül címezhető tárolócellák számát határozza meg.

Példa: 8 bit szóhosszúság esetén 256 (2^8), 16 bit szóhosszúság esetén pedig 65536 (2^{16}) különböző cím képezhető.

A legtöbb processzor lehetővé teszi, hogy a cím két összekapcsolt szó lehessen, vagyis a szóhosszúság megkétszerezhető.

A kereskedelmi forgalomban kapható mikroprocesszorok **közvetlenül címezhető tárolócelláinak száma** PC-k esetén 64 Kbyte és 4 Gbyte között van, nagy számítógépeknél pedig > 1 Gbyte. Az adattároló fontos része a max. húsz, gyors hozzáférésű, sokszor párosával címezhető, általános célú regiszter, amely az aritmetikai és logikai műveletek elvégzésére, akkumulátorként, utasításszámlálóként stb. használható.

Az **utasításkészlet** a (firmware-ként) rendelkezésre álló, pl. adatátviteli, számolási, ugró-, be-/kiviteli, különleges stb. utasítások összessége. Minél átfogóbb az utasításkészlet, annál teljesítőképesebb a mikroprocesszor. Másrészt viszont a nagy utasításkészlet a chipen drága tárolóhelyeket foglal el, és így csökkenti az elérhető műveleti sebességet. Ha az utasításkészlet kicsi, akkor a programozás ugyan nehezebb, a meglévő utasítások azonban gyorsabban hajthatók végre.

Ismertebb mikroprocesszorok:

típus	szóhosszúság	jellemző műveleti sebesség (MIPS)	címtartomány (Mbyte)	kapcsolóelemek száma
8080	8	0,5	0,064	5000
6800	16	0,67	0,016	
Z 80	8	0,62	0,064	
8088	8 (16)	1	1	29000
80286	16	10	16	134000
80386	32	20	4096	275000
80486	32	40	4096	1200000
Alpha	64	90		1700000
Pentium	32 (64)	112	4096	3100000

A mikroprocesszorok a tömeggyártás miatt viszonylag olcsók; 1992-ben az Intel cég 30 millió 486-os processzort adott el, darabonként kb. 200 DM áron.

A mikroprocesszorok buszrendszere

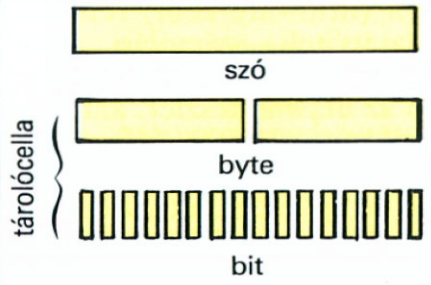
A mikroprocesszoron belül a működési egységek közötti információcsere belső gyűjtővezetékeken, az ún. **buszon** zajlik. Az órajelgenerátor segítségével a buszadminisztráció gondoskodik az összes művelet helyes időbeli lefutásáról.

A processzor a **címbuszt** arra használja, hogy az adatátvitelhez egy meghatározott tárolócella vagy adatvezeték címét kijelölje. Az átvitel a processzortól a tárolóhoz vagy az illesztőegységhez legtöbbször egyirányú. A címbusz mérete határozza meg, hogy adott számítógépben mekkora lehet a megcímezhető memória nagysága.

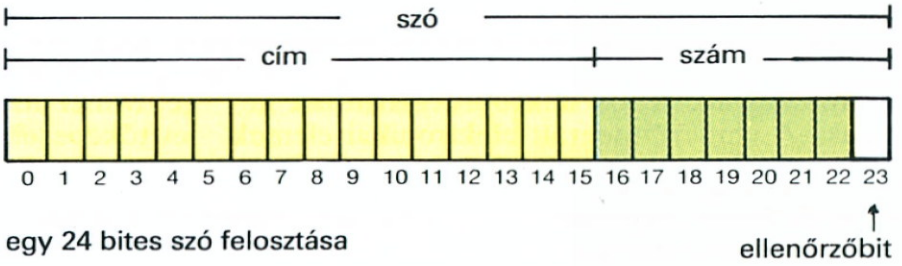
Az **adatbuszon** a processzor és a számítógép más egységei közötti adatforgalom zajlik. Az adatforgalom általában kétirányú. A busz szélessége az alkalmazott mikroprocesszortól függ.

A **vezérlőbusz** vezérlőjeleket továbbít. Elsősorban a vezérlőegység és az illesztőegységek közötti adatbuszon folyó adatáramlás irányát határozza meg; pl. az „írást", vagy az „olvasást".

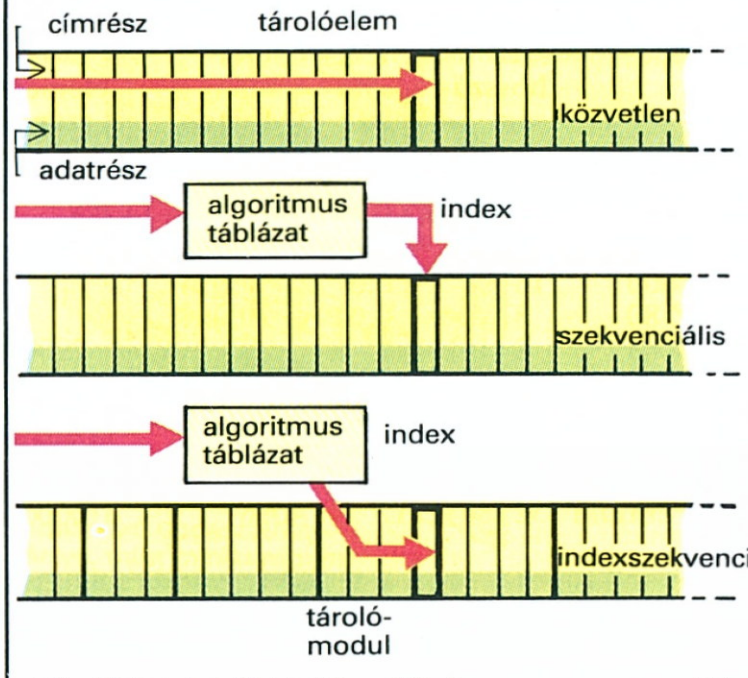
A **buszszélesség** az egyidejűleg átvihető, vehető és feldolgozható bitek számát határozza meg. PC-k esetén az adatbusz szélessége 8, 16 vagy 32 bit, a címbusz szélessége pedig 20, 24 vagy 32 bit.



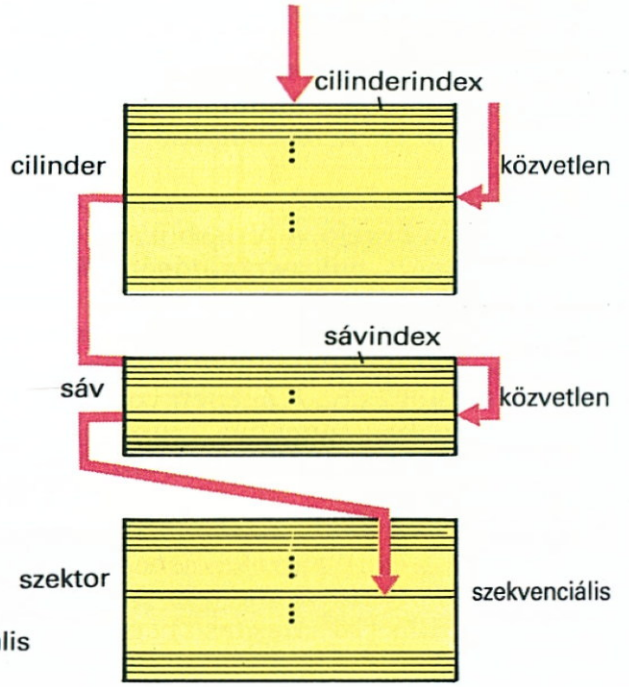
Tárolóelem felosztása



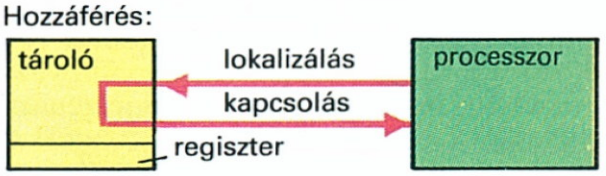
egy 24 bites szó felosztása

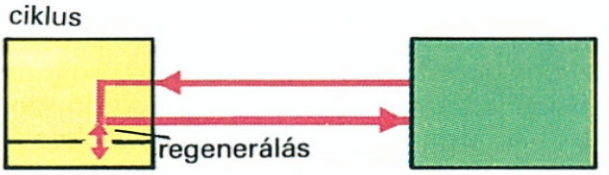


Tárolóelemhez történő hozzáférés

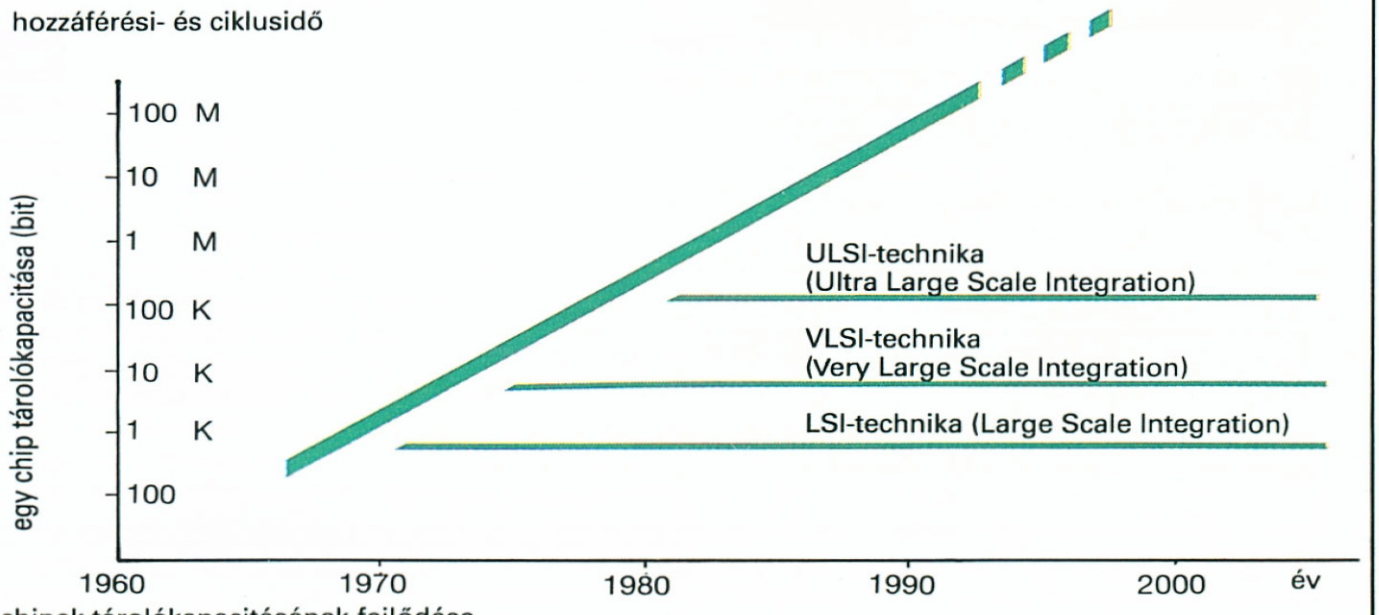


Mágneslemezes tárolóhoz történő hozzáférés (2 index)





hozzáférési- és ciklusidő



chipek tárolókapacitásának fejlődése

A számítógépnek azokat az alkotórészeit, amelyeknek az adatok megőrzése a feladatuk, **(adat) tárnak** (angolul: *memory* vagy *storage*) nevezzük. A belső tárak (munkatár, operatív tár) a központi egység szerves részei. A külső tárak, pl. a mágneslemezek, a számítógép perifériájához tartoznak.

A **hozzáférési idő** (angolul: *access time*) az az időtartam, amely egy tárolócella címzéséhez és tartalmának átviteléhez szükséges. Nagy számítógépek esetén ez az érték 1 ns körül van, PC-k esetén kb. 80 ns. A **ciklusidő** (angolul: *cycle time*) az az időtartam, amely az egyik tárolóművelet kezdetétől a következő művelet elejéig tart.

A digitális számítógépek különálló bistabil (bináris) tárolóelemekből állnak, ami azt jelenti, hogy valamennyi elem a két lehetséges állapot egyikét foglalja el. Az egyik állapotból a másikba történő átmenet valamilyen külső jel (pl. villamos feszültségimpulzus) hatására következik be. A tárolóelemek legkisebb, külön vezérelhető (címezhető) csoportja egy tárolócellát alkot. Az adatoknak a tárolóba való bevitelét *írásnak*, a tárolóból való kivitelét pedig *olvasásnak* nevezzük. Íráskor a tárolócellában előzőleg tárolt adatok legtöbbször elvesznek, a cellák tartalma *felülíródik*.

A **cím** minden egyes tárolócellát egyértelműen azonosít. Ha egy cella tartalma olvasásra kerül, akkor a megfelelő utasításnak a cella címét is tartalmaznia kell. Minden tárolócella olvasása saját címével, átvitele egységes egészként történik.

Tárolókapacitás

A kapacitást az adattároló adatfelvevő képessége jellemzi, ezt bit vagy byte egységekben mérjük.

A **bit** a bináris alakban megadott adatok legkisebb egysége. Egy bit az 1 vagy a 0 bináris értékeket tárolhatja.

Egy **byte** 8 bitnek felel meg.

A **szó** vagy **memóriaszó** több tárolócellának egy új tárolóegységgé történő összefoglalását jelenti. Egy adott szó az adatbusz szélességétől függően egy vagy több byte-ból állhat.

A tárolókapacitást a bit és a byte többszöröseinek segítségével adjuk meg (az egységek átszámítási táblázata a 9. oldalon található).

A tárolók típusai

Félvezetős tárolók:

A **ROM** (angolul: **R***ead* **O***nly* **M***emory*) fix tároló, programozása a gyártáskor történik, a felhasználó a tartalmát csak olvasni tudja, megváltoztatni azonban nem.

A **PROM** (angolul: **P***rogrammable* **ROM**) tárolójának tartalmát a felhasználó programozza, ezután azonban megváltoztathatatlan.

Az **EPROM** (angolul: **E***rasable* **PROM**) a tároló tartalma – különleges módszerekkel, pl. ultraibolya fénnyel történő besugárzás hatására – törölhető és újra programozható.

Az **EEPROM** (angolul: ***E**lectrically* **EPROM**) viszonylag kevesebb ráfordítással, egy speciális programozó készülék villamos impulzusainak segítségével bitenként programozható és törölhető.

A **RAM** (angolul: **R***andom* **A***ccess* **M***emory*) tetszőleges hozzáférésű tár. Olvasható, törölhető és újra írható. A RAM tárolóknak külső tápfeszültségre van szükségük az adatok tárolásához, vagyis a számítógép kikapcsolásakor vagy áramkimaradás esetén a tárolt adatok elvesznek. Operatív tár céljára használják.

Adathozzáférés

A tárban lévő adatokat különböző módon lehet beírni és kiolvasni.

Szekvenciális adathozzáférés. A tárolócellákból az adatokat az adatbevitel sorrendjében lehet kiolvasni. Mielőtt a keresett információhoz jutunk, a térben előtte levő összes cellát (ill. ezek címét) ki kell olvasni. A hozzáférési idő akkor rövid, ha az adatok eleve a kívánt sorrendben helyezkednek el.

Példa: mágnesszalagos tároló.

Közvetlen (tetszőleges) hozzáférés (angolul: *random access*) során az adatok írása és olvasása minden helyzetben a tároló tetszőleges helyére történhet. A keresett tárolócella címét egy táblázat segítségével lehet kiolvasni vagy kiszámítani. A hozzáférési idő független az adatoknak a tárolóban való elhelyezkedésétől.

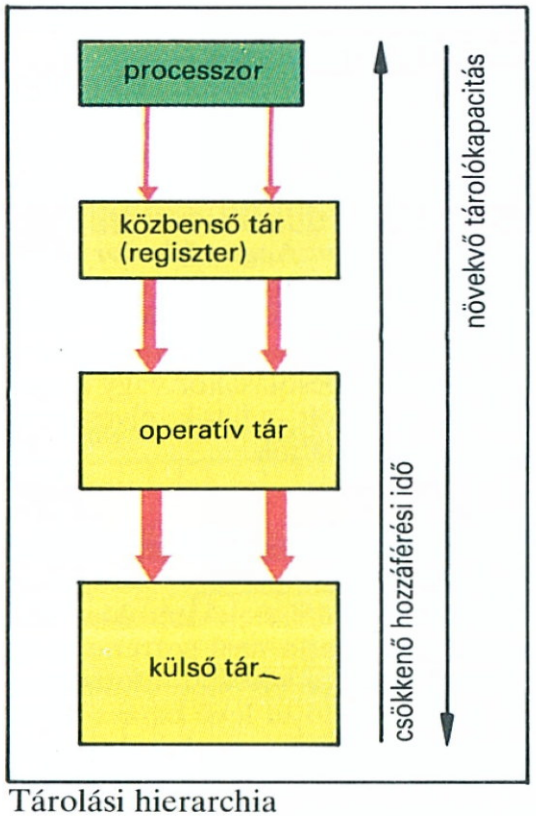
Példa: Félvezetős tároló.

Indexszekvenciális hozzáférés a közvetlen és a közvetett adathozzáférés közötti átmenetet képviseli (angolul: *semidirect access*), amelynek esetén a közel azonos nagyságú tárolóblokkokat indexek jelölik meg. Az egyes modulokon belül szekvenciális keresés lehetséges. Ez a kevert tárolási forma a tárolót jobban kihasználja, a hozzáférési idő a közvetlen, és a szekvenciális adatelérésre jellemző idők közé esik.

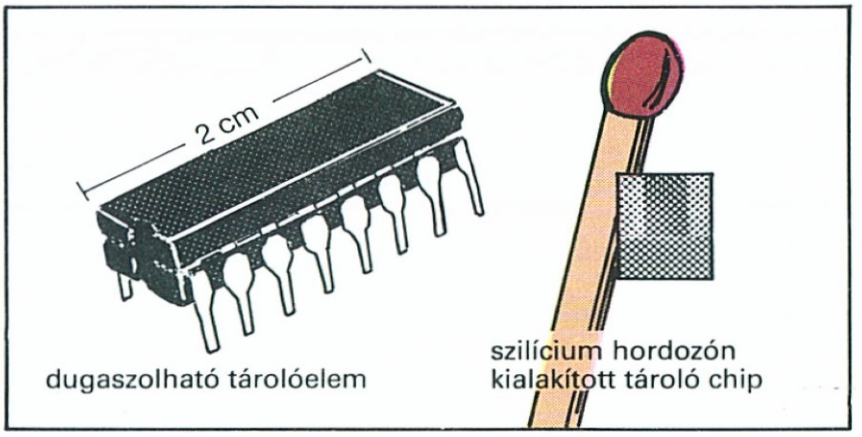
Példa: mágneslemezes tároló.

Adatbiztonság

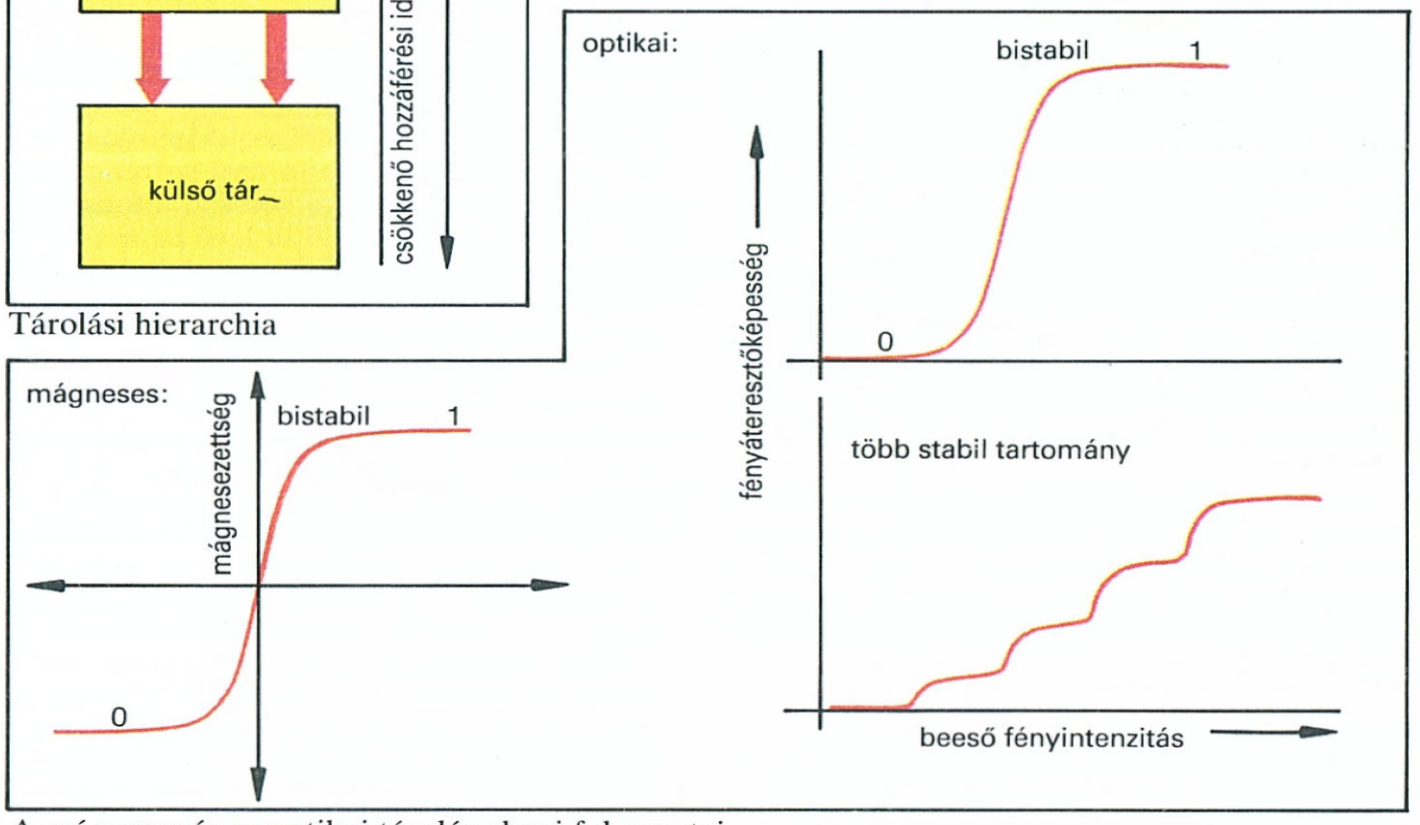
Az adatbiztosítás feladata az adatok illetéktelen olvasásának vagy írásának a megakadályozása, amelyet rendszerint a felhasználók által választható jelszó biztosít.



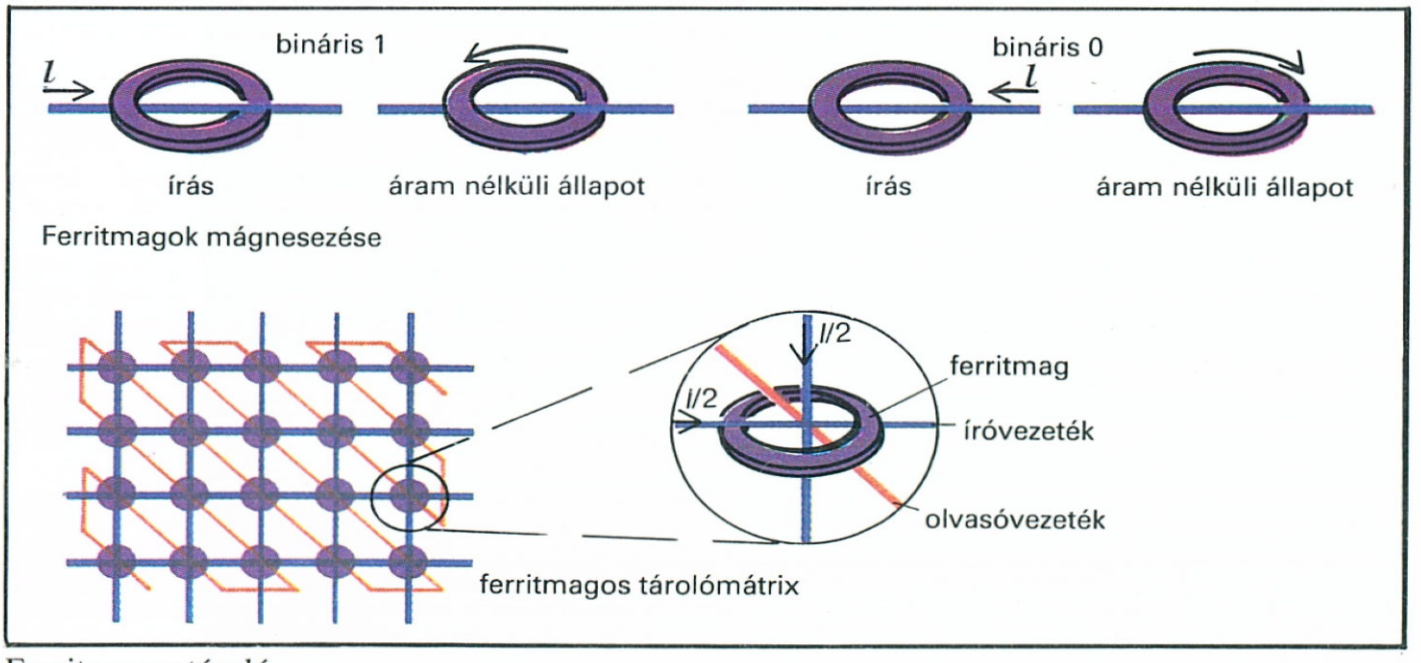
Tárolási hierarchia



Félvezetős tároló



A mágneses és az optikai tárolás elemi folyamatai



Ferritmagos tároló

Az **operatív tár** (munkatár) a központi egység része, amely az adatokat és a programokat azonnali hozzáférésre kész állapotban tárolja. Az operatív tár mindig RAM, azaz közvetlen hozzáférésű tároló. Mivel a nagy számítási sebességek eléréséhez rövid hozzáférési időkre van szükség, csak a viszonylag drága félvezetős tárak (lásd alább) jöhetnek szóba. Az operatív tárnak lehetőleg nagy kapacitásúnak kell lennie, vagyis sok tárolócellát kell tartalmaznia, ez azonban hosszú címeket igényel, amelyek olvasása növeli a hozzáférési időt, és csökkenti a számítási sebességet. A tárolókapacitást és a hozzáférési időt tehát optimalizálni kell.

Példa: nagy számítógépek esetén az operatív tár hozzáférési ideje és kapacitása jellemzően 10 ns ill. >256 Mbyte, a PC-k esetén pedig kb. 80 ns (néhányszor 25 ns cache-regiszterrel) és mintegy 8 Mbyte (4 Gbyte-ig bővíthető).

Az operatív tár szóhosszúsága legtöbbször 16-tól 64 bitig terjed, gyakran közvetlenül címezhető byte-okra van felosztva.

A **regiszterek** különösen kis hozzáférési idejű tárolócella-együttesek. A gyakran használt adatok és címek, valamint az azonnal további feldolgozásra kerülő részeredmények tárolására szolgálnak. A regiszterek teljes mikroprogramokat, pl. két szám szorzását leíró mikroprogramot is tárolhatnak. Az elérési idejük ekkor kb. 25 ns. A regiszterek ügyes felhasználásával a számítási sebesség fokozható.

Az operatív tár **tárblokkokra** történő felosztása lehetővé teszi a hozzáférés időbeli átfedését, és ezáltal ugyancsak megnöveli a számítógép számítási sebességét.

Félvezetős tároló

A tranzisztorokból és ellenállásokból felépített integrált áramkörök kompakt, tokozott egységet alkotnak, amelyet **tárolóchipnek** nevezünk. Az áramköri elemek párologtatási technológiájával nagy integráltsági fokot lehet elérni, így 1–100 mm^2 nagyságú és 0,1 mm vastagságú chipek sok Mbyte információ tárolására képesek. A rövid összekötő távolságok mikroszekundumnál rövidebb hozzáférési időket tesznek lehetővé.

A **MOS-tárolók** (angolul: **M***etal* **O***xid* **S***emiconductor*) – kb. 50 nm átmérőjű – térvezérlésű tranzisztorokat tartalmaznak, amelyek vezérléséhez rendkívül kis áramok szükségesek. Hőtermelésük ennek megfelelően csekély, és nagy integrációs sűrűséget tesz lehetővé. A közepes hozzáférési idő kb. 100 ns. Az 1990-es években a 8 Mbyte kapacitású MOS-chipek dominálnak. A 64 Mbyte-os chipek már a laboratóriumi vizsgálatok stádiumában vannak, és valószínűleg 1000 Mbytos chipek is előállíthatók. A MOS-tárolókat elsősorban operatív tárként alkalmazzák. A **bipoláris félvezetős tárolók** nagyon gyors tranzisztorokat tartalmaznak, de viszonylag sok hőt adnak le. Integráltsági fokuk ezért kisebb, mint a MOS-technikával előállított tárolóelemeké, hozzáférési idejük viszont <30 ns. Elsősorban gyors regiszterekként alkalmazzák.

Ferritgyűrűs tároló

A ferritgyűrűs tárolókban apró, oxidkerámia alapú mágneses anyagból (ferritből) készült gyűrűk alkotják a bistabil tárolóelemeket. Minden gyűrű két író- és egy olvasóvezetékre van felfűzve. A ferritgyűrűs tároló egy x sorból, y oszlopból és z rétegből álló háromdimenziós mátrixot képez. A mágnesezettség irányától függően minden gyűrű a 0 vagy az 1 bináris értékek egyikét reprezentálja.

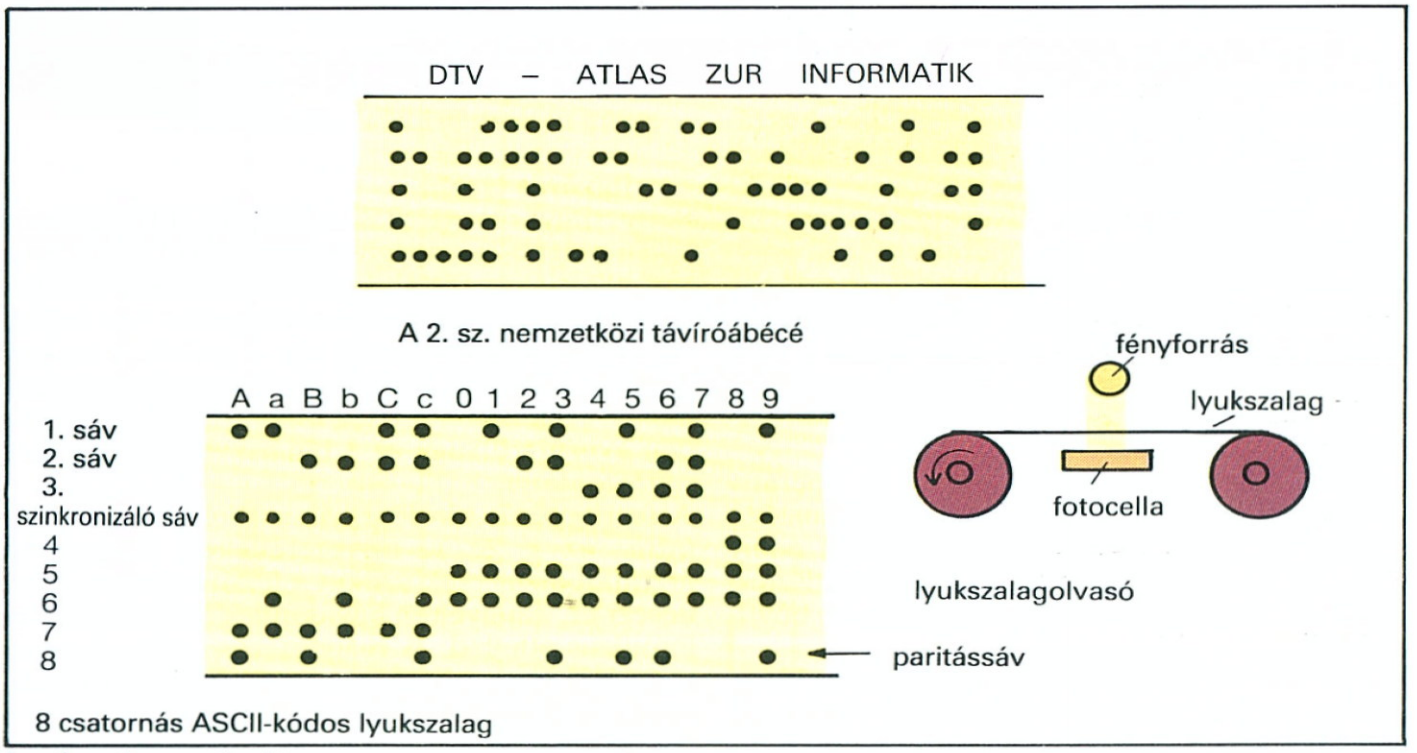
A ferritgyűrűs tároló egyik pozíciójának (x_i, y_i) *írása:* az x_i és y_i írővezetékeken átbocsátott áramimpulzus csak azokat a ferritgyűrűket mágnesezi át, amelyeken egyidejűleg mindkét áramimpulzus áthalad. A mágnesezési irány az áram irányától függ. Mivel az egy vezetéken áthaladó áramimpulzus erőssége csak a felmágnesezéshez szükséges érték 50%-át éri el, a többi ferritgyűrű mágneses állapota nem változik.

A ferritgyűrűs tároló egyik pozíciójának (x_i, y_i) *olvasása:* az x_i és y_i írővezetékeken átbocsátott áramimpulzus hatására az átmágnesezés folytán akkor és csak akkor indukálódik áramlökés az olvasóvezetékben, ha a tárológyűrű állapota az 1 bináris értéknek felelt meg. Ha a tárológyűrű állapotának bináris értéke 0, akkor nem történik átmágnesezés, és az olvasóvezetékben nem keletkezik áramlökés. Mivel a kiolvasás a tároló állapotát megváltoztatja, az eredetileg tárolt információt az olvasást követő folyamattal vissza kell állítani *(felfrissítés)*.

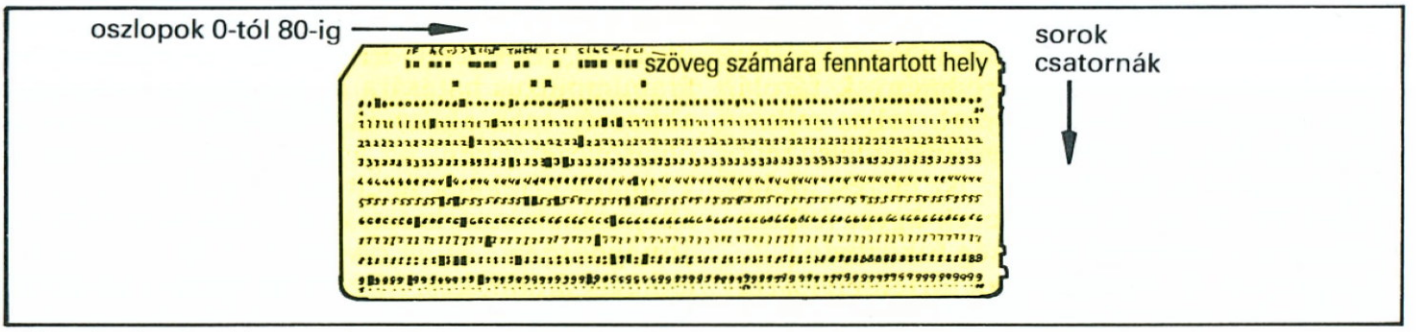
A ferritgyűrűs tárak hozzáférési ideje rövid, de helyigényük nagy és drágák. A félvezetős tárak gyakorlatilag teljesen kiszorították őket.

Optikai gyors tárolók

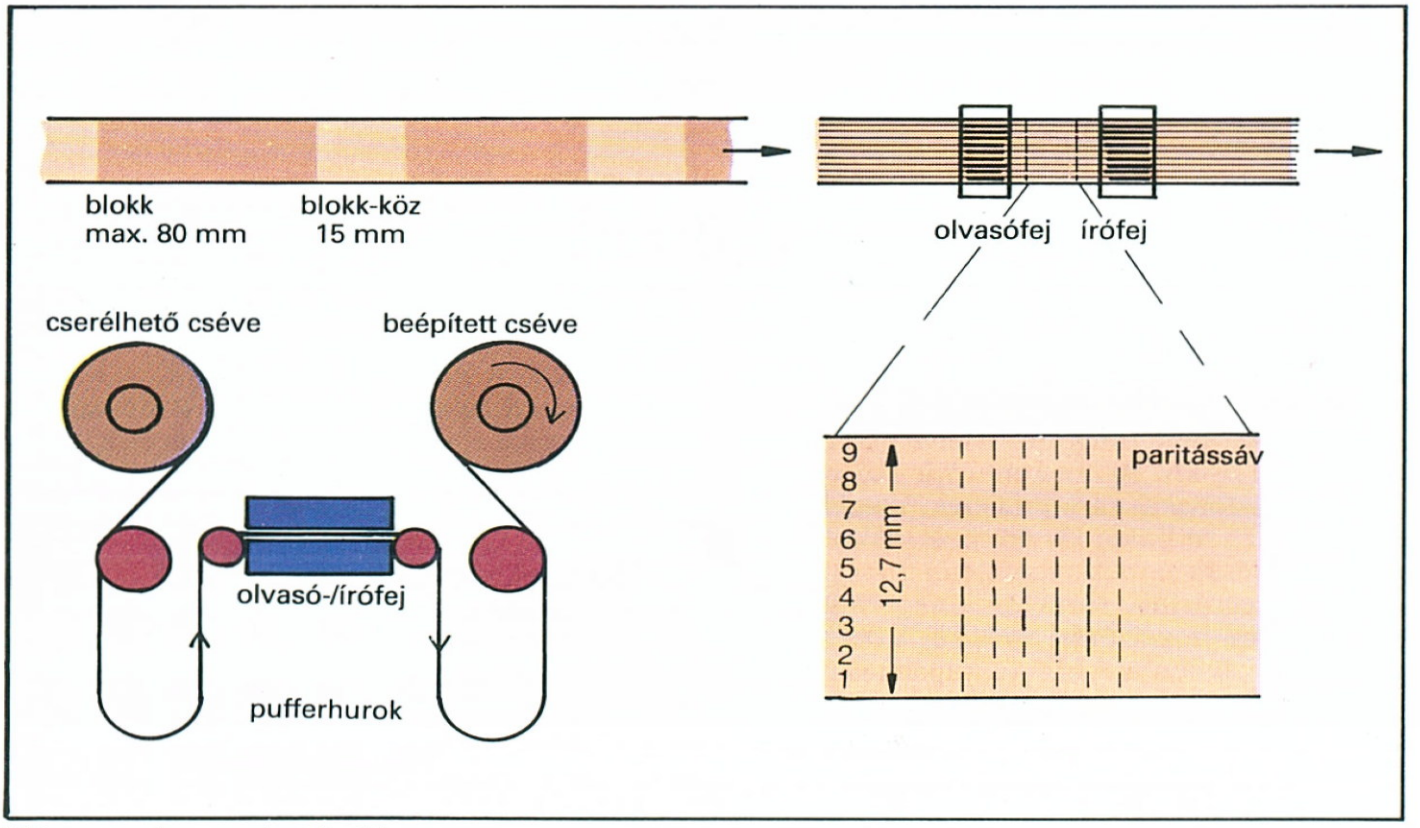
Néhány alacsony hőmérsékletre lehűtött kristály (pl. a gallium-arzenid (GaAs) vagy az indium-antimonid (InSb) törésmutatója ultraibolya fénnyel történő megvilágítás hatására lépcsőzetesen megváltozik. Ez az effektus adattárolásra is felhasználható. A beírás és kiolvasás pl. CO_2 lézer segítségével történhet. A hozzáférési idő a nanoszekundumnál rövidebb tartományba esik. Ezek az optikai tárolók még kísérleti stádiumban vannak és egyelőre még nincsenek kereskedelmi forgalomban (197. o.)



Lyukszalagos tároló



BCD-kódos lyukkártya betűk és számok tárolására



Mágnesszalagos adattároló

Áramkimaradás esetén az operatív tárban lévő információ általában elvész. Külső tárak esetén ez nem áll fenn. Ezek a tárak olyan adatokat tárolnak, amelyek a számítógépben éppen lezajló folyamatok számára nem szükségesek. Elsősorban a hosszú távú tárolás céljait szolgálják. Napjainkban a modern periferiális tárolók elsősorban mágnesezhető anyagokból állnak.

Lyukszalagot és lyukkártyát már csak elvétve használnak, mert ezeknek a tárolóknak a hozzáférési ideje hosszú, a tárolási sűrűségük pedig kicsi.

Lyukszalag

Adathordozójuk 17,5–25,4 mm széles, felcsévélhető papír- vagy műanyag szalag. A kódolt adatokat a szalagtovábbítás irányával párhuzamos sávokba (csatornákba) rendezett lyukak segítségével tárolja. 5 és 8 lyukat alkalmazó kódokat szoktak használni. A szalagtovábbítás irányára merőleges lyuksor egy jelet ábrázol. A szalagtovábbító mechanizmus számára kisebb lyukakból külön sáv (ütemsáv) áll rendelkezésre.

Az *5 csatornás lyukszalag* a nemzetközi távíróábécét alkalmazza. Maximálisan $2^5 = 32$ különböző jelet lehet lyukasztani. Mivel ez nem elegendő, a betűk és számok közötti átváltás külön jel segítségével történik.

A *8 csatornás lyukszalagok* a 7 bites ASCII-kódot használják, a 8. sáv az ellenőrzőbit helye. Maximálisan $2^7 = 128$ különböző jel kódolható.

A lyukszalagolvasók a szalagokat villamos vagy fotoelektromos módszerrel tapogatják le. A szalaglyukasztók adatbevitele billentyűzettel történik, amelynek minden leütésére a szalagon a továbbítás irányára merőlegesen egy lyuksor jelenik meg.

Lyukkártya

Az adathordozó 82,55 mm × 187,32 mm méretű különleges vékony kartonkártya. A kártya méretei megfelelnek az 1890-ben forgalomban levő dollár bankjegyek méretének, amikor HOLLERITH az USA-ban a népszámlálás kiértékelésére bevezette a lyukkártyákat. A jeleket négyszögletes lyukak kombinációja kódolja. Egy lyukkártya legfeljebb 80 jelet tartalmazhat.

A lyukak pozícióit villamos úton olvassák ki, és a kiolvasás sebessége a max. 30 kártya/s értéket is elérheti. Az 1970-es évek kezdete óta már csak elvétve használják. Olcsó és megbízható adattároló.

Mágnesréteges adattárolók

Az adathordozó nem mágnesezhető anyagra felvitt, nagyon vékony, mágnesezhető réteg. Ide tartoznak a **hajlékonylemezek,** a **merevlemezek** és a **kazetták.** A réteget egy meghajtómechanizmus egy vagy több író-/olvasófej előtt vezeti el. Az információ tárolása bináris alakban egy sávban történik.

Tárolási sűrűségnek nevezzük a tárolt információ hosszúság- vagy területegységre eső mennyiségét, amelyet **bpi** (angolul: ***b**its **p**er **i**nch*) ill. **bpi**^{2} (angolul: ***b**its **p**er square **i**nch*) egységekben adunk meg. Használatosak még a bit/mm, a bit/mm^2 valamint a byte/mm és a byte/mm^2 egységek is.

Átszámítások: 1 bpi = 0,0394 bit/mm,
1 bit/mm = 25,4 bpi;
1 bpi^2 = 0,00155 bit/mm^2,
1 bit/mm^2 = 645 bpi^2.

A **sávsűrűség** egysége **tpi** (***t**racks **p**er **i**nch*).

Példa: A modern mágneslemezek jellemző sávsűrűsége 96 vagy 136 tpi.

A tárolási sávok egymástól mért távolsága 0,05 mm-től 0,1 mm-ig terjed, a lineáris tárolási sűrűség eléri a 300 bit/mm (kb. 7620 bpi), a felületi tárolási sűrűség pedig a 4500 bit/mm^2 ($2{,}9 \times 10^6$ bpi^2) értéket.

Az adatbeírás során az író-/olvasófej elektromágnesének légrésében egy áramimpulzus mágneses mezőt hoz létre, amely közvetlenül a fej alatt megváltoztatja a réteg mágnesezettségének irányát. Olvasáskor a fej előtt elmozgatott szalag mágneses pozíciója az író-/olvasófejben villamos áramot indukál, amely tovább feldolgozható.

A mágnesréteges adattárolás technikája gyorsan fejlődik. A merevlemezes tárolók (115. o.) egyre kisebbek és növekvő kapacitás mellett egyre gyorsabbak lesznek.

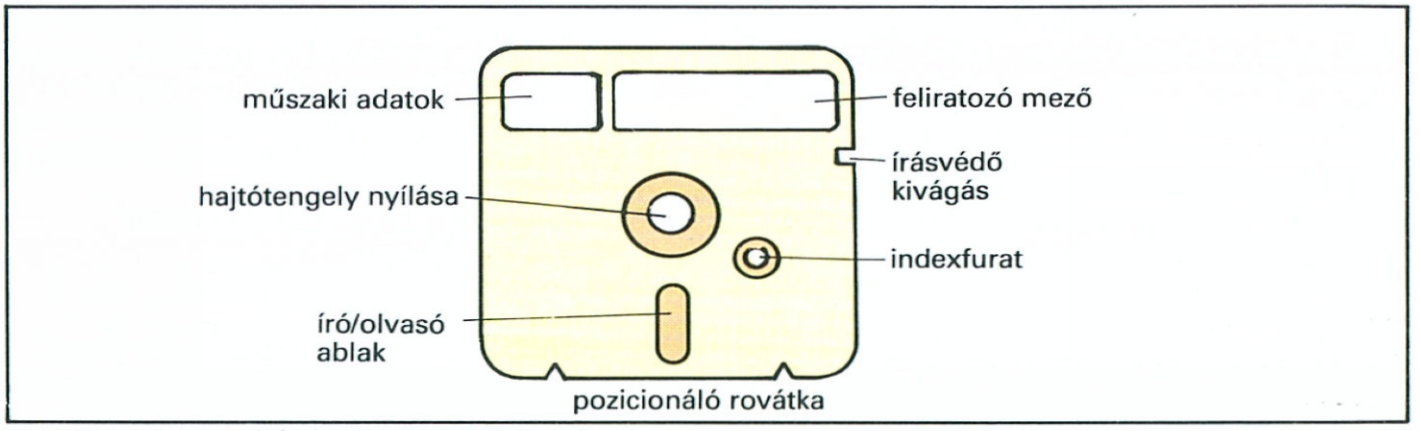
Példa: a modern 3½ collos merevlemezes tárolók max. 2 Gbyte tárolókapacitást is elérnek.

Mágnesszalag. A mágnesszalagos adattárolók a magnetofonhoz hasonló elven működnek. Az adathordozó 0,038 mm vastag 12,7 mm (½ inch) széles, 0,015 mm vastag mágnesezhető réteget hordozó műanyag szalag. A szalagok szokásos hosszúsága 730 m (2400 láb). A szalagokat egy cserélhető csévéről egy beépítettre csévélik, ezért a szalag végét elérve vissza kell csévélni. A szalag általában csak egyik irányban írható, de mindkét irányban olvasható. A meghajtó mechanizmus a szalagot 5 m/s sebességgel mozgatja a 9 egymás mellett elhelyezett író-/olvasófej előtt. Az információt 9 párhuzamos sáv (8 adat- és egy paritássáv) tárolja. Egyidejűleg egy byte írása vagy olvasása végezhető el.

Mivel a csévélés elindításakor és megállításakor nem történhet információátvitel, az adatokat helyközzel *(blokk-közzel)* elválasztott blokkokban tárolják. Az adatátvitel mindig blokkonként megy végbe.

Példa: Ennek a könyvnek a szövegoldalai kb. 7×10^8 jelet tartalmaznak, ezeket mintegy 5 m hosszú szalagon lehetne tárolni, és az olvasófejjel egy másodpercnél rövidebb idő alatt lehetne beolvasni.

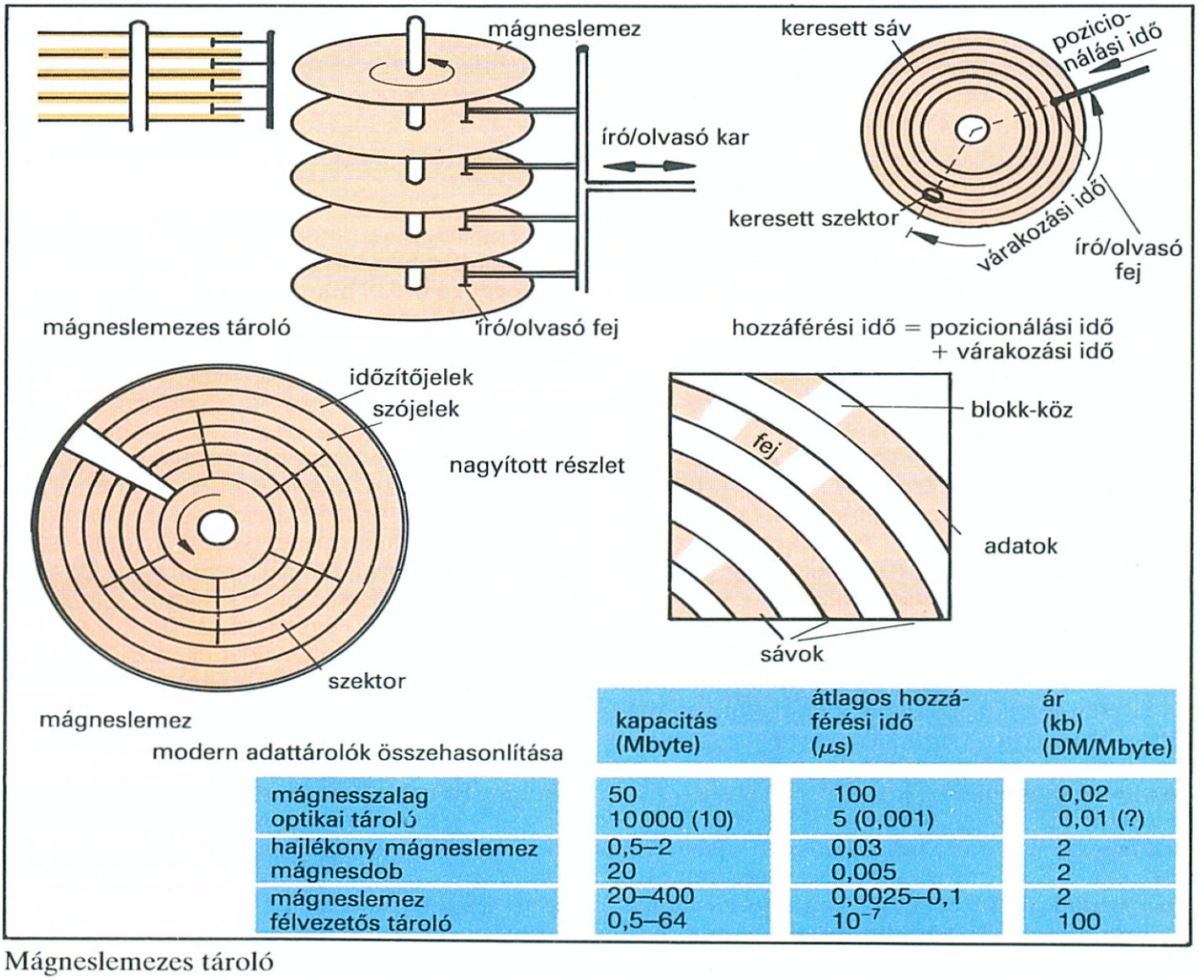
A szalagtovábbító mechanizmus működési módja miatt a szalag nem mindenhol feszes, hanem a pálya mentén az író-/olvasófej két oldalán egy-egy pufferhurkot alakítanak ki.



Hajlékonylemez (5½ collos floppy diszk)

típus		DD	DD	HD	HD	ED
oldalak száma		2	2	2	2	2
koercitív erő	Oe	300	300	600	600	600
sávok száma		2×40	2×80	2×80	2×80	2×80
sávsűrűség	TPI	48	135	96	135	135
szektorok száma		9	9	15	18	36
fordulatszám (percenként)		300	300	300	300	300
lemezméret	Zoll	5¼	3½	5¼	3½	3½
adatátviteli sebesség	KBd	250	250	500	500	500
tárolókapacitás (formázás nélkül)	Mbyte	0,5	1	1,6	2	4
tárolókapacitás (formázva)	Mbyte	0,36	0,72	1,2	1,44	2,88

A PC-khez használt hajlékonylemezek műszaki adatai



modern adattárolók összehasonlítása

	kapacitás (Mbyte)	átlagos hozzáférési idő (μs)	ár (kb) (DM/Mbyte)
mágnesszalag	50	100	0,02
optikai tároló	10000 (10)	5 (0,001)	0,01 (?)
hajlékony mágneslemez	0,5–2	0,03	2
mágnesdob	20	0,005	2
mágneslemez	20–400	0,0025–0,1	2
félvezetős tároló	0,5–64	10^{-7}	100

Mágneslemezes tároló

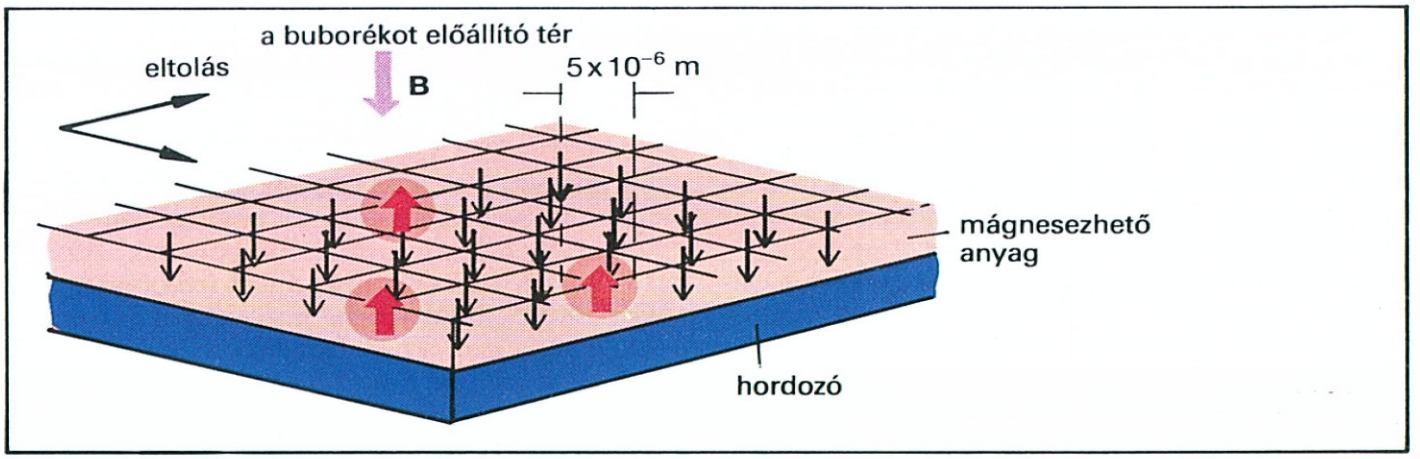
Az információ-hozzáférés szekvenciális. A hozzáférési idő ezért a perc nagyságrendet is eléri.
A mágnesszalagok különböző kivitelben olcsó tömegtárként használhatók (kapacitásuk 8 Gbyte is lehet). Elsősorban akkor alkalmazzák őket, amikor az adatok ugyanabban a sorrendben kerülnek felhasználásra, amilyen sorrendben rögzítésre kerültek.
A **kazettás szalagokat** általában kis számítógépekhez használják. Ezek legtöbbször magnetofonkazetták, és ritkán használnak egy sávnál többet.

A **mágneslemezes tárak** hanglemezre emlékeztető forgó merev lemezt tartalmaznak. A mágnesezhető réteg a 3½, 5¼, 8, 12 és 14 coll (angolul: *inch*) átmérőjű merev alumíniumhordozó két oldalát borítja. A cserélhető lemezkötegekben 20 lemez is foroghat azonos fordulatszámmal a közös tengely körül. Az adatok max. 1600 koncentrikus kör alakú és azonos tárolókapacitású sávon helyezkednek el, tehát a legbelső sáv tárolási sűrűsége a legnagyobb. A mágnesszalag blokkfelosztásához hasonlóan minden sávot szektorokra lehet felosztani. A tárolási kapacitás lemezfelületenként néhány Gbyte is lehet. Az egymással szemben elhelyezett író-/olvasófejek minden egyes sáv fölött szilárdan is beépíthetők. Ilyenkor az átlagos hozzáférési idő a lemez teljes körülfordulási idejének a fele, mintegy 2,5 ms. Azonban egy lemezköteg minden lemezfelületéhez gyakran csak egyetlen, pozicionálható író-/olvasófej tartozik. Ezek fésűhöz hasonló alakú tartószerkezethez vannak erősítve, amely a fejeket együtt, sugár irányban mozgatja. A közösen elért sávok együttesét *cilindernek* nevezzük. A hozzáférési idő ebben az esetben az író-/olvasófej megfelelő cilinderekre való pozicionálási idejének és annak az időtartamnak (a *várakozási időnek*) az összege, amelynek elteltével a keresett szektor az író-/olvasófej alatt megjelenik. A két időtartam összege átlagosan 18–24 ms. Legtöbbször az egy sávban tárolt összes adatok átvitelre kerülnek. A mozgó író-/olvasófejjel működő mágneslemez-kötegek tárolókapacitása a 400 Mbyte-ot is elérheti.
Mivel az író-/olvasófej és az adathordozó közötti távolság kb. 1/1000 mm, a lemezeket illetve a lemezkötegeket pormentesen lezárják.

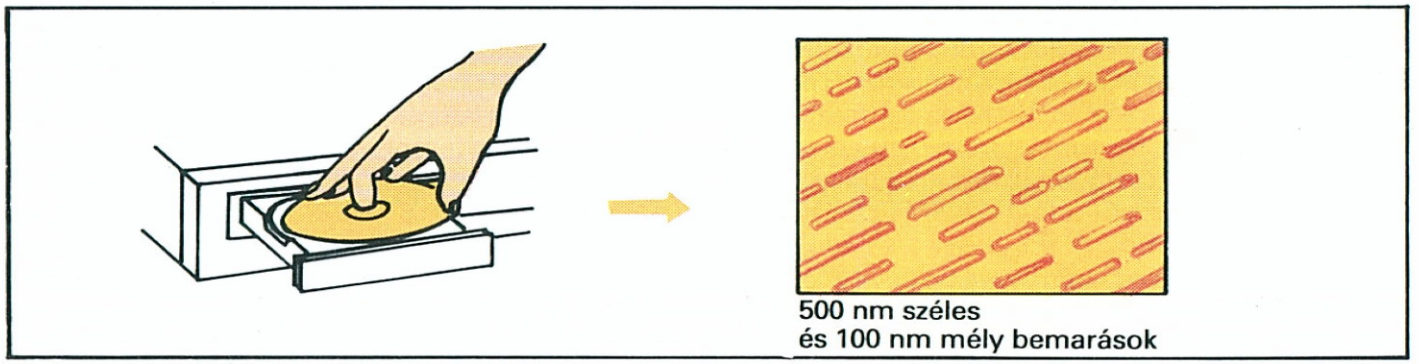
Merevlemeznek vagy **winchesternek** (angolul: *hard disk*) nevezzük a számítógép nem kivehető, pormentesen lezárt mágneslemezét. A merevlemez átmérője 5¼, 3½ vagy 2½ coll lehet, tárolókapacitása eléri a néhány Gbyte értéket, hozzáférési ideje pedig 10 és 80 ms közé esik.
A merevlemez nyugalmi állapotában az író-/olvasófej egy adott lemezterületre felfekszik. A forgáskor keletkező légpárna a fejet felemeli, így az átvitelre kész állapotban, mikrométeres távolságban lebeg a lemez felett.
Hajlékonylemeznek (angolul: *floppy disk* vagy *floppy*) nevezzük a korlátozott élettartamú, kivehető mágneslemezes tárat. Az átmérőértékek a következők: a normállemezé 8 coll (8″), a minilemezé 5¼ coll (5¼″), a mikrolemezé 3½ collt (3½″), a kompaktlemezé pedig 3 coll (3″). A mágnesezhető réteget hajlékony vagy merev műanyag korongra viszik fel, az így készült lemezt négyzet alakú karton- vagy műanyag borítóba helyezik. Az adatok átvitele a borítóba vágott író/olvasó ablakon, a meghajtás pedig percenként 360 fordulat sebességgel a hajtótengely nyílásán keresztül történik. A hajlékonylemez egyik oldala max. 96 koncentrikus sávot tartalmaz. A sávok (max. 64) szektorra oszlanak. Az újonnan használatba vett hajlékonylemezek szektorait az első használat előtt legtöbbször a számítógéppel alakítják ki **(formázás).** A szektorok viszonylagos helyzetét az indexfurat határozza meg. A *formázott* hajlékonylemezek borítóján szektoronként egy-egy furat található. A borító szélén egy kivágás szolgál az írásvédelemre. Ha az 5¼ collos lemezeken a kivágás le van zárva, akkor a lemezt csak olvasni lehet. A 3½ collos lemezek esetén ennek pontosan a fordítottja érvényes: az adatokat akkor lehet törölni vagy felülírni, ha a kivágás le van zárva. A hajlékonylemezek tárolókapacitása eléri a 4 Mbyte-ot, átlagos hozzáférési idejük kb. 100 ms. A mágnesezhető réteg felpárologtatásával valamint a lemezek előzetes formázásával és lézeres sávvezetéssel (floptical lemezek) a tárolókapacitás 21 Mbyte-ig növelhető.
Elsősorban PC-kben külső adattárolóként az adatrögzítés és a szövegfeldolgozás során kerülnek felhasználásra.
Előnyei: olcsó, nagy tárolási sűrűség, könnyű kezelhetőség. Hátrányai: viszonylag hosszú hozzáférési idő és viszonylag rövid élettartam.

Mágnesdobok
A mágnesezhető réteg a hossztengelye körül egyenletes sebességgel forgó henger palástján helyezkedik el. Az információt a henger tengelyére merőleges sávok tárolják, és a sávok saját címmel ellátott szektorokra tagolódnak. Minden sávhoz egy szilárdan beépített író-/olvasófej tartozik. Az átlagos elérési idő: 5 ms. A tárolási kapacitás felső határa kb. 20 Mbyte.
A mágnesdobos tárakat napjainkban már alig gyártják.

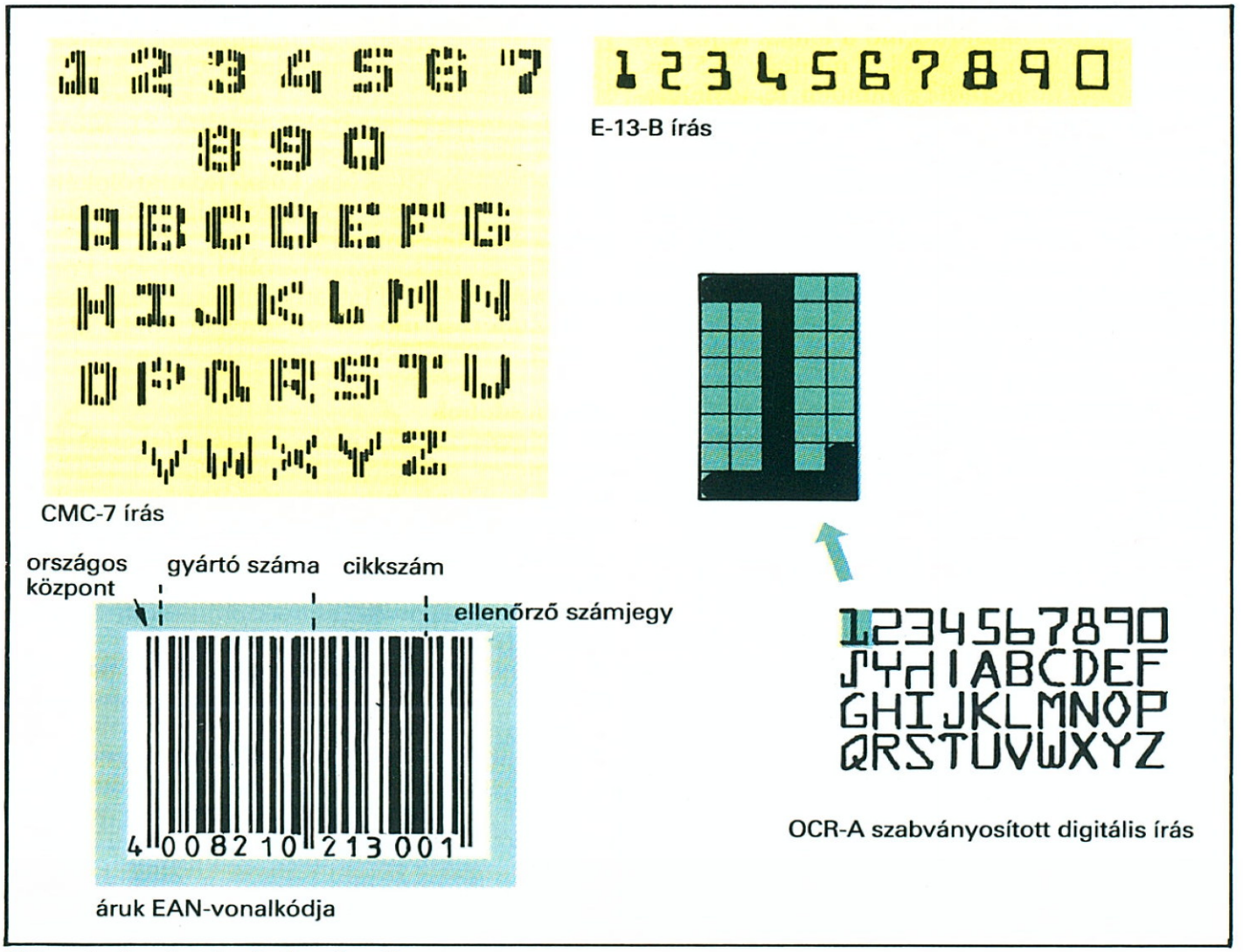
A **mágneskártyák** 10 cm × 40 cm nagyságú, kb. 0,1 cm vastag, egyik oldalukon mágnesezhető réteggel bevont, hajlékony műanyag kártyák, amelyeket kártyalerakóban helyeznek el. Innen egy szerkezet a kívánt kártyát kiválasztja és egy dobra feszíti. Az írás és az olvasás ugyanúgy történik, mint a mágnesszalagos tárak esetén. Egy kártya tárolókapacitása 200 Mbyte. A mágneskártyákat napjainkban már nem használják.



Mágnesbuborékos tároló (részlet)



A CD-lemez egy nem törölhető optikai adattároló



Géppel olvasható adathordozók

Mágnesbuborékos tároló. Ha egy jól körülhatárolt mágneses tér bizonyos anyagokra, pl. vékony, gránátszerű szilárdtestrétegre hat, akkor annak belsejében hengeres alakú mágneses tartomány, ún. *mágnesbuborék* keletkezik, amelynek átmérője kb. 0,001 mm. Csak két, egymással ellentétes mágnesezési irány lehetséges. Egy további mágneses tér segítségével ezek a mágnesbuborékok a mágneses térre merőleges síkban egyszerűen és gyorsan eltolhatók. A buborékos tároló cellákra oszlik. Az elfoglalt és a szabad pozíciók az 1 és a 0 bináris értékeknek felelnek meg. A tárolt adatok feszültségkimaradás esetén sem vesznek el. A mágneslemezes tárolókkal ellentétben a mágnesbuborékos tárolóknak nincs mechanikusan mozgó alkotórésze, ami gyors, kopásmentes hozzáférést tesz lehetővé.

A tárolási sűrűség: 10 Kbyte/mm^2 (kb. $6{,}5\times10^6$ bpi^2), a hozzáférési idő 0,5 ms.

A tárolóknak ez a típusa még nem terjedt el.

Optikai tárolók

Az adatokat lézersugár írja egy forgó optikai lemezre ill. olvassa a lemezről. Az optikai lemezt (angolul: *optical disk, laser disk*) mágnesezhető anyag vagy félvezető réteg vonja be, amelynek törésmutatója lézersugárral való megvilágítás hatására nagymértékben megváltozik. Egy 30 cm átmérőjű optikai lemez – szektorokra osztva – 50 000 sávot tartalmaz, ami több, mint 10 Gbyte tárolóterületnek felel meg. A tárolt adatok törölhetők. Ez egy olcsó, tömegméretekben használható, kis hozzáférési idejű tároló.

Termomágneses módszerrel az írás a következőképpen történik: keskeny lézersugárnyaláb felmelegít egy vékony, mágnesezhető réteget. Ezt a pozíciót egy nagyon gyenge mágneses tér felmágnesezi. A besugárzott hely rendkívül kicsi, ennek megfelelően magas az (elméletileg) elérhető tárolási sűrűség (kb. 5×10^8 bpi^2). Olvasáskor egy lineárisan polarizált fénysugár tapogatja le az adathordozót. A visszavert sugár polarizációsíkja az észlelt mágnesezési iránynak megfelelően megváltozik. Az új polarizációs irányt egy optikai cella meghatározza és annak megfelelő jelet továbbít. Egy teljes sáv átlagos hozzáférési ideje kb. 1 ms.

Holográfiás módszer. Az adatokat optikailag bistabil rétegek tárolják a lemez felületén holografikus minták formájában. A közepes hozzáférési idő a μs-tartományba esik.

A napjainkban elterjedt **kompakt optikai lemezek** (angolul: **C***ompact* **D***isk,* **CD**) 600–800 Mbyte tárolókapacitású optikai tárolók. Leolvasásukhoz CD-ROM meghajtó szükséges.

Számítógépekkel olvasható további adathordozók

A **mágnestintás karakterfelismerő** (angolul: **M***agnetic* **I***nk* **C***haracter* **R***ecognition,* **MICR**) adathordozóján a betűket és a számokat ferrittartalmú mágnesezhető tintával vagy írószalaggal az ember számára olvasható alakban ábrázolják. Az adatokat mágneses karakterolvasók tapogatják le és belső gépi ábrázolású alakra fordítják. Kétféle mágneses írásmód használatos:

Az **E-13-B írás** minden jelet úgy jellemez, hogy a jelkészlet egyes elemeinek különböző részei különböző erősségűek.

A **CMC-7 írás** jelkészletének minden eleme 7 függőleges, egymástól különböző távolságban levő vonalból áll.

A mágneses karakterolvasó az adathordozót a magnetofonszalaghoz hasonlóan olvasófejek előtt mozgatja. Az ennek hatására indukálódott jelek a számítógépbe jutnak, amely dekódolja és kiértékeli az adatokat. A mágneses karakterírókban írógépbillentyűzet szolgál az adatok bevitelére.

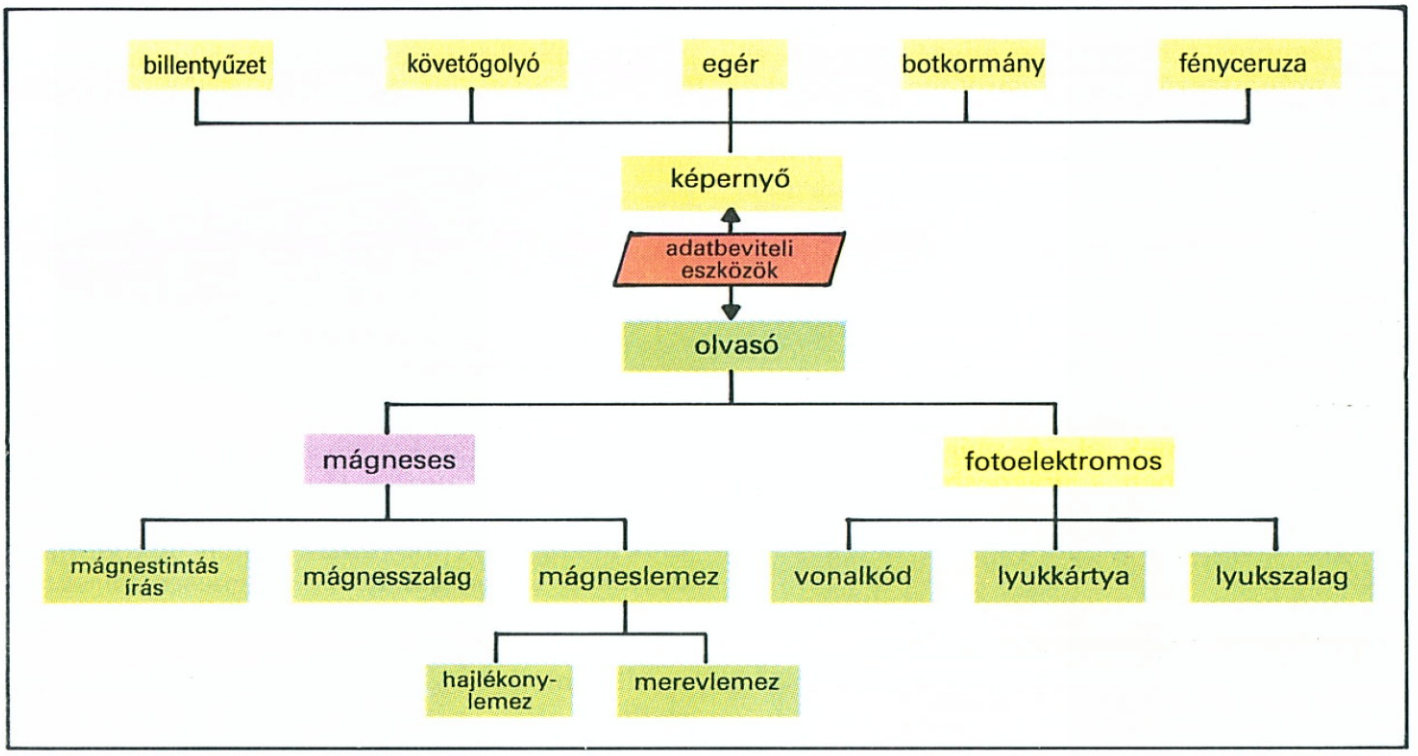
Alkalmazása: bankkiutalások, átutalások, receptek stb.

Az **optikai karakterfelismerők** (angolul: **O***ptical* **C***haracter* **R***ecognition,* **OCR**) – legtöbbször szabványosított – adathordozója nyomtatott vagy írott betűket tartalmaz, amelyek optikai úton letapogathatók és belső gépi ábrázolású adatokká alakíthatók. Elsősorban a – közvetlenül is jól olvasható – **OCR-B írást** alkalmazzák. A karakterolvasó előre megadott mezők x-szel történt megjelölésével kitöltött adathordozókat is képes felismerni *(űrlapbeolvasó).*

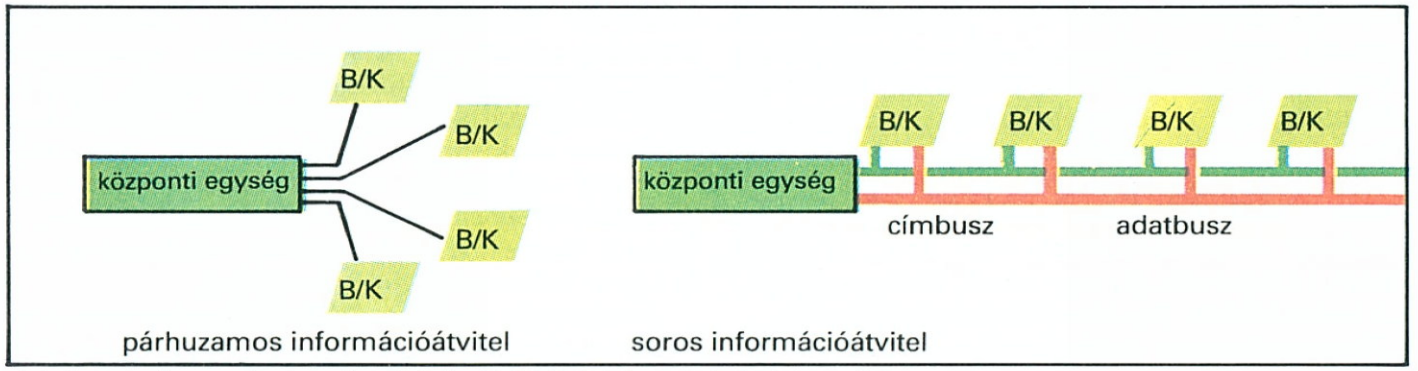
A **szkenner** vagy letapogató (angolul: *scanner*) olyan olvasókészülék, amely közönséges írógéppel vagy nyomtatott betűvel írt szöveg felismerésére használható. Az írott szöveget pontról pontra veszik fel, ezután egy program segítségével összehasonlítják a szövegmintát a számítógépben tárolt karakterkészlettel és megkísérlik a jelek felismerését.

Műanyag kártyák (pl. igazolványok vagy hitelkártyák) hátoldalán rögzített **mágnescsíkokon** – szemmel nem látható – adatok tárolhatók (ezek jellemzőit Németországban a DIN 9785 szabvány írja elő).

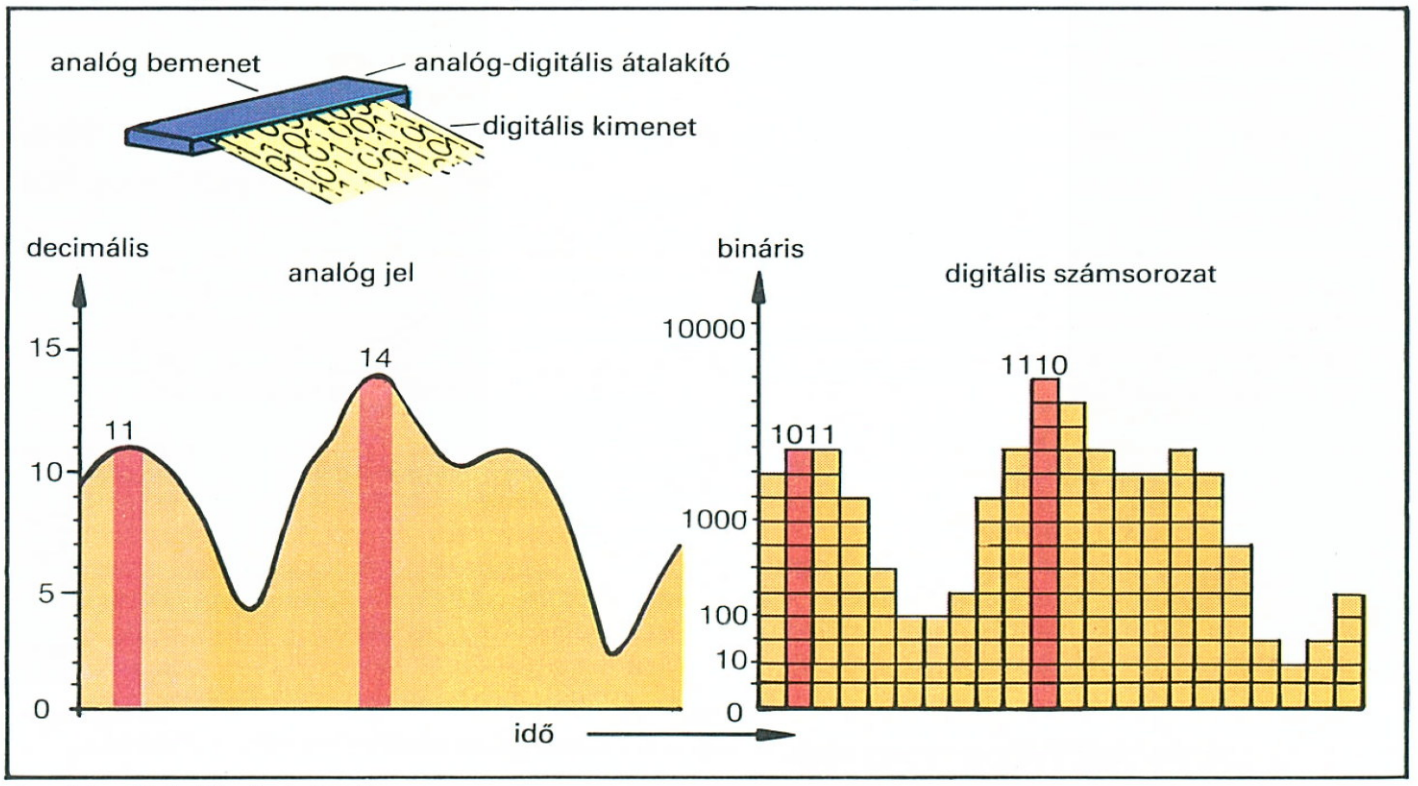
A **vonalkódok** (angolul: *bar code*) gépi olvasás számára alkalmas kódok, amelyek egymástól különböző távolságra lévő különböző erősségű csíkokból állnak. Az **OCR-A írást** és az **EAN-kódot** (angolul: **E***uropean* **A***rticle* **N***umber*) elsősorban a fogyasztási cikkeket gyártó ipar alkalmazza az árucikkek megjelölésére. Többek között a következő információkat kódolják: az áru származási országa, gyártó cég, árucikk neve stb. A pénztárban elhelyezett leolvasó készülék a vonalkódot fotoelektromos úton leolvassa és dekódolásra a számítógépbe továbbítja. Az árucikk árát a számítógép az árucikk számának a külön tárolt árlistával történő összehasonlítása útján állapítja meg és nyomtatja ki.



Fontosabb adatbeviteli készülékek



A központi egység és a be-/kiviteli (B/K) készülékek közötti kapcsolat



Analóg-digitális átalakítás

A számítógép központi egysége és a külvilág közötti kommunikációt a be-/kiviteli (angolul: *input/output, i/o*) készülékek valósítják meg, amelyek a számítógép perifériáihoz tartoznak. A számítógép fejlődésének korai szakaszában – az 1930-as években – erre a célra a technika más területeiről átvett készülékek, pl. szalaglyukasztók és írógépek szolgáltak. Napjainkban az ipar különlegesen erre a célra kiképzett vagy az új feladatokra áttervezett, nagyon gyors be-/kiviteli készülékeket alkalmaz. A közvetlen kommunikáció gyakorlatilag késleltetés nélkül lezajlik, a közvetett kommunikáció pedig a mágneslemezes, a hajlékonylemezes és a lyukkártyás adathordozókkal valósítható meg.

Az **adatbevitelt** vagy speciális adatbeviteli készüléken (mint pl. billentyűzet, mágneslemez-olvasó), vagy más számítógépekről távadatátvitel segítségével lehet megvalósítani.
Az adatok **kivitele** közvetlenül vagy közvetett módon speciális adatkiviteli eszközökön, pl. nyomtatón, képernyőn mehet végbe. Az adatok távadatátvitel felhasználásával más számítógépek tárolójába is juttathatók.

Az **illesztőegység** (angolul: *interface*) – szűkebb értelemben – a be/kiviteli egységek és a központi egység közötti átmenet. Az interfész az adatcsatornán belül elhelyezett, nagymértékben szabványosított, többszörösen dugaszolható csatlakozó (hardver illesztés).

A *V.24 interfész* az adatkommunikáció legszélesebb körben elterjedt illesztőegysége. Csatlakozója 50 pólusú, amely nyolc kábelcsoportot köt össze, pl. az adat-, az analóg-, az órajel- és a vezérlőkábeleket.

Az *X-interfészek* olyan szabványosított illesztőegységek, amelyeket a Datex-hálózatban történő adatátvitelre terveztek és az egyszerűsített V.24-szabványt követik.

A *Centronics-interfész* 8 bitet továbbít párhuzamosan max. 36 vezetéken, és elsősorban a PC és a nyomtató közötti átvitel céljaira szolgál.

A be-/kiviteli készülékek adatátviteli sebessége általában lényegesen kisebb, mint a központi egységé. Tehát gondoskodni kell arról, hogy a be-/kivitel a központi egységet ne hátráltassa. Egy központi egység az adatok közbenső tárolásával általában több be-/kiviteli készüléket is kiszolgál.

A **párhuzamos adatátvitel** során az adatátvitelben részt vevő minden készülék saját, független adatátviteli csatornával rendelkezik. A központi egység egyszerre több be-/kiviteli készüléket vezérel. Az adatok kódolása és dekódolása szinkronban van. Ez a módszer drága, mert az egyes adatcsatornák csak ritkán és egyenlőtlenül vannak kihasználva.
Soros adatátvitel. A központi egységet minden perifériával egy közös vezeték, az ún. **busz (sín)** köt össze; minden készüléknek saját címe van. Az egyes készülékekkel végbemenő adatátvitel egymás után (sorosan) történik. A busz vezérlését általában külön mikroprocesszor látja el. Az adatvezetékek gyakran vannak használatban, ezért várakozási idők léphetnek fel. Távadatátvitel céljaira elsősorban a soros átvitel jön szóba, mert az adatkódolás és -dekódolás szinkronizálása nagyon nagy ráfordítást igényel.

Megszakítás (angolul: *interrupt*). Mivel az adat be- és kivitel a központi egység idejének csak töredékét veszi igénybe, a be-/kivitel a központi egység pillanatnyi tevékenységét egy jel segítségével szakítja meg. A központi egység megvizsgálja a megszakítójel prioritását, és ennek megfelelően aktivizálja a be-/kimenetet. A megszakított folyamat egy közbenső tárolóban marad, és az adatátvitel befejezése után folytatódik. Ez a módszer drága, de hatékonyabb, mint valamennyi be-/kiviteli készülék ciklikus lekérdezése.

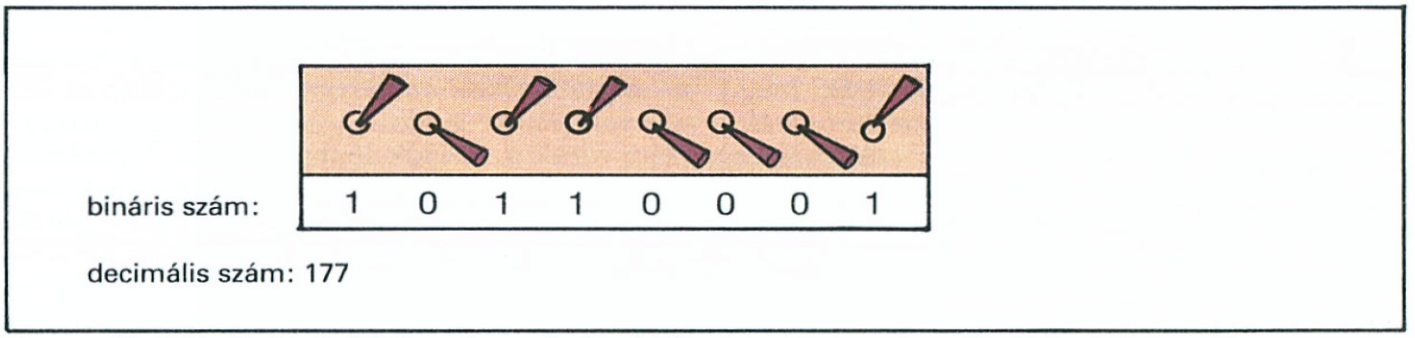
Átalakító vagy **konverter** (angolul: *converter*). A számítógép központi egysége csak digitális jeleket dolgoz fel. Ha a bemeneti jelek analóg formában állnak rendelkezésre, akkor ezeket digitális jelekké kell átkonvertálni, ha pedig a kimeneten analóg jelekre van szükség, akkor ennek fordítottja szükséges, vagyis a központi egység jeleit a megfelelő analóg jelekké kell alakítani. Az átalakító gyakran a megfelelő periféria szerves része.

Az **analóg-digitális konverter** (AD-konverter) a bemeneti készülék analóg jeleit a központi egységben elvégzendő további feldolgozás számára digitális jelekké alakítja. Az átalakítás analóg összehasonlítójelek segítségével történik. Az AD-konverter meghatározza pl. az időben változó jelek azonos időintervallumokra osztott szakaszainak amplitúdóját (a jel letapogatása). Az így létrejött számok az átalakítás után bináris számjegysorozat alakjában jelennek meg.

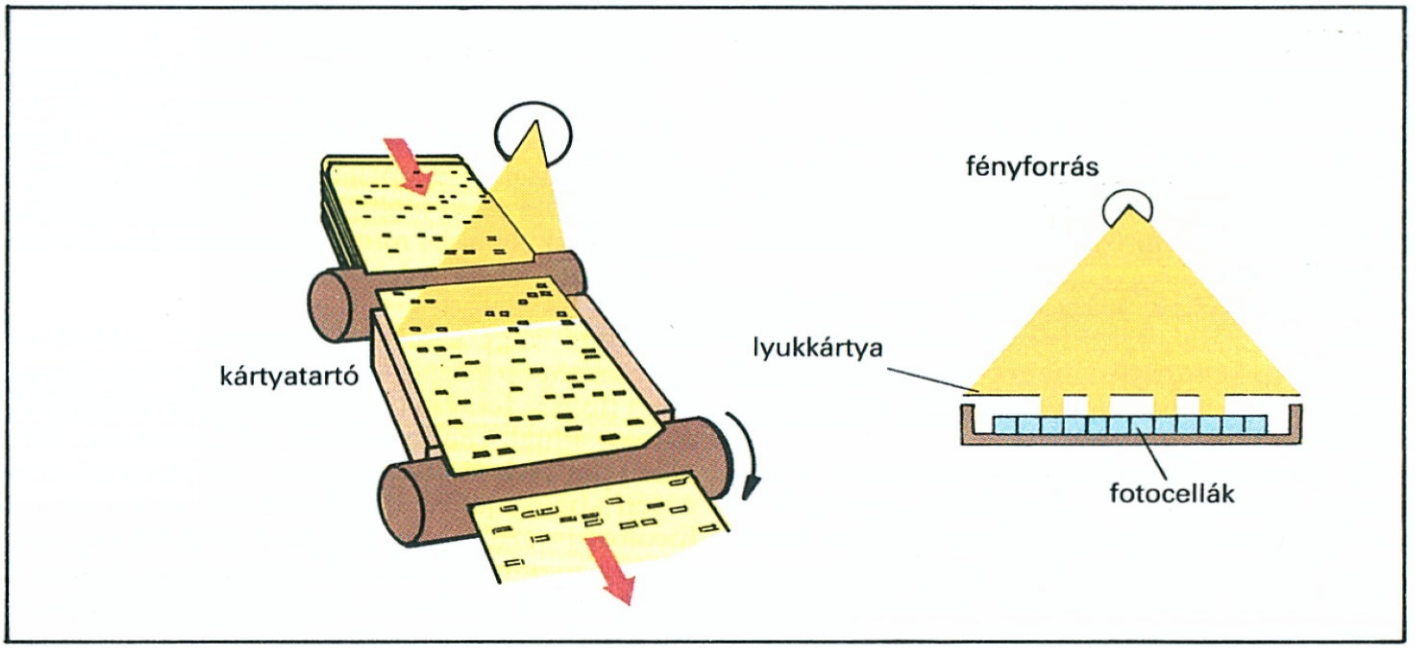
Példa: Egy újság-raszterkép szürkeségi fokozatainak megállapítása (két helykoordináta és pontonként 64 fényerőfokozat felhasználásával) a fénykép analóg-digitális átalakítását jelenti.

Digitális-analóg konverterek (DA-konverterek) a központi egység kimenetén megjelenő digitális jeleket analóg jelekké alakítják át. Az egyes bitek áramforrásokat kapcsolnak be, amelyek a kimeneten összegződnek. A lépcsőzetes jeleket egy speciális integrált áramkör kisimítja, és villamos feszültséggé alakítja át.

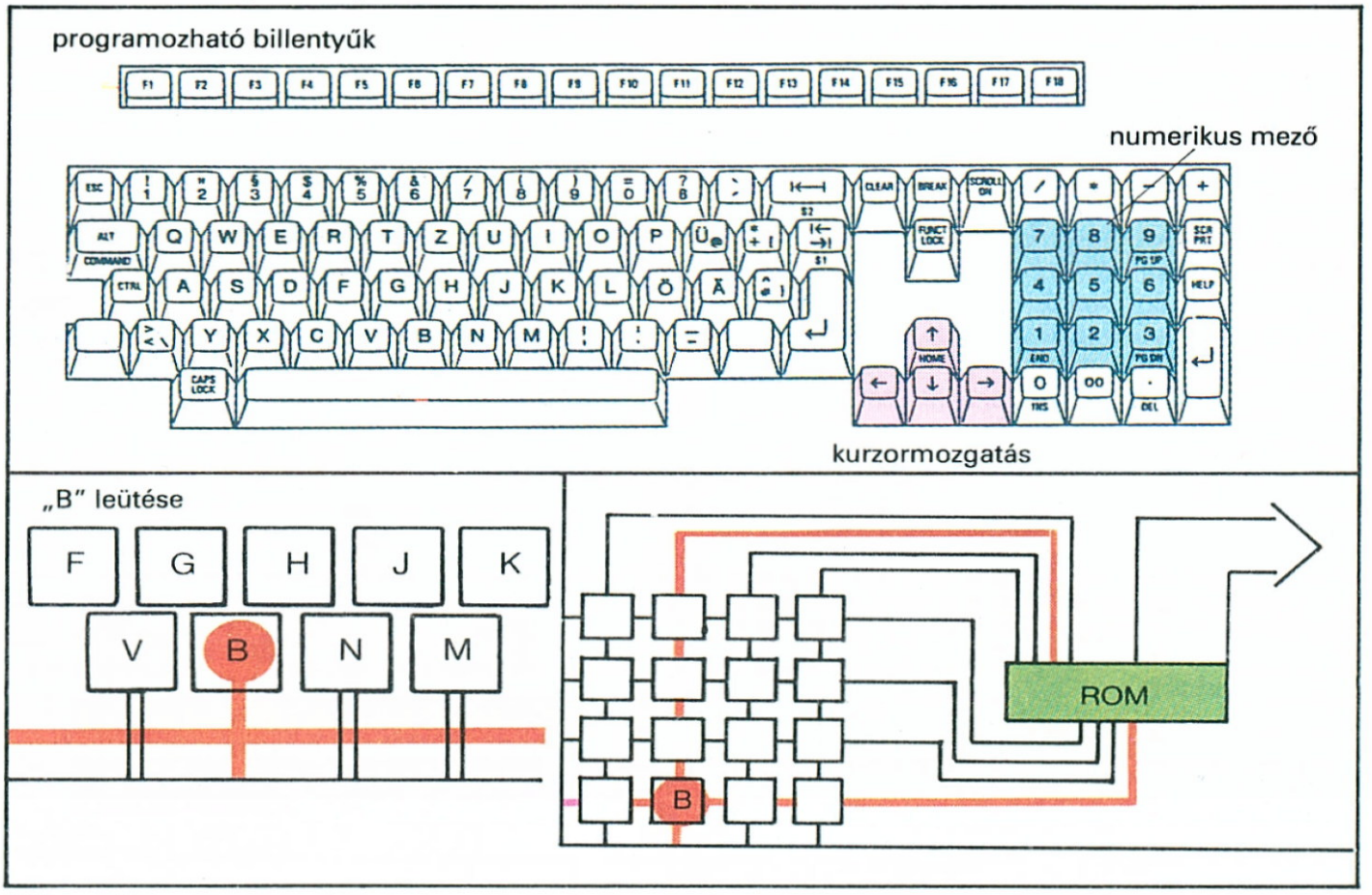
Az átalakítás általában információveszteséggel jár, mert az analóg érték gyakran két szomszédos digitális érték közé esik.



Kapcsolótábláról történő közvetlen adatbevitel



Fotoelektromos lyukkártyaolvasó



Billentyűzet

Az adatbevitelhez különböző periferiális eszközök állnak rendelkezésre.

Kapcsolótábla
A digitális adatok egyedi kapcsolók vagy dugaszolható csatlakozások sorozatával ábrázolhatók. Ezt a módszert a kézi kapcsolású telefonközpontoktól vették át, de ma már ritkán használják.

Billentyűzet
Ez a leginkább elterjedt készülék a közvetlen adatbevitel számára. Erre a célra egy számjegyekkel és különleges jelekkel, gyakran programozható billentyűkkel kibővített írógép-billentyűzetet használnak. A bevitt jeleket általában egy párhuzamosan kapcsolt képernyő azonnal megjeleníti. A képernyőn villogó szimbólum (angolul: *cursor*) jelzi a következő írásjel pozícióját.

A billentyűzet jeleinek manapság használt különös elrendezése azzal a céllal keletkezett, hogy az írás sebessége csökkenjen; száz évvel ezelőtt ugyanis a mechanikus leütőszerkezet még nem tudta az írás sebességét követni. Ez az indoklás manapság már érvényét vesztette, az új típusú, optimális jelelrendezésű, ergonómiai szempontból helyes billentyűzetek ennek ellenére nem tudnak elterjedni.

Képernyő
A képernyő (angolul: *monitor*) elsősorban adatkiviteli készülék, azonban **fényceruza** segítségével közvetlenül a képernyőről is lehet a központi egységhez adatokat továbbítani.

Ha a képernyő felületét a fényceruza fényérzékeny hegyével megérintjük, a pontok és vonalak mint adatok átvihetők, valamint egyes helyzetek megjelölhetők. Elsősorban a számítógépes tervezésnél, CAD, (184. o.) használják.

Maszkgeneráló programok segítségével a képernyő egyes tartományai úgy megvilágíthatók, hogy az adatbevitel csak a maszkon belül legyen lehetséges. Ez a módszer egyszerűsíti és biztonságossá teszi pl. az űrlapok feldolgozását.

Ahhoz, hogy a kurzort (helyőrt) a képernyő kívánt helyére irányítsuk, négy darab, nyilakkal jelölt billentyű áll rendelkezésre. Ezenkívül más billentyűk is vannak, amelyekkel a kurzor pl. a sor végére, vagy a következő oldalra gyorsan átvihető.

A kurzor mozgatását néhány, a számítógéphez kábellel csatlakozó segédberendezés is megkönnyíti:

A **botkormány** (angolul: *joystick*) egy rögzített pont körül minden irányban kitéríthető. A mozgatás irányát és időtartamát a számítógéphez digitális vagy analóg jelek továbbítják, ennek hatására a kurzor vagy valamilyen más képelem a képernyő megfelelő pozíciójába helyeződik át. A botkormányba épített nyomógombokkal a kívánt pozícióban akciókat lehet kiváltani.

Az **egér** (angolul: *mouse*) olyan adatbeviteli készülék, amelynek felső oldalán nyomógombok, alsó oldalán pedig egy mozgatható golyó helyezkedik el. A golyó elfordulását egy sima alátéten (angolul: *mouse pad*) szenzorok érzékelik és kábelen a számítógéphez továbbítják; ennek hatására a képernyőn egy grafikus szimbólum a megfelelő irányban elmozdul. A szimbólum pozíciója szerint az egér egyik nyomógombjának rövid lenyomása, rákattintás *(clicking)* az aktuális programot aktivizálja.

A **követőgolyó** (angolul; *track ball, joyball*) hátára fordított egérhez hasonlít. A készülék burkolatának felső nyílásán keresztül a golyó hozzáférhető, és közvetlenül kézzel szabadon mozgatható. Előnye: a követőgolyót a billentyűzetben is el lehet helyezni (pl. az Apple számítógépek esetén).

Adatolvasó készülékek
A legtöbb adathordozó, pl. a lyukkártyák, a mágnesszalagok, és a mágneslemezek átmeneti vagy állandó tárolásra szolgáló adathordozók. Igény szerint az adatokat különleges olvasókészülékek olvassák be és továbbítják a központi egységhez. Az adatok általában az adathordozón maradnak, tehát ez egy adatmásolási folyamat. A **lyukkártya-** ill. a **lyukszalagolvasók** a lyukakkal kódolt adatokat villamos vagy fotoelektromos módszerrel olvassák be, átalakítják villamos impulzusokká, amelyek dekódolóberendezésen át a központi egységbe jutnak.

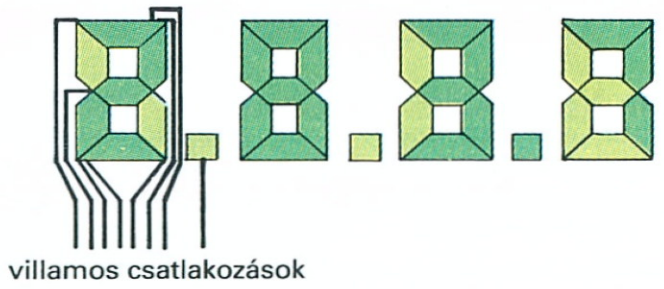
A lyukkártyák beolvasása kártyakötegenként történik, és a beolvasás sebessége a 30 kártya/s értéket is elérheti. Hátránya: a kártyák sorrendjét nem szabad felcserélni.

A lyukszalagolvasók másodpercenként max. 2000 jelet olvashatnak be. Hátránya: a szalagok könnyen megsérülnek vagy a szalagolvasóban begyűrődnek.

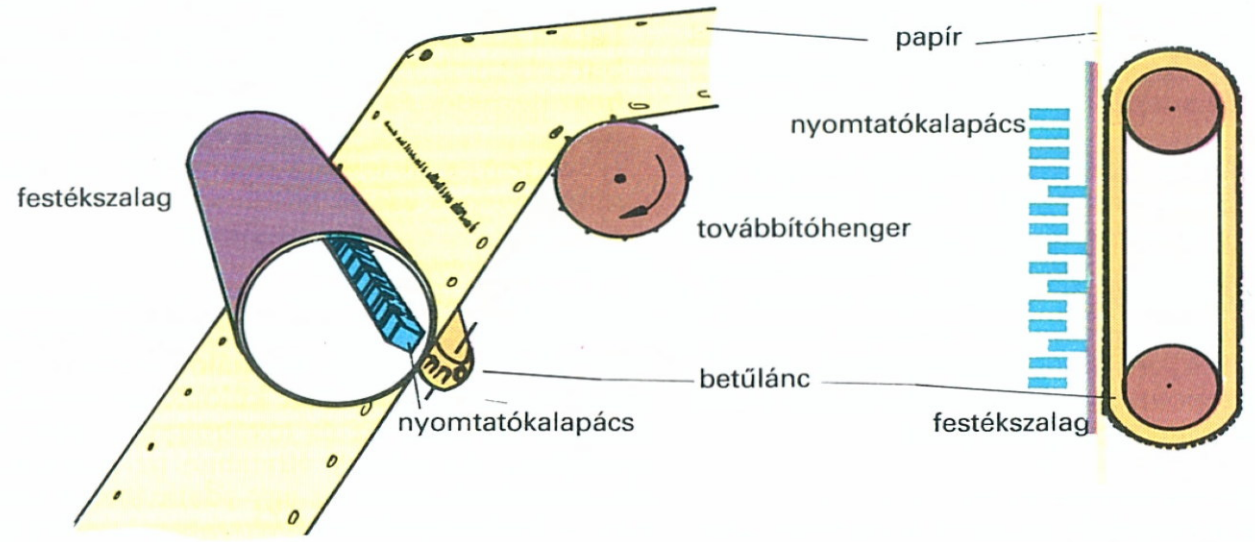
Időközben mindkét mechanikus adathordozót kiszorították a mágneses elven működő adathordozók.

A **normáltintás, a mágnestintás, a vonalkódos** és az **űrlapbeolvasók** (117. o.) az adatokat szabványos nyomtatványokról olvassák be. Ezeket elsősorban a kereskedelemben és kérdőívek kiértékelése során alkalmazzák.

A **szkenner** (optikai letapogató) a papíron megjelenő tetszőleges szöveget és képet digitális jelsorozattá alakítja, amely a számítógépben tovább feldolgozható. A papíron lévő optikai információt nagyszámú fotoelemből álló diódasor tapogatja le. Az eközben keletkező világos-sötét jeleket digitális jelekké alakítják. Az emberi kézírás olvasására azonban ezt a készüléket még nem lehet megbízhatóan felhasználni.

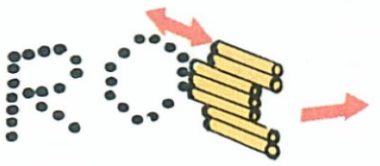


Közvetlen kijelzés folyadékkristályokkal

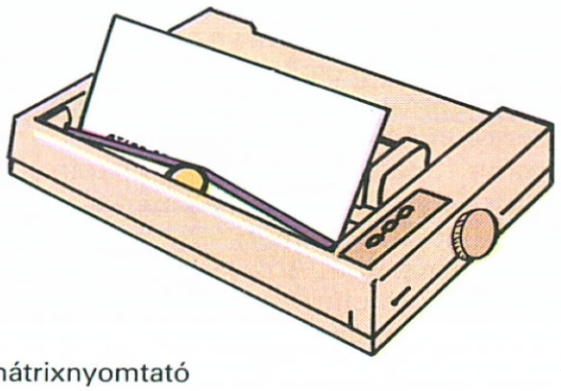


betűláncos nyomtató

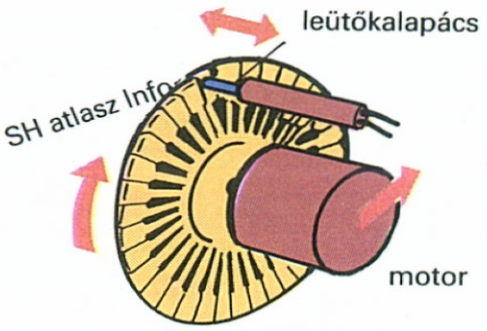
5×7-es nyomtató mátrix



tűk a mátrixnyomtatófejben (5×7)



mátrixnyomtató



margarétafej

Különböző nyomtatók

Az adatkivitelre szolgáló különféle perifériák száma igen nagy. Sok készülék az adatokat normál szöveggel jeleníti meg. Mivel a központi egység adatátviteli sebessége mindig nagyobb, mint az egyes adatkiviteli berendezéseké, ezért az adatok először egy közbenső tárba jutnak, és innen kerülnek a perifériára. Számos adatkiviteli készülék (pl. a mátrixnyomtató) vezérléséhez saját mikroprocesszor szükséges.
Számos kiviteli készülék digitális jeleket dolgoz fel, mások, mint pl. az elektronikus hangszerek, analóg bemenőjeleket igényelnek.

Digitális kijelző
Zuse Z2 típusú számítógépe az adatokat már 1938-ban közvetlenül binárisan, lámpasorral jelezte ki. A modern lumineszkáló diódák és folyadékkristályok az eredményt matematikai jelekkel, dekadikus számjegyekkel és betűkkel jelenítik meg.

Képernyő
A **képernyő,** más néven **monitor (display)** az ember és a számítógép közötti legfontosabb kommunikációs eszköz. Az 1940-es évek óta használják.
A közönséges monitor a televízió képernyőjének elve szerint működik, és fekete-zöld, feketenarancssárga, fekete-fehér vagy színes változatban kerül forgalomba. Mivel a képernyő az adatokat különálló képpontokból állítja össze és (mechanikus alkatrészek nélkül) nagy átviteli sebességgel dolgozik, ezért különösen alkalmas a közvetlen adatmegjelenítésre. Azonban az újonnan érkező adatok az előzőeket kitörlik. A képernyő egyes képpontjai a **képelemek, képpontok** vagy **pixelek** (angolul: **p***icture* **el***ement*).

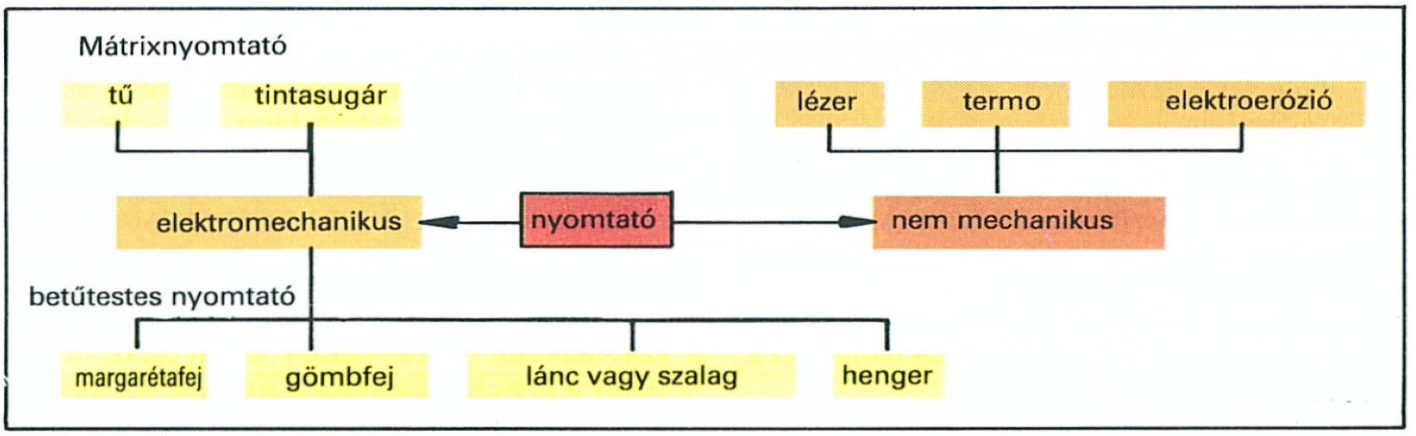
Az **alfanumerikus képernyők** kb. 65 000 képpontból álló rasztert használnak. Általában 24 sorban max. 80 betűt képesek megjeleníteni. Másodpercenként 70 képváltás vibrálásmentes képet biztosít.
A **grafikus képernyők** 16 millió képpontot is ábrázolhatnak. Gyakran speciális adattárolóval vannak felszerelve, amely a képeket automatikusan veszi fel, és azokat a filmekhez hasonlóan mozgóképekként jeleníti meg. Ilyen pl. az építészetben használt a „walk-through" ábrázolás.
A **vektorképernyőkhöz** nem alkalmaznak pontrasztert. Az elektronsugár a képkörvonalak mentén történő folytonos mozgatásával ábrázolja a képeket.
A **lapos képernyők** folyadékkristályos – (angolul: **l***iquid* **c***rystal* **d***isplay,* **LCD**) –, gázplazmás vagy elektronlumineszcens megjelenítőkkel működnek. Mint raszterképernyők 4096 × 4096 képelemet tartalmazhatnak. Csekély beépítési mélységük miatt különösen a hordozható (pl. laptop) készülékekben alkalmazzák őket. Kivétel: (a nagy energiafogyasztás miatt) a plazmaképernyő.

Írógépek
A különálló betűkarokból felépülő mechanikus szerkezetük miatt az írógépek adatátviteli sebessége kezdetben viszonylag kicsi volt. Az 1962-ben bevezetett *gömbfejes írógépben* a betűk egy mozgatható félgömb felületén helyezkedtek el. Ez a kiviteli sebességet 15 jel/s értékre duplázta. A gömbfejes írógépben nyugvó kocsi zavarmentes papírtovábbítást tett lehetővé. Ennél is gyorsabb a *margarétafej,* amelyben kalapács üti le a kör alakban perifériálisan elhelyezett, forgó betűket a festékszalagra és a papírra; az írás sebessége eléri a 60 jel/s értéket. A nyomtatókkal összehasonlítva azonban az írógép lassú adatkiviteli eszköz.

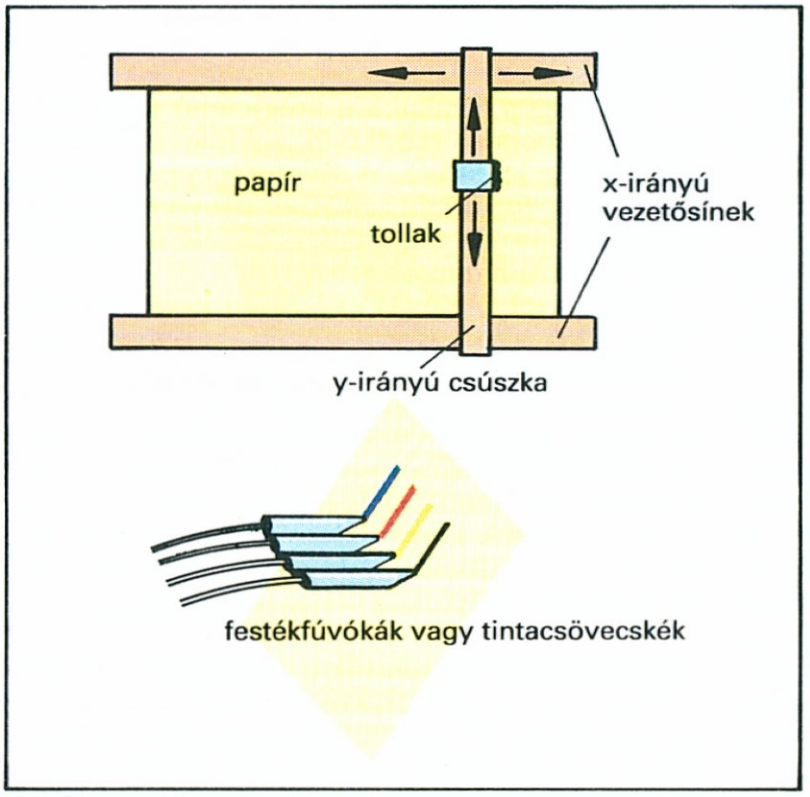
Nyomtatók
A számítógép kimenetén megjelenő nagy adatátviteli sebesség rendkívül sokféle nyomtatási módot tesz lehetővé. A számítógéppel vezérelt nyomtatók forradalmasították a nyomdatechnikát. Az adatokat alapvetően kétféle módszerrel lehet papírra nyomtatni:
– Minden jel egységes egészként kerül nyomtatásra, tehát a nyomtatás egyedi, korlátozott számú betűformával történik.
– Minden jel egy mátrix egyes pontjaiból tevődik össze, amelyet minden alkalommal egy raszter pontjaiból újra össze kell állítani. Ezzel a módszerrel tetszőleges jelet ki lehet nyomtatni. Ha a pontok kicsik és sűrűn követik egymást, akkor nagy felbontású szöveges vagy grafikus ábrázoláskor folytonos vonal jelenik meg.

A **mechanikus nyomtatók** a betűtestet, a festékszalagot és a papírt elektromechanikusan préselik egymásra. A nyomtatási sebességet a súrlódás és a mechanikai tehetetlenség korlátozza.
A **nem mechanikus nyomtatókkal,** mint pl. a lézernyomtatókkal lényegesen nagyobb nyomtatási sebesség érhető el.
Mindkét módszerrel közel azonos nyomtatási minőséget lehet elérni.

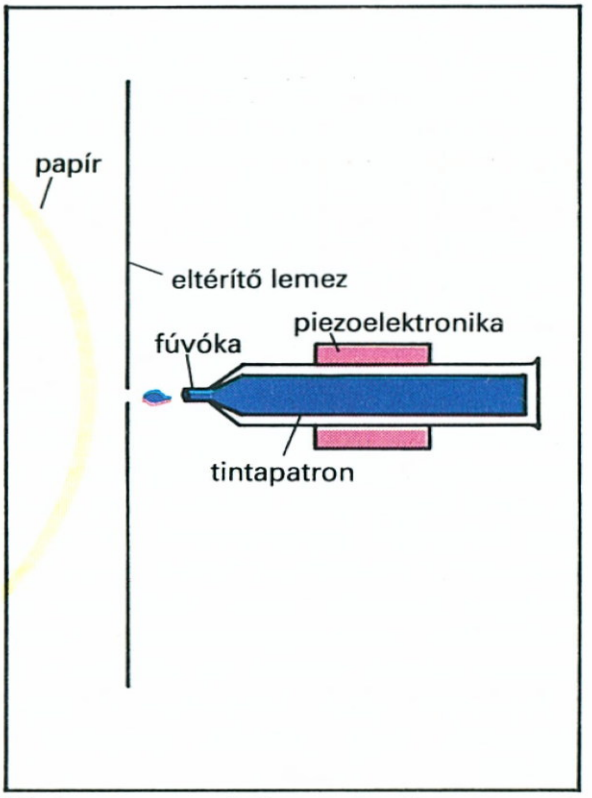
Betűtestes nyomtatók. A legegyszerűbb ilyen nyomtató az írógép (lásd fent). A **sornyomtatókban** egy sor minden egyes (összesen általában 120) betűpozíciójához külön betűkészlet tartozik, amelyek egymással párhuzamosan, nyomtatóhengeren vannak elrendezve. A megfelelő pillanatban kalapácsok ütik a papírt az egyes betűtestek felé. Kinyomtatott soronként a henger egy fordulatot végez. A „végtelenített" papír leporelló formában áll rendelkezésre, amelynek két szélén a perforáció pontos papírtovábbítást tesz lehetővé. Különösen gyors nyomtatás érhető el az acélszalagos nyomtatókkal, amelyeknél a betűket az acélszalagba marták. A szalag egyenletesen mozog a nyomtatókalapács előtt. A nyomtatás gyakran „röptében" történik, vagyis a betűtestek és a papír nyomtatás közben folyamatosan mozgásban van. A nyomtatási sebesség 50 sor/s sebességet is elérhet.



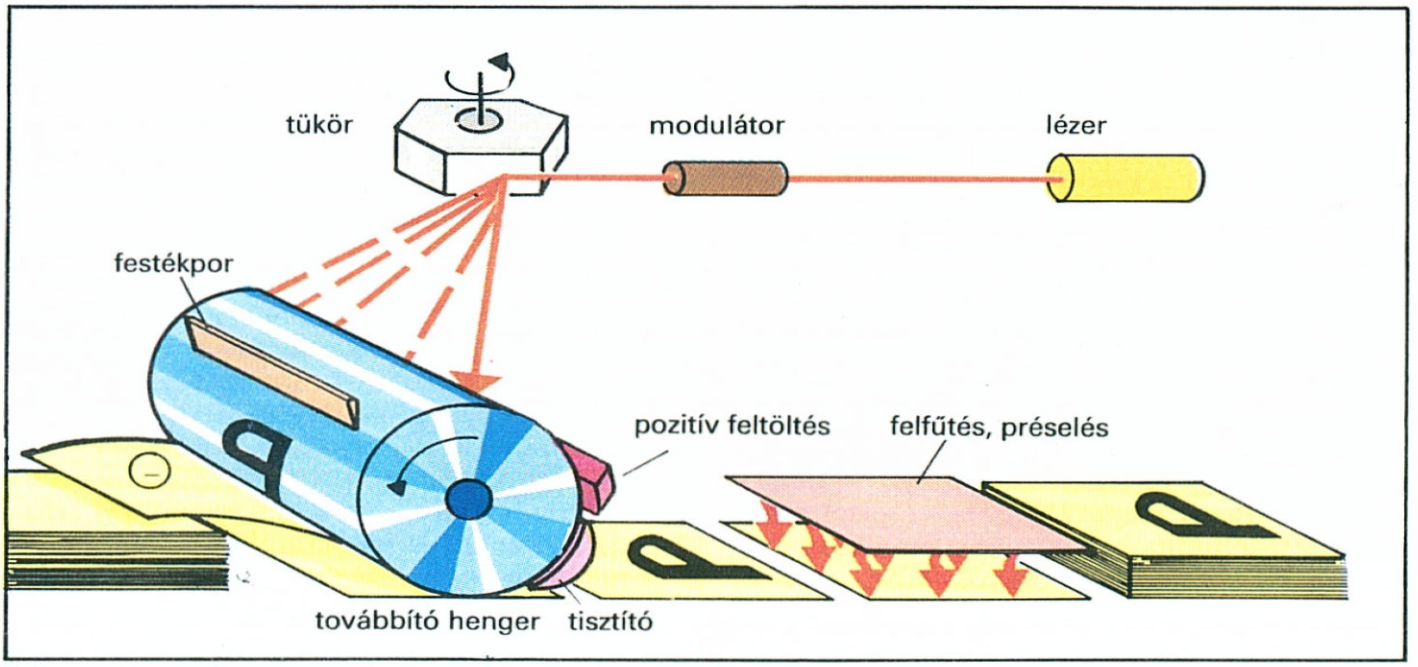
Különböző nyomtatótípusok



Kétdimenziós rajzgép



Tintasugaras mátrixnyomtató



Lézernyomtató

Mátrixnyomtatók

Tűs nyomtatók. Az ebben a nyomtatótípusban található vékony volframtűk a papír és a nyomtatófej között elhelyezett festékszalagra ütnek, és a papírra pontonként nyomtatnak. Legtöbbször 24 tű helyezkedik el egymás felett egy oszlopban. Az egyes tűk mozgását mikroprocesszor vezérli úgy, hogy a nyomtatásban a kívánt betű jelenjen meg. A tűmozgatást oszlopról oszlopra apró elektromágnesek végzik. A mikroprocesszor megfelelő programozásával tetszőleges jel (vastag betű vagy aláhúzott betű) nyomtatása is lehetséges. Ehhez a tűk számát megfelelően kell megválasztani, pl. a japán íráshoz 24 × 24 tű szükséges. Raszterképek és rajzok szintén ábrázolhatók. Többszörös, kissé eltolt nyomtatás elmossa a rasztereket, az egyes pontok összemosódnak, és a nyomtatás minősége elérheti a margarétafejes nyomtatók minőségét. A margarétafejes nyomtatók mellett csak a tűs nyomtatók képesek arra, hogy átütéssel több példányban nyomtassanak.

Többszínű festékszalagokkal (fekete, kék, sárga, vörös) és ismételt leütéssel színes rasznernyomás is készíthető.

A tűs nyomtatókkal max. 600 jel/s nyomtatási sebesség érhető el.

A **tintasugaras nyomtatók** érintés nélküli mátrixnyomtatók. Elektrosztatikus vagy piezoelektromos úton apró tintacseppeket visznek a tintatartályból közvetlenül a papírra. Ez a módszer különösen színes nyomtatásra alkalmas. A tintasugaras nyomtatók másodpercenként 4000 tintacseppet is kibocsáthatnak és elérhetik a 300 jel/s nyomtatási sebességet. Halkan és súrlódásmentesen működnek.

Az 1970-es évek közepe óta a **lézernyomtatók** határozzák meg a mátrixnyomtatók további fejlődését. A másológéphez hasonlóan működnek: egy számítógéppel vezérelt kis lézer nagyon szűk sugárnyalábot bocsát ki és azt egy tükörről visszaverődve pontról pontra a pozitív villamos töltésű, fóliával bevont nyomóhenger felületére vetíti. A felületen minden fényimpulzus egy villamosan semleges látens képpontot hoz létre. Ha a henger „írva" van, akkor pozitív töltésű, finom eloszlású festékpor (angolul: *toner*) szórja be. A festékpor csak a villamosan semleges képpontokon tapad meg. Ha ezután negatív töltésű papírlap simul a hengerre, akkor a porrészecskék a papírra tapadnak, amelyen hő és nyomás hatására rögződnek. A hangtalanul működő lézernyomtató oldalanként max. 8 millió képpontot ír és nyomtatási sebessége másodpercenként a 10 oldalt is elérheti. A nagyon finom beosztású raszter miatt a nyomtatás nagyon jó minőségű.

A **hőnyomtatók,** mint pl. a faxkészülékek, különleges hőérzékeny papírt használnak. Pontmátrixukat apró villamos fűtőlemezkék alkotják.

A **rajzgépek** vagy **plotterek** (angolul: *to plot*) egy- vagy többszínű grafikus ábrák rajzolására alkalmas készülékek. Ezekben léptetőmotorok egy csúszkát mozgatnak, amelybe (a papír helyzetétől függően vízszintesen vagy függőlegesen) több színes tollat lehet elhelyezni. A tollakat relék nyomják a papírra.

Ha a lépéstávolság a vonalszélesség kb. 1/10 részénél kisebb, akkor a rajzolt görbe lépcsőzetes szerkezete szabad szemmel nem látható.

Az **íróhengeres** és **dobos rajzgépekben** olyan végtelen papírszalagot alkalmaznak, amely a szélein levő perforáció segítségével továbbítható. A csúszka csak az egyik tengely mentén, a papírtovábbítás irányára merőlegesen mozog. Elsősorban időfüggő folyamatok ábrázolására és nagyon nagyméretű tervrajzok készítésére használják.

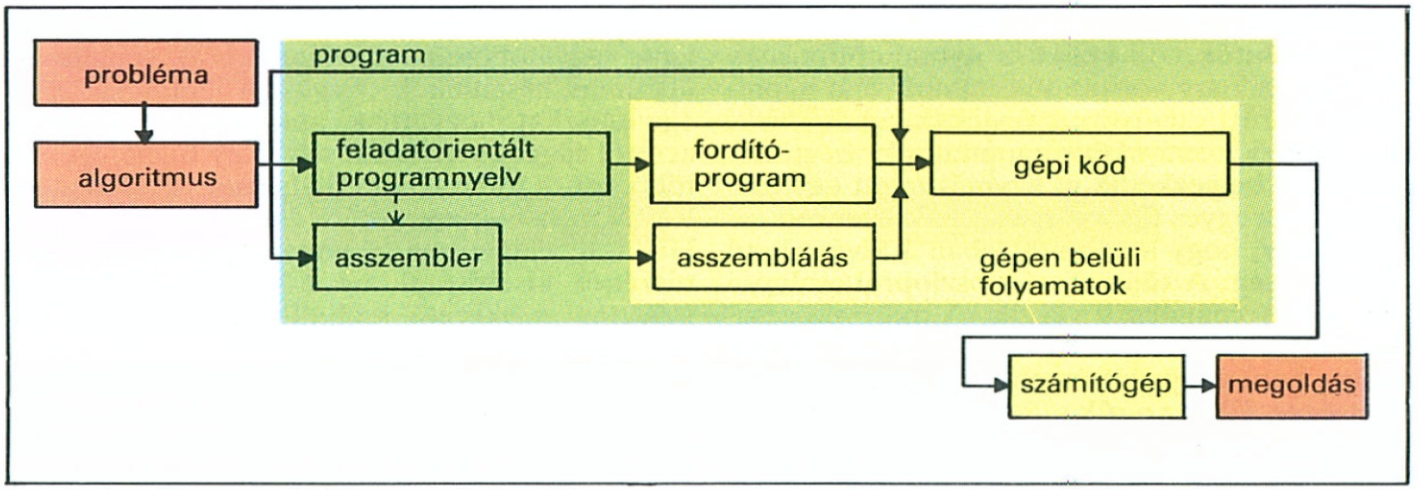
A **kétdimenziós rajzgépben** a papír sík alapzaton fekszik. A tollat mikroprocesszorral vezérelt tartóba erősítik, amely a papír fölött elhelyezett sínen mozoghat. Az x-tengelyt a sín mozgása, az y-tengelyt pedig a rajzolófej sínben való mozgása jelenti. A színes tollakat relék nyomják a papírra.

Egyéb adatkiviteli eszközök

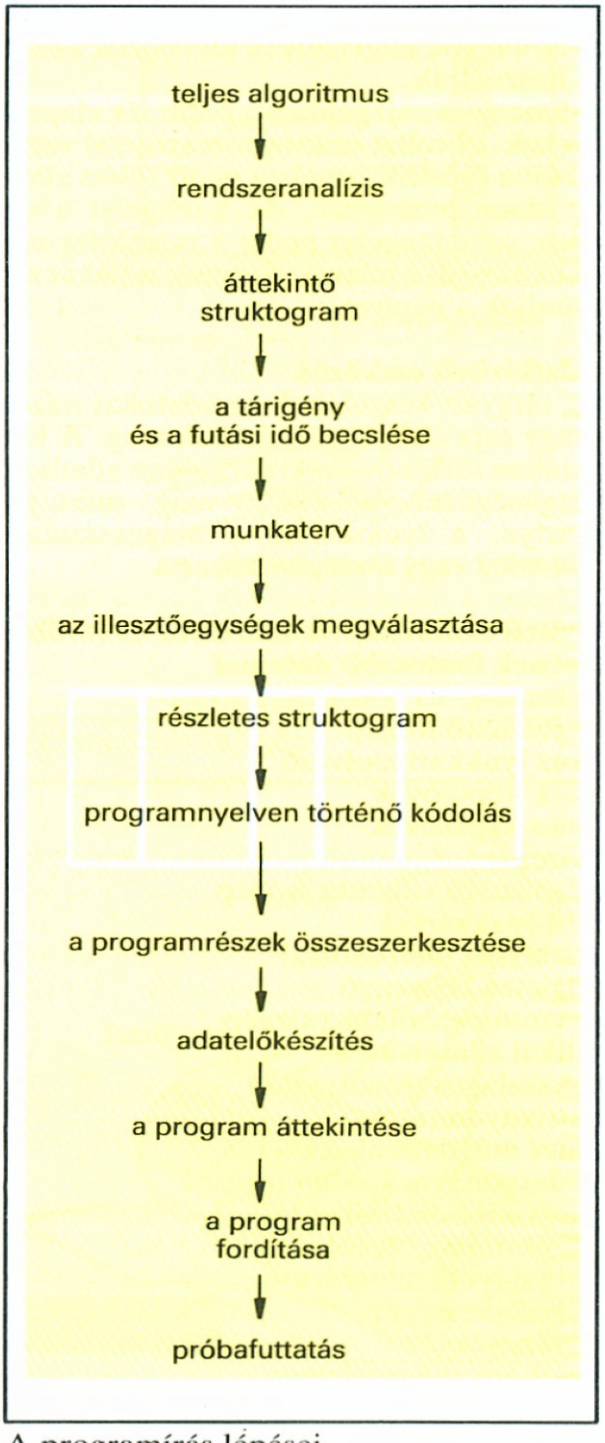
Az eddig tárgyalt készülékek az adatokat írásos szöveg vagy rajz alakjában jelenítik meg. A kódolt formában történő adatkivitel olyan adathordozók segítségével valósítható meg, mint pl. a lyukkártya, a lyukszalag, a mágnesszalag, a mágneslemez vagy a mágneskazetta.

A be-/kiviteli eszközök feltalálásának és továbbfejlesztésének fontosabb dátumai

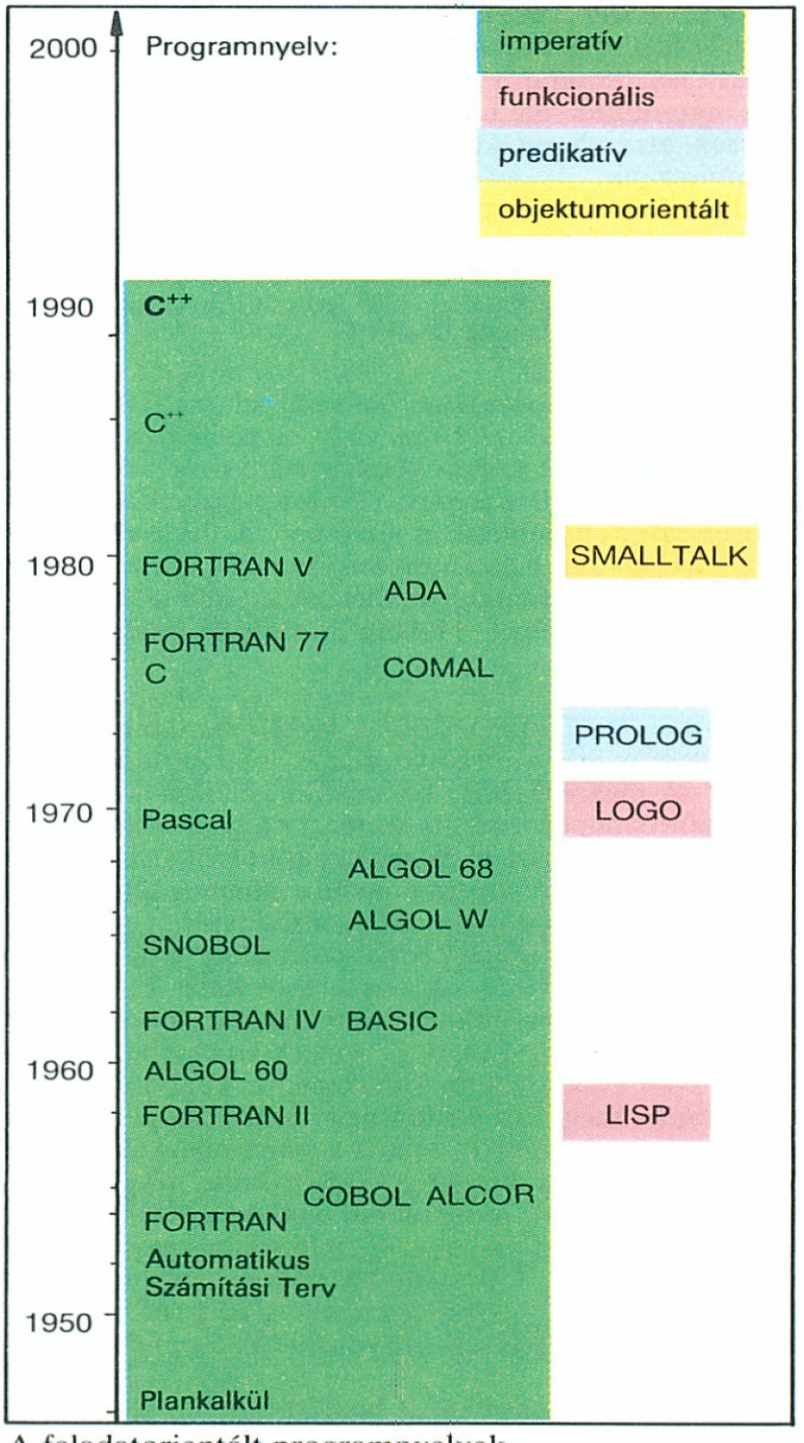
1725 lyukszalag- és lyukkártyaolvasó
1870 megbízható írógép
1890 gyors lyukkártyaolvasó
1898 kerek képcsövek
1902 színes képcsövek
1914 távírógép
1920 megbízható villamos írógép
1928 IBM-lyukkártya
1938 adatkivitel lámpasorral
1939 szögletes képernyő
1946 elektromechanikus rajzgép
1950 optikai adatolvasók
lyukszalagos íróautomata
mátrixnyomtatók
1951 fémes mágnesszalag-olvasó
1952 modern mágnesszalag-olvasók
1954 sornyomtatók (dobos)
1958 mágnestintás olvasó
1961 optikai lyukszalagolvasó
1962 gömbfejes írógép
fényceruza
1972 hajlékony mágneslemez
margarétafejes írógép
1975 lézernyomtató
1984 acélszalagos nyomtató
1990 elektroeróziós nyomtató



A feladattól a megoldásig vezető út vázlata



A programírás lépései



A feladatorientált programnyelvek fejlődése

Adott feladat megoldásának műveletek sorozatával leírt módját **algoritmusnak** nevezzük. Ahhoz, hogy egy számítógép a megoldási utat lépésről lépésre kövesse, az algoritmusban leírt valamennyi alapműveletet „meg kell értenie".

Program

A program segítségével a számítógép végre tud hajtani egy algoritmust, az eredményt el tudja tárolni és át tudja adni. Ugyanahhoz a megoldáshoz az ember is eljuthat, de a gép ezt a feladatot átveszi, mert:

Az algoritmus részleteiben számos elemi alapműveletből áll. Ha az ember intellektuális képességeit ennek megoldására fordítanánk, az a szellemi erőforrások pazarlásával lenne egyenértékű.

A feladat megoldása során néhány különböző művelet azonos előírás szerinti ismétléséről van szó, és a gép a számtalan ismétlés során sem téveszt.

A műveleteket már az egyszerű számítógépek is sok nagyságrenddel gyorsabban végzik el, mint az ember.

A programot és az adatokat maga a számítógép tárolja, és a legmegfelelőbb időpontban, pl. éjszaka dolgozza fel.

Programnyelvek

A programnyelvek (angolul: *programming languages*) az algoritmusokat és az adatstruktúrákat úgy fogalmazzák meg, hogy a számítógép a lefordított és az eredményként létrejött programot végre tudja hajtani. Napjainkban már több, mint ezer programnyelv létezik, de közülük csak mintegy 20 terjedt el széles körben.

Az első **feladatorientált** ill. **magas szintű programozási nyelv** (angolul: ***H**igh **L**evel **L**anguage,* **HLL**) a Zuse által kifejlesztett *Plankalkül* volt; az első nemzetközileg is sikeres programnyelvek az 1950-es évek közepe óta a **FORTRAN** és a **COBOL.** A feladatorientált nyelvek az algoritmust egyszerűen és a problémához illeszkedő módon írják le, és a leírás a felhasznált számítógéptípustól független. Megkülönböztetünk

- imperatív **(ADA, ALGOL, BASIC, COBOL, FORTRAN, PASCAL),**
- funkcionális és applikatív **(LISP, LOGO),**
- predikatív **(PROLOG)** és
- objektumorientált **(Smalltalk)** nyelveket.

A feladatorientált nyelvek teljesítőképessége az alábbi szempontok figyelembevételével értékelhető: világos felépítés, természetes megfogalmazás, hatékony fordítási lehetőség, optimalizált programfutás, kevés hibaforrás.

Néhány programnyelv, mint pl. a BASIC szintaxisa bizonyos számítógéptípusok belső felépítését is figyelembe veszi.

Egy feladatorientált nyelven megírt program a számítógép számára érthetetlen. Ezeket a megfelelő fordítóprogrammal (lásd alább) először le kell fordítani.

A **gépi kódok** (angolul: *machine codes*) az algoritmust úgy fejezik ki, hogy az alkalmazott számítógéptípus azt közvetlenül fel tudja dolgozni. A gépi kódú programoknak többek között az alábbi tulajdonságaik vannak:

- bináris számjegyek sorozatai,
- valamennyi matematikai műveletet erősen korlátozott számú alapműveletre vezetik vissza,
- a felhasználható be-/kimeneti eszközök előre adottak.

Hátrányai: Ugyanaz a gépi kódú program egy másik géptípuson nem futtatható. A gépi kódú programot maga a programozó is nehezen tudja áttekinteni, megértéséhez nagyon pontos dokumentáció szükséges, ennek ellenére a hibaszázalék magas. A program írása időigényes, az utólagos kiegészítés vagy átdolgozás nehéz, sőt legtöbbször lehetetlen.

Az 1950-es évek közepéig a programozás kizárólag gépi kódban történt. A nagyon hatékony fordítóprogramoknak köszönhetően a gépi kódú programozás napjainkban már fölösleges.

Az **asszembler nyelvek** olyan számítógéporientált programnyelvek, amelyekben a bináris formában kifejezett műveletek és utasítások helyett érthető szimbólumokat lehet alkalmazni. Az asszembler programok hatékonyak és relatívan kevés tárolóhelyet igényelnek, azonban gyakran az adott hardverre szabottak, és más számítógéptípuson nem futtathatók. Ezen túlmenően hátrányuk még, hogy terjedelmes programok esetén áttekinthetetlenek és nehezen érthetőek; a hibákat ezért csak fáradságos munkával lehet korrigálni.

A **fordítóprogramok** a számítógépen belüli olyan programnyelvek, amelyek egy *A (forrásnyelvű)* programot egy *B (tárgynyelvű)* programmá fordítanak le.

A **kompilerek** olyan fordítóprogramok, amelyek egy feladatorientált nyelven írt forrásprogramot asszembler vagy gépi kódú programmá alakítanak át, az első kompilert Heinz Rutishauser írta 1952-ben, és *Automatikus Számítási Tervnek* nevezte el.

A **dekompilerek** az alacsonyabb szintű forrásnyelven írt programot magasabb szintű célnyelvre fordítják le.

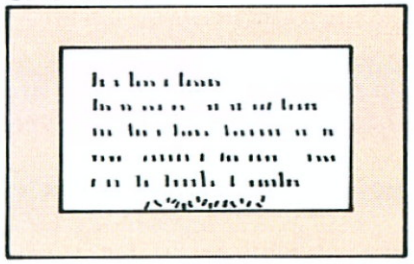
Az **asszemblerek** az asszembler-programot automatikusan a megfelelő gépi kódra fordítják.

A **kompiler-kompilerek** (fordítóprogramot generáló programok) kompilereket hoznak létre, amelyek az egyik programnyelvről a másikra képesek a kívánt programot fordítani.

Az **interpreterek** a programot lépésről lépésre fordítják le, és minden utasítást közvetlenül végrehajtanak.

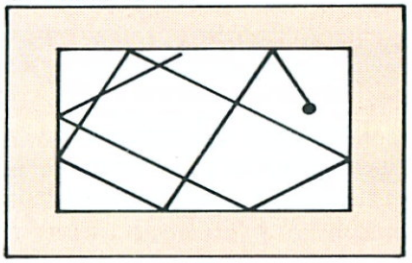
Látható:

CASCADE



a képernyőn lehulló jelek

SPRINGBALL

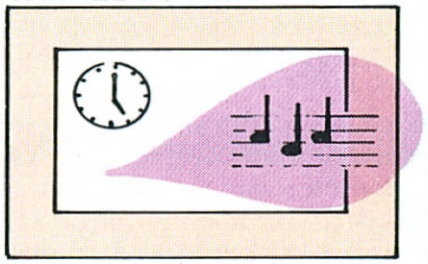


pályavonalak a szövegben

MARIJUANA

üzenet

YANKEE DOODLE

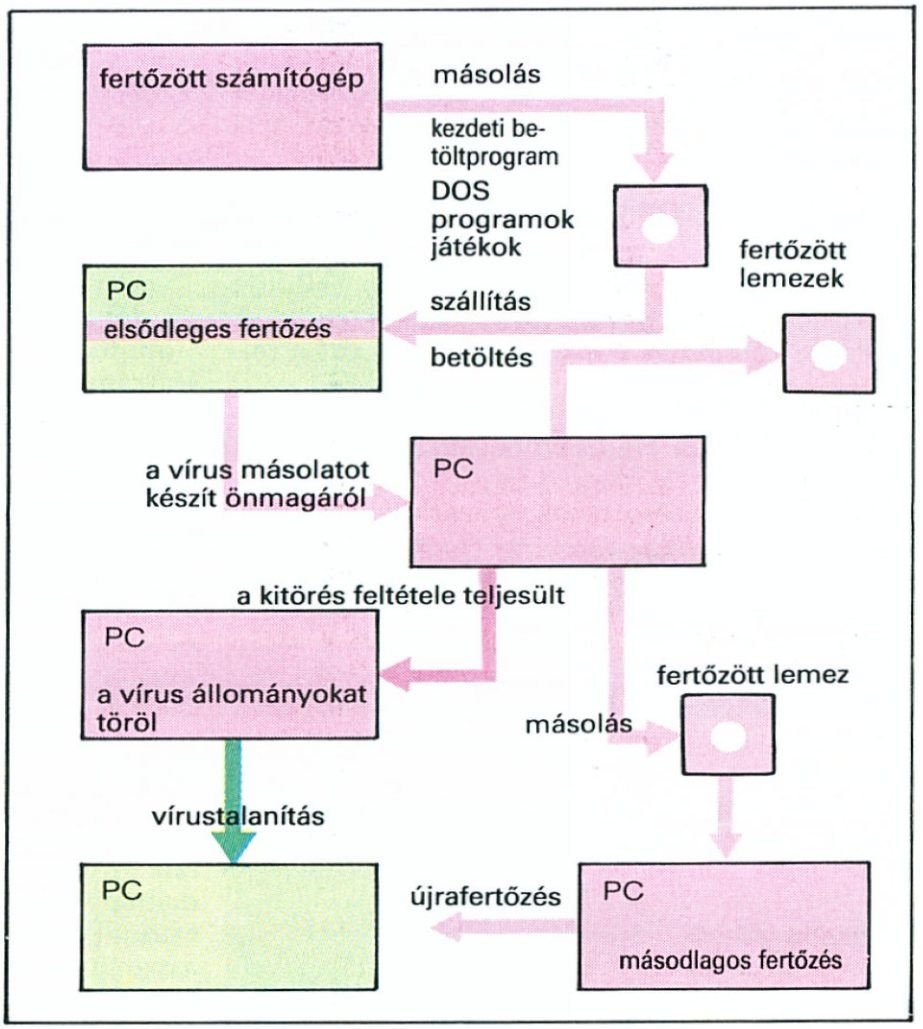


17:00 órakor zene

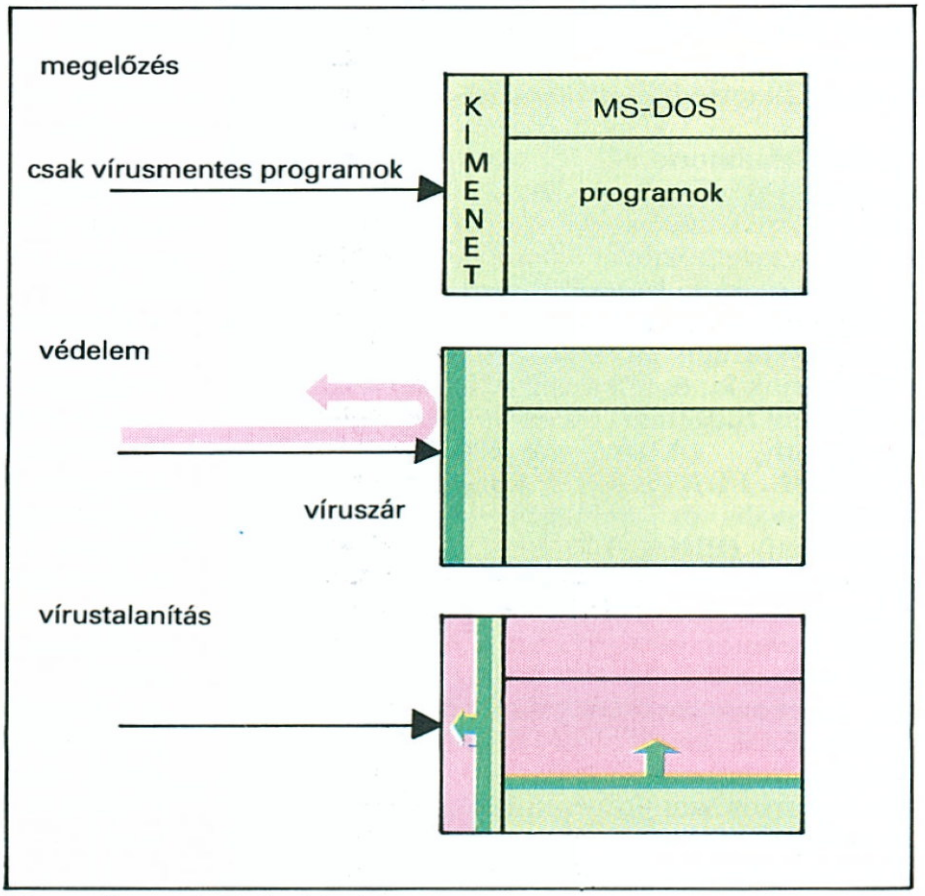
Nem látható:

FAT törlése
állományszegmensek törlése
programleállítás
tároló-túlcsordulás
időtúllépés
állományok megváltoztatása
nyomtatóregiszterek blokkolása
⋮

Az aktivizálódott vírus által kifejtett hatás



Egy számítógépes vírusfertőzés lefolyása



Vírusellenes intézkedések

A **számítógépvírus (vírus)** általános elnevezés a számítógépben nemkívánatos programrészek megjelölésére. Legtöbbször az operációs rendszerbe ágyazódik be vagy a felhasználói programokba férkőzik és alkalomadtán onnan fejti ki káros hatását, pl. teljes állományokat kitörölhet, vagy rendszerprogramokat változtathat meg. A vírusprogramot az egyéb programoktól az különbözteti meg, hogy képes önmagát sokszorosítani. A legegyszerűbb esetek az 1980-as évek kezdete óta léteznek, pl. az a részprogram, amely a kifizetésre kerülő bérlistákat átvizsgálta, és ellenőrizte, hogy a programozó neve szerepel-e rajta. Abban az esetben, ha a keresés eredménye nemleges, vagyis ha a programozó elbocsátásra került, akkor a program olyan utasítást tartalmazott, hogy az egész bérlistát törölje ki. Az ilyen alprogramot *„trójai ló"* típusúnak nevezzük, amely tulajdonképpen nem vírus, mivel nem sokszorosítja önmagát.

A vírusok terjesztése a számítógépes bűnözéshez tartozik. Ha a kár bizonyítható, az elkövetővel szemben kártalanítási igénnyel lehet fellépni. A fertőzött programot **gazdaprogramnak** nevezzük.

A számítógépvírusok csak működő programokba képesek behatolni. A fertőzés gyorsan megtörténik, kezdetben a felhasználó nem vesz észre semmit. A vírusprogramok rendszerint rövidek, hosszúságuk általában 100 és 300 sor közé esik.

Példák viszonylag ártalmatlan vírusokra:

A CASCADE vírus hatására az egyes betűk elhagyják eredeti szövegbeli helyüket és a képernyő alján gyűlnek össze.

A FU-MANCHU vírus szitkozódó szavakat jelenít meg a képernyőn, valahányszor a szövegben valamilyen vezető politikus neve fordul elő.

A vírusok felépítése és működésmódja

A számítógépvírusok rendszerint két, különböző feladat ellátására szánt részből állnak:

A **fertőző rész** (angolul: *infector*) a vírust a gazdaprogramba másolja. Az esetek 70%-ában az operációs rendszert fertőzi meg. A vírusprogramok kedvelt célpontjai a kezdeti betöltőprogram és a végrehajtó (angolul: *executable*) programok, mint pl. az átfedő állományok (angolul: *overlay files*).

A **romboló rész** (angolul: *detonator*) a vírusprogramnak az a része, amely a tulajdonképpeni hatást fejti ki, pl. törli a tároló bizonyos részeit, gátolja az adatok áramlását, nem kívánt üzeneteket ír ki, megváltoztatja a képernyőn való megjelenítést, megszakítja a működő programot stb. A romboló rész mindaddig nyugalomban van a gazdaprogramban, míg egy adott esemény, pl. péntek 13-a be nem következik, vagy egy adott beprogramozott jelszó meg nem jelenik stb.

Példa: a LEHIGH vírus kb. 1 Kbyte hosszúságú program, amely a személyi számítógépek operációs rendszerét támadja meg. A vírus romboló része a nagyon gyakran használt COMMAND.COM állományba épül be. Ha a gazdaprogram a merevlemezről egy programot behív, az is megfertőződik. Ha a LEHIGH nagy állományba beépült, a vírus aktivizálódik: a romboló részben található utasítások törlik a merevlemez egyes tárolószegmenseit. A másik négy másolt vírus tovább terjed, és mindegyik a maga negyedik másolatára vár, hogy szintén aktivizálódhasson.

A vírusoknak nem kell feltétlenül romboló részszel rendelkezniük ahhoz, hogy hatásukat kifejtsék. Az is elegendő, ha pl. a vírus újra és újra bemásolja magát a gazdaprogramba, míg végül az egész tárolót megtölti és blokkolja.

A **programférgek** (angolul: *worms*) a számítógépes hálózatokban terjedő olyan speciális vírusok, melyek szaporodnak. Eredetileg a nyugalmi időszakokban a hálózaton „átmászva" megvizsgálták azt és felderítették a kieső komponenseket. A kárt okozó férgek nyomában azonban „lyukak" maradnak vissza.

A számítógépvírusokkal szemben foganatosítható óvintézkedések

Megelőzés. A vírus csak fertőzött program segítségével terjedhet el. A fertőzés forrásai gyakran az egyszerűen hozzáférhető programok, pl. az iskolák vagy a közkönyvtárak programjai, vagy az ismerősöktől másolt játékprogramok. Különösen veszélyeztetett a kezdeti betöltőprogram, ezért ajánlatos mindig saját programot használni, mivel az eredeti lemez írásvédett, a vírus tehát nem kerülhet rá. A számítógépes hálózatok fertőzése a különleges védőintézkedések eredményeképpen már nagyon ritkán fordul elő.

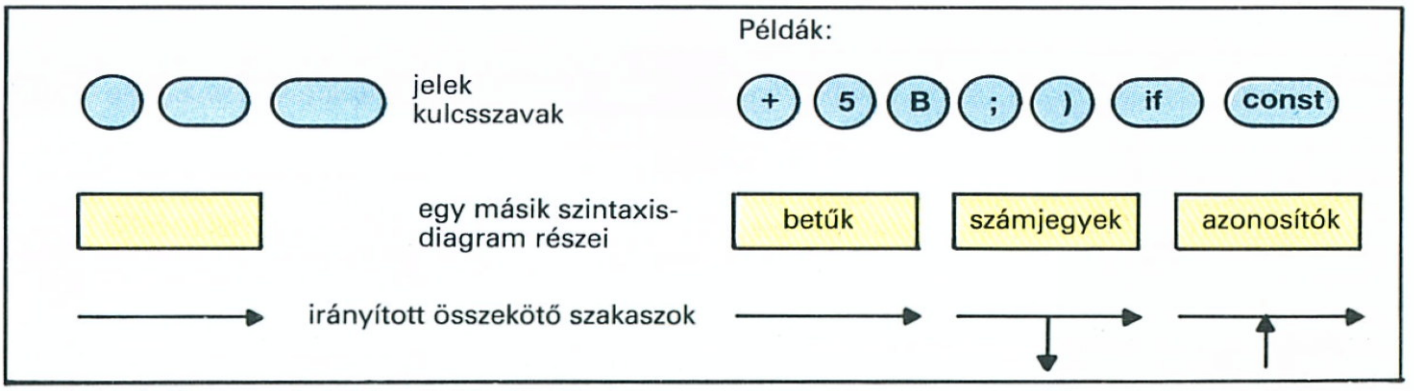
Korai felismerés. Vírus jelenlétére figyelmeztető jel pl., ha a szabad tárolóterület egyre csökken, annak ellenére, hogy nem tárolunk újabb adatokat; ha az egyes programok egyre terjedelmesebbé válnak; ha a számítógép sebessége csökken; ha a kezdeti betöltés egyre hosszabb ideig tart.

Védelem. Léteznek vírusfelismerő programok, pl. a VIRUS-SHIELD, amelyek a vírus behatolási kísérletét felismerik, azt a felhasználónak jelzik, és a fertőzést megakadályozzák.

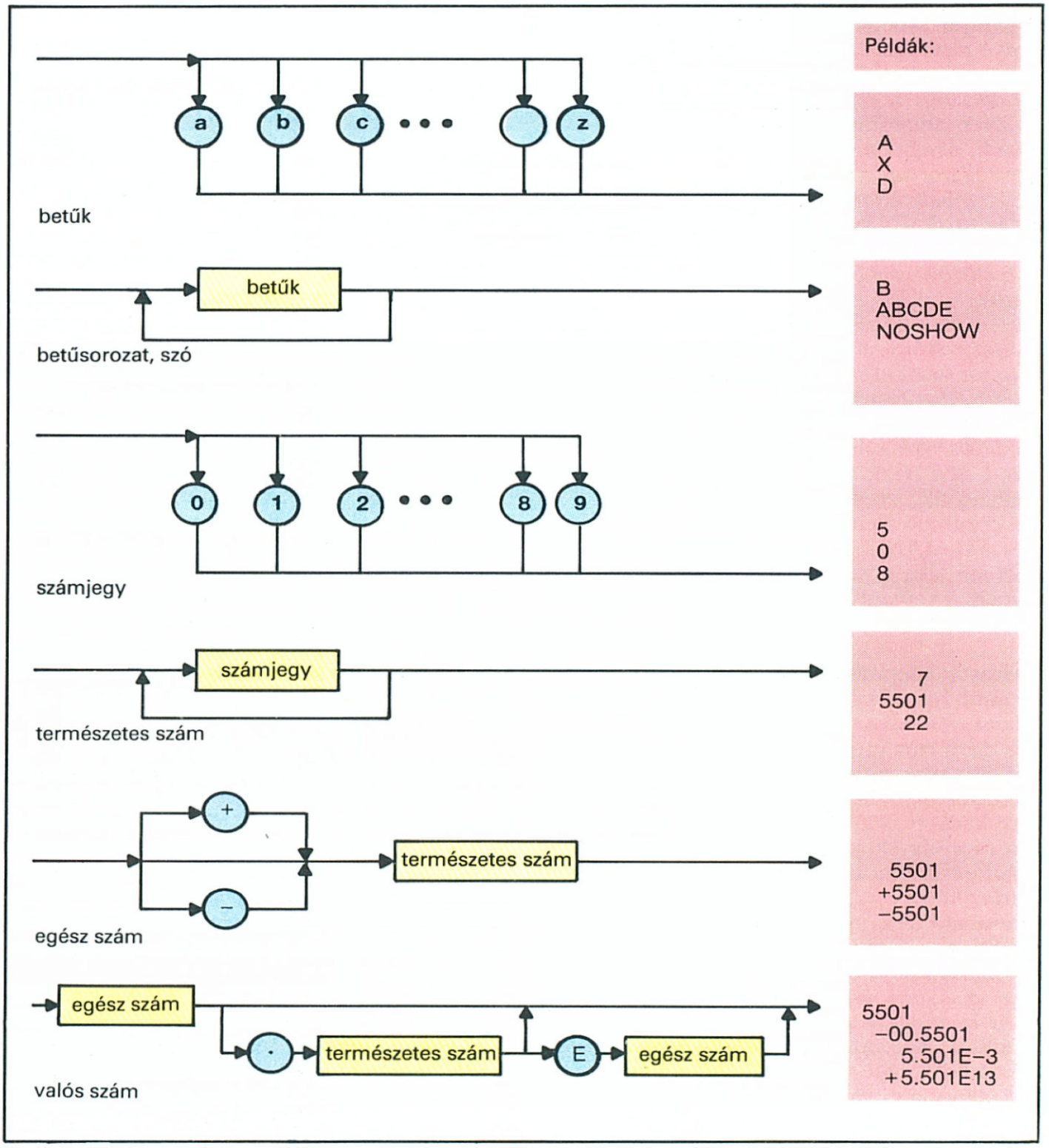
Gyógyítás. Ha a számítógép már fertőzött, akkor különleges víruskereső programok (angolul: *serum*) segítségével a betolakodó megtalálható, utasításai törölhetők és az eredeti állapot helyreállítható.

Példák: a VIRUSCAN, a TBAV, a FLU-SHOT+, a VSHIELD nevű PC-programok az ismert vírusokat felismerik. Meghatározott időközökben, pl. minden bekapcsolás után megvizsgálják az operációs rendszert és a felhasználói programokat.

Az **álcázott vírusok** (angolul: *stealth virus*) különösen nehezen felismerhetők. Ha víruskereső programmal találkoznak, akkor azt a részüket mutatják, amelyet a víruskereső program „korrektnek" tekint.



A szintaxisdiagramoknak a programnyelvelemek ábrázolására szolgáló részei



Egyszerű szintaxisdiagramok

A magasabb szintű programnyelvek fejlődése az 1950-es évek közepén kezdődött el, és az 1960-as években olyan feladatorientált programnyelvek váltak uralkodóvá, mint a FORTRAN, az ALGOL 60 vagy a COBOL. Harminc év elteltével teljesen géptől független nyelvek terjedtek el, ilyen az ALGOL 68, a PASCAL, a C és a SIMULA 67. Azonban a géphez kötődő és viszonylag egyszerű nyelvek, mint a BASIC vagy a FORTRAN különböző változatai (dialektusai) még mindig széles körben használatban vannak. A következő oldalon elsősorban a géptől független PASCAL nyelvből vett példákat mutatunk be, de néha a géphez közeli BASIC nyelv példái is előfordulnak. A tulajdonképpeni nyelvelemet (kulcsszót) félkövér betűvel **(begin)** ill. csupa nagybetűvel (BEGIN) jelöljük.

A szintaxisdiagramok

A szintaxisdiagramok grafikai jelek felhasználásával részletesen leírják egy programnyelv szintaxisát. A programon belül az egyes nyelvi elemek összeállítása csak akkor korrekt, ha a szintaxisdiagram alapján ugyanolyan sorrend kialakítható belőlük.

A blokkdiagramok és a struktogramok (144. o.) a program menetét lépésről lépésre írják le, a szintaxisdiagramhoz nem sok közük van.

A szintaxisdiagramok olvasása:

Az olvasás irányát nyilak → ↑ ↓ határozzák meg.

A *kulcsszavak (terminális szimbólumok)* a nyelv tulajdonképpeni elemei.

Ezeket körökbe vagy lekerekített sarkú téglalapokba írják.

Példák a PASCAL nyelvből: [:=], (:), ((.

[**begin**], [**array**].

Példák a BASIC nyelvből: [=], (:), ((.

[BEGIN], [DIM].

Az ún. *nemterminális szimbólumok* nyelvtani konstrukciókat ábrázoló jelek vagy jelsorozatok, amelyeket téglalap alakú keretbe írnak.

Példák: [azonosító], [számjegy], [betű],

[adattípus], [utasítás].

A nemterminális szimbólumok egyéb, nemterminális szimbólumokra és/vagy kulcsszavakra bonthatók.

Figyelem: a példákban ábrázolt szintaxisdiagramok nem mindig mutatják be az összes lehetőséget.

Állandók

Az állandók olyan rögzített számértékek, amelyek a számítógép számára alkalmas formában a programba építhetők, ez rendszerint már a program írásakor megtörténik.

A decimális számértékek ábrázolására gyakorlatilag az összes programnyelv három különböző forma egyikét használja. Ezekben a szokásos tizedesvessző helyett tizedespont áll. Az előjel nélkül felírt számok megállapodás szerint pozitívnak tekinthetők.

Egész szám:

+ *számjegysorozat*
számjegysorozat
− *számjegysorozat*

Példák: +1234, 1234, −1234

Valós szám:

+ *számjegysorozat.számjegysorozat*
− *számjegysorozat.számjegysorozat*
számjegysorozat.számjegysorozat

Példák: +123.45678, −123.45678, 0.12345

Valós szám hatványkitevős ábrázolása:

± *számjegysorozat.számjegysorozat*
E ± *számjegysorozat*

Példák: +123.456E24; 0.1234E − 16 és −12.3E+9

Ezeknek a szokásos hatványkitevős alakban a következő számok felelnek meg:

$123{,}456\times10^{24}$; $0{,}1234\times10^{-16}$; $-12{,}3\times10^{9}$

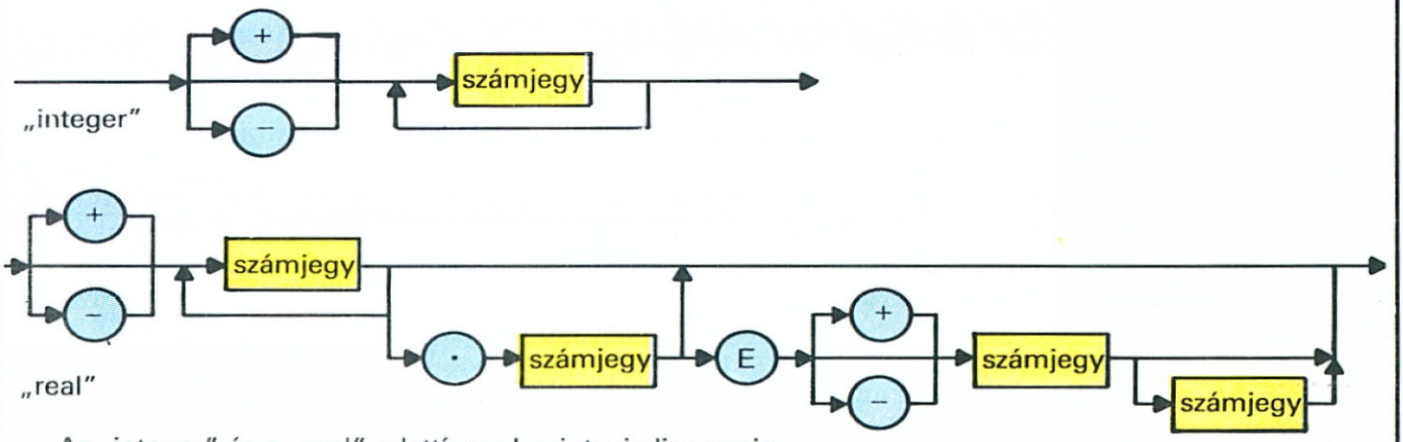
Változók

A változók a számítógépben meghatározott tárolóterületet képviselnek. Ha a program futása során a változó ill. a változó azonosítója (133. o.) megjelenik, akkor a program a megfelelő tárolóterület tartalmát használja fel. A számítógép rendszerint belső úton határozza meg a változóhoz rendelt tárolóterület maximális nagyságát.

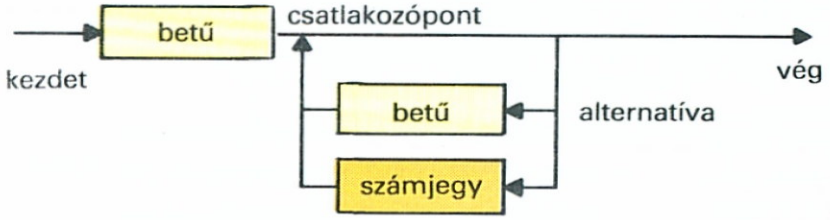
Egy változónak tulajdonképpen két jelentése van: ha az értékadó utasítás (137. o.) jobb oldalán áll, akkor az utasítás a változó helyen a hozzárendelt tárolóterület tartalmát használja fel. Ha a változó az értékadó utasítás bal oldalán áll, akkor a változóhoz rendelt tárolóterület a jobb oldalon meghatározott új értéket veszi fel. A tárolóterület előző tartalma törlődik (a tároló *felülírása*).

A változó ábrázolási formáját (és gyakran számjegyeinek maximális számát is) a programnyelv definíciója rögzíti. A változó azonosítója mindig betűvel kezdődik, amelyet esetleg további betűk és/vagy számjegyek követnek.

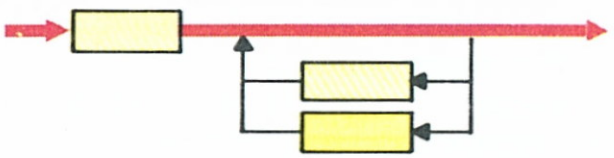
A változókat első megjelenésük előtt számos programnyelvben deklarálni kell. Ezzel egyidejűleg a programozó azt is rögzíti, hogy a változó mekkora tárolóterületet vehet igénybe.



Az „integer" és a „real" adattípusok szintaxisdiagramja

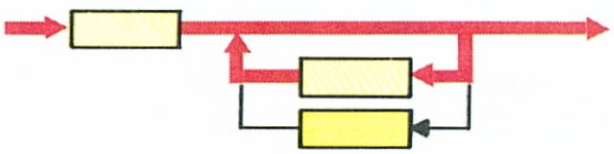


Az azonosítók szintaxisdiagramja

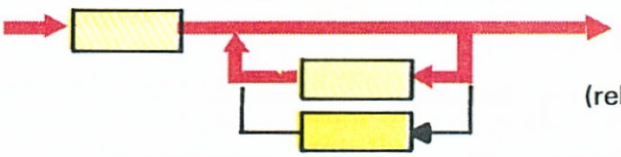


példák:

Pascal	*Basic*
a	A
x	X
d	D

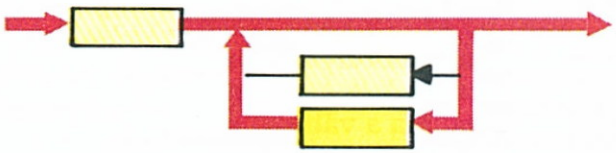


da	DA
ex	EX
pu	PU

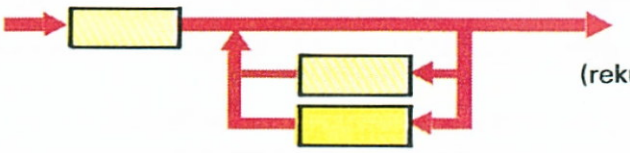


(rekurzív)

kim	KIM
kártya	KÁRTYA
vezérlés	VEZÉRLÉS



t8	T8
s2	S2
k0	K0



(rekurzív)

d9t	D9T
ak8	AK8
mig26	MIG26

Az azonosítók szintaxisdiagramjainak értelmezése

Néhány adattípus és azonosító szintaxisdiagramja

Műveletek

A matematikai kifejezések operandusok (pl. állandók, változók, függvények), operátorok (pl. +, –, /, >, <, =) és zárójelek összekapcsolásával állíthatók elő. Az így keletkezett kifejezések feldolgozása – a matematikában szokásos módon – a legmagasabb rendű operátorral kezdve egymás után történik.

A legismertebb operátorok **rangsora** a következő: a szorzás és az osztás magasabb rendű, mint az összeadás és a kivonás, ezután következnek a relációs műveletek.

Azonos rendű operátorok esetén balról jobbra haladva kell elvégezni a műveleteket. A zárójelek a műveletek szokásos sorrendjétől eltérő sorrendet rögzítenek.

A programozás során az alsó és felső indexeket is normál sormagasságba írjuk és zárójelbe tesszük. A szorzásjelet mindig ki kell tenni.

Példák a műveletek felírási módjára:

programozási nyelvek	jelentés
A + B	A + B
A – Y/X	$A - \frac{y}{x}$
1 + (A + B)/(A – B)	1 + (A + B)/(A – B)
PI * (R1 * R1 – R * R)	$\pi(R_1^2 - R^2)$
(A – B) * (A – B)	$(A - B)^2$

A programnyelvekben léteznek közvetlenül végrehajtható standard függvények, mint pl. a gyökvonás, négyzetreemelés, logaritmus, szinuszfüggvény stb. A többi függvényeket felhasználásuk előtt definiálni kell. A speciális matematikai jeleket, pl. a Σ és a Π jelet le kell fordítani, pl. az előzőleg definiált SUM vagy PROD függvényekkel.

Azonosítók

Az azonosítók (angolul: *identifier*) olyan jelek vagy jelsorozatok, amelyek a program különböző objektumait, pl. a változók nevét, az állandókat, az adatmezőket, az eljárásokat jelölik. Az azonosítók formáját a programnyelvek rögzítik.

Az azonosítók mindig betűvel kezdődnek, helyközt nem tartalmazhatnak. Gyakorlati okokból az azonosítókban szereplő jelek száma korlátozott, pl. a PASCAL-ban az azonosítók legfeljebb 127, a BASIC-ben pedig általában legfeljebb 6 jelből állhatnak.

Azonosítóként rendszerint nem használhatók a programnyelv definiált elemei, a kulcsszavak ill. az előre lefoglalt szavak.

Az azonosítókat explicit vagy az első előforduláskor implicit módon deklarálni kell.

Az azonosítók **érvényességi tartománya** a program összes olyan részét magában foglalja, amelyre az azonosítót deklaráltuk. Blokkszerkezetű programban előfordulhat, hogy ugyanaz az azonosító az egyes blokkokban különböző objektumot jelöl, azonban ezt jobb elkerülni.

Standard adattípusok

Logikai (Boolean) adattípus. A megfelelő azonosító logikai kifejezésekben fordul elő. Értéktartománya az igaz és a hamis elemekből áll.

Példák: PASCAL: **true, false.** BASIC: TRUE, FALSE.

A matematikai logikának a Boole-típusú azonosítókra alkalmazható operátorai a ∧ (konjunkció), a ∨ (diszjunkció) és a ¬ (negáció).

A PASCAL nyelvben a megfelelő kulcsszavak **and, or** és **not.** A BASIC az AND, OR és NOT kulcsszavakat, valamint az EQUIV és az IMPL összetett operátorokat alkalmazza.

Integer (egész) adattípus. A megfelelő azonosító aritmetikai kifejezésekben fordul elő. Értéktartománya a pozitív és negatív egész számok részhalmaza, amely a korlátozott tárolókapacitás miatt – a számítógéptől függően – alulról és felülről korlátos, pl. $-2^{16} + 1 = -65535$ és $+2^{16} + 1 = +65535$ közé esik.

Ha egy *integer* típusú azonosító értéke nagyobb vagy kisebb a definiált értéktartománynál, akkor a számítógép **túlcsordulást** (angolul: *overflow*) vagy **alulcsordulást** (angolul: *underflow*) jelez.

Az *integer* típusú azonosítókra alkalmazható aritmetikai műveletek az összeadás, a kivonás, a szorzás, az egész számú osztás és a modulus. Ezeknek a műveleteknek az eredménye is *integer* típusú.

Az egész számú osztás (matematikai jelölése: ÷) eredménye is egész szám.

Az esetleg megjelenő maradékot a számítógép kerekíti vagy levágja.

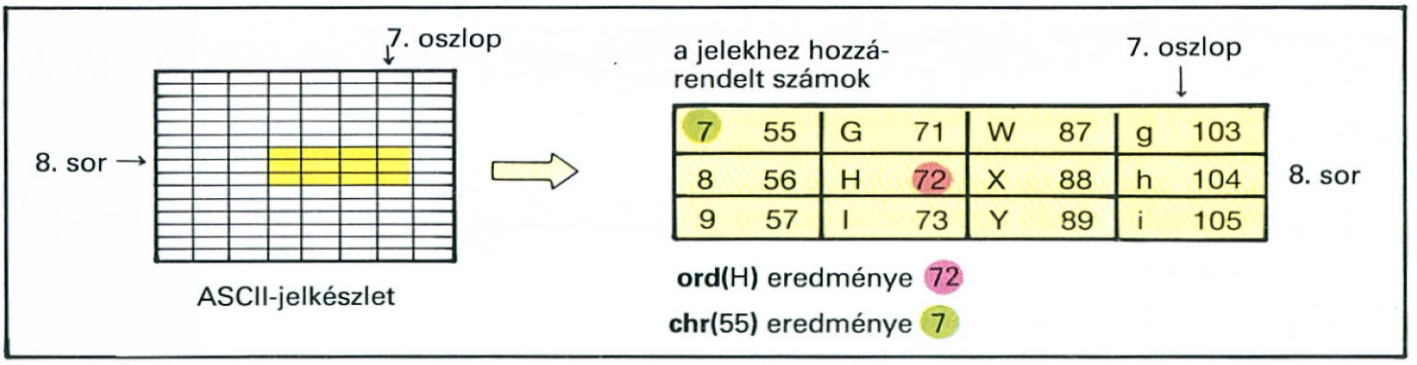
A PASCAL a +, −, *, **/, div** és a **mod** operátorokat alkalmazza.

Az *a* **mod** *b* kifejezés helyett *a* – (*a* **div** *b*) * *b* is írható.

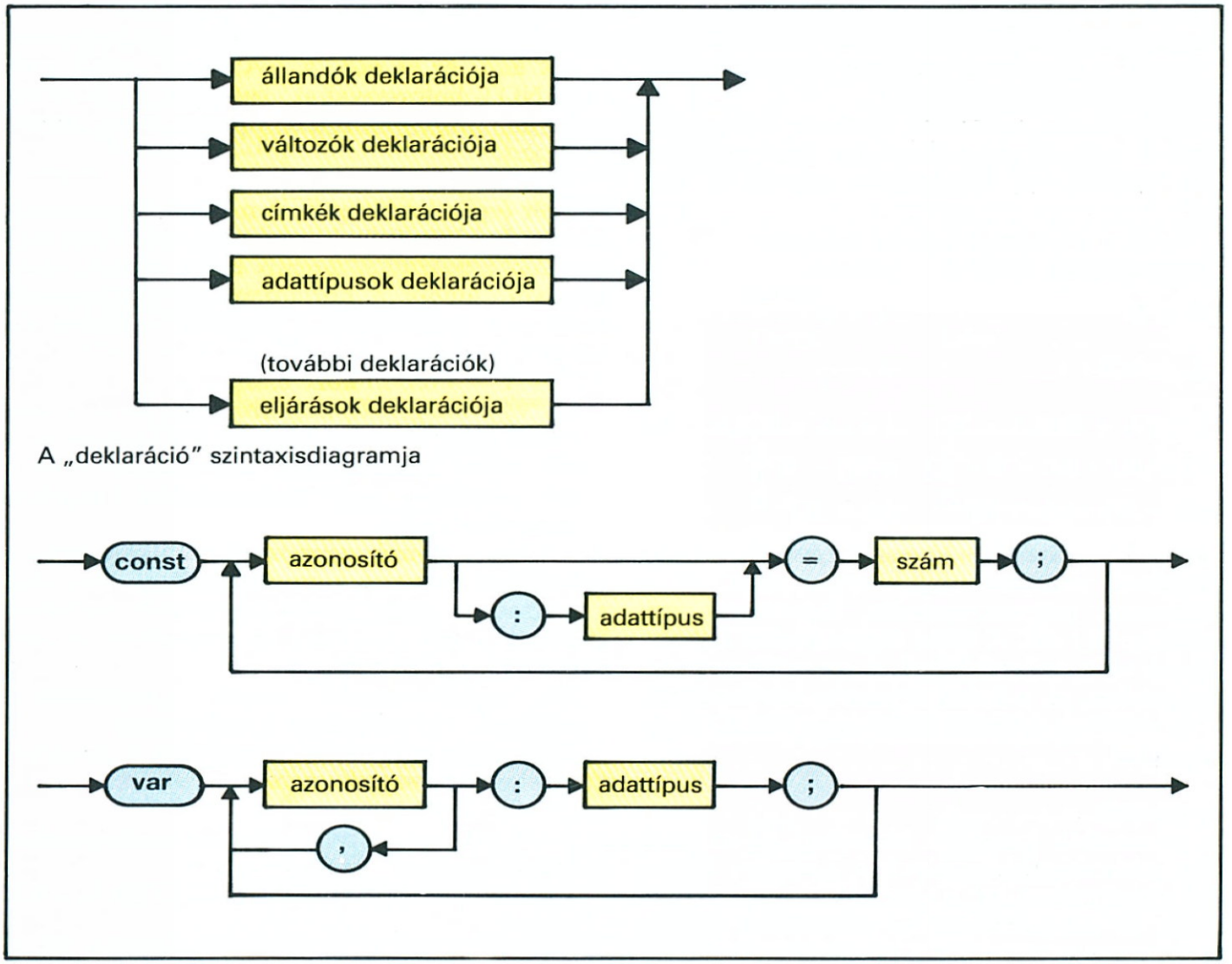
A BASIC a +, −, * és / operátorokat alkalmazza.

Real (valós) adattípus. A megfelelő azonosító aritmetikai kifejezésekben fordul elő. Értéktartománya a valós számok halmaza, amelybe az egész számok is beletartoznak. A *real* típusú azonosító számértéke a megfelelő valós szám közelítő értéke. A tárolószavak – gyakran megválasztható – korlátozott hosszúsága miatt az értéktartomány is (de facto) korlátozott.

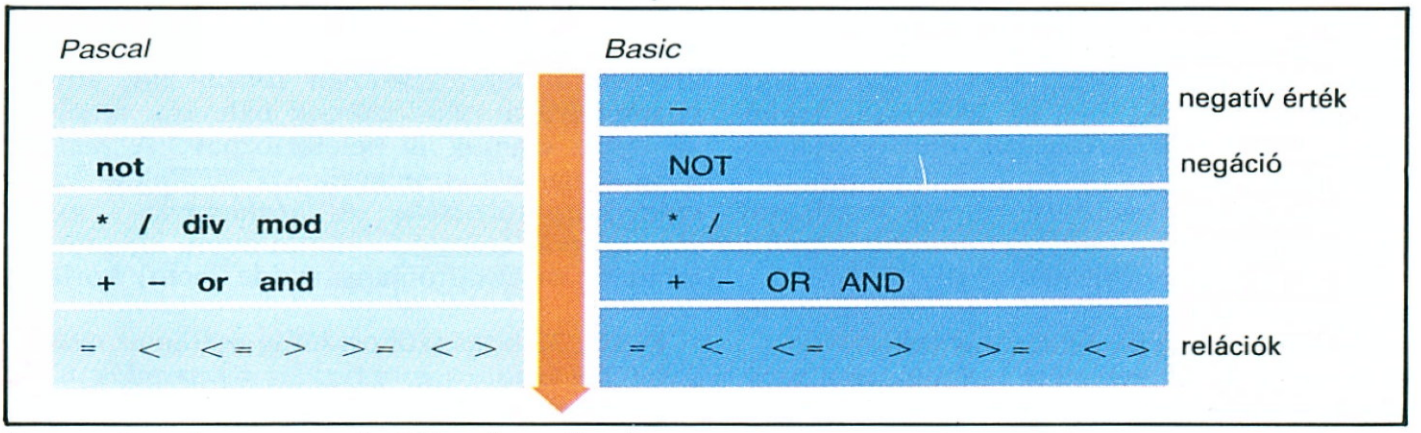
Példa: 32 bites szóhosszúság esetén az első 26 bit ábrázolja a mantisszát, a maradék 6 bit a kitevőt. Normalizált lebegőpontos ábrázolásban (65. o.) ez hétjegyű mantisszának, valamint –11 és +10 közé eső kitevőnek felel meg.



Az **ord** és **chr** *Pascal*-függvények eredménye ASCII-kód felhasználása esetén



Állandók és változók deklarációja a *Pascal* nyelvben



Az operátorok rangsora (csökkenő rendben)

A *real* típusú azonosítókra alkalmazható műveletek az összeadás, a kivonás, a szorzás és az osztás. Ezeknek a műveleteknek az eredménye is *real* típusú. A PASCAL nyelv a +, −, * és / műveleteket alkalmazza.

Ha a PASCAL az / műveletet *integer* típusú azonosítókra alkalmazza, akkor az eredmény *real* típusú.

A BASIC nyelvben ugyanazok a műveletek használatosak, mint a PASCAL-ban.

Ha a BASIC az / műveletet *integer* típusú azonosítóra alkalmazza, akkor az eredmény is *integer* típusú. Az esetleg megjelenő maradékot a számítógép levágja.

Ha egy kifejezésben *real* és *integer* típusú azonosítók is előfordulnak, akkor az eredmény mindig *real* típusú lesz.

A standard függvények (139. o.) eredménye *real* típusú. Kivételek: a néhány programnyelvben megtalálható **trunc** és **round** standard függvények *integer* adattípust eredményeznek.

A **char (karakter) adattípus** értéktartománya karakterek (angolul: *character*) olyan rendezett halmaza, amely legalább a nagybetűket, a számokat és a helyközt tartalmazza. A rendezettség azt jelenti, hogy a karakterek halmazában minden elemhez egy egész szám rendelhető.

Példa: az ASCII-kód karakterkészlete (a NUL karakterhez rendelt) 0-tól (a DEL karakterhez rendelt) 127-ig terjed. Ez a hozzárendelés relációs műveleteket tesz lehetővé.

A *char* típusú azonosítókat a PASCAL nyelvben felső indexbe írt vesszők (') közé, más programnyelvekben pedig idézőjelbe („") tesszük.

A legtöbb programnyelvben létezik két, karakter típusú azonosítókra alkalmazható standard függvény: az egyik a *char* típusú azonosítókhoz rendel hozzá egy természetes számot, a másik pedig egy (nagyon korlátozott számú elem közül kiválasztott) természetes számhoz rendel egy *char* típusú azonosítót. Ezek a függvények tehát a *char* típusú azonosítókat *integer* típusúvá alakítják és fordítva.

Példa: a PASCAL nyelvben ASCII-kódban az **ord**(H) függvény a 72, míg az **ord**(h) a 104 értéket adja eredményül, fordítva pedig a **chr**(72) a H karaktert szolgáltatja. A két függvény együttes hívása egymást kölcsönösen semlegesíti:

$$\mathbf{ord}(\mathbf{chr}(a_1)) = a_1$$

$$\mathbf{chr}(\mathbf{ord}(a_2)) = a_2$$

A programnyelvek a *char* típusú azonosítók között rendszerint lehetővé teszik a $<$, $\leq$, $=$, $\geq$, $>$ és $\neq$ relációs műveletek alkalmazását.

Példa: a PASCAL nyelvben ASCII-kódban érvényes a h > H egyenlőtlenség, mivel a karakterekhez rendelt számokra is érvényes, hogy 104 > 72.

Deklarációk

A deklarációk a program ill. a blokk előtt rögzített megállapodások, amelyek az egyes azonosítók természetét, pl. a hozzá tartozó *adattípust* határozzák meg. A deklarációk nem változtatják meg a program állapotát, csupán azt adják meg, hogy az elkövetkező programrészben az adott azonosítót hogyan kell kezelni.

A PASCAL nyelv explicit deklarációt követel, egyéb nyelvekben azonban implicit deklaráció is létezik. Ez azt jelenti, hogy amikor egy nem deklarált azonosító a program futása során először megjelenik, maga a számítógép határozza meg az összefüggésből vagy az azonosító formájából az onnan kezdve érvényes deklarációt.

Példa: az olyan változók, amelyek azonosítója I-vel kezdődik, a FORTRAN nyelvben *integer* típusúak.

Állandók deklarációja

Az **állandóknak** lehet azonosítója, de ez nem kötelező. Az állandókat deklaráljuk, egy értéket rendelünk hozzájuk, és ezután az érvényességi tartományon belül már nem változhatnak. Az állandók adattípusát legtöbbször maga a számítógép határozza meg az ábrázolás alakjából. Az 5 állandó nyilvánvalóan *integer,* a 3.14 állandó pedig *real* típusú.

Az állandó deklarációja a PASCAL nyelvben:

const K_1; K_2; ...; K_n**;**

K_1, K_2, ..., K_n: állandók.

Az állandó azonosítója és értéke között egyenlőségjel (=) áll.

Példák: **const** pi = 3.1416; h = 0.66262E–33; E = 1680.

a számok ábrázolásából kiderül, hogy az első két állandó *real,* a harmadik pedig *integer* típusú.

Változók deklarációja

A **változókat** az azonosítók jelölik. A deklaráció minden változóhoz egy adattípust rendel.

A változók deklarációja a PASCAL nyelvben:

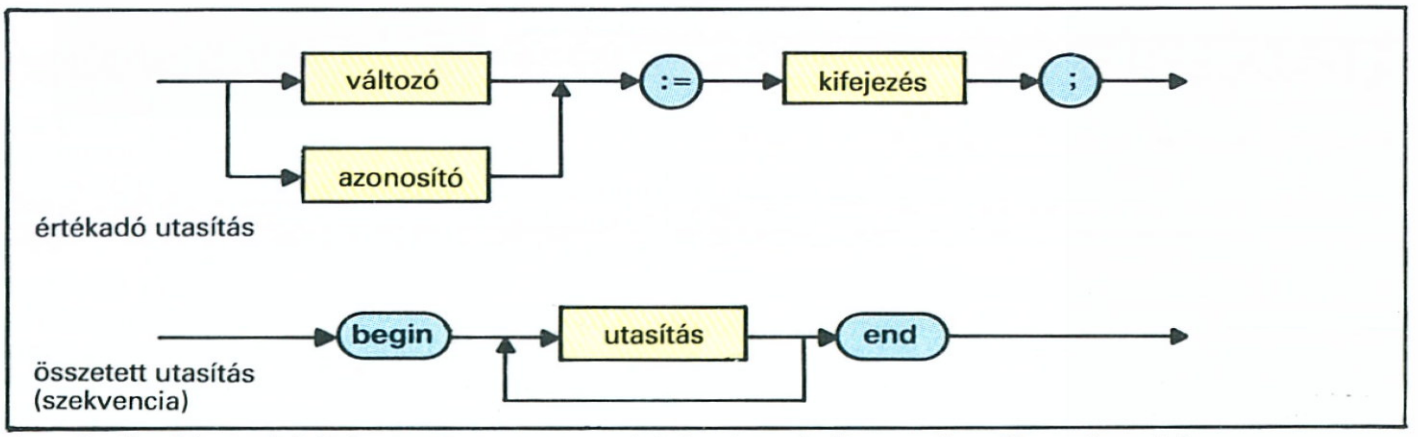
var V_1, V_2, ..., V_n: T**;**

V_1, V_2, ..., V_n: a különböző változók azonosítói.

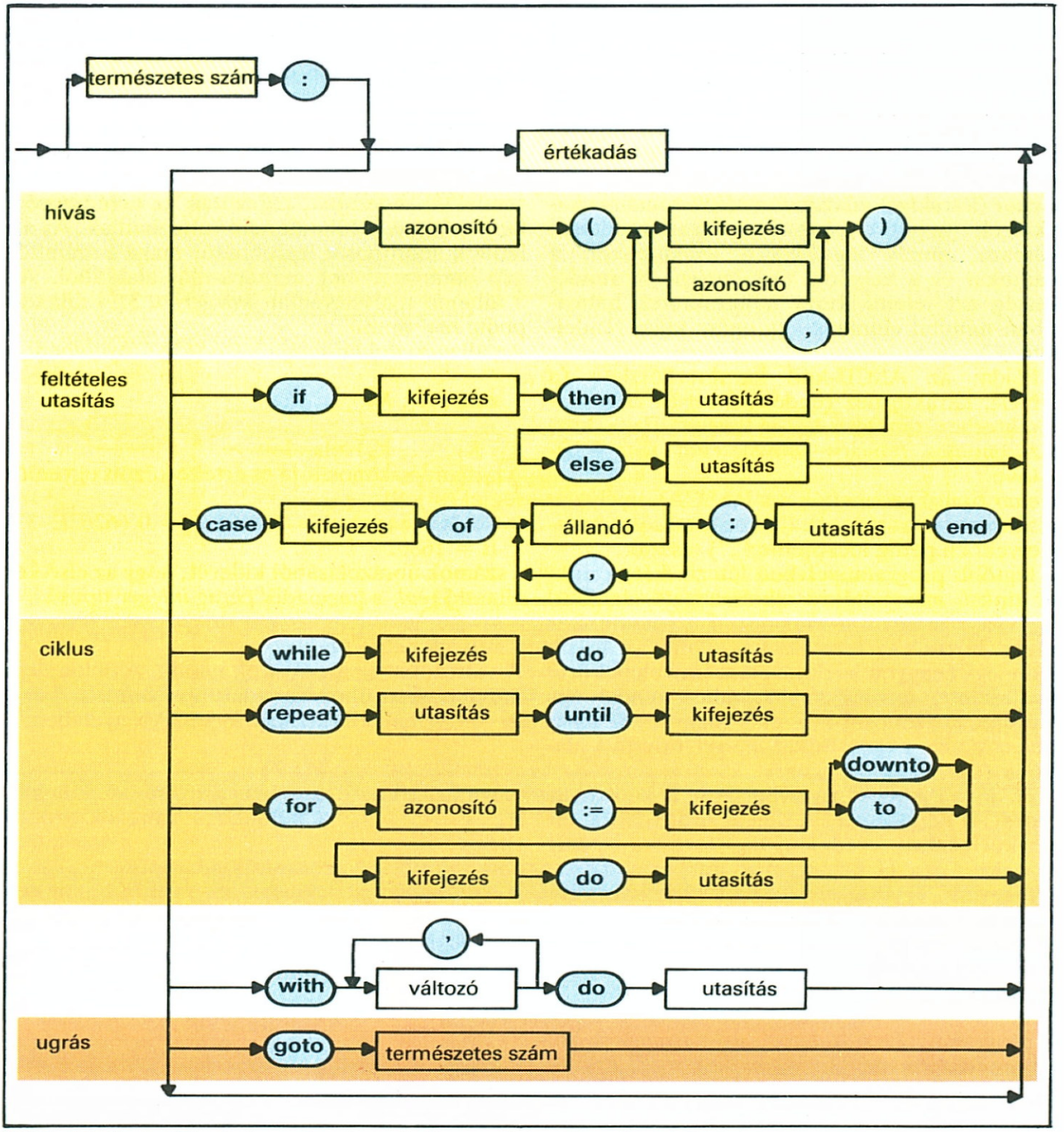
T: az előtte álló azonosítók adattípusa.

A különböző adattípusok azonosítóit külön kell deklarálni.

Példák: **var** c, x, HANS ZX81: **real;** I, E605: **integer;** Q, P: **boolean;** dtv: **char;**



Az „értékadó utasítás" és az „összetett utasítás" szintaxisdiagramja a *Pascal* nyelvben



Különböző utasítások és azok szintaxisdiagramjai a *Pascal* nyelvben

A **magyarázó szövegek** (angolul: *comments*) a programban emlékeztetőül szolgálnak a programozónak. A fordítóprogram figyelmen kívül hagyja a magyarázó szövegeket, amelyeknek a kezdetét és végét a programban természetesen jelezni kell.
A PASCAL nyelvben minden magyarázó szöveget kapcsos zárójelek közé kell írni. Ha kapcsos zárójel nem áll rendelkezésre, akkor kerek zárójelet és csillagot kell alkalmazni: (*...*).
A BASIC nyelvben a magyarázósor elejét REM, a magyarázat végét pedig a sor vége jelöli. Ha egy magyarázathoz egyetlen sor nem elég, akkor minden újabb magyarázósornak a REM jellel kell kezdődnie.

Utasítás
Az utasítások (angolul: *statements*) az adatmanipulációra vonatkozó előírások. Kétféle utasítástípus létezik: 1. azok, amelyek a tulajdonképpeni számolási feladatok elvégzését szolgálják, pl. az értékadó utasítás, függvényhívás, be- és kimeneti utasítások stb., 2. az olyan utasítások, amelyek a program futásának normális szekvenciális menetét megváltoztatják, pl. a feltétel nélküli elágazó (ugró) utasítás.
A PASCAL nyelvben minden utasítást pontosvesszővel (;) kell befejezni.
A BASIC nyelvben az utasítás vége implicitebb: minden programsor csak egyetlen utasítást tartalmazhat.

Összetett utasítás (szekvencia)
Az összetett utasításokat (angolul: *compound statement*) több olyan utasítás sorozata alkotja amelyeket közvetlenül egymás után kell végrehajtani.
A PASCAL nyelvben a szekvenciákat a **begin** és az **end** nyelvi elemek zárják közre, amelyek a legmagasabb prioritású zárójelnek felelnek meg.
Az összetett utasítások alakja:

begin S_1; S_2; ...; S_n **end;**

vagy áttekinthetőbben írva:

begin
  S_1; S_2; ...; S_n
end;

S_1; S_2; ...;S_n értékadó utasításokat jelentenek.
(Közvetlenül az **end** előtt a pontosvesszőt el lehet hagyni.)

A gyakran használt utasítások áttekintése

Az értékadó utasítás (lásd alább) olyan aritmetikai vagy logikai kifejezést tartalmaz, amelynek értékét egy változó rendeli.
A **feltételes utasítás** (143. o.) olyan értékadó utasítás, amelynek végrehajtása valamilyen logikai kifejezés értékétől függ.
A **feltétel nélküli elágazó utasítás** (ugró utasítás) (141. o.) megszakítja a szekvenciális programfutást, és az utasítások végrehajtását egy megjelölt helyen *(branch target)* folytatja.
Ciklusnak (144–147. o.) nevezzük az utasítások olyan sorozatát, amely ismételten végrehajtásra kerül.
Ha egy bizonyos alprogramot **hívunk,** akkor az alprogram végrehajtásra kerül. Ennek eredményeként vagy egy érték adódik át a hívó programnak, vagy egy korábban rögzített programrész ékelődik be a szekvenciális programlefutásba. Például be-/kiviteli műveleteket, eljárásokat, függvényeket ill. standard függvényeket lehet a programból még hívni.
A **blokk** (155. o.) olyan speciális utasítás, amely a programon belül az összetett utasításhoz hasonló zárt struktúrát alkot. Az azonosítók ekkor vagy kizárólag a blokk részére vannak deklarálva (lokális azonosítók), vagy az egész program számára (globális azonosítók), beleértve a szóbanforgó blokkot is.

Az **értékadó utasítás** (angolul: *assignment statement*) olyan elemi utasítás, amely egy aritmetikai vagy logikai művelet értékét megállapítja, és ezt követően egy – az értékadás jelétől balra feltüntetett – azonosítóhoz rendeli.
Az értékadó utasítás alakja:

$V = A,$

V: a változó azonosítója,
=: az értékadó utasítás jele.
A: operandusokból, műveletekből és zárójelekből álló kifejezés, amelynek értékét a *V* változóhoz rendeljük.
Figyelem: az értékadó utasítást nem szabad egyenletnek tekinteni, mert csak az egyik irányban olvasható: balról jobbra. Ezt a PASCAL nyelv egyértelműbben rögzíti.
Az értékadó utasítás a PASCAL nyelvben a következő alakú:

V :=A;

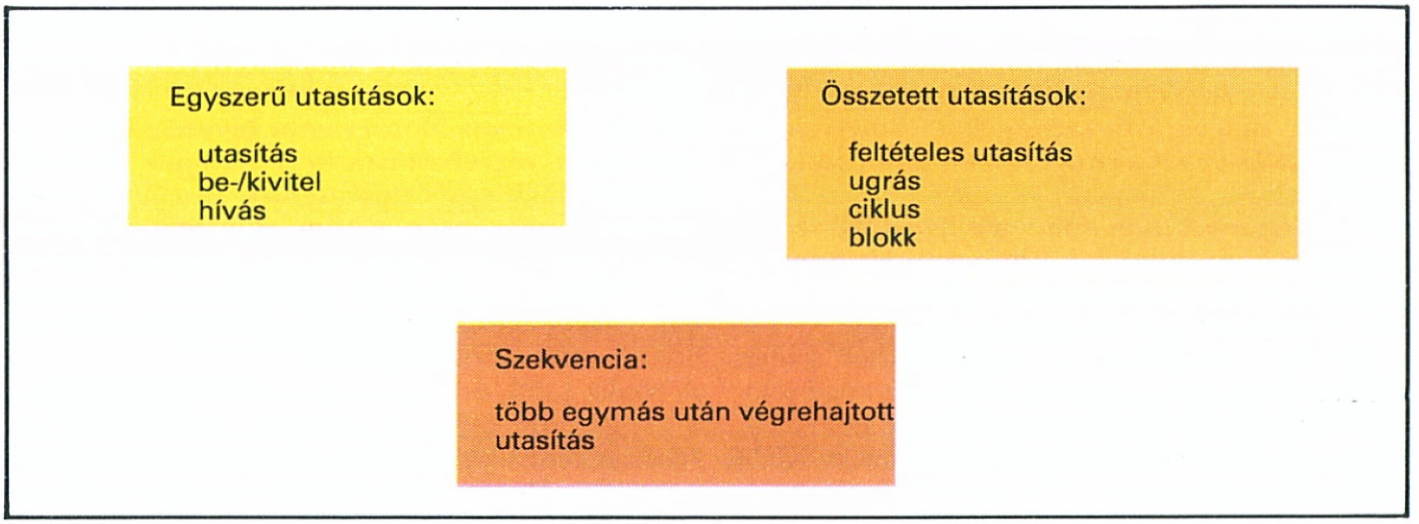
:= az értékadó utasítás jele,
; az utasítás végét jelenti.
Példák:

```
x := y * y + 1;
z := 2.349;
epsilon := 5.3E–12;
megoldás := –p/2 + sqrt ((p)/4 – q);
év := 1993;
gameover := (shipx = missilex);
```

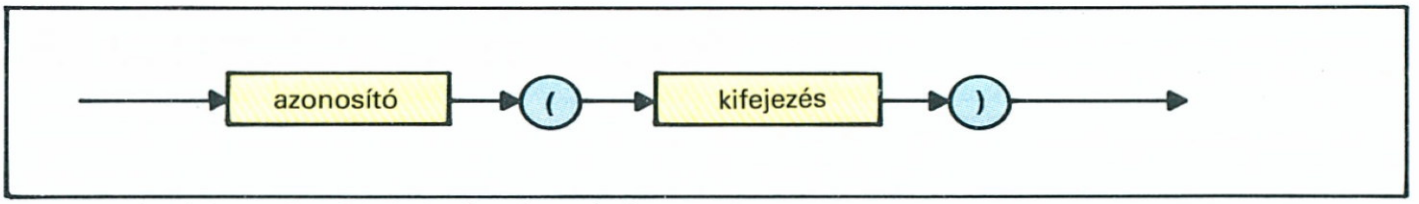
Az értékadó utasítás alakja a BASIC nyelvben:

LET $V = A$

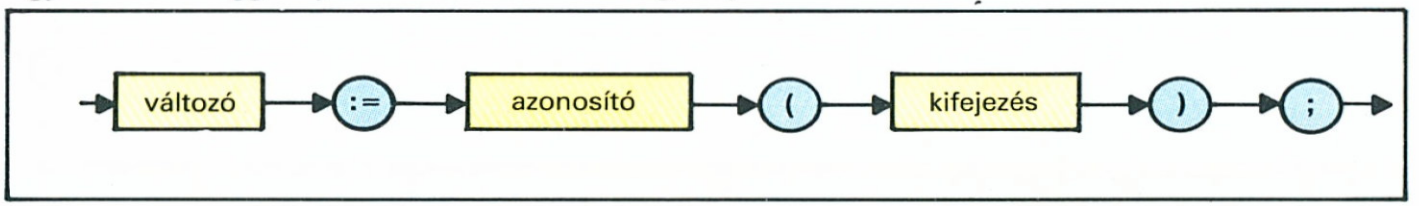
(Néhány BASIC fordítóprogram a $V = A$ alak használatát is megengedi.)



Utasítások



Egy standard függvény hívásának szintaxisdiagramja



Standard függvény hívásával megvalósuló értékadás szintaxisdiagramja *Pascal* nyelven

függvény	függvényhivatkozás		
	Pascal	*Basic*	feltétel
sin *A*	**sin** (*A*)	SIN (*A*)	*A* ívmértékben
cos *A*	**cos** (*A*)	COS (*A*)	
arc tg *A*	**arc tg** (*A*)	ATN (*A*)	$-\pi/2 < A < +\pi/2$
ln *A*	**ln** (*A*)	LOG (*A*)	$A \geq 0$
e^A, exp (*A*)	**exp** (*A*)	EXP (*A*)	
$\|A\|$	**abs** (*A*)	ABS (*A*)	*A* abszolútértéke
A^2	**sqr** (*A*)		
$+\sqrt{A}$	**sqrt** (*A*)	SQRT (*A*)	$A > 0$
csonkítás	**trunc** (*A*)	FIX (*A*)	*A* egész része
kerekítés	**round** (*A*)	RND (*A*)	*A* kerekítése a rákövetkező egész számmá
előjel-függvény		SGN (*A*)	$= -1$, ha $A < 0$ $= 0$, ha $A = 0$ $= +1$, ha $A > 0$

Gyakran előforduló standard függvények

A **standard függvények** (angolul: *intrinsic function*), vagy más néven **könyvtári függvények** az értékadó utasításokban operandusokként szerepelnek. Ezek gyakran használt függvények, amelyek hívásuk után egy változóhoz értéket rendelnek. A standard függvények azonosítói a függvény szokásos matematikai elnevezésére emlékeztetnek vagy könnyen érthető rövidítések.

Azokat a függvényeket, amelyek standard függvényként nem érhetők el a programon belül külön pl. **függvényként** (157. o.) kell definiálni.

A standard függvények nem csak aritmetikai és logikai műveleteket hajthatnak végre, hanem a számítógép belső adatait is kezelhetik, vagy pl. be-/kimeneti utasításokat is hívhatnak.

A standard függvények **hívása:**

V = a standard függvény azonosítója (A)

V: változó, amelyhez a standard függvény értéket rendel.

=: az értékadó utasítás jele.

A: a standard függvény argumentuma (angolul: *parameter*). Az argumentum pl. operandusokból, műveletekből és zárójelekből van összeállítva.

Példák:

A *szinusz* szögfüggvény. A phi változó 0.6109 értéke esetén az x = sin(phi) parancs az x változóhoz a 0,57 értéket rendeli.

A fenti függvényhívás egyenértékű az x = 0.57 értékadással.

Figyelem: a szögfüggvények argumentumát rendszerint ívmértékben kell megadni.

A *csonkítás* transzferfüggvény. Az a = 13.45 esetén az y = trunc(a) parancs az y változóhoz a 13 értéket rendeli.

Ez a standardfüggvény egy valós számhoz – a tizedespont mögött álló számjegyek levágásával – egész számot rendel, y tehát *integer* típusú változó.

A *kerekítés* transzferfüggvény. Ha a = 8.767, akkor a z = round(a) parancs a z változóhoz a 9 értéket rendeli. A z változó *integer* típusú.

Példa PASCAL nyelven: x := **sin***(phi)*;

Példa BASIC nyelven: X = SIN(PHI).

A **be-/kiviteli függvények** ill. utasítások (angolul: *inpout/output statements*) a számítógép és a külvilág közötti kapcsolatot valósítják meg. Adatokat olvasnak be és az eredményeket adják ki. A programnyelvek és a hozzájuk tartozó fordítóprogramok számos be- és kiviteli függvényt tartalmaznak.

Az alábbiakban csak a megfelelő standard függvényeket fogjuk bemutatni. Ebben az esetben a beviteli egység a billentyűzet, az adatkivitelre szolgáló egység pedig a képernyő.

A **beviteli függvény** általános alakja:

read(V_1, V_2,...,V_n)

Hatása:

A beviteli készülék beolvasása a billentyűzeten beadott első értéket és azt a V_1 változóhoz rendeli, majd a második beadott értéket olvassa be, amelyet a V_2 változóhoz rendel stb. Az utolsó érték beolvasása és a V_n változónak történő értékadás után a beolvasási művelet befejeződik.

Példák PASCAL nyelven:

read(ev,honap,nap).

Ha az 1933, a 10 és a 22 számok kerülnek bevitelre, akkor a következő értékadások történnek meg: ev := 1933; honap := 10; és nap := 22.

Példák BASIC nyelven:

INPOUT (EV, HONAP, NAP).

Az értékadások a PASCAL nyelven bemutatott példához hasonlóan történnek.

A **kiviteli** függvény általános alakja:

write(V_1, V_2,...,V_n)

Hatása:

Az adatkiviteli készülék a V_1 változóval jelölt tárolócella tartalmát megjeleníti, amelyet egy üres hely követ, majd a V_2 változóval jelölt érték megjelenítése következik stb. Legutoljára a V_n változó kerül sorra, ezzel a folyamat befejeződik. Minden szám egymás után, egyetlen sorban, helyközzel elválasztva jelenik meg. Ha a megjelenítésre egy sor nem elegendő, akkor az adatkivitel a következő sorban folytatódik.

Egy újabb kiviteli utasítás hatására megjelenő első jel egy új sor elején helyezkedik el.

Példa PASCAL nyelven:

writeln(ev,honap,nap).

A korábbi példa adataival a kiviteli egységen a következő számok jelennek meg: 1933 10 22.

Példa BASIC nyelven:

WRITE(EV, HONAP, NAP).

Ennek hatására a kiviteli egységen a következő lesz látható:

19331022

A *write* függvény argumentuma felső vesszők vagy idézőjelek közé tett tetszőleges karaktereket is tartalmazhat.

Példa PASCAL nyelven:

writeln('szuletett:',ev,honap,nap).

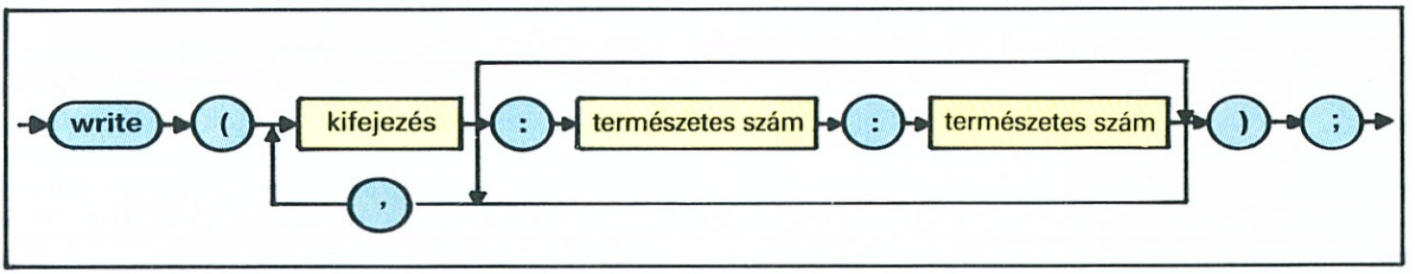
Ennek hatására az adatkiviteli készüléken a következő jelenik meg:

szuletett:1933 10 22.

Példa BASIC nyelven: WRITE „SZULETETT":EV,HONAP,NAP.

Ennek hatására az adatkiviteli készüléken a következő jelenik meg:

SZULETETT:19331022.

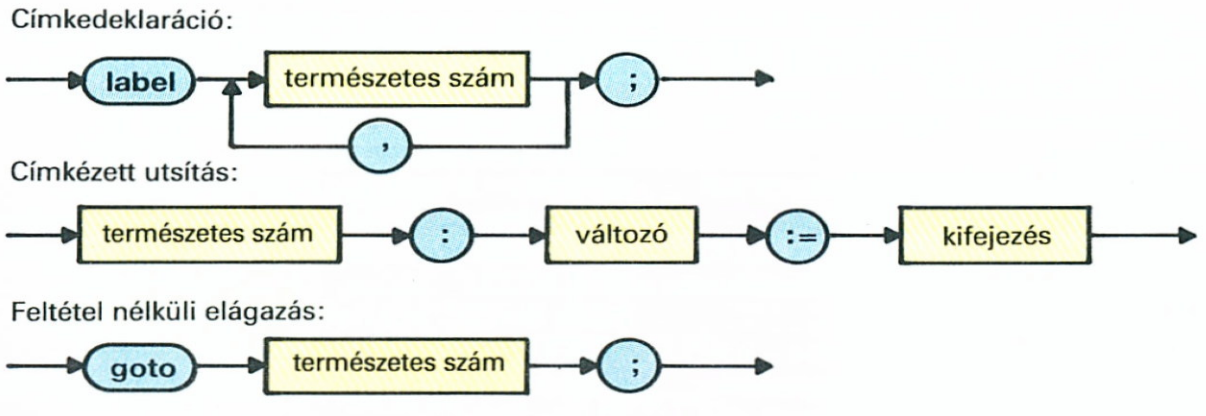


Egyszerű adatkivitel szintaxisdiagramja a *Pascal* nyelvben

Példa:
x = 1234,56789
y = -0,0001234

Utasítások:	Adatkivitel:		
write (x, y);	1.23456E+03	-1.2340E-04	standard adatkivitel
writeln (x, y);	1.23456E+03	-1.2340E-04	
write (x:10:3, y:15:5);	1234.568	−0.00012	formázott adatkivitel
writeln (x:10:3); **write** (y:15:5);	1234.568 −0.00012		
write (x:10:3);	1234.568		

Standard és formázott adatkivitel a *Pascal* nyelvben



Ugró utasítással kapcsolatos szintaxisdiagramok a *Pascal* nyelvben

```
110 PELDA UGRO UTASITASRA A BASIC NYELVBEN
120 PRINT       „A KOR SUGARA  TERULETE"
130 INPUT R                                   ugró utasítás célpontja
140 FL = 3.142*SQR (R)
150 PRINT R, FL
160 GOTO 130                                  ugró utasítás
```

2.3, 7.18, 22.3 és 51.1 értékek bevitele után a kimenő adatok:

A KOR SUGARA	TERULETE
2.3E+00	1.662118E+01
7.18E+00	1.619776E+02
2.23E+01	1.562485E+03
5.11E+01	8.204422E+04

Ugró utasítás egy *Basic* programban

Egyszerű formázott adatkivitel függvények

Dekadikus számok beolvasásakor a számok formális ábrázolása legtöbbször nem játszik szerepet.
A számok megjelenítésére két lehetőség kínálkozik: vagy előre rögzítünk egy bizonyos ábrázolási módot, vagy a számok standard ábrázolási formában jelennek meg.
Egy szám **standard ábrázolása,** a képernyőn pl. attól függően különböző, hogy egész vagy valós számot kell-e megjeleníteni.

Integer adattípus esetén: a szám előjellel együtt jelenik meg (a pozitív előjel nem kerül kijelzésre), és a számot egy helyköz követi.
Példák: –22, 123456, 1

Real adattípus esetén: a szám lebegőpontos formában, tíz hatványaként jelenik meg, a kitevőben két számjegy szerepel. A szám előtt álló pozitív előjel nem kerül kijelzésre, a kitevőben azonban az előjel minden esetben szerepel. A megjelenített szám rendszerint kerekített. A számok jobbra ütköztetve jelennek meg, a számjegysorozat végén álló zérusokat elhagyjuk.
Példák:
7.00005E+13 –6.67204E–11 2.24000E+01

A számok **formázott ábrázolását** a számjegyek teljes száma és a tizedespont két oldalán álló jegyek száma határozza meg.
Ebben az ábrázolási módban a megjelenítésre különböző függvények állnak a programozó rendelkezésére.
Pl. a PASCAL nyelvben egy lebegőpontos szám egyszerű megjelenítése:

writeln(*A:m:n*);

A: kifejezés, amely a megjelenítésre kerülő számot ábrázolja,
m: az *A* megjelenítésére rendelkezésre álló pozíciók száma,
n: a tizedespont után álló jegyek száma.

Példa: a **writeln**(pi:10:7,p/e:12:3); utasítás a következő nyomtatási képet adja eredményül:
3.1415926
–0.003

Ha a **writeln**(...); utasítást egy további követi, akkor a kivitelre kerülő újabb adatok a következő sorban jelennek meg.
Ha az utasítás befejezése után nem kell soremelést végrehajtani, akkor a **writeln** utasítást a **write** utasítással kell helyettesítenünk.

Példák:

```
writeln(pi:10:7);
writeln(p/e:12:3).
```

A monitoron megjelenő kép:

```
3.1415926
     -0.003
```

```
write(pi:10:7);
write(p/e:12:3).
```

A monitoron megjelenő kép:
3.1415926 –0.003

Címkézett utasítás
A **címkék** (angolul: *labels*) egyes utasításokat jelölnek meg; a címkéket általában deklarálni kell.
A program vezérlését *ugró utasítással* (lásd alább) csak címkézett utasításra lehet átadni.
A címkézett utasítás alakja

M: S

M: természetes szám vagy tetszőleges jelsorozat.
S: utasítás.
A programon belül nem fordulhat elő két azonos címke.

Példa PASCAL nyelven:
M24: y := 7 * x + z * (sin(phi/2));

A BASIC sororientált nyelv. Minden egyes sor egy természetes szám formájában címkét is visel (de a szám után elmarad a PASCAL nyelvű példában látható kettőspont).

Ugró utasítás (elágazás)
Ha a program utasításainak szekvenciális feldolgozási módját meg kell szakítani, és az utasítások végrehajtását a program egy másik helyén kell folytatni, akkor ezt az **ugró utasítás** (angolul: *goto statement*) segítségével lehet végrehajtani.
Az ugró utasítás alakja:

goto M

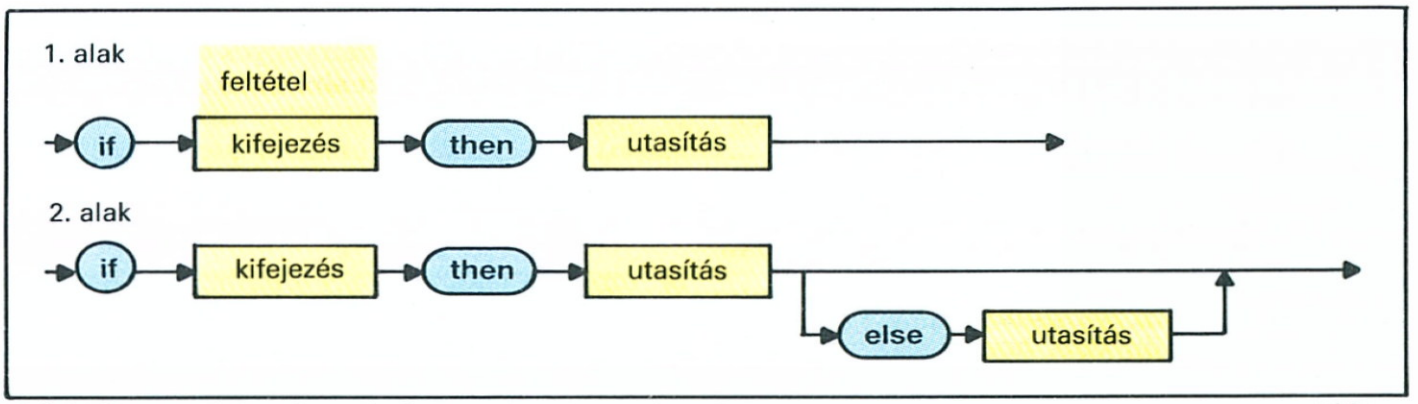
M: az ugrás célját jelölő címke.
Itt **feltétel nélküli elágazásról** van szó, tehát az utasítás további vizsgálódás nélkül végrehajtásra kerül, mihelyt a program futása a *goto M* utasításhoz ér.

A *feltételes ugró utasítások* (*feltételes elágazások,* 143. o.) egy logikai kifejezést tartalmaznak. A program végrehajtásában az ugrás csak akkor következik be, ha a feltétel teljesül.

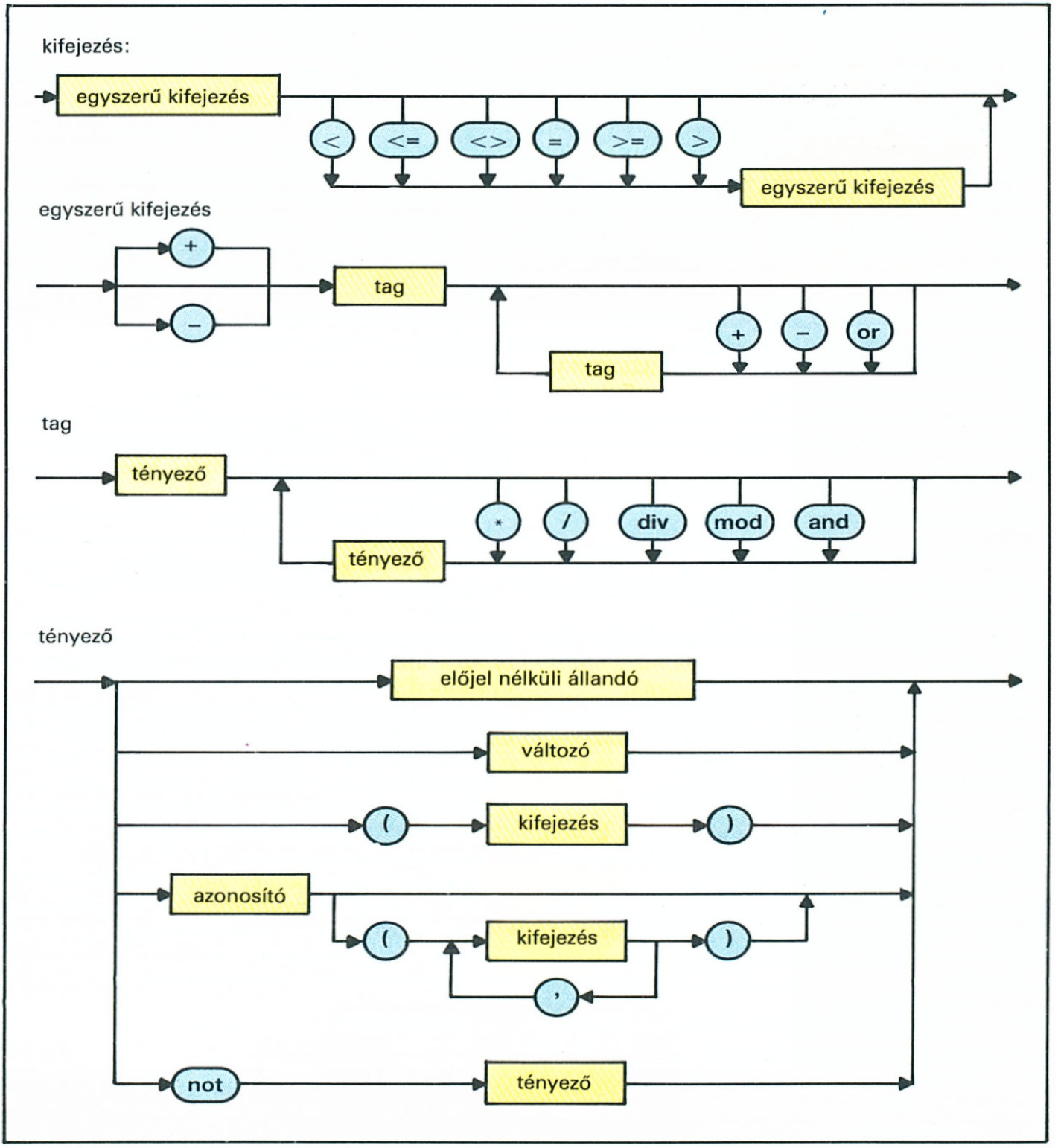
Az ugró utasítás végrehajtása után a számítógép először a célutasítást, majd az utána következő utasításokat dolgozza fel.
Figyelem: az ugrás nem történhet kívülről egy ciklus vagy egy eljárás (159. o.) belsejében.

A feltétel néküli ugrásra (**goto** *M*) a PASCAL nyelvben is van lehetőség, alkalmazását azonban a PASCAL nyelv alkotói nem tanácsolják, mert megnehezíti a strukturált program olvasását és megértését.

A BASIC nyelvben az ugró utasítás (GOTO M) gyakran megkönnyíti a programírást. A BASIC-program olvashatósága vagy érthetősége ezáltal alig változik.



Feltételes utasítás szintaxisdiagramja a *Pascal* nyelvben



Feltételek szintaxisdiagramjai a *Pascal* nyelvben

A **feltételes utasítás** (angolul: *if statement*) olyan vezérlő utasítás, amely a program lefutási sorrendjének adott helyen – előre megadott feltételtől függő – megváltoztatására szolgál. A feltételes utasítás a program menetében megkülönböztetéseket, feltételes ugrásokat és elágazásokat tesz lehetővé.

A feltételes utasítás 1. alakja

if *B* then *S*

B: a feltétel (lásd alább). Ez olyan kifejezés, amelynek értéke *igaz* vagy *hamis* lehet.
S: egyszerű, összetett vagy feltételes utasítás.
A feltételes utasítás hatása: ha a *B* feltétel teljesül, vagyis ha a kifejezés értéke *igaz,* akkor az utasítást a számítógép végrehajtja. Ha a feltétel nem teljesül, vagyis ha a kifejezés értéke *hamis,* akkor a számítógép az *S* utasítást átugorja és a közvetlenül utána álló utasítást hajtja végre.

Példa a PASCAL nyelvből:

```
if z1 > z2 then goto M;
if z3 > z2 then max := z3;
max := z2;
write (max, 'a legnagyobb szam');
```

A feltételes utasítás 2. alakja

if *B* then S_1 else S_2,

ahol
B: a feltétel (lásd alább).
S_1, S_2: egyszerű, összetett vagy feltételes utasítások.
Hatása: ha a *B* feltétel teljesül, akkor a számítógép az S_1 utasítást végrehajtja, az S_2 utasítást pedig átugorja, tehát a program futása az S_2 után következő utasítással folytatódik. Ha a feltétel nem teljesül, vagyis ha a *B* kifejezés értéke *hamis,* akkor a számítógép az S_2 utasítást hajtja végre. Ezután a program futása sorban folytatódik.

Példa a PASCAL nyelvből:

```
if z1 ≥ z2 then max := z1 else max := z2;
```

Ha a leírásban zárójelek nem szerepelnek, akkor a számítógép az első utasítást belülről kifelé haladva hajtja végre.

Példa:
if B_1 then if B_2 then S_1 else S_2
a következő zárójeles utasításnak felel meg:
if B_1 then (if B_2 then S_1 else S_2)

Néhány programnyelv kettőnél több irányba történő elágazást is megenged.

Példa az ADA nyelvből:
if B_1 then S_1 elsif B_2 then S_2 elsif B_3 then S_3
elsif B_4 then S_4 ... else S_n end if;

A **feltételek** operandusok, műveletek és zárójelek összekapcsolásával előállított kifejezések.
Az operandusok állandók, változók vagy függvények lehetnek. Az aritmetikai műveleteken (pl. +, −, *, /) kívül relációs műveletek (pl. >, <, =, ≠) és logikai operátorok (pl. △, ▽, ¬) is szerepelhetnek a feltételben.

Ha a feltétel logikai műveleteket is tartalmaz, akkor a feltételt zárójelbe kell tenni.

Az adott feltétel kifejezésének értéke a két Boole-állandó egyike, tehát *igaz* (angolul: *true*) vagy a *hamis* (angolul: *false*) lehet.

Példa:
x = 23 és y = 34 esetén az (x > y) feltétel értéke *hamis,* az (x < y) feltétel értéke ellenben *igaz.*

Ha a feltételen belül az *igaz* és a *hamis* Boole-állandókat logikai operátorok kötik össze, akkor az eredmény a megfelelő **igazságtáblázatból** olvasható ki.

A 48. oldalon szereplő igazságtáblázatokban (műveleti táblázatokban) pl. az *1* érték az *igaz,* a *0* érték pedig a *hamis* Boole-állandónak felel meg.

Példa:
az ÉS operátor (angolul: **and**) a PASCAL nyelvben:

hamis **and** *hamis* → *hamis*

hamis **and** *igaz* → *hamis*

igaz **and** *hamis* → *hamis*

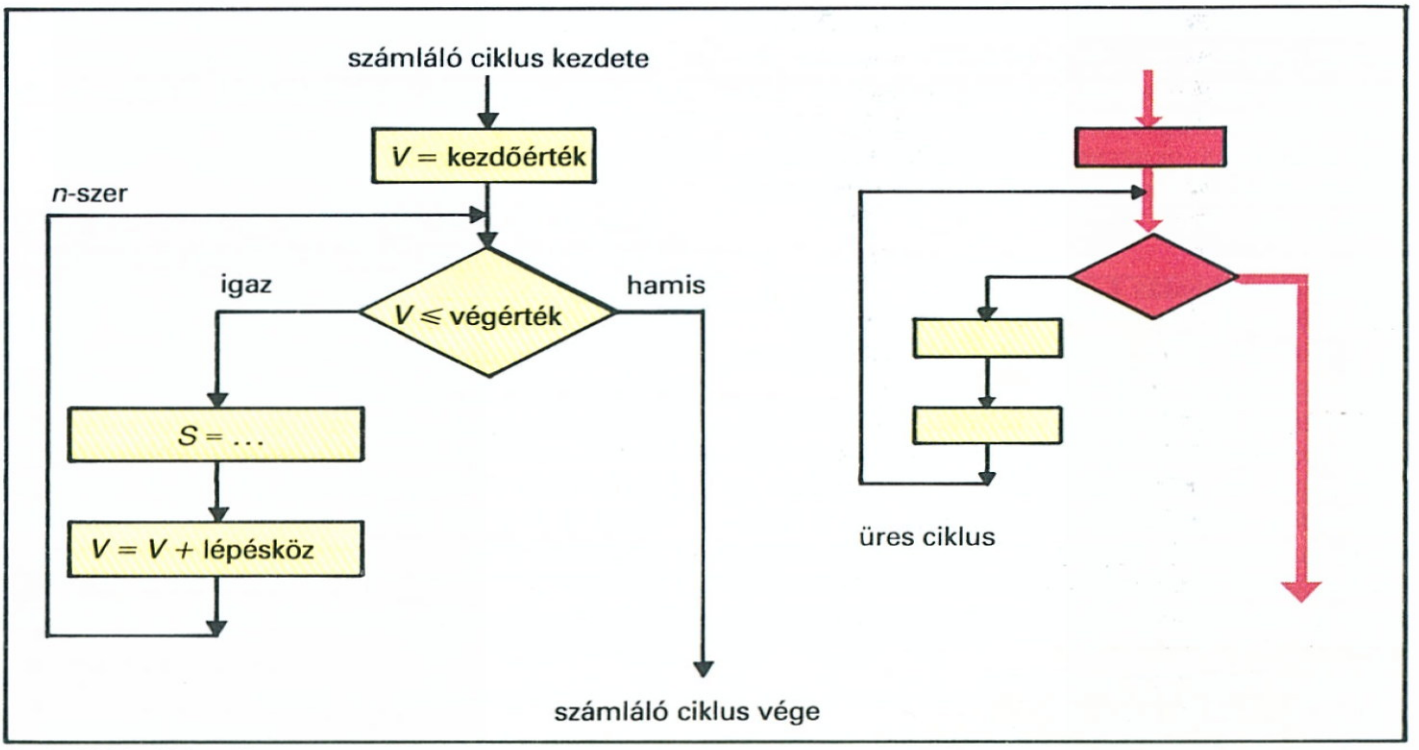
igaz **and** *igaz* → *igaz*

A megfelelő igazságtáblázat a következő:

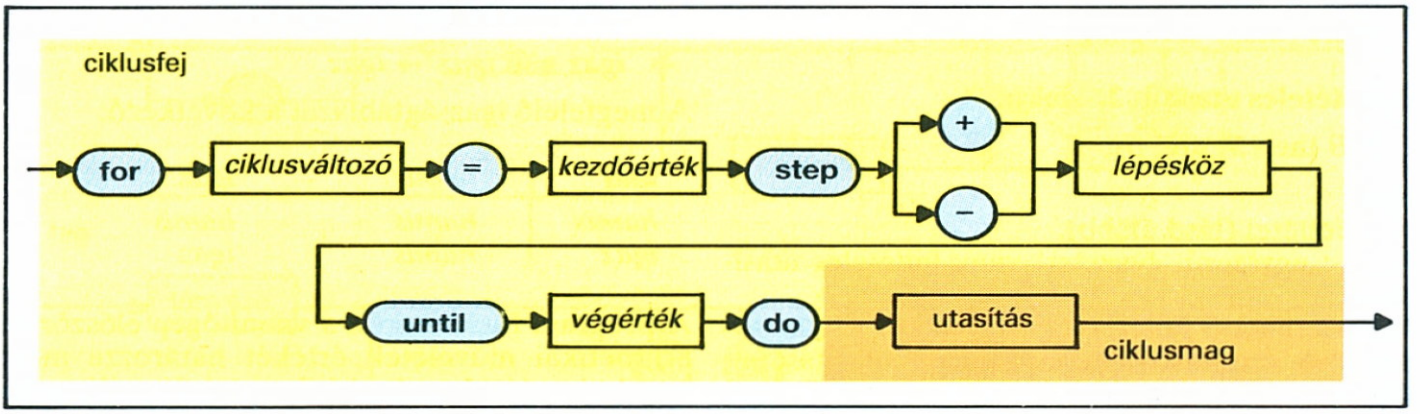
and	*hamis*	*igaz*
hamis	*hamis*	*hamis*
igaz	*hamis*	*igaz*

A program futása során a számítógép először az aritmetikai műveletek értékét határozza meg, ezután a relációs műveleteket, végül pedig a logikai műveleteket végzi el.

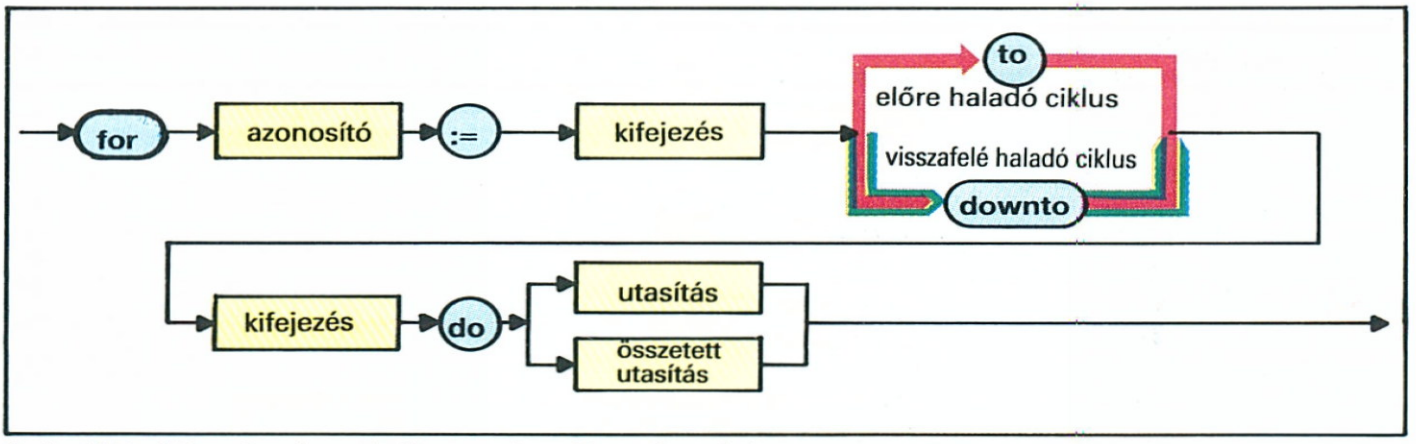
Példa a PASCAL nyelvből:
Feltétel: (szam < 100) **and** (szam < = 200)
Ha szam = 51, akkor a zárójel tartalma (*igaz* **and** *igaz*). Ennek a logikai műveletnek az eredménye az igazságtáblázat szerint *igaz.* A feltétel tehát teljesül.



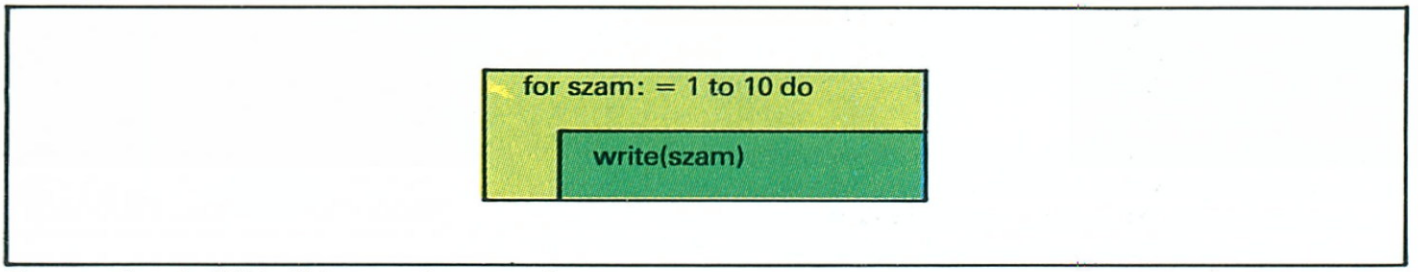
Számláló ciklusok



Számláló ciklusok általános szintaxisdiagramja



Számláló ciklusok szintaxisdiagramja a *Pascal* nyelvben



Egyszerű számláló ciklus struktogramja

A **ciklusok** (angolul: *loops*) a programon belül egyszerű és összetett utasításokat, valamint függvényeket ismételek mindaddig, amíg egy előre meghatározott kilépési feltétel teljesül. Háromfajta ciklus létezik, amelyek a ciklus befejezésének kritériumaiban különböznek egymástól.

A *számláló ciklus* előre megadott számú ismétlés után fejeződik be. Ezt követően a program utasításainak végrehajtása sorrendben folytatódik.

A *feltételes ciklusok* két fajtája akkor fejeződik be, ha egy – előre rögzített – kilépési feltétel teljesül. A kilépési feltétel teljesülésének vizsgálata a cikluson belül két helyen történhet: a tulajdonképpeni ciklus végrehajtása előtt vagy a ciklusmagon belül a ciklus végén (lásd alább). Az egyes ciklusfajták nagyon különböző hatásokat idézhetnek elő.

A feltételes ciklusok programozásakor mindig fennáll a veszélye annak, hogy a program **végtelen ciklusba** kerül. Ha ugyanis a kilépési feltétel soha nem teljesül, akkor a program futása a ciklusnál megakad.

Számláló ciklus

A számláló ciklus általános alakja (ciklusutasítás):

for $V = A_1$ step A_2 until A_3 do S

V: a ciklusváltozó, amely *csak* cikluson belül van definiálva.

A_1: V_1 kezdőértéke, amely tetszőleges kifejezés lehet.

A_2: a lépésköz, amely előre haladó ciklus esetén pozitív, visszafelé haladó ciklus esetén pedig negatív érték.

A_3: V végértéke.

S: az ismétlődő egyszerű vagy összetett utasítás, amelyet *ciklusmagnak* nevezünk. A *for* és a *do* közötti utasítások a *ciklusfejet* alkotják.

A ciklus hatása (előre haladó ciklus esetén) a következő: a program a V ciklusváltozóhoz az A_1 kezdőértékét rendeli, ezt az értéket felhasználva végrehajtja az S utasítást, majd visszaugrik a ciklus kezdetéhez. Ezután a ciklusváltozó értékét az A_2 lépésközzel megnöveli. Az új értékkel ismét végrehajtja az S utasítást és újból visszaugrik a ciklus kezdetéhez. A program a ciklusmagba való minden egyes belépés *előtt* ellenőrzi a ciklusfejben, hogy még fennáll-e a $V <$ végérték feltétel. Amíg az ellenőrzés eredménye *igaz,* az S utasítás ismételten végrehajtódik. Ha az ellenőrzés eredménye *hamis,* a ciklus befejeződik, és a program végrehajtása a ciklusmag után következő utasítással folytatódik.

Az ismétlések n számát nyilvánvalóan már a ciklusfejben rögzítettük:

$n = \text{abs}(\text{trunc}(A_3 - A_1)/A_2))$.

Figyelem: nem szabad ugró utasításnak kívülről a ciklusmagba vezetnie; azonban a ciklust a ciklus belsejében szereplő ugró utasítással el lehet hagyni.

Üres ciklusról akkor beszélünk, ha előre haladó ciklus esetén a kezdőérték > végérték, vagy visszafelé haladó ciklus esetén a kezdőérték < végérték feltétel teljesül.

A számláló ciklust legtöbbször az adatmező-elemek (151. o.) indexeinek szisztematikus megváltoztatására használjuk. Mivel a ciklusfejben rögzítjük, hogy a ciklusmagot hányszor kell megismételni, ebben az esetben végtelen ciklus nem fordulhat elő.

Példák:

számláló ciklusok a PASCAL nyelvben.

Előre haladó ciklus:

```
for V := A1 to A3 do
begin
  S1; S2; ...; Sn
end;
```

Visszafelé haladó ciklus:

```
for V := A1 downto A3 do
begin
  S1; S2; ...; Sn
end;
```

Ha a ciklusmag csak egyetlen utasításból áll, akkor a **begin** és az **end** elhagyható.

A lépésköz értéke 1. [Ha a lépésköz ≠ 1, akkor a ciklust a while utasítás (147. o.) segítségével kell programozni.]

Egyszerű számpélda a PASCAL nyelvből:

(A példában számokat kell kinyomtatni 1-től 10-ig.)

```
for szam := 1 to 10 do writeln (szam:3);
```

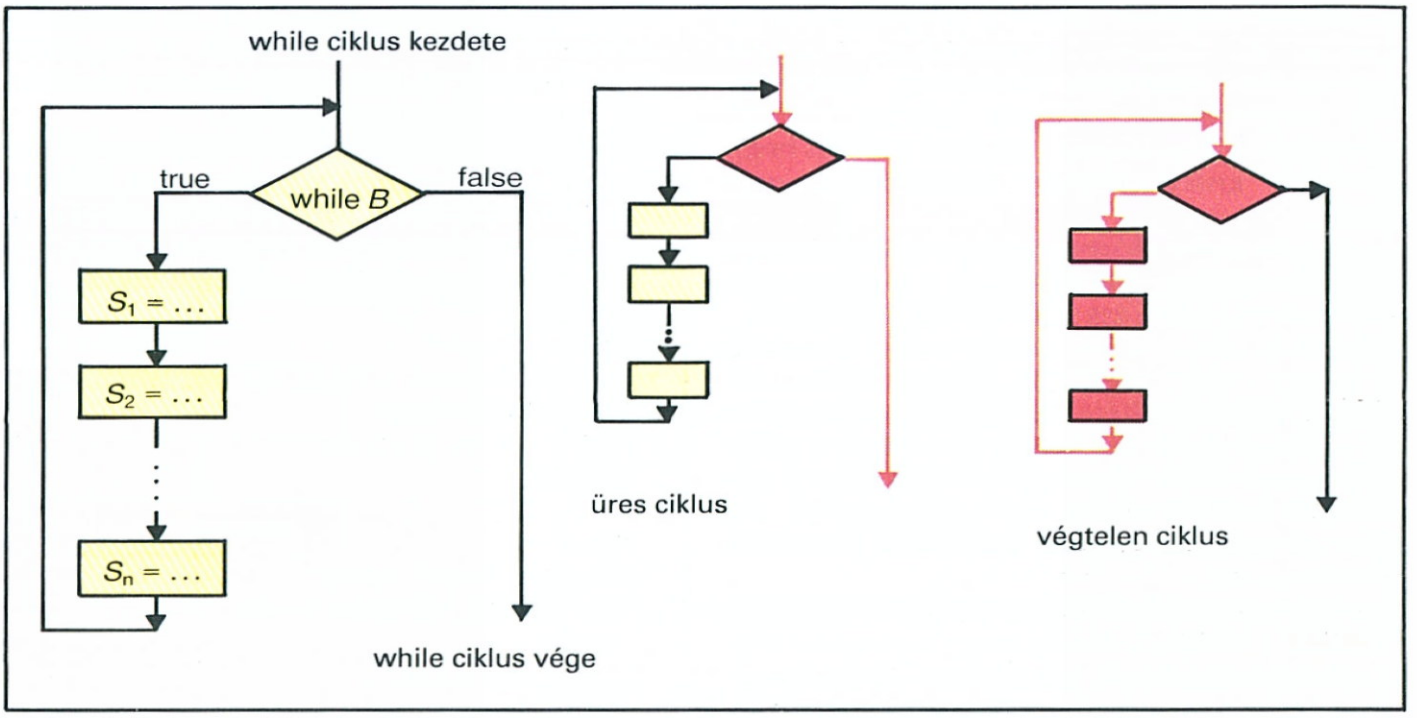
A nyomtatásban megjelenő eredmény:

```
 1  2  3  4  5  6  7  8  9  10
```

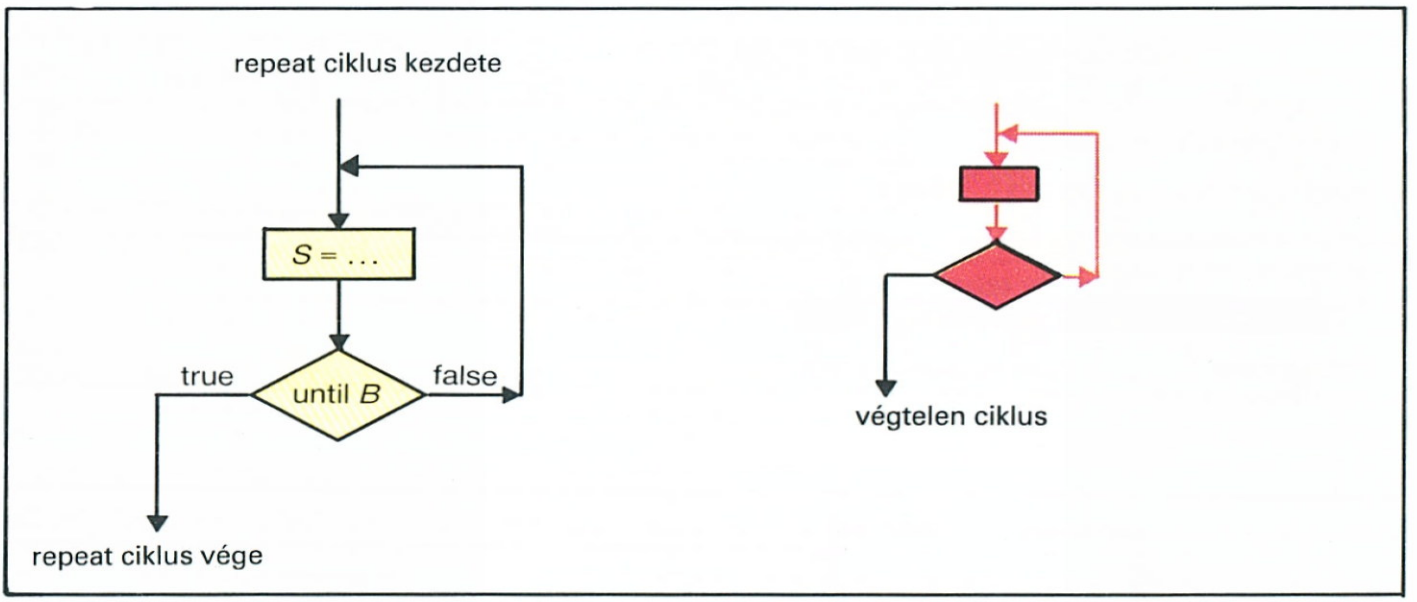
Számláló ciklus a BASIC nyelvben:

```
FOR V = I1 TO I3 STEP I2
S1
S2
 .
 .
 .
Sn
NEXT V
```

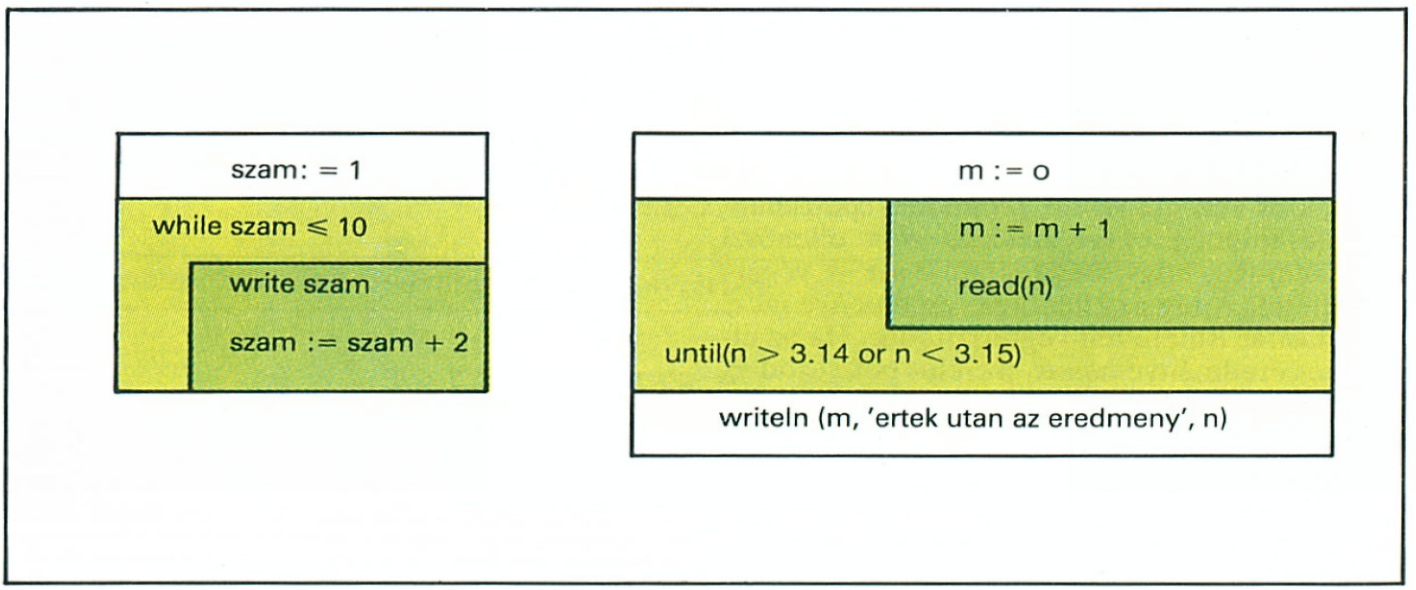
Az I_1 kezdőérték, az I_3 végérték és az I_2 lépésköz *integer* típusú adatok. Ha a lépésköz negatív (tehát $-I_2$), akkor visszafelé haladó ciklusról van szó. Ha a STEP utasítás hiányzik, akkor I_2 értéke automatikusan 1, vagyis a STEP hiányának jelentése STEP 1.



while ciklus



repeat ciklus



Egyszerű while és repeat ciklusok struktogramja

Feltételes ciklusok

A feltételes ciklusoknak két fajtáját különböztetjük meg aszerint, hogy a ciklusból való kilépés feltételének ellenőrzése a ciklus melyik részén történik:

A **while ciklus** a kilépési feltételt a tulajdonképpeni ciklusutasítás (a ciklusmag) végrehajtása *előtt* ellenőrzi, és addig ismétli önmagát, amíg a feltétel *már nem* teljesül.

A *while* utasítás általános alakja:

while *B* do *S*

B: a feltétel; olyan kifejezés, amelynek értéke *igaz* vagy *hamis* lehet.

S: a ciklusmag; az ismételten végrehajtásra kerülő egyszerű vagy összetett utasítás.

A while ciklus hatása: ha a program futása során a *while* utasítást eléri, akkor megvizsgálja a *B* feltételt. Ha a vizsgálat eredménye *igaz*, a program végrehajtja az *S* utasítást. Ezt követően a program ismét a *while* utáni pontra ugrik. Ez a folyamat addig ismétlődik, amíg a *B* feltétel vizsgálatának eredménye *hamis* értékre nem vált. Ekkor a program átugorja az *S* utasítást és sorrendben folytatja a további programrészek végrehajtását.

Üres ciklushoz akkor jutunk, ha a *B* feltétel kezdettől fogva *hamis* értékű. Ha ezzel szemben *B* soha nem veszi fel a *hamis* értéket, akkor a program **végtelen ciklusba kerül.**

Példa: while utasítás a PASCAL nyelvben.

```
while B do
begin
  S1; S2; ...; Sn
end;
```

Ha a ciklusmag csak egyetlen utasítást tartalmaz, akkor a **begin** és az **end** elhagyható.

A PASCAL nyelvből vett egyszerű számpélda, amelyben a lépésköz 2:

```
  szam := 1;
while szam < = 10 do
begin
  write (szam:3);
szam := szam + 2;
end;
```

A nyomtatásban megjelenő eredmény:

1 3 5 7 9

A BASIC nyelvben nincs while utasítás. Ugyanehhez az eredményhez jutunk azonban feltételes utasítás és számláló ciklus (vagy ugró utasítás) kombinációjának segítségével is.

A **repeat ciklus** a kilépési feltételt csak a tulajdonképpeni ciklusutasítás (a ciklusmag) végrehajtása *után* ellenőrzi, és mindaddig ismétli önmagát, amíg a feltétel *már teljesül* (a feltétel teljesülésekor a ciklus véget ér). A repeat utasítás általános alakja:

repeat *S* until *B*

S: a ciklusmag; az ismételten végrehajtásra kerülő egyszerű vagy összetett utasítás.

B: a feltétel; olyan kifejezés, amelynek értéke *igaz* vagy *hamis* lehet.

A repeat utasítás hatása: ha a program futása során a *repeat* utasításhoz ér, akkor végrehajtja az *S* utasítást, majd ezt követően megvizsgálja a *B* feltételt.

Ha a vizsgálat eredménye *hamis* értéket ad, a program visszaugrik az *S* utasításhoz, azt újra végrehajtja majd ismét megvizsgálja a *B* feltételt. Ez a folyamat addig ismétlődik, amíg a soron következő vizsgálatnál a *B* feltételre *igaz* érték adódik. Ekkor a számítógép sorban folytatja a további programrészek végrehajtását.

A while utasítással szemben a repeat utasítás végrehajtása során sohasem fordulhat elő üres ciklus. Végtelen ciklus azonban ebben az esetben is lehetséges.

A repeat utasítást elsősorban akkor használjuk, ha biztosak akarunk lenni abban, hogy a számítógép valamilyen ciklusbeli utasítást legalább egyszer végrehajt.

Példa: repeat utasítás a PASCAL nyelvben.

```
repeat
S1; S2; ....; Sn;
until B;
```

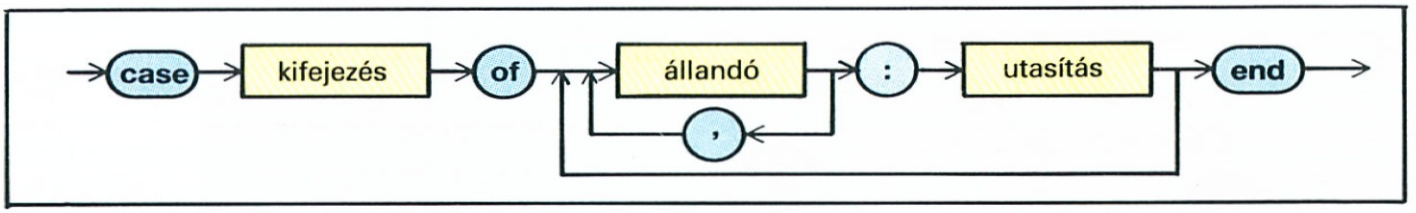
A repeat ciklusban az S_1; S_2; ...; S_n utasításokat nem szükséges **begin** és **end** közé zárni.

A PASCAL nyelvből vett egyszerű számpélda:

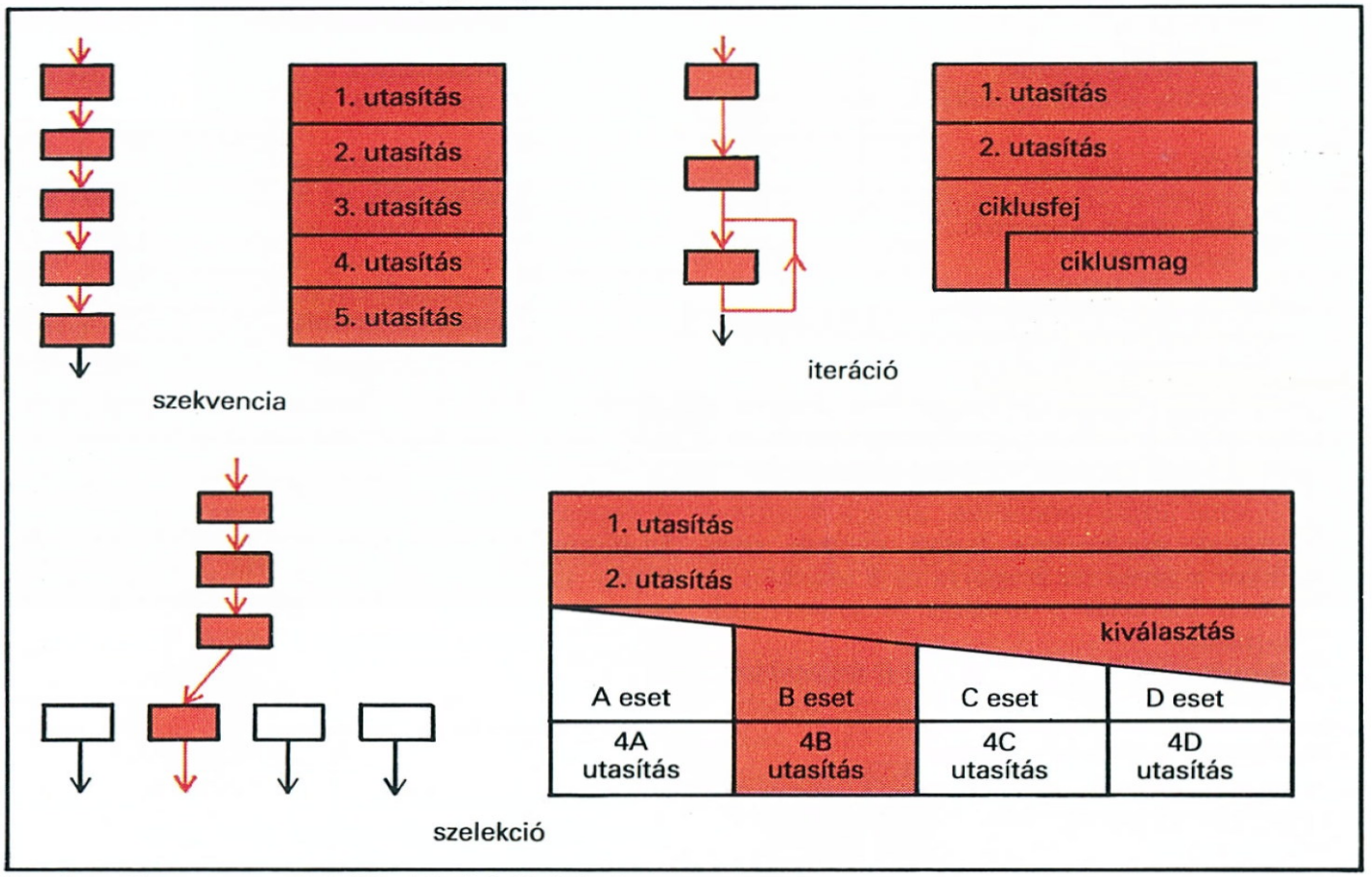
(a feladatban egy tetszőleges hosszúságú számsorozatot kell beolvasni, amíg egy n < 3,15 szám megjelenik. Ezt a számot valamint a beolvasott elemek számát ki kell nyomtatni.)

```
  m : = 0;
  repeat
    m : = m + 1 ;
    read (n);
  until n < 3.15;
  writeln (m, 'ertek utan az eredmeny', n);
```

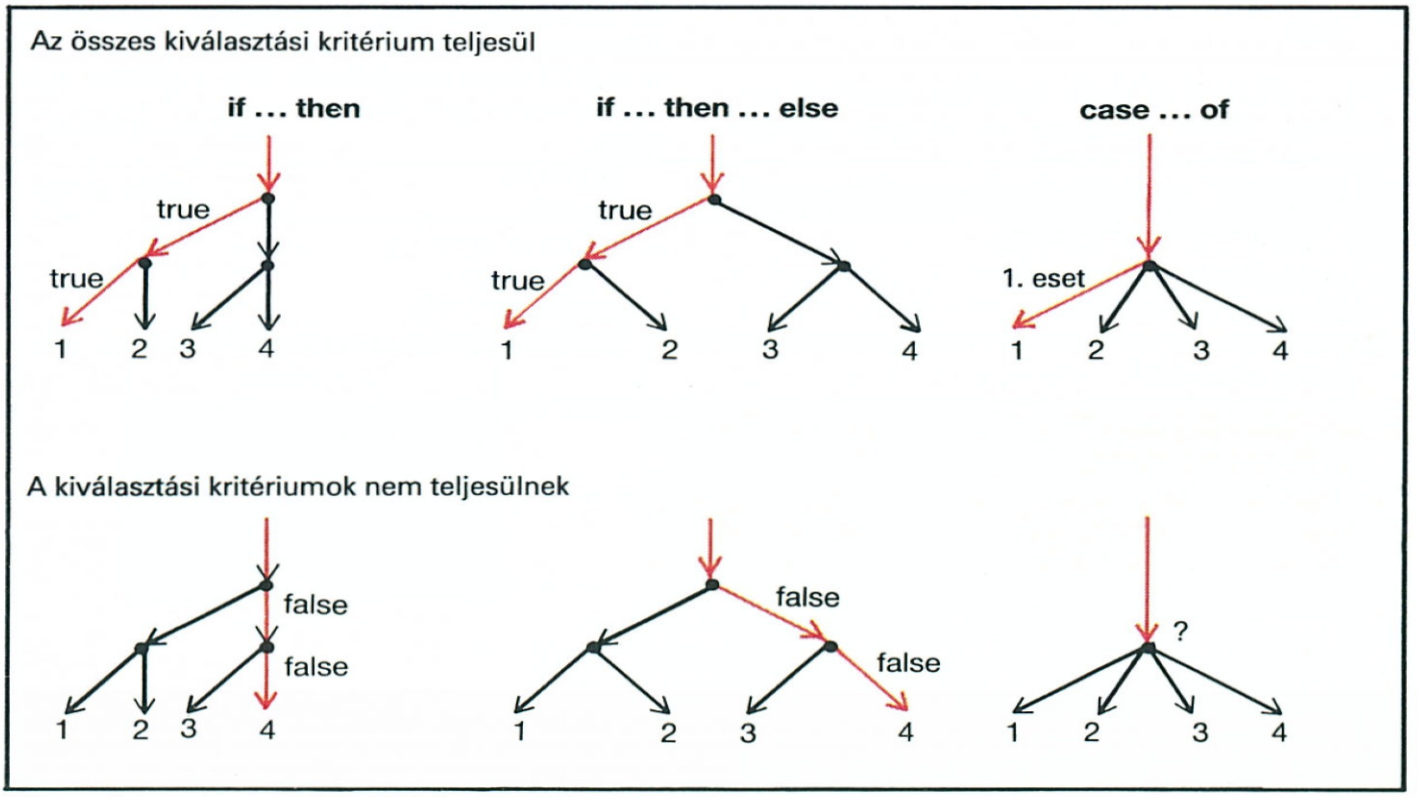
A BASIC nyelvben nincs repeat utasítás.



Kiválasztási utasítás szintaxisdiagramja a *Pascal* nyelvben



Strukturált programok különböző felépítései



Három lehetséges szelekció lefutása a *Pascal* nyelvben

A program szekvenciális sorrendben történő lefutását többek között feltételes utasításokkal (143. o.) lehet megváltoztatni. Ez az egyszerű lehetőség azonban gyakran nem elegendő, és következménye csak a feltételes utasítások egymásba ágyazott, áttekinthetetlen szövevénye lenne. Ebben az esetben a szelekciós utasítások mutatnak kiutat.

Szelekciós utasítás

A szelekciós utasítás olyan vezérlést valósít meg, amely a program lefutásában többszörös elágazást tesz lehetővé.

A szelekciós utasítás általános alakja:

case A of
K_{11}; K_{12}; ...; K_{1n}: S_1
K_{21}; K_{22}; ...; K_{2n}: S_2
.
.
.
K_{m1}; K_{m2}; ...; K_{mn}: S_m
else: S
end case

A: kifejezés (szelektor).
K_{ij}: az A-val megegyező típusú állandók.
S_i: utasítások, vagy utasítások sorozata.

Hatása: ha a program a lefutása során szelekciós utasításhoz ér, akkor a számítógép egymás után minden K_{ij} értéket összehasonlít az A kifejezés értékével. Ha a K_{ij} állandók egyike az A értékével megegyezik, akkor a számítógép végrehajtja a megfelelő S_i utasítást. A program végrehajtása az *end case* után következő utasítással folytatódik.

Ha a K_{ij} állandók egyike sem egyezik meg A értékével, akkor a számítógép az *else* után álló, tehát az S utasítást hajtja végre, majd sorrendben folytatja a további programrészek végrehajtását.

Figyelem: az egyes programnyelvek ezt az esetet eltérő módon kezelik.

Példa a PASCAL nyelvben:
adott hónapban a napok számának meghatározása.

```
case honap of
  4,6,9,11:napok : = 30;
  1,3,5,7,8,10,12:napok : = 31;
  2if szokoev then napok : = 29 else
  napok : = 28
end;
```

A fenti példában a szelektor 1-től 12-ig terjedő egész értékeket vehet fel.

Figyelem: a PASCAL nyelvben a program lefutása bizonytalan, ha az A szelektor – esetünkben a *honap* – a K állandó – itt 1, ..., 12 – egyik értékével sem egyezik meg.

A legtöbb PASCAL fordítóprogram ebben az esetben az **end;** után következő utasításokkal folytatja a program végrehajtását.

A szelekciós utasításnak az egymásba ágyazott feltételes utasításokkal szemben több előnye van: lényegesen egyszerűbb és áttekinthetőbb, mint az *if... then... else* utasítások sorozata; a fordítóprogramok a szelekciós utasítást nagyon hatékony gépi programmá alakítják át; a számítási idő lényegesen megrövidül.

Strukturált program

A strukturált programok a megoldási algoritmust feladatorientált részprogramokra és ezek közötti kapcsolatokra bontják. Az egyes részstruktúrákat illeszkedési helyek választják el egymástól. A részstruktúrák mindaddig további részekre bonthatók, amíg az egyes részek programozása már egyszerűvé nem válik. A felbontás lehetőség szerint úgy történik, hogy minden részstruktúra teljesen és lehetőleg egyszerűen leírható és így – a többi programrészektől függetlenül – ellenőrizhető legyen. A programrészeknek az illesztési helyeken történő egymáshoz kapcsolása eredményezi a teljes programot. A strukturált program szerkezete különböző konfigurációkat követhet.

Szekvencia. A részstruktúrák időben egymás után kerülnek feldolgozásra.

Példa: összetett utasítás a PASCAL nyelvben.

```
begin
    ...;
    ...;
    ...
end;
```

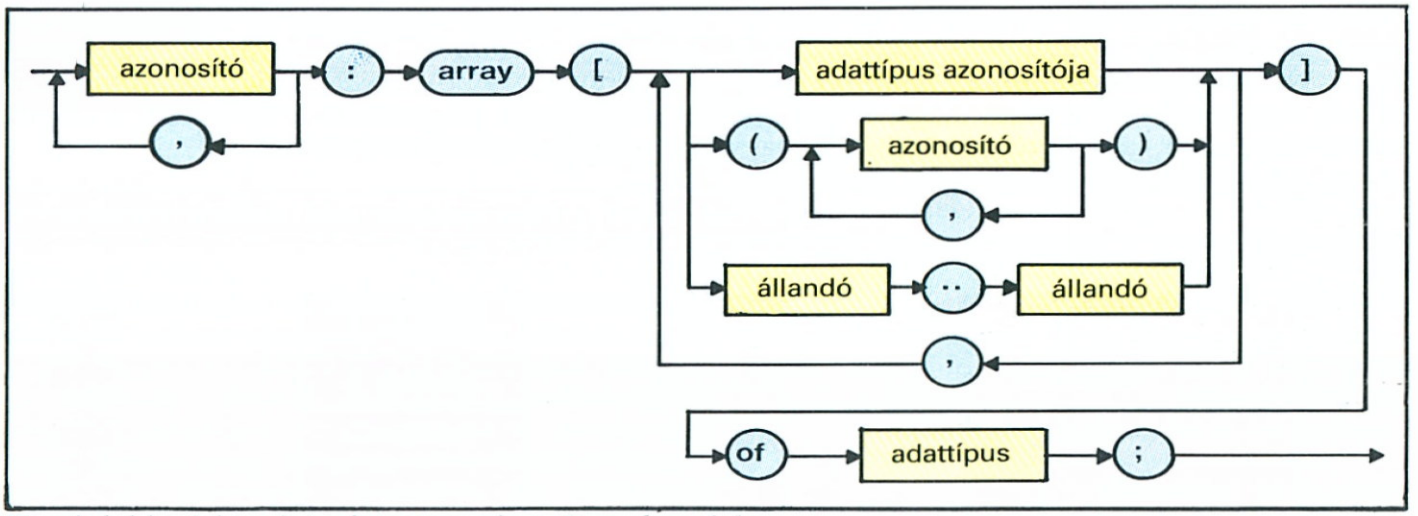
Szelekció. A részstruktúrák közül előre megadott feltételek alapján kiválasztunk egyet, és csak ez kerül feldolgozásra. Az alternatívák számának megfelelően általában különböző programozási lehetőségek léteznek.

Példa: a szelekciós utasítás a PASCAL nyelvben.

```
case ... of
  ...:...;
  ...:...;
  ...:...
end;
```

Iteráció. A részstruktúra egy ciklus magja, amely mindaddig ismételten végrehajtásra kerül, amíg egy előre megadott feltétel nem teljesül. A program ezután a következő részstruktúrát dolgozza fel – amely esetleg ismét iteráció lehet.

Példa: ciklusok (144. o.).



Tömbdeklaráció szintaxisprogramja a *Pascal* nyelvben

1-dimenziós tömb:

tömbelemek azonosítói: ALPH [1] ALPH [2] ALPH [3] ALPH [4] . . . ALPH [25] ALPH [26]

tömbelemek: a b c d . . . y z

az ALPH[4] tömbelem

tömbdeklaráció
ALPH : **array** [1..26] **of char** ;

tárolóigény 26

2-dimenziós tömb:

az a[3,2] tömbelem

3. sor

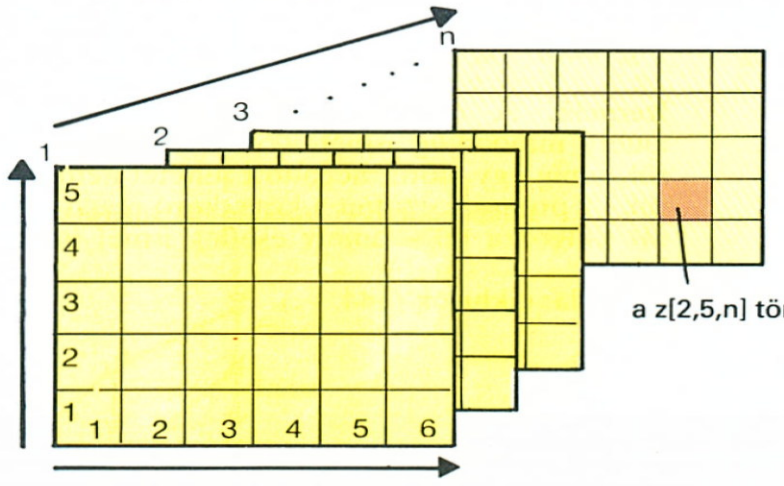
2. oszlop

a_{11}	a_{12}	a_{13}	. . .	$a_{1\,n}$
a_{21}	a_{22}	a_{23}	. . .	$a_{2\,n}$
a_{31}	a_{32}	a_{33}	. . .	$a_{3\,n}$
.				
.				
.				
a_{m1}	a_{m2}	a_{m3}	. . .	a_{mn}

tömbdeklaráció
a : **array** [1..n, 1..m] **of real** ;
vagy
a : **array** [1..n] **of array** [1..m] **of real;**

tárolóigény: n×m

3-dimenziós tömb



a z[2,5,n] tömbelem

tömbdeklaráció:
z : **array** [1..6, 1..5, 1..n] **of real** ;

tárolóigény: 5 × 6 × n

Tömbök és a megfelelő tömbdeklarációk a *Pascal* nyelvben

Tömbnek (angolul: *array*) nevezzük az azonos adattípusú elemek rendezett sorozatát. A tömb minden **eleme** (komponense) indexes változó. A tömbelemek jelölésére használt **index** a tömb elemeinek helyzetét egyértelműen meghatározza a tömbön belül.

Azokat a tömböket, amelyekben az elemek helyzetét egyetlen index adja meg, 1-dimenziós tömböknek vagy **vektoroknak** nevezzük.

Példa: Az év sorszámmal megjelölt hónapjai 1-dimenziós tömböt alkotnak.

honap P[i], $i = 1, \ldots, 12$.

A *honap* a tömb azonosítója, i pedig az index.

A *honap*[5] pl. a május tömbelemet jelenti.

Az n-dimenziós tömböknek elemenként n indexe van.

Példa: Egy ötismeretlenes egyenletrendszerben az ismeretlenek együtthatói 2-dimenziós tömböt képeznek, amelyet az együtthatók mátrixának is nevezünk. A tömb elemei 5 sorba és 5 oszlopba rendezhetők. A mátrix minden elemének 2 indexe van, pl. $a[i, j]$ a mátrix i-edik sorának és j-edik oszlopának eleme.

A tömbök újabb adattípust (133. o.) képviselnek, amelyeket a program fejrészében deklarálni kell. Ez a deklaráció foglalja le a helyet a tömb – bizonyos esetekben nagyszámú – elemei számára.

Tömbdeklaráció

A tömbdeklaráció rögzíti a tömb azonosítóját (amelyet **indexes változónak** nevezünk), az indexek tartományát, a tömb dimenzióinak számát és a tömb elemeinek adattípusát.

Példa: 1-dimenziós tömb deklarációja a PASCAL nyelvben.

V: array [A] of T

V: a tömbelemek (egyindexes) azonosítója.
A: a V azonosító indexeinek tartományát meghatározó kifejezés.
T: a tömbelemek adattípusa, pl. *real* vagy *boolean*.
Ebben az esetben a tömb egyik eleme pl. V[3].

Az **indextartomány** megadása a legegyszerűbb esetben egy egyszerű adattípus nevének megadásával történhet. Pl. az *array* [*boolean*] deklarációval megadott tömbnek csak két komponense van: az *igaz* és a *hamis* Boole-állandók, az *array* [*char*] deklarációval megadott tömb elemeit pedig a teljes karakterkészlet alkotja.

Az *array* [*integer*] tömbdeklaráció memóriatúlcsorduláshoz (angolul: *memory overflow*) vezethet, mert a számítógép megpróbál az összes számnak elegendő tárolóterületet biztosítani.

Általában az egyszerű adattípusok adott résztartományát szokás megadni:

array[*min*...*max*]

min, max: a résztartomány legkisebb ill. legnagyobb értéke.

Példák a PASCAL nyelvből:

honap: **array** [1...12] **of char;**
vektor: **array** [20...50] **of real;**

A *honap* és a *vektor* tömbök egy-egy eleme pl. a *honap*[5] és a *vektor* tömbök egy-egy eleme pl. a *honap*[5] és a *vektor*[28].

Az indexes változókra egyszerű kifejezéssel vagy standard függvénnyel is hivatkozhatunk, pl. honap [n + 6] vagy vektor [**abs**(i + p)].

n-dimenziós tömb deklarációja a PASCAL nyelvben:

V: **array** [$A_1, A_2, \ldots, A_n$] **of** T**;**

V: a tömbelemek (n-indexes) azonosítója.
T: a V azonosítóval megjelölt tömb elemeinek adattípusa.
$A_1, A_2, \ldots, A_n$: a V azonosító egyes indexeinek tartományát meghatározó kifejezések.

A 3-dimenziós tömb egyik eleme pl. V[3,24,2].

Az egyes indextartományok megadása a fent leírthoz hasonlóan történik. Minden dimenziót külön, vesszővel elválasztva kell megadni.

Példa a PASCAL nyelvből:

matrix: **array** [1..8,1..10] **of real;**

Ez a deklaráció a matrix tömb számára 8 sort és 10 oszlopot foglal le. A tömb elemei pl. a matrix[2,2] és a matrix[1,8].

Tömbdeklaráció a BASIC nyelvben:

DIM V (I)

V: a tömbelemek azonosítója.
I: az indextartomány felső határa, amely egész szám. Az alsó indexhatár mindig 1, tehát a teljes indextartomány: 1, 2, ..., I.

A BASIC nyelv néhány változatában az indextartomány alsó határa mindig 0.

A tömbelemek adattípusát a BASIC nyelvben a tömbazonosító első betűje fejezi ki.

Példák: DIM VEKTOR(50)
DIM HONAP(12)
Ennek PASCAL nyelvbeli megfelelői:
vektor**:array** [1..50] **of real;**
illetve honap**:array** [1..12] **of integer;**

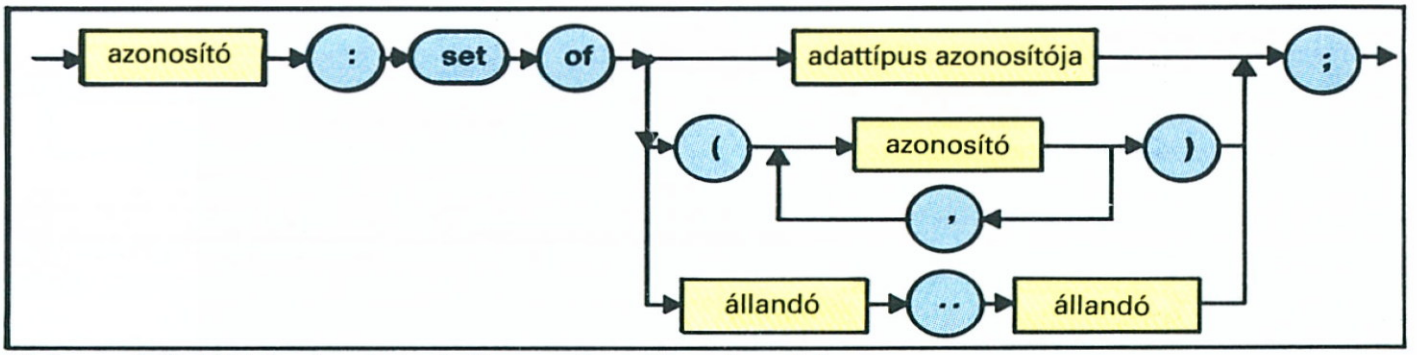
Az n-dimenziós tömbök deklarációja a BASIC nyelvben:

DIM V ($I_1, I_2, \ldots, I_n$)

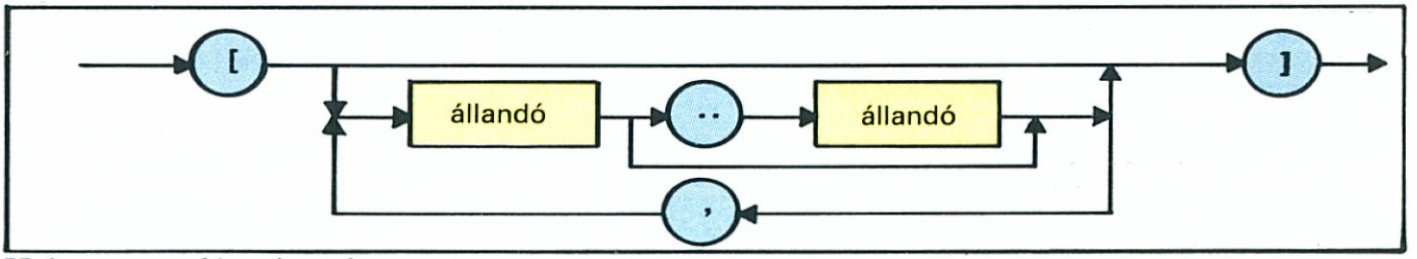
$I_1, I_2, \ldots, I_n$: az egyes indextartományok felső határai.

Példa: DIM MATRIX(8,10,2)

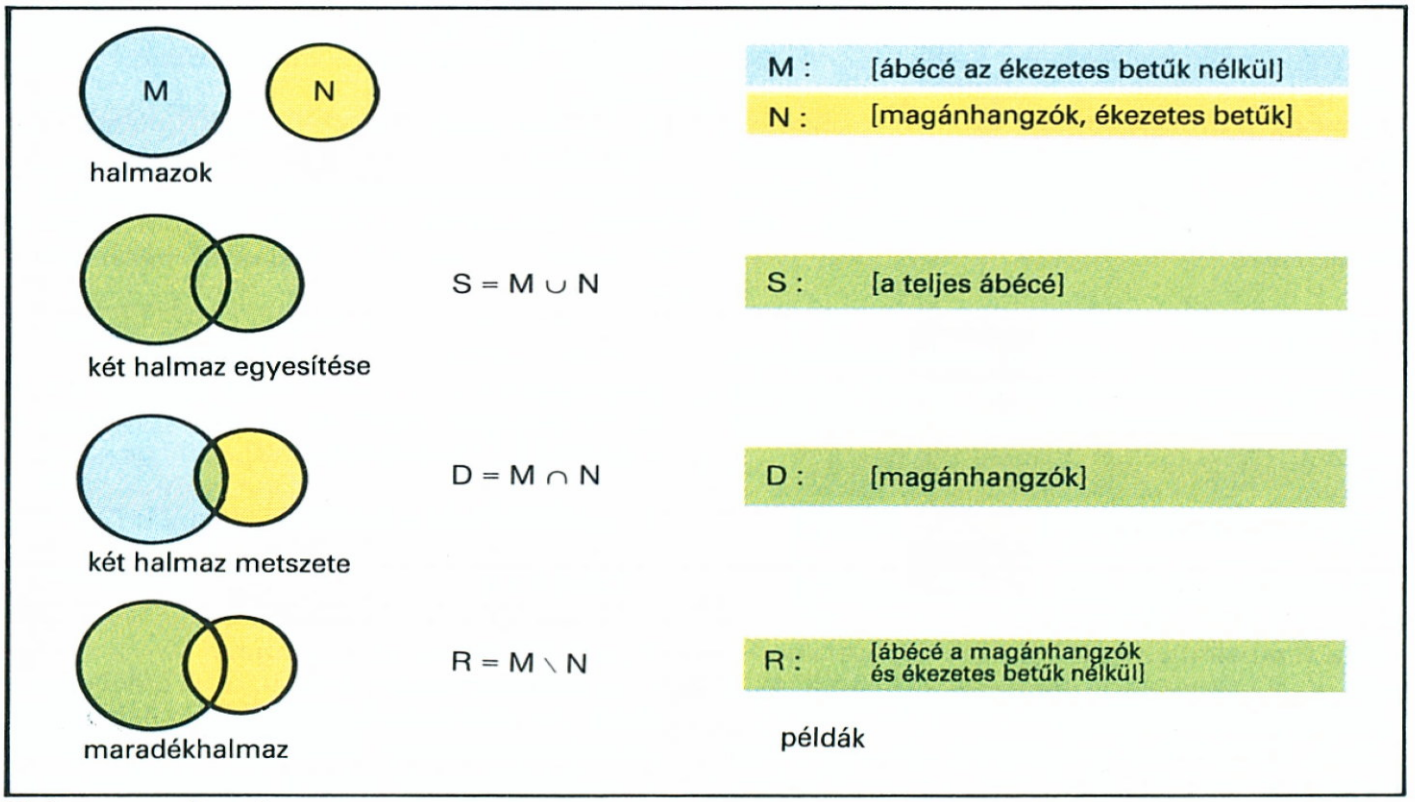
Ez a deklaráció a 3-dimenziós tömb számára 8 sort, 10 oszlopot és 2 réteget foglal le.



Halmaz definiálásának szintaxisdiagramja a *Pascal* nyelvben



Halmazmegadás szintaxisa



Halmazoperátorok

$M = N$	*M, N* ugyanazokból az elemekből áll. Az elemek sorrendje különböző lehet. Ugyanazt az elemet csak egyszer szabad számításba venni.
$M \neq N$	az *M, N* halmazok legalább egy elemben különböznek.
$M \subseteq N$	*M* az *N* alaphalmaz részhalmaza. *M* valamennyi eleme az *N* halmaznak is eleme, de ez fordítva nem igaz.
$M \subset N$	*M* az *N* halmaz valódi részhalmaza, azaz $M \neq N$.
$M \supseteq N$	*M* az *N* halmaz alaphalmaza.
$M \supset N$	*M* az *N* valódi részhalmaza, azaz $N \neq M$.

M és *N* halmaz összehasonlításának operátorai

Néhány programnyelv lehetőséget nyújt véges halmazok közvetlen feldolgozására is. Bár a halmazokat a számítógépben tömbök segítségével is lehet ábrázolni, de az ilyen programok meglehetősen áttekinthetetlenek.
Egy *M* halmazt speciális adattípusként lehet bevezetni, a halmaz minden elemének azonos, egyszerű adattípusnak kell lennie.

Halmaz megadása
A halmaz elemei a programban szögletes zárójel között jelennek meg. Az egyes elemeket egymástól vessző választja el.
Példák:
Betűkből álló halmaz: [A, H, N, S, X, Y, Z].
A hét első három napja: [H, K, Sze].
Az első 7 ptl. prímszám halmaza: [3, 5, 7, 11, 13, 17, 19].
A páros számok egy részhalmaza: [22, 24, 26, 28].
Az a_1, succ(a_1), . . . , a_2 halmaz: [a1. . a2].
A 3-tól kezdődő egész számok halmaza: [3. . n].
Az ASCII-jelek halmaza: [char].
Üres halmaz: [].

Halmaz deklarációja
A deklaráció rögzíti a halmaz azonosítóját, valamint a halmaz elemeinek adattípusát.
A halmaz elemeinek száma, azaz a halmaz *számossága* korlátozott. A PASCAL nyelvben pl. a halmaz számossága < 256.
A halmaz deklarációja a PASCAL nyelvben:

V: **set of** *BT*;

V: a halmaz azonosítója.
BT: a halmaz elemeinek adattípusa. Csak egyszerű adattípusok fordulhatnak elő, tehát: egész számok, valós számok, logikai értékek, karakterek, vagy a felsorolt halmazok részhalmazai.
Példák a PASCAL nyelvben:
abc: **set of** char;
(*abc* annak a halmaznak az azonosítója, amely a számítógép számára rendelkezésre álló jeleket tartalmazza, pl. az ASC II-kód valamenynyi jelét.
nap: **set of** 1 . .31;
(A *nap* azonosítójú halmaz az 1 és a 31 közötti egész számokat tartalmazza).

Műveletek halmazokkal
Az operandusok két, azonos adattípushoz tartozó halmaz. A művelet eredménye ismét egy halmaz.

Egyesítés. Az *M* és az *N* halmazok egyesítése az *S* halmaz:
$S = M \cup N$
Példa a PASCAL nyelvben:
S : = M + N;
A + jel itt az egyesítés jele.
Metszet. Az *M* és az *N* halmazok metszete a *D* halmaz:
$D = M \cap N$
Példa a PASCAL nyelvben:
D : = M * N;
A * jel itt a közös rész képzését szimbolizáló jel.
Különbség. Az *M* és az *N* halmazok különbsége az *R* halmaz:
$R = M \setminus N$
Példa a PASCAL nyelvben:
R : = *M* – *N*;
A – jel itt az különbségképzés \ jele.

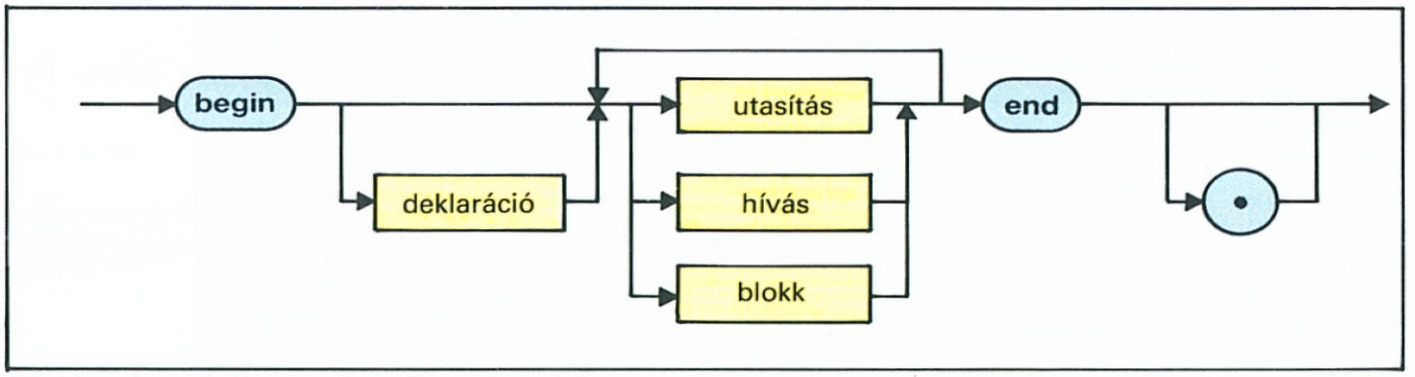
Számosság. Néhány programnyelv lehetőséget nyújt arra, hogy az *M* halmaz elemeinek számát (a halmaz *számosságát*) a *card*(*M*) utasítással meghatározzuk.

Halmazok összehasonlítására szolgáló műveletek
Az operandusok két, azonos adattípushoz tartozó halmaz. Az összehasonlítás eredménye vagy *igaz*, vagy *hamis* érték lehet.
Az *M* és az *N* halmazok **egyenlősége**:
$M = N$
Példa a PASCAL nyelvben:
M = N;
Az *M* és az *N* halmazok **nem egyenlőek**:
$M \neq N$
Példa a PASCAL nyelvben:
M < > *N*;
Az *M* halmaznak a *V* **részhalmaza:**
$V \subseteq M$
Példa a PASCAL nyelvben:
V <= M;
Tartalmazás. Az *M* halmaz tartalmazza a *V* részhalmazt:
$M \supseteq V$
Példa a PASCAL nyelvben:
M > = V;

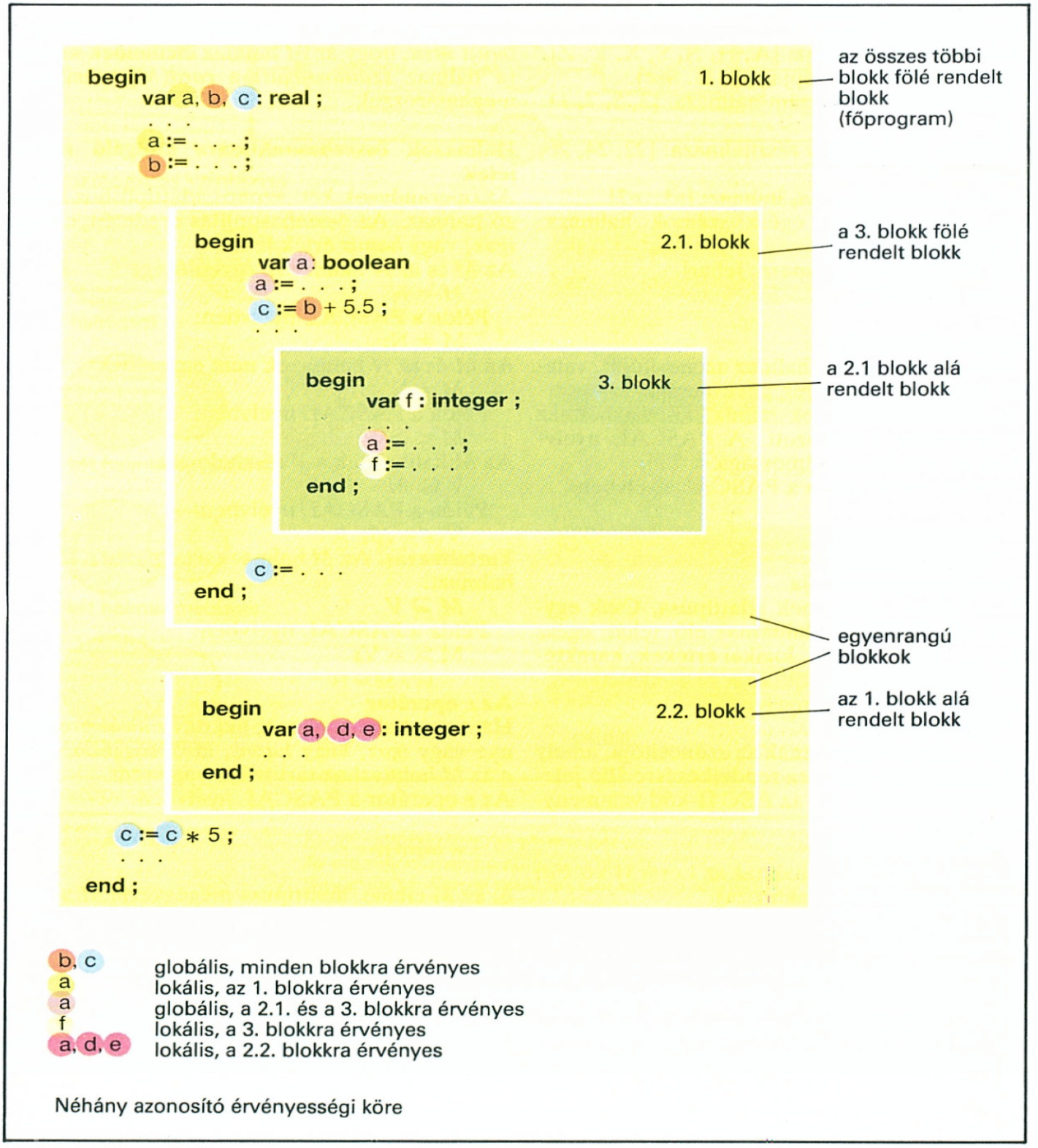
Az ε operátor
Ha *a* egy halmaz eleme, akkor az $a \varepsilon M$ eredménye vagy *igaz*, vagy *hamis*, attól függően, hogy *a* az *M* halmazhoz tartozik-e vagy sem.
Az ε operátor a PASCAL nyelvben:

a **in**[M];

a: az M eleme, adattípusa megegyezik *M* adattípusával.
Az a elem egyszerű kifejezés is lehet.
Példa a PASCAL nyelvben:
v **in** [a, b, c, d, e, f];
(A halmaz elemeit a szűkített ábécé első hat betűje alkotja. Ha v = a vagy = b vagy = c vagy = e vagy = f, akkor a művelet eredménye *igaz*, különben *hamis*.)



Blokk szintaxisdiagramja a *Pascal* nyelvben



Blokkok hierarchiája

A hosszabb programok könnyen áttekinthetetlenné válnak. Ilyen helyzetekben segíthet a program strukturálása, azaz a program egymástól többé-kevésbé független, egyedi programblokkokra történő felosztása. Ennek során minden blokkot egy-egy utasításnak tekinthetünk. A felosztás csak **strukturált programnyelvekben**, pl. az ALGOL, a PASCAL és a C nyelvekben lehetséges. A gépközeli, pl. a FORTRAN és a BASIC nyelveken írt programok alig strukturálhatók.

Blokk

A blokk (angolul: *block*) egymás után rendezett deklarációkból és utasításokból áll. A legtöbb programnyelvben a blokk *begin* utasítással kezdődik, a blokk utolsó utasítása pedig a blokk végét jelző *end*. A *begin* és az *end* nyelvelemek legnagyobb prioritású zárójeleknek tekinthetők.

A *szekvencia,* az utasítások sorozata (149. o.) már egy egyszerű blokknak tekinthető.

A blokkok párhuzamosan egymás mellé rendelt vagy egymásba ágyazott szerkezetűek lehetnek, ilyenkor adott blokk több másik blokkot foglal magában. A blokkszerkezetű program áttekinthetőbbé válik és a programnak a blokkok között fennálló hierarchikus szerkezetét szemmel láthatóan is tükrözi.

Példa a PASCAL nyelvből:

```
begin
  begin
    begin...end;
    begin...end;
    begin...end
  end;
  begin
  ...
  ...
  ...
  end
end;
```

A legkülső blokk két egyenrangú blokkból áll, az egyik közülük három további blokkot – alprogramot – foglal magában. Tehát a program több, egymásba skatulyázott blokkot tartalmaz.

A PASCAL nyelvben magát a teljes programot is blokknak tekintjük. Az összes többi programrész fölé rendelt blokk – tehát a teljes program, a főprogram – végét ponttal **(end.)** jelöljük.

Az azonosítók érvényességi köre

A strukturált programokban az azonosítókhoz (változókhoz, állandókhoz, címkékhez, eljárásokhoz stb.) **érvényességi kört** (angolul: *scope*) is rendelünk.

Adott blokkon belül csak azok az azonosítók érvényesek, amelyeket az adott blokkban deklaráltunk. Az egyenrangú blokkokban ezekre az azonosítókra nem lehet hivatkozni. Az adott blokk határain belül érvényes azonosítók tehát **lokális** jellegűek.

Lokális azonosítóknak nevezzük azokat az azonosítókat, amelyeket egy adott blokkban explicit módon deklaráltunk és ezért csak az adott blokkon belül érvényesek.

A **globális azonosítókat** olyan blokkokban deklaráljuk, amelyek további blokkokat foglalnak magukban. Ezek az azonosítók azon a blokkon kívül, amelyben a deklaráció történt – ez lehet a teljes főprogram is –, az illető blokkba hierarchikusan beleágyazott blokkokban is érvényesek.

A kétféle azonosító használata rendkívül megkönnyíti a programozást. Pl. egy több helyen felhasznált globális állandóra csak egyszer kell értékadó utasítást felírni.

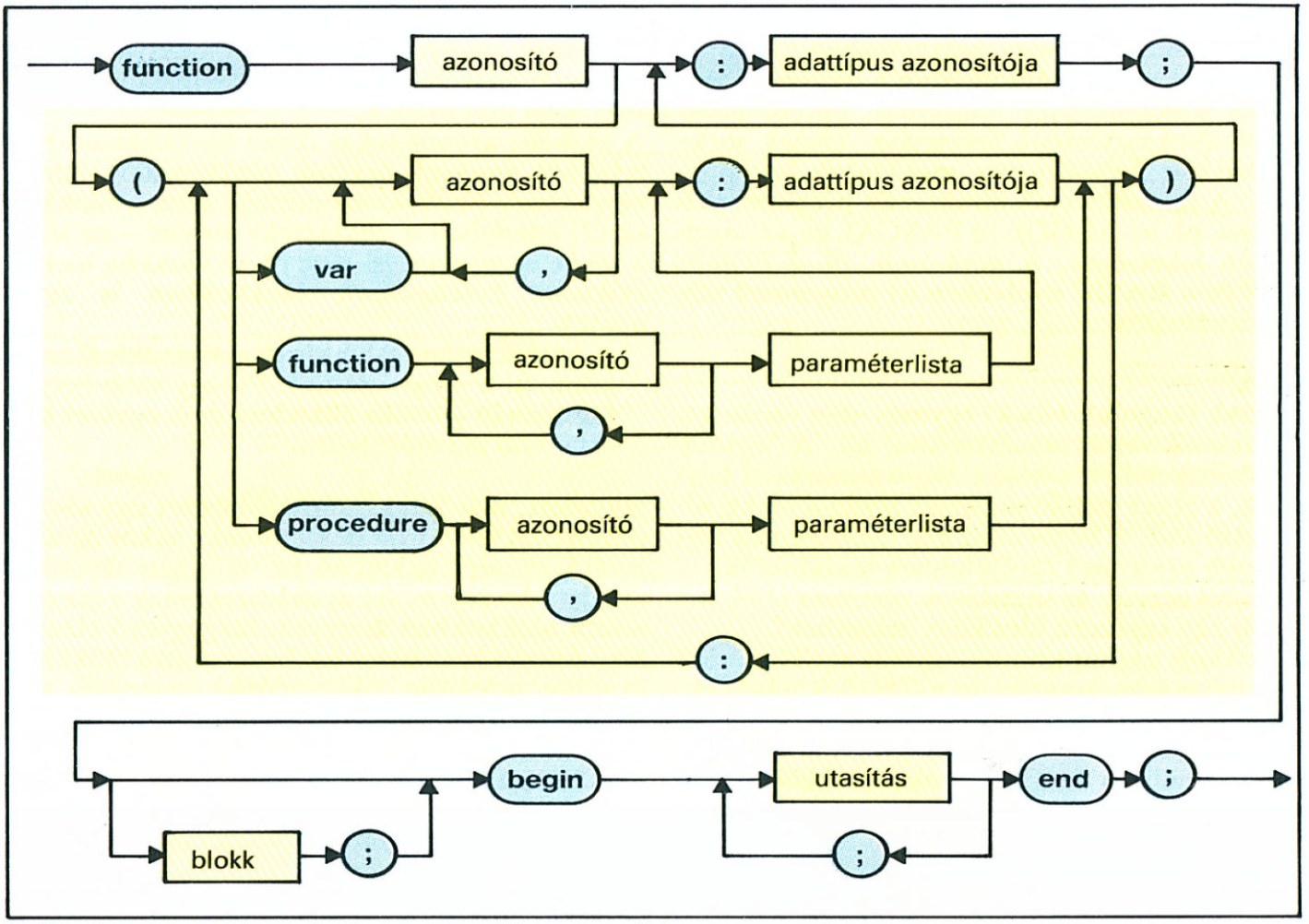
Ha adott, már felhasznált azonosítót egy alsóbb szintű blokkban újra deklarálunk, akkor az azonosító az alprogramban az újonnan deklarált alakban érvényes. Az új deklarációnak a felsőbb szintű blokkokban szereplő, megegyező elnevezésű azonosítóra nincs hatása. Az alsó blokkban azonban a lokális deklarációnak prioritása van a felsőbb szintű blokkokban szereplő deklarációval szemben.

Példa a PASCAL nyelvből:

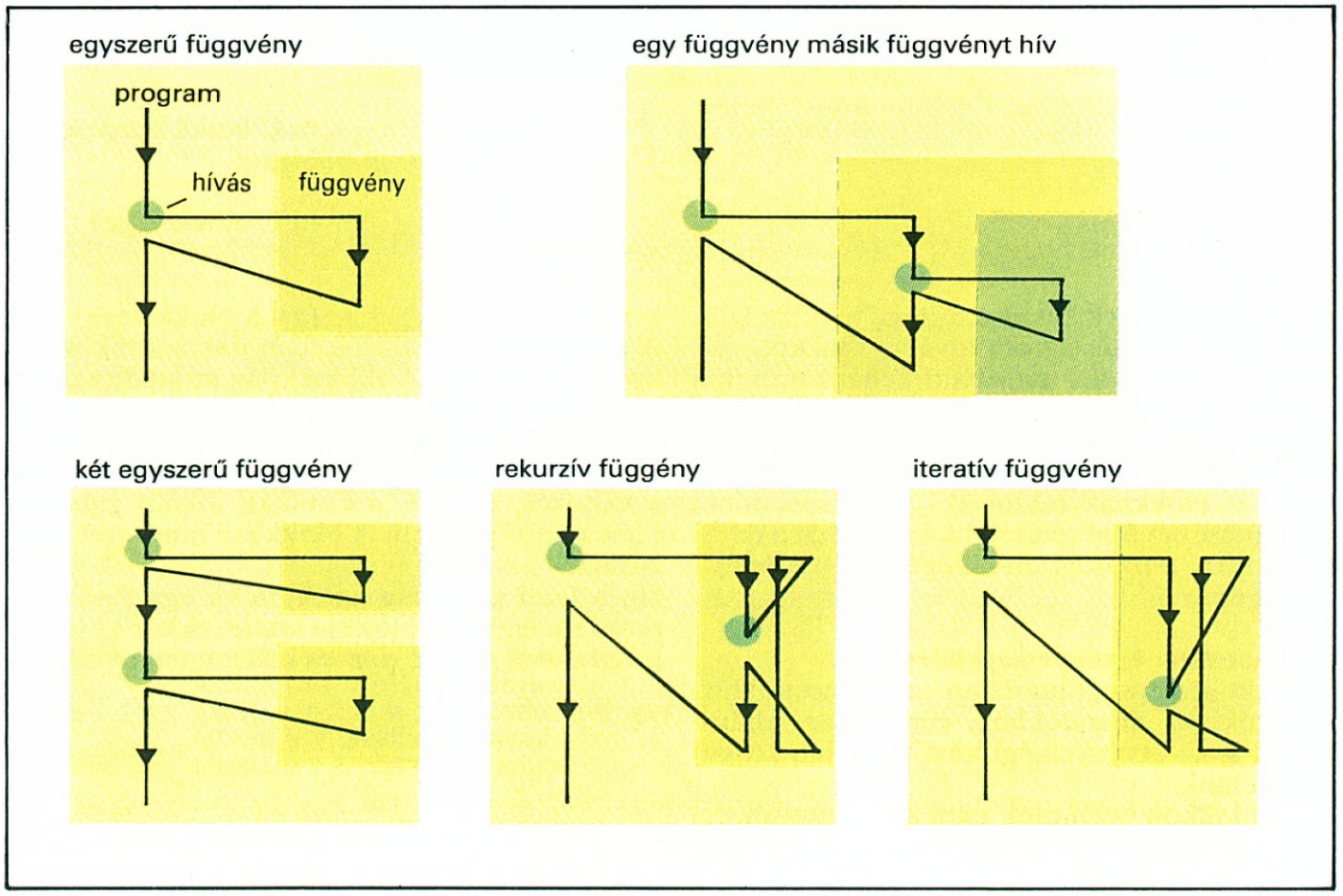
```
begin                          (az A blokk kezdete)
    var a, b, c: real;
    ...
      begin                    (a B blokk kezdete)
        var a, d, e: integer;
        ...
      end;                     (a B blokk vége)
      ...
      ...
      begin                    (a C blokk kezdete)
        var b, f, g: boolean;
        ...
      end;                     (a C blokk vége)
    ...
    ...
end;                           (az A blokk vége)
```

A felsőbb szintű A blokkban deklaráltuk a **real** típusú *a* változót. A B blokkban azonban az *a* lokális **integer** típusú változó, míg a *b* globális és továbbra is **real** típusú. A *c* változó mindkét alsóbb szintű blokkban (B, C) globális. A *d, e, f* és *g* változók a B ill. a C blokk lokális változói, ezek a felsőbb szintű A blokkban nincsenek deklarálva.

Ha a fenti példában az A blokk egy PASCAL program legkülső blokkja lenne, akkor az utolsó programsor végére pontot kellene tenni: **end.**



Adott függvény szintaxisdiagramja a *Pascal* nyelvben



Függvényhívások

A programkönyvtárak standard függvényeket tartalmaznak, amelyek a programtörzs bármely részéből hívhatók. A függvények ennek eredményeképpen értéket rendelnek valamilyen változóhoz. A modern programnyelvek lehetővé teszik, hogy a programban a standard függvényeken kívül egyéb függvényeket is definiáljunk, amelyeket azután a standard függvényekhez hasonlóan lehet használni. A függvények nagyon egyszerű alprogramok.

Az *eljárásokkal* (159. o.) ellentétben a függvények csak egyetlen értéket szolgáltatnak a függvényhívást tartalmazó programrész számára.

Függvény

Függvénynek (angolul: *function*) vagy függvényeljárásnak nevezzük az olyan összetett utasítássorozatot, amely a függvényhívással történő végrehajtás után egyetlen értéket ad eredményül. A függvények a program egyszerűsítésére szolgálnak.

Példa: ha egy algoritmus gyakran felhasználja az x! értéket és a faktoriálisképzés a programkönyvtár standard függvényei között nem szerepel, akkor pl. mint *fakult*(*x*) *függvény* deklarálhatjuk és a programban a standard függvényekhez hasonlóan használhatjuk.

A függvények függvényfejből és függvénytörzsből állnak.

A **függvényfej** a függvényt és paramétereit deklarálja.

A függvényfej általános alakja:

function $F(a_1, a_2, \ldots, a_n)$: T

F: a függvény azonosítója (a függvény neve).

$a_1, a_2, \ldots, a_n$: a *formális* (és deklarált) függvényparaméterek, amelyek a függvényhívással történő aktualizáció során az *aktuális* paraméterekre cserélődnek ki, tehát azokra a bemenő adatokra, amelyekkel a függvény értékét ki akarjuk számítani. A függvények paraméterei az *eljárások* (159. o.) paramétereinek felelnek meg.

T: az eredménynek, vagyis annak az értéknek az adattípusa, amelyet a függvény a függvényhívást tartalmazó programrészletnek szolgáltat.

A **függvénytörzs** tartalmazza a függvény értékének kiszámításához szükséges utasításokat.

A függvénytörzsben a függvény azonosítójához legalább egyszer valamilyen értéket kell rendelni, tehát a függvénytörzsnek legalább egy

$F = \ldots$

értékadó utasítást kell tartalmaznia.

Függvényhívás (függvényhivatkozás). A függvényt (valamilyen kifejezésben szereplő) azonosítójával lehet behívni a programba. A függvényhíváshoz az azonosítón kívül nincs szükség speciális utasításokra.

A függvényhívás alakja (két paramétert tartalmazó függvény esetén):

```
...
a1 = ...
a2 = ...
...
V = 1 + F(a1, a2)
...
```

A V-re vonatkozó értékadó utasítás jobb oldalán olyan kifejezés áll, amely az $F(a_1, a_2)$ függvényt operandusként alkalmazza, pl.

V = 25 + X * F(A1,A2).

A függvényhívásban szereplő aktuális paraméterek matematikai kifejezések is lehetnek, pl. V = 25 + X * F(SIN(A1),A2 + 1.2). A formális paraméterek azonosítói csak helyfoglalásra szolgálnak, nem kell a függvényhívásban felhasznált (aktuális) paraméterazonosítókkal megegyezniük.

Példa egy függvényre a PASCAL nyelvben:

```
function arcsin (x: real): real;
  var w: real;
  begin
    w : = 1 - x*x;
    arcsin : = arctg(x/sqrt(w))
  end;
```

(Ebben a programrészletben feltételeztük, hogy az **arctg** standard függvényként adott, az arcsin függvény azonban nem.) Mind a függvényparaméter, mind a függvény értéke **real** típusú adatok.

Ha a program tetszőleges helyén az árkusz szinusz függvény értékére van szükségünk, akkor elegendő az előzőekben definiált *arcsin*(*x*) függvényt operandusként beírni a megfelelő helyre:

```
a : = 1 + arcsin(x);
```

Példa egy függvényre a BASIC nyelvben:

```
DEF FNA = ATN(Y/SQRT(1 - Y * Y))
```

(Itt is feltételeztük, hogy az ATN standard BASIC-függvény.)

Az árkusz szinusz inverz szögfüggvény felhasználása értékadó utasításban a következőképpen történik:

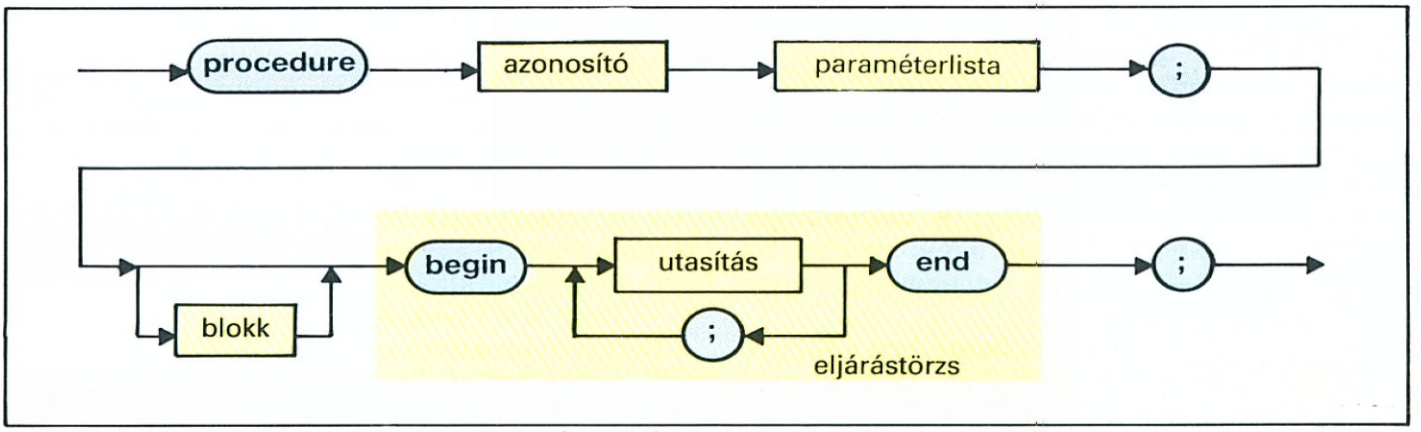
```
LET A = 1 + FNA(X)
```

Figyelem: A BASIC nyelv legtöbb változata csak egyparaméteres függvényeket enged meg, és a függvény definíciója csak egy utasítást tartalmazhat.

A függvényben szereplő mennyiségek érvényességi köre. A formális függvényparaméterek és a függvénytörzsben deklarált mennyiségek lokális jellegűek, csak a függvényen belül érvényesek.

A többértelműség elkerülése érdekében célszerű, ha a függvények definíciója lehetőleg nem tartalmaz be-/kiviteli utasításokat, és a függvénytörzsben szereplő globális azonosítókhoz sem rendelünk lokális értéket.

A függvények *rekurzív* módon is felhasználhatók, ami azt jelenti, hogy közvetve vagy közvetlenül önmaguk függvényhívását tartalmazhatják. Ennek során természetesen ügyelni kell arra, hogy a rekurzív függvényhívások ne hozzanak létre végtelen utasítássorozatot.



Egy eljárás szintaxisdiagramja a *Pascal* nyelvben

eljárásdeklaráció

procedure P (a_1,a_2,. . .,a_n);

Az eljárás paraméterei:

értékhívás	$V_1, V_2, \ldots, V_n : T$	(tetszőleges kifejezések)
referenciahívás	**var** $V_1, V_2, \ldots, V_n : T$	([indexes] változók)
függvény	**function** $V_1,V_2,\ldots,V_n: T$	(függvénynevek)
eljárás	**procedure** $V_1,V_2,\ldots,V_n$	(eljárásnevek)

eljáráshívás

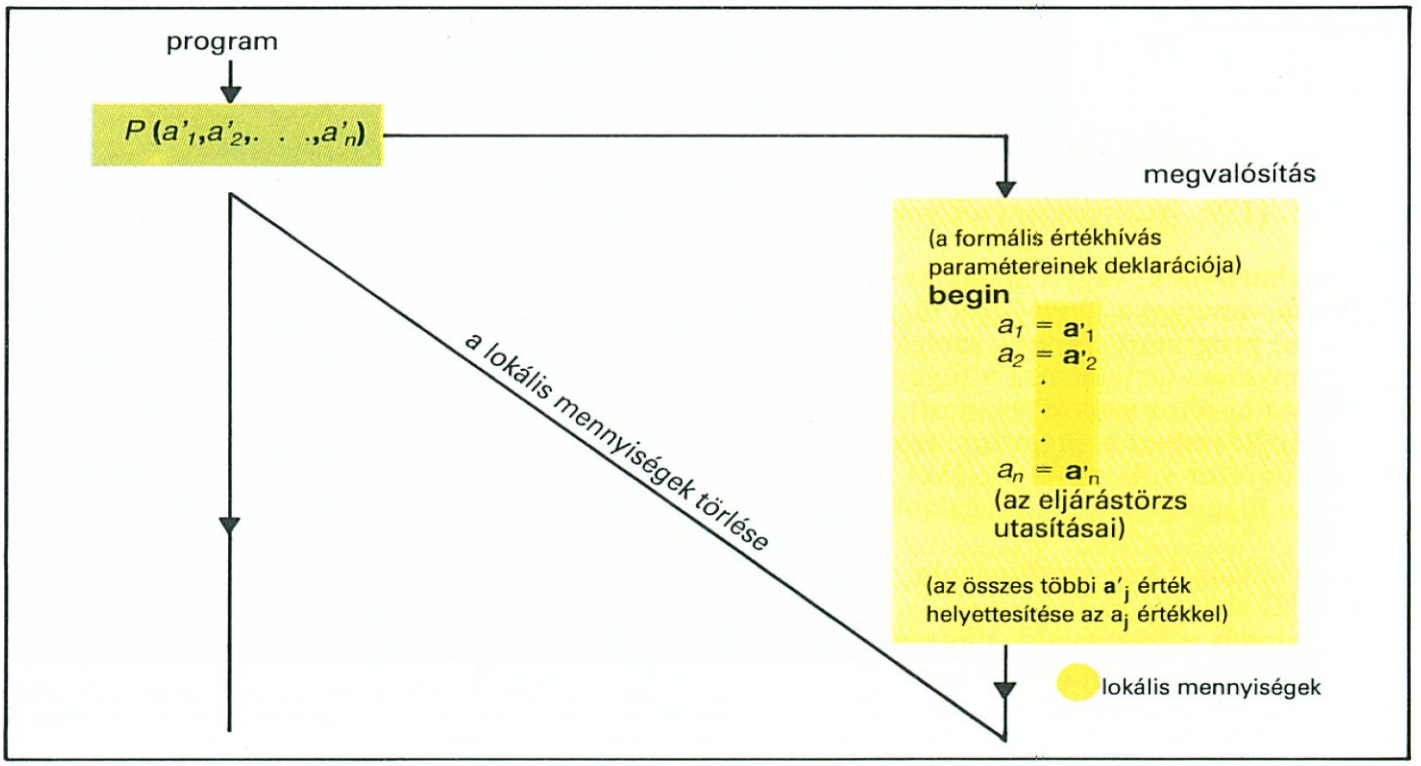
eljárásnév

$P(a'_1, a'_2, \ldots, a'_n)$

a: formális paraméter

a'_i: aktuális paraméter

Eljárás a *Pascal* nyelvben



A P eljárás bemásolása a hívást tartalmazó programba

A programnak azt a részét, amely más programot hív, de amelyet más programok nem hívhatnak, *főprogramnak* nevezzük. A hívható programrészeket *alprogramnak* nevezzük.
A **szubrutin** vagy **alprogram** (angolul: *subroutine*) utasítások olyan sorozata, amely önálló egységet alkot és egyetlen utasítás megadásával a programban többször hívható. Felhasználásával a hosszú és áttekinthetetlen programok áttekinthető programrészekre bonthatók, majd ismét teljes programmá kapcsolhatók össze. A rekurzív és az iteratív eljárások pl. szubrutinok segítségével egyszerűen előállíthatók.

A legtöbb programnyelvben az alprogramok két típusát lehet megkülönböztetni: a **függvényekkel** (157. o.) egyetlen értéket lehet kiszámítani a függvényhívást tartalmazó programrész számára, az **eljárások** ezzel szemben híváskor átmenetileg teljes egészében a főprogramba kerülnek, tehát ezek a szó valódi értelmében vett alprogramok.

Eljárások
az eljárás (angolul: *subroutine, procedure*) olyan névvel ellátott programblokk, amelyet a néven kívül önálló programnyelvi elem is jelöl. A PASCAL nyelvben ez a nyelvi elem pl. a **procedure** utasítás. Ez a blokk a neve segítségével egységes egészként hívható.
Mivel az eljárások hívása – a függvényektől eltérően – nem egyetlen értéket szolgáltatnak, az eljárás számára nem kell adattípust deklarálni.

Az **eljárásdeklaráció** a fölérendelt program változó deklarációjának (135. o.) része, amelyet a főprogram utasításai előtt helyezünk el. Az eljárásdeklaráció az *eljárásfejből* és az *eljárástörzsből* áll.

Az **eljárásfejet** az utasítás, az azonosító (az eljárás neve) valamint a *formális* (és deklarált) eljárásparaméterek listája alkotják. Az eljárásfej alakja:

procedure P (a_1, a_2, ..., a_n)

P: az eljárás azonosítója (az eljárás neve).
a_1, a_2, ..., a_n: az eljárás *formális* paraméterei (lásd alább), amelyek az eljárás hívásakor az *aktuális* eljárásparaméterekre cserélődnek ki. A formális és az aktuális eljárásparaméterek számának és adattípusának meg kell egyeznie.
Példák:
Eljárásfej a PASCAL nyelvben.
procedure, fajta **(...);**
Eljárásfej FORTRAN nyelvben:
SUBROUTINE RENDEZ (...)

Az eljárástörzs deklarációk sorozatából és az eljáráshoz tartozó utasításokat tartalmazó blokkból áll.

Példa eljárásdeklaráció a PASCAL nyelvben.

```
procedure rendez (...);
  ...
  begin
    ...
    ...
  end;
```

Eljárásdeklaráció pl. a FORTRAN nyelvben:

```
SUBROUTINE RENDEZ (...)
  ...
  ...
RETURN
END
```

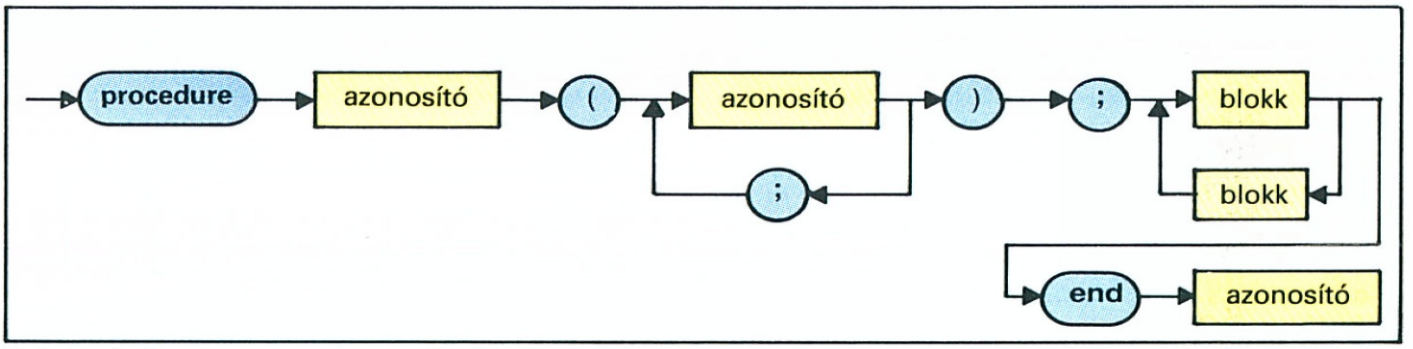
Az eljárásdeklarációk további eljárásokat foglalhatnak magukban. Ilyen módon egymásba ágyazott alprogramok jönnek létre.

Az **eljárásparaméterek** formális paraméterek, amelyek csupán helyfoglalásra szolgálnak. A paraméterek változók, valamint függvények és/vagy eljárások azonosítói lehetnek.
Példa: *procedure P1* (a_1, a_2, ..., a_n), *procedure P2* (V_1, V_2, ..., V_m).

Az **eljáráshívás** során – nevének megjelölésével – külön utasítás hívja az eljárást. Ha az eljárásban formális paraméterek (*a*) szerepelnek, akkor ezeket a híváskor az aktuális paraméterek (*a'*) helyettesítik.
Az eljáráshívás alakja:
$P(a'_1, a'_2, \ldots, a'_n)$
Példa a PASCAL nyelvben: rendez **(...);**
Példa a FORTRAN nyelvben: CALL RENDEZ (...)

Az eljárás hívásakor a hívás helyére az **inkarnációnak** nevezett *módosított* eljárástörzs lép. Ez azt jelenti, hogy az eljáráson belül deklarált változók és az eljárástörzs utasításai – az ún. a *bemásolás* során – a hívást tartalmazó programba kerülnek, és végrehajtásukra ott kerül sor. Ezt követően a lokális változók megszűnnek, az eljáráshívást tartalmazó program pedig a hívást követő utasítással folytatódik *(visszaugrás)*.

Az eljárásban szereplő mennyiségek **érvényességi köre** a programtörzsek lokális és globális azonosítóinak érvényességi körével egyezik meg (155. o.).



Szintaxisdiagram

array	**begin**	**boolean**	**comment**	+	−	x	/	÷	↑ (exp)
do	**else**	**end**	**false**	<	⩽	=	⩾	>	≠
for	**go to**	**if**	**integer**	,	.	:	;	:=	'
label	**own**	**procedure**	**real**	(	)	[	]	`	␣
step	**string**	**switch**	**then**	$_{10}$(10 alapú)	¬ (nem)			v (vagy)	
true	**until**	**value**	**while**	∧ (és)	≡ (azonosan egyenlő)			⊃ (tartalmazza)	

a b c d e f g h i j k l m n o p q r s t u v w x y z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

Kulcsszavak és karakterkészlet

procedure programnév (*paraméterek*);
deklarációk

alprogramok

procedure név (*paraméterek*);
deklarációk
begin
blokk
end
end név ;

begin
blokk
end

end programnév

Egy program általános felépítése

	jelentés
abs(*x*)	| *x* |
sign(*x*)	*x* előjele (+1, ha *x*>0, 0, ha *x*=0, −1, ha *x*<0)
sqrt(*x*)	$\sqrt{x}$
sin(*x*)	sin *x*
cos(*x*)	cos *x*
arctg(*x*)	arctg *x* főértéke
ln(*x*)	ln *x*
exp(*x*)	exp *x*, e^x

Standard matematikai függvények

első: ↑
második: x / ÷
harmadik: + −
negyedik: < ⩽ = ⩾ > ≠
ötödik: ¬
hatodik: ∧
hetedik: ∨
nyolcadik: ⊃
kilencedik: ≡

Műveletek prioritási sorrendje

a	**false**	**false**	**true**	**true**
b	**false**	**true**	**false**	**true**
¬*b*	**true**	**true**	**false**	**false**
a ∧ *b*	**false**	**false**	**false**	**true**
a ∨ *b*	**false**	**true**	**true**	**true**
a ⊃ *b*	**true**	**true**	**false**	**true**
a ≡ *b*	**true**	**false**	**false**	**true**

Példa:
ha *a* **igaz**, *b* pedig **hamis**,
akkor *a* v *b* **igaz**.

Igazságtáblázat

Az 1950-es években egy nemzetközi munkacsoport programnyelvet fejlesztett ki a matematikai algoritmusok általános megfogalmazására. Ebből keletkezett 1960/62-ben az **ALGOL 60** (angolul: **ALGO***rithmic* **L***anguage*). A be-/kiviteli utasítások csak az ALGOL különböző változataiban, pl. az ALCOR nyelvben jelentek meg. Az ALGOL továbbfejlesztése az ALGOL W (1966) változaton keresztül az ALGOL 68 nyelvhez vezetett, amelynek végleges változatát 1973-ban rögzítették.

Az ALGOL 60 mint az első programnyelv kontextustól független nyelvtannal rendelkezik, amelyet szintaxisdiagramok segítségével lehet ábrázolni. Nyelvtanának formális struktúrája, amelyet Backus-Naur formának (BNF) nevezünk, nagy hasonlóságot mutat az emberi nyelvek nyelvtanaival.

Az ALGOL 60 sok modern, műszaki-tudományos probléma megoldására használt programnyelv mintájául szolgált. Ez egy rendkívül rugalmas nyelv, kifejezései nagymértékben megegyeznek a matematikai algoritmusok alakjával.

Az ALGOL 60 *imperatív* és (az első) *blokkszerkezetű programnyelv*, vagyis alegységekből épül fel. A blokkokon belül az azonosítók lokális jellegűek, vagyis a blokkon kívül nincsenek deklarálva. A blokkok egymásba ágyazódhatnak, a külső blokkok számára a belső blokkok azonosítói nem érvényesek, ezek tehát **lokális** azonosítók. A belső blokkok ezzel szemben felhasználhatják a külső blokkok azonosítóit – a **globális** azonosítókat – vagy azokat a saját tartományukban újra deklarálhatják.

Az ALGOL 60 szabad formátumú nyelv (angolul: *free format language*): külső felépítésének, pl. a beírt szöveg sorhosszúságának nincs különösebb jelentősége. Az utasítások végét mindig a következő pontosvessző jelenti. Ezért az ALGOL programok a 60-as években uralkodó FORTRAN programokkal összehasonlítva sokkal áttekinthetőbbek és jobban olvashatóak.

Az ALGOL 60 programnyelvben elsőként 60 rekurzív alprogram – *eljárás* – állt rendelkezésre.

Az ALGOL 60 fordítóprogramjai a nyelv összetettsége és sokoldalúsága miatt más nyelvekkel összehasonlítva lassúak. A gépi kód tárolóigénye is viszonylag nagy.

Az ALGOL 60 fordítóprogramja a teljes programot lefordítja gépi programra, és csak ezt követően lehet a programot tesztelni és futtatni.

Az **ALGOL 68** az ALGOL 60 továbbfejlesztett változata, amelyben szintén egymásba ágyazott szerkezet valósítható meg, tetszőleges bonyolultságú adattartományokat lehet felépíteni, kevés az alapművelet és a szemantika tökéletesen leírható. A legtöbb számítógép csak az ALGOL 68 egyes részeihez rendelkezik fordítóprogrammal.

Az ALGOL 60 programok **blokkszerkezetűek.** A legkülső főprogram tartalmazza az összes alprogram deklarációit mint további blokkokat. Az alprogramok is egymásba ágyazhatók.

A programok általános felépítése. A **procedure** nyelvelem jelöli a program kezdetét, amelyet a program neve, a zárójelbe írt paraméterek és pontosvessző követ. A programot az **end** nyelvelem és a program neve zárja le.

A deklarációk a program ill. a blokk elején állnak, ezután következnek az utasítások és az alprogramok definíciói. Minden programtörzset **begin** és **end** zár közre.

A nyelv néhány eleme

Az **ALGOL 60 jelkészlete** latin kis és nagy betűkből, számjegyekből és számos különleges karakterből áll. A nagy betűk csak ritkán használatosak.

Az **ALGOL 60 nyelvelemeit** kis betűkkel kell írni. A könnyebb olvashatóság kedvéért az utasítások (és a pontosvessző) félkövér írással jelennek meg. A helyközöknek nincs jelentősége.

Példák: **string, label, go, to, step, real.**

A **változók deklarációja** a blokk elején történik a következő alakban:

adattípus változó azonosítója;

Real, integer, boolean és **label** adattípusok állnak rendelkezésre.

Példák: **real** y 1, pi, Anna **; integer** i, k, Rosi;

Az értékadást a : = operációs jel szimbolizálja, vagyis a : = jobb oldalán álló kifejezést rendeljük a bal oldalán álló azonosítóhoz. (Az értékadás előtt az azonosító értéke nincs meghatározva.)

Példa: y : = p/r+ k;

Az egész számokkal végzett műveletek integer típusú azonosítókkal történnek. A műveleti jelek a következők: +, −, ×, ÷, ↑ (exp), <, <=, =, >=, >, < >.

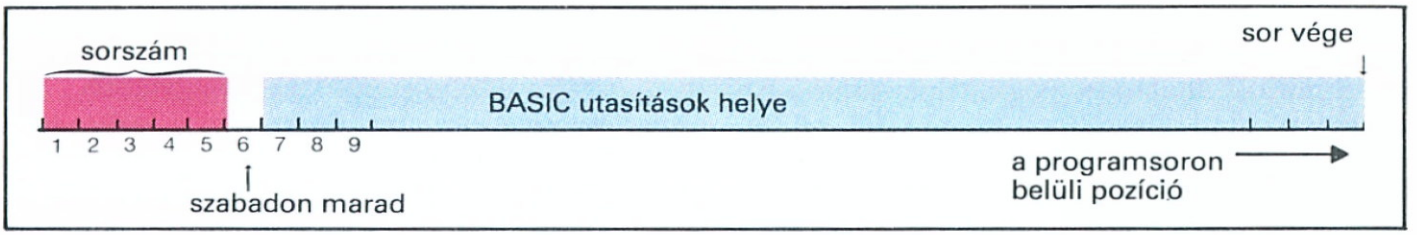
A **valós számokkal végzett műveletek real** típusú azonosítókkal történnek. A műveleti jelek a következők: +, −, ×, /, ↑ (exp), valamint az előbb felsorolt relációs műveleti jelek.

A **műveletek rangsora** (felülről lefelé haladva a következő: ↑, ×, /, ÷, +, −. A zárójelek ezt a sorrendet a szokásos módon megváltoztatják.

Az **adatmezőket** az alsó és a felső határ megadásával kell deklarálni. Ha az adatmező komponenseinek típusa nincs deklarálva, akkor azok **real** típusúnak tekinthetők. A legtöbb fordítóprogram három dimenziót, valamint negatív indexeket is megenged. Indexként matematikai kifejezések is deklarálhatók.

Példák: **array** a[5:28], b[−n,m],
sakk[1:8, 1:8]; **integer array** Y [**if** a < 0 **then** −3:10 **else** 3:10];

Az **alprogramok** formális szerkezete megegyezik a főprograméval. Az ALGOL 60 alprogramjait a **procedure** *név (paraméterlista)*; deklaráció vezeti be és az **end** *név utasítás* zárja le.



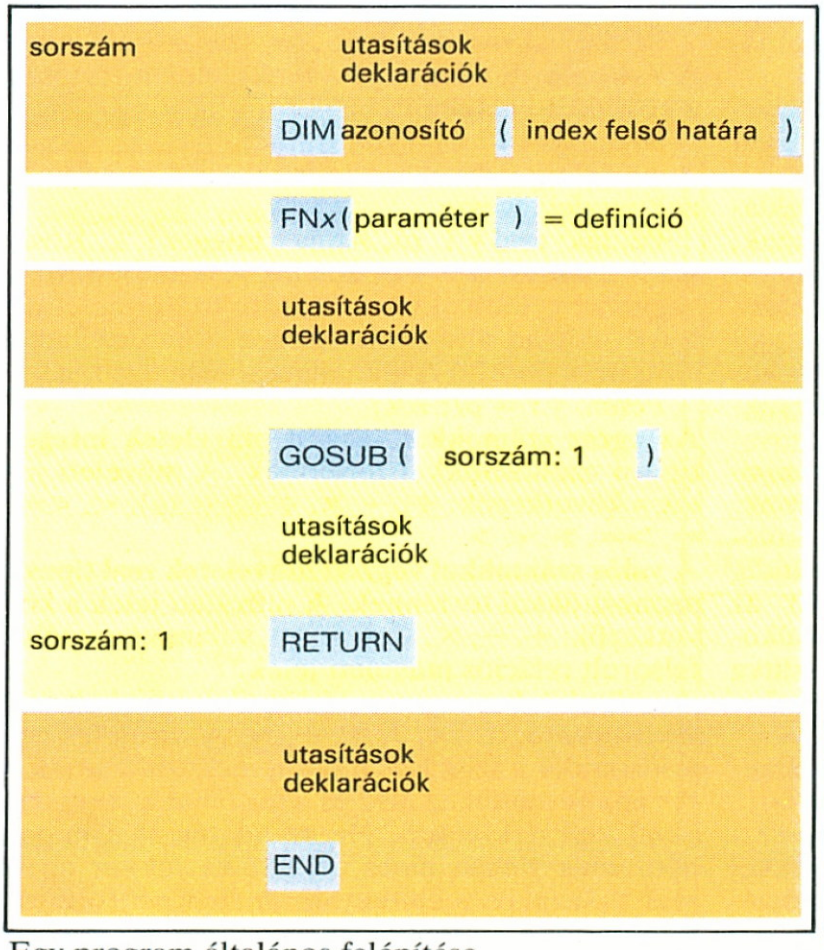
Programsor a BASIC nyelvben

AND	COMMON	DELETE	DIM	ELSE	END
FN*x*	FOR	GOSUB	GOTO	IF	LET
NOT	ON	OR	OUT	RETURN	RUN
SAVE	STEP	STOP	THEN	TO	WAIT
WHILE					
DATA	GET	INPUT	INPUT#	LIST	PRINT
PRINT#	PUT	READ	STRING#	WRITE	WRITE#

+ − * /
(egész számú osztás)
= <> < %
> >= # =>
AND NOT OR

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9

Kulcsszavak és karakterkészlet



Egy program általános felépítése

	jelentés
ATN(*x*)	arctg x főértéke
COS(*x*)	cos *x*, x radiánban
EXP(*x*)	e^x, exp(*x*)
LOG(x)	ln *x*
SGN(*x*)	x előjele (+1, ha x>0, 0, ha x=0, −1, ha x<0)
SIN(*x*)	sin *x*, x radiánban
SQR(*x*)	$\sqrt{x}$
TAN(*x*)	tg x, x radiánban
ABS(*x*)	*x*

Standard matematikai függvények

	hatás
INPUT *x*	x beolvasása (párbeszédes üzemmód)
INPUT#*n*, *x*	x beolvasása (*n* beviteli készülékről)
LIST *n*1, *n*2	sorok kiírása *n*1-től *n*2-ig (képernyőre)
PRINT *x*	x kiírása (párbeszédes üzemmódban)
PRINT# *n*, *x*	x kinyomtatása (*n* kiviteli készülékre)

Egyszerű be-/kiviteli utasítások

JOHN KEMENY és THOMAS KURTZ, a Dartmouth College (USA) munkatársai az 1960-as évek közepén a General Electric Corp. támogatásával egy rendkívül egyszerű programnyelvet fejlesztettek ki, amelyet BASIC-nek neveztek (angolul: **B***eginners* **A***ll Purpose* **S***ymbolic* **I***nstruction* **C***ode*). A kis számítógépek és a PC-k kategóriájában kezdők számára ez a legkedveltebb programnyelv.

A BASIC nagyon korlátozott lehetőségekkel rendelkezik, de *imperatív programnyelvnek* tekinthető, tehát a BASIC program deklarációk és utasítások sorozatából áll. A BASIC-ben a soroknak nagyon fontos szerepe van. Emiatt és a nagy számú ugró utasítás miatt a hosszabb BASIC programok áttekinthetetlenek.

A BASIC legnagyobb előnye a magasabb szintű programnyelvekkel, pl. a PASCAL nyelvvel szemben az, hogy könnyen megtanulható és a kisebb feladatok már kezdetben gyorsan és sikeresen megoldhatók. Hátránya ezzel szemben, hogy kevés adattípus áll rendelkezésre, nincsenek eljárások, és terjedelmes programok írására a nyelv nem alkalmas.

A BASIC programok fordítása és végrehajtása sorról sorra történik. A sorok bármikor törölhetők, megváltoztathatók és új sorok is beilleszthetők. Mivel a BASIC nyelvtana meszszemenően figyelembe veszi a számítógép logikai szerkezetét (géporientált nyelv), fordítóprogramjai nagyon hatékonyak.

A „standard" BASIC nyelvnek különböző változatai léteznek, közülük néhány nagyon magas szintű, és az ALGOL-hoz hasonló elemeket tartalmaz.

A **BASIC program** nem strukturált. Minden programsort szám *(sorszám)* jelöl. Párbeszédes üzemmódban a program bármikor megváltoztatható.

A programok általános felépítése. A program a legalacsonyabb sorszámú sorral kezdődik és a sorszámok növekvő sorrendje szerint folytatódik, hacsak ezt a sorrendet ugró utasítás nem változtatja meg.

A program utolsó sorának alakja mindig: *sorszám* END.

A nyelv néhány eleme

A BASIC jelkészlete latin nagy betűkből, számjegyekből és különleges karakterekből áll.

A **BASIC nyelvelemeit** nagy betűkkel kell írni.

Példák: AND, CALL, ELSE, IF, INPUT, PRINT, THEN.

A **változók deklarációja** a BASIC nyelvben optimális. Ha a program a futás során nem deklarált numerikus változóhoz ér, akkor azt a nyelv automatikusan valósnak és lebegőpontos formátumúnak tekinti.

Az implicit deklaráció gyakori hibaforrás: egy elírás hatására a program azonnal új változót vezet be.

Az explicit deklaráció a megfelelő változó azonosítójának neve után írt jellel történik:

% deklarálja az *egész (integer)* adattípust
Példa: a PA%, a I% és a T4% azonosítók egész számokat jelölnek.

$ deklarálja a karakterláncokból álló *füzér (string)* adattípust.
Példa: az N$, az A1$ és az NN$ azonosítók stringeket jelölnek.

kétszeres pontosságot deklarál.
Példa: az AA#, a B2# és az X# azonosítók olyan számokat deklarálnak, amelyek minden műveletben kétszeres pontosságú számként szerepelnek.

Az **állandók definíciója** a READ és a DATA utasítások segítségével történhet.

Példa:

n READ A, B%, C, PI
n DATA – 3,18,
2.345E – 12, 3.1416

Az A, B%, C és PI azonosítókhoz a –3,0 a 18, a $2{,}345 \times 10^{-12}$ ill. a 3,1416 numerikus értékeket rendeltük.

Az **értékadást** az = műveleti jel szimbolizálja, vagyis az = jel jobb oldalán álló kifejezést rendeljük a bal oldalán álló azonosítóhoz. A LET utasítás felhasználása az értékadásban nem kötelező.

Példa:

n X = PI * R + K
vagy
n LET X = PI * R + K

A **numerikus műveletek** valós, egész és kétszeres pontosságú számokkal végezhetők. A felhasználható műveleti jelek a +, −, *, / valamint az =, < >, <, > <= és > = relációs jelek.

Az egész számú osztás a \ műveleti jel segítségével történik. Az operandusokat a számítógép az osztás előtt kerekíti.

A **tömböket** első előfordulásuk előtt a DIM utasítással deklaráljuk. Az index alsó határa mindig 0.

Példa: az *n* DIM SA(7, 11) utasítás a kétdimenziós SA tömböt deklarálja, amelynek indexei 0-tól 7-ig, ill. 0-tól 11-ig terjednek. SA tehát egy 8×12-es mátrix.

Valamennyi tömbelem kiindulási értéke 0.

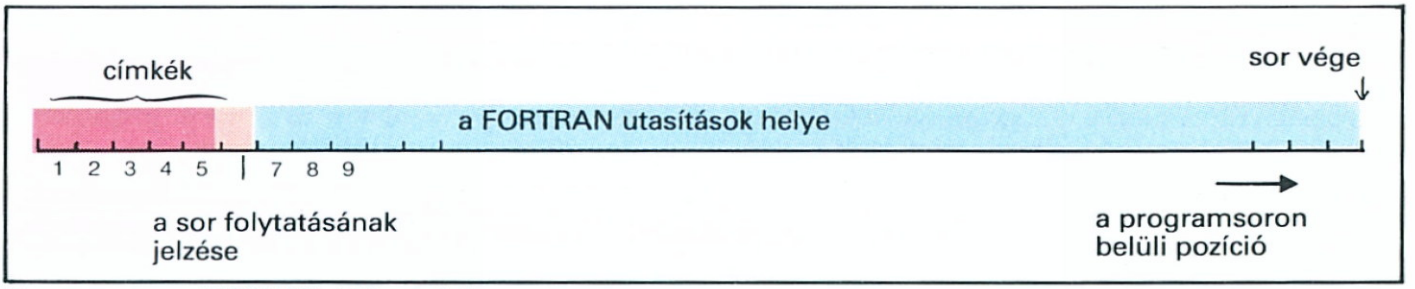
Az **alprogramok** (szubrutinok) beépítésének lehetősége a BASIC nyelvben nagyon korlátozott.

A **függvényutasítások** egy sor hosszúságúak és hívásuk előtt a DEF deklarációval definiálhatók. Híváskor a függvény egyetlen értéket számít ki. A függvény azonosítója FN betűkkel kezdődik, ezután – a felhasználó által választott – tetszőleges további betűk következnek.

Példa:

n DEF FNU(X) = 3.14 * X + K

Adott programrészlet kiválasztása a GOSUB *(n)* utasítással történik. Ennek hatására a program futása a megadott *n* sorszámra ugrik, majd azt, és az utána következő utasításokat a *m* RETURN utasítást tartalmazó sorig végrehajtja, ennél a sornál pedig visszaugrik a GUSUB után következő sorra.



Programsor a FORTRAN nyelvben

ASSIGN	CALL	CHARACTER	COMMON	COMPLEX
CONTINUE	DATA	DIMENSION	DO	DOUBLE PRECISION
EQUIVALENCE	ELSE	ELSEIF	END	ENTRY
FN	FUNCTION	GOTO	IF	IMPLICIT
INTEGER	LOGICAL	PARAMETER	REAL	RECORD
RETURN	SAVE	STOP	SUBROUTINE	THEN
TO				
BLOCK DATA	CLOSE	ENDFILE	FORMAT	INQUIRE
OPEN	PRINT	READ	REWIND	UNIT
WRITE				

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

Kulcsszavak és karakterkészlet

```
+  -  *  /
**   (kitevő)
//   (összefűzés)
.TRUE.      .FALSE.
.NOT.       .AND.
.OR.
.EQV.       .NEQV.
.EQ.        .NE.
.LT.        .GT.
.LE.        .GE.
```

```
főprogram
    PROGRAM
        program neve
        deklarációk
        utasítások
    END
alprogramok
    utasításfüggvények
        FN
            utasítás
    függvények
        FUNCTION
            deklarációk
            utasítások
        END
    eljárások
        SUBROUTINE
            deklarációk
            utasítások
        END
```

Egy program általános felépítése

	jelentés
ABS(*x*)	$\lvert x \rvert$
SQRT(*x*)	$\sqrt{x}$
SIN(*x*)	sin*x*
COS (*x*)	cos*x*
ATAN(*x*)	arctg x főértéke
LOG(*x*)	ln *x*

	hatás
READ *n*, *x*	x beolvasása a hozzátartozó *n* formátummal együtt
WRITE *n*, *x*	x kiírása
PRINT *n*, *x*	x kinyomtatása

ahol:

n FORMAT (m_1 *A* m_2)

m_1: soronkénti azonosítók száma

adattípus

m_2: az azonosító m_2 karakterből áll

Néhány standard mat.-i függv. és be-/kiviteli utasítás

A **FORTRAN** (angolul: **FOR***mula* **TRAN***slation*) programozási nyelvet az IBM mutatta be 1954-ben. Az évek során újabb változatai jelentek meg, pl. a FORTRAN II, IV és V. 1978 óta a FORTRAN 77 van használatban, az itt következő ismertetés is ennek alapján készült. 1992 óta az USA-ban a FORTRAN 90 a FORTRAN nyelv szabványos változata, a FORTRAN 2000 pedig fejlesztés alatt áll.

A FORTRAN 90 párhuzamos számítógépek programozására is alkalmas.

A FORTRAN tudományos feladatok megoldására szolgáló *imperatív programnyelv*. Sok standard függvényt tartalmaz, be-/kiviteli utasításai nagyon rugalmasak és különösen jól használhatók nagy adathalmazok feldolgozására.

A FORTRAN célorientált nyelv. A strukturálási lehetőségek hiánya és az ugró utasítások nagy száma miatt a FORTRAN programok gyakran áttekinthetetlenek. A FORTRAN fordítóprogramok nagyon hatékonyak, mert a nyelv nem annyira a programozót, mint sokkal inkább a számítógép szerkezetét veszi tekintetbe.

A **FORTRAN program** a főprogramból és alprogramokból (szubrutinokból) áll, amelyeket nem szabad egymásba ágyazni. A COMMON utasítás teszi lehetővé, hogy a különböző programrészek ugyanazokhoz a változókhoz és tömbökhöz férhessenek hozzá. A fordítóprogram a főprogramot és a szubrutinokat külön fordítja gépi kódra, amelyek csak közvetlenül a programfutás előtt kapcsolódnak össze.

A programok általános felépítése. A főprogram kezdetét PROGRAM, a végét pedig END utasítás jelzi, az alprogramok ezután következnek. A fordítás során sem a fő- és az alprogramok, sem az egyes alprogramok között nincs kölcsönhatás.

A FORTRAN nyelv a programok számára nem ír elő rögzített struktúrát. Azonban a változók adattípusát mindig a változó első megjelenése előtt kell deklarálni.

Az alprogramok kezdetét és végét a FUNCTION és END, ill. a SUBROUTINE és END nyelvelemek jelzik.

A nyelv néhány eleme

A FORTRAN jelkészlete a latin ábécé nagy betűiből, számjegyekből és különleges karakterekből áll.

A **FORTRAN utasításait** nagy betűkkel kell írni.

Példák: GO, THEN, ELSE, COMMON, FUNCTION.

A **változók deklarációja** a FORTRAN nyelvben meglehetősen szabad. Az implicit deklaráció az azonosító első betűjének segítségével történik: ha az azonosító I, J, K, L, M vagy N betűvel kezdődik, akkor a megfelelő változó egész (INTEGER), egyébként pedig valós (REAL) típusú.

Példák (explicit deklarációra): INTEGER V, EV, NAP
REAL SZAM, HONAP

Implicit deklaráció esetén ugyanezeknek a változóknak az azonosítói a következő alakúak: IV, IEV, NAP, SZAM, HONAP.

A változók implicit deklarációja gyakori hibaforrás: elírás hatására a program azonnal új változót vezet be.

Az **állandók definíciója** a program első sorában történik. A program későbbi utasításaiban a megfelelő azonosító hívja be az állandót:
PARAMETER (azonosító = állandó)

Példa: PARAMETER (PI = 3.1416)

Az **értékadást** az = jel szimbolizálja, vagyis az = jel jobb oldalán szereplő kifejezést rendeljük a bal oldalán álló azonosítóhoz. Az értékadás során az adattípus automatikus átalakítása megy végbe. Ha az azonosító egész típusú, akkor az eredmény is egész típusú, ami a FORTRAN nyelvben gyakori hibaforrás.

Az egész, a valós és a *kétszeres pontosságú* valós számokkal végezhető **műveletek** a következő műveleti jelek segítségével történnek: +, −, *, /, ** (exp), .EQ., .GT., .GE., .NE., .LT. és .LE.

A standard függvények paramétere mindhárom típusú változó lehet.

A **tömbök** dimenziója 7-ig terjedhet. Az indextartományokat a DIMENSION utasítás segítségével (explicit módon), vagy az adattípusok deklarációján belül (implicit módon) lehet deklarálni. Ha az indexek alsó határa a deklarációból hiányzik, akkor az alsó határ automatikusan 1. A tömbelemeket explicit vagy implicit módon deklarálhatjuk.

Példa:
DIMENSION VECTOR(1:8,1:12)
vagy DIMENSION VECTOR(8,12)
vagy REAL VECTOR(8,12)
(A VECTOR azonosító 8 × 12 elemű mátrixot ábrázol.)

A FORTRAN nyelv a tömbök elemeit oszlopfolytonosan egymás után tárolja.

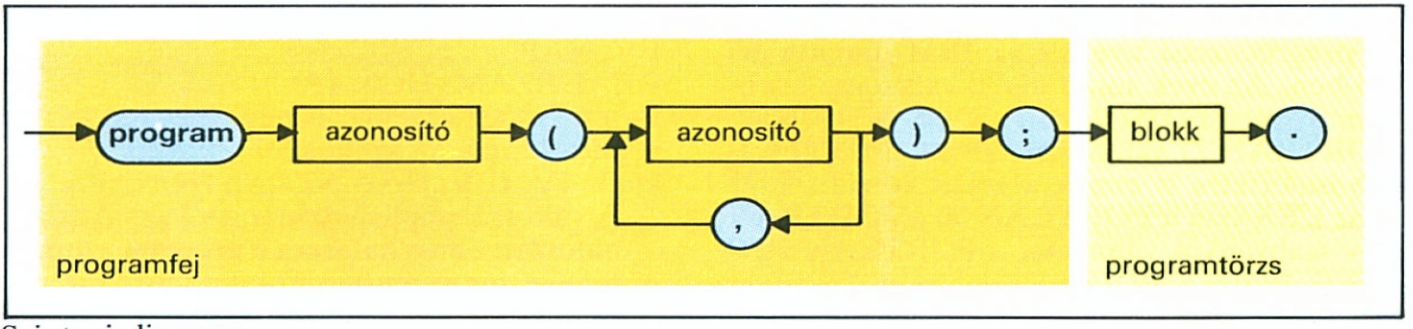
Az **alprogramokat** háromféle módon lehet felírni. Hívásuk az alprogram azonosítójával és a paraméterlistával történik.

Az *utasításfüggvény* kulcsszava FN. Egyetlen utasításból áll és csupán a gyakran előforduló kifejezésekre használatos rövidítés.

A FUNCTION olyan alprogram, amely – az FN utasításhoz hasonlóan – egyetlen értéket (számot, karakterláncot vagy Boole-féle értéket) szolgáltat a hívást tartalmazó programrésznek.

A SUBROUTINE olyan alprogram, amely hívásakor átmenetileg teljes egészében a hívás helyére kerül.

Az ENTRY deklaráció a szubrutinon belül egy alternatív belépési pontot definiál. Lehetővé teszi, hogy a szubrutinokat részekre osszuk, és (az első sor helyett) csak az adott sortól kezdve hívjuk be a főprogram valamilyen részébe.



Szintaxisdiagram

and	**array**	**begin**	**case**	**const**
div	**do**	**downto**	**else**	**end**
file	**for**	**function**	**goto**	**if**
in	**label**	**mod**	**nil**	**not**
of	**or**	**packed**	**procedure**	**program**
record	**repeat**	**set**	**then**	**to**
type	**until**	**var**	**while**	**with**

+	−	*	/	
:=	.	,	;	
:	,	=	<>	
<	<=	>=	>	
(	)			
^	..			

a b c d e f g h i j k l m n o p q r s t u v w x y z

0 1 2 3 4 5 6 7 8 9

Kulcsszavak és karakterkészlet

programfej

program

program neve
be-/kiviteli állományok

programtörzs

deklarációs rész

adattípusok
állandók
változók
címkék
függvények
deklarációk
utasítások
eljárások
deklarációk
utasítások

utasításrész

begin

utasítások

end

Egy program általános felépítése

	jelentés
abs (*x*)	$\lvert x \rvert$
sqr(*x*)	x^2
sqrt(*x*)	$+\sqrt{x}$
sin(*x*)	sin *x*
cos(*x*)	cos *x*
arctan(*x*)	arctg x főértéke
ln(*x*)	ln *x*
exp(*x*)	e^x
round(*x*)	x kerekítve a következő egész számra
trunc(*x*)	x egész része

Standard matematikai függvények

	hatás
read(*x*)	x beolvasása
readln(*x*)	x beolvasása, soremelés
write(*x*)	x kiírása
writeln(*x*)	x kiírása, soremelés

Egyszerű be-/kiviteli utasítások

A PASCAL programozási nyelv alapjait NIKLAUS WIRTH közölte 1971-ben, a nyelv szabványos változata pedig 1983-ban jelent meg. A PASCAL, amelyet BLAISE PASCAL matematikusról neveztek el, az ALGOL és a FORTRAN legjobb elemeit egyesíti.

A PASCAL *imperatív programnyelv,* tehát a program a számítógépnek szóló utasítások sorozata; a bemeneti adatokat a számítógép változóként tárolja és dolgozza fel. A PASCAL szabad formátumú nyelv (angolul: *free format language*): külső felépítésének, pl. a beírt szöveg sorhosszúságának nincs jelentősége, mivel az utasítások végét mindig a következő pontosvessző jelenti. A PASCAL nyelv elemei jól olvasható és könnyen érthető megfogalmazást tesznek lehetővé.

A PASCAL legnagyobb előnye az ALGOL nyelvvel szemben az, hogy az adattípusokból nagy választék áll rendelkezésre, és a programozó még maga is definiálhat újabb adattípusokat. A PASCAL fordítóprogramok nagyon hatékonyak, ami részben annak köszönhető, hogy valamennyi azonosító deklarációja már a program elején megtörténik.

A PASCAL fordítóprogramja a teljes programot lefordítja gépi kódra, és csak ezt követően lehet a programot futtatni. A nagyobb számítógépek a programot közvetlenül gépi kódra fordítják *(one-pass-compiler).* A PC-k közbülső PASCAL-változatokat alkalmaznak, amelyek azonban egyszerűsített alakban is lehetővé teszik a program működését.

A PC-khez használatos, 16 bit szóhosszúságú PASCAL fordítóprogram kb 128 Kbyte RAM tárolóterületet igényel.

A PASCAL kibővített változatai több járulékos utasítással rendelkeznek, és felhasználóbarát programozási segédanyagot tartalmaznak.

A **TURBO PASCAL** nagy és kis betűk használatát is megengedi, ami könnyen érthető értékadó utasításokat tesz lehetővé, mint pl.
EndOfTime : = False;.

A **PASCAL programok** *blokkszerkezetűek,* a programrészek egymásba ágyazódnak. Az egyetlen főprogram tartalmazza az összes alprogram deklarációit mint további blokkokat. Az alprogramok is egymásba ágyazhatók.

A programok általános felépítése. A **program** nyelvelem jelöli a program kezdetét, amelyet a program szabadon választható neve követ. A programot pont (.) zárja le. Valamennyi blokk hasonló felépítésű.

A **programfej** a főprogramban a program nevét és a felhasznált be-/kimeneti adatokat tartalmazza. Az alprogramokban (függvényekben, eljárásokban) a programfej a blokk nevét és a paramétereket jelöli meg. A **programtörzs** az adattípusok, a változók, az állandók, a címkék és az alprogramok deklarációjával kezdődik **(deklarációs rész)**. Ezt követi az **utasításrész,** a megoldási algoritmust tartalmazó tulajdonképpeni program. Az utasításrészt a **begin** és az **end** nyelvelemek zárják közre.

A nyelv néhány eleme

A PASCAL nyelv legtöbb elemét a szintaxisdiagramokkal kapcsolatban már bemutattuk (131. o.)

A PASCAL ábécé jelkészlete a latin kis betűkből, számjegyekből és különleges karakterekből áll. Valóban ábécének nevezhető, mivel az elemeket rendezett sorban tartalmaza, pl. a < b < c < d < ... stb.

A **PASCAL nyelv elemei** betűkből, számokból és különleges karakterekből tevődnek össze. A könnyebb olvashatóság kedvéért az utasítások (és a műveleti jelek) gyakran vastagon írva jelennek meg.

Példák: **label, var, function, if, not, : = .**

A **változók deklarációja** a blokk elején történik a következő alakban:

var *azonosító* **:** *adattípus*;

Példa: **var** i, k, l**: integer;** szam**: real;**

Az **állandók definíciója** a deklarációs részben történik. A megfelelő azonosító az utasításrészben hívja az állandót:

const *azonosító* = *állandó*;

Példa: **const:** pi = 3.1416;

Az **értékadást** a := operációs jel szimbolizálja, vagyis a := jobb oldalán álló kifejezést rendeljük a bal oldalán álló azonosítóhoz.

Példa: x:= pi * r + k;

Az **egész számokkal végzett műveletek integer** típusú azonosítókkal történnek. A műveleti jelek a következők: +, −, *, **div,** =, <, <=, >>=, <> és **mod.**

A **valós számokkal végzett műveletek real** típusú azonosítókkal történnek. A műveleti jelek a következők: +, −, *, /, valamint az előbb felsorol relációs műveleti jelek. A valós számok a *standard függvények* paramétereiként is előfordulnak.

A **tömbök** csak 1-dimenziósak lehetnek, és indextartományukat a program elején deklarálni kell. A tömbelemek tetszőleges adattípushoz tartozhatnak.

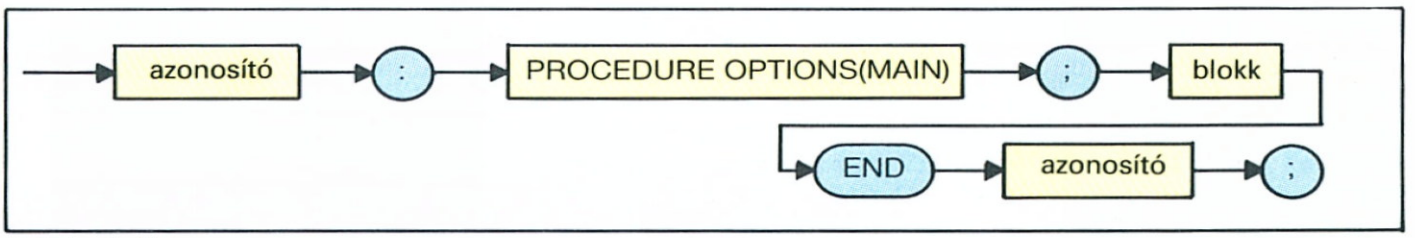
Példák: v1**: array** [1..8] **of real;**
v2**: array** [7..12] **of integer;**

Az n-dimenziós tömbök indexes tömbelemeket alkalmaznak, pl.:
vector**: array** [1..8] **of array** [7..12] **of** v2;
(Ez egy 8×6-os mátrix.)

Az **alprogramok** kétféleképpen írhatók fel. Hívásuk az alprogram azonosítójával és a paraméterlistával történik.

function. Hívásakor csak egyetlen érték adódik át a hívó programnak.

procedure. Hívásakor az eljárástörzs mint egész átmenetileg a főprogramba kapcsolódik.



Szintaxisdiagram

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

AUTOMATIC	EXTERNAL	+ − * /
BASED	FIXED	** (kitevő)
BINARY	FLOAT	|| (karakterkapcsolás)
CONTROLLED	POINTER	& (és)
DECIMAL	STATIC	| (vagy)
		¬ (nem)
		= > < >=
		<= ¬= ¬< ¬>
		% (fordításkor végrehajtandó)

Kulcsszavak, jelkészlet

STREAM-adatok:		RECORD-adatok:	
GET LIST(*x*)	*x* beolvasása standard formában	READ(*x*)	*x* rekord beolvasása, tárolás a beolvasottal megegyező formában
PUT LIST(*x*)	*x* kivitele standard formában	WRITE(*x*)	*x* rekord kivitele, a beolvasottal megegyező formában
GET DATA	a következő adat beolvasása		
PUT DATA(x)	*x* azonosítójának és értékének kivitele		
GET EDIT((*x*), *format*	*x* beolvasása deklarált formában		
PUT EDIT((*x*), *format*	*x* kivitele deklarált formában		

Egyszerű B/K-függvények

```
a program neve : PROCEDURE OPTIONS(MAIN);
                    deklarációk
                    utasítások

1. név: PROCEDURE ( paraméter ) RETURNS ( adattípus );
                    paraméterdeklaráció
                    deklarációk
                    utasítások
                RETURN ( azonosító );
                END 1. név ;

2. név; PROCEDURE ( paraméter );
                    paraméterdeklaráció
                    deklarációk
                    utasítások
                RETURN;
                END 2. név ;

                END programnév ;
```

függvények

eljárások

PL/1 program általános felépítése

Az IBM az 1960-as években fejlesztette ki az univerzálisan felhasználható **PL/1** (angolul: *Programming Language Number* **1**) programnyelvet. Az első szabványos verziója 1976-ban jelent meg. A PL/1 nyelv egyesíti és kiegészíti a FORTRAN (formázott adatok, az alprogramok független fordítása), az ALGOL (blokk struktúra) és a COBOL (sokrétű adattípusok) előnyeit.

A PL/1 *szabad formátumú imperatív* programnyelv. A PL/1 számos mennyiséghez *attribútumot* rendel. Ez hatékony programozást és sokrétű adatkezelést tesz lehetővé. Az expliciten nem deklarált attribútumok implicit módon deklarálódnak. Lehetőség van a hibafeltételek és megfelelő kezelésük programozására is.

Példa: Az ON OVERFLOW ... utasítással rögzíteni lehet, hogy mi történjen akkor, ha e bizonyos mennyiség értéke a megengedett számtartományt túllépi.

A PL/1 fordítóprogramok a forrásprogramot egészként kezelik, és a deklarációkat utólag a program elejére helyezik. A fordítás első menete után a programon még lehet változtatni.

Többféle PL/1 változat – pl. kibővített változat is – létezik, de legtöbbjük a PL/1 részhalmazának tekinthető, mint pl. a PL/C.

A PL/1 programnyelv nem terjedt el széles körben.

A **PL/1 program** *blokkszerkezetű*. Az (egyetlen) főprogram tartalmazza az összes alprogram deklarációit mint további blokkokat. Az alprogramok is egymásba ágyazhatók. A főprogramot és az alprogramokat egymástól függetlenül lehet gépi kódra fordítani.

A programok általános felépítése. A program elejét a program neve és a PROCEDURE OPTIONS(MAIN); karaktersorozat jelöli. A program vége az END nyelvi elemmel, valamint a program nevével és pontosvesszővel zárul.

A program felépítése nem követ merev struktúrákat, még a deklarációk elhelyezése is tetszőleges.

A nyelv néhány eleme

A **PL/1 jelkészlete** a latin nagy betűkből, számjegyekből és különleges karakterekből áll.

A **PL/1 nyelv elemei** betűkből tevődnek össze, nincsenek fenntartott szavak.

Példák: IF, THEN, END, RETURNS, GET, GENERIC.

A PL/1 nyelv elemeit rövidíteni lehet – ez a program olvashatóságát nem javítja. A fordítóprogram pl. a DEC rövidítést DECIMALként értelmezi.

A **változódeklaráció** – beleértve az attribútumok deklarációját is – szabadon kezelhető. Az adattípus implicit deklarációja az azonosító első betűjének alapján történik, mint a FORTRAN nyelvben.

A **számoknak** négy *attribútuma* van: valós/komplex, fixpontos/lebegőpontos, decimális/bináris és a jegyek száma, azaz a pontosság.

Példa: DECLARE X REAL FIXED DECIMAL(8, 4);

Az X azonosító értéke valós, decimális fixpontos szám, összesen 8 jegye van, ebből 4 a tizedespont után áll.

Az **értékadást** az = műveleti jel szimbolizálja, vagyis az = jel jobb oldalán álló kifejezés értékét rendeljük a bal oldalán álló azonosítóhoz (vagy azonosítókhoz). Az adattípusok átalakítása automatikusan történik.

Példa: X, Y = PI * R + K;

A **műveletek** a +, –, *, /, ** (exp.) valamint 8 relációs műveleti jel segítségével történnek. A műveleti jelek a tömbökre mint egészekre is érvényesek. A standard függvényeket minden (értelmes) adott adattípusra alkalmazni lehet.

A **tömböket** (a tömbelemek adattípusát és az indexek felső határát) deklarálni kell. A legtöbb fordítóprogram maximálisan három dimenziót engedélyez, és csak > 0 indexeket fogad el.

Példa:

DECLARE VECTOR(8, 12) FLOAT;

(8×12-es mátrix deklarálása, a tömbelemek lebegőpontos számok.)

Lehetőség van a tömbök indextartományának dinamikus deklarálására is (ilyenkor az indextartomány felső határa csak a program futásakor kerül meghatározásra).

Az **alprogramok** kétféle módon alakíthatók ki. A paraméterek adattípusát explicit módon deklarálni kell. Az alprogramok maguk is tartalmazhatnak alprogramot.

Az olyan alprogramokat, amelyek csak az adattípusban különböznek, a GENERIC nyelvelemmel lehet deklarálni.

A rekurzív alprogramokat külön ilyen típusúnak kell deklarálni (erre szolgál a RECURSIVE nyelvi elem).

Az alprogramok hívása az alprogramok neve és a paraméterlista segítségével történik.

A **függvények** hívásukkor a hívó programnak csak egyetlen értéket adnak át.

A függvény első sorának általános alakja:

Függvénynév: PROCEDURE *(paraméter)* RETURNS *(adattípus);*

Az **eljárások** hívásukkor mint egészek, átmenetileg a hívó programba kapcsolódnak. Az eljárás első sorának általános alakja:

Eljárásnév: PROCEDURE *(paraméter);*

Valamennyi alprogram utolsó utasítása:

END *alprogramnév;*

	ALGOL 60	BASIC
a program kezdete	**procedure** *név(paraméter);*	*sorszám utasítás*
magyarázó szöveg	**comment** *szöveg;*	REM *szöveg*
típus deklaráció	**integer** *azonosító;* **real** *azonosító;* **boolean** *azonosító;*	1. betű I,...,N " A,...,H,O,...,P implicit módon
értékadó utasítás	*azonosító := kifejezés;*	*azonosító = kifejezés*
utasítás vége	;	sor vége
feltétel nélküli ugró utasítás	**go to** *címke;*	GOTO *sorszám*
tömb	*azonosító[kifejezés];* *azonosító[A1, A2, A3];*	*azonosító(változó)* *azonosító(V1,...,Vn)*
függvényhívás	*név(paraméter);*	FN*név(paraméter)*
eljáráshívás	*név(paraméter);*	GOSUB*(sorszám)*
a program vége	**end** *név*	*sorszám* END

Különböző programnyelvek néhány eleme

nyelv	feltételes utasítás	valódi alternatíva
ALGOL 60	**if** *feltétel* **then** *utasítás*	**if** *feltétel* **then** *1. utasítás* **else** *2. utasítás*
BASIC	IF *feltétel* THEN *utasítás* vagy IF *feltétel* GOTO *címke*	IF *feltétel* THEN *1. utasítás* ELSE *2. utasítás*

FORTRAN	IF *feltétel* THEN *utasítás*	IF *feltétel* THEN
		1. utasítás
		ELSE
		2. utasítás
		ENDIF

PASCAL	**if** *feltétel* **then** *utasítás*	**if** *feltétel* **then** *1. utasítás* **else** *2. utasítás*
PL/1	IF *feltétel* THEN *utasítás*	IF *feltétel* THEN *1. utasítás* ELSE *2. utasítás*

Feltételes utasítások

Az 1960-as években az MIT (Massachusetts Institute of Technology, USA) munkatársa JOHN MCCARTHY kifejlesztette a **LISP** (angolul: **LIS***t* ***P****rocessing*) programnyelvet. Ez a nyelv a hosszú futási idők (és a számtalan dialektus) miatt csak az 1970-es évek végén terjedt el. Valójában nem is közönséges értelemben vett programnyelv, bár a szokásos programnyelvekhez hasonló elemei is vannak. A LISP programok – elsősorban a számtalan zárójel miatt – áttekinthetetlenek, és kevéssé tekinthetők felhasználóbarátnak.

LISP egy *applikatív programnyelv,* vagyis a programok elsősorban függvénydefiníciókból és -hívásokból állnak.

LISP alkalmas a nem-numerikus problémák – mindenekelőtt a szimbólumok és különböző struktúrák között fennálló kapcsolat – vizsgálatára. Felhasználása elsősorban a tételek bizonyítása a mesterséges intelligencia és a nyelvelemzés területére esik.

A LISP alapelemei az **azonosítók,** amelyeket a LISP nyelvben **atomoknak** nevezünk.

Az atomok betűkből, számjegyekből, jelekből, betű- és jelsorozatokból állnak. Magukat az atomokat csak ritkán kell manipulálni, azonban a tulajdonságaikat leíró táblázatokat gyakrabban.

Példák atomokra: 22, C, REGGEL.

Az atomok képezik az (a.b) alakú – **listának** nevezett – kifejezések alapjait.

Az a és b atomok maguk is szimbolikus kifejezések lehetnek.

Példák: (X.NIL), ((REGGEL.12).NIL), (AGFA.((FILM.(DIN.NIL))))

A NIL listaelem üres listát jelent.

A függvények az atomok és listák kezelésére szolgálnak. A függvénynév az első listaelemként jelenik meg. A lista többi eleme a függvény paramétere.

Példák: az A + (B * C) művelet listaként (PLUS A(TIMES B C)) alakban jelenik meg.

A listák feldolgozására három **alapfüggvény** áll rendelkezésre:

– CAR egy lista bal oldali elemét szolgáltatja:
(CAR(A B C D)) = A.

– CDR a bal oldali elem kivételével egy lista valamennyi elemét szolgáltatja:
CDR (A B C D)) = (B C D).

– CONS a függvény első és második paraméterét egyetlen listává egyesíti:
(CONS A (B C D)) = (A B C D).

Míg az imperatív programnyelvek elsősorban az iterációt (ciklust) alkalmazzák, a LISP nyelv súlypontja a rekurzió. Blokkszerkezetek nem lehetségesek. Alprogramokat egyetlen kifejezésként lehet megadni.

A LISP-programot interpreter dolgozza fel, vagyis a számítógép a programot sorról sorra hajtja végre.

Az adatfeldolgozás területén a LISP nyelv használt először *Garbage Collection* eljárást: a többé már nem használt adatok a tárolóban periodikusan kitörlődnek.

Az 1970-es évek kezdetén DENNIS RITCHIE (a BELL-Laboratórium munkatársa) az USA-ban egy új programnyelvet fejlesztett ki: a **C**-t. Ez a nyelv lényegében a PASCAL alapjaira épít, de asszemblerhez hasonló nyelvi elemeket is tartalmaz. A nyelv terjedelme viszonylag kicsi, különösen be-/kiviteli műveletek valamint a standard függvények hiányoznak. A hiányzó elemeket a felhasználó programkönyvtárakból választhatja ki, amelyeket külön le kell fordítani és **include** állománynévként a C-program elején a főprogramba kell illeszteni.

A számos rövidítés és az asszemblerrel való hasonlóság miatt a C nyelven írt programok viszonylag áttekinthetetlenek.

A C nyelv azonban mégis nagyon elterjedt, mert minden olyan számítógépen, amely az igen sikeres UNIX operációs rendszerrel működik, a C nyelv elérhető. Ezenkívül C nyelven a viszonylag kisméretű nyelvi terjedelme miatt hatékony és gyorsan futó programok írhatók.

Időközben megjelent a párhuzamos számítógépeken is futtatható C^{++} nyelv is.

A C nyelv feladatorientált nyelvek elemeiből átvette a deklarációt, az adattípusokat, a ciklusokat, a feltételes utasításokat, az ugrásokat, a blokkszerkezeteket és a függvényeket. A logikai adattípusok viszont hiányoznak.

Az asszemblerből átvett elemek pl. a változók pontosságának deklarációja, és a memóriaterület lefoglalásának lehetősége. A futási sebesség fokozására szolgál, hogy a kritikus mennyiségeknek regiszterekben lehet helyet foglalni.

A **változókat** deklarálni ill. definiálni kell. A globális változókat a program elején definiálni, és azon a helyen, ahol fellépnek, **extern** típusúként deklarálni is kell. Az azonosítókon és az alapvető **int, float** és **char** adattípusokon kívül a C nyelv tárolási osztály szerinti megkülönböztetést is tartalmaz.

Tárolási osztályok:

auto. A változó csak azon a blokkon belül ismert, amelyben deklaráltuk, tehát lokális változó.

register. Az előzőhöz **(auto)** hasonló, de egyetlen regiszterben kerül tárolásra.

static. A változó a blokk elhagyása után is megtartja értékét, tehát globális változó.

Létezik a *record*-hoz hasonló strukturált adattípus is – a C nyelvben ennek elnevezése: **struct** –, amely lehetővé teszi több adattípus összefogását. A tömbindex zérus értéket is felvehet, dinamikus tömbdeklaráció viszont nem lehetséges.

Alprogramként – akár paraméterekkel, akár paraméterek nélkül – függvények használhatók, eljárások azonban nincsenek. Függvény nem jelenhet meg másik függvény paramétereként.

Függvényhíváskor a program sem a teljességet, sem az adattípusokat nem vizsgája meg.

A forrásnyelvű C programok mint egészek kerülnek gépi kódú fordításra és végrehajtásra.

FORTRAN	PASCAL	PL/1
sorszám PROGRAM	**program** *azonosító* (In/Outfile)	*név:* PROCEDURE OPTIONS(MAIN);
C *szöveg* (C áll az 1. oszlopban)	{*szöveg*} vagy {* *szöveg* *}	/**szöveg* */
1. betű I,...,N " A,...,H O,...,Z implicit	*azonosító,* ... **:integer;** *azonosító,* ... **:real;** *azonosító,* ... **:boolean;**	DECLARE *azonosító,* ... FIXED; DECLARE *azonosító,* ... FLOAT; DECLARE *azonosító,* ... LOGICAL;
azonosító = kifejezés	*azonosító := kifejezés;*	*azonosító = kifejezés;*
sor vége	;	;
GOTO *sorszám*	**goto** *címke*	GOTO *címke;*
azonosító(változó) *azonosító(V1,...,Vn)*	*azonosító [kifejezés]* *azonosító [A1,...,An]*	*azonosító(változó)* *azonosító (V1,...,Vn)*
név(paraméter)	*név(paraméter)*	*név(paraméter)*
CALL *név(paraméter)*	*név(paraméter);*	*név(paraméter);*
sorszám END	.	END *név;*

Összeállítás a különböző programnyelvek elemeiből

for *ciklusváltozó* **:=** *kezdőérték* **step** *lépésköz* **until** *végérték* **do** *utasítás*

FOR *ciklusváltozó = kezdőérték* TO *végérték* STEP *lépésköz*

utasítás
NEXT *ciklusváltozó*

DO *címke ciklusváltozó = kezdőérték, végérték, lépésköz*
utasítás
címke CONTINUE

for *ciklusváltozó := kezdőérték* **to** *végérték* **do** *utasítás*
(lépésköz + 1)
for *ciklusváltozó := kezdőérték* **downto** *végérték* **do** *utasítás*
(lépésköz – 1)

DO *azonosító = kezdőérték* TO *végérték* BY *lépésköz;*
utasítás END;

Számlálóciklusok

Az **ADA** *imperatív programnyelvet* AUGUSTA **ADA** BYRONRÓL (1815–1852), CHARLES BABBAGE munkatársnőjéről, az első programozónőről nevezték el. A nyelvet, amely a NATO hivatalos programozási nyelve, az 1970-es évek közepén fejlesztették ki az amerikai Védelmi Minisztérium megbízásából. Alapjául egy kiterjesztett, további nyelvelemeket tartalmazó PASCAL-változat szolgált.

Az ADA különálló modulokból áll, amelyeket **csomagnak** (angolul: *package*) nevezünk. Minden csomag a többitől függetlenül írható meg, és gépi kódra történő fordításuk is külön történik. A modulokat pl. grafikus programok vagy be-/kiviteli utasítások alkotják. A különböző modulok azonosítói formálisan megegyezhetnek. A teljes program az egyes modulokból épül fel.

Taszknak (angolul: *task*) nevezzük azokat a speciális modulokat, amelyek párhuzamos üzemmódban dolgozhatók fel, és egymással kölcsönhatásba is léphetnek. Időbeli lefutásukat egymással egyeztetik, de külső órával történő vezérlésük is lehetséges.

Az ADA nyelvben – a ciklusváltozók kivételével – minden **azonosítót** deklarálni kell. Ehhez **integer, float, character, string** és **boolean** adattípusok állnak rendelkezésre. A numerikus mennyiségek kívánt pontosságát rögzíteni lehet. A nyelv elemei, pl. az értékadó, az ugró utasítás, a ciklus, a feltételes utasítás és a blokk a PASCAL nyelv megfelelő elemeivel egyeznek meg.

A felhasználó a szokásos adatstruktúrákon, pl. a tömbön, rekordon és a karakterláncon kívül saját adatstruktúrákat is létrehozhat. A tömbök dimenzióinak száma nincs korlátozva. Az indextartomány tetszőleges és dinamikusan deklarálható. A nyelv a korlátozott indextartományú résztömbök meghívását is lehetővé teszi.

Ezzel a módszerrel lehetséges pl. az is, hogy adott vektorból egy részletet „kivágjunk": a SZALAG(4..9) utasítás pl. a SZALAG(4)-től a SZALAG(9)-ig terjedő hat vektorkomponenst meghívja.

A függvények és az eljárások kezelése a PASCAL alprogramokéhoz hasonlóan történik.

Az ADA nyelv szokatlan tulajdonsága, hogy a programozó az esetleg fellépő végrehajtási hibák kezelését minden modulra előre rögzítheti.

Példák: a program megszakítása és hibajelzés, folytatás és hibajelzés. Ha a hiba, pl. egy értéktartomány-túllépés lehetősége előre látható, akkor a hiba megszüntetésére szolgáló eljárást előre be lehet programozni.

A nyelv nagy terjedelme miatt az ADA fordítóprogramjai bonyolultak és sok tárolóterületet igényelnek.

Az 1950-es évek végén az USA-ban GRACE HOPPER a kereskedelmi folyamatok programozására új programnyelvet fejlesztett ki. Ez a nyelv a **COBOL** (angolul: **CO***mmob* **B***ussiness* **O***rient-ed* **L***anguage*). Egyetlen másik nyelven sem írtak annyi kereskedelmi forgalomban lévő programot, mint a COBOL nyelven. Ennek ellenére a COBOL – a PL1 kivételével – nem befolyásolta a programnyelvek fejlődését. A COBOL-nak számos változata létezik, ezek azonban jól „átvihetők" egyik számítógépről a másikra.

A COBOL nyelvet gyakorlatilag csak a létező egyéb nyelvek ellenőrzésére, karbantartására használják.

A COBOL egyszerű, *imperatív programnyelv*. Mivel megengedi a kitöltő szöveg használatát, a program szinte folyamatosan olvasható. A COBOL számos be-/kimeneti utasítással rendelkezik, ezzel szemben magasszintű nyelvelemeket, pl. blokkokat, iterációkat, eljárásokat nem tartalmaz.

A COBOL lehetővé teszi a végrehajtási hibák (többek között a túlcsordulás) kijavítását is, pl. a következő utasítás segítségével:

ADD VAT TO AR, ON SIZE ERROR PERFORM KORR1.

A programok formális felépítésében még felismerhetők a lyukkártya idejéből származó kényszerű elemek: egy sorba csak egy utasítás írható, soronként 80 pozíció használható.

Minden program négy részből áll:

Az AZONOSÍTÁSI FŐRÉSZ (angolul: *identification division*) a program általános dokumentációját, a programnyelvet, a dátumot és a megjegyzéseket tartalmazza.

A KÖRNYEZETLEÍRÓ FŐRÉSZ (angolul: *environmental division*) írja le a gépi környezetet: a számítógép típusát, a fordítóprogramot, az illesztőegységeket, a be-/kiviteli eszközöket, az állományok hozzárendelését stb.

Az ADATLEÍRÓ FŐRÉSZ (angolul: *data division*) tartalmazza a felhasznált tárolóterület, az állományok, az állandók stb. adatait.

Az ELJÁRÁSI FŐRÉSZ (angolul: *procedure division*) tartalmazza az adatok feldolgozására szolgáló tulajdonképpeni utasításokat.

Minden **azonosítót** deklarálni kell, ehhez nagy számú adattípus áll rendelkezésre. A PICTURE kulcsszóval történő deklaráció rendkívül rugalmas adatábrázolást tesz lehetővé.

Példa: AR PICTURE $$$ $$$ $$9.99.

Az AR azonosító értéke maximálisan 8 egész és 2 tizedesjegyből álló szám. A szám elején álló nullák nem jelennek meg a kiírásban, és közvetlenül az első számjegy előtt dollárjel áll.

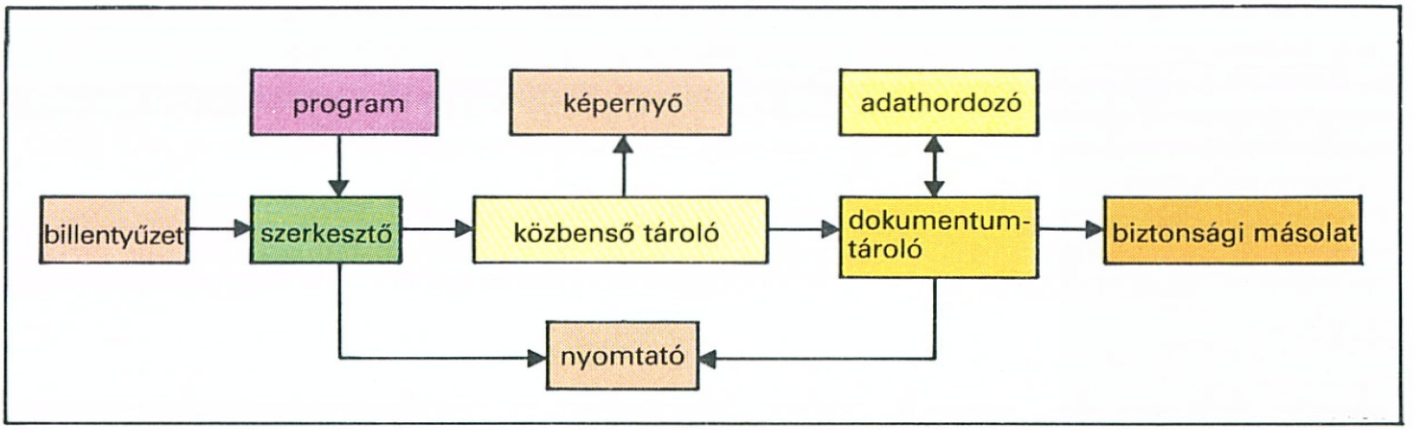
A COBOL nyelv az adatrekordokon kívül max. 3-dimenziós **tömbök** használatát engedi meg. Az adatrekordok tömbelemek lehetnek és fordítva.

Az alprogramok címkézett utasításcsoportok, amelyek a

PERFPRM *címke1* THRU *címke 2.*

utasítással hívhatók.

Egyszerű szintaxisa miatt a COBOL programok írása fáradságos és hosszadalmas. Csak az 1980-as évek óta állnak rendelkezésre megfelelő gyors fordítóprogramok.



Szövegszerkesztő program komponensei

Egy dokumentum nyers változata:

a feldolgozás pozíciója

fejléc → *D*: SH 45 PAGE 1 LINE 4 COLUMN 20

MENÜ

H	magyarázatok	I	S	címsor	I	^H	súgó
B	bekezdés formázása	I	R	aktuális sor	I	^K	blokk
F	jelzők a jobbszélen	I	M	szegély	I	^P	nyomtatás
D	nyomtatási forma	I	P	jelölés	I	^O	képernyő
I	utasítások jegyzéke	I	V	szövegmozgatás	I	^B	félkövér

szöveg bal széle L----*----*----*----*----*----*----*----*----*----*----*R

^BSzövegszerkesztés^B

szöveg jobb széle

kurzor-pozíció →

A PC-k leggyakoribb alkalmazási területe a szövegek számítógépes tárolása, szerkesztése és feldolgozása. ← feldolgozás alatt álló sor

^Bdokumentumnak^B nevezzük a feldolgozott szöveget.

szerkesztési jel

A szövegfeldolgozó program központi része a ^Bszerkesztő (editor); ^B ez, olyan szoftver, amely általában az operációs rendszerhez is hozzáfér. A PC-khez nagymértékben specializált szövegszerkesztő programok állnak rendelkezésre, ilyenek például, az MS Word vagy a WordStar. ← túl hosszú sor

A szöveg formális szerkezete, mint pl. a sorszélesség, oldalszám, a sorok oldalankénti száma, a betűméret, a grafikus ← a szótár szerint elírási hiba

1KURSIV 2HELP 3SET LM 4SET RM 5SAVE 6SUPER 7SUB 8SAVE-T 9NORMAL ← a speciális billentyűk funkciói

Szerkesztés és formázás utáni változat:

D: SH 45 PAGE 1 LINE 5 COLUMN 18

L----*----*----*----*----*----*----*----*----*R 1

Szövegszerkesztés 2

A PC-k legyakoribb alkalmazási területe a szövegek számítógépes tárolása, szerkesztése és feldolgozása. A feldolgozott szöveget **dokumentumnak** nevezzük. A szövegfeldolgozó program központi része a **szerkesztő (editor);** ez olyan szoftver, amely általában az operációs rendszerhez is hozzáfér. A PC-khez nagymértékben specializált szövegszerkesztő programok állnak rendelkezésre, ilyenek például az MS Word vagy a WordStar. 3 4

5 A szöveg formális szerkezete, mint pl. a sorszélesség, oldalszám, a sorok oldalankénti száma, a betűméret, a grafikus

Szerkesztés, formázás

1 soronként 60 leütés
2 két margóhoz illesztés
3 elválasztás szerkesztési jel nélkül
4 hibajavítás
5 sort beljebb kezd

Szövegszerkesztés a WORDSTAR programmal: a szövegrész a szerkesztés alatt, és a szerkesztés után

A PC-k leggyakoribb alkalmazási területe a szövegek számítógépes tárolása, szerkesztése és feldolgozása. A feldolgozott szöveget **dokumentumnak** nevezzük. A szövegfeldolgozó program központi része a **szerkesztő (editor);** ez olyan szoftver, amely általában az operációs rendszerhez is hozzáfér. A PC-khez nagymértékben specializált szövegszerkesztő programok állnak rendelkezésre, ilyenek pl. az MS Word, a WordPerfect, vagy az AmiPro.

Az 1980-as években a szövegszerkesztő programok átlagosan kb. 15 000 utasítást tartalmaznak. Egy évtizeddel később ez a szám már félmillióra tehető.

A szövegszerkesztő programok legnagyobb fejlettségi fokát a *kiadványszerkesztés* (*Desktop Publishing,* DTP) jelenti, amely az ábrákat is beleértve már nyomdakész végterméket hoz létre.

Üzemmódok

Párbeszédes üzemmódban működő szövegszerkesztő rendszerek. A billentyűzet segítségével írt szöveg, valamint minden változtatás közvetlenül az adatbevitel után megjelenik a képernyőn.

A szövegszerkesztés **WYSIWYG** (angolul: ***W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et*) elve azt jelenti, hogy a tárolt és behívható szöveg ábrázolás a képernyőn – a különleges karakterekkel és a grafikával együtt – pontosan megfelel a nyomtatási képnek.

A **kötegelt feldolgozású szövegszerkesztő rendszerekben** a szöveg az adathordozón teljesen kész formában rendelkezésre áll, csupán a név, a cím, a megszólítás stb. részeket kell beilleszteni. A dokumentum külső megjelenési formája (mint pl. a sortávolság, a felhasznált betűkészlet, betűtípus) egészként változik meg.

A szövegszerkesztés fázisai

A szöveg megtervezése. A szöveg beírása a számítógépbe párbeszédes üzemmódban és gyakorlatilag mindig billentyűzet segítségével történik. A beírt szöveg a képernyőn rögtön megjelenik. A rutinszerűen ismétlődő folyamatokat a *szövegmodulok* használata nagymértékben egyszerűsíti. A bonyolult kifejezéseket, megszólításokat, üdvözlő fordulatokat, valamint egész szövegrészeket tárolni lehet, ezeket kívánság szerint újra elő lehet hívni, és a szövegbe be lehet illeszteni. A gyakran használt utasításokat *makrók* segítségével automatizálni lehet; a makrók ezen túlmenően programfeladatokat is elláthatnak.

A szövegmodulok és a makrók kiválasztása háromféleképpen történhet: rövidítésekkel, menün keresztül, vagy egérrel egy grafikus szimbólumra „kattintással" a képernyőn.

Egy **szöveg módosítása** a szöveg részeinek javítását, törlését, beillesztését vagy a szöveg átformázását jelenti. A változtatás helyét a kurzor pozíciója határozza meg. Arra is lehetőség van, hogy a felhasználó más dokumentumokból fejezeteket vegyen át. A korábban kijelölt szavakból szójegyzéket lehet összeállítani. A legtöbb szerkesztőprogramban szótár, valamint olyan eljárás is rendelkezésre áll, amely a helyesírási hibákat felderíti, kijelzi, és kívánságra kijavítja.

Formázás. A nyers dokumentumot speciális utasításokkal lehet formázni: ez azt jelenti, hogy a sorszélességet, az elválasztást, a két margóhoz igazított (a két margóhoz kizárt) ábrázolásmódot, az oldalhosszúságot, betűtípust, az aláhúzást stb. meg lehet határozni.

Tárolás. Mivel a dokumentumok rendszerint hosszabbak, mint a képernyőn megjeleníthető sorok száma, ezért a szövegszerkesztő program két tárolót használ. Az egyik – pl. a merevlemezes tároló – az egész dokumentumot tartalmazza, a másik – pl. a központi egység RAM operatív tára – csak azt a részletet, amely éppen feldolgozás alatt áll. Ha a felhasználó a szövegen végrehajtott változtatásokat befejezettnek tekinti, akkor a dokumentumot a tároló utasítással egy adathordozóra rögzíti.

A legtöbb szövegszerkesztő program az éppen feldolgozás alatt álló szövegrészt a gyors operatív memóriából rendszeres időközönként automatikusan a merevlemezre másolja. Mivel azonban a háttértároló is megsérülhet, érdemes biztonsági másolatokat (angolul: *backup copies*) készíteni, vagyis az adatokat egy külső adattárolón (hajlékony mágneslemezen vagy mágnesszalagon) is érdemes eltárolni. Ezek a biztonsági másolatok mérsékelik azokat a károkat, amelyek hibás kezelés vagy egyéb okok miatt keletkezhetnek. Ezen túlmenően számos felhasználói program az adatállomány mindenkori legutolsó változatát SIK vagy BAK kiterjesztéssel eltárolja. Így adatvesztés esetén a biztonsági adatfile-okra *(backup files)* lehet támaszkodni.

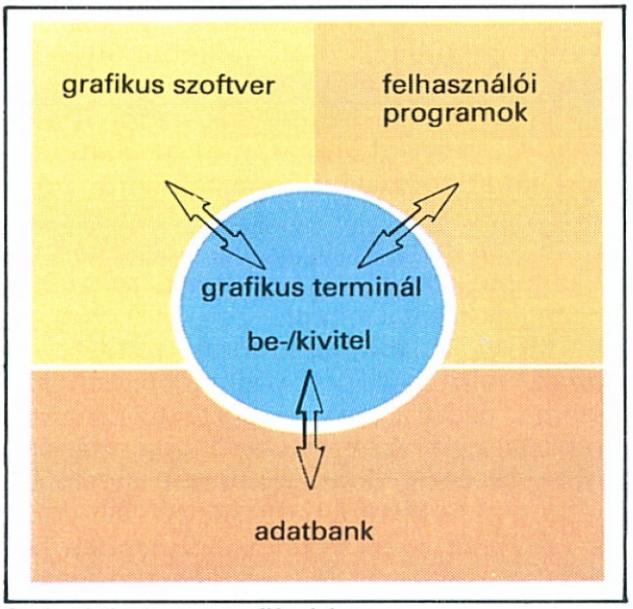
További felhasználás. Az adathordozón található dokumentumot sokoldalúan lehet más célokra is felhasználni, pl. automatikusan ki lehet nyomtatni, meg lehet címezni, borítékolni lehet és el lehet küldeni.

Szerkesztőprogram (editor)

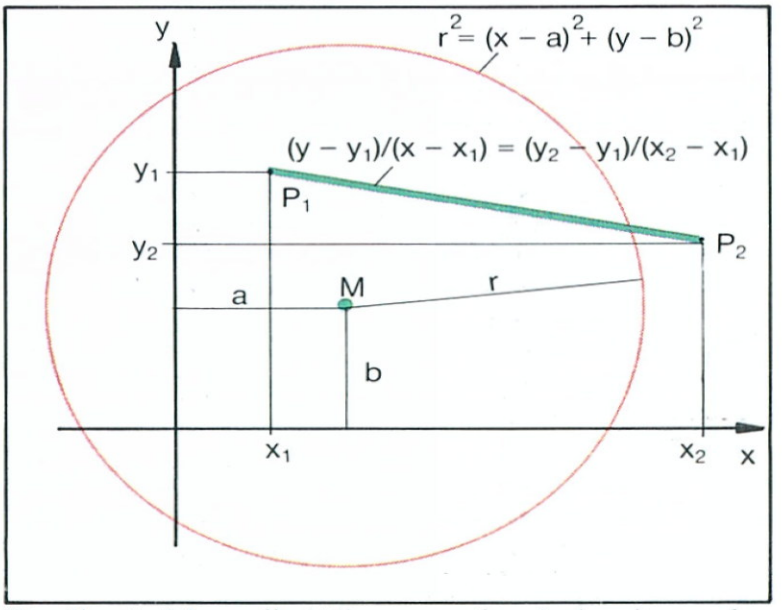
Nagy számítógépeken a speciális szövegszerkesztő programok helyett gyakran az operációs rendszerhez tartozó szoftvert, az ún. *editort* lehet használni. Az editorok tulajdonképpen a programok írására szolgálnak, ezért pl. szövegek formázására csak korlátozottan alkalmasak.

A **sorszerkesztők** (angolul: *line editors*) a lyukkártyát szimulálják, azaz a szöveget sorszámozott, legtöbbször 80 karakter hosszúságú sorokra bontják. A szövegnek egyszerre mindig csak egyetlen sorát lehet feldolgozni. A sorszerkesztőket már alig használják.

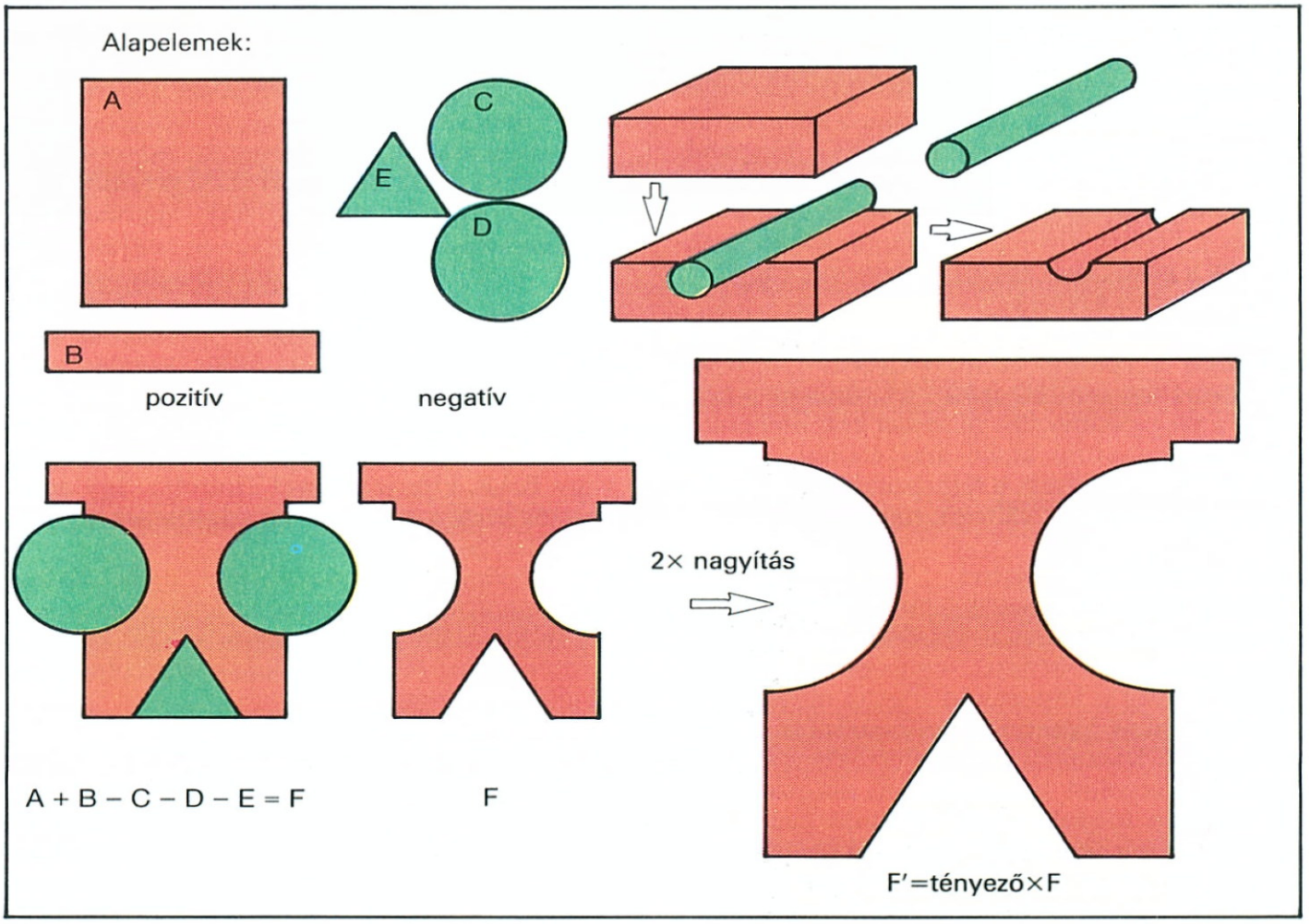
A **teljes képernyős szövegszerkesztők** *(full screen editors)* a monitort olyan négyzetekre osztják *(parketting)*, amelyek mindegyike csak egy jelet tud felvenni. A szöveges jelek korlátos számú halmaza a képernyőn *ablakként* jelenik meg.



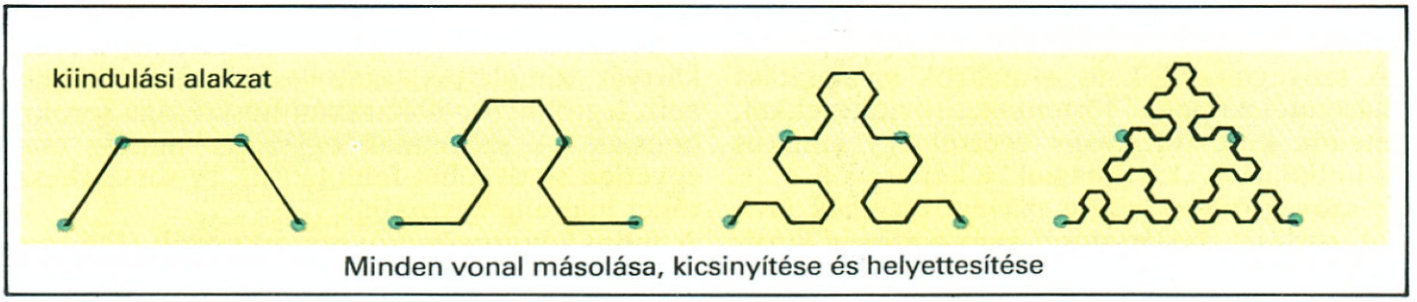
Számítógépes grafikai környezet



Derékszögű koordinátákra vonatkozó algoritmusok



Pozitív és negatív elemek manipulációja



Fraktálképzés egy grafikus alapelem felhasználásával

A **számítógépes grafika** (angolul: *computer graphics*) grafikus adatok bevitelét, tárolását, feldolgozását és kivitelét jelenti. A felhasználó a megfelelő szoftverrel dialógust folytat, ennek segítségével kezeli a grafikus megjelenítést, mindaddig, amíg az kívánságainak megfelel, és végül egy kiviteli eszközön pl. egy plotteren megjeleníti.

A **grafikus magrendszer** (angolul: ***G**raphics **K**ernel **S**ystem*, **GKS**): a számítógépes grafika szerkesztésére és generálására szolgáló alapvető függvényeknek a felhasználástól és a hardvertől teljesen független meghatározása.

A számítógépes grafikus rendszerek akkor eredményesek, ha
- implementálásuk és alkalmazásuk *egyszerű*,
- a legtöbb alkalmazás számára *teljesek*,
- a felhasználó által elkövetett kis hibákkal szemben *toleránsak*,
- az alkalmazott hardveren *hatékonyan* és *gyorsan* futtathatók,
- memóriakihasználásuk *gazdaságos*.

Grafikus rendszerek

A grafikus rendszerek három részből állnak:
1. interaktívan alkalmazható grafikus **szoftver.**
2. A speciális felhasználók (pl. építészek, géptervezők, elektrotechnikusok, elektronikusok stb. igényeihez illeszkedő **felhasználói program.** Ide tartoznak a gazdasági és a tudományos adatok grafikus ábrázolását és kiértékelését végző programok is, különösen az interpolációra és az extrapolációra használt programok.
3. Könnyen hozzáférhető **adatbank.** Ezek nemcsak a grafikus rendszerek előző két részéhez tartozó algoritmusokat tárolják, hanem pl. az építészet legfontosabb alakzatait, elektronikus kapcsolási rajzokat, betűtípusokat, koordinátarendszereket, alfanumerikus jeleket, matematikai képleteket, DIN szabványokat, tervezési és építési előírásokat stb., valamint a felhasználó által kifejlesztett minden programot és algoritmust is tartalmazzák, és azonnal hozzáférhető állapotban – pl. a képernyőn való megjelenítésre – készenlétben tartják.

A modern, feladatorientált programnyelvek (többek között a TURBO PASCAL) számítógépes grafikai elemeket is tartalmaznak. Ilyen elemek pl. a koordináta-rendszerek és adatpontok képernyőn történő ábrázolása.

Egyszerű grafikus alapelemek

Az alfanumerikus jelek mellett az alapvető grafikus elemekhez tartozik a pont, a vonal, a kör, a háromszög, a négyzet, a téglalap, a sokszög stb. A kocka, gömb, henger stb., mint háromdimenziós térbeli elemek szintén rendelkezésre állnak. Minden grafikus elem arra szolgál, hogy a felhasználó a számára szükséges objektumot könnyen felépíthesse.

Példa: egy fémcsavart grafikusan egy hengerből, egy hatszögletű felső részből, egy spirálból és egy gömbből lehet összeállítani.

A **grafikus alapelemek alkalmazása** az egyes elemeknek a képernyőre hívásával történik. A felhasználó – az editor segítségével – meghatározza a grafikus elem nagyságát, térbeli helyzetét és torzítását. Ha egyetlen elem nem elég a kívánt objektum megalkotásához, akkor a felhasználó más grafikus alapelemeket is behívhat, ezeket megfelelő módon manipulálhatja, és elemről elemre összeállíthatja a kívánt képet.

Minden elem negatív is lehet, vagyis ha egy pozitív elemmel átfedés jön létre, akkor a negatív elem alakja a pozitív alakból kivonódik.

Példa: Ha egy negatív hengert oldalirányban, merőlegesen és részlegesen átfedve egy (pozitív) négyzetes hasábra helyezünk, akkor hornyolt hasáb keletkezik.

A felhasználó néhány alapelemből – **sejtnek** nevezett – újabb elemet állíthat össze, amelyet a továbbiakban egészként kezelhet.

Példa: ha egy gépcsavar sejtet képez, akkor méretváltoztatáskor a nagyítás vagy kicsinyítés mértékét az egész sejtre vonatkozóan kell csak megadni.

A **grafikus elemekre vonatkozó algoritmusok** pl. az analitikus geometriának a derékszögű koordináta-rendszerben érvényes összefüggésein alapulnak.

A felhasználó behívja a tárolóból a megfelelő képletet, és megadja a szabad paraméterek értékét. Minden alakzathoz rendszerint több egyenlet is tartozik, ezek közül a számítógép és a felhasználó az optimálisat választja ki.

A két- vagy háromdimenziós térben a **pontok** helyét x,y ill. x,y,z koordinátájuk határozza meg.

A **sokszögeket** csúcspontjukkal és megfelelő összekötő vonalakkal lehet megadni.

A **szakaszokat** a síkon a szakasz végpontjainak koordinátáival felírható

$y - y_1 = (y_2 - y_1)(x - x_1)/(x_2 - x_1)$

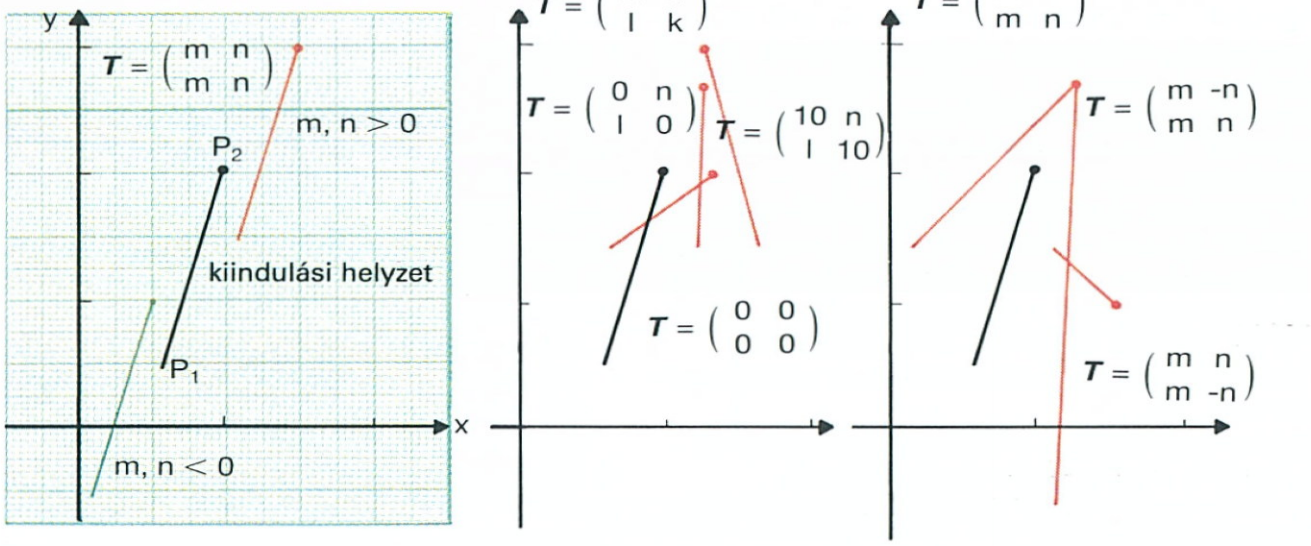
egyenlettel lehet megadni.

A **körök** az $r^2 = (x - a)^2 + (y - b)^2$ középponti egyenlettel írhatók le, ahol r a kör sugara; a,b: a kör középpontjának koordinátái.

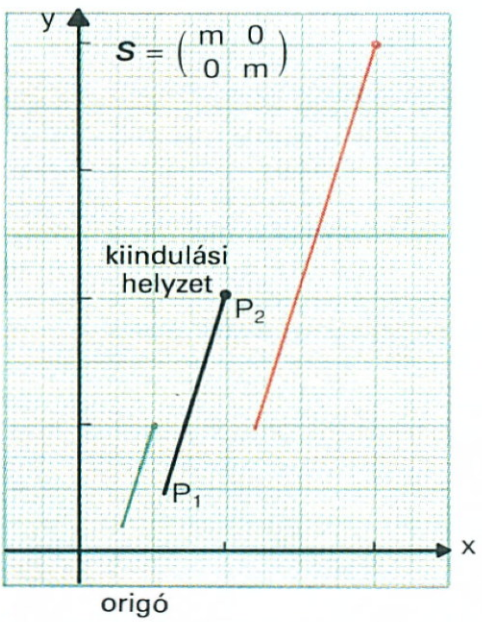
A **szabálytalan alakzatok** matematikailag az adatpontokra szakaszokat és felületeket illesztő Best Fit algoritmusok segítségével leírhatók.

A **grafikus elemek szerkesztése** eltolás, torzítás, forgatás, nagyítás, kicsinyítés, másolás, tükrözés és átfedés műveletek segítségével történik. Az elemeket ill. a sejteket egészként vagy részletekben (szegmensekként) lehet szerkeszteni. A megfelelő elemet fényceruzával ill. a kurzorral lehet bekeretezni vagy kijelölni. A grafikus programrendszer visszajelzését a kiválasztott szegmens kiemelése, a megfelelő vonalak bekeretezése vagy színének megváltozása jelenti.

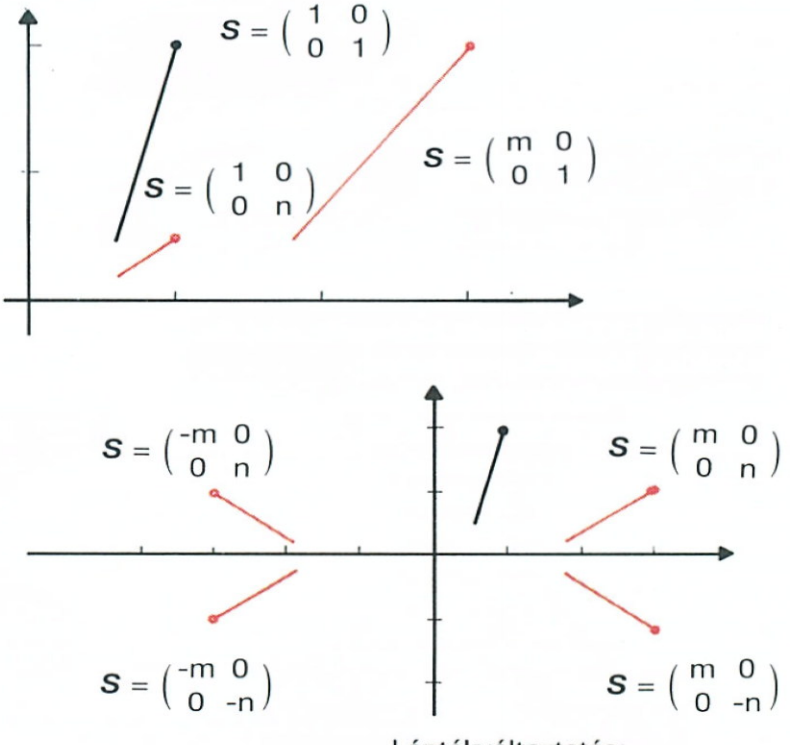
Eltolás:



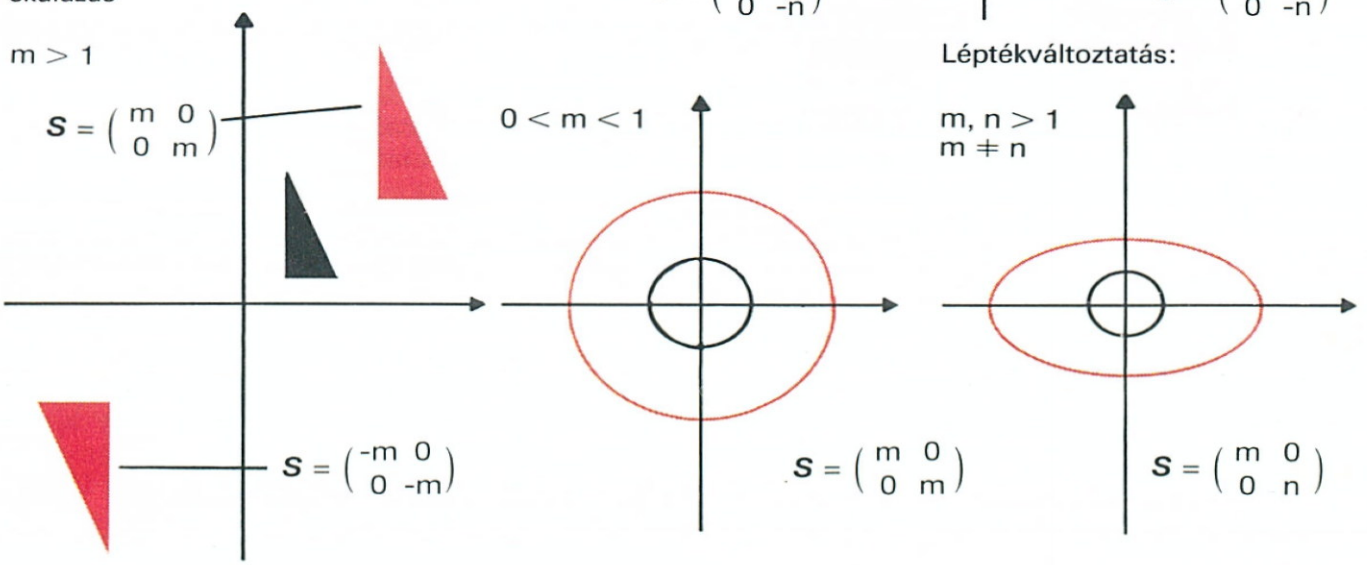
Léptékváltoztatás:



S-re vonatkozó egyéb lehetőségek



skálázás



Eltolás, kicsinyítés és nagyítás mátrixok segítségével

A grafikus elemek (legtöbbször derékszögű koordináta-rendszerben) *koordinátamátrixokként)* ábrázolhatók. A változtatások pontról pontra történnek az eltolást, a torzítást és a forgatást végző mátrixok segítségével. Ez az eljárás a *szerkesztés*.

A műveletek egyszerűen programozhatók, mert a mátrix matematikai struktúráját minden feladatorientált programnyelv *tömbként* tartalmazza. A mátrixok összeadását/vagy szorzását standard alprogramokkal lehet elvégezni.

A TURBO PASCAL nyelvben pl. az **A** és a **B** mátrixok közötti értékátadás utasítása: A := B;

Koordinátamátrix

Pont. Egy pont egyértelmű meghatározásához a síkban 1×2-es sormátrix, a térben pedig 1×3-as sormátrix szükséges:

(x y) ill. (x y z)

A **szakaszt** két végpontja egyértelműen meghatározza. A síkban a két végpont koordinátái 2×2 típusú mátrixot (x_1,y_1 és x_2,y_2), térben pedig 2×3 típusú mátrixot (x_1,y_1,z_1 és x_2,y_2,z_2) alkotnak:

$$\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} \quad \text{illetve} \quad \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{pmatrix}$$

Egy háromdimenziós **test** helyzetét a térben legalább három pontjának megadásával lehet rögzíteni. A megfelelő P_1, P_2 és P_3 pontok koordinátáit 3×3 típusú mátrix tartalmazza:

$$\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix}$$

A grafikus **szerkesztés** a koordinátamátrixok és a hozzájuk tartozó transzformációs mátrixok összekapcsolásával történik. A **mátrixszámítás** erre vonatkozó szabályai a következők:

Összeadás:

$$\begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix} + \begin{pmatrix} c_1 & c_2 \\ d_1 & d_2 \end{pmatrix} = \begin{pmatrix} a_1 + c_1 & a_2 + c_2 \\ b_1 + d_1 & b_2 + d_2 \end{pmatrix}$$

Szorzás

$$\begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix} \cdot \begin{pmatrix} c_1 & c_2 \\ d_1 & d_2 \end{pmatrix} = \begin{pmatrix} a_1 c_1 + a_2 d_1 & a_1 c_2 + a_2 d_2 \\ b_1 c_1 + b_2 d_1 & b_1 c_2 + b_2 d_2 \end{pmatrix}$$

Eltolás

Egy grafikus elem eltolása a koordinátamátrix és az eltolási mátrix összeadása útján történik.

Szakasz eltolása egy síkban. A szakasz két (P_1 és P_2) végpontjára vonatkozó $\boldsymbol{T}$ eltolási mátrix:

$$\boldsymbol{T} = \begin{pmatrix} m & n \\ m & n \end{pmatrix},$$

ahol

m, n: a P_1 és a P_2 pont x ill. az y irányú eltolása, ahol m ill. n külön-külön pozitív vagy negatív lehet.

Eltolás:

$$\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} + \boldsymbol{T} = \begin{pmatrix} x_1+m & y_1+n \\ x_2+m & y_2+n \end{pmatrix} = \begin{pmatrix} x_1' & y_1' \\ x_2' & y_2' \end{pmatrix}$$

Térbeli test. A három térbeli pontra vonatkozó $\boldsymbol{T}$ eltolási mátrix:

$$\boldsymbol{T} = \begin{pmatrix} m & n & l \\ m & n & l \\ m & n & l \end{pmatrix},$$

ahol

m, n, l: az x, y ill. z irányú eltolásnak felel meg.

Eltolás:

$$\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix} + \boldsymbol{T} =$$

$$\begin{pmatrix} x_1+m & y_1+n & z_1+l \\ x_2+m & y_2+n & z_2+l \\ x_3+m & y_3+n & z_3+l \end{pmatrix} = \begin{pmatrix} x_1' & y_1' & z_1' \\ x_2' & y_2' & z_2' \\ x_3' & y_3' & z_3' \end{pmatrix}$$

Léptékváltoztatás

A lépték megváltoztatása úgy történik, hogy a koordinátamátrixot megszorozzuk az **S** léptékmátrixszal. Ha (m, n, l) > 1, akkor a megfelelő grafikus elem nagyobb, ha pedig 0 < (m, n, l) < < 1, akkor kisebb lesz. A léptéktényező minden tengelyre különböző lehet. Ha a léptéket úgy választjuk meg, hogy m ≠ n, akkor egy körből pl. ellipszis keletkezik.

Skálázásról beszélünk, ha a különböző tengelyekre vonatkozó léptéktényezők megegyeznek, és pozitívak. A skálázott grafikus elemek egymáshoz hasonlóak, tehát megfelelő szögeik és irányítottságuk is megegyezik.

Ha valamennyi léptéktényező megegyezik és negatív, akkor az új grafikus elem a réginek a koordinátatengelyre vonatkozó tükörképét jelenti.

Az $\boldsymbol{S}$ mátrix megváltoztatja a grafikus elem léptékét, és egyidejűleg eltolását is eredményezi.

Síkbeli szakasz. Az $\boldsymbol{S}$ léptékmátrix főátlón kívüli elemei zérussal egyenlőek:

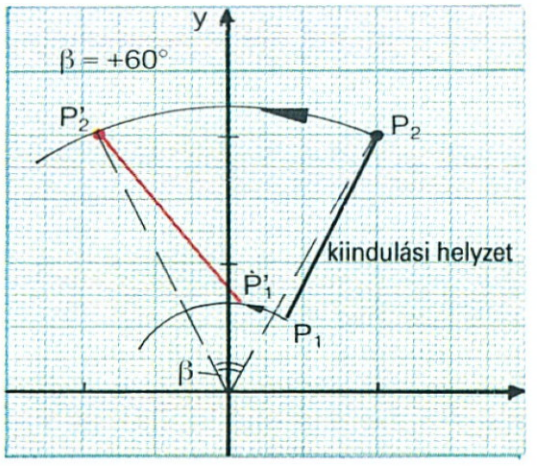
$$\boldsymbol{S} = \begin{pmatrix} m & 0 \\ 0 & n \end{pmatrix},$$

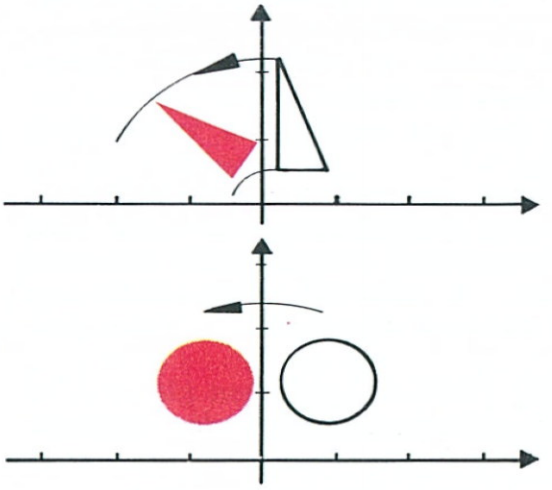
ahol

m, n: az x ill. y irányú léptékváltozás. Ha 0 < (m, n) < 1, akkor a grafikus elem kisebbé válik, és a koordináta-rendszer origója felé mozdul el. Ha (m, n) > 1, akkor a grafikus elem nagyobb lesz, és az origótól távolabbra kerül.

A lépték megváltozása:

$$\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} \cdot \begin{pmatrix} m & 0 \\ 0 & n \end{pmatrix} = \begin{pmatrix} mx_1 & ny_1 \\ mx_2 & ny_2 \end{pmatrix} = \begin{pmatrix} x_1' & y_1' \\ x_2' & y_2' \end{pmatrix}$$



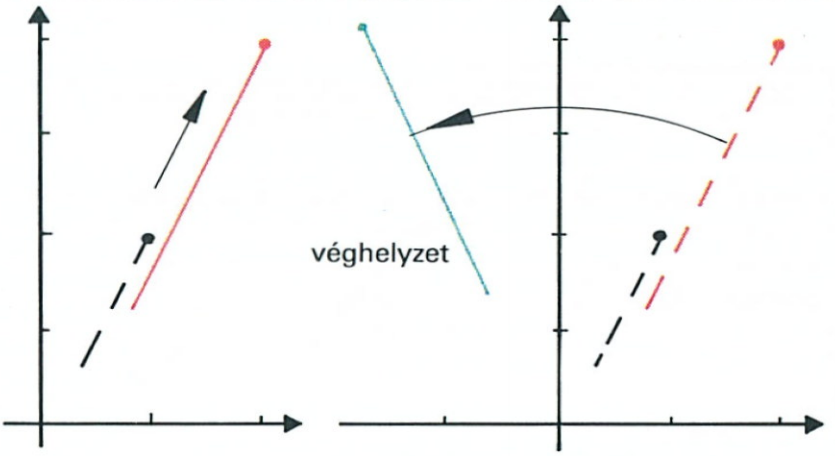


Három grafikus elem 60°-os forgatása egy síkban

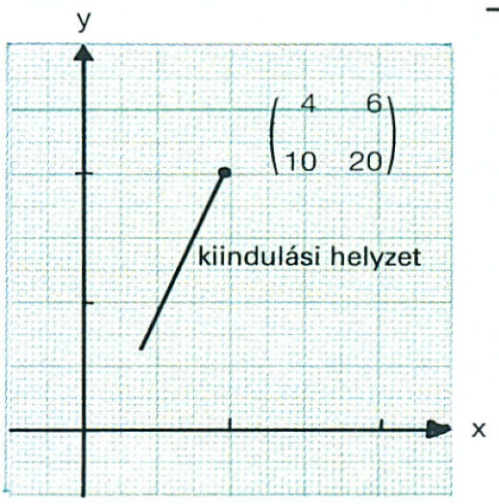
$$\Rightarrow \cdot \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 68 & 12 \\ 20 & 40 \end{pmatrix}$$

$$\cdot \begin{pmatrix} 0{,}643 & 0{,}766 \\ -0{,}766 & 0{,}643 \end{pmatrix} =$$

$$\begin{pmatrix} -4{,}05 & 13{,}8 \\ -17{,}8 & 41{,}0 \end{pmatrix}$$



nagyítás (2×) forgatás (+60°)

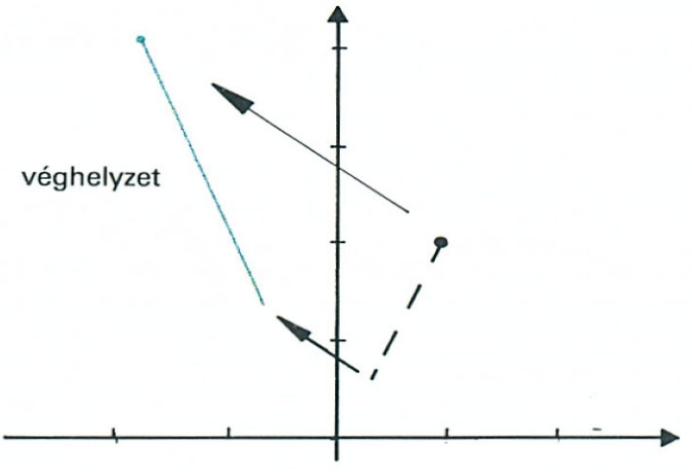


$$\Rightarrow \cdot \begin{pmatrix} 1{,}286 & 1{,}532 \\ -1{,}532 & 1{,}286 \end{pmatrix} = \begin{pmatrix} -4{,}05 & 13{,}8 \\ -17{,}8 & 41{,}0 \end{pmatrix}$$

konkatenáció:
nagyítás és forgatás

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 0{,}643 & 0{,}766 \\ -0{,}766 & 0{,}643 \end{pmatrix}$$

$$= \begin{pmatrix} 1{,}286 & 1{,}532 \\ -1{,}532 & 1{,}286 \end{pmatrix}$$



konkatenáció (2× és +60°)

Az összeláncolás lerövidíti a számítást

Térbeli test. Az **S** léptékmátrix a következő alakú:

$$\mathbf{S} = \begin{pmatrix} m & 0 & 0 \\ 0 & n & 0 \\ 0 & 0 & l \end{pmatrix},$$

ahol

m, n, l: a léptékváltozás az x, y ill. z irányban. Ha a főátlóban minden elem megegyezik, akkor *skálázásról* beszélünk.

Forgatás

Az $\boldsymbol{R}$ forgatási mátrix a grafikus elem minden pontját elforgatja a koordináta-rendszer középpontja körül. A forgatás β szöge valamennyi pontra azonos. Megállapodás szerint a pozitív – tehát +β szöggel való – forgatás az óramutató járásával ellentétes irányban, a negatív forgatás pedig az óramutató járásával megegyező irányban történik. Az $\boldsymbol{R}$ mátrix a grafikus elemet egységes egészként forgatja és tolja el.

Síkbeli szakasz. $\boldsymbol{R}$ Minden eleme egyszerű szögfüggvény:

$$\boldsymbol{R} = \begin{pmatrix} \cos\beta & \sin\beta \\ -\sin\beta & \cos\beta \end{pmatrix}$$

Az új koordináták a koordinátamátrix és a megfelelő forgatási mátrix szorzásával kaphatók meg.

$$\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} \cdot \boldsymbol{R} = \begin{pmatrix} x_1\cos\beta - y_1\sin\beta & x_1\sin\beta + y_1\cos\beta \\ x_2\cos\beta - y_2\sin\beta & x_2\sin\beta + y_2\cos\beta \end{pmatrix} =$$

$$= \begin{pmatrix} x_1' & y_1' \\ x_2 & y_2' \end{pmatrix}$$

Példa:

Forgassuk el +50°-kal a $P_1(x_1 = 6,\ y_1 = 4)$ és a $P_2(x_2 = 10,\ y_2 = 20)$ végpontok által meghatározott szakaszt a kétdimenziós koordinátarendszer középpontja körül. A forgatási mátrix a következő alakú:

$$\begin{pmatrix} \cos 50° & \sin 50° \\ -\sin 50° & \cos 50° \end{pmatrix} = \boldsymbol{R} = \begin{pmatrix} 0{,}643 & 0{,}766 \\ -0{,}766 & 0{,}643 \end{pmatrix}$$

A koordinátamátrix és a forgatási mátrix szorzása:

$$\begin{pmatrix} 6 & 4 \\ 10 & 20 \end{pmatrix} \cdot \boldsymbol{R} = \begin{pmatrix} 0{,}794 & 7{,}17 \\ -8{,}89 & 20{,}5 \end{pmatrix} = \begin{pmatrix} x_1' & y_1' \\ x_2' & y_2' \end{pmatrix}$$

Az elforgatott szakasz végpontjainak (P_1' és P_2') koordinátái tehát

$x_1' = 0{,}794$, $y_1' = 7{,}17$ és $x_2' = -8{,}89$, $y_2' = 20{,}5$.

A szakaszt az óramutató járásának irányában forgattuk el, hosszúsága nem változott.

Térbeli test. A forgatást térbeli komponensekre bontjuk. A forgatási mátrix megfelelő komponensei $\boldsymbol{R}_x$, $\boldsymbol{R}_y$ és $\boldsymbol{R}_z$ a következő alakúak:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{pmatrix}, \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}$$

és

$$\begin{pmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Összeláncolás, konkatenáció

A grafikus elemeket általában nem lehet egyetlen egyszerű traszformációval a kívánt helyre mozgatni és ugyanakkor a léptéket is megváltoztatni.

Példák:

Nagyítsunk fel egy grafikus elemet úgy, hogy a belsejében egy pont (pl. a súlypont) helyzete ne változzon. Megoldás: először megváltoztatjuk a léptéket, azután az adott pontot (az egész grafikus elemmel együtt) régi helyzetébe toljuk vissza.

Forgassunk el adott pont körül egy grafikus elemet. Megoldás: a forgatási mátrix alkalmazása után eltoljuk a rendszert a megadott pontba.

Egyszerűbb megoldást jelent, ha az egyes transzformációkat összekapcsoljuk egymással (konkatenáció), majd a grafikus elemet a láncolt mátrix segítségével transzformáljuk.

Amíg csak léptékváltoztatásra és forgatásra van szükség, valamint csak a főátló elemei különböznek 0-tól, a transzformációk összeláncolásának sorrendje nem játszik szerepet.

Nagyítás és forgatás

Nagyítsunk adott szakaszt m-szeresére, valamint forgassuk el +β szöggel. A transzformáció egyes lépései a következők:

Először a nagyítást végezzük el:

$$\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} \cdot \begin{pmatrix} m & 0 \\ 0 & m \end{pmatrix} = \begin{pmatrix} mx_1 & my_1 \\ mx_2 & my_2 \end{pmatrix}$$

Azután következik a forgatás:

$$\begin{pmatrix} mx_1 & my_1 \\ mx_2 & my_2 \end{pmatrix} \cdot \begin{pmatrix} \cos\beta & \sin\beta \\ -\sin\beta & \cos\beta \end{pmatrix} =$$

$$= \begin{pmatrix} mx_1\cos\beta - my_1\sin\beta & mx_1\sin\beta + my_1\cos\beta \\ mx_2\cos\beta - my_2\sin\beta & mx_2\sin\beta + my_2\cos\beta \end{pmatrix} =$$

$$= \begin{pmatrix} x_1' & y_1' \\ x_2' & y_2' \end{pmatrix}$$

A transzformációkat fordított sorrendben elvégezve ugyanerre az eredményre jutunk.

Az összeláncolás felhasználásával elvégzett transzformáció a következő módon történik:

Először összekapcsoljuk a két transzformációs mátrixot:

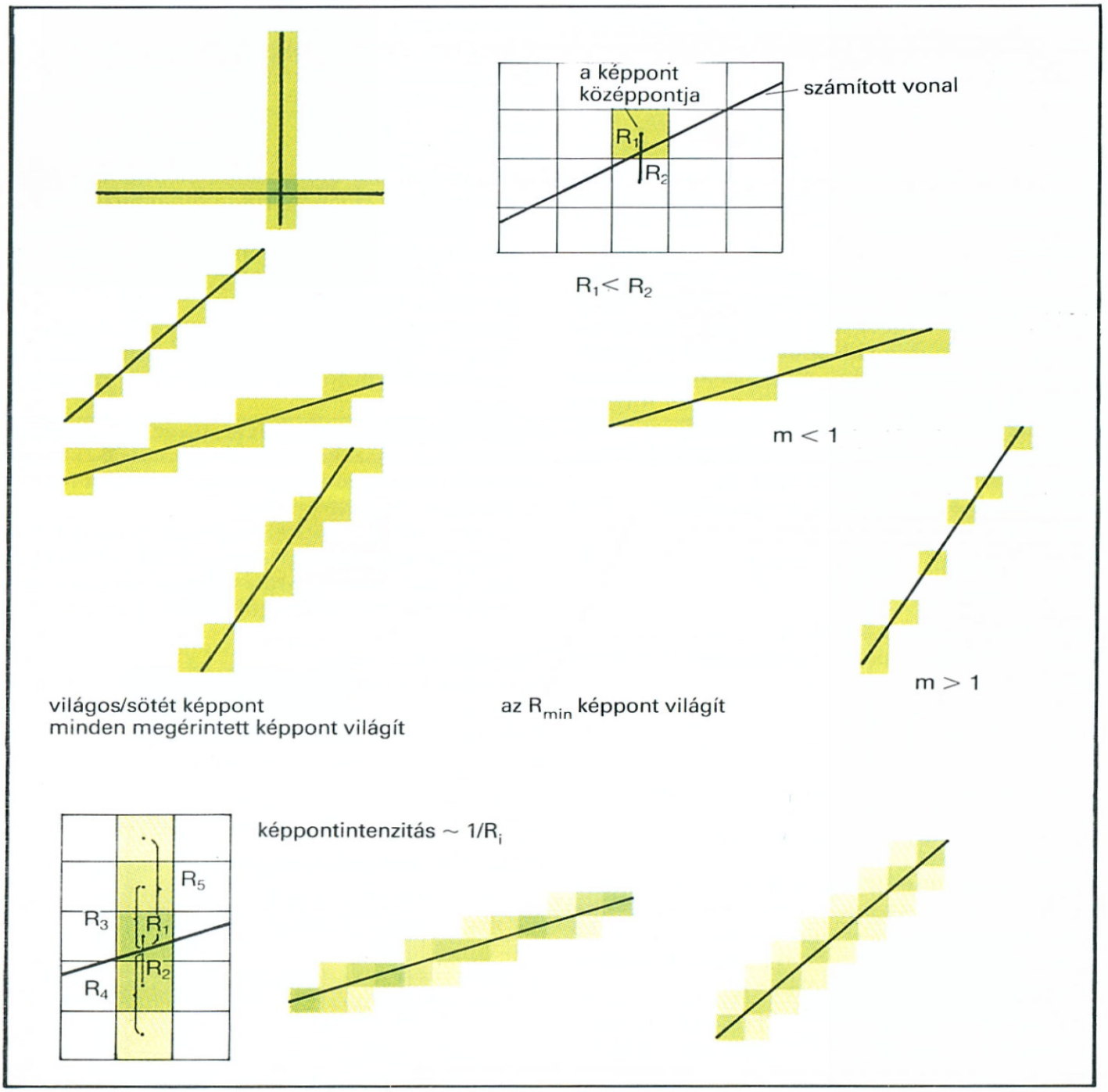
$$\begin{pmatrix} m & 0 \\ 0 & m \end{pmatrix} \cdot \begin{pmatrix} \cos\beta & \sin\beta \\ -\sin\beta & \cos\beta \end{pmatrix} = \begin{pmatrix} m\cos\beta & m\sin\beta \\ -m\sin\beta & m\cos\beta \end{pmatrix}$$

Ezt összeszorozzuk a koordinátamátrixszal:

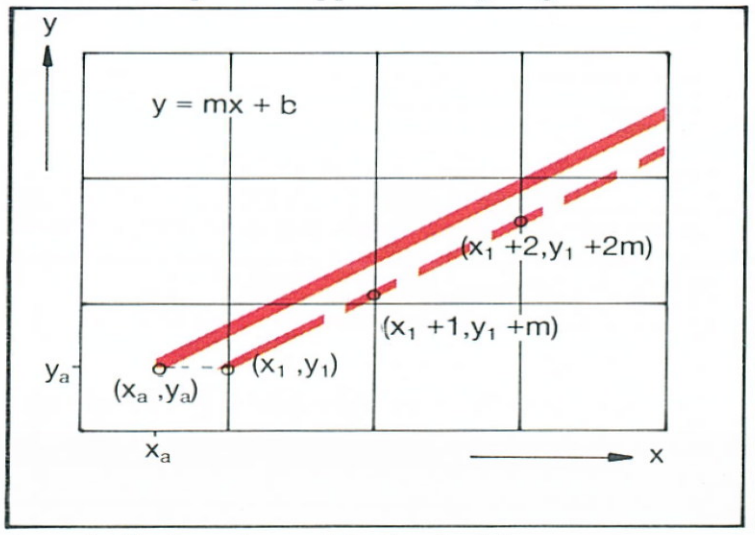
$$\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} \cdot \begin{pmatrix} m\cos\beta & m\sin\beta \\ -m\sin\beta & m\cos\beta \end{pmatrix} = \begin{pmatrix} x_1' & y_1' \\ x_2' & y_2' \end{pmatrix}$$

Ugyanahhoz az eredményhez jutottunk, mint az előző esetben.

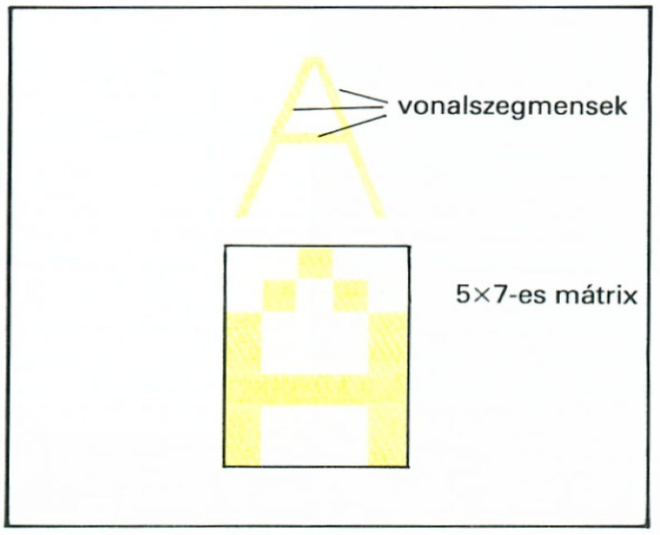
A transzformációk összeláncolásának előnye főleg akkor érezhető, ha sok transzformációt kell elvégezni.



Vonalak felépítése képpontok segítségével



Az y értékek oszloponkénti számítása a normál alak felhasználásával



A karakterek szerkezete a képernyőn

A bitleképzéses terminálon a raszterképet véges számú **képpont** vagy **képelem** (angolul: ***P**icture **e**lement*, **pixel**) alkotja. A képpont a legkisebb címezhető képegység. Címe megegyezik a képpontnak a képernyőn elfoglalt x, y pozíciójával. Minden ábrázolt vonal különálló képpontokból tevődik össze.

Elég, ha csak egyenes szakaszok ábrázolását tárgyaljuk, mert a képernyőn minden görbe egyenes szakaszokból álló szegmensekből épül fel.

Az emberi szemnek a pontokból álló vonal mindaddig folytonosnak tűnik, amíg a pontok egymástól mért távolsága a szem optikai felbontóképességénél kisebb.

Optimális körülmények és kb. 50 cm látástávolság esetén az emberi szem két, egymástól 1/100 mm távolságban fekvő, egymástól elkülönülő pontot még éppen meg tud különböztetni.

A számítógépprogram megadja egy vonal kezdő- és végpontját. A grafikus ábrázolás feladata az, hogy ezt a két pontot képpontok sorozatával kösse össze. Tehát egy olyan algoritmusra van szükség, amely a közbenső képpontokat meghatározza. A vonal képpontkoordinátái és a hozzájuk tartozó intenzitásértékek egy rövid idejű tárolóba – a videopufferbe (angolul: *video buffer*) – kerülnek, az adatkiviteli készülék processzora pedig átveszi és megjeleníti ezeket.

Az **egyenes** Descartes-féle **normálalakja** a következő:

$y = mx + b$

m: az egyenes meredeksége.

b: az egyenes és az *y* tengely metszéspontjának ordinátája.

Az (x_a, y_a) és (x_b, y_b) végpontokkal rendelkező szakasz valamennyi (x, y) pontja kielégíti a **két ponton átmenő egyenes egyenletét:**

$(y - y_a)/(x - x_a) = (y_b - y_a)/(x_b - x_a)$

A megfelelő algoritmus hardveres vagy szoftveres formában állhat rendelkezésre.

Vonalstruktúra

A számítógépes grafikában – az egyenes egyenlete alapján felírt – algoritmus határozza meg azt, hogy az egyenes szakasz két végpontja között melyik képpontoknak kell világítaniuk.

Ha csak világos/sötét intenzitásértékek fordulhatnak elő, akkor két lehetőség közül lehet választani:

1. Minden olyan képpont, amelyet a kiszámított vonal érint, világít. Ennek a módszernek az a hátránya, hogy az aktivált képpontok sűrűsége az ábrázolni kívánt vonal meredekségétől függ.
2. Az algoritmus oszlopról oszlopra kiszámítja, hogy melyik képpontnak a középpontja fekszik a kérdéses vonalszegmenshez legközelebb, és csak ezt az egy képpontot aktiválja.

   Ez a módszer csak $|m| \leqslant 1$ esetén szolgáltat egymással érintkező, világító képpontokat. Ha $|m| > 1$, akkor a vonalban szakaszok lépnek fel.

   Ezt a nehézséget néha az x, y koordináták felcserélésével lehet áthidalni.

Az első módszer vizuális szempontból nem kielégítő, a második módszer sok számítógépidőt igényel.

Az algoritmus lényegesen egyszerűsödik – és gyorsabbá is válik –, ha a szakasz kezdőpontjából kiindulva a normálalak segítségével számítja ki a következő képpontot. Az x_a, y_a pontból kiindulva kiszámítjuk a következő képpont x_1, y_1 egész számú koordinátáit. Ezután az x-érték mindig egy oszloppal lép tovább, tehát értéke egész marad. A következő képpontok koordinátái $x_1 + 1$, $y_1 + m$; $x_2 + 2$, $y_2 + 2\,m$ stb.

A *Bresenham-algoritmus* a szakasz kezdő- és végpontjainak koordinátáit egész számra kerekíti, és ezekből az értékekből indul ki. A további számításokhoz így elegendő az egész számú összeadás és kivonás algoritmusa, ami a számítási időt lényegesen lerövidíti.

Hátránya: bizonyos körülmények között két egymást metsző, közel azonos meredekségű egyenes a képernyőn egybeeshet.

A vonalstruktúra és a szürkeségi skála. A vonalak finomszerkezete a világos-sötét raszteren fokozatokat mutat. Ezt a struktúrát az emberi szem már nem érzékeli, ha az érintett képpontok intenzitáseloszlására finomabb fokozatokat alkalmazunk: az algoritmus ismét az *R* távolságot határozza meg, *R* = (a képelem középpontja és a vonalszegmens között mérhető távolság). Ekkor azonban *mindkét* szomszédos oszlopban álló képpont aktiválódik. Az egyes képpontok intenzitása – a legegyszerűbb esetben – az *R* távolsággal fordítottan arányos: a vonalszegmenshez közelebbi képpont erősebben világít. Bonyolultabb intenzitáseloszlás-függvények alkalmazása a képernyőn megjelenő vonalak lépcsőzetességét valamennyire kiegyenlíti.

Ezek az algoritmusok viszonylag gyorsak, mert az eloszlásfüggvény értékeit táblázatból veszik át.

Karakterek és szimbólumok

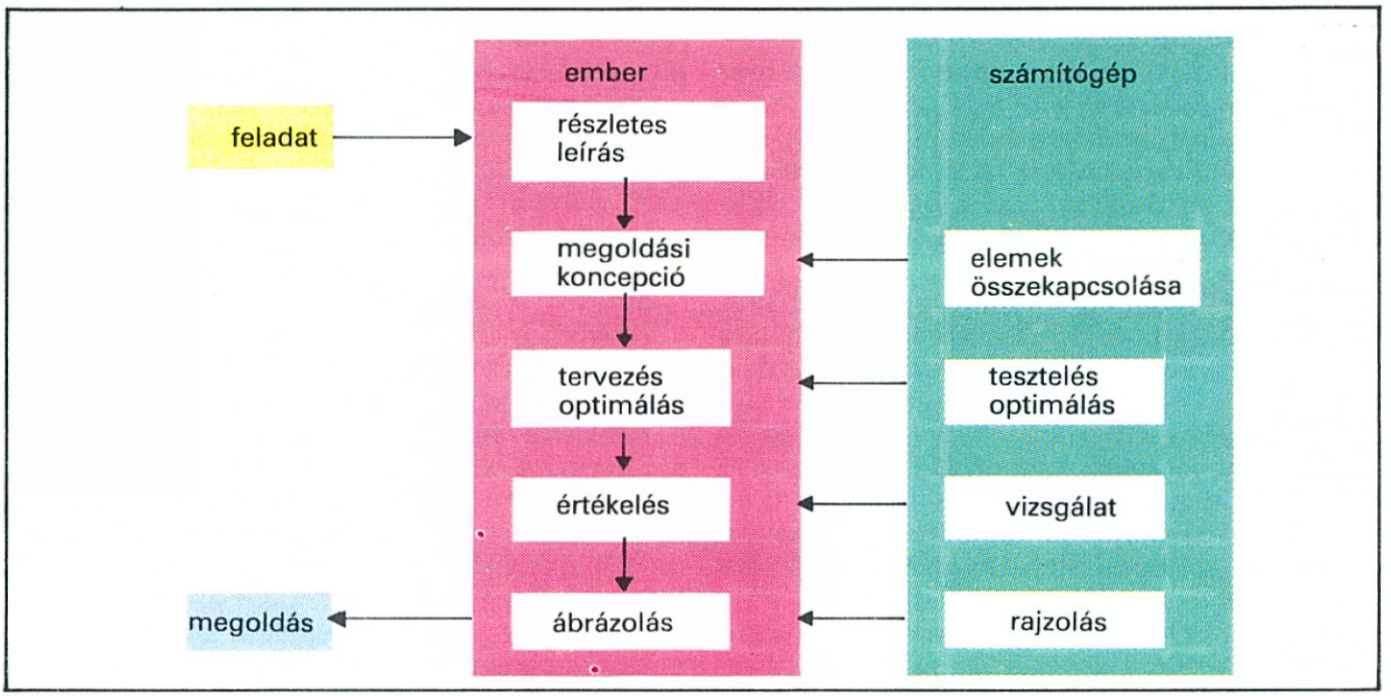
Minden grafikus ábrázolás vonalak mellett más jeleket (feliratokat, méreteket stb.) is tartalmaz. Ezek előre beprogramozott standard jelek lehetnek – mint pl. az alfanumerikus terminálok esetén –, vagy pedig szabadon programozhatóak.

A **vonalas karakterek (vektorjelek)** különálló vonalszegmensekből tevődnek össze, amelyek léptékét a szakaszok hosszának azonos tényezővel való szorzásával egyszerűen lehet változtatni.

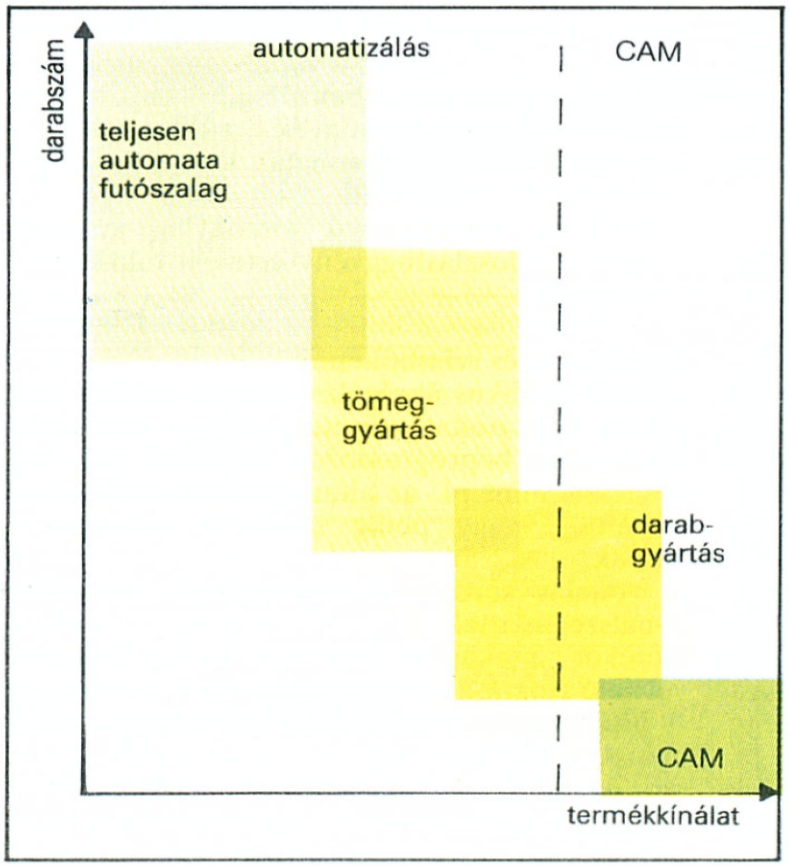
A **bitleképezéses karakterek** adott rasztertartományban különálló képpontokból tevődnek össze. A mátrixméretek 7×9 képponttól a nagyfelbontású grafika esetén 100-nál is több képpontot tartalmazó jelméretekig terjedhetnek. Mivel itt raszteres ábrázolásról van szó, a méretváltoztatások időigényesek. Szürkeárnyalatok bevezetése ebben az esetben is javít a megjelenített kép minőségén.

rövidítés	angol szakkifejezés	magyar szakkifejezés
CAD	Computer-Aided Design	számítógéppel támogatott tervezés
CAE	Computer-Aided Engineering	számítógéppel támogatott mérnöki munka
CAL	Computer–Aided Learning	számítógéppel támogatott tanulás
CAM	Computer-Aided Manufacturing	számítógéppel támogatott gyártás
CAP	Computer-Aided Production Planning	számítógéppel támogatott gyártástervezés
CAQ	Computer-Aided Quality Control	számítógéppel támogatott minőségellen.
CIM	Computer-Integrated Manufacturing	számítógépes termelés

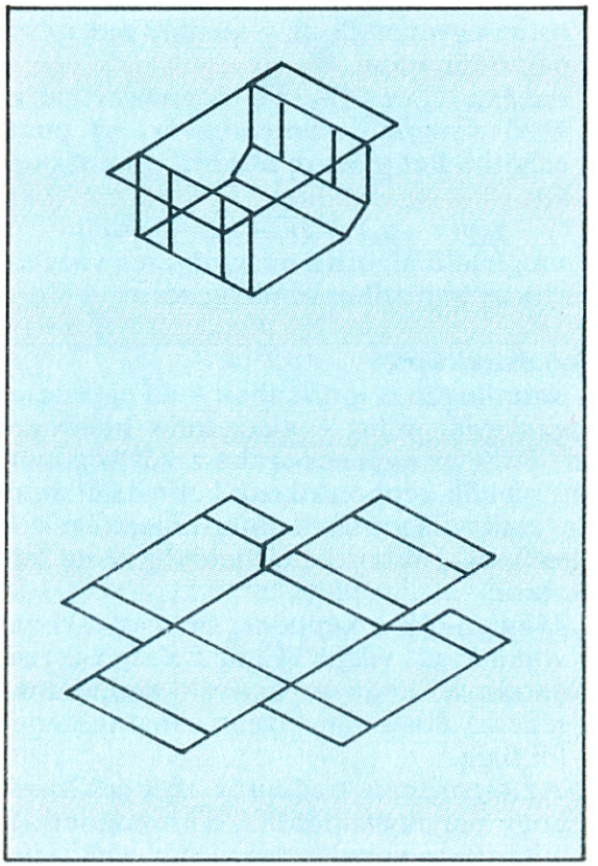
Jelölések



Számítógéppel támogatott tervezés (CAD)



Az automatizálás és a CAM optimális tartományai



Doboz kiterítése síkban a CAD programmal

A **számítógéppel támogatott** vagy **számítógépes tervezés** (angolul: *Computer-Aided Design*, **CAD**) a tervező és a megfelelően programozott számítógép együttműködésén alapul.

A CAD fejlődése a mérnöki irodákban kezdődött az 1960-as évek elején. Időközben a CAD egyéb területeken, pl. a divattervezésben, a formatervezésben, az építészetben és a művészetben is elterjedt.

Az ember és a számítógép együttműködésében az ember az alkotó résztvevő, aki a feladat vázlatos tervét nagy vonalakban felvázolja. A számítógép végzi a részletek kidolgozását, a munka rutin jellegű részét. Ennek során döntő jelentőségű a nagy számítási sebesség, a gyors és sokoldalú ábrázolási lehetőség, valamint a nagy adatmennyiségek tárolásának képessége és az adatokhoz történő gyors hozzáférési lehetőség. Az ember és a számítógép szinergetikus rendszert alkot, amely növeli a kreativitást és a termelékenységet.

Az együttműködés nem szűnik meg a tervrajz megszületésével: további lehetőséget jelent, hogy a számítógép a rajzot numerikus adatokká alakítja át, ezeket az ipari gyártás számára feldolgozza, és ezt követően a termelést egészen a termék elkészültéig vezérli és ellenőrzi. Ezeket a folyamatokat a **számítógéppel támogatott mérnöki tevékenység** (angolul: *Computer Aided Engineering*, **CAE**) és **számítógépes termelés** (angolul: *Computer Integrated Manufacturing*, **CIM**) néven foglaljuk össze.

A CAD (és az egyéb számítógéppel támogatott tevékenységek) előnyei a következők: viszonylag rövid idő telik el az elgondolás, a tervrajz és a termék elkészülése között, a kidolgozás minden szakaszában gyorsabb, mint a hagyományos eljárások, a termék minősége jobb, és az előállítás költségei csökkennek.

Példa: CAD a mechanikai alkatrészek tervezése során.

A **koncepció** kidolgozása során az ahhoz szükséges elemek (pontok, vonalak, görbék, testek) a számítógép operatív tárolójában matematikai eljárások formájában állnak rendelkezésre. Az elemek a képernyőre hívhatók, törölhetők, elmozgathatók, nagyíthatók, kicsinyíthetők, elforgathatók és bonyolult objektumokká kapcsolhatók össze. Ezeknek a műveleteknek a felhasználásával a tervező a vázlatból elkészíti az első műszaki tervet. A fekete-fehér kétdimenziós körvonalazástól a különböző vetületekben és perspektívákban megjeleníthető, színes, háromdimenziós, részletes rajzokig minden lehetőség a rendelkezésére áll.

Tesztelés. A számítógép különleges programok segítségével megvizsgálja a tervezett objektumnak pl. a feszültség és nyújtás hatására bekövetkező deformációját, hővezetési folyamatait, valamint az anyag tulajdonságait. Az eredmények alapján a tervező módosítja a tervet, és ismét lefuttatja a tesztet.

Vizsgálat és értékelés. További programok vizsgálják meg, hogy a terv megfelel-e a szabványok előírásainak és az adott tűrési határon belül a gyártási lehetőségeknek, valamint ellenőrzik az egymáshoz kapcsolódó részek illeszkedését stb. A lényeges részek kinagyításával és a látószög megváltoztatásával a terv külső megjelenésének vizsgálata is lehetséges. Az építészeti tervek készítése során a dinamikus ábrázolás lehetővé teszi, hogy pl. a tervezett épület belsejébe jussunk, és a tervezett megvilágítást értékeljük.

Rajzolás. Ha a terv az ellenőrzés során optimálisnak bizonyul, akkor a számítógép – ugyancsak a CAD szoftverjének ellenőrzésével – elkészíti a kívánt léptékű végleges tervrajzot. Egyéb speciális programok különleges rajzoknak, pl. a különböző szintek metszeteinek elkészítését is lehetővé teszik, vagy pl. azt, hogy a háromdimenziós objektumokat síkba kiterítsük, hogy azokat lemezből el lehessen készíteni. Ha építészeti tervről van szó, a számítógép a tervezett objektumot különböző látószögekből szemlélve egy ugyancsak megtervezett tájba tudja helyezni, és azt be tudja mutatni.

A CAD ipari alkalmazása. Fémfeldolgozás. Házak, gyárak, gépek, közlekedési eszközök, hajók, repülőgépek, rakéták építése. Mikroelektronikai chipek készítése a tervezéstől és a kapcsolásoktól kezdve egészen a gyártáshoz szükséges maszkok elkészítéséig. A villamos energia és a különböző folyadékok szállítására szolgáló hálózatok tervezése.

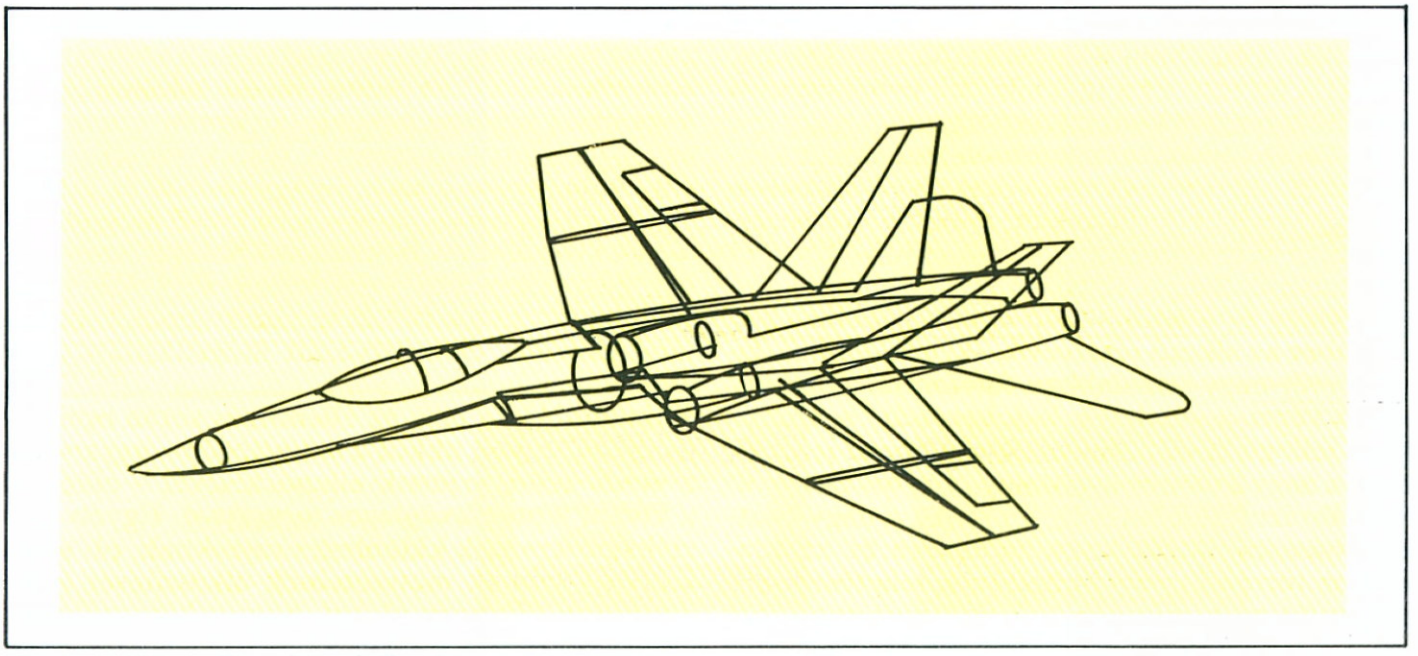
A CAD megvalósításához hardver és szoftver szükséges.

A **CAD-hardver** – a feladattól függően – különböző méretű számítógépeket foglal magában: a különleges célokra kifejlesztett mikroprocesszortól a számos terminálhoz hozzáférési lehetőséggel rendelkező nagy számítógépig. A számítógépeket a CAD céljára a szokásos be-/kiviteli eszközökön kívül speciális készülékekkel lehet kiegészíteni, illetve bővíteni: pl. nagyfelbontású grafikus képernyővel, különleges billentyűzettel, elektronikus vezérlésű rajzolókészülékkel, digitalizáló berendezéssel, másológéppel stb.

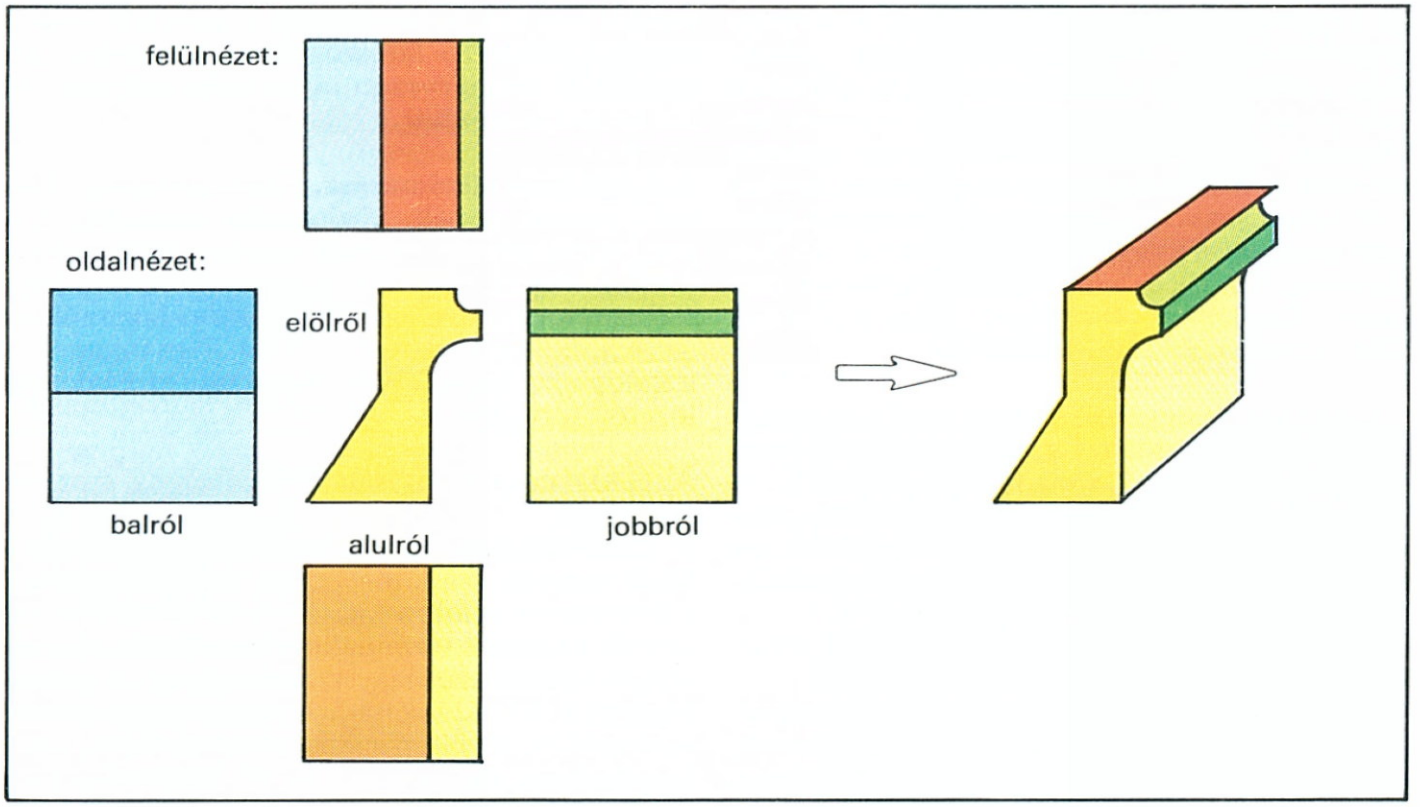
A **CIM hardverjét** a termelési környezethez igazítják, tehát ütésálló, hőtől és nedvességtől védett kivitelben készítik. A be-/kiviteli eszközöket részben közvetlenül összekötik a szerszámgépekkel.

Időközben nagyon változatos **CAD-szoftver** áll rendelkezésre. Kifejezetten a CAD számára kifejlesztett programnyelvek és kiterjedt adatbankok léteznek. A CAD-programok hozzáférési ideje általában rövid, és terjedelmes RAM tárolót igényelnek.

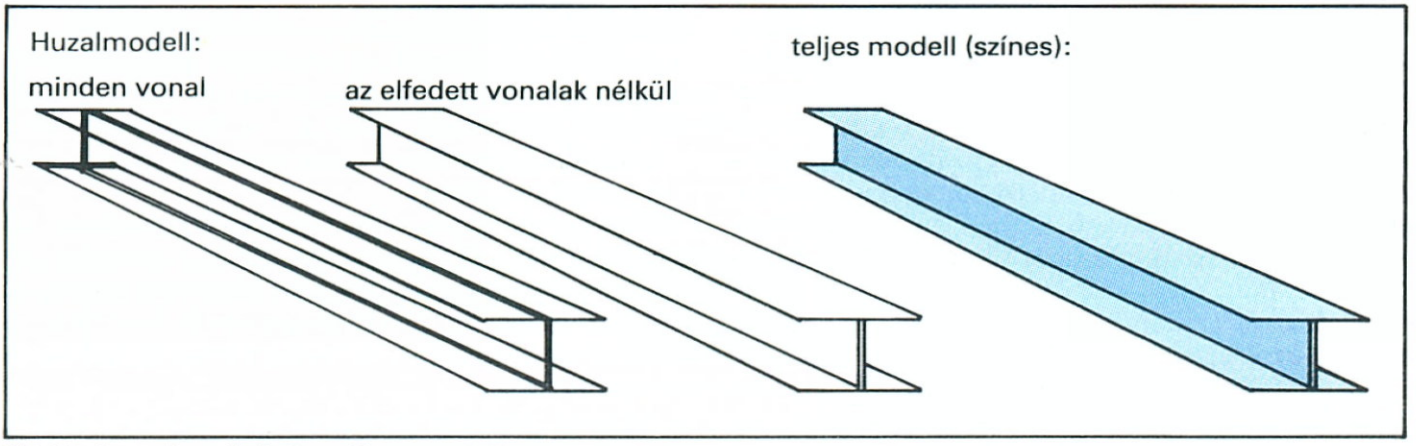
A nagyon gyors processzorok és az olcsó RAM tárolók kifejlesztésével az 1980-as években vált gazdaságossá a számítógépek használata, és ekkor vált lehetővé a CAD elterjedése is.



Egy repülőgép huzalmodellje



Teljes modell összeállítása a körvonalrajzokból



CAD: huzalmodell és teljes modell

Háromdimenziós CAD

A térbeli objektumok képernyőn történő háromdimenziós ábrázolásának, átalakításának és a CAD-adatbankban történő tárolásának (lásd alább) különböző módszerei léteznek.

A kétdimenziós CAD-programokhoz hasonlóan a térbeli objektumok tervezéséhez is számos segédprogram áll rendelkezésre.

Példák: felületek vonalkázása, szöveg beírása, méretek és tűréshatárok feltüntetése, a szabványokkal és a műszaki előírásokkal való egyezés ellenőrzése, kapcsolódás a feladatorientált programnyelvekhez stb. További programok teszik lehetővé a különböző perspektívájú ábrázolást és a vetületek megjelenítését.

Huzalmodell. A huzalmodell (angolul: *wire-frame model*) olyan háromdimenziós grafika, amelyen a felületek nincsenek kitöltve. Úgy néz ki, mintha a tárgy merev huzalokból lenne felépítve. A szemléletesség fokozása érdekében segédvonalakat, pl. a kontúrokat is fel lehet rajta tüntetni. Bár egy test huzalmodelljét nem lehet olyan plasztikusan és egyértelműen interpretálni, mint egy kitöltött felületekkel és árnyékolással rendelkező valósághű képet, mégis számos előnnyel rendelkezik: gyorsan lehet a vonalakat törölni, megváltoztatni vagy új vonalakat húzni. Ezeknek a műveleteknek a számítási ideje csekély, és az egész modell által igénybe vett tárolóterület is kicsi. Hátrányai: mivel a tárgy minden élét látni lehet, ezért az ábra gyakran nem egyértelmű. Segítséget jelent, ha utólag azokat a vonalakat, amelyek az adott látószögből nem láthatók, a felhasználó manuálisan vagy a program maga automatikusan kitörli.

Teljes modell. A választott perspektívában látható felületek kitöltöttek, így az ábrázolás a szemlélő számára egyértelmű. Színek alkalmazása tovább erősíti a háromdimenziós benyomást. Hátránya: a megfelelő programok lassúak és nagyon sok tárolóhelyet igényelnek.

A huzalmodell és a teljes modell közötti átmenetek optimális átmeneti megoldást szolgáltatnak.

A gyors processzorral és olcsó RAM-tárolóval készült modern számítógépek részben ellensúlyozzák ezeket a hátrányokat; a háromdimenziós tárgyak teljes modellje időközben nagymértékben kiszorította a huzalmodellt.

A teljes modell felépítése két, egymástól nagymértékben különböző módon történhet: háromdimenziós alapelemekből (térfogati modell), vagy kétdimenziós vonalas rajzokból (felületi modell).

A **grafikus alapelemekből történő felépítés** nagymértékben megegyezik a kétdimenziós tervek számára kidolgozott építőszekrényelvvel. A szerkesztő a képernyőn térbeli alapelemekkel (pl. kockával, hasábbal, hengerrel, gömbbel, gúlával stb.) dolgozik, és ezekből állítja össze a kívánt tárgyat. A szerkesztéshez szükséges számítási igény meglehetősen nagy.

A program a grafikus alapelemek kombinálását a Boole-algebra segítségével végzi.

Az *ÉS* Boole-operátor pl. két érintkező grafikus elemet egy új elemmé egyesít. Ha az A átfedi a B grafikus elemet, akkor a B *ÉS* -A művelet érvényes, így az A test alakját a B alakjából kivágással el lehet távolítani.

Ez az építőszekrényelv egyszerű, szemléletes és viszonylag kevés tárolóhelyet vesz igénybe. Alkalmazásai területei: építészet, gépek tervezése, vezetékezési tervek, elektronikus alkatrészek tervezése stb. Hátránya: a modellnek alapelemekből kell összeállíthatónak lennie.

A **kétdimenziós rajzokból történő felépítés** különböző nézetben ábrázolt, léptékhelyes, vonalas rajzok segítségével történik. A mintarajzok digitalizálók, elektronikus rajzasztalok vagy optikai letapogatók (szkennerek) segítségével jutnak a számítógép adattárolójába. Ezekből a megfelelő programok háromdimenziós képet állítanak össze és megjelenítik a képernyőn, ezután a képeken változtatásokat lehet végrehajtani. Ez a módszer akkor jár előnyökkel, ha a modellt grafikus alapelemekből nem lehet összeállítani, pl. a szárnyfelületek vagy az autókarosszériák esetén. Az ilyen módszerrel működő programok nagy tárolóhelyet, de csak kevés számítási időt vesznek igénybe.

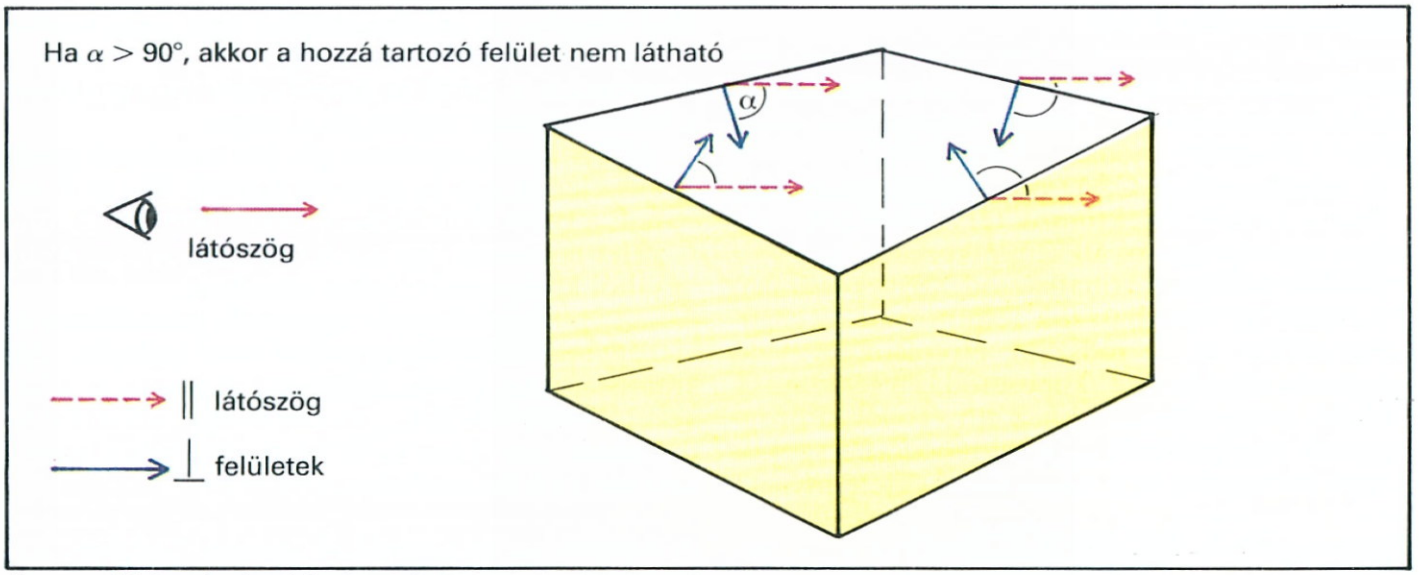
CAD-adatbankok

Minden CAD-rendszerhez speciális adatbankrendszer tartozik, amelyek nem csak a terveket, a rajzokat, a perspektivikus ábrázolásokat és az alfanumerikus információt tartalmazzák, hanem a tervezett szerkezetek analízisét végző, alkatrészlistát készítő és költségmegállapító programokat is.

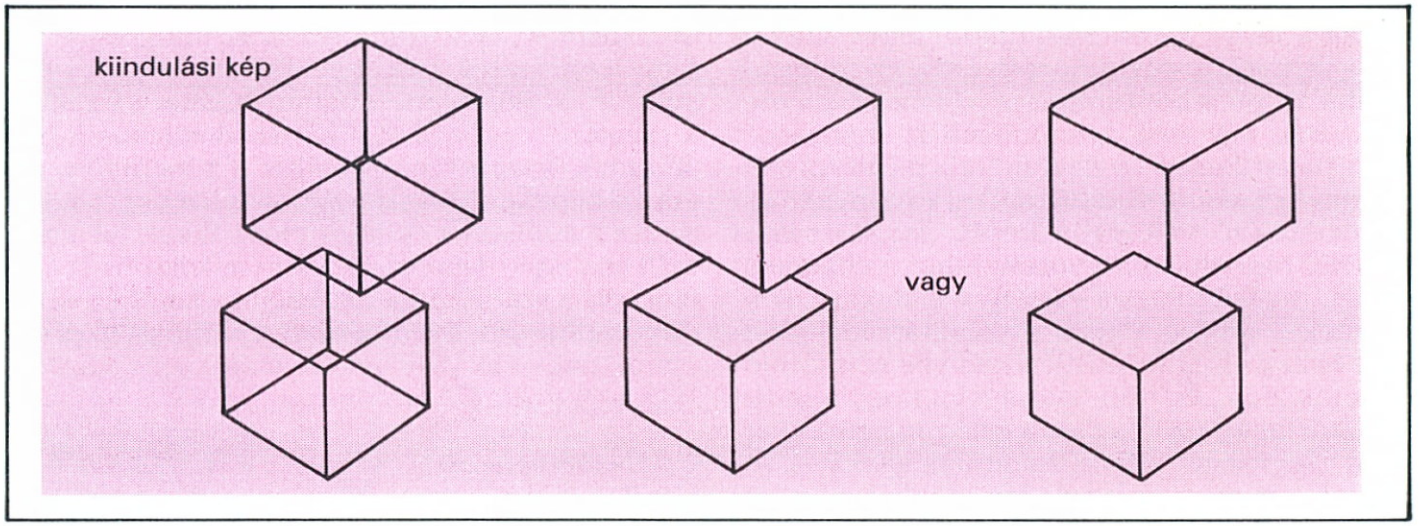
Ahhoz, hogy egy tervet tárolni lehessen és később hozzá lehesen férni, kétféle lehetőség kínálkozik:

Valamennyi adat tárolásra kerül, azaz pl. egy vonal minden eleme, és egy felület minden pontja. Ez ugyan nagyon sok tárolóhelyet foglal el, de a terv készítője gyorsan a képernyőre viheti át az adatokat.

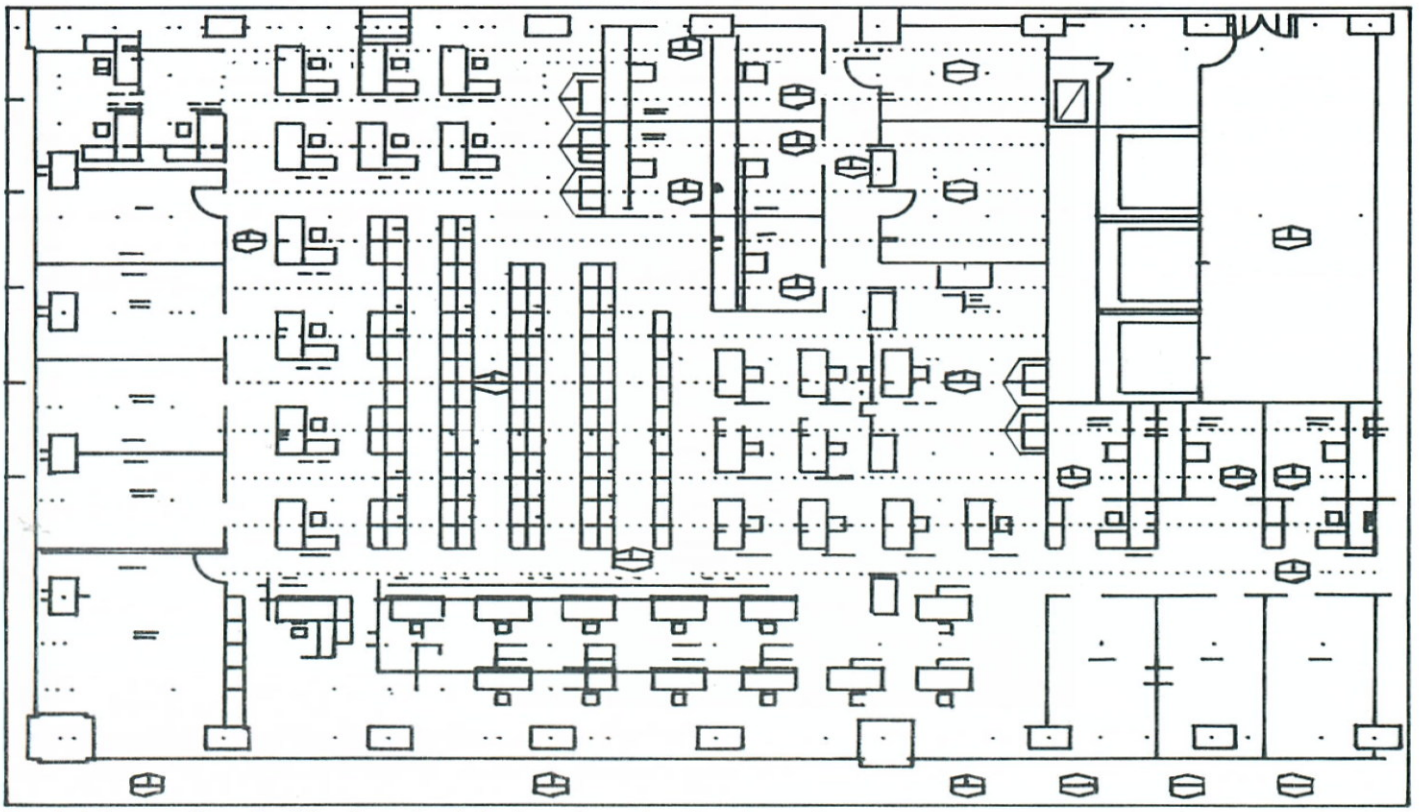
A másik módszer szerint **csak a** lényeges pontok **koordinátái** kerülnek a tárolóba. Híváskor a számítógép a tárolt koordináták alapján rekonstruálja a tervet. Bár a tárolóigény ebben az esetben csekély, a terv megjelenítése hosszadalmas. Kompromisszumos megoldást jelent, ha a terv – Boole-operációkkal összekapcsolt – alapelemek kombinációjaként kerül tárolásra.



A nem látható rész kiszámítására szolgáló algoritmus



Az eltakart felületek eltávolítása számítógéppel



Irodaház egyik szintjének számítógéppel optimált alaprajza

A számítógéppel támogatott tervezés (CAD) alkalmazása különösen az építészetben mutatkozott előnyösnek, és ezen a területen gyakorlatilag el is terjedt. Az 1970-es években elsősorban rajzolási segédeszközként, valamint a szabványos építési elemek, pl. az ablakok, ajtók, víz- és áramcsatlakozások elhelyezésére alkalmazták. Az 1990-es évek óta használatos különleges programok már az építészeti tervek kreatív továbbfejlesztésére irányulnak: pl. elkészítik a tervezett objektum valósághű képét a leendő környezetében, ábrázolják az objektumot a nap folyamán változó megvilágítási viszonyok mellett, vagy építésre kész részletes kiviteli terveket állítanak össze. A programok természetesen figyelembe veszik pl. az ipari szabványok és a helyi építési hatóságok előírásait is.

Az ilyen programok még a mai napig sem terjedtek el teljesen. A számítógép elsősorban a szakértői rendszer szerepét látja el a tervezés költségigényes szakaszában, a tervrajzok részleteinek elkészítése során.

A nagyon gyors mikroprocesszorok (pl. a Pentium) megjelenése óta az építészeti terveket PC-n is rövid idő alatt el lehet készíteni, az eredményt nagyfelbontású színes képernyőn meg lehet jeleníteni, és ki is lehet nyomtatni. A kész, digitalizált rajzok kevés tárolóterületet vesznek igénybe.

Példa: egy hajlékony mágneslemez elegendő egy családi ház teljes építészeti tervdokumentációjának tárolására.

Az építészeti programok nemcsak a tervezés hosszadalmas és költséges részeinek munkáját veszik le a tervező válláról, hanem jelentős mértékben meggyorsítják a tervezés ismétlődő, terv – változtatás – új terv lépésekből álló kreatív szakaszát is. Ha pl. a tervezés előrehaladott időszakában valami miatt szükséges vagy ajánlatos változtatni a terven, akkor az összes megelőző részletet is megfelelően át kell dolgozni. A hagyományos módszerekkel ez viszonylag lassú folyamat, amelynek során gyakran lépnek fel hibák. Ugyanez a számítógép segítségével másodpercek alatt elvégezhető.

A megfelelő programok eleinte csak szoftver alakjában léteztek, időközben azonban számos ilyen feladat, pl. a perspektivikus ábrázolás, a felületek felismerése vagy a modell eltolása már fix programozású ROM tárolókban is elérhető.

Építészeti programok

Perspektivikus ábrázolás. A program a kétdimenziós metszetekből a perspektivikus ábrázolás szabályai szerint háromdimenziós terveket készít. A látószög tetszés szerint választható. Ennek során – esetleg zavaróan kuszának tűnő – vonalakból álló rajz jelenik meg, mivel a képen először az objektum összes éle látható.

Az **eltakart vonalakat és felületeket** egy olyan algoritmus tünteti el a rajzról, amely a háromdimenziós tervet hátulról előre haladva szintről szintre elemzi, és a képről mindent töröl, ami a kívánt látószög alatt nem látható. Csak ezután jelenik meg az objektum letisztult képe. Az ilyen algoritmusok sok számítógépidőt igényelnek. Nagyon gyors processzorok szükségesek ahhoz, hogy a számítógép egy képet a másodperc törtrésze alatt megjelenítsen.

Realisztikus képek. Hagyományos úton a tervrajz felvázolását háromdimenziós, mérethű modell építése követi. Az ezen történő változtatások rendkívül sok időt vesznek igénybe és drágák. A számítógép alkalmazása esetén ezt a feladatot a képernyőn megjelenített, realisztikus, könnyen változtatható ábrákkal lehet megoldani. Az ábrák nemcsak színesek és valósághűek, hanem lehetővé teszik azt is, hogy az objektumot különböző látószögből és különböző (közvetlen és közvetett) megvilágítási viszonyok között tanulmányozzuk. Bemutatják és kívánságra megváltoztatják a felületi szerkezeteket, a tervezett új épületet beleillesztik a valódi környezet képébe stb. A modern programok az építészeti terv gyors optimálását is elvégzik. A szoftver pl. dinamikus képsorozatok segítségével az objektumot úgy modellezi, mintha a szemlélő körbejárná.

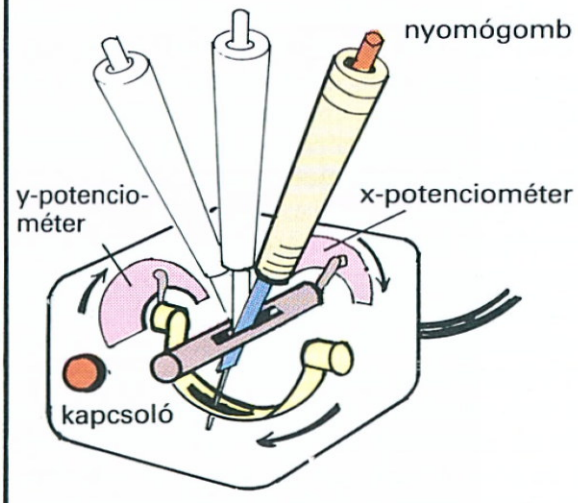
A **megvilágítás hatása** két, egymással ellentétes módon ábrázolható:

Teljes megvilágítás (angolul: *radiosity*). Megfelelő algoritmusok a dőlésszög, a fényvisszaverő képesség és az összes fényforrás ismeretében kiszámítják minden egyes felületen a fényintenzitás eloszlását. Ha a számítógép ezt a műveletet egyszer elvégezte, és az eredményt eltárolta, akkor a modell képét adott megvilágítási viszonyok esetén tetszőleges látószögből meg tudja jeleníteni. Ez a módszer különösen matt felületek ábrázolására alkalmas.

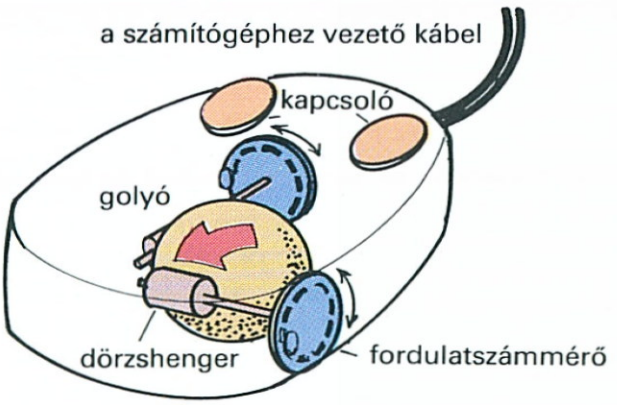
Sugárkövetés (angolul: *ray tracing*). Ezzel a módszerrel csak azokat a fénysugarakat vesszük figyelembe, amelyek a néző szemébe juthatnak. A program a szemlélt ábrán minden egyes sugarat fordított irányban, egészen a visszaverő felületekig és a fényforrásig követ. Így a sima, jól visszaverő felületekről nagyon valósághű képek keletkeznek. A módszer azonban nagyon számításigényes; ha a látószöget megváltoztatjuk, az egész algoritmust újra le kell futtatni.

Részletes tervezés. Az optimális modell és az építész adatai alapján további algoritmusok végzik a részletek tervezését. Ez különösen a toronyházak emeleteiről készített rajzok esetén érdekes. Az emeleten minden helyiséget a leendő feladatának megfelelően optimálisan rendeznek el.

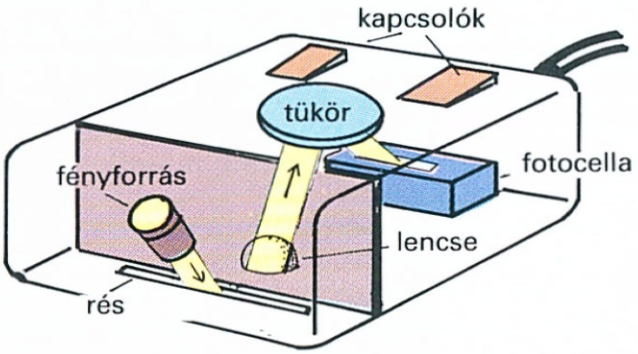
A CAD **előnyei** az építészetben: a rugalmasság és a terv gyors megváltoztatásának lehetősége; a tetszőleges látószögből történő szemlélésnek és a mozgás modellezésének lehetősége; a felhasznált színek és a felületi szerkezetek változatossága; a nagyon pontos és részletes rajzok készítése, valamint az időmegtakarítás.



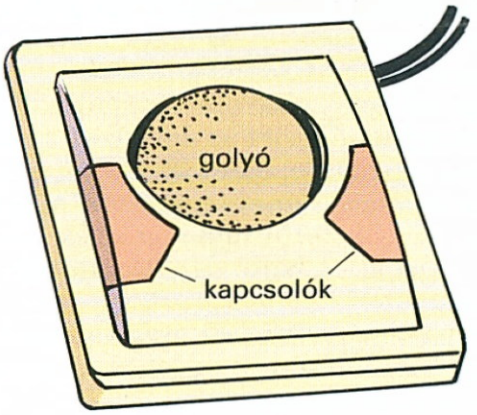
Botkormány (joystick)



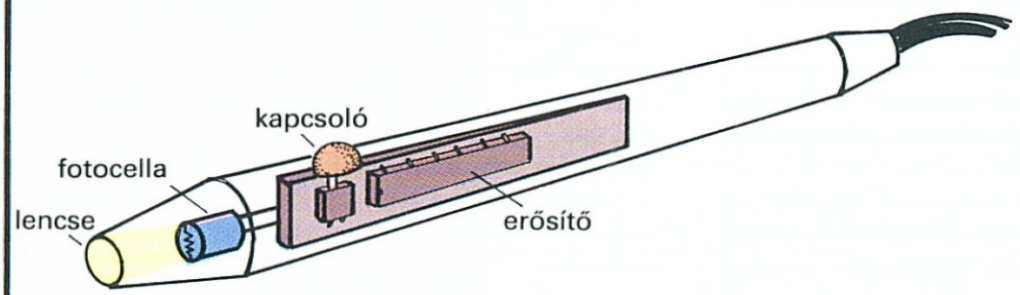
Egér



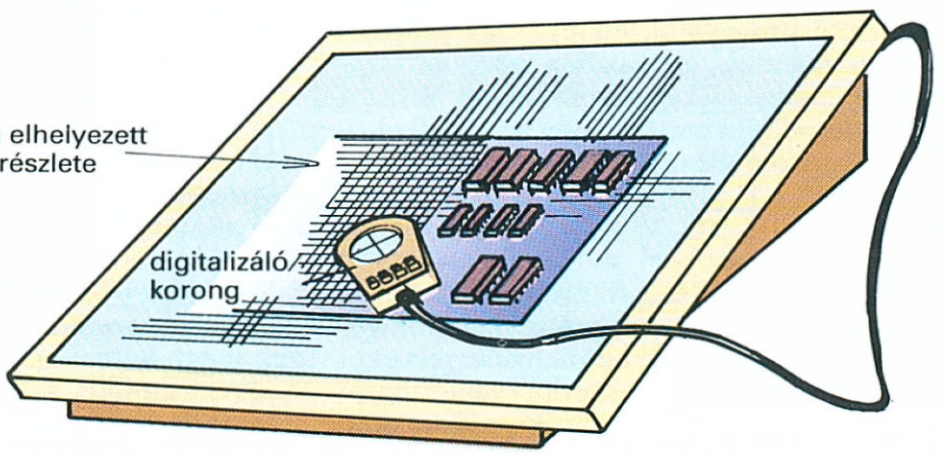
Optikai egér



Követőgolyó (track ball)



Fényceruza



Elektronikus rajztábla

A számítógépes tervezés (CAD) során használt adatbeviteli eszközök

A digitális számítógépek univerzálisan alkalmazhatók, tehát a CAD céljaira nem szükséges speciális központi egységet kifejleszteni. A perifériás eszközökkel azonban más a helyzet, mert a CAD olyan követelményeket állít, amelyeket a szokásos be-/kiviteli eszközökkel rendszerint nem lehet kielégíteni. Például egy átlagos PC alfanumerikus képernyőjén (640×480 képpont, és 256 szín) egy terv finomabb részleteit nem lehetne kellő felbontással ábrázolni. Az adatbeviteli eszközök közül a billentyűzet a CAD-munkaállomás számára gyakran túlságosan lassú és ügyetlen, inkább az olyan felhasználóbarát készülékek mutatkoznak alkalmasabbnak, mint az egér vagy a fényceruza. A számítógépes tervezést piktogramos menük is megkönnyítik.

A **piktogramok** a képernyő szélein vízszintes és/vagy függőleges sávokon jelennek meg. Az *egérrel* egy (általában nyíl alakú) *egérkurzort* lehet a képernyőn mozgatni úgy, hogy az a kívánt piktogramra mutasson. Az egér megfelelő gombjának lenyomásával (rákattintással) a megfelelő – pl. a „Sűrű csíkozással kitölteni" (192. o.) – folyamatot el lehet indítani.

A CAD olyan járulékos perifériákat is igényel, amelyek a „közönséges" számítógép tartozékaként csak ritkán fordulnak elő. Ilyen pl. az optikai letapogató készülék (szkenner), az elektronikus rajzasztal, a rajzgép (plotter), fotografikus megvilágító berendezés stb.

CAD bemeneti egységek

Néhány készülék csupán **kurzort** vagy más jelet mozdít el a monitoron, ezzel a kívánt pozícióban folyamatot indíthat el; ilyen készülék a billentyűzet, egér, fényceruza és botkormány. Más készülékek ezzel szemben közvetlenül információt továbbítanak a számítógépbe: pl. az elektronikus rajzasztal, a szkenner és a fényceruza.

Billentyűzet. Az alfanumerikus billentyűzeten – a helyközbillentyűn kívül – négy nyílbillentyű (fel, le, jobbra, balra) szolgál arra, hogy a kurzort a kívánt pozícióba juttassuk. Hasznos, ha a kurzor átlós mozgatására külön billentyűk is rendelkezésre állnak. A számítógéphez akár 80 mezős billentyűzetet is csatlakoztatni lehet, de időközben ezeket a funkciókat lehívható menük, vagy piktogramok helyettesítik.

Az **egér** rendkívül felhasználóbarát és olcsó adatbeviteli eszköz (121. o.), amelynek segítségével a kurzor (vagy gyakran egy *nyíl* alakú jel) a képernyőn kényelmesen mozgatható.

Az elektronikus egérnek nemcsak a neve, de a megjelenése is emlékeztet az ismert rágcsálóra: tenyérnyi nagyságú, általában szürke vagy fehér színű, és a végén vékony, hosszú kábel található.

Valójában egy kis dobozról van szó, amelynek alján gördülő golyó emelkedik ki. Ha az egeret sima felületre helyezzük és eltoljuk, akkor az egér mozgását súrlódó kerekek segítségével lehet „érzékelni", és analóg vagy digitális villamos jelekké lehet átalakítani. Az egérben található két szenzor az egér elmozdulását automatikusan két egymásra merőleges komponensre bontja. A villamos jel a kábelen keresztül jut a mikroprocesszorhoz, amely a jelek alapján a kurzor mozgatásához szükséges vezérlő jeleket kialakítja. Az egér megfelelő mozgatásával el lehet érni, hogy a kurzor vagy az egérnyíl a képernyőn a kívánt helyzetbe (pl. egy piktogramra, vagy a menü egy sorára, egy képelemre, valamelyik sor elejére vagy egy fogaskerék középpontjára) jusson. Az egérre szerelt egy vagy több nyomógombbal különböző történéseket lehet kiváltani. Ezt a folyamatot **kattintásnak** nevezzük. Az egérrel a kurzort sokkal gyorsabban és kényelmesebben lehet mozgatni, mint a billentyűzet segítségével. Az egeret először CAD-rendszereken vezették be, de időközben a más jellegű számítógépes munkát, pl. a szövegszerkesztést is nagymértékben megkönnyíti.

Léteznek **„farok nélküli" egerek** is. Ezekben az egér mozgását érzékelő szenzorok villamos jelei az egérben elhelyezett adót modulálnak. Az adó jelét egy detektor veszi, és a képernyő vezérlésére alkalmas jelekké alakítja.

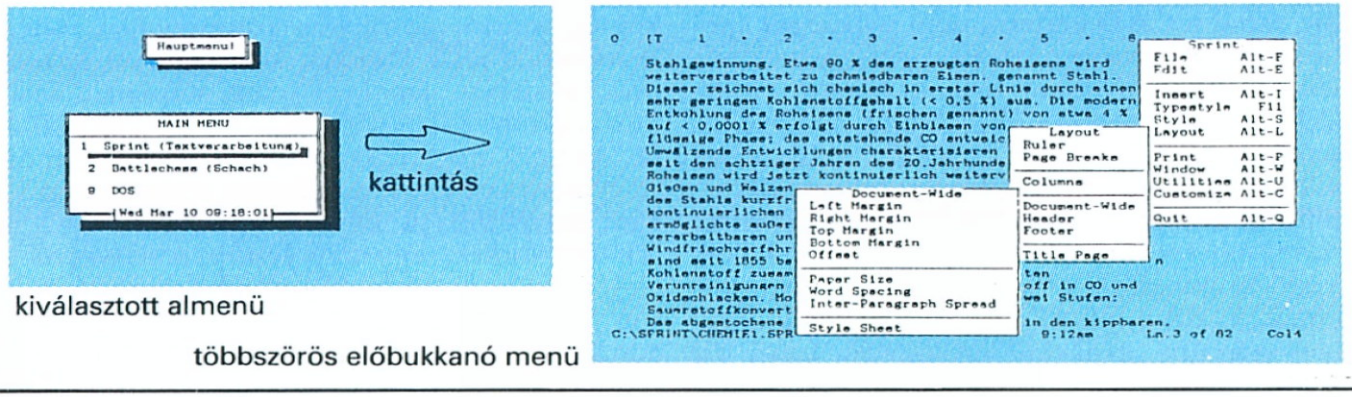
A **követőgolyó** (121. o.) lényegében egy hátára fordított, rögzített egér. A golyót közvetlenül a felhasználó görgeti.

A **botkormány** (121. o.) esetében a gördülő golyót csuklón mozgatható rúd helyettesíti. Elsősorban a képernyőn való függőleges, vízszintes és átlós mozgások irányítására alkalmas. A számítógépes játékok kedvelői is gyakran alkalmazzák.

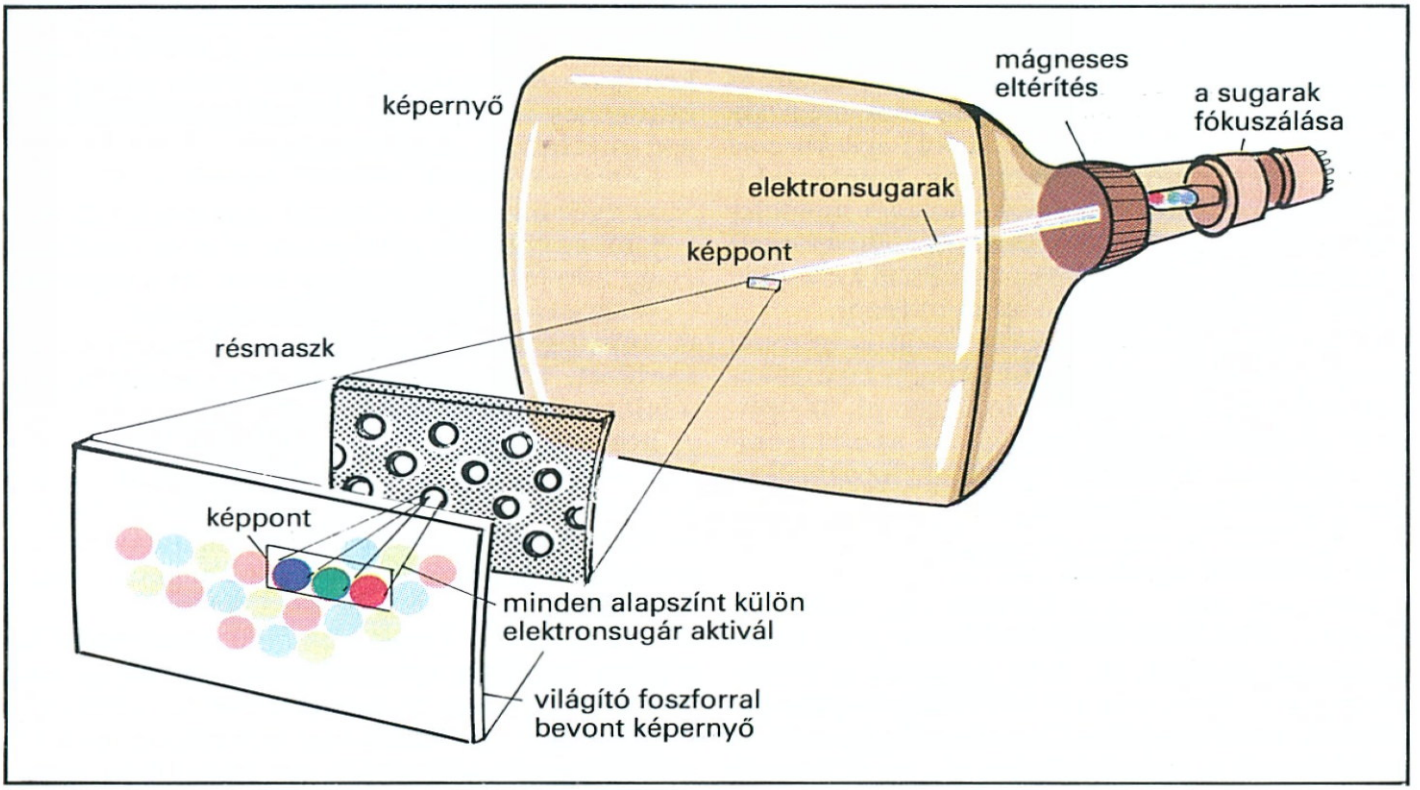
A **fényceruzát** (angolul: *light pen*) már az 1950-es években kifejlesztették. A fényceruzában a közönséges ceruza grafitját fényérzékeny detektor (pl. egy fotodióda) helyettesíti, és a készüléket a számítógéppel egy kábel köti össze. Ha a fényceruza hegyével megérintjük a képernyőt, a számítógép azonosítja az érintés helyének pozícióját. A fényceruzán található kapcsoló megnyomásával különböző folyamatokat is el lehet indítani, pl. egy menüt vagy egy piktogramot ki lehet választani, vagy egy képelemet meg lehet jelölni és/vagy el lehet mozgatni, vagy vonalat, kört és más alakzatot lehet rajzolni. Fényceruzával szabadkézi rajz is készíthető.

A fényceruza pozícióját a ciklikus képfelfrissítés során a megérintett képpont világos-sötét állapotváltozásának időpontjából lehet meghatározni. A vektorképernyőkön (195. o.) a fényceruza helyzetét nem lehet megállapítani, mert hiányzik a periodikus világos-sötét állapotváltozás.

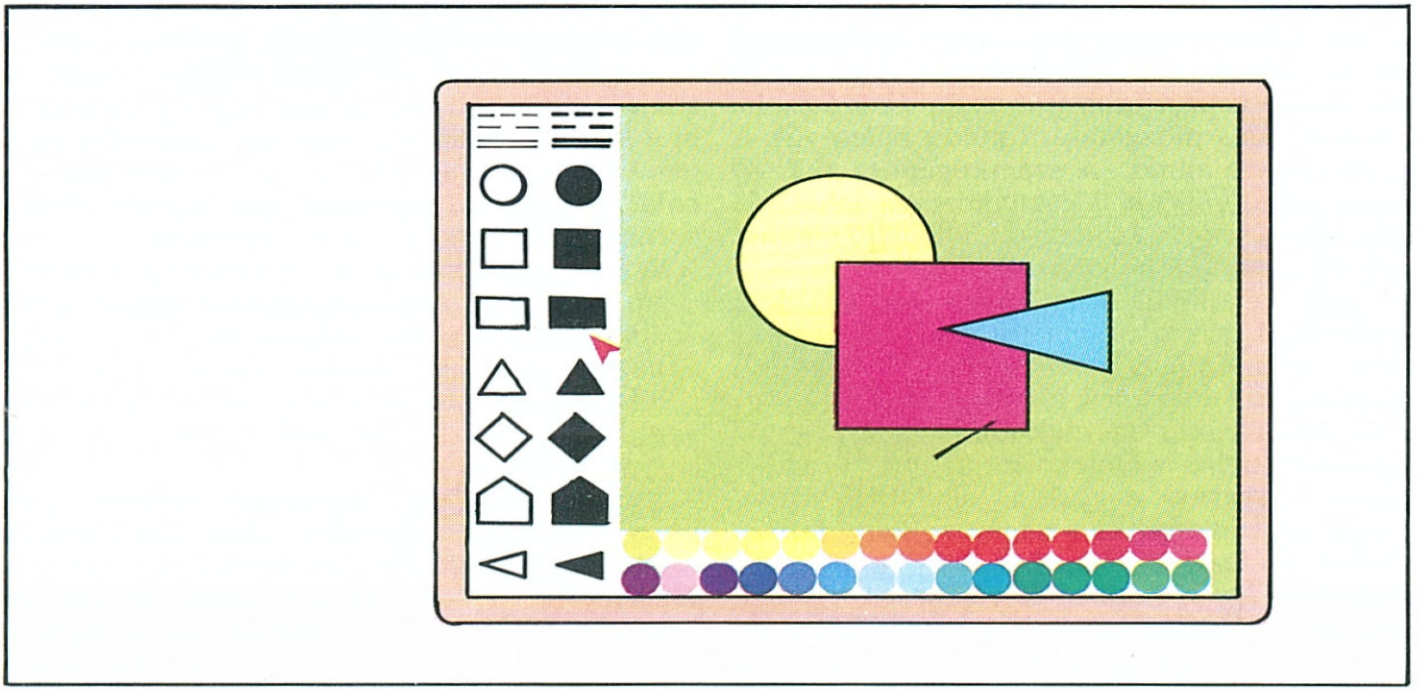
A **grafikus tábla** vagy **digitalizáló** (angolul: *graphics tablet* vagy *digitizer*) olyan elektronikus rajztábla, amelyen koordináta-rendszer van feltüntetve, és a készülék egy számítógéppel áll összeköttetésben. A készülékhez egy elektronikus ceruza, vagy különösen pontos munkához egy szálkereszttel ellátott nagyítós korong (angolul: *puck*) is tartozik. Az elektronikus rajztáblával grafikákat lehet digitalizálni.



Előbukkanó menük



Grafikus raszterképcsövek



Színes menüket és egérkurzort tartalmazó menüválaszték

A grafikus táblában nagyon vékony villamos huzalokból készült, sűrű szövésű háló található, amely rasztert képez. Minden metszésponton periodikusan ismétlődő áramimpulzus jelenik meg. Ezt a helyet a – kapcsoló megnyomásával aktivált – elektronikus ceruza vagy szálkeresztes lupe az impulzus futásidejéből ki tudja számítani. A rajzot a grafikus táblára helyezzük, ceruza, lupe vagy *puck* segítségével az összes fontos pont helyzetét megállapítjuk, és az adatokat a számítógép tárolójába juttatjuk. A görbe lefutását az egyes pontok között segédprogramok határozzák meg lineáris interpoláció alkalmazásával, így véges idő alatt elő lehet állítani a rajz teljes digitalizált képét.

Az elektronikus ceruza helyett korongot is használhatunk, amelynek helyzetét az általa kiváltott nagyfrekvenciás feszültségimpulzusok segítségével egy megfelelő érzékenységű raszteren meg lehet határozni. A nyomásra vagy fényre érzékeny felületek azt is lehetővé teszik, hogy közvetlenül a táblán rajzoljunk, miközben a készülék a rajzot rögtön digitalizálja.

A **szkenner** vagy **optikai letapogató** (angolul: *scanner*) a papíron megjelenő képeket vagy szöveget leolvassa és digitális jelsorozattá alakítja, amelyet tárolásra vagy további feldolgozásra a számítógépbe juttat. A papírt a fénymásolóhoz hasonlóan a képes vagy szöveges oldallal lefelé egy üveglapra kell helyezni, majd ezt az alatta található letapogató-berendezés „beolvassa". A digitális kamerák hasonló feladatok elvégzésére használhatók.

CAD kiviteli készülékek

A szokásos adatkiviteli készülékek felbontóképessége a grafikus információ megjelenítésére általában nem elegendő, a rajzok részleteit ezeken alig lehetne felismerni. A CAD igényeinek kielégítésére a képernyőt, a nyomtatót és a rajzológépet tovább kellett fejleszteni. A továbbfejlesztett készülékek felbontóképessége (vagyis az ábrázolt adatpontok négyzetcentiméterenkénti száma) lényegesen növekedett. Természetesen az ehhez szükséges adatkiviteli sebesség – egysége baud (bit/s) – is nagyságrendekkel nagyobb.

Különleges fényképezőgépekkel a képernyőn megjelenő képeket közvetlenül le lehet fényképezni. A CAD céljaira használt legfontosabb adatkiviteli eszköz a nagyfelbontású (grafikus) képernyő.

A **grafikus képernyő** képpontjainak száma eléri a 16 milliót. A képpontok legtöbbször 2084×1024 elemű mátrixba vannak rendezve. A képpont – *grafika* megjelenítéséhez a képernyőnek több, mint egymillió képpontot kell tartalmaznia.

Összehasonlítás: a közönséges alfanumerikus (vákuumképcsöves) képernyő legtöbbször 300 000-nél kevesebb képpontot tartalmaz.

A közbenső tárolóban, a *videopufferben* minden egyes képpontnak 1 bit felel meg, tehát a képelemek egyenként vezérelhetők. A legegyszerűbb képernyőkben a képpontoknak csak be-vagy kikapcsolt állapota létezik. A grafikus képernyőkben képpontonként 16 millió intenzitásfokozat is lehetséges. A színes képernyők esetén ehhez a (vörös, kék, sárga) szín megjelölésére képpontonként még további 3 bit járul. A videopuffernek tehát ebben az esetben legalább 2048×1024×24×3 bit, azaz kb. 19 Mbyte tárolókapacitással kell rendelkeznie.

A képi információnak egy gyors hozzáférési idejű tárolóban állandóan elérhetőnek kell lennie, mivel a képet – ha nem akarjuk, hogy vibráljon – a másodperc törtrészei alatt állandóan fel kell frissíteni. A szokásos képfrekvencia 72 Hz.

Különleges algoritmusok segítségével a tárolóterületet és ezzel együtt a videopuffert csökkenteni lehet. A grafikus ábrázolásra alkalmas képernyők ennek ellenére csak az 1980-as évek óta jelentek meg, amióta a megbízható és gyors RAM tömegtárolók (Mbyte-chipek) sorozatgyártása megkezdődött.

Az alfanumerikus képernyők videopufferéhez lényegesen kisebb tárolókapacitás is elegendő: a képernyő többnyire 24 sorra és 80 oszlopra, tehát 1920 részre osztható. Minden részt csak az adott karakterkészlet egyetlen eleme foglalhatja el. Mivel minden szabványos karaktert 8 bites kód azonosít, és a karakterek a ROM tárolóban megtalálhatók, a videopuffer számára 2 Kbyte tárolókapacitás elegendő.

Raszterképernyő

A raszterképernyők képe a televízióképhez hasonlóan soronként és oszloponként pontokból épül fel.

A televízió képcsövében az egyes képpontokat elektronsugarak és (lyukakat tartalmazó) maszkok határozzák meg. A lapos képernyőkben ezt a feladatot megfelelően elrendezett elektródok látják el.

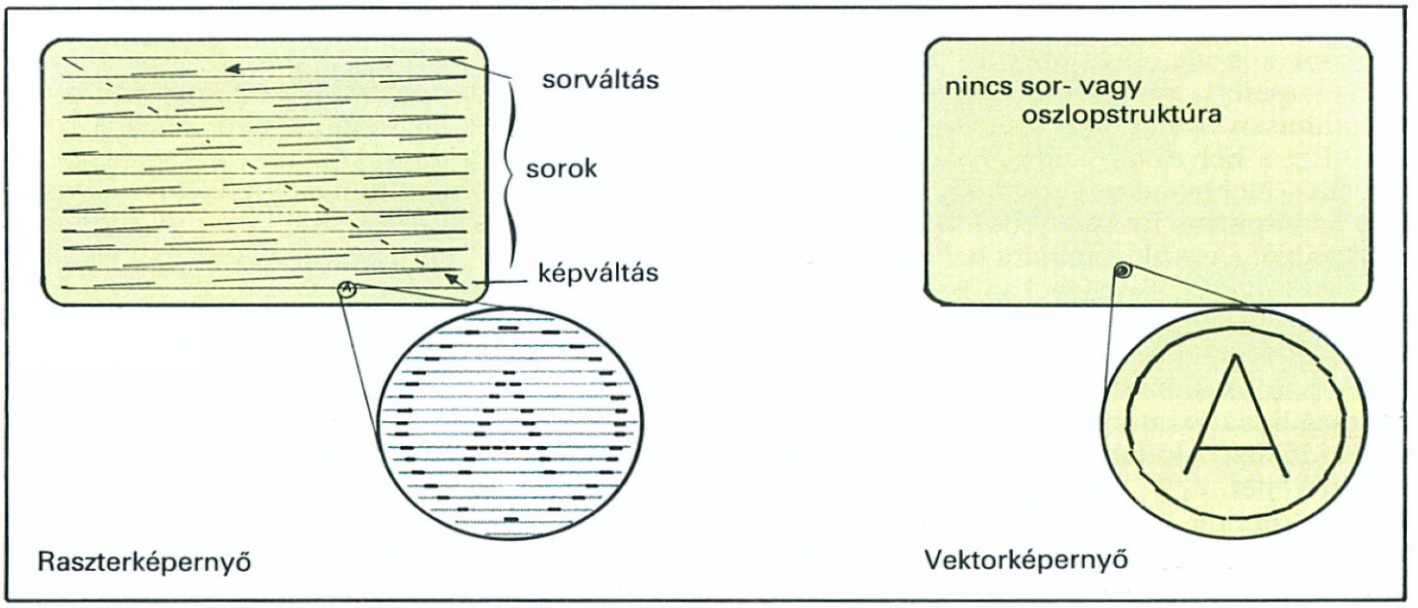
A kép időben ismétlődő periodikus felépítése minden egyes képpont helyzetét egyértelműen rögzíti. Ezért a képpont helyzetét pl. fényceruzával lokalizálni lehet.

A legtöbb vákuumcső foszforbevonatának felvillanási ideje nagyon rövid. Az elektronsugár által gerjesztett minden egyes képpont 1/60 s és 1/30 s közé eső időtartam alatt kialszik, tehát gyors egymásutánban fel kell frissíteni.

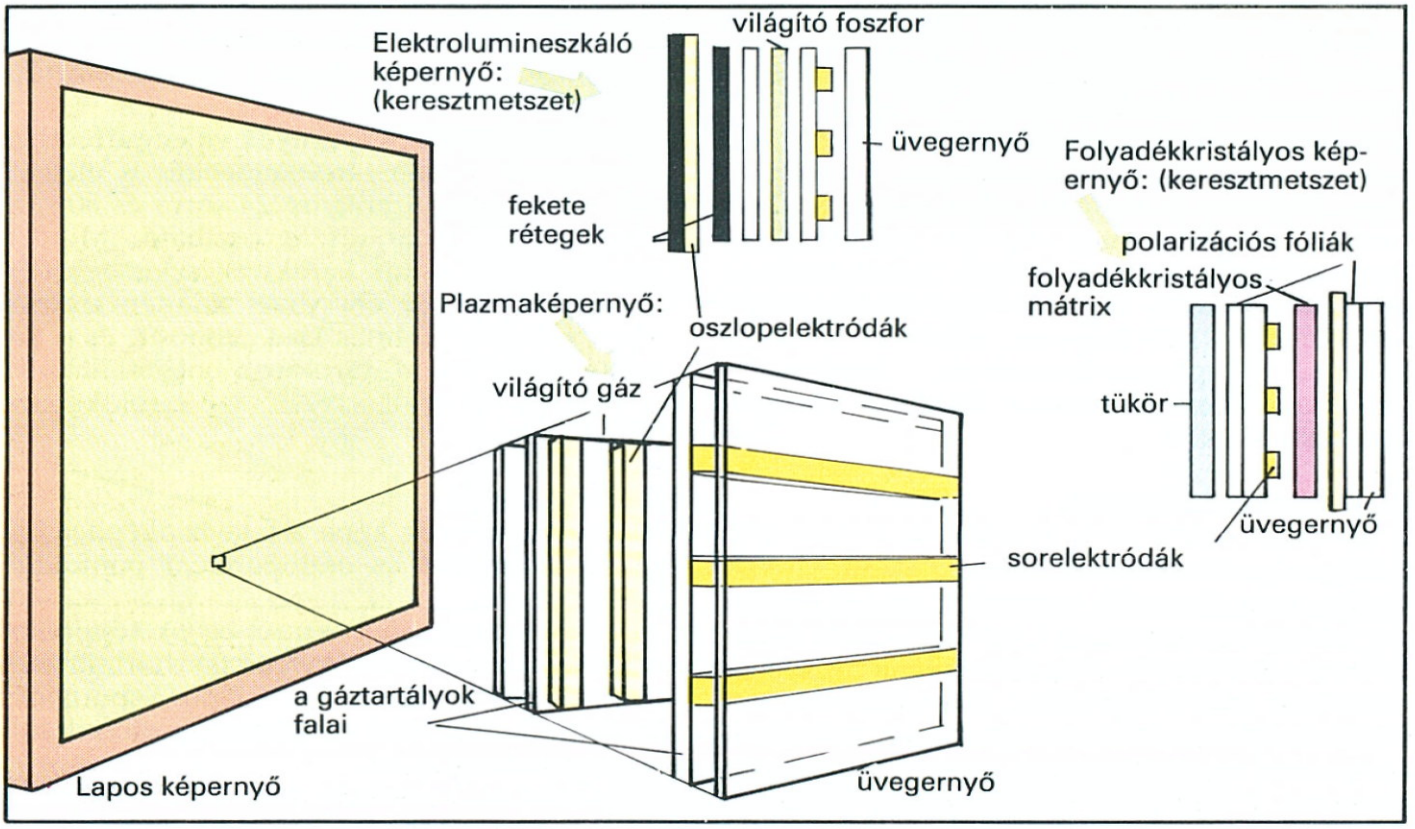
A raszterképernyő lineáris képpontsűrűsége kb. 3 képpont/mm. Ez közönséges távolságból szemlélve az emberi szem felbontóképességénél kisebb. A képpontokból összetett vonalak tehát hézagmentesen jelennek meg.

A raszterképernyők előnye a nagy felbontóképesség, valamint az, hogy a kép részben vagy teljesen gyorsan törölhető.

Hátránya: nagyon nagy felbontású képernyők esetén a kép felépítése viszonylag hosszú időt, valamint gyors tárolókat és gyors processzorokat igényel.



A raszter- és vektorképernyők képfelépítése



A lapos képernyő szerkezete

	alfanumerikus	bitleképezéses	vektor	vektor (felfris.)
a kép keletkezése	szabványos raszter	képpontraszter	vonalak	vonalak
felbontóképesség	csekély	nagy	nagyon nagy	nagyon nagy
információtartalom	csekély	magas	magas	közepes
részleges törlés	igen	igen	nem	igen
szürkeségi fokozatok	igen	igen	nem	igen
színek	néhány	igen	nem	néhány
mozgóképfelvétel	alig	igen	nem	igen

Az ábrázolási módok összehasonlítása

Vektorképernyők

A vektorképernyők eltérítő elektronikája az elektronsugarat két pont között csak egyenes vonal mentén téríti el. Az elektronsugár a pontokat összekötő vonalat folytonosan húzza meg, így minden kép egyenes szakaszokból épül fel.

A vákuumképcsövek világító foszforbevonata hosszú utánvilágítású, a képet csak ritkán kell felfrissíteni. A videopuffernek ennélfogva nem kell nagy tárolókapacitással rendelkeznie.

A vektorképernyőn a megjelenített ábra egészet alkot: csak az egész képet lehet törölni, a részleteket külön nem lehet javítani.

Kompromisszumos megoldásként közepes utánvilágítású foszfort alkalmaznak. A képet ilyenkor is időnként fel kell frissíteni *(vector refresh)*, a kép korrekciójára korlátozottan lehetőség van. A videopuffer – a raszteres képernyővel szemben – még ebben az esetben is viszonylag kevés tárolókapacitással működik, mert csak a szakaszok végpontjait tárolja, a közbeeső képpontokat viszont nem.

Előnye a gyors képfelépítés.

Hátránya viszont az, hogy kevés javítási lehetőséget nyújt, csak kevés szín jeleníthető meg, fényceruza alkalmazása nem lehetséges és csak statikus képek ábrázolhatók.

Lapos képernyők

A lapos képernyők mélysége mindössze néhány centiméter, ezzel szemben a vákuumcsöves képernyőké kb. fél méter. A lapos képernyők tették lehetővé a hordozható PC-k megjelenését. A réteges szerkezet és a (rázkódásra érzékeny) helyigényes vákuumképcső hiánya elvileg tetszőleges képernyőnagyságot tesz lehetővé, a méreteket csupán a költségek korlátozzák. Világító gázplazma-, folyadékkristály- és elektrolumineszkáló rétegek helyettesítik a hagyományos raszteres vektorképernyőkben alkalmazott foszforbevonatot. A világító réteget mindkét oldalon hajszálvékony átlátszó elektródák hálózata vonja be, és ez látja el a képpontok vezérlését. Elektronsugárra tehát nincs szükség.

A lapos képernyők részaránya a termelésben már 1993-ban kb. 45% volt. 2000-ig ez a szám a 70%-ot is elérheti, közülük valószínűleg az LCD-képernyők (lásd alább) részesedése lesz a legnagyobb.

A **plazmaképernyőben** (angolul: **P***lasma* **D***is-charge* **P***anel,* **PDP**) két, átlátszó lap erősen ritkított világító neon- vagy argongázt zár közre. Az első lapot nagyon vékony, átlátszó és vízszintes elrendezésű elektródasorok borítják, a hátsó lapon pedig megfelelő függőleges elrendezésű elektródák találhatók. A két elektródamintázat között állandó villamos feszültség van jelen, ez a feszültség azonban gázkisülés megindításához még nem elegendő. Ha egy függőleges és egy vízszintes elektródra egyidejűleg feszültségimpulzust kapcsolunk, akkor a két elektróda metszéspontjában a gáz világítani kezd. Ha a feszültségimpulzus megszűnik, akkor az alapfeszültség a kisülést fenntartja. Ezzel a módszerrel egy világos-sötét képpontokból álló raszterképet lehet felépíteni. A kép törlése az összes villamos feszültség kikapcsolásával történik. A képpontok legtöbbször narancs színűek, de több színből álló képeket is létre lehet hozni.

Hátránya: legtöbbször csak egyetlen szín érhető el, intenzitásfokozatok nincsenek, és a kép különálló részeit nem lehet törölni.

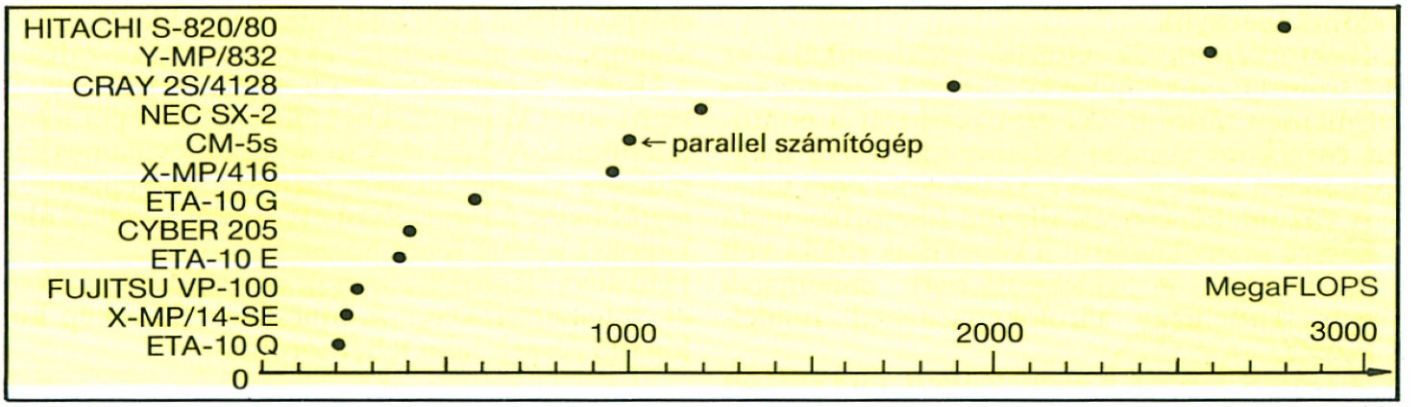
Az **elektrolumineszkáló képernyő** (angolul: **EL***ectroluminescence display,* **EL-***display*) a plazmaképernyőhöz hasonló felépítésű, de a világító gázréteget nagyon vékony, éppen ezért átlátszó foszforréteg helyettesíti. Villamos feszültségimpulzus hatására két elektróda metszéspontjában a foszfor – legtöbbször sárga színnel – világít. A fényelnyelő hátlap a világító pontokat fekete háttérben tünteti fel.

Előnyei: az egyes pontok intenzitása független az alkalmazott villamos feszültségtől, állandó alapfeszültségre nincs szükség, a pontok törlése egyenként is lehetséges.

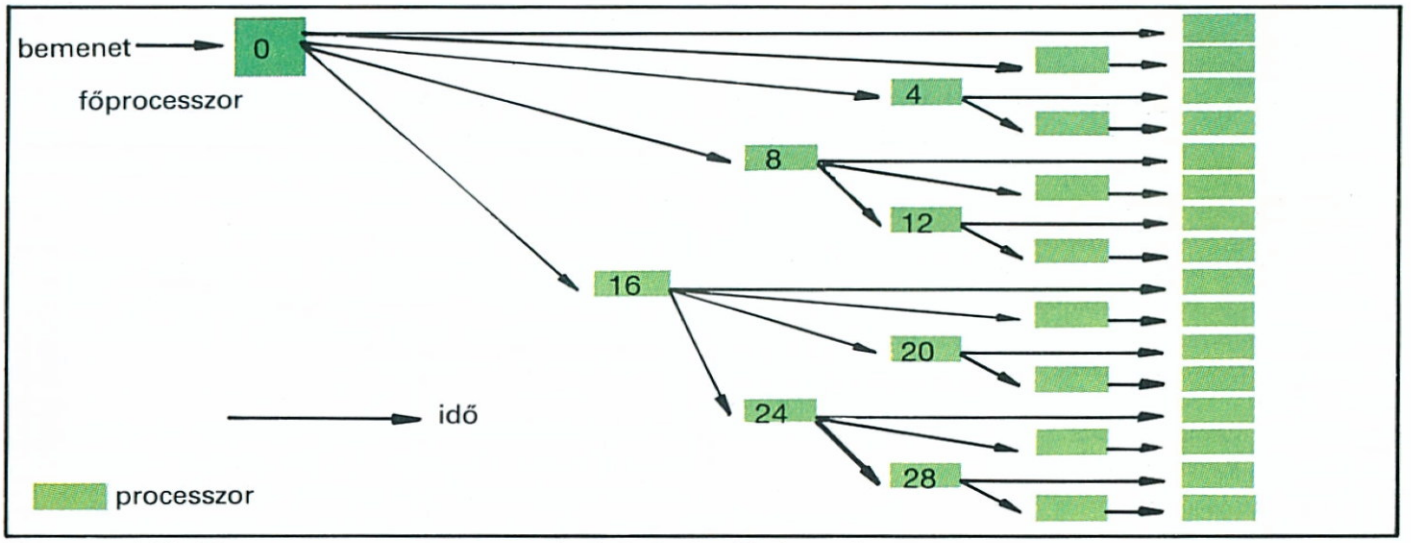
A **folyadékkristályos képernyőben** (angolul: **L***iquid* **C***rystal* **D***isplay,* **LCD**) átlátszó elektródasorok és polarizációs szűrők milliméternél vékonyabb, de nagy síkbeli kiterjedésű folyadékkristályt zárnak közre. Az átlátszó rétegek hátsó oldalát tükröző felület alkotja. Ha a képernyőre elölről fény esik, annak legnagyobb részét a szűrők elnyelik. A kevés maradékot a tükör visszaveri, és a képernyő emiatt szürke színűnek látszik. Ha két elektróda metszéspontjában villamos feszültségimpulzus jelenik meg, akkor a folyadékkristály molekuláinak optikai tulajdonságai úgy változnak meg, hogy ezen a helyen a beeső fénysugár teljesen elnyelődik, így ez a hely a világosszürke háttérben sötétnek látszik.

Több egymás mögött elhelyezett különböző folyadékkristály-réteg segítségével színes képeket is elő lehet állítani.

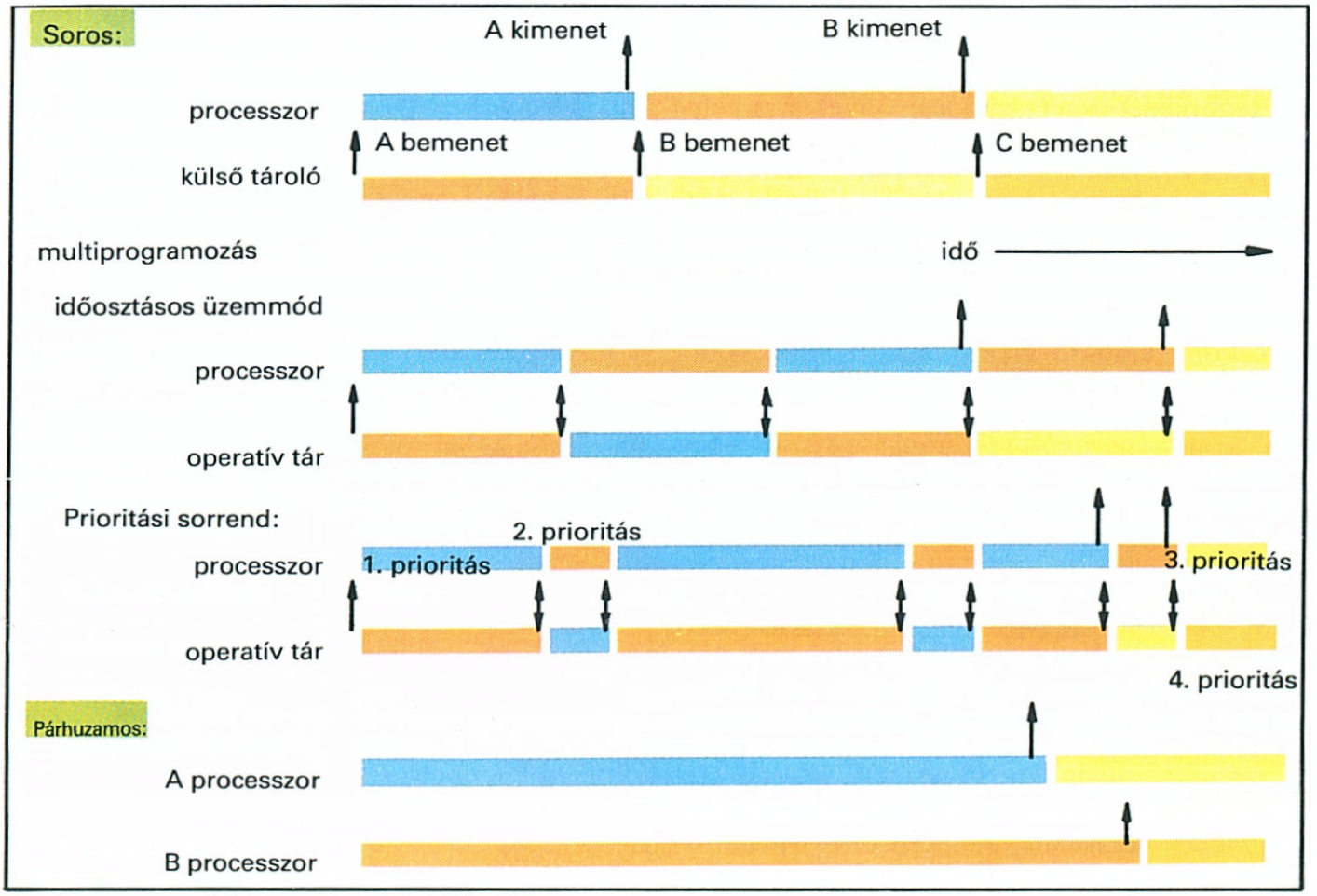
Előnyei: az LCD-képernyők képe pontonként törölhető, tehát raszterképek előállítására kiválóan alkalmas. További előnye, hogy a képernyő vezérlése nagyon kevés villamos energiát igényel.



Néhány szuperszámítógép teljesítménye



Egy speciális 32 processzoros szuperszámítógép logikai szerkezete



Több program feldolgozásának időbeli szervezése

Az egyes processzorok hardver elemeinek teljesítőképessége valószínűleg elérte a tetőpontját, bár az ilyen kijelentésekkel nagyon óvatosnak kell lenni. De még számos olyan probléma van, amelyet az 1990-es évek számítógépei egyelőre nem oldottak meg. Ezekhez tartozik elsősorban az általános időjárás-előrejelzés és az összetett szerves molekulák háromdimenziós szerkezetének elemzése, de katonai területen is léteznek olyan feladategyüttesek, amelyeket a kifinomult mikrochipek ellenére máig sem sikerült megnyugtatóan megoldani.

Úgy tűnik, hogy a számítógépek teljesítőképességének jelentős növekedése – megfelelő programozás mellett – csak a számítógép alkotórészeinek párhuzamos felépítésével, valamint a számítógépes hálózatok fejlesztésével valósítható meg. Ezt az árak alakulása is elősegíti: a kompakt felépítésű, igen nagy integráltságú mikroprocesszorok, a **VLSI-chipek** (angolul: *Very Large Scale Integration*) előállítási költsége időközben alacsonyabb, mint a számítógépes hálózatok kommunikációs költségei.

Az elektronikus komponensek javítása

Komponenssűrűség. A számítógépen belül a fénysebesség korlátozza az elektronikus alkatrészek közötti adatátviteli sebességet, amelynek lényeges növekedése nem várható, mert az elektronikus alkatrészek sűrűsége már a jelenlegi technológiával elérhető felső határ közelében van. Pl. az egyik legnagyobb teljesítőképességű számítógépnek (CRAY3) a magja alig nagyobb, mint egy cipősdoboz. Az ohmos ellenállás következtében benne keletkező hőt intenzív hűtéssel kell elvezetni. A legnagyobb teljesítményű számítógépek némelyikét cseppfolyós nitrogénnel hűtik. A szupravezető elemek használata esetén a komponensek sűrűségét tovább lehetne fokozni, de nem olyan mértékben, mint amilyenhez az 1970-es évek óta hozzászoktunk.

Optikai tárolók. Az újabb fejlesztések térfogategységenként több tárolóhelyet ígérnek. Az 1990-es évek eleje óta olyan optikai tárolók vannak tesztelés alatt, amelyek egy CD lemezen kb. 10^{12} byte tárolását teszik lehetővé.

Ez a tárolási kapacitás hozzávetőleg egymillió, egyenként 300 oldalas könyvnek felel meg.

A **SERODS**-technológiával (angolul: *Surface-Enhanced Raman Optical Data Storage*) talán még nagyobb információtárolási sűrűség érhető el. 1 bit tárolására a hozzávetőleg mindössze száz molekulából álló anyag csoportosítására (angolul: *cluster*) van szükség.

Hibaszegény chipek. Az elektronikus alkotórészek integráltsági fokát és árát a gyártási hiba is korlátozza. Első közelítésben a hibaarány a chipek felületével exponenciálisan növekszik. Bizonyos integráltsági sűrűség fölött (128 MB/chip?) a gazdaságos előállítás már nem tűnik lehetségesnek.

Példa: ha az előállított összes chip ¼-e hibátlan (ez valóságos érték), akkor a chip felületének megkétszerezése és azonos komponenssűrűség esetén ez az arány már csak $0{,}25^2$, azaz 6,3%. Ha a hibavalószínűség 90% (ez nem szokatlan a VLSI-chipeknél), akkor a megfelelő kinyerés 1%-ra csökken.

A jövőben a VLSI chipek hibatűrő tervezése jelenthet (legalább részben) kiutat. Jelenleg az ilyen algoritmusok még ritkák, és számítások elvégzésére hosszabb időt vesznek igénybe.

Multiprogramozás

A *multiprogramozás* alkalmazásával fokozható a hardveren áthaladó információáramlás. Ha pl. az n program futása megáll, mert egy B/K-csatornát hívott meg, akkor a program a készen álló m program feldolgozását kezdi meg, és folytatja mindaddig, amíg ez a program is megáll, és így tovább. A prioritásokat a felügyelőprogram állapítja meg.

Azt a megoldást, amikor az operatív tárba betöltött programnak rögzített időtartam áll rendelkezésére, **időszeletelésnek** (angolul: *time slicing*) nevezzük.

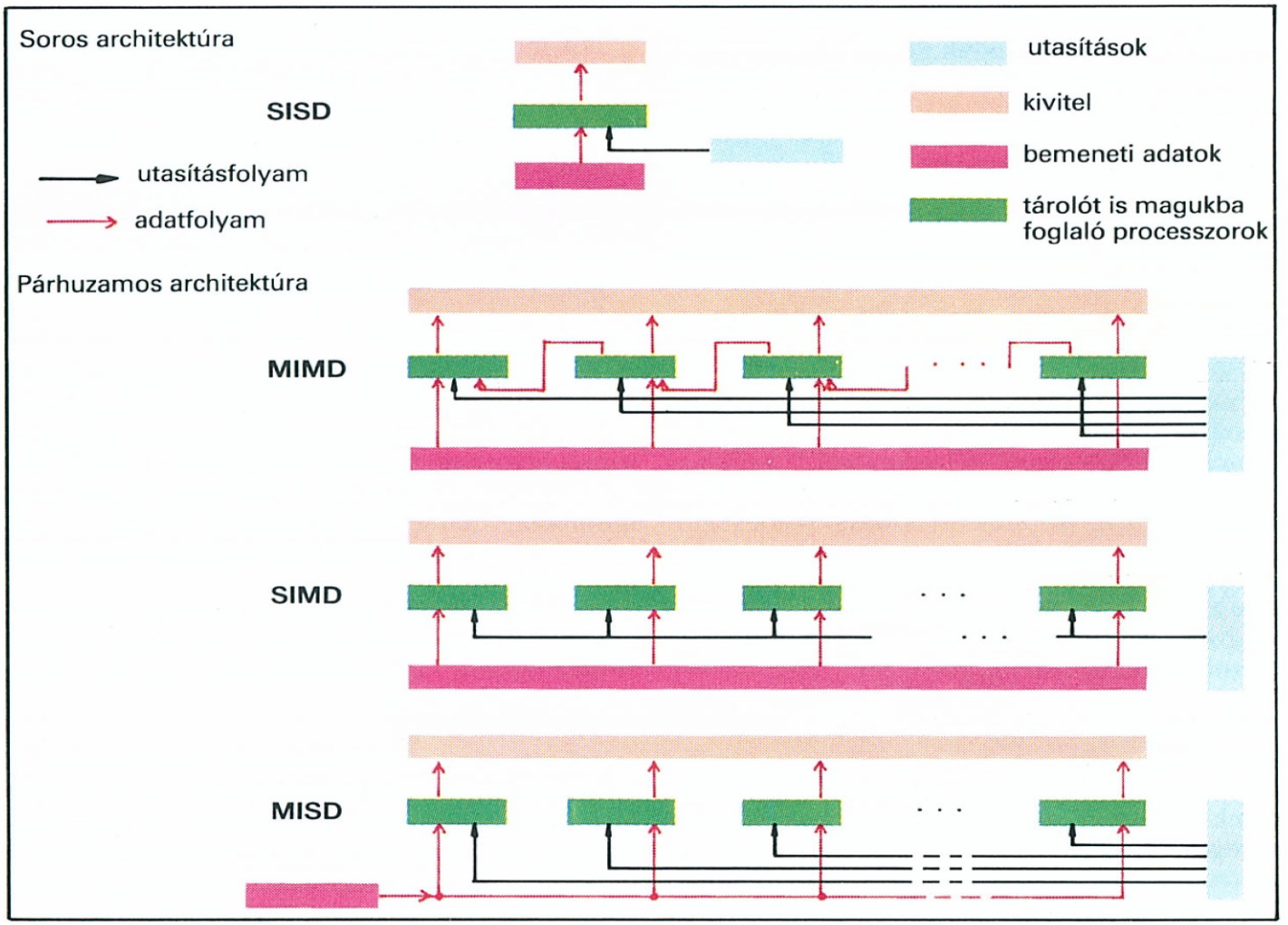
A számítógép architektúrájának tökéletesítése

A számítógépek architektúráját az adott felépítésű számítógép által egyidejűleg feldolgozott utasítás- és adatáramlás mennyisége szerint osztályozhatjuk. Ez a fogalom elsősorban a hardverre vonatkozik, de a szoftver is jelentős szerepet játszik benne.

A **SISD**-architektúra (egyszeres utasításfolyam, egyszeres adatfolyam; angolul: *Single Instruction stream, Single Data stream*) a hagyományos, soros felépítésű számítógépre jellemző, amelyet *Neumann-féle számítógépnek* is nevezünk. Ebben a gépben egyetlen processzor van, amely az utasításokat lépésről lépésre, azaz **szekvenciálisan,** rögzített sorrendben hajtja végre. A processzor csak akkor kezd az $n + 1$-edik lépéssel foglalkozni, ha az n-ediket már befejezte.

SIMD-architektúra (egyszeres utasításfolyam, többszörös adatfolyam; angolul: *Single Instruction stream, Multiple Data stream*). Az ilyen felépítésű számítógépben több „saját tárolóval is rendelkező" processzor van. Valamennyi processzor **párhuzamosan** működik, vagyis egyidejűleg ugyanazt az utasítást hajtja végre, azonban az egyes processzorokat más és más adatfolyam éri el. Az első SIMD-architektúrájú gép a 64 processzoros ILLIAC IV volt, amelyet az 1960-as évek végén építettek.

MISD-architektúra (többszörös utasításfolyam, egyszeres adatfolyam; angolul: *Multiple Instruction stream, Single Data stream*). Ez a számítógéptípus is több processzort tartalmaz, amelyek egyetlen adatfolyamon **párhuzamosan** (egyidejűleg) különböző utasításokat hajtanak végre. Ennek a gépnek egyelőre alig van alkalmazási területe, és a leendő fejlesztési lehetőségek még nem ismertek.



Számítógép-architektúra

Futószalagszerű feldolgozás

Ahogy 1950-ben elképzelték (megaFLOPS)	Ahogy 1992-ben képzelték (100 teraFLOPS)
népszámlálások kiértékelése	átfogó időjárás előrejelzés
gazdasági tervezés a világkormányzat számára	világgazdasági modellek
szerkezeti anyagok feszültségvizsgálata	szerves molekulák elemzése
a hangnál sebesebb repülés dinamikája	szélcsatorna-szimuláció
lokális katonai logisztika	globális katonai logisztika
repülésirányítás	neuronális hálózatok
az „egységes térelmélet" bizonyítása	a világegyetem modellje
nagymesteri színvonalú sakkjáték	hologramok szintézise

● = időközben megoldott feladat

A leendő szuperszámítógépeknek szánt feladatok

MIMD-architektúra (többszörös utasításfolyam, többszörös adatfolyam; angolul: ***M**ultiple **I**nstruction stream, **M**ultiple **D**ata stream*) esetén nagyszámú processzor egymástól függetlenül és **párhuzamosan,** egyidejűleg dolgozza fel a különböző adatfolyamok adatait, és ennek során a részeredményeket átveszik egymástól. Ez a legáltalánosabb és potenciálisan a legnagyobb teljesítőképességű számítógép-architektúra. Az első, szorosabb értelemben is parallel számítógépnek tekinthető típusok az 1970-es évek közepe óta jelentek meg a számítógéppiacon.

A számítógépek teljesítőképességét gyakran **FLOPS** (angolul: ***F**loating point **O**perations **P**er **S**econd*), vagyis a lebegőpontos műveletek per másodperc egységekben adják meg.

Példa: az INTEL RX számítógép 128 mikroprocesszort tartalmaz, 32 bit szóhosszúsággal és egyenként 8 MB tárolókapacitással. A gép teljes számítási teljesítménye 5 gigaFLOPS.

Az 1990-es években az ilyen architektúrájú gépek processzorainak a száma a 64 000-et, teljesítménye a teraFLOPS nagyságrendet is elérheti.

Soros működésű számítógépek

A soros adatfeldolgozást (angolul: *serial processing*) alkalmazó számítógépek sebessége – az ügyesebb programozáson kívül – csak az egyes alkatrészek, a hardver sebességének növelésével fokozható. Ezen a téren azonban valószínűleg már a felső határ közelébe jutottunk.

A soros számítógép **teljesítményének növekedése** az elmúlt néhány évtized alatt rendkívül figyelemreméltó volt (és a jövő számítógépeivel szemben túlzott elvárásokhoz vezetett). Az 1940-es évek leggyorsabb számítógépe még alig néhány kFLOPS sebességgel működött, az 1990-es évek elején ez a teljesítőképesség már néhány száz gigaFLOPS. Ez 8 nagyságrenddel történő növekedést jelent.

A soros számítógép lényeges hátránya, hogy – a *multiprogramozás* (lásd fent) és a *futószalagszerű feldolgozás* (lásd alább) kivételével – az egyes számítógép-komponensek kihasználtsága nagyon rossz, mivel a legtöbb időt az adatokra történő várakozással töltik. Ennek ellenére a soros számítógép még az ezredfordulóig meghatározó lesz a számítógépek piacán.

Parallel működésű számítógép

A párhuzamos adatfeldolgozás (angolul: *parallel processing,* vagy *massive parallel processing*) elsősorban MIMD architektúrájú számítógépekkel valósítható meg. A megoldási algoritmus egyes részeit bizonyos számú processzor egymástól függetlenül és párhuzamosan (vagyis egyidejűleg) oldja meg. A részeredményeket a processzorok egymás között kicserélik. Ennek az eljárásnak az előfeltétele az, hogy a programot párhuzamos folyamatokra lehessen felbontani. Gyakorlatilag elkerülhetetlen, hogy néhány processzor a másik processzor eredményeire valamennyi ideig ne várjon. Tehát n számú parallel processzor a számítási sebességet csak $< n$ tényezővel növeli.

A **futószalagszerű adatfeldolgozás** *(pipelining)* a párhuzamos adatfeldolgozás egyik változatának tekinthető. Ezt a technikát a nagy teljesítményű, soros számítógépekben széles körben alkalmazzák. A vezérlőműben az egyes működési egységek időbeli átfedéssel különálló műveleteket dolgoznak fel. Pl. az *a* utasítás végrehajtása alatt áll, a *b* utasítást egy párhuzamos folyamat dekódolja, a *c* utasítás pedig egyidejűleg a vezérlőműbe kerül. A futószalagszerű feldolgozás nem mindig lehetséges, pl. ugró utasításoknál felmondja a szolgálatot. A futószalagszerű feldolgozás során előfordulhat, hogy olyan utasítások is végrehajtásra kerülnek, amelyek fölöslegesek. Ez a feldolgozási mód mégis növeli egyes soros számítógépek teljesítőképességét.

Az **ILP** (angolul: ***I**nstruction **L**evel **P**arallelism*) módszer alkalmazása során különlegesen kifejlesztett processzorokat használnak a szokásos programok párhuzamosan elvégezhető szakaszainak feldolgozására. Ez különösen a PC-k esetén alkalmazható.

A parallel számítógépek komponenseinek kihasználtsága lényegesen nagyobb, mint egy soros számítógépé. Bár a processzorok számával növekszik a teljesítőképesség, azonban pl. a számítógépen belüli adminisztrációra, felügyeletre és szinkronizációra igénybe vett műszaki erőforrások nagysága is jelentősen növekszik.

Néhány olyan problémát sorolunk fel, amelyek a feladat jellegénél fogva különösen alkalmasak parallel számítógéppel történő megoldásra: bérelszámolás, neuronális hálózatok, hullámok interferenciája, képfeldolgozás, szerves óriásmolekulák analízise.

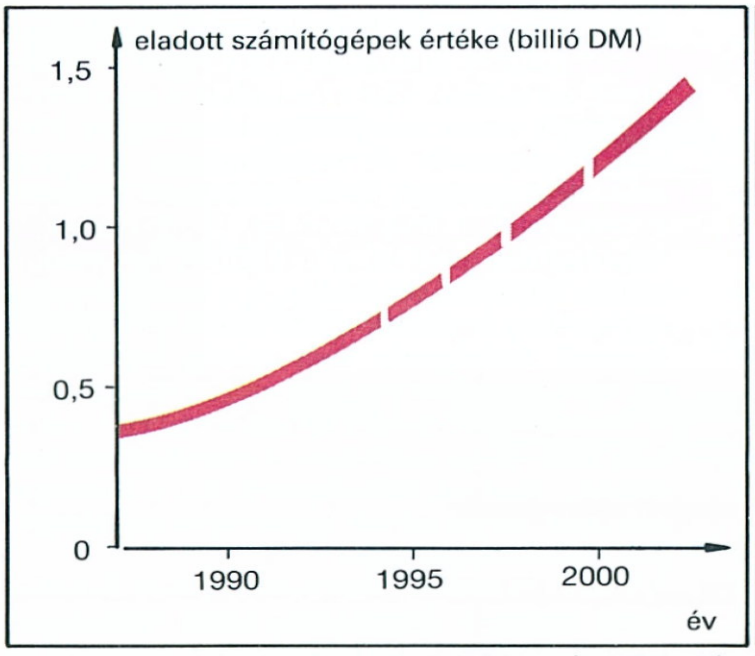
A parallel számítógép nagy hátránya az, hogy hiányoznak hozzá a megfelelő programnyelvek. A soros számítógépekre kifejlesztett rendkívül nagyszámú program miatt a parallel számítógépek az 1990-es években is csak a legnagyobb teljesítményű gépek kategóriájára korlátozódnak.

Térben elkülönült, párhuzamosan kapcsolt számítógépek. Mihelyt a távolsági összeköttetések adatátviteli sebessége a 100 megabaud értéket túllépi, a különböző nagyságú helyi számítógépeket egyetlen parallel számítógéppé lehet összekapcsolni. A programok blokkokra oszlanak, amelyek egyenként mindig a legoptimálisabb számítógépre kerülnek, és egy időben, párhuzamos üzemmódban kerülnek feldolgozásra.

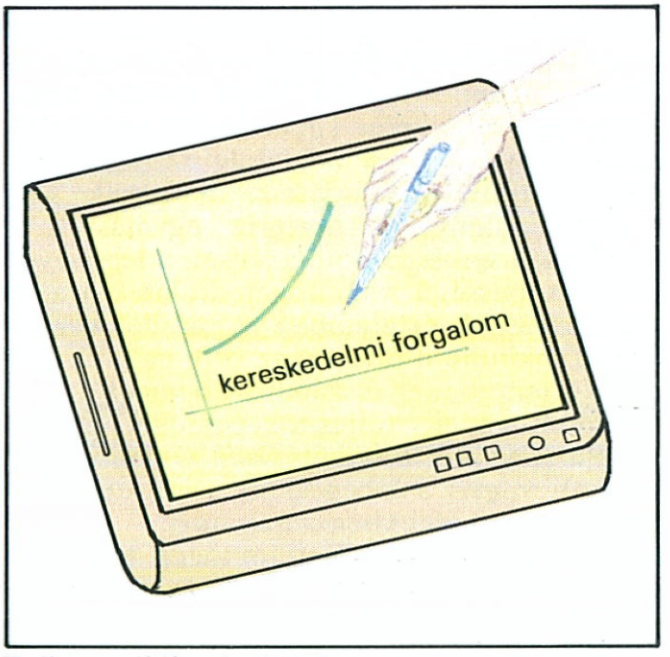
A fenti eljárás előfeltételét a megfelelően összehangolt programok, a szabványos B/K illesztőegységek és a viszonylag ritka részeredménycsere jelenti.

A fenti eljárás abban különbözik a számítógép-hálózatoktól, hogy azokban a mindenkori szabad gép különállóan dolgozza fel az egyedi programokat és a hozzájuk tartozó adatokat.

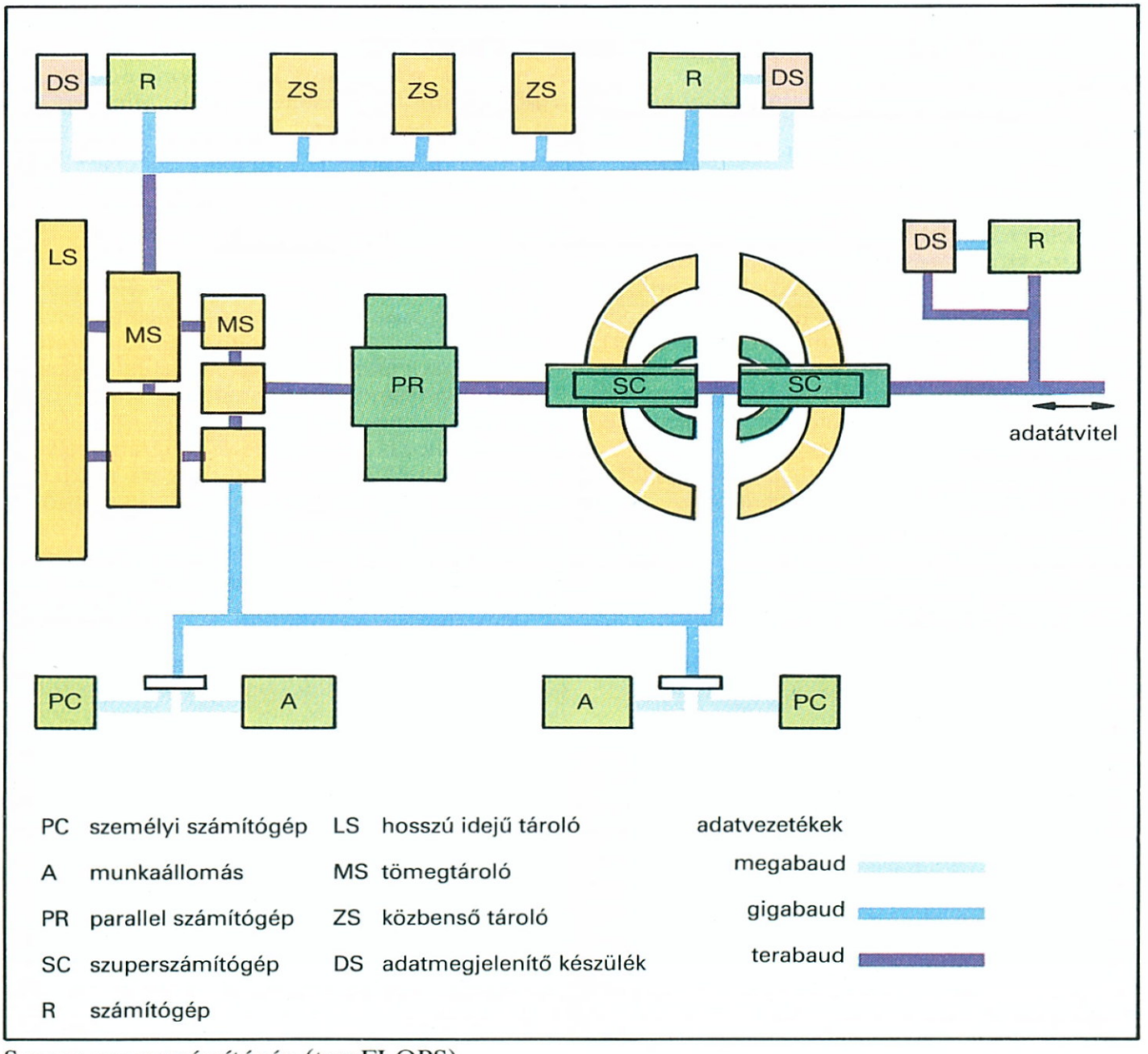
A parallel számítógépek a jövőben egyes intézmények, mint pl. az egyetemek, a kutatóintézetek és a katonaság munkájában jelentős szerepet fognak játszani.



A világ számítógép-piacának fejlődése (a PC-ktől)



Felhasználói számítógép



Szuper-szuperszámítógép (teraFLOPS)

A számítógép típusainak fejlődése a jelenlegi állapotot figyelembe véve az ezredfordulóig hozzávetőleg megjósolható. A számítógéppiacot valószínűleg két fejlődési irány fogja uralni: az egyik a „személyi" használatra alkalmas hordozható számítógépek, a másik pedig a legmagasabb igényeket kielégítő nagy számítógépek területén várható. A két véglet között minden átmenet is megtalálható.

Felhasználói számítógép

Az ezredforduló körül már valószínűleg olyan chipeket lehet készíteni, amelyeken 100 millió tranzisztor is elfér. Mivel a chip az adattárolót is magában foglalja majd, ez már valódi mikroprocesszor lesz, amelynek teljesítménye minden bizonnyal arra is elegendő, hogy a beszéddel közölt utasításokat is megértse. A képernyőkre valószínűleg még szükség lesz, mert a felhasználó a szöveget és a képeket látni is szeretné. A mikroprocesszor, az akkumulátorok és néhány illesztő egység a képernyő keretrészében fog elhelyezkedni. A számítógép maga tulajdonképpen egy karórában is elférne. A felhasználó nagyszámú programot tárolhat és behívhat, valamint számos adatbankhoz is hozzáférhet.

Más számítógépekkel az információcsere infravörös sugarakkal, telefonon, rádión vagy közvetlen vezetékeken valósul majd meg.

Alkalmazási példák. *Papírhelyettesítés:* kérdőívek, amelyeket a központ azonnal kiértékel; leltározás; árumegrendelés; biztosítási esetekben kárfelvétel; körlevelek; levelezés; telefonkönyv; lexikonok; árukatalógusok stb.

Táblahelyettesítés: Az iskolában a tanító közvetlenül felhasználói számítógépére ír, és ezzel egy elektronikus falitáblát vezérel. A tanulók az információt közvetlenül felhasználói számítógépükben tárolják.

Szövegszerkesztés, adatbank, számítógépes játékok.

Hátrányai. A felhasználói számítógép olcsó ugyan, de csak korlátozott ideig használható. A kompakt, zárt építésmód javítást vagy kiegészítést nem tesz lehetővé. További elektronikus komponensek újabb készüléket igényelnek. A beépített akkumulátorok kapacitása is véges, ezért rendszeres feltöltésre szorulnak.

A csúcsteljesítményű számítógépek

A 3. évezred elejének szuper-szuperszámítógépét három >T< betű jellemzi: **t**era-FLOPS teljesítmény, **t**erabaud információáramlási sebesség, és az operatív tár **t**erabit nagyságrendű kapacitása.

Ezeket az értékeket a hagyományos soros számítógép továbbfejlesztésével nem lehet elérni. Ennek oka az igénybevett elektronikus ráfordítások exponenciális növekedése – különösen a gallium-arzenid (GaAs)-technika esetén –, de az egyre áttekinthetetlenebbé váló operációs rendszerek is.

A gyártók által megadott teljesítményértékek napjainkban mindig maximális értéket jelentenek, amelyeket a programvégrehajtások során nem lehet elérni. A tesztek eredménye szerint a számítási teljesítmény a legtöbb program futása alatt alig éri el a megadott érték 10%-át.

A szuper-szuperszámítógép parallel számítógép lesz, vagyis az algoritmus a problémát több különálló processzoron egyidejűleg (parallel) végrehajtandó egyedi lépésre bontja és oldja meg. A számítógép teljesítőképességét gyakorlatilag csak a parallel processzorok száma korlátozza.

A probléma jellege és a költségek alapján háromféle számítógép-architektúra lesz uralkodó:

Néhány nagy teljesítményű számítógépet párhuzamosan kapcsolnak, és mindegyik számítógép egy közös operatív tárhoz férhet hozzá.

Példa: az X-MP számítógép két parallel működésű CRAY 1 számítógépből áll.

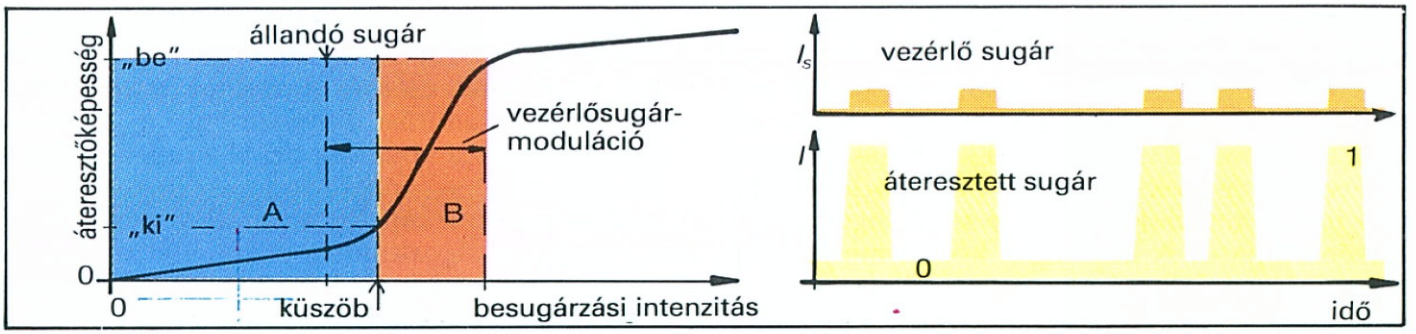
Saját tárolóval rendelkező **mikroprocesszorok ezrei** párhuzamos adatfeldolgozó hálózatot alkotnak.

Példa: az 1991-ben üzembe helyezett CM-5s számítógépet 450 mikroprocesszorból felépülő hálózat képezi.

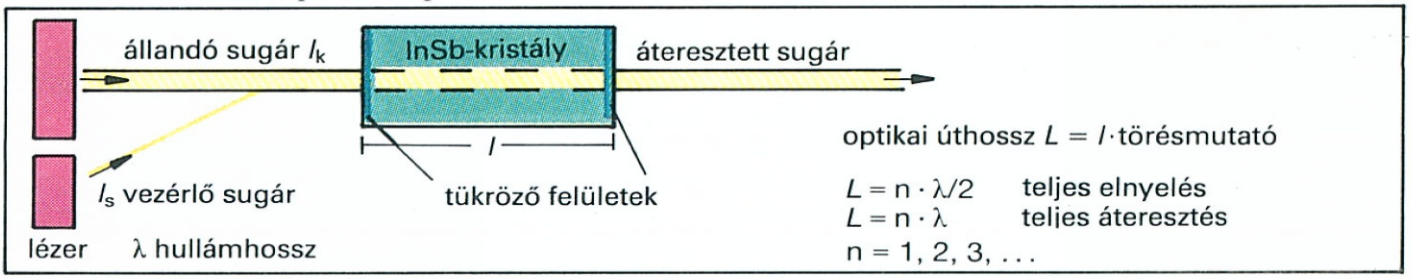
Nagyszámú, legkülönbözőbb teljesítményű **számítógép** olyan számítógépes hálózatot alkot, amelynek egységeit szükség esetén parallel működésű szuper-szuperszámítógéppé lehet összekapcsolni. A feladatmegoldás során minden számítógép az általa optimálisan végrehajtható részfeladatot kapja meg.

A *világszintű szuper-szuperszámítógép* valószínűleg nem optimális megoldás, mivel az információátvitel maximális sebessége 3×10^8 m/s. Ez azt jelenti, hogy két, egymástól 1000 km távolságban lévő számítógépnek a kapcsolatfelvételhez legalább 3,3 milliszekundumra van szüksége.

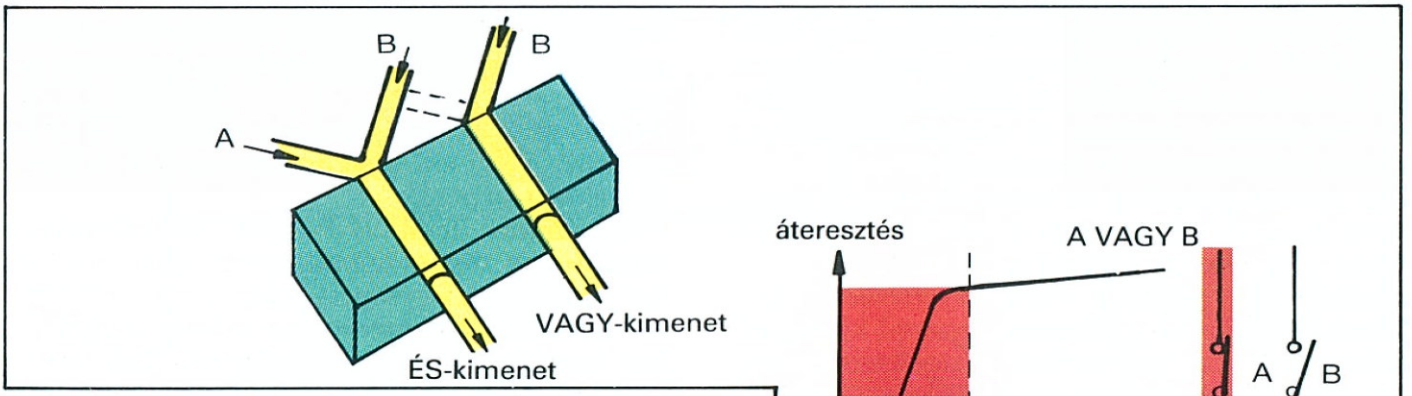
Alkalmazási példa: egy meteorológus megvizsgálja, hogy egy éppen akkor képződött felhőből tornádó alakulhat-e ki. Időjárásmodellje háromdimenziós, és több nemlineáris parciális differenciálegyenlet megoldását kívánja. Munkaállomásáról egyszerű adatvezetéken keresztül egy szuperszámítógéphez továbbítja a szükséges programkomponenseket, a mérési adatok átvitele pedig egy közepes teljesítményű vezetéken történik. A szuperszámítógép nagy átviteli sebességű adatvezetéken keresztül egy távolabbi adatbankkal lép kapcsolatba. A modellszámítások eredményei visszajutnak a meteorológus munkaállomásába, amely azokat háromdimenziós képek ábrázolására alkalmas speciális számítógépbe továbbítja. A meteorológus közben PC segítségével az időjárási helyzet keresztmetszeteit figyeli, és összehasonlítja ezeket a tőle nagy távolságban lévő központi időjárás-előrejelző intézet közepes teljesítményű számítógépében tárolt képekkel.



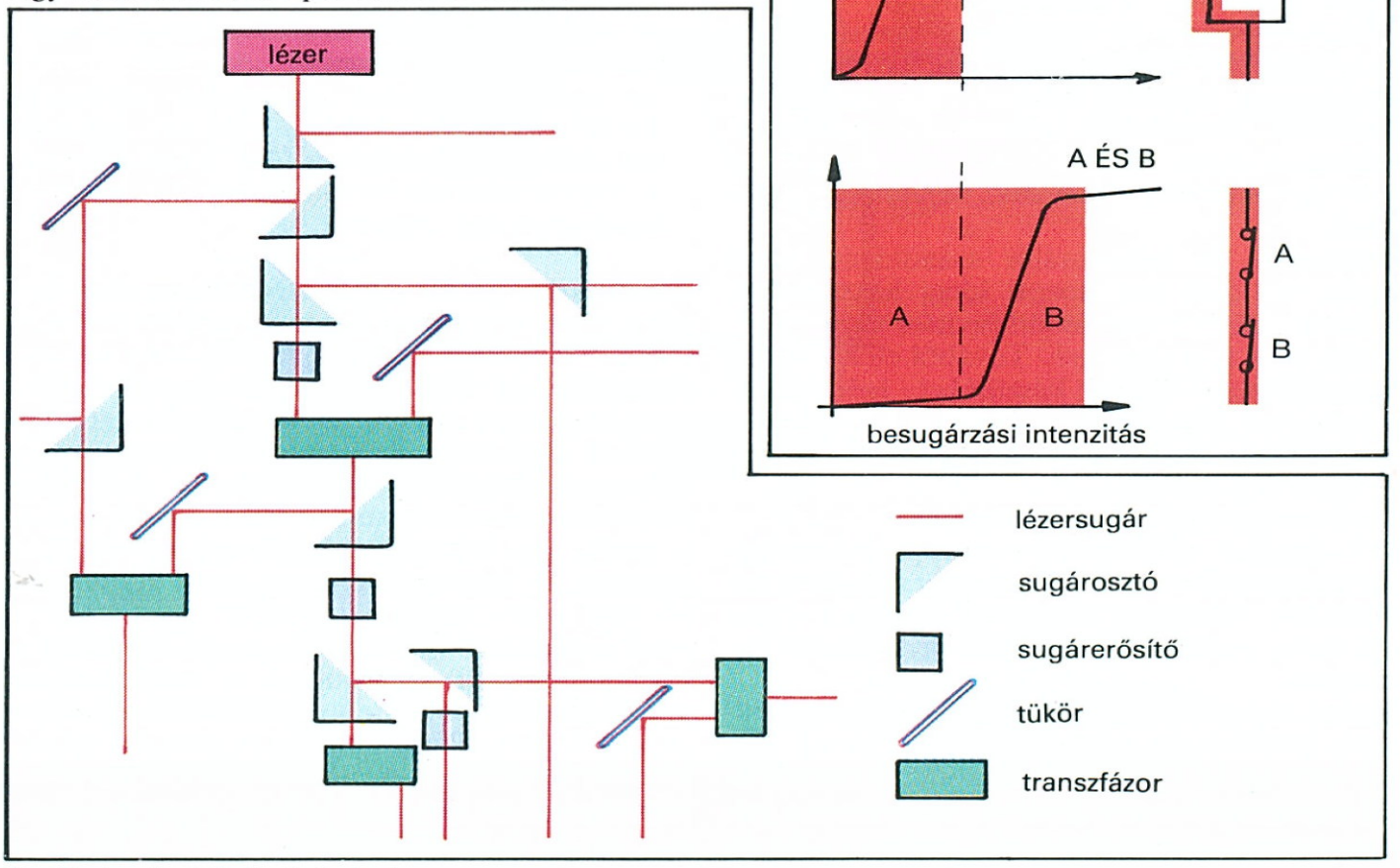
A transzfázor mint optikai kapcsoló



A transzfázor működési elve



Egy transzfázor, két párhuzamos művelet



Optikai számítógép (részlet)

A hagyományos számítógép teljesítőképességét elektronikus alkatrészeinek kapcsolási sebessége korlátozza. Napjainkban a leggyorsabb kapcsolóelemek már kb. 10^{-9} s alatt kapcsolnak, és ezzel valószínűleg elérték a megvalósítható alsó határt, amelyet az alkalmazott tranzisztor bázisában létrejövő elektronvándorlási sebesség határoz meg.
A kapcsolási időt optikai kapcsolók alkalmazásával három nagyságrenddel, vagyis kb. 10^{-12} s-ra lehet csökkenteni.

Az **optikai kapcsolóknak** legalább két stabil állapota van. Az állapotváltozás a kapcsolóelemre irányított intenzitásmodulált fénysugarakkal történik.

A bistabil optikai anyagok létezését elméletileg már az 1960-as években megjósolták, és 1976-ban kísérletileg is igazolták.

Az optikai kapcsolók alapját a **nemlineáris optika** jelenségei képezik, különösen egyes anyagoknak az a tulajdonsága, hogy törésmutatójuk a rájuk eső lézersugár intenzitásától függően megváltozik. A *Fabry Pèrot interferométer* pl. akkor működik optikai kapcsolóként, ha a tükrök közötti teret nemlineáris optikai viselkedésű anyag tölti ki. Az interferométer „zárt" állapotában a ráeső lézersugár nem tud áthaladni rajta. Ha a lézersugár intenzitását folyamatosan növeljük, akkor az interferométer meghatározott besugárzási intenzitástól (küszöbértéktől) kezdve átlátszóvá válik, és a lézersugár áthalad rajta. Ez a folyamat megfordítható.

A **transzfázor** olyan különleges optikai kapcsoló, amelyet a ráeső lézersugár intenzitásváltozása vezérel. Az *áteresztőképesség-besugárzási intenzitás* a karakterisztika lépcsőzetes függvény: meghatározott küszöbértéktől kezdve a besugárzási intenzitás kismértékű növelésekor az áteresztőképesség meredeken majdnem a 100%-os értékig növekszik.

Ez hasonlít a pnp típusú tranzisztorok karakterisztikájára: az emittor-kollektor feszültség kismértékű növelésének hatására a bázisáram meredeken megnövekszik.

Felépítése. A transzfázor parányi, négyszögletes, két végén polírozott indium-antimopid-kristály, amely Fabry-Pèrot interferométerként működik. Az egyik végét merőlegesen áthaladó infravörös lézersugárral (ún. *állandó sugárral*) világítják meg, az áteresztett fény intenzitását eddig miniatűr fénydetektorral mérik. A kristályt cseppfolyós levegővel hűtik.

Kapcsolás. A lézer intenzitását kevéssel a küszöbérték alá állítják be. Ha egyidejűleg egy második lézersugár (ún. *vezérlő sugár*) világítja meg ugyanezt a felületet, akkor a transzfázor már nagyon csekély intenzitás hatására is áteresztővé válik. A detektor ilyenkor nagyon intenzív sugárzást mér (állandó sugár + vezérlő sugár. A transzfázort tehát a vezérlő lézersugár nagyon kismértékű modulációja működteti.

A transzfázor két („átlátszatlan" és „átlátszó") állapota a 0/1 bináris jeleknek, ill. a logikai kijelentések igaz/hamis értékeinek felel meg. Az állandó sugár mindkét állapot fenntartásához szükséges, ezért annak állandóan működésben kell lennie.

A **logikai kapcsolásokat** a transzfázorral rendkívül egyszerűen meg lehet valósítani, egyetlen alkotóelem elegendő.

ÉS művelet. Csak a két sugár *együttes* intenzitása változtatja meg a transzfázor állapotát.

VAGY művelet. Már a két sugár *egyike* is megváltoztatja a transzfázor állapotát.

NEM művelet. A transzfázor bemeneti felületéről *visszavert* intenzitását egy detektor méri. Ha a besugárzás intenzitása a küszöbértéknél nagyobb, akkor a detektor kis intenzitásértékeket mér, a küszöbérték alatt viszont a beeső sugár jelentős része visszaverődik.

Az optikai számítógép felépítése

A legegyszerűbb megoldás az, ha a hagyományos számítógép felépítését alapul véve az elektronikus alkatrészeket optikaira cseréljük ki. Az optikai integrált áramkörök (optikai chipek) száloptikát és fényvezető rétegeket tartalmaznának. Már egy ilyen elrendezés is kb. ezredrészére csökkentené a számolási időt.

A számítási sebesség növelésének további lehetőségei:

Párhuzamos feldolgozás. A transzfázor – a tranzisztorral ellentétben – egyidejűleg több fénysugarat is fel tud dolgozni. Az optikai számítógépet tehát a komponensek számának túlságos növelése nélkül parallel számítógépként lehet alkalmazni.

Nembináris kapcsolási logika. A normális transzfázor karakterisztikája egyetlen lépcsőből áll, azonban olyan nemlineáris optikai kapcsolóelemek is léteznek, amelyek karakterisztikája többfokozatúan lépcsőzetes jellegű. Ez újszerű kapcsolási logikához vezet, amely nem csupán kettő, hanem több, egymástól megkülönböztethető állapoton alapszik.

Az optikai számítógép előnyei: nagyon gyors, energiafelhasználása csekély, kompakt felépítésű és nem zavarérzékeny.

A fejlődés jelenlegi állapota

Az optikai kapcsolókat időközben már sorozatban gyártják. A száloptikás adathálózatokban és telefonközpontokban kapcsolóként használják őket.

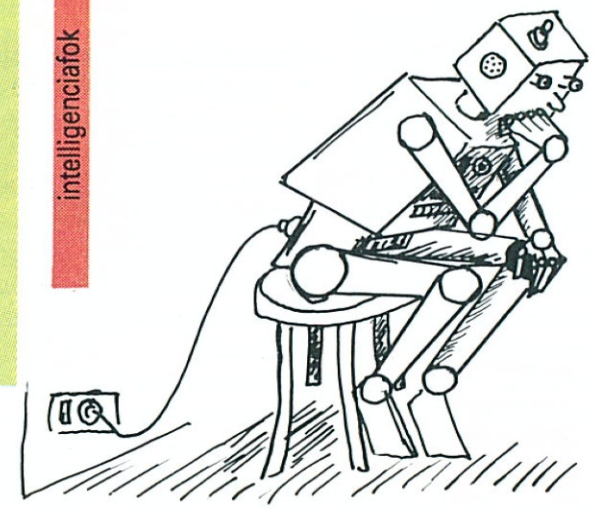
Kisebb (pl. rendkívül gyors mátrixszorzásra alkalmas) számítógép-prototípusok is működnek már, ezek teljesítménye még korlátozott. Az eddig elért számítási sebesség kb. 10^{12} bináris művelet másodpercenként, de 10^{15} másodpercenkénti művelet is elérhetőnek látszik.

Artificial Intelligence → mesterséges, utánzott, nem valódi, látszat intelligencia

mesterséges intelligencia
↑
KBS
↑
szakértői rendszer
↑
adatbázis

intelligenciafok

program
szenzorok
robot
működtető szerkezetek



adatbázis
+
adatelőkészítés
+
ember–gép kapcsolat
+
párbeszéd
↓
problémamegoldás

Mesterséges intelligencia

A **mesterséges intelligencia** (angolul: ***A**rtificial **I**ntelligence*, **AI**). Az alkalmazott informatika egyik részterülete. Feladata az, hogy olyan számítógépprogramokat hozzon létre, amelyek az intelligens viselkedést utánozzák. Semmi esetre sem biztos, hogy ez a cél egyáltalán elérhető.

Más részterületek, mint pl. a pszichológia és a nyelvészet az alkalmazott informatikához hasonlóan, de más nézőpontokat érvényesítve közelítenek a mesterséges intelligencia kérdésköréhez.

Kezdetben speciálisan erre a célra kidolgozott programnyelveken (LISP és PROLOG) írták a mesterséges intelligencia kutatási céljaira szolgáló programokat, ma már a hatékonyság miatt a C vagy a PASCAL nyelvet részesítik előnyben.

A probléma megfogalmazása

Intelligencia. Az *intelligencia* fogalmának általánosan elfogadott definíciója még nem született meg, és hasonló mondható el a *mesterséges intelligenciával* kapcsolatban is.

Az angol AI *(artificial intelligence)* kifejezés mesterséges intelligenciát jelent, ez azonban teljesíthetetlen elvárásokat ébreszt; helyesebb lenne „látszat-intelligenciáról" beszélni.

A **Turing-tesztet,** amelyet ALAN TURINGRÓL neveztek el, gyakran a sikeresen kifejlesztett mesterséges intelligencia kritériumának tekintik: egy ember adatvezetéken keresztül több, számára láthatatlan partnerrel társalog, a partnerek egyike azonban egy gép. Ha az ember nem tudja eldönteni, hogy a partnerek közül melyik a gép, akkor ez a gép a teszt szerint mesterséges intelligenciával rendelkezik.

Eddig még egyetlen számítógép sem teljesítette a Turing-tesztet. Ennek az állításnak néhány vizsgálat ellentmondani látszik, azonban az ember és a speciális társalgóprogram (pl. ELIZA) között mindig csak több-kevesebb „üres" beszélgetés jött létre.

STEVEN ROSE-nak a mesterséges intelligenciával szemben támasztott követelménye az, hogy egy vérbeli pókerjátékosokból álló társaság a játék során a számítógépet ne ismerje fel.

A mesterséges intelligencia (leendő) alkalmazásának **általános problematikája:** valószínűleg számos munkahely megszűnik, és a munkahelyek dehumanizálódnak. Növekszik az egyre összetettebbé váló és végső soron áttekinthetetlen rendszerektől való függőség, és ezeknek a rendszereknek a döntéseit kell elfogadnunk. Tisztázatlan, hogy a gép hibás művelete esetén kit terhel a felelősség.

Egyáltalán nem biztos, hogy mesterséges intelligencia valaha is létezni fog, és ha igen, akkor az emberi intelligenciát vissza fogja-e tükrözni. Ennek fő oka: minden mesterséges intelligencia program logikus következtetéseken alapszik. Ha a formális logikát gondosan nem korlátozzuk, akkor az alkalmazás előbb vagy utóbb – a mesterséges intelligenciára nézve katasztrofális – paradoxonokba torkollik.

A logikus következtetések problematikájára már BERTRAND RUSSEL (1872–1970) rámutatott: „A fodrász mindenkit megborotvál, aki nem maga borotválkozik. Kérdés: ki borotválja a fodrászt?"

Mesterséges intelligencia kutatásának története

A mechanikus számológépek már pár száz évvel ezelőtt átvettek néhány szellemi tevékenységet az embertől. Logikusnak tűnik, ha ezt a fejlődést a gondolkodó gép megszületéséig extrapoláljuk. ADA BYRON, LEIBNIZ, BOOLE és TURING az intelligens gépet elképzelhetőnek tartották.

A *mesterséges intelligencia* fogalma a számítógépek programozásával kapcsolatban először 1956-ban az USA-ban merült fel. Abból az elgondolásból indultak ki, hogy lehetségesnek kell lennie olyan programnak, amely az emberi gondolkodást szimulálja, vagyis tanulásra képes, és az újonnan szerzett tudást alkalmazni tudja. Az első működő mesterséges intelligencia program matematikai tételek bizonyítását végezte.

Az 1960-as években a kutatás arra összpontosult, hogy az emberi feladatmegoldás módszereinek algoritmusait megtalálják, és ezeket a számítógépbe programozzák. Az informatikusok és a pszichológusok a „józan paraszti észt" alábecsülték. A nehézségek növekedésével csökkentek az elvárások. A mesterséges intelligencia definíciói időközben igénytelenebbé váltak, pl.: „a mesterséges intelligencia olyan rendszer, amely a feladatot a rendelkezésre álló adatok segítségével meg tudja oldani".

Az 1990-es évek informatikusa megállapíthatja: Nem létezik (még) olyan program, amely „józan paraszti ésszel" rendelkezne, amely önálló tanulásra lenne képes, vagy amely a saját hibáit felismerné.

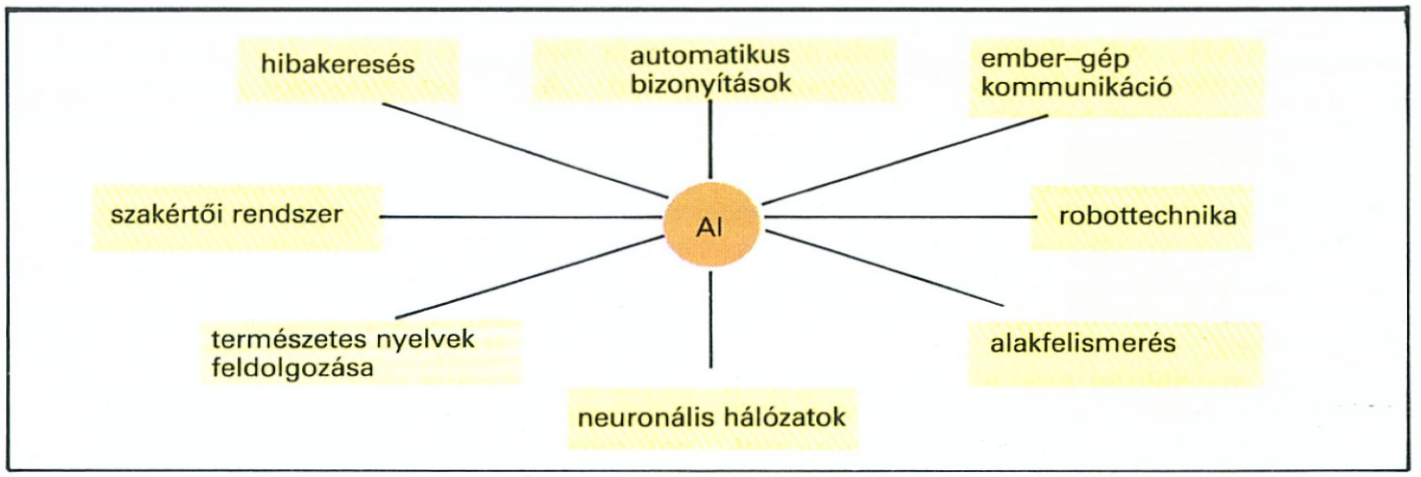
Jelenleg a mesterséges intelligencia fejlesztése azokra a konkrét és formalizálható tartományokra koncentrálódik, amelyeknek intuitíven nincs közük az intelligenciához: pl. szakértői rendszerek, neurális hálózatok, sakkprogramok stb.

A mesterséges intelligencia alkalmazási területei

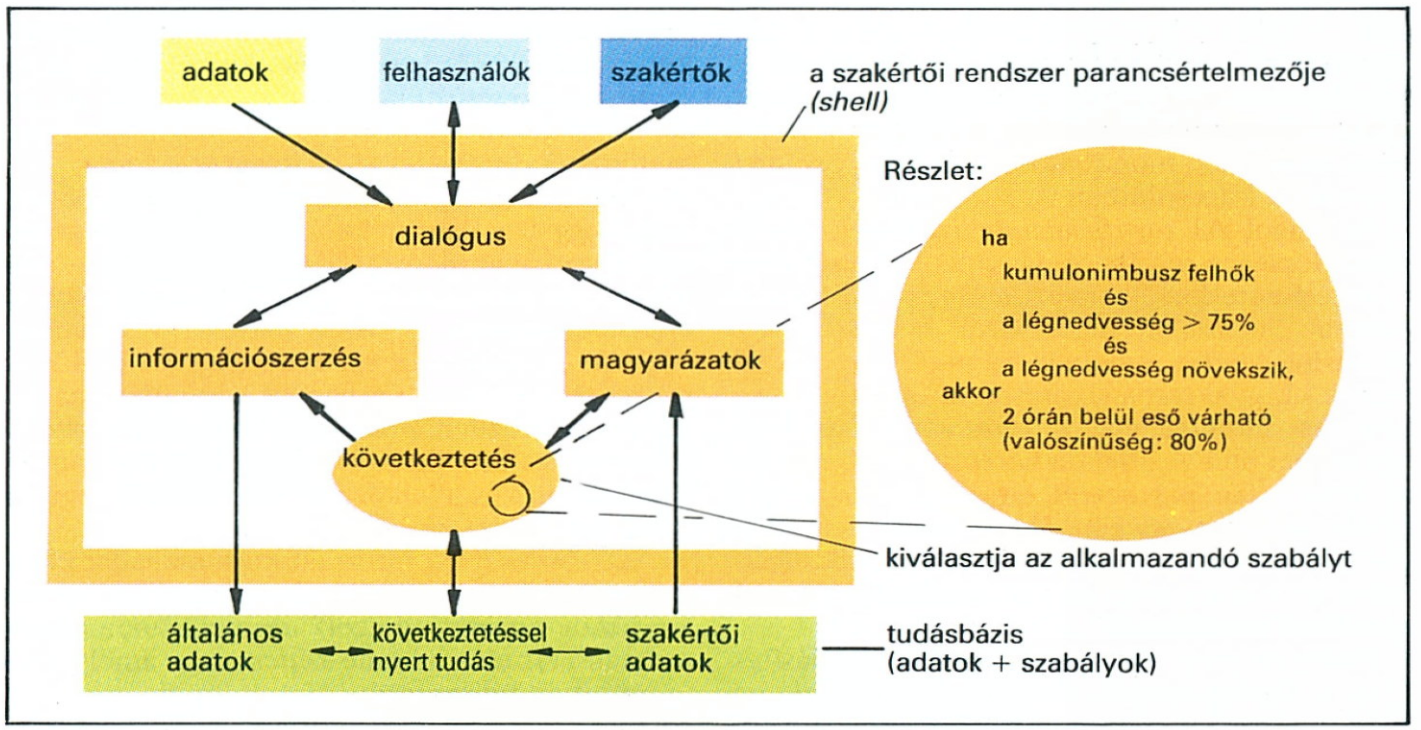
A **szakértői rendszerek** szűk szakmai tudásterületre korlátozva precízen megfogalmazott algoritmusok segítségével kijelentéseket kombinálnak, és ebből következtetéseket vonnak le.

A **tudásalapú rendszerek** (angolul: ***K**nowledge-**B**ased **S**ystems,* **KBS**) olyan teljesen általános rendszerek, amelyek valamilyen formában információt gyűjtenek és dolgoznak fel, nem szükségszerűen a formális logika szabályait követik.

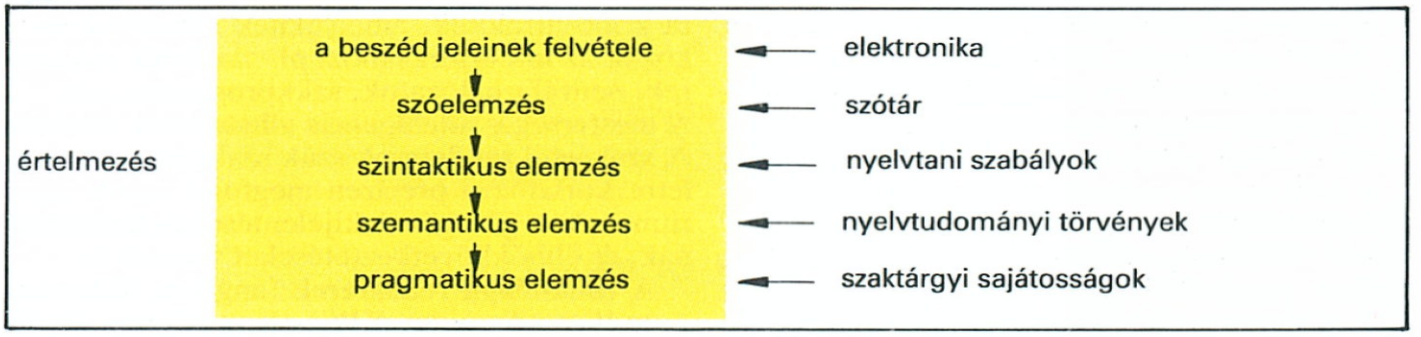
A szakértői rendszerek nagyon gyakran alkalmaznak olyan kifejezéseket, mint „ha..., akkor...", de a szakértői rendszer következtetési módjának nem kell az ember számára megszokott logikus következtetéssel megegyeznie. Rendkívül fontos, hogy a felhasználó ismerje az alkalmazás határait, valamint azokat a kiértékelési kritériumokat és szabályokat, amelyek alapján a szakértői rendszer a tárolt adatokat feldolgozza, és hozzáférhetővé teszi.



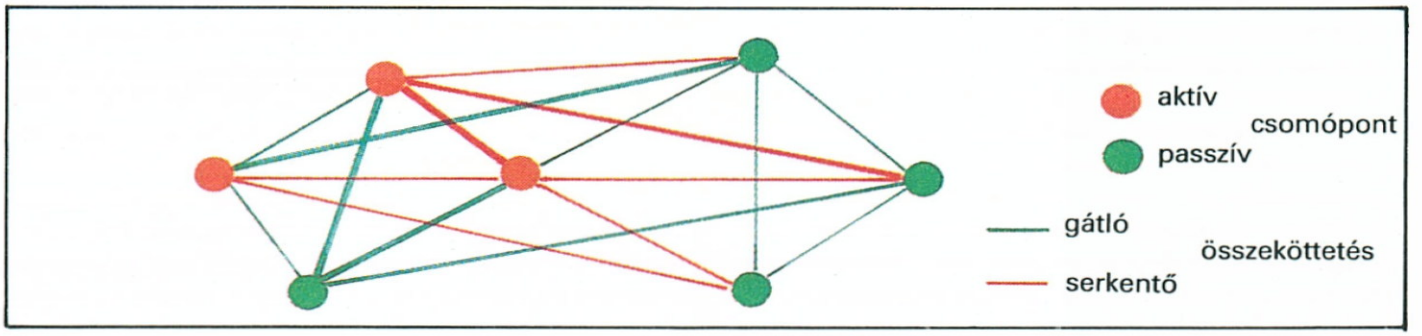
A mesterséges intelligencia részterületei



Egy szakértői rendszer részei



Beszédfelismerés a mesterséges intelligencia keretein belül



Neuronális hálózat

A szakértői rendszer legtöbbször egy számítógépprogram, tehát egy szoftver, amelyet gyakran speciálisan kifejlesztett hardverkomponensek egészítenek ki. *Egyszerű szakértői rendszerek* már régóta használatban vannak, ilyenek pl. a növény- és állathatározók: az adatokhoz való hozzáférés egyszerű szabályok alapján (pl. a külső megjelenésre vonatkozó kérdéssorozatok formájában) történik. A szakértői rendszerek az emberi szakértői vizsgálatot (még?) nem helyettesítik. A rutinmunkát veszik át és segítséget nyújtanak a döntésekhez: számítógépprogramokban hibakeresést, kémiai analízist, bankok számára hitelképesség-vizsgálatot stb. végezhetnek. Léteznek *tanulásra képes szakértői rendszerek* is, amelyek a felhasználóval folytatott dialógus alapján önállóan következtetéseket vonnak le, és ennek alapján pl. egy adatbázishoz történő hozzáférés elsőbbségi sorrendjét megváltoztatják. A bevált szakértői rendszerek viszonylag egyszerűek, és adatbankok feldolgozására ill. kezelésére használhatók.

Példák:

A Digital Equipment cég **XCON-rendszere** a rendelkezésre álló komponensekből és a megadott műszaki paraméterek alapján egy teljes és optimális számítógéprendszert állít össze. Egyidejűleg megtervezi a szükséges beépítési diagramokat is. Ennek a szakértői rendszernek a hibaszázaléka 10 évi működés adatai alapján 5%-nál kisebb.

A **MYCIN** rendszer vérrel terjedő fertőzésekre vonatkozó lexikális adatokat tartalmaz orvosok számára. Az orvos a MYCIN rendszerrel párbeszédes kapcsolatban áll, és így állapítja meg a beteg diagnózisát. A rendszer viszonylag szokatlan válaszokat is kiértékel. MYCIN figyelembe veszi, hogy minden tárolt ismertetőjegyhez, szóbeli közléshez és adathoz bizonytalanság is tartozik. A szakértői rendszer minden olyan szabályt felsorol, amely a diagnózishoz és a terápiás javaslathoz vezetett, és ezekre valószínűségi kijelentésként kvantitatív jellemzést is ad.

A **PROSPECTOR** a beprogramozott algoritmus segítségével összegyűjti és kombinálja az előforduló, gazdaságilag fontos nyersanyagok összes geológiai és morfológiai adatát. Egy újonnan feltárt terület megfelelően részletes adatai alapján megállapítja, hogy adott területen nagy valószínűséggel milyen gazdaságilag értékesíthető ásványok fordulnak elő.

Fontos, hogy a szakértői rendszereket mindig csak a tervezettnek megfelelő, nagyon szűk szakterületen alkalmazzák. A MYCIN rendszernek például az „általános fertőző betegségek" diagnosztizálásában kevés hasznát lehet venni.

A **neuronális hálózatokon** olyan számítógép-architektúrákat értünk, amelyek szerkezetüket és működésüket tekintve a legegyszerűbb idegrendszerekhez hasonlítanak. Nagyszámú kapcsoló és szabályozó elemből, ún. *csomópontból* állnak, amelyek (a neuronokhoz hasonlóan) aktív vagy passzív állapotban lehetnek, és (az idegszálakkal analóg módon) hálózatszerűen kapcsolódnak egymáshoz. Ezt a hálózatot egymáshoz hasonló feladatoknak és ezek megoldásának betáplálásával lehet „programozni". Ha a hálózat topológiája optimális és az egyes csomópontok kiszámítására szolgáló algoritmusok megfelelőek, akkor a neuronális hálózat egymással rokon feladatokat (remélhetőleg) meg tud oldani, vagyis a neuronális hálózat „tanul" vagy pontosabban: „betanítják".

A neuronális hálózatok pl. az alakfelismerés területén használhatók. Ez olyan probléma, amelyet a hagyományosan programozott számítógépek rendkívül nagy ráfordítással és ezért gazdaságtalanul tudnának csak megoldani.

A neuronális hálózatok hátránya: az egyes esetekben talált megoldásokat nem magyarázzák meg. A hálózat felhasználója a megoldásokat nem tudja megismételni, tehát ellenőrzésük sem lehetséges.

A számítógépes sakk a mesterséges intelligencia talán leglátványosabb alkalmazási területe. Az egyszerű sakkprogramok *szélességben* végeznek keresést, tehát az összes sakkfigurának minden lehetséges helyzetét több lépésre előre (mélységben) megvizsgálják. A program kiértékeli a lehetséges helyzeteket, és végül egy optimális lépést javasol. A hat lépésnél mélyebb keresés számítási igénye rendkívül nagy. A *stratégiaprogramok* csupán néhány lépéskombinációt és ezeknek a játék kimenetelére gyakorolt hatását vizsgálják. Ezek a programok lényegesen kevesebb számítást igényelnek.

A legjobb sakkprogramok jelenleg elérik egy nemzetközi nagymester játékának színvonalát.

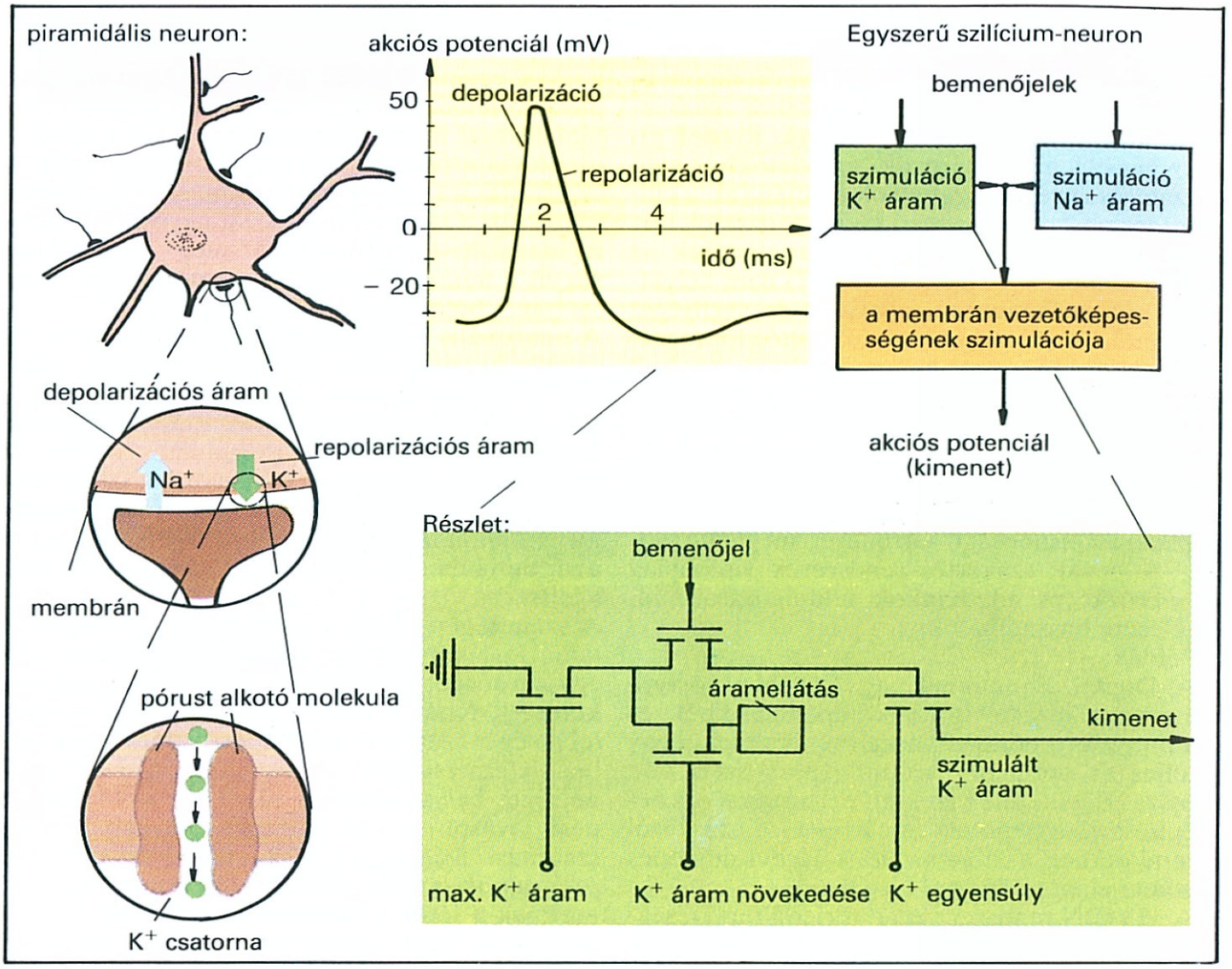
A **robottechnika** a mesterséges intelligenciát arra használja fel, hogy előírt műveletek sorozatát rugalmasan alakítsa. A környezet lényeges jellemzőit szenzorok érzékelik és a megfelelő jeleket egy processzorba továbbítják. A processzor a célkitűzést és a munkatervet a környezeti tényezőkhöz igazítja, és ennek megfelelően vezérli a robot végrehajtó szerkezetét.

Az **észlelő rendszerek** *(perception systems)* optikai és akusztikai jeleket dolgoznak fel. Pl. kép- és alakfelismerés céljaira használhatók.

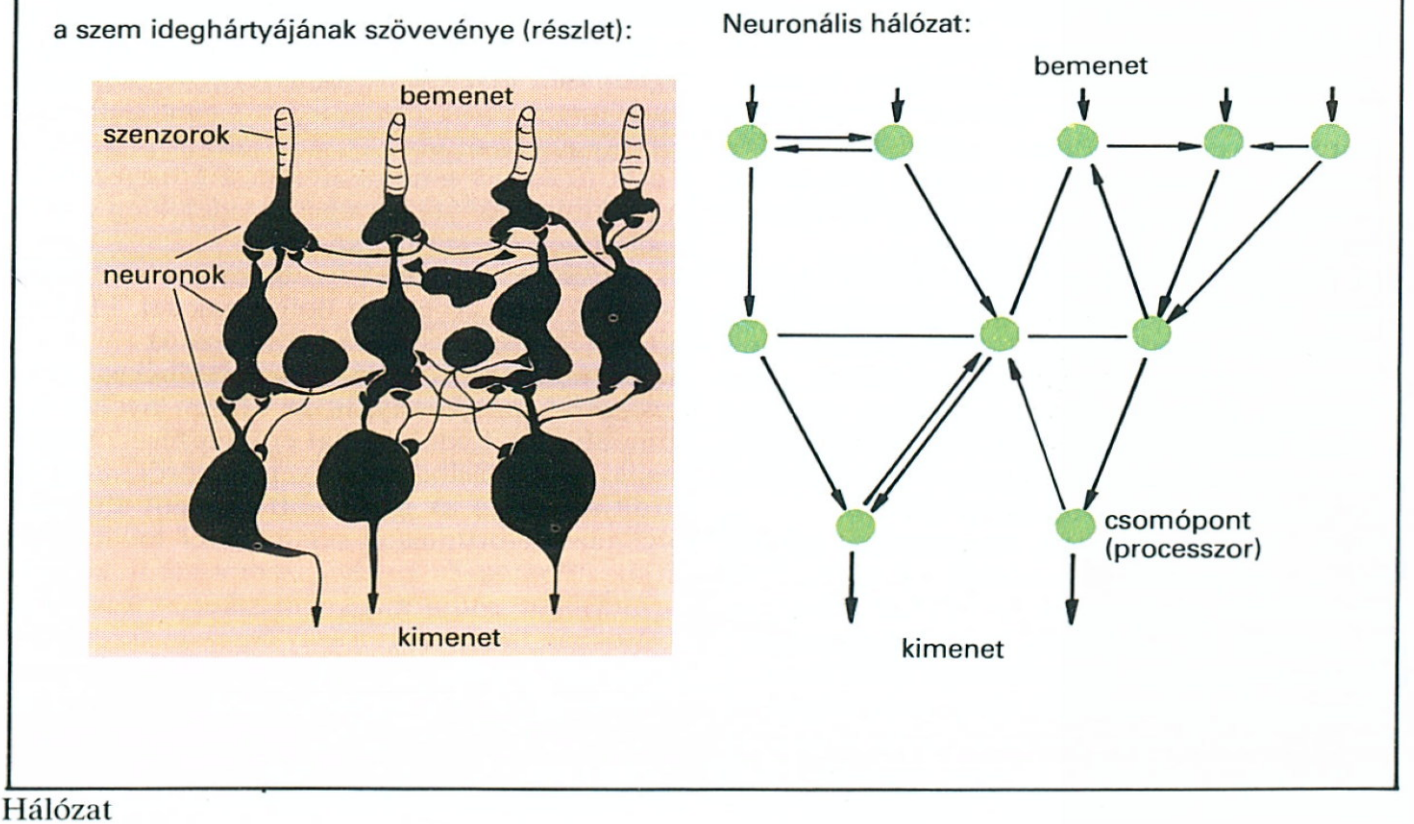
Az **automatikus fordító és szedéshibákat javító rendszerek** megkönnyítik az idegen nyelvű dokumentumok feldolgozását és elemzését. Működésük első fázisában a szöveget szóról szóra lefordítják, majd az így kapott anyagot általános nyelvtani algoritmusok segítségével tovább alakítják. Pontos és nyelvi szempontból korrekt fordításokat eddig még nem sikerült készíteni, elsősorban azért, mert a szavaknak gyakran több jelentésük is van.

Híres példa: ha az *out of sight, out of mind* szólást oroszra, majd ismét angolra fordítjuk, akkor az eredmény: *invisible idiot.*

Az automatikus fordítás csak nyers fordításnak tekinthető, amely emberi közreműködéssel még tökéletesítésre szorul.



Piramidális neuron és a szilícium-neuron



Hálózat

Az első nagy számítógépeket *elektronikus agyaknak, gondolkodó gépeknek* nevezték, mert munkájuk hasonlónak tűnt az emberi agy munkájához. Ez a felfogás naiv és túlhaladott. Mégis érdemes összehasonlítást végezni, mert néhány számítógépbeli folyamatból az agy működésére is lehet következtetéseket levonni, másrészt a neurofiziológia ötleteket adhat a számítógép-architektúra kialakítására.

Cortex (agykéreg). Az agykéreg, az emberi intelligencia központja kb. 3×10^9 alkotóelemet, ún. neuront (idegsejtet) tartalmaz, amelyek átmérője 5 és 100μm közé esik. A neuronokat idegszálak hálózata köti össze egymással. Az információ (az izgatás) felvétele speciális sejtek (szenzorok) segítségével történik. Magát az információt kb. 1 ms-os áramimpulzusok sorozata kódolja. Ezeket az impulzusokat az idegsejt elektromosan töltött membránjának pórusain átfolyó ionáramok hozzák létre. A jelátviteli sebesség eléri a 100 m/s értéket. Az információfeldolgozás párhuzamos és valós idejű *(real time)*.

Bár a neuronok az agykéreg legkisebb építőkövei, kapacitásukat mégsem lehet a számítógép elektronikus alkotóelemeivel egy lapon említeni. Már egyetlen neuron is nagy mennyiségű információt dolgoz fel, ezért inkább a mikroprocesszorhoz hasonlítható.

Számítógép. Az elektronikus alkotóelemeknek mind a mérete, mind a száma (ha a számítógép tárolóelemeit is számításba vesszük) hasonló az agykéreghez. A számítógép alkotórész-sűrűsége azonban lényegesen kisebb. Rendszerint csak két kapcsolóelem lép egymással kölcsönhatásba. Az információ feldolgozása még túlnyomórészt soros üzemmódban történik.

A jelek időtartama a nanoszekundum tartományba esik, tehát kb. 6 nagyságrenddel alacsonyabb az agykéreg megfelelő áramimpulzusainál. Az átviteli sebesség megegyezik a fénysebességgel.

Az agykéregben a párhuzamos feldolgozás messzemenően kompenzálja a viszonylag csekély információátviteli sebességet. Az embernek pl. egy arc azonosításához kb. 1/10 másodpercre van szüksége. A legnagyobb teljesítményű számítógép ugyanezt a feladatot több perc alatt tudja csak elvégezni, anélkül hogy egyértelmű azonosításról beszélhetnénk.

Az agykéreg működésének szimulációja

Az agykéregben végbemenő információfeldolgozás elektronikus szimulációja alapvetően kétféle módon lehetséges.

Az egyik eljárás során a cortex globális működését igyekeznek utánozni. Eközben csak a helyes végeredmény számít, a megoldáshoz vezető út nem játszik szerepet. A másik lehetőség az, hogy az egyes neuronok működésmódját elektronikus úton szimulálják, majd az így kapott szilikon-neuronokat működőképes neuronális hálózattá kapcsolják össze.

Globális modell. Terjedelmes szoftver segítségével az agy egyes funkcióit mint egészet utánozni lehet. A bemeneti információt detektorok (pl. fényérzékeny detektorok) sorozata szolgáltatja. A program ezeket a jeleket átveszi, és az előzőleg rögzített feltételek szerint kiértékeli. Ennek során megvalósítható az, hogy a program tanuljon. Ez azt jelenti, hogy egy feladat sikeres megoldása után a következő, hasonló feladatot a gép gyorsabban tudja elvégezni.

Példa: az **alakfelismerés.** Meg kell találni az „A" betűt egy olyan rendezetlen halmazban, amelyben a betűk három dimenzióban irányítottan helyezkednek el. Egy optika pontról pontra letapogatja a bemutatott képet, és minden térbeli ponthoz egy intenzitásértéket rendel. Az azonosító algoritmus megkíséreli a megegyező intenzitásértékű pontokat olyan módon összekötni, hogy az „A" jel felismerhető legyen.

A fenti globális módszer fontos szerepet játszik a *robotok „látásának"* megvalósításában.

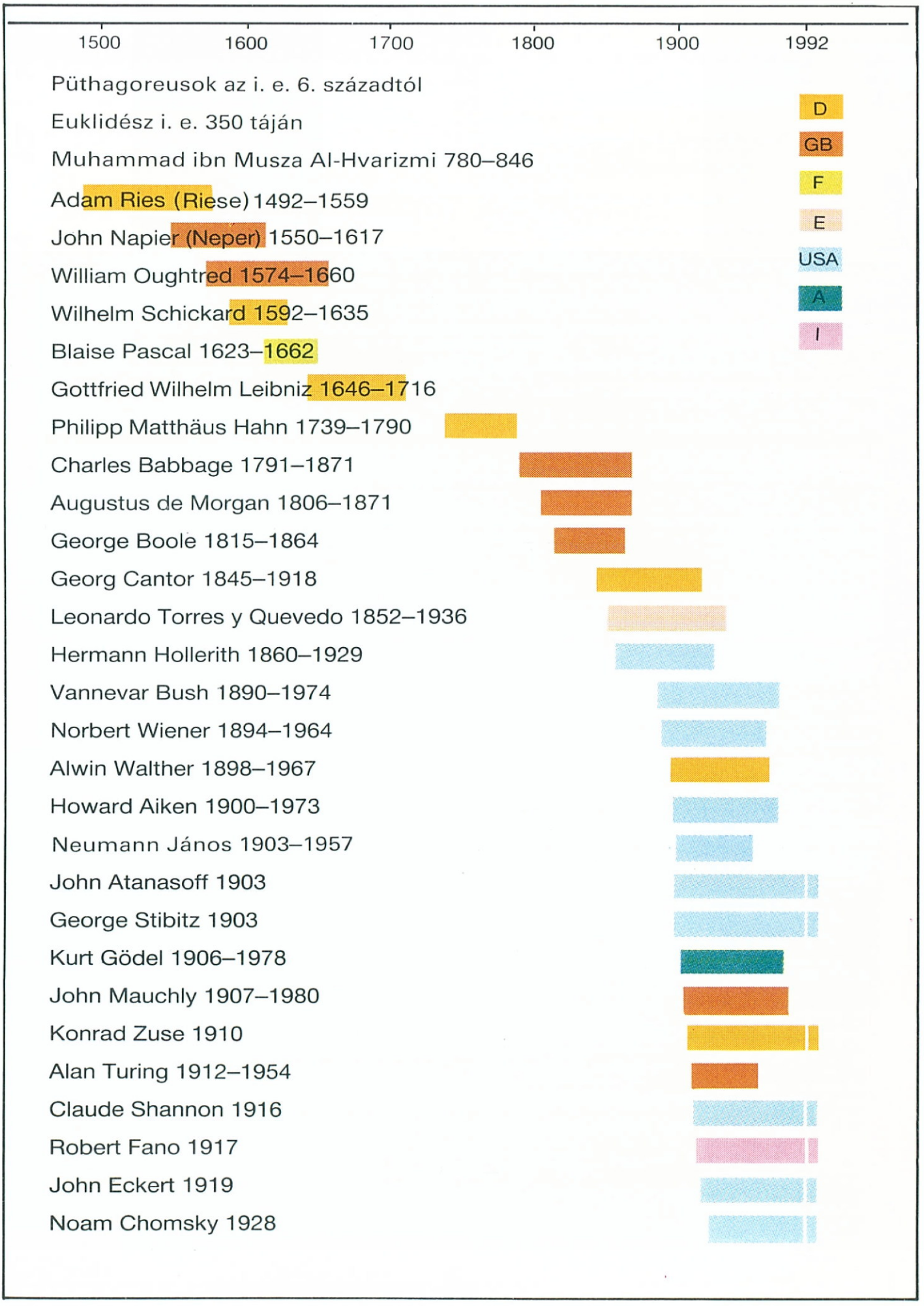
Részletmodell. Egy elektronikus építőelemet úgy terveznek meg, hogy a bemenőjelek bizonyos sorozatára a kimenetén egyetlen cortexbeli neuron kimenőjéhez hasonló jelet bocsásson ki. Ebben az esetben a problémamegoldásra alkalmas algoritmust a hardver tárolja, amely valós idejű üzemmódban működik.

Példa: **szilikon-neuron.** CMOS (angolul: **C**omplementary **M**etal **O**xide **S**emiconductor) technológiával előállított félvezetőkből integrált áramköri kapcsolást valósítanak meg úgy, hogy a kapcsolás egy sejtmembrán ionáteresztőképességét szimulálja. Az állandó átvezetési áramú integrátor villamos kimenőjele pl. a sejt belsejében uralkodó Ca^{++}-koncentrációt modellezi, amely időben állandó bemenőjel esetén az impulzus frekvenciájának csökkenéséért felelős. Ezzel a neuron egyik lényeges tulajdonságának, az *adaptációnak* szimulációjára nyílik lehetőség.

Neuronális hálózatok

Mikroprocesszorok ill. szilikon-neuronok (amelyeket itt *csomópontoknak* nevezünk) rétegekben felépített hálózatot alkotnak, és a neuronok serkentő és gátló tulajdonságait szimulálják. A kutatók azt remélik, hogy a szilikon-neuronok megfelelő kölcsönhatása révén az agykéreg működése utánozható lesz.

Példa: a kötőhártya egyszerűsített szerkezetének szimulációja nagyon egyszerű minták felismerésére.



Az informatika kiemelkedő képviselői

Az informatika mérföldkövei

i.e. 6. sz. Püthagorasz-féle számolódeszka
szorobán, abakusz, számolópad
i.e. 1. sz. Antikythera-számoló
1. sz. Alexandriai Héron útmérője

9. sz. Al-Hvarizmi számtankönyve (›Liber algebrae at almucabola‹)
13. sz. az analóg mechanizmusok (órák) széles körű elterjedése
14. sz. gelosia-módszerrel működő számolódeszka Európában
1518 Adam Ries számtankönyve (›Rechenung auff der linihen‹)

17. sz. eleje Neper-pálcák
Schott hengeres számolópálcái
1622 logarléc, számolópálca
1623 Schickard összeadógépe
1641 Pascal összeadógépe („Pascaline")
1670 bordáshengerrel működő számológép a négy alapművelet elvégzésére
1674 Hahn számológépe („számolómalom")
1679 kettes számrendszer

kb. 1800 planiméter (analóg számolóeszköz)
kb. 1810 lyukkártya, lyukszalag
1818 sorozatban gyártott számológép („Arithmomètre")
1821 Babbage „Difference Engine" nevű gépének építése megkezdődik
1834 Babbage „Analytical Engine" nevű gépének építése megkezdődik
1844 digitalizált távadatátvitel (Morse)
1847 a formális kijelentéslogika alapjai
kb. 1850 logikai kapcsolások
elektromágneses relék
1875 Lord Kelvin harmonikus analizátora („Harmonic Analyzer")
1876 árapályszámító berendezés
1886 Hollerith lyukkártyás gépe

1904 elektroncsövek
1905 villamos meghajtású számológép
1914 lebegőpontos számolómű
1932 relékkel megvalósított számjegytároló
1937 első számítóközpont (London)
Turing-gép

1938 az első szabadon programozható számítógép (Z1)
1941 az első programvezérlésű számítógép (Z3)
1943 COLOSSUS számítógép
1944 Harvard-Mark-I számítógép
mágnesdobos tároló
1946 az univerzális számítógép Neumann-féle elvei
teljesen elektronikus számítógép (ENIAC)
az első programnyelv („Plankalkül")
1948 tranzisztor
nagy teljesítményű analóg számítógép
az információelmélet alapjai

1951 az UNIVAC I elektronikus számítógép sorozatgyártása
1952 mágnesszalagos tároló
1953 az IBM 701 nagyszámítógép sorozatgyártása
1954 FORTRAN
1960 ALGOL
1962 integrált áramköri kapcsolás (chip)
1964 integrált áramköröket tartalmazó számítógépek
1966 az informatika szak bevezetése az egyetemeken

1970 az első adatvédelmi törvény (Hessen, NSZK)
1972 mikroprocesszorok
szuperszámítógép (CRAY I)
hajlékonylemezes tároló
1974 az első PC (Mark-8)
1976 optikai kapcsolók
1979 MS-DOS operációs rendszer

1981 IBM PC
1988 megabit-chipek
1991 parallel számítógépek

access time	hozzáférési idő
accumulator	akkumulátor
arithmetic instruction	aritmetikai utasítás
arithmetic unit	aritmetikai egység
array	tömb
assembler	asszembler
backup	biztonsági másolat
batch processing	kötegelt feldolgozás
binary	bináris
bit	bit (az információtartalom egysége)
blank	szóköz
bootstrap	kezdeti programbetöltő
buffer	puffer, közbenső tár
card file	kártyaállomány
card punch	kártyalyukasztó
carriage return	kocsi vissza
central processing unit	központi feldolgozóegység
central processor	központi egység
channel	átviteli csatorna
character	jel, karakter
character set	jelkészlet
(to) check	ellenőrizni
clone	klón, hasonmás
column	oszlop
compiler	fordítóprogram
console	kezelőpult
(to) control	szabályozni, vezérelni
control unit	vezérlőegység, vezérlőmű
counter	számláló
CPU-time	számítógépidő
cycle time	ciklusidő
data base	adatbank
data processing	adatfeldolgozás
(to) debug	hibát keresni
default	alapértelmezés
diagnostic program	tesztprogram
digit	számjegy
digital	digitális
digitizer	digitalizáló
directory	könyvtár
disk	mágneslemezes tár
down time	kiesési idő
drum	mágnesdobos tár
(to) dump	kiíratni
end of file	állomány vége
even	egész számú
external memory	külső adattároló
false	hamis (Boole-algebra)
feedback	visszacsatolás
figure	jel, ábra
file	állomány
firmware	firmware (beépített mikroprogramok)
fixed point number	fixpontos szám
flag	jelölő
floating point number	lebegőpontos szám
floppy (disk)	hajlékonylemez
flow chart	folyamatábra
function	függvény, alprogram

gap	blokk-köz, rés
gate	kapuáramkör, kapu
hard copy	nyomtatott képernyőmásolat
hardware	hardver
input	bevitel, bemenet
input unit	bemeneti egység
instruction	utasítás
integer	egész számú
interface	illesztőegység
interrupt	megszakítás
I/O (input/output)	be-/kivitel, B/K
job	munka, futtatás
joystick	botkormány
keyboard	billentyűzet
keypunch	kártyalyukasztó
label	címke
light pen	fényceruza
line	sor
linkage editor, linker	kapcsolatszerkesztő
(to) load	betölteni
location	tárhely, rekesz
loop	ciklus
mainframe	univerzális nagyszámítógép
main storage	főtár
manual	kézikönyv
memory	memória
menu	menü, a lehetséges utasítások listája
message	üzenet
monitor	képernyő
mouse	egér
multi-address computer	többcímű számítógép
multi-programming	multiprogramozás
object program	tárgyprogram
odd	páratlan
off-line	offline, nem közvetlen kapcsolat
on-line	online, közvetlen kapcsolat a számítógéppel
operating system	operációs rendszer
operator	operátor, gépkezelő
optical disk	optikai adattároló
output	kivitel, kimenet
output unit	kimeneti egység
parallel access	párhuzamos hozzáférés
parity bit	paritásbit
parity check	paritásvizsgálat
plotter	rajzológép
pointer	mutató
port	kapu
printer	nyomtató
procedure	eljárás, alprogram
(to) process	feldolgozni
program	program
protected storage	védett tároló
punched card	lyukkártya
punched tape	lyukszalag

queue	várakozási sor
queue time	várakozási idő
random	tetszőleges, véletlenszerű
random access	véletlen hozzáférésű
random access memory	véletlen hozzáférésű memória
read only memory	csak olvasható, fix memória
real time processing	valósidejű üzemmód
record	rekord
redundancy	redundancia
release	felszabadítás
remote data processing	táv-adatfeldolgozás
resident program	mindig hozzáférhető, rezidens program
response time	válaszidő
(to) rewind	visszatekercselés
row	sor
run	programfutás
(to) scan	letapogatni
schedule	ütemezés
screen	képernyő
(to) screen	átnézni, megvizsgálni
(data) security	adatbiztonság
serial access	soros hozzáférés
set	halmaz
shift key	váltóbillentyű
sign	előjel
soft copy	képernyőmásolat
software	szoftver
source program	forrásnyelvű program forrásprogram
special character	különleges karakter
spreadsheet	táblázatkezelő
stack	veremtároló
stand-alone	önálló, autonóm rendszer a számítógépről lekapcsolt rendszer
statement	utasítás
storage	tárolás
string	füzér, karakterlánc
subroutine	szubrutin, alprogram
tape	mágnesszalag
task	taszk, feladat
teleprocessing	táv-adatfeldolgozás
terminal	terminál
time-sharing	időosztásos üzemmód
time-slicing	időszeletelés
tools, toolbox	eszközök, szerszámosláda
track	sáv (mágneses táron)
trackball	követőgolyó
trigger	kiváltójel
true	igaz (Boole-algbra)
unit	készülék, egység
(to) update	aktualizálni
utility	segédprogram
window	ablak, képernyőrészlet
word	szó
word processing	szövegszerkesztés
working storage	operatív tár
workstation	munkaállomás

Név- és tárgymutató

Készült a Gyomai Kner Nyomda Rt.-ben,
a nyomda alapításának 113. esztendejében
Felelős vezető: Papp Lajos vezérigazgató
☏ 66/386-172

SH atlasz
Informatika

Az SH atlasz Informatika áttekintést nyújt a szakterület történetéről, megmagyarázza az alapvető szakkifejezéseket, valamint tömören és érthetően tárgyalja az informatika legfontosabb alkalmazási területeit.
A könyv sok színes ábrát és részletes leírásokat tartalmaz.

A tartalomból:
A számítógépek története.
Információelmélet.
Számrendszerek és számábrázolás.
Adatátvitel. Adatvédelem és adatbiztonság.
Digitális számítógépek. Operációs rendszer.
Központi egység. Tárolók. Be-/kiviteli eszközök.
A számítógép programozásának általános kérdései.
Néhány programnyelv. Szövegszerkesztés.
Grafikus adatfeldolgozás. Mesterséges intelligencia.

A könyv végén angol–magyar szakkifejezés-gyűjtemény, valamint részletes név- és tárgymutató található.

1490 Ft

9 789638 455703